Государственное образовательное учреждение
высшего профессионального образования
«Самарский государственный аэрокосмический университет
имени академика С.П. Королева»

Государственное образовательное учреждение
высшего профессионального образования
«Нижегородский государственный университет
имени Н.И.Лобачевского»

Самарский научный центр Российской академии наук

Институт систем обработки изображений
Российской академии наук

ВЫСОКОПРОИЗВОДИТЕЛЬНЫЕ ПАРАЛЛЕЛЬНЫЕ ВЫЧИСЛЕНИЯ НА КЛАСТЕРНЫХ СИСТЕМАХ

Материалы
четвертого Международного научно-практического семинара
и Всероссийской молодежной школы

30 сентября – 2 октября 2004 года

Самара
2004

УДК 681.3.012:51

ББК 32.973.26-018.2:22

Высокопроизводительные параллельные вычисления на кластерных системах. Материалы четвертого Международного научно-практического семинара и Всероссийской молодежной школы. / Под редакцией член-корреспондента РАН В.А. Сойфера, Самара.

Сборник сформирован по итогам научного семинара, посвященного теоретической и практической проблематике параллельных вычислений, ориентированных на использование современных многопроцессорных архитектур кластерного типа.

Отв. за выпуск к.т.н. М.А. Чичева

Поддержка семинара

	Министерство образования Российской Федерации (Программы «Фундаментальные исследования и высшее образование», «Интеграция науки и высшего образования России»)
CRDF	Американский фонд гражданских исследований и развития (CRDF Project SA-014-02, российско-американская программа «Фундаментальные исследования и высшее образование» (BRHE))
РФФИ	Российский фонд фундаментальных исследований
Российская Академия Наук	Отделение информационных технологий и вычислительных систем РАН
	Комиссия РАН по работе с молодежью (программа «Поддержка молодых ученых – Базовые кафедры и научно-образовательные центры»)
NSTLab	Нижегородская лаборатория программных технологий
intel.	Компания Intel Technologies
UNIVERSITÀ DELLA CALABRIA CAMPUS DI ARCAVACATA	Калабрийский университет (Италия)

30 сентября - 2 октября 2004 года на базе Самарского государственного аэрокосмического университета имени академика С.П. Королева был проведен четвертый Международный научно-практический семинар и Всероссийская молодежная школа «Высокопроизводительные параллельные вычисления на кластерных системах» (HPC-2004).

Главная направленность мероприятия – обмен опытом и активизация научно-практической деятельности в области решения задач математической физики, моделирования, обработки и анализа сигналов и других задач, требующих высокопроизводительных вычислительных ресурсов, в частности, обсуждение основных аспектов организации вычислений на кластерных компьютерных системах.

В рамках семинара рассмотрены следующие актуальные вопросы:

- принципы построения кластерных вычислительных систем;
- методы управления параллельными вычислениями в кластерных системах;
- параллельные алгоритмы решения задач, требующих высокопроизводительных ресурсов;
- программные среды и средства для разработки параллельных программ;
- прикладные программные системы параллельных вычислений;
- методы анализа и оценки эффективности параллельных программ;
- проблемы подготовки специалистов в области параллельных вычислений.

В сборник включены материалы сообщений, которые представлены как в виде статей, так и в виде кратких тезисов. Материалы сборника упорядочены в алфавитном порядке по фамилии первого автора. Тексты докладов напечатаны в том виде, как они были представлены авторами. Содержание опубликованных материалов отражает исключительно точку зрения их авторов на рассматриваемую проблему и может не совпадать с точкой зрения Редакционной коллегии.

ПРОГРАММНЫЙ КОМИТЕТ

Каляев А.В.	-	председатель программного комитета, академик РАН, директор НИИ многопроцессорных вычислительных систем Таганрогского государственного радиотехнического университета
Бурцев В.С.	-	академик РАН, Институт проблем информатики РАН
Воеводин В.В.	-	член-корреспондент РАН, д.ф.-м.н, заведующий лабораторией, заместитель директора НИВЦ МГУ
Гергель В.П.	-	д.т.н., профессор кафедры МО ЭВМ ННГУ
Золотарев В.И.	-	к.ф.-м.н., директор Петродворцового телекоммуникационного центра СПбГУ
Казанский Н.Л.	-	д.ф.м.н., профессор, заместитель директора Института систем обработки изображений РАН
Крюков В.А.	-	профессор, д.ф.-м.н., зав отделом Института Прикладной Математики РАН
Кузьмичев В.С.	-	д.т.н., профессор, директор Самарского регионального информационного центра
Левин В.К.	-	академик, научный руководитель, заместитель директора ФГУП «НИИ «КВАНТ», председатель научного совета «Вычислительные системы массового параллелизма»
Нестеренко Л.В.	-	к.ф.-м.н., директор по стратегии и технологиям Нижегородской лаборатории Intel (INNL)
Сойфер В.А.	-	член-корреспондент РАН, ректор СГАУ
Стронгин Р.Г.	-	д.т.н., профессор, ректор ННГУ
Соловов А.В.	-	к.т.н., директор Самарского регионального ресурсного центра
Фурсов В.А.	-	д.т.н., профессор, директор Института компьютерных исследований СГАУ
Четверушкин Б.Н	-	член-корреспондент РАН, директор Института Математического Моделирования РАН
Шорин В.П.	-	академик, председатель Президиума Самарского научного центра РАН

ОРГАНИЗАЦИОННЫЙ КОМИТЕТ

Сойфер В.А.	-	председатель оргкомитета, член-корреспондент РАН, ректор СГАУ
Фурсов В.А.	-	заместитель председателя оргкомитета, д.т.н., профессор, директор Института компьютерных исследований СГАУ
Гришагин В.А.	-	заместитель председателя оргкомитета, к.ф.-м.н., доцент кафедры МО ЭВМ ННГУ
Бочкарев С.К.	-	к.т.н., зам. проректора по научной работе СГАУ
Казанский Н.Л.	-	д.ф.м.н., профессор, заместитель директора Института систем обработки изображений РАН
Кравчук В.В.	-	к.т.н., начальник отдела телекоммуникаций Самарского научного центра РАН
Попов С.Б.	-	к.т.н., доцент кафедры ТК СГАУ
Санчугов В.И.	-	д.т.н., профессор, зам. председателя Президиума СНЦ РАН
Чичева М.А.	-	ученый секретарь семинара, к.т.н., с.н.с. Института систем обработки изображений РАН

АЛГОРИТМЫ ТРИАНГУЛЯЦИИ НЕОРИЕНТИРОВАННЫХ ГРАФОВ В ПАРАЛЛЕЛЬНЫХ АЛГОРИТМАХ ЛОГИЧЕСКОГО ВЫВОДА ДЛЯ ВЕРОЯТНОСТНЫХ СЕТЕЙ

О.Н. Абросимова

Нижегородский государственный университет им. Лобачевского, г. Нижний Новгород

Наблюдаемые события редко могут быть описаны как прямые следствия строго детерминированных причин. На практике широко применяется вероятностное описание явлений. Обоснований тому несколько: и наличие неустранимых погрешностей в процессе экспериментирования и наблюдений, и невозможность полного описания структурных сложностей изучаемой системы, и неопределенности вследствие конечности объема наблюдений. Часть из указанных проблем решается в *вероятностных байесовых сетях* [3].

Байесова сеть – графовая модель причинно-следственных отношений между случайными переменными. Байесова сеть [3] состоит из следующих понятий и компонент:

1) множество случайных переменных и направленных связей между переменными;
2) каждая переменная может принимать одно из конечного множества взаимоисключающих значений;
3) переменные вместе со связями образуют ориентированный граф без ориентированных циклов (*Directed Acyclic Graph – DAG*);
4) каждой переменной-потомку A с переменными-предками B_1, ..., B_n приписывается таблица условных вероятностей $P(A|B_1,..., B_n)$.

Для более полного знакомства с вероятностными сетями может быть использована, например, работа [4].

Байесовы сети широко используются в медицинских исследованиях (например, диагностика заболеваний лимфатических узлов), в космических и военных системах управления (система поддержки принятия решений Vista в Центре управления полетами NASA), при создании различного программного обеспечения (управление помощником в Office), при обработке изображений и видео (синтез изображений из видеосигнала) и в других областях [3].

Одной из задач для вероятностных сетей является *задача вывода*. Суть данной задачи состоит в том, чтобы вычислить вероятности интересующих (*ненаблюдаемых*) значений переменных байесовой сети при условии, что имеется некая информация о значениях некоторых других

(*наблюдаемых*) переменных. Имея совместное распределение вероятностей, мы знаем все о нашей модели. Однако вычисление совместного распределения связано с определенными проблемами, а именно требуется большой объем памяти и вычислительных ресурсов в случае решения задачи со многими переменными. Используя предположение о том, что состояния переменных в графических моделях зависят только от состояний их соседей и не зависят от других переменных модели (допущение, применимое к любым баейсовым сетям), совместное распределение вероятности может быть разделено на более мелкие составляющие, связанные с подмножествами переменных графической модели.

Одним из алгоритмов, предназначенных для решения задачи вывода, позволяющим декомпозировать исходный граф для облегчения процесса вычислений, является метод, известный в литературе, как *Junction Tree Inference Engine* [4].

Junction Tree Inference Engine – точный алгоритм вывода на вероятностных сетях. Алгоритм осуществляет вывод на *дереве клик* (Junction Tree), которое строится из исходного графа. Для построения дерева клик требуется выполнить морализацию и триангуляцию исходного графа. *Морализация* представляет собой процедуру, которая выполняется следующим образом: всех родители каждого узла связываются в полный подграф, после чего ориентацию ребер убирают. На основании триангуляции графа известными алгоритмами (например, метод Maximum Cardinality Search [4]) формируется набор клик, т.е. граф представляется в виде связанного набора максимальных полных подграфов [4].

Время, требуемое при работе алгоритма вывода Junction Tree Inference Engine, складывается из времени обработки каждой клики дерева. Время, требуемое на обработку каждой клики, зависит от числа узлов, входящих в нее. Таким образом, время выполнения алгоритма напрямую зависит от построенного набора клик, а, следовательно, от выполненной триангуляции.

Для реализации параллельного алгоритма Junction Tree Inference Engine данная зависимость еще более актуальна. В случае распараллеливания по кликам дерева более желательна ситуация наличия множества мелких клик, чем несколько крупных. Если имеется одна или две больших клики, то часто возникает ситуация ожидания процессоров друг другом вследствие того, что процессор, получивший для обработки большую клику, занимает большее время для ее обработки в то время, как остальные процессоры простаивают в ожидании [1].

Доказано [6], что поиск триангуляции, порождающей клики минимального размера – *NP*-полная задача.

Алгоритм One Step Look Ahead Triangulation (триангуляция с просмотром на шаг вперед) описан в [4]. Это итерационный оптимизирую-

щий алгоритм. В кратком виде он может быть представлен следующем образом.

На каждой итерации оптимизируется некоторый критерий, заданный на подмножестве вершин графа.

Алгоритм One Step Look Ahead Triangulation.

Пусть все вершины незанумерованы. Установить счетчик $i:=n$,

Пока есть незанумерованные вершины, повторять:

1) выбрать любую незанумерованную вершину v, оптимизирующую критерий $c(v)$;
2) пометить ее номером i;
3) сформировать C_i – множество, содержащее всех незанумерованных соседей помеченной вершины v;
4) добавить ребра в граф так, чтобы C_i было полным подграфом;
5) исключить помеченную вершину v и уменьшить i на 1.

Качество алгоритма с точки зрения вычислительной сложности в приложениях, связанных с вероятностными сетями, будет зависеть от критерия $c(v)$, который используется при выборе вершин.

В качестве критерия $c(v)$ могут быть использованы [4]:

– число ребер, которое нужно добавить, чтобы C_i был полным подграфом;

– число еще не занумерованных соседей вершины v.

Первый критерий подходит для моделей с любыми типами переменных. Второй критерий – только для моделей с дискретными случайными переменными. Используя различные критерии, можно получать различные триангуляции для одного графа.

Для оценки эффективности распараллеливания алгоритма Junction Tree Inference Engine был введен показатель, называемый *коэффициентом максимально возможного ускорения*. Он рассчитывается, как отношение веса всего дерева к весу клики, содержащей наибольшее число узлов. Вес клики – величина, определяющая трудоемкость выполнения операция для узлов, входящих в клику. Вес всего дерева – сумма весов всех клик, его составляющих.

Коэффициент максимально возможного ускорения определяет теоретическое ускорение, которое может быть достигнуто при распараллеливании по кликам. Этот показатель можно применять и при оценке эффективности триангуляции. Чем выше коэффициент ускорения, тем лучше сбалансировано дерево клик и лучше триангуляция.

Кроме коэффициента максимально возможного ускорения для оценки качества триангуляции будем использовать следующие показатели: количество ребер, добавляемых при триангуляции, и число узлов в клике максимального размера.

Ниже представлены алгоритмы, которые предлагаются для решения обозначенной проблемы.

Алгоритм 1. Выделение дерева-остова

В описываемом подходе для морализованного графа строится дерево-остов. Для каждого ребра, не вошедшего в дерево-остов, необходимо построить цикл, который образуется при добавлении его к остову. Построенные циклы могут иметь сложную структуру, т.е. один цикл может являться частью другого или целиком входить в какой-то цикл. Используя некоторые допущения, построенные циклы можно сократить и исключить уже триангулированные участки.

После исключения общих участков все циклы можно поделить на классы. В зависимости от того, какому классу принадлежит цикл, выполняется или не выполняется его триангуляция.

В результате такой схемы будет триангулирована только часть циклов, и, следовательно, исходный граф полностью триангулирован не будет. Для получения триангулированного графа после обработки циклов необходимо запустить алгоритм One Step Look Ahead Triangulation, который добавит недостающие ребра.

Алгоритм 2. Формирование кластеров

Предыдущий алгоритм может быть немного модифицирован. После того, как выделены и обработаны все циклы, необходимо их собрать в один подграф и триангулировать его алгоритмом One Step Look Ahead Triangulation.

Алгоритм 3. Просмотр на несколько шагов вперед

В основе описываемого алгоритма лежит алгоритм One Step Look Ahead Triangulation. Алгоритм One Step Look Ahead Triangulation на каждом шаге выбирает узел с минимальным значением критерия и, таким образом, анализирует ситуацию только на один шаг вперед. Если просмотреть значения критериев для выбираемых вершин на всех шагах, то можно заметить, что они ведут себя, как правило, как показано на рис. 1.

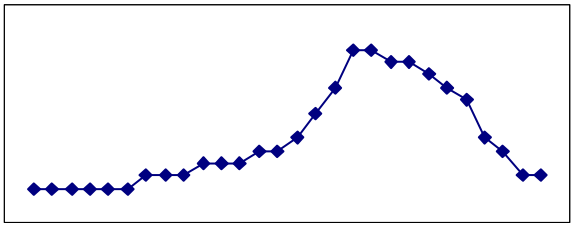


Рис. 1. Поведение критерия

Из рис. 1 видно, что существует некоторый пик среди значений критерия и возникает вопрос, нельзя ли этот пик обойти. Для решения

этой задачи предлагается алгоритм One Step Look Ahead Triangulation развить на несколько шагов вперед, т.е. на каждом шаге выбирать узлы так, чтобы избежать образования пика. Рассмотрим случай просмотра на 2 шага вперед.

Для всех узлов $i = \overline{1,n}$ морализованного графа выполнять шаги:

1) посчитать для узла *i* значение критерия *c(i)*;
2) посчитать для всех узлов $j = \overline{1,n}$, $j \neq i$ значения критерия *c(j)*, с учетом того, что выбрана вершина *i* на шаге 1.

Узел *i* теперь выбирается как узел с минимальным суммарным критерием *c(i) + c(j)* после прохода по всем узлам $i = \overline{1,n}$.

Схема может быть расширена на произвольное число шагов.

Описанная выше схема представляет собой перебор всех возможных вариантов и по этой причине требует больших временных затрат. Для графов с числом узлов уже более 50 просмотр на 4, 5 шагов вперед требует значительного времени.

Для сокращения времени работы предлагается следующая идея. Так как алгоритм выбирает узел, исходя из минимального значения критерия, то уже на первом шаге узлы с большим значением критерия рассматривать нет смысла. Таким образом, предлагается на первом шаге вычислить значение критерия для всех узлов, выбрать узлы с минимальным значением критерия и узлы, значение критерия для которых отличается от минимального на некоторую заданную величину Δ.

Идею, описанную выше, можно развить далее. Не только на первом шаге, но и на всех последующих выбирать не все узлы, а узлы с минимальным значением критерия *c(j)* и те, значение критерия для которых отличается от минимального на заданную величину Δ.

Алгоритм 4. Разрушение максимальной клики

Данный подход связан с тем, чтобы, зная максимальную клику в триангулированном графе, попытаться ее уменьшить. Для получения максимальной клики можно использовать алгоритм One Step Look Ahead Triangulation. Под максимальной кликой понимается клика, содержащая наибольшее число узлов.

После того, как максимальная клика найдена, нужно выполнить триангуляцию с узлов, которые входят в эту клику. Порядок обработки узлов, входящих в максимальную клику, играет существенную роль, т.к. неправильно выбранный порядок может привести к добавлению лишних ребер. Чтобы избегать подобных ситуаций, предлагается выбрать только один узел из максимальной клики, причем такой, который в морализованом графе имеет наименьшее число соседей (или минимальную степень), а затем запускать алгоритм One Step Look Ahead Triangulation для всех узлов графа, начиная с этого узла.

Результаты экспериментов:

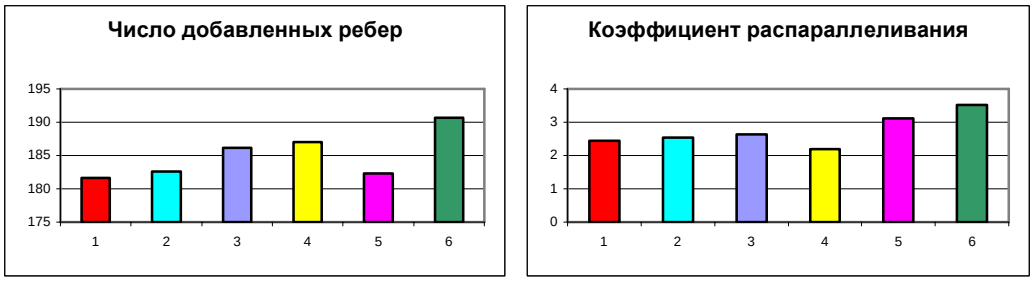


Рис. 2. Сравнение результатов работы экспериментов

Алгоритм, связанный с формированием кластеров, дает худший результат. Использование этого алгоритма приводит к существенному добавлению лишних ребер, существенному снижению коэффициента распараллеливания и увеличению максимальной клики.

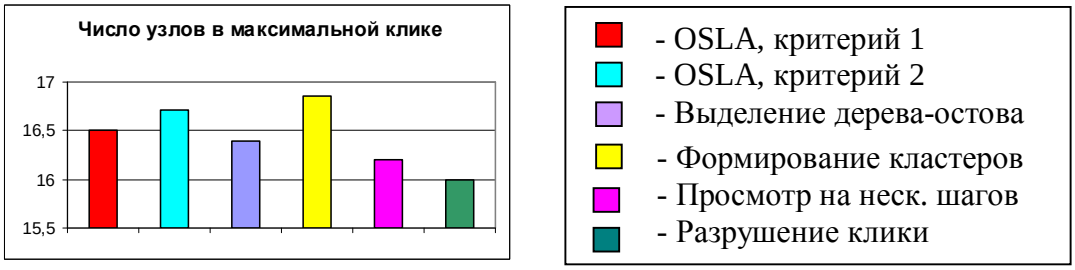


Рис. 3. Сравнение результатов работы экспериментов

Для алгоритма выделения остова можно заметить, что этот алгоритм добавляет большее количество ребер по сравнению с алгоритмом One Step Look Ahead.

Для алгоритма One Step Look Ahead лучше использовать в качестве критерия число ребер, которое нужно добавить, чтобы подграф C_i был полным подграфом. В этом случае добавляется меньше ребер в граф, значение коэффициента распараллеливания больше и максимальная клика в среднем меньше.

Для алгоритма прохода на несколько шагов вперед наилучших результатов удалось добиться при использовании так называемой модифицированной схемы при проходе на 3 шага вперед. В этом алгоритме удалось добиться наилучших результатов по двум наиболее существенным показателям – число узлов в максимальной клике и значение коэффициента распараллеливания.

Алгоритм разрушения максимальной клики – наиболее эффективный алгоритм из всех описанных выше. Он проигрывает алгоритму One Step Look Ahead по числу добавляемых ребер, но с большим отрывом лидирует по двум другим показателям.

Работа выполнялась в рамках проекта «Масштабируемые алгоритмы вывода и обучения графических моделей в рамках библиотеки PNL».

Литература

1. *Абросимова О.Н., Белов С.А., Сысоев А.В.* Балансировка вычислительной нагрузки для параллельных алгоритмов на вероятностных сетях. Высокопроизводительные параллельные вычисления на кластерных системах. Материалы третьего Международного научно-практического семинара // Под ред. проф. Р. Г. Стронгина. Н. Новгород: Изд-во Нижегородского госуниверситета, 2003.
2. Лекции по теории графов под ред. В.А. Емеличева. М: Наука, 1990.
3. Научная сессия МИФИ-2003. V Всероссийская научно-техническая конференция «Нейроинформатика-2003»: Лекции по нейроинформатике. Часть 1. М.: МИФИ, 2003. - 188 с.
4. *Cowell R., Dawid A., Lauritzen L., Spiegelhalter D.* Probabilistic networks and expert systems, Springer-Verlag, NewYork. 1999. - 321 с.
5. *Huang C., Darwiche A.* Inference in Belief Networks: A Procedural Guide.
6. *Yannakakis M.* Computing the minimum fill-in is NP-complete // SIAM Journal on Algebraic and Discrete Methods. 1981.

ОБОБЩЕННЫЙ КОНВЕЙЕРНЫЙ ПАРАЛЛЕЛИЗМ И РАСПРЕДЕЛЕНИЕ ОПЕРАЦИЙ И ДАННЫХ МЕЖДУ ПРОЦЕССОРАМИ

Е.В. Адуцкевич

Институт математики НАН Беларуси, г. Минск, Беларусь

Введение

Для отображения алгоритмов, заданных последовательными программами, на параллельные компьютеры с распределенной памятью требуется распределить операции и данные алгоритма между процессорами, а также установить порядок выполнения операций и обмена данными.

Среди важнейших задач различают, в частности, следующие задачи: временное отображение [1], согласование распределения операций и данных по процессорам [1–3], пространственно-временное отображение [4]. Существенным этапом в решении перечисленных задач является поиск функций (таймирующие функции, функции размещения, развертки графов), удовлетворяющих некоторым ограничениям.

Одной из эффективных схем распараллеливания является использование нескольких таймирующих функций для получения конвейерного параллелизма [1, 4]. Такая схема имеет ряд достоинств: возможность получения высокой степени параллелизма без явного указания параллельных циклов, регулярный код, упрощенная синхронизация. Применение этой схемы с использованием части таймирующих функций в качестве функций размещения позволяет решить проблему пространственного и временного отображения. В то же время, при таком подходе

требует еще отдельного рассмотрения проблема согласования распределения операций и данных по процессорам.

В этой работе предлагается единая процедура для получения конвейерного параллелизма и решения перечисленных задач. Согласованное решение этих задач позволяет получать таймирующие функции и функции размещения операций и данных, наилучшим образом подходящие друг другу.

1. Основные определения

Будем рассматривать аффинные гнезда циклов, т.е. будем предполагать, что выражения индексов элементов массивов и границы изменения параметров циклов являются аффинными функциями от параметров циклов и внешних переменных.

Пусть в гнезде циклов имеется K операторов S_β и используется L массивов a_l Область изменения параметров гнезда циклов для оператора S_β будем называть индексной областью и обозначать V_β. Область изменения индексов l-го массива будем обозначать W_l Обозначим n_β – число циклов, окружающих оператор S_β , v_l – размерность l-го массива, тогда $V_\beta \subset \mathbf{Z}^{n_\beta}$, $W_l \subset \mathbf{Z}^{v_l}$.

Индексы элементов l-го массива, встречающегося в операторе S_β и относящегося к q-му входу элементов этого массива в оператор, будем выражать аффинной функцией $\overline{F}_{l,\beta,q}(J) = F_{l,\beta,q}J + G_{l,\beta,q}N + f^{(l,\beta,q)}$, где $J \in V_\beta$, $N \in \mathbf{Z}^e$ – вектор внешних переменных алгоритма, матрицы $F_{l,\beta,q} \in \mathbf{Z}^{v_l \times n_\beta}$, $G_{l,\beta,q} \in \mathbf{Z}^{v_l \times e}$ и векторы $f^{(l,\beta,q)} \in \mathbf{Z}^{v_l}$ не зависят от внешних переменных.

Реализацию (выполнение) оператора S_β при конкретных значениях β и вектора параметров цикла J будем называть операцией и обозначать $S_\beta(J)$.

Операция $S_\beta(J)$, $J \in V_\beta$, зависит от операции $S_\alpha(I)$, $I \in V_\alpha$, если:

1) $S_\alpha(I)$ выполняется раньше $S_\beta(J)$;

2) $S_\alpha(I)$ и $S_\beta(J)$ используют один и тот же элемент какого-либо массива и, по крайней мере, одно из использований есть переопределение (изменение) элемента;

3) между операциями $S_\alpha(I)$ и $S_\beta(J)$ этот элемент не переопределяется. Зависимость операции $S_\beta(J)$ от операции $S_\alpha(I)$ будем обозначать $S_\alpha(I) \to S_\beta(J)$.

Обозначим $P=\{(\alpha,\beta)\mid \exists I\in V_\alpha, J\in V_\beta, S_\alpha(I)\to S_\beta(J)\}$. Множество P определяет пары зависимых операторов. Для каждой пары $(\alpha,\beta)\in P$ обозначим $V_{\alpha,\beta}=\{J\in V_\beta \mid \exists S_\alpha(I)\to S_\beta(J)\}$.

Функции $\overline{\Phi}_{\alpha,\beta}: V_{\alpha,\beta}\to V_\alpha$ такие, что если $S_\alpha(I)\to S_\beta(J)$, $I\in V_\alpha$, $J\in V_{\alpha,\beta}\subseteq V_\beta$, то $I=\overline{\Phi}_{\alpha,\beta}(J)$, назовем функциями зависимостей. Будем предполагать, что функции зависимостей являются аффинными: $\overline{\Phi}_{\alpha,\beta}(J)=\Phi_{\alpha,\beta}J+\Psi_{\alpha,\beta}N-\varphi^{(\alpha,\beta)}$, где $J\in V_{\alpha,\beta}$, $(\alpha,\beta)\in P$, $N\in\mathbf{Z}^e$, матрицы $\Phi_{\alpha,\beta}\in\mathbf{Z}^{n_\alpha\times n_\beta}$, $\Psi_{\alpha,\beta}\in\mathbf{Z}^{n_\alpha\times e}$ и векторы $\varphi^{(\alpha,\beta)}\in\mathbf{Z}^{n_\alpha}$ не зависят от внешних переменных.

Обозначим $n=\max\limits_{1\le\beta\le K} n_\beta$. Пусть функции $t_\xi^{(\beta)}: V_\beta\to\mathbf{Z}$, $1\le\beta\le K$, $1\le\xi\le n$, ставят в соответствие каждой операции $S_\beta(J)$ алгоритма целое число $t_\xi^{(\beta)}(J)$. Пусть функции $t_\xi^{(\beta)}$ являются таймирующими функциями (t-функциями), т.е.

$$t_\xi^{(\beta)}(J)\ge t_\xi^{(\alpha)}(\Phi_{\alpha,\beta}J+\Psi_{\alpha,\beta}N-\varphi^{(\alpha,\beta)}),\quad J\in V_{\alpha,\beta},\ (\alpha,\beta)\in P. \tag{1}$$

Условия (1) называются условиями сохранения зависимостей. Будем предполагать, что функции $t_\xi^{(\beta)}(J)$ являются аффинными: $t_\xi^{(\beta)}=\tau^{(\beta,\xi)}J+b^{(\beta,\xi)}N+a_{\beta,\xi}$, где $1\le\beta\le K$, $1\le\xi\le n$, $J\in V_\beta$, $\tau^{(\beta,\xi)}\in\mathbf{Z}^{n_\beta}$, $b^{(\beta,\xi)}, N\in\mathbf{Z}^e$, $a_{\beta,\xi}\in\mathbf{Z}$. Предполагается, что $\tau^{(\beta,\xi)}$, $b^{(\beta,\xi)}$, $a_{\beta,\xi}$ не зависят от N.

Пусть функции $d_\xi^{(l)}: W_l\to\mathbf{Z}$, $1\le l\le L$, $1\le\xi\le n$, ставят в соответствие каждому элементу $a_l(F)$ массива a_l целое число $d_\xi^{(l)}(F)$. Будем интерпретировать r-мерное пространство, в которое отображают функции $d_\xi^{(l)}$, как r-мерное пространство виртуальных процессоров. Мы будем рассматривать аффинные функции $d_\xi^{(l)}$: $d_\xi^{(l)}(F)=\eta^{(l,\xi)}F+z^{(l,\xi)}N+y_{l,\xi}$, где $1\le l\le L$, $1\le\xi\le r$, $F\in W_l$, $\eta^{(l,\xi)}\in\mathbf{Z}^{\nu_l}$, $z^{(l,\xi)}, N\in\mathbf{Z}^e$, $y_{l,\xi}\in\mathbf{Z}$. Предполагается, что $\eta^{(l,\xi)}$, $z^{(l,\xi)}$, $y_{l,\xi}$ не зависят от N.

2. Получение конвейерного параллелизма с помощью таймирующих функций

Пусть найдены n независимых наборов t-функций $t_\xi^{(1)},\ldots,t_\xi^{(K)}$, $1\le\xi\le n$. Требование независимости наборов будем формализовать условием

$$\operatorname{rang} T^{(\beta)}=n_\beta,\quad 1\le\beta\le K, \tag{2}$$

где $T^{(\beta)}$ – матрица размера $n\times n_\beta$, строки которой составлены из векторов $\tau^{(\beta,\xi)}$.

Будем использовать r наборов функций $t_\xi^{(\beta)}$, $1\le\beta\le K$, $1\le\xi\le r<n$, для пространственного отображения операций алгоритма в r-мерное

пространство виртуальных процессоров, а оставшиеся $n-r$ наборов – для упорядочения (выполнения в лексикографическом порядке) вычислений, выполняемых процессорами. Примем за единицу времени время, необходимое для выполнения самой длительной итерации алгоритма и обмена данными.

Утверждение 1. *Пусть M – наименьшее целое число, параметрически зависящее от внешних переменных и пределов изменения параметров гнезда циклов и превосходящее любую из внешних переменных и любой из пределов изменения параметров циклов. В рамках описанной схемы распараллеливания $O(M^r)$ виртуальных процессоров могут реализовать алгоритм за время $O(M^{n-r})$.*

Таким образом, описанная схема распараллеливания позволяет получить наилучшее по порядку время реализации алгоритма в r-мерном пространстве виртуальных процессоров. При этом обмен данными между любыми двумя процессорами сводится к передаче данных от одного и того же процессора другому процессору. Получаемый способ обработки данных можно рассматривать как обобщение классической конвейерной обработки.

3. Постановка задачи

Пусть $t_\xi^{(1)},...,t_\xi^{(K)}$, $1\le\xi\le n$, – независимые наборы таймирующих функций и $d_\xi^{(1)},...,d_\xi^{(L)}$, $1\le\xi\le r$, – наборы функций размещения массивов данных между процессорами. Определим ограничения, которым должны удовлетворять функции $t_\xi^{(\beta)}$ и $d_\xi^{(l)}$.

Условия сохранения зависимостей (1) можно записать в виде:

$$(\tau^{(\beta,\xi)}-\tau^{(\alpha,\xi)}\Phi_{\alpha,\beta})J+(b^{(\beta,\xi)}-\tau^{(\alpha,\xi)}\Psi_{\alpha,\beta}-b^{(\alpha,\xi)})N+\tau^{(\alpha,\xi)}\varphi^{(\alpha,\beta)}+a_{\beta,\xi}-a_{\alpha,\xi}\ge 0,\quad J\in V_{\alpha,\beta},\ (\alpha,\beta)\in P,\ 1\le\xi\le n. \tag{3}$$

Условия для распределения данных между процессорами, не требующего обменов данными, можно получить, рассмотрев величины

$$\delta_\xi^{l,\beta,q}(J)=t_\xi^{(\beta)}(J)-d_\xi^{(l)}(\overline{F}_{l,\beta,q}(J))=$$
$$=(\tau^{(\beta,\xi)}-\eta^{(l,\xi)}F_{l,\beta,q})J+(b^{(\beta,\xi)}-\eta^{(l,\xi)}G_{l,\beta,q}-z^{(l,\xi)})N+a_{\beta,\xi}-\eta^{(l,\xi)}f^{(l,\beta,q)}-y_{l,\xi},$$
$$1\le\xi\le r,$$

характеризующие для фиксированных l,β,q,J расстояние между процессором, в котором выполняется операция, и процессором, в котором хранится необходимый для выполнения операции элемент массива. Эти условия можно записать в виде:

$$\tau^{(\beta,\xi)}-\eta^{(l,\xi)}F_{l,\beta,q}=0,\quad b^{(\beta,\xi)}-\eta^{(l,\xi)}G_{l,\beta,q}-z^{(l,\xi)}=0,\quad a_{\beta,\xi}-\eta^{(l,\xi)}f^{(l,\beta,q)}-y_{l,\xi}=0. \tag{4}$$

Если первые два условия из (4) выполняются, а третье условие не выполняется, то соответствующее распределение операций и данных между процессорами требует только локальных, т.е. не зависящих от J и N, коммуникаций.

Обозначим

$\mathbf{t}^{(\xi)} = (\tau^{(1,\xi)},...,\tau^{(K,\xi)},\eta^{(1,\xi)},...,\eta^{(L,\xi)},b^{(1,\xi)},...,b^{(K,\xi)},z^{(1,\xi)},...,z^{(L,\xi)},a_{1,\xi},...,a_{K,\xi},\ y_{1,\xi},...,y_{L,\xi})$ – вектор, составленный из параметров функций $t_\xi^{(\beta)}$ и $d_\xi^{(l)}$, $1 \le \xi \le n$, и запишем условия (3), (4) в матричном виде:

$$\mathbf{t}^{(\xi)}\hat{\mathbf{\Phi}}_{\alpha,\beta}J + \mathbf{t}^{(\xi)}\hat{\mathbf{\Psi}}_{\alpha,\beta}N + \mathbf{t}^{(\xi)}\hat{\boldsymbol{\phi}}^{(\alpha,\beta)} \ge 0, \quad J \in V_{\alpha,\beta},\ (\alpha,\beta) \in P, \tag{5}$$

$$\mathbf{t}^{(\xi)}\Delta^F_{l,\beta,q} = 0, \quad \mathbf{t}^{(\xi)}\Delta^G_{l,\beta,q} = 0, \quad \mathbf{t}^{(\xi)}\Delta^f_{l,\beta,q} = 0. \tag{6}$$

Задачей этой работы является разработка способа получения векторов $\mathbf{t}^{(\xi)}$, $1 \le \xi \le n$, таких, что выполняются условия (2), векторы $\mathbf{t}^{(1)},...,\mathbf{t}^{(n)}$ удовлетворяют условиям (5) и векторы $\mathbf{t}^{(1)},...,\mathbf{t}^{(r)}$ удовлетворяют условиям (6) для как можно большего числа l, β, q.

4. Процедура получения таймирующих функций и функций размещения операций и данных по процессорам

Введем обозначения:

$T_{1:0}^{(\beta)} = 0^{(n_\beta)}$, $T_{1:\xi}^{(\beta)}$ – матрица, строки которой составлены из векторов $\tau^{(\beta,i)}$, $1 \le i \le \xi$;

$S_\beta^{(1)} = \mathbf{Z}^{n_\beta}$, $S_\beta^{(\xi)} = \{s \in \mathbf{Z}^{n_\beta} \mid \tau^{(\beta,i)}s = 0,\ 1 \le i \le \xi - 1,\ s \ne 0\}$, $2 \le \xi \le n$.

Утверждение 2. Пусть $\operatorname{rang} T_{1:\xi-1}^{(\beta)} = r$, $r < n_\beta$, и $s_\beta^{(\xi)} \in S_\beta^{(\xi)}$. Если $\tau^{(\beta,\xi)}s_\beta^{(\xi)} \ne 0$, то $\operatorname{rang} T_{1:\xi}^{(\beta)} = r + 1$.

Условие утверждения 2 можно записать в матричном виде:

$$|\mathbf{t}^{(\xi)}\mathbf{s}_\beta^{(\xi)}| \ge 1. \tag{7}$$

Условие (7) можно использовать для нахождения n векторов $\mathbf{t}^{(\xi)}$, определяющих n независимых, т.е. удовлетворяющих условию (2), наборов t-функций $t_\xi^{(1)},...,t_\xi^{(K)}$. Для этого при последовательном нахождении векторов $\mathbf{t}^{(\xi)}$, $\xi = 1,2,...,n$, необходимо следить за тем, чтобы условие (7) выполнялось для таких β, для которых $n - \xi + 1 = n_\beta - \operatorname{rang} T_{1:\xi-1}^{(\beta)}$.

Условия сохранения зависимостей (5) можно свести к системе неравенств [1] $\mathbf{t}^{(\xi)}D_{\alpha,\beta} \ge 0$, где $D_{\alpha,\beta}$ – матрица, которую можно получить, исходя из вида области $V_{\alpha,\beta}$. Если ввести дополнительные векторные переменные $z_{\alpha,\beta}$, то эту систему неравенств можно свести к уравнениям:

$$\beta^{(\xi)} D_{\alpha,\beta} - z_{\alpha,\beta} = 0, \quad z_{\alpha,\beta} \geq 0. \tag{8}$$

Согласно постановке задачи, для всех n векторов $\beta^{(\xi)}$ должны выполняться условия (8). Кроме того, чем меньше значение координат вектора $z_{\alpha,\beta}$, тем меньше значения разностей $t_{\xi}^{(\beta)}(J) - t_{\xi}^{(\alpha)}(I)$, $J \in V_{\alpha,\beta}$, $I = \overline{\Phi}_{\alpha,\beta}(J)$, и потенциально меньше время реализации алгоритма. Таким образом, удовлетворение условий сохранения зависимостей и задачу уменьшения времени реализации алгоритма можно свести к задаче минимизации (обнуления, если возможно) векторов $z_{\alpha,\beta}$, удовлетворяющих условиям (8).

Задачу получения размещения операций и данных между процессорами, обеспечивающего только локальные коммуникации, можно свести (см. (6) к задаче обнуления векторов $z^F_{l,\beta,q}$ и $z^G_{l,\beta,q}$, удовлетворяющих условиям:

$$|\beta^{(\xi)} \Delta^F_{l,\beta,q}| - z^F_{l,\beta,q} = 0, \quad |\beta^{(\xi)} \Delta^G_{l,\beta,q}| - z^G_{l,\beta,q} = 0, \tag{9}$$

не требующего обмена данными, – к задаче обнуления величин $z^f_{l,\beta,q}$, удовлетворяющих условиям:

$$|\beta^{(\xi)} \Delta^f_{l,\beta,q}| - z^f_{l,\beta,q} = 0, \tag{10}$$

если удалось добиться равенств $z^F_{l,\beta,q} = 0^{(n_\beta)}$, $z^G_{l,\beta,q} = 0^{(e)}$.

Из сказанного следует, что задачу согласованного получения конвейерного параллелизма и распределения операций и данных между процессорами можно свести к задаче нахождения n векторов $\beta^{(\xi)}$, исходя из условий минимизации векторов $z_{\alpha,\beta}$, обнуления для как можно большего числа l, β, q векторов $z^F_{l,\beta,q}$ и $z^G_{l,\beta,q}$, обнуления величин $z^f_{l,\beta,q}$ (если удалось обнулить векторы $z^F_{l,\beta,q}$ и $z^G_{l,\beta,q}$), при выполнении ограничений (8)–(10) и ограничений (7) для таких β, что $n - \xi + 1 = n_\beta - \operatorname{rang} T^{(\beta)}_{1:\xi-1}$.

Обозначим: D – множество матриц $D_{\alpha,\beta}$; D_F, D_G – множество матриц $\Delta^F_{l,\beta,q}$, $\Delta^G_{l,\beta,q}$, фигурирующих в условиях распределения данных между процессорами, требующих только локальных коммуникаций; D_f – множество векторов $\Delta^f_{l,\beta,q}$, фигурирующих в условиях распределения данных между процессорами, не требующих обмена данными; $L^{(\xi)} = \{\beta \mid n - \xi + 1 = n_\beta - \operatorname{rang} T^{(\beta)}_{1:\xi-1}\}$;

$$\rho(z_{\alpha,\beta}, z^F_{l,\beta,q}, z^G_{l,\beta,q}, z^f_{l,\beta,q}) = \sum_{\alpha,\beta} \lambda_{\alpha,\beta} z_{\alpha,\beta} + \sum_{l,\beta,q} (\lambda^F_{l,\beta,q} z^F_{l,\beta,q} + \lambda^G_{l,\beta,q} z^G_{l,\beta,q} + \lambda^f_{l,\beta,q} z^f_{l,\beta,q}),$$

где суммирование $\sum_{\alpha,\beta}$ осуществляется по всем таким α,β, что $D_{\alpha,\beta} \in D$, суммирование $\sum_{l,\beta,q}$ – по всем таким l,β,q, что $\Delta^F_{l,\beta,q} \in D_F$, $\Delta^G_{l,\beta,q} \in D_G$, $\Delta^f_{l,\beta,q} \in D_f$; $\lambda_{\alpha,\beta}$, $\lambda^F_{l,\beta,q}$, $\lambda^G_{l,\beta,q}$, $\lambda^f_{l,\beta,q}$ – векторы весовых коэффициентов, соответствующих столбцам матриц $D_{\alpha,\beta}$, $\Delta^F_{l,\beta,q}$, $\Delta^G_{l,\beta,q}$ и векторам $\Delta^f_{l,\beta,q}$. Весовые коэффициенты определяются исходя из ряда условий для отражения роли этих столбцов и векторов в согласованном получении конвейерного параллелизма и распределении операций и данных между процессорами, а также для выражения предпочтений при размещении массивов данных.

Предложим процедуру решения поставленной задачи. Процедура является рекурсивной и содержит n рекурсий. Результат выполнения ξ-той рекурсии – вектор $\beta^{(\xi)}$.

Процедура (получение таймирующих функций и функций размещения операций и данных по процессорам).

Положить сначала $\xi = 1$.

Шаг 1. Выбрать вектор $s^{(\xi)}_\beta \in S^{(\xi)}_\beta$, $\beta \in L^{(\xi)}$. Найти вектор $\beta^{(\xi)}$, как решение оптимизационной задачи:

$$\min\Big\{\rho(z_{\alpha,\beta}, z^F_{l,\beta,q}, z^G_{l,\beta,q}, z^f_{l,\beta,q}) \;\Big|\; |\beta^{(\xi)} s^{(\xi)}_\beta| \ge 1, \beta \in L^{(\xi)},\ \beta^{(\xi)} D_{\alpha,\beta} - z_{\alpha,\beta} = 0, D_{\alpha,\beta} \in D,$$
$$|\beta^{(\xi)} \Delta^F_{l,\beta,q}| - z^F_{l,\beta,q} = 0, |\beta^{(\xi)} \Delta^G_{l,\beta,q}| - z^G_{l,\beta,q} = 0, |\beta^{(\xi)} \Delta^f_{l,\beta,q}| - z^f_{l,\beta,q} = 0,$$
$$\Delta^F_{l,\beta,q} \in D_F, \Delta^G_{l,\beta,q} \in D_G, \Delta^f_{l,\beta,q} \in D_f, \xi \le r\Big\},$$

положив $\lambda^f_{l,\beta,q} = 0$ для всех l,β,q, если $\xi \le r$. Если $\xi \le r$ и есть такие l,β,q, что $z^F_{l,\beta,q} = 0^{(n_\beta)}$, $z^G_{l,\beta,q} = 0^{(e)}$, но $z^f_{l,\beta,q} \ne 0$, то перейти на шаг 2, иначе на шаг 3.

Шаг 2. Для таких l,β,q, что $z^F_{l,\beta,q} = 0^{(n_\beta)}$, $z^G_{l,\beta,q} = 0^{(e)}$, но $z^f_{l,\beta,q} \ne 0$ придать весовым коэффициентам $\lambda^f_{l,\beta,q}$ ненулевые значения и снова найти вектор $\beta^{(\xi)}$ как решение оптимизационной задачи.

Шаг 3. Определить множество $L^{(\xi+1)} = \{\beta \mid n - \xi = n_\beta - \operatorname{rang} T^{(\beta)}_{1:\xi}\}$.

Шаг 4. Если $\xi = n$, то выйти из процедуры; иначе увеличить ξ на 1 и перейти на шаг 1.

Литература

1. *Воеводин В.В., Воеводин Вл.В.* Параллельные вычисления // Санкт-Петербург. БХВ-Петербург. 2002. - 608 с.

2. *Dion M., Robert Y.* Mapping affine loop nests // Parallel Computing, 1996. Vol. 22. P. 1373-1397.

3. *Лиходед Н.А.* Распределение операций и массивов данных между процессорами. // Программирование. 2003. № 2. С. 73-80.

4. *Lim A.W., Lam M.S.* Maximizing parallelism and minimizing synchronization with affine partitions // Parallel Computing. 1998. Vol. 24. № 3-4. P. 445-475.

АДАПТАЦИЯ ПОСЛЕДОВАТЕЛЬНОЙ МЕТОДИКИ РЕШЕНИЯ НЕЛИНЕЙНЫХ ЗАДАЧ ДИНАМИКИ КОНСТРУКЦИЙ ДЛЯ МНОГОПРОЦЕССОРНЫХ ЭВМ

В.Г. Баженов, А.В. Гордиенко, А.И. Кибец, П.В. Лаптев

НИИ механики ННГУ, г. Нижний Новгород

Введение

Численное решение трехмерных нелинейных задач нестационарного деформирования конструкций, как правило, трудоемко и требует применения достаточно мощной вычислительной техники. Одним из наиболее перспективных путей достижения высокой производительности является создание кластеров – многопроцессорных вычислительных систем (МВС), имеющих локальную память, и связанных между собой однородной коммутационной сетью, предназначенных для быстродействующей параллельной обработки. Большое количество современных пакетов прикладных программ для решения задач математической физики основано на алгоритмах последовательных вычислений (метод конечных элементов, конечно-разностный метод, метод граничных элементов и т.д.). Поэтому эффективное применение разработанных ранее программ с использованием таких МВС требует их модернизации с учетом особенностей параллельных вычислений. Целью настоящей работы является адаптация алгоритмов программного комплекса «Динамика-3» для организации параллельных вычислений на многопроцессорных ЭВМ. Ожидаемый эффект заключается не только в ускорении вычислений, но и в увеличении возможного количества используемых конечных элементов в дискретной модели расчетной области, а также моделировании более сложных конструкций.

Численная методика решения, и ее программная реализация

Программный комплекс «Динамика-3» предназначен для решения трехмерных задач нестационарного деформирования конструкций, включающих массивные тела и оболочки. Конструкции могут подвергаться силовым и кинематическим воздействиям. Отдельные конструктивные элементы могут вступать в динамический контакт друг с другом или взаимодействовать с внешними телами. В рамках пакета реализована следующая численная методика. Определяющая система уравнений сплошной среды формулируется в переменных Лагранжа. Уравнение движения выводятся из вариационного принципа Журдена. Кинематические соотношения

формулируются в метрике текущего состояния. Упругопластическое деформирование материала описывается соотношениями теории течения с кинематическим и изотропным упрочнением. На контактирующих поверхностях задаются условия непроникания. Гипотезы, принятые в теории тонкостенных элементов конструкций, вводятся на этапе дискретизации определяющей системы уравнений. Благодаря этому можно упростить стыковку разных типов конструктивных элементов и учесть особенности их напряженно-деформированного состояния.

Решение определяющей системы уравнений, при заданных начальных и граничных условиях, основано на методе конечных элементов и явной конечно-разностной схеме интегрирования по времени типа «крест». Расчетная область разбивается 8-узловыми элементами, в узлах которых определяются перемещения, скорости и ускорения. С помощью изопараметрического преобразования, искаженный в общем случае КЭ, отображается на куб. Компоненты скорости перемещений узлов аппроксимируются внутри конечного элемента полилинейными функциями формы. Скорости деформаций аппроксимируются линейными функциями в виде суммы безмоментных и моментных составляющих. Основные интегралы в элементе находятся с применением формул численного интегрирования. Для этого в выбранных квадратурных точках вычисляются скорости деформаций и напряжения.

Заменяя интегрирование по области суммированием по элементам, получим дискретный аналог уравнений движения, который интегрируется по явной конечно-разностной схеме типа «крест». В дискретном аналоге уравнений движения матрица масс является диагональной и не возникает необходимости в ее обращении. Благодаря этому возможно разбиение конструкции на отдельные блоки, внутри которых вычисления на каждом временном слое производятся независимо. Сшивка блоков осуществляется путем согласования сил и ускорений между блоками, после которого происходит переход на следующий временной шаг. На рис. 1. схематично представлен последовательный алгоритм расчета.

Алгоритм адаптации численной методики для организации параллельных вычислений

Наиболее предпочтительным алгоритмом для распараллеливания упомянутой выше методики решения трехмерных задач динамики упругопластических конструкций на имеющейся вычислительной системе кластерного типа, является метод пространственной декомпозиции расчетной области (domain decomposition method). Согласно ему распараллеливание производится путем распределения вычислений в разных точках расчетной области в разные процессоры.

Рис. 1

Поскольку при численном решении трехмерных задач динамики существует необходимость обработки больших объемов информации, один узел выделяется для организации сбора и распределения данных, производимых при сшивке подобластей и моделировании контактного взаимодействия на каждом шаге интегрирования по времени.

В основу алгоритма адаптации были заложены следующие принципы:

- длины параллельно выполняемых ветвей (процессов) программы равны между собой;
- минимизация простоев из-за ожидания данных и передачи управления;
- минимизация числа и объема временных массивов для оптимизации работы с кэш-памятью;
- переход от исходной программы, работающей с полными массивами, к программе обрабатывающей только локальную порцию, распределенную на процессор: изменение размеров массивов и соответствующее ему преобразование текста программы.

Для реализации такого подхода исследуемая конструкция разделяется согласно числу процессоров на подконструкции, дискретные модели которых будут иметь, примерно, равное количество конечных элементов. В соответствии с этим база данных решения задачи разбивается на ряд фрагментов $БД_L$, которые распределяются по процессорам ($1 \le L \le K$). Массивы узлов A и конечных элементов B разделяются на фрагменты A_L, B_L, соответствующие подконструкциям, которые хранятся и обрабатываются в локальных базах данных $БД_L$ процессоров, отведенных под расчет напряженно-деформированного состояния. Процессор 0 используется для управления параллельными процессами (стыковки подконструкций). В

массивах A_L выделяются внутренние и внешние узлы, (внешние узлы могут вступать в контакт с узлами других подконструкций, внутренние – нет). Такое разделение массива данных на области позволяет обеспечить равномерную загрузку всего расчетного кластера.

Решение начально-краевой задачи осуществляется в соответствии с блок-схемой представленной на рис. 2.

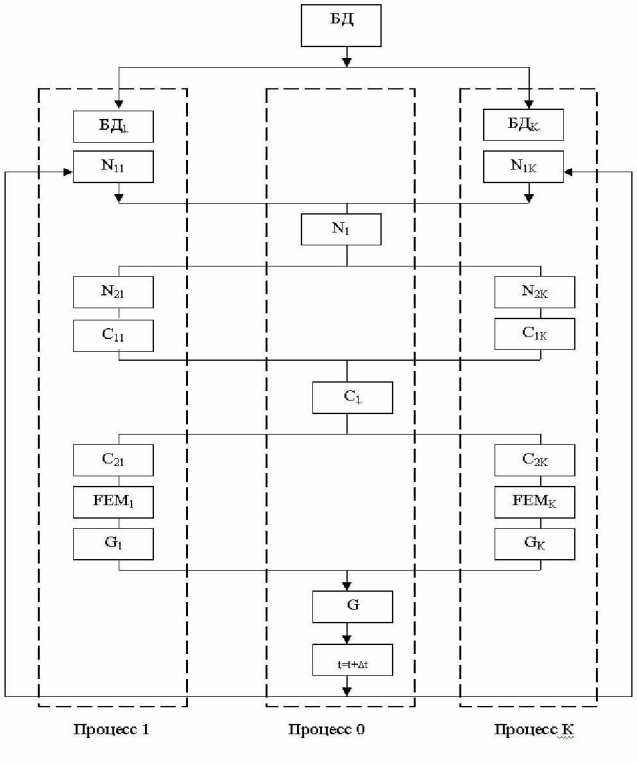


Рис. 2

Расшифровка обозначений, принятых в блок-схеме, приведена в таблице 1.

Таблица 1

1	2	3	4
1		$БД_L$	Распределение БД решения задачи по процессорам
2	Интегрирование уравнений движения	N_{1L}	Занесение информации о внешних узлах КЭ-сетки из памяти процессора L на буфер сшивки в процессоре 0
3		N_1	Определение результирующих узловых сил внешних узлов КЭ-сетки на буфере сшивки и их занесение в локальную память процессоров
4		N_{2L}	Интегрирование уравнений движения
5	Реализация условий контакта	C_{1L}	Занесение информации об узлах КЭ-сетки на буфер контакта из локальной памяти процессора L

6		C_1	Определение давления в зонах контакта и узловых сил от контактного давления
7		C_{2L}	Корректировка скорости перемещений и перемещений узлов с учетом контактного давления и их распределение из буфера контакта (процессор 0) в память процессора *L*
8		*FE* M_L	Расчет деформаций, напряжений, узловых сил от напряжений и внешней нагрузки в процессоре *L*
9		G_L	Предварительная обработка результатов счета в процессоре *L*
10		G	Занесение результатов счета в графическую базу данных

Последовательность действий при решении начально-краевой задачи имеет следующий вид:

1. В процессоре 0 формируется или считывается сформированная заранее начальная база данных решения задачи *БД*, которая разделяется пользователем на фрагменты и распределяется по другим процессорам;
2. Выполняется интегрирование уравнений движения узлов по времени:

Ÿ просматриваются все базы данных $БД_L$ и в рабочий массив *C* (буфер сшивки), расположенный в процессоре 0, из массивов A_L заносятся узловые силы внешних узлов подконструкций (процедуры N_{1L} на рис. 2);

Ÿ для идентичных узлов подконструкций в процессоре 0 вычисляются результирующие узловых сил (процедура N_1 на рис. 2);

Ÿ результирующие узловые силы распределяются по локальным базам данных $БД_L$, после чего осуществляется интегрирование уравнений движения и корректировка узловых скоростей перемещений и перемещений (процедуры N_{2L} на рис. 2).

3. Проверяются условия контактного взаимодействия деформируемых тел:

Ÿ просматриваются все базы данных $БД_L$ и в рабочий массив *C* (буфер контакта) в процессоре 0, из массивов A_L, B_L заносятся информация о внешних узлах и гранях КЭ (координаты и скорости перемещений узлов), попавших в зоны вероятного контакта, определенные на текущем временном слое без учета сил от контактного давления (процедуры C_{1L} на рис. 2);

Ÿ в процессоре 0 проверяются условия непроникания, вычисляются узловые силы от контактного давления (процедура C_1 на рис. 2);

Ÿ контактные узловые силы распределяются по массивам AL локальных баз данных БДL, скорости перемещений и координаты узлов

корректируются в AL с учетом сил от контактного давления (процедуры C2L) .

4. В конечных элементах подконструкций определяются компоненты тензоров деформаций и напряжений, вычисляются узловые силы от напряжений и внешнего давления (процедуры FEML);
5. Результаты счета обрабатываются (процедуры G_L) и передаются на центральный процессор, где с помощью процедуры G_1 заносятся в графические базы данных.
6. Осуществляется переход на следующий временной слой.

Таким образом, обработка информации в изложенном алгоритме на параллельных процессорах преимущественно ведется автономно. Необходимость в обмене информацией между процессорами возникает дважды: при сшивке подконструкций: на этапе интегрирования уравнений движения и при численном моделировании условий контакта на несогласованных сетках. С целью достижения наибольшей эффективности использования вычислительной техники, объединенной в обычную локальную сеть, для модернизации алгоритмов ППП «Динамика-3» была выбрана библиотека MPI.

РАЗРАБОТКА И РЕАЛИЗАЦИЯ СИСТЕМЫ ФУНКЦИОНАЛЬНОГО ПАРАЛЛЕЛЬНОГО ПРОГРАММИРОВАНИЯ НА ВЫЧИСЛИТЕЛЬНЫХ СИСТЕМАХ

С.Е. Бажанов, В.П. Кутепов, Д.А. Шестаков

Московский Энергетический Институт (ТУ), г. Москва

Введение

В докладе обсуждается реализуемый на кафедре прикладной математики проект создания системы функционального параллельного программирования для кластерных систем [1].

Главная особенность проекта состоит в том, что в нем комплексно на единой основе решаются взаимосвязанные задачи:

– создание системы функционального композиционного параллельного программирования, включающей высокоуровневый язык и инструментальные средства программирования: подсистемы верификации программ, анализа их структурной и вычислительной сложности, эквивалентных преобразований и контроля типовой правильности [1];

– разработка системы управления процессом параллельного выполнения функциональных программ на вычислительных системах, в частности на кластерах.

Работа выполнена при поддержке РФФИ, проект 03-01-00588

1. Язык функционального программирования

Большинство существующих языков функционального программирования (Lisp, ML, Hope, Haskell и др.) создано на основе λ-исчисления, базовыми понятиями которого являются λ-связывание переменных в выражении, конкатенация выражений и λ-аппликация. Хотя рекурсия выражается в λ-исчислении (через парадоксальный комбинатор Карри), тем не менее, по практическим соображениям во всех λ-основанных языках в явном виде используется рекурсия. Простой механизм различения связанных и свободных переменных в задании функций, единообразный способ задания функций высших типов – главные достоинства этой ветви функциональных языков.

В тоже время, λ-функции плохо поддаются структуризации и распараллеливанию, да и сам подход к заданию функций путем λ-связывания переменных в выражении плохо согласуется с общепринятой трактовкой функций и обычно используемыми способами их задания (использование последовательной композиции, условных конструкций, подстановки и рекурсивных определений).

Созданный в рамках проекта язык функционального параллельного программирования является композиционным, в нем используется четыре простых операции композиции функций: последовательная композиция, конкатенация (через эти обе операции выражается оператор подстановки), условие и объединение (графиков) ортогональных функций, а также определение функций через систему функциональных уравнений. Теоретической базой языка являются работы [3]. Было выполнено несколько реализаций ранних версий языков функционального программирования, основанных на указанном подходе. Одна из этих версий, реализованная для персональных компьютеров, до сих пор используется, в том числе и для учебных целей [5].

В разработанном языке композиционного функционального программирования, сокращенно FPTL (Functional Parallel Typified Programming Language), с одной стороны воплощены многие механизмы, которые стали неотъемлемыми в современных языках программирования: модульность, инкапсуляция, типизация и др. С другой стороны, в языке FPTL введен целый ряд механизмов, которые либо обязаны исследованиям по теории программирования, в частности, схемотологии программ, либо продиктованы чисто прагматическими соображениями. К наиболее важным из них следует отнести:

– схемный характер задания функций [4], упрощающий анализ функциональных программ и динамическое выявление параллелизма при их выполнении;

– возможность использования (импортирования) в программе функций, описанных на традиционных языках программирования (C, C++, Pascal и др.), что делает реализованную систему функционального программирования полиязычной [6];

– статический контроль типовой правильности, который обеспечен сознательным ограничением возможностей отношений между типами с тем, чтобы типизация не была превращена в новую проблему при программировании и контроле правильности задания типов;

– теоретически обоснованная асинхронная модель параллельного вычисления значений функций [3], снимающая вопрос о ее корректности и существенно упрощающая ее реализацию.

2. Инструментальная среда программирования

Арсенал средств, представляемых современными ОС и средами программирования, помогающих программисту разрабатывать программу, весьма широк. Однако многие принципиального характера механизмы, существенные с позиций разработки качественной программы и обоснования ее правильности, практически никак не представлены в реализации всех известных императивных, функциональных, объектно-ориентированных и логических языков. Это касается анализа структуры программы, оценивания сложности, ее эквивалентных целенаправленных преобразований, верификации и др. [7].

В инструментальной среде программирования на FPTL сделана, пожалуй, первая попытка создания комплексной системы программных средств, выполняющих названные функции [8]. Отметим наиболее важные моменты в реализации этих функций.

Разработанная модель и алгоритмы анализа структурной сложности функциональных программ [9] позволяют определять характеристики программы, касающиеся их «устройства» с позиций использования циклических и рекурсивных определений. Именно по этим характеристикам наиболее объективно можно судить о логической сложности организации программы, ими, в основном, определяются и сложность самого анализа программы и проверки ее правильности, эти характеристики также используются при построении алгоритмов планирования выполнения функциональных программ на вычислительных системах.

Система верификации функциональных программ [8] интересна тем, что в ней, в отличие от традиционных подходов к доказательству правильности программы, (выполняемых путем введения специфицирующих входо-выходных логических утверждений, обычно соотносимых с выделяемыми участками программы), программа сначала транслируется в логическую форму (формулу логики первого порядка или хорновскую формулу этой логики). Имея такое представление своей

программы, программист может использовать весь арсенал стандартных логических средств для доказательства правильности программы, ее эквивалентности другой программе и т.п.

Подсистемы типового контроля, управления процессом разработки функциональных программ, использования функций, представляемых на других языках программирования, и др., разработанных для инструментальной среды, более детально описаны в [1, 8].

3. Управление параллельным выполнением функциональных программ на вычислительных системах

Организационная структура и состав программных средств системы управления параллельным выполнением функциональных программ подробно описаны в статье [1]. Реализация системы управления осуществляется для кластерных ВС, которые структурно представляют собой множество узлов (локальных сетей) из однородных компьютеров, объединяемых посредством использования серверов, выполняющих роль локальных управлений узлами ВС. При объединении серверных узлов соответственно их управление образует иерархическую структуру, отражающую декомпозицию множества узлов на подмножества и правила делегирования функций межузлового управления вышестоящим серверам.

Основные функции управления серверов каждого уровня:

- сбор данных о загруженности подчиненных им компьютеров (для серверов самого низшего уровня) или узлов (для серверов высших уровней), вычисление по ним усредненной загруженности и ее девиации, а также прогнозирование загруженности;
- реакция на отказы и восстановления системы и обеспечение высокой отказоустойчивости;
- администрирование подчиненных компонентов;
- динамическое управление конфигурацией (масштабированием, изменением структуры коммуникаций и др.).

Каждый компьютер ВС сам выполняет функции о контроле своей загруженности путем измерения (стандартными, как правило, средствами ОС) таких параметров, как объем свободной памяти, интенсивность обмена страницами между дисковой и оперативной памятью, интенсивность межкомпьютерного обмена, количество выполняемых процессов и др.

Общий поход к реализации данного управления параллельными процессами на ВС такой организации подробно рассмотрен в [7]. Его основная задача – обеспечение равномерной загрузки компьютеров ВС и минимизация времени выполнения функциональной программы. Для решения этой задачи в реализации управления используется ряд эвристических правил, позволяющих компьютеры (и узлы) ВС дифференци-

ровать по степени их загруженности, количественно отраженной в шкале: недогружен, нормально загружен и перегружен. Компьютер ВС перегружен по памяти, если у него нет свободной оперативной памяти и высока интенсивность обменов между оперативной памятью и дисками. Компьютер перегружен по интерфейсу, если у него большая интенсивность обменов с другими компьютерами (трафик по сети обмена сравним с ее пропускной способностью).

Стратегия равномерной загрузки предполагает перемещение части процессов перегруженных компьютеров на недогруженные. Естественно, что центральной проблемой при регулировании загрузки является выбор процессов-кандидатов для перемещения на другой компьютер. Эту роль выполняет планировщик, размещенный на каждом компьютере, во взаимодействии с интерпретатором [1, 8], также устанавливаемом на каждом компьютере.

В качестве кандидатов на перемещение рассматриваются только процессы, связанные с вычислением значений рекурсивно определяемых функций, т.е. функциональных переменных, а также импортируемых функций из других языков, если программист отмечает их как достаточно сложные по вычислению [9].

Эвристические соображения, стоящие за этим, обоснованы в [7]. Запрет на перемещение между компьютерами базисных и нерекурсивно определяемых функциональных переменных (как правило, несложных с вычислительной точки зрения) позволяет исключить деградацию работы ВС из-за частых обменов между ее компьютерами.

Более детально стратегии выбора процесса-кандидата на перемещение рассматриваются в докладе [9].

Заключение

В настоящее время завершена реализация языка, основной частью которого является интерпретатор, реализующий процесс параллельного выполнения функциональной программы. Выполненные эксперименты по сравнению его эффективности (времени, затрачиваемому на выполнение программы) вполне удовлетворительны [8]. Несмотря на то, что интерпретатор значительно усложнен из-за того, что в нем реализованы сложные механизмы динамического выявления в программе параллельных процессов, по эффективности он не уступает, а часто и превосходит «последовательные» интерпретаторы известных языков Lisp, Haskell, ML и др. [8].

В стадии завершения (идут отладочные работы) находятся программные средства для управления параллельным выполнением функциональных программ на кластерах. Получены первые позитивные результаты этой работы на практике [9]. Завершение проекта должно показать, что сложные рекурсивные функциональные программы можно

успешно выполнять параллельно, без участия программиста в какой-либо предварительной работе по так настоятельно навязываемому ему распараллеливанию программы.

Литература

1. *Бажанов С.Е., Кутепов В.П., Шестаков Д.А.* Язык функционального параллельного программирования и его реализация на кластерных системах // Теория и системы управления (в печати).
2. *Кутепов В.П., Фальк В.Н.* Функциональные системы: теоретический и практический аспекты // Кибернетика, 1979. №1.
3. *Кутепов В.П., Фальк В.Н.* Модели асинхронных вычислений значений функций в языке функциональных схем // Программирование, 1978. №3.
4. *Кутепов В.П.* Исчисление функциональных схем и параллельные алгоритмы // Программирование, 1976, №6.
5. *Грызунов В.Б.* Разработка и реализация системы функционального программирования // Автореф. дис. на соиск. учен. степени канд. техн. наук. М.: МЭИ, 1990.
6. *Борздова Т.В.* и др. Полиязычная система параллельного программирования, основанная на одном семействе функциональных языков // Программирование, 1984. №2.
7. *Кутепов В.П.* Об интеллектуальных компьютерах и больших компьютерных системах нового поколения // Изв. РАН. Теория и системы управления, 1996. №5.
8. *Бажанов С.Е., Кутепов В.П.* Функциональное параллельное программирование: язык, его реализация и инструментальная среда разработки программ // Доклад на четвертом Международном научно-практическом семинаре «Высокопроизводительные параллельные вычисления на кластерных системах».
9. *Кутепов В.П., Шестаков Д.А.* Анализ структурной сложности функциональных программ и его применение для планирования их выполнения на вычислительных системах // Доклад на четвертом Международном научно-практическом семинаре «Высокопроизводительные параллельные вычисления на кластерных системах».

МОДЕЛИРОВАНИЕ СВЕРХИЗЛУЧЕНИЯ СИСТЕМЫ ЗАРЯЖЕННЫХ ОСЦИЛЛЯТОРОВ

В.В. Березовский

Поморский Государственный Университет им. М.В. Ломоносова, г. Архангельск

Рассмотрен классический аналог сверхизлучения Дикке [2] в классической системе нелинейных заряженных осцилляторов, взаимодействующих через собственное поле излучения. В начальный момент ос-

цилляторы несфазированы – их фазы случайны. Зависимость частоты колебаний нелинейных осцилляторов от амплитуды приводит к автофазировке осцилляторов и росту интенсивности излучения.

Сверхизлучение (СИ) – это кооперативное излучение, возникающее вследствие самопроизвольного зарождения и усиления корреляций первоначально независимых атомов. Основу СИ составляет эффект фазировки атомов, т.е. возникновение корреляций между первоначально независимыми атомами. Природу фазировки атомов позволяет лучше понять классическая модель сверхизлучения (КМС), в которой атомы заменены классическими ангармоническими осцилляторами Лоренца, а поле описывается классическими уравнениями Максвелла. Существуют два механизма фазировки атомов: нелинейный и линейный диполь-дипольный. Эти механизмы конкурируют друг с другом: согласно первому из них каждый атом создает себе облако из окружающих атомов, колеблющихся с той же фазой, а согласно второму – в противофазе. Таким образом, в результате диполь-дипольного взаимодействия возникает экранировка СИ. Диполь-дипольное взаимодействие является дальнодействующим, вследствие чего характер СИ существенным образом зависит от формы тела.

В этой работе в рамках нелинейной КМС учтено дипольное взаимодействие атомов в системе малого размера ($l<<L<<\lambda$, где $l=n^{1/3}$ – характерное расстояние между атомами (n – концентрация атомов), L – размер тела, λ – длина излучаемых волн) произвольной формы. Строгий математический расчет удается провести до конца только для тел эллипсоидальной формы. По этой причине есть необходимость в численном моделировании КМС для учета диполь-дипольного взаимодействия. Рассмотрим заряженные осцилляторы ($a=1,2,...,N$) с зарядами величины e, массы m. Осцилляторы находятся в точках с координатами r_a. С учетом нелинейности и диполь-дипольного взаимодействия и после ряда упрощений уравнение движения осцилляторов имеет вид [1]:

$$\ddot{F}_a + i\delta\left(|F_a|^2 - 1\right)F_a = i\beta\sum_{b\neq a}\frac{3n_{ab}(n_{ab}F_b) - F_b}{(r_{ab})^3} - \frac{1}{2}\beta_0\sum_b F_b \qquad (1)$$

где F_a – комплексная безразмерная амплитуда, $\boldsymbol{n}_{ab}=\boldsymbol{r}_{ab}/r_{ab}$, $\mathbf{r}_{ab}=\mathbf{r}_a-\mathbf{r}_b$, $\beta=e^2/(2m\omega_0)$, $\omega=\omega_0+\delta$, $\delta=3\gamma\omega_0 b^2/2$, $\beta_0=2e^2\omega_0^2/3mc^3$, b – характерная амплитуда колебаний, ω_0 – частота колебаний осцилляторов, γ – их ангармонизм. Первое слагаемое в левой части – результат диполь-дипольного взаимодействия, второе моделирует радиационное трение. Интенсивность излучения, усредненная по быстрым осцилляциям диполей, равна

$$\boldsymbol{I}=(1/3c^3)\, e^2\omega^4|F_a||F_b|\cos(\varphi_a-\varphi_b)$$

Так как F - комплексная величина, рассматривается двухмерная задача, ориентированная на исследование развития системы нелиней-

ных классических осцилляторов с учетом их взаимодействия. Рассмотрим систему (1). В общем случае мы можем представить ее в следующем виде:

$$Lf = Af\ ,$$

где линейный дискретный оператор L соответствует разностному выражению:

$$Lf^k = \frac{f^k - f^{k-1}}{\Delta t}\ ,$$

а нелинейный оператор A

$$Af = i\beta \sum_{b \neq a} \frac{3n_{ab}(n_{ab} f_b) - f_b}{r_{ab}^3} - \frac{1}{2}\beta_0 \sum_b f_b - i\delta(|f_a|^2 - 1) f_a\ ,\ a = (1,2,...N)\ .$$

Для параллельной реализации ядра алгоритма оказывается достаточным «расщепить» индексные массивы, используемые для генерации дискретных операторов, и применить процедуру межпроцессорных обменов при реализации простейших блоков умножения операторов на элементы вектора. Пусть требуется вычислить произведение $A \times B$, где $A = \|a_{ij}\|_{n \times n}, B = \|b_i\|_n$. Умножение матрицы на вектор можно выполнить на n процессорах, организованных в E-структуру. Каждый i-тый процессор (i=0,..N-1) в начале выполнения программы содержит i-тый элемент вектора B и i-тый столбец матрицы операторов A. Умножение производится за n тактов, каждый такт включает в себя накопление и сдвиг. Накопление включает в себя применение оператора из ячейки матрицы A, отмеченной шаблоном, к имеющемуся на данный такт у процессора элементу вектора B и прибавление результата к значению, хранящемуся в накопителе. Сдвиг состоит из цикличного сдвига шаблона вверх и передаче текущего элемента вектора B соседнему процессору. В случае произвольного количества процессоров меньшего N, столбцы матрицы A и элементы вектора B распределяются по процессорам, схема же вычислений остается той же.

Схематично вычисления происходят, как показано на рис. 1: где

$$a_{ii} b_i = -i\delta(|b_i|^2 - 1) b_i \times \Delta t\ ,\ a_{ij} b_i = (-\frac{1}{2}\beta_0 b_i + i\beta \frac{3n_{ji}(n_{ji} b_i) - b_i}{r_{ji}^3} + b_i) \times \Delta t\ .$$

Для упрощения, предполагается, что F направлена только вдоль оси ***x,*** то скалярное произведение $\boldsymbol{n}_{ab}(\boldsymbol{n}_{ab}\boldsymbol{F}_b) = F_b cos^2(\angle \mathbf{n}_{ab}\boldsymbol{x})$ и $cos^2(\angle \boldsymbol{n}_{ab}\boldsymbol{x})$ не изменяется с течением времени. В начале работы высчитываются $cos2(\angle nabx)$ и r_{ab}^3 для каждой ячейки матрицы A, для чего каждому процессору передаются координаты всех точек.

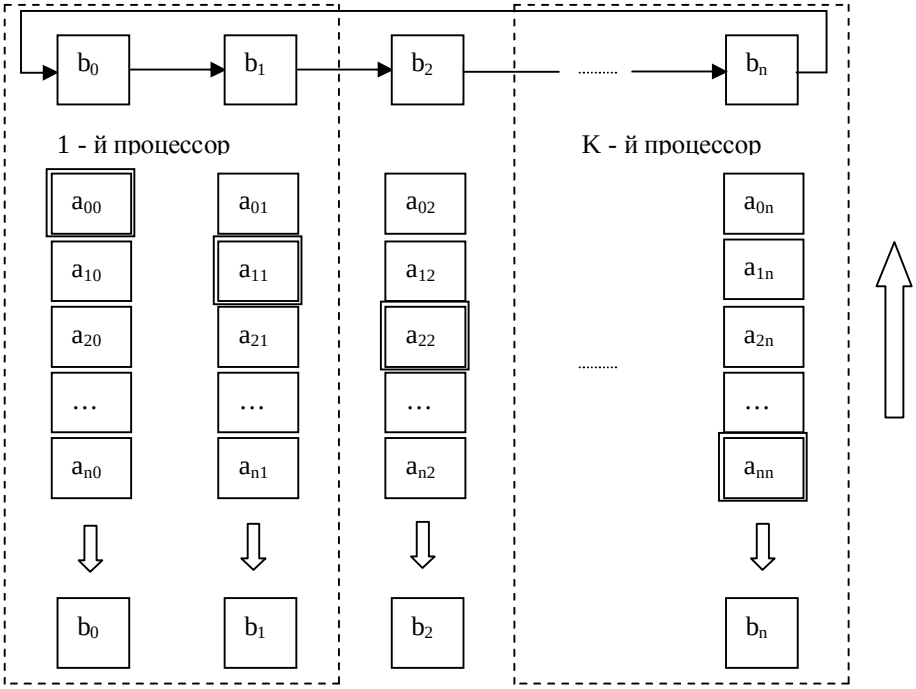


Рис. 1. Схема вычислений

После чего начинается основной ход программы. Шаг вычислений состоит из применения текущего оператора к текущему элементу вектора. Например, для 0-го процессора вначале применяется оператор a00 к элементу b0, результат сохраняется в накопителе, b0 передается процессору 1 и шаблон циклично смещается вверх к ячейке an0. На следующем такте к полученному от соседа слева элементу bn применяется оператор an0, результат прибавляется к значению в накопителе и сохраняется, затем сдвиг шаблона вверх и передача bn соседу справа. И так N тактов. Когда цикл завершится, значение из накопителя умножается на Δt и складывается с полученным слева b0. В конце все bi пересылаются 0-му процессору, который сохраняет их на диске.

Структура обменов является однородной. В дополнение к этому корневой процесс выполняет незначительный объем вычислений, обусловленный необходимостью сборки значений шагов по времени от остальных процессов. Основные этапы вычислений на одном временном шаге представлены на рис. 2. В соответствие с ним мы имеем следующую схему на каждом шаге:

Ÿ – вычисление значений шагов по времени для каждого процесса;

Ÿ – сборка значений шагов по времени на корневом процессе;

Ÿ – сохранение результатов.

Настоящая работа содержит результаты, полученные при численном решении системы (1) для системы осцилляторов, находящихся в теле произвольной формы со случайным образом заданной начальной фазой φ_a.

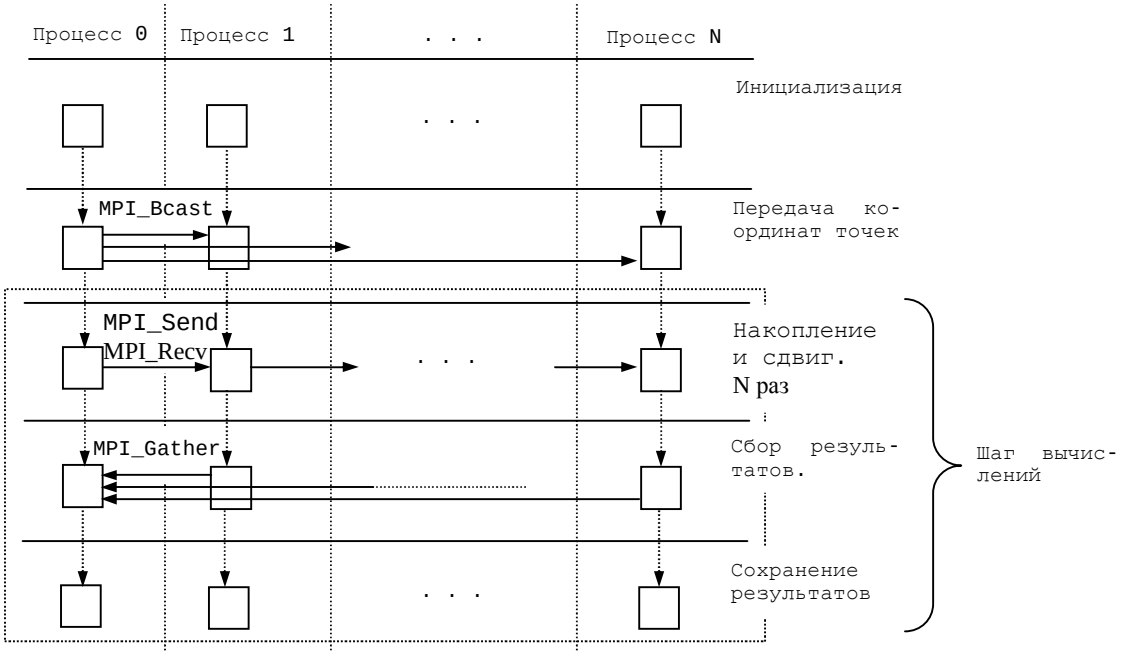


Рис. 2. Схема вычислений

Возбужденные в начальный момент времени полностью несфазированные ангармонические осцилляторы, взаимодействующие через собственное поле излучения, находятся в неустойчивом состоянии. Поляризация системы инициирует развитие этой неустойчивости и приводит к росту интенсивности излучения [3]. В рамках этой работы удалось сделать расчет для числа осцилляторов до 10^4. Благодаря применению технологии параллельных вычислений, существует принципиальная возможность произвести численный расчет для числа осцилляторов порядка 10^6, что близко к количественным оценкам числа участвующих в сверхизлучении осцилляторов в последних экспериментах [4].

Литература

1. *Меньшиков Л.И.* УФН 169 113 (1999).

2. *Dicke R.H.* Phys. Rev. 93 99 (1954).

3. *Ильинский Ю. А., Маслова Н. С.* ЖЭТФ 94 171 (1988).

4. *Зайцев С.В., Гордеев Н.Ю., Graham L.A., Копчатов В.И., Карачинский Л.Я., Новиков И.И., Huffaker D.L., Копьев П.С.* Физика и техника полупроводников // 1999. 33. 1456.

СИСТЕМА АВТОМАТИЗИРОВАННОГО ТЕСТИРОВАНИЯ ПАРАЛЛЕЛЬНЫХ АЛГОРИТМОВ ВЫВОДА И ОБУЧЕНИЯ В ВЕРОЯТНОСТНЫХ СЕТЯХ

Р.В. Виноградов

Нижегородский государственный университет им. Лобачевского, г. Н. Новгород

Введение

В данной работе рассматривается возможность автоматизации процесса тестирования параллельных алгоритмов на классе задач вывода и обучения в вероятностных графических моделях. Необходимость создания специализированных средств, упрощающих тестирование, вызвана высокой трудоемкостью этого процесса.

Для демонстрации результатов работы дадим ряд основных понятий рассматриваемой темы [1, 3, 8]. *Байесовская сеть* (*Bayesian belief network*) – это направленный ациклический граф (*directed acyclic graph – DAG*), обладающий следующими свойствами:

– каждая вершина представляет собой событие, описываемое случайной величиной, которая может иметь несколько состояний;

– все вершины, связанные с *родительскими* вершинами, определяются таблицами условных вероятностей или функцией условных вероятностей;

– для вершин, не имеющих *родителей*, вероятности ее состояний являются безусловными (*маргинальными*).

В байесовских сетях могут сочетаться эмпирические частоты появления различных значений переменных, субъективные оценки «ожиданий» и теоретические представления о математических вероятностях тех или иных следствий из априорной информации. *Клика* в графе – это полный подграф, не содержащийся ни в каком другом полном подграфе исходного графа. Под *семьей* вершины в орграфе понимается список, включающий родительские вершины и саму вершину.

Существующая реализация алгоритмов в вероятностных сетях

Существует большое количество программных продуктов, реализующих идею использования графических вероятностных моделей. Однако многие из них обладают рядом недостатков [6]:

Недоступность исходного кода. Подобные продукты исключают собственную интеграцию в другие приложения;

Отсутствие реализации неориентированных моделей (*2D MRF – Markov random fields*), динамических моделей (*DBN – dynamic Bayes net*), непрерывных случайных величин и т.д.

Библиотека классов Open Source Probabilistic Networks Library (OpenPNL) [5, 6] лишена этих недостатков. В настоящее время PNL вместе с разрабатываемой компонентой Statistical Learning (OpenSL) являются частью масштабного проекта Open Source Machine Learning (Open Source ML) [5].

В рамках библиотеки PNL были разработаны параллельные аналоги алгоритмов вывода и обучения в вероятностных сетях, тестирование которых рассматривается в данной работе.

Схема взаимодействия компонентов системы

На рис. 1 представлена схема, отражающая логику работы и последовательность действий системы, рассматриваемой в данной статье.

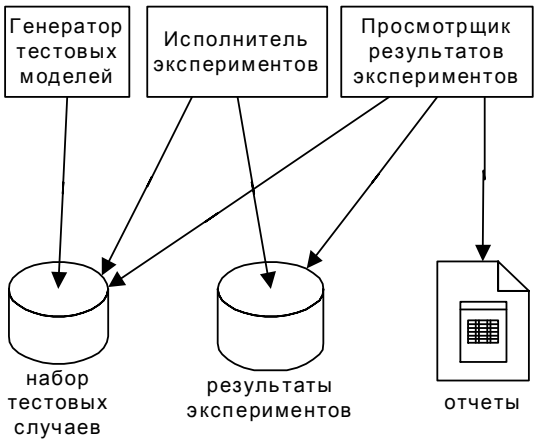


Рис. 1. Схема взаимодействия компонентов системы

Средством редактирования «набора тестовых случаев» служит «Генератор тестовых моделей», рассмотренный ниже. «Исполнитель экспериментов» упрощает подготовку и запуск параллельных методов в ходе проведения серийных экспериментов, содержит тестовый модуль, способный таймировать критические по времени этапы работы алгоритмов, и проверять корректность их результатов.

В ходе тестирования результаты отдельных экспериментов сохраняются для последующей обработки, что обеспечивает возможность сравнения и анализа результатов (как эффективности разрабатываемых параллельных методов, так и совпадения результатов) на разных этапах тестирования и отладки. «Исполнитель экспериментов» отказоустойчив к возникающим исключениям, что способствует длительному проведению большого числа экспериментов без участия тестировщика. «Просмотрщик результатов экспериментов» позволяет сканировать «базу результатов экспериментов», формировать статистические данные о кор-

ректности выполненных тестов и документировать результаты экспериментов в табличные формы, содержащие числовые показатели эффективности работы параллельных методов.

Генерация случайной байесовской сети

Одна из задач, возникающих на этапе тестирования параллельных алгоритмов, реализованных на базе библиотеки PNL, связана с подбором тестовых случаев (вероятностных моделей), объективно отражающих реальность. Подход очевиден – случайная генерация. Этот способ формирования моделей подразумевает наличие некоторого набора параметров, ограничивающих топологию модели и размеры таблиц вероятностей, заданных на вершинах сети. В данной работе рассматривалось случайное формирование как ориентированных, так и неориентированных графических вероятностных моделей. Более подробно остановимся на генерации байесовских сетей.

Произвольная байесовская сеть (рис. 2) необходима для изучения поведения показателя эффективности и масштабируемости разрабатываемых параллельных алгоритмов.

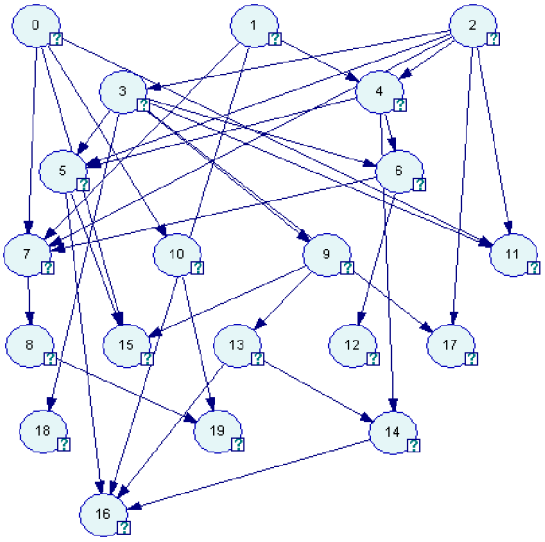


Рис. 2. Пример байесовской сети, имеющей случайную топологию

При формировании произвольного графа, лежащего в основе байесовской сети, основным параметром, явно ограничивающим число связей у каждой вершины и косвенно ограничивающим размер таблиц распределения вероятности, служит максимальный размер *семьи* вершин в сети.

При создании графа учитывается еще один параметр, определяющий число вершин, не зависящих от других вершин (размер семьи таких вершин равен 1, а распределение вероятности является маргинальным). С целью ограничить размер таблиц условных вероятностей рассматривается параметр, определяющий максимально возможное число состояний дискретных случайных величин, соответствующих вершинам модели.

Важно отметить, что реализация определенных алгоритмов в рамках библиотеки PNL подразумевает ее дальнейшее применение для решения практических задач. Имеющиеся ограничения на случайность структуры графа и размеры вещественных матриц, присутствуют и в реальных моделях, отражающих проблематику конкретной практической задачи.

Топология графа, лежащая в основе сети toyQMR (рис. 3), совпадает с топологией двудольного графа. В сети такого рода каждая вершина нижнего уровня образует некоторое количество связей с вершинами верхнего уровня, однако вершины одного уровня не могут находиться в зависимости между собой.

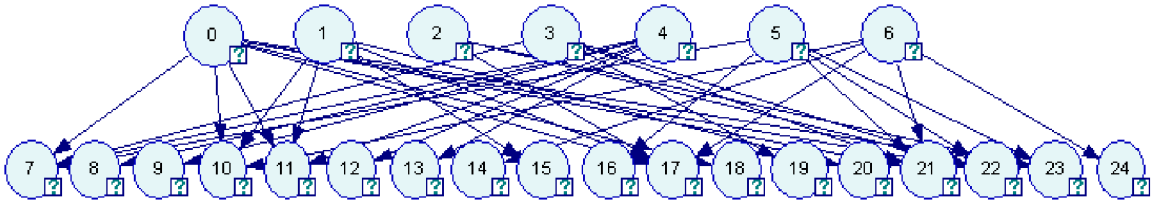


Рис. 3. Пример байесовской сети с топологией toyQMR

Ориентированные модели с подобной топологией часто встречаются в медицинской диагностике. В этом случае вершины верхнего слоя отражают сущности медицинских тестов и симптомов, а вершины нижнего слоя, подчиненные вершинам верхнего слоя, соответствуют диагнозам пациентов. Общее описание подобного рода байесовских сетей, а также уникальные свойства и поведение таких моделей применительно к алгоритмам вывода можно найти в статье [2].

Байесовская сеть с топологией пирамиды (рис. 4) имеет структуру, графическая интерпретация которой выглядит следующим образом:

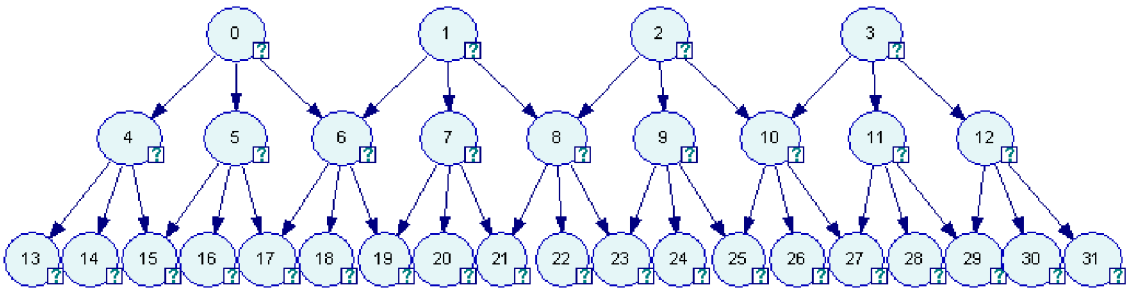


Рис. 4. Пример байесовской сети с топологией PYRAMID

1) все вершины графа можно собрать в некоторое количество упорядоченных сверху вниз слоев так, чтобы выполнялись пункты 2 и 3;
2) можно «подвинуть» слои так, чтобы полученный граф представлял собой пирамиду, симметричную относительно некоторой вертикальной оси;
3) каждая вершина *i*-го уровня может находиться в отношении «ребенок-родитель» только с соседними вершинами (*i-1*)-го уровня.

Байесовские сети, имеющие такую топологию, частично используются в диагностических целях (в том числе в медицине). Важно также отметить, что в силу симметричности модели и небольшого размера семьи вершин модели PYRAMID представляют определенный интерес:

1) подобная топология важна для изучения свойств масштабируемости параллельных алгоритмов вывода, основанных на распределении вершин сети по процессорам. Примером может служить итерационный метод *Loopy Belief Propagation* [2];
2) алгоритм вывода *Junction Tree Inference* [1], реализующий идею выделения в сети дерева клик (при удачной триангуляции графа исходной сети размер клик будет ограничен) также представляет интерес для исследования вопросов трудоемкости и скорости работы метода на задачах небольшого размера. Алгоритм может работать на сетях небольшого размера, поскольку является точным и весьма трудоемким.

Информацию о подобной топологии байесовских сетей можно найти в статье [2].

При формировании сети с топологией полного графа, случайный механизм используется только для задания маргинальных распределений вероятностей на независимых вершинах и условных распределений, при этом топология сети определяется однозначно.

Структура полного графа попадает в одну из вырожденных ситуаций, наличие которых необходимо в ходе тестирования на предмет отказоустойчивости и универсальности разрабатываемых методов. Например, алгоритм вывода Junction Tree Inference [1] на сети, имеющей топологию полного графа, создаст вырожденное дерево клик с одним корнем, в результате чего, исчезнет необходимость в распространении вероятностей между кликами.

В данной работе для создания набора тестовых случаев (под тестовым случаем понимается вероятностная модель с определенной топологией) использовалось два формата представления моделей:

- DSL (Decision Systems Laboratory) – формат хранения моделей, предоставленный лабораторией систем принятия решений [7];
- формат представления объектов библиотеки PNL, основанный на XML 1.0.

Автоматизированное тестирование

Автоматизация процесса тестирования заключается во внедрении средств (рис. 1), позволяющих упростить подготовку исходных наборов данных в отдельности для каждого из алгоритмов, ускорить создание необходимых конфигурационных файлов, содержащих параметры запуска параллельных MPI- или OpenMP-реализаций, обеспечить хранение результатов как атомарных, так и серийных экспериментов, уско-

рить сбор и обработку результатов (автоматическое документирование и проверка правильности результатов). Документирование данных посредством COM-сервера Word позволило сократить время обработки результатов для последующего анализа.

Работа выполнялась в рамках проекта «Масштабируемые алгоритмы вывода и обучения графических моделей в рамках библиотеки PNL», действующего в Нижегородском университете, по заказу компании Intel.

Литература

1. *Cowell R., Dawid A., Lauritzen L., Spiegelhalter D.* Probabilistic networks and expert systems // Springer-Verlag, NewYork. 1999. - 321 с.
2. *Kevin P. Murphy, Yair Weiss, Michael I. Jordan* Loopy Belief Propagation for Approximate Inference // An Empirical Study. University of California, Berkeley. Seminar 37-551: Belief Propagation Algorithms.
3. *Хабаров С.П.* Экспертные системы.
4. *Терехов С.А.* Введение в байесовы сети // Лекция для школы-семинара «Современные проблемы нейроинформатики», Москва, МИФИ, 29-31 января 2003 года.
5. Библиотека Open Source Probabilistic Networks Library.
6. *Victor Eruhimov, Kevin P. Murphy, Gary Bradski* Intel's open-source probabilistic networks library.
7. Проект GeNIe от Decision Systems Laboratory - University of Pittsburgh.
8. Dictionary of Algorithms and Data Structures.

ЯЗЫК МОДЕЛИРОВАНИЯ ПРОСТРАНСТВЕННО РАСПРЕДЕЛЕННЫХ ПАРАЛЛЕЛЬНЫХ ПРОЦЕССОВ

С.В. Востокин

Самарский государственный аэрокосмический университет, г. Самара

Введение

В последнее время в компьютерной индустрии наметилась тенденция к интеграции вычислительных ресурсов для решения некоторой общей задачи. Она обусловлена развитием и широким распространением технических средств, таких как высокопроизводительные компьютеры и коммуникационные среды высокой пропускной способности. Технологический аспект разработки программного обеспечения для такого рода систем связан с решением комплекса проблем, включающих проблемы надежности, масштабируемости алгоритмов и программ, мобильности программ, разработки технологий повторного использования. Широкое распространение средств связи также дает возможность

реализовать программное управление внешними по отношению к вычислительной системе пространственно распределенными процессами, которые протекают параллельно во времени. Разработка качественного программного обеспечения для распределенных систем не возможна без применения системного подхода, опирающегося на формальную модель пространственно распределенных параллельных процессов. В докладе представляется новый графический язык моделирования, предназначенный для использования в качестве основы программных сред и средств автоматизации параллельного программирования.

В первой части доклада излагается спецификация языка моделирования пространственно распределенных дискретных систем, которая включает: определение семантики языка моделирования; формальное определение синтаксиса языка моделирования с использованием формы Бекуса-Наура; определение графической нотации описания моделей систем.

Во второй части описывается метод хранения сети объектов структурных элементов модели, метод преобразования текстового представления в сеть объектов, а также набор команд, однократное исполнение последовательности которых позволяет восстановить сеть из структурных элементов модели в памяти.

В третьей части рассматривается метод построения модели распределенного вычислительного процесса по известному последовательному алгоритму. Рассматривается тестовая задача организации вычислительного процесса для решения системы линейных уравнений методом Гаусса-Зейделя. Метод основан на параллелизме по данным. Суть его заключается в разделении области обрабатываемых данных на фрагменты, в которых организуется локальная обработка. Используемый метод распараллеливания сводит решение задачи к организации корректного взаимодействия нескольких последовательных процессов, работающих с локальными данными. Данный вычислительный процесс может быть наглядно описан на введенном языке моделирования. Рассматривается набор диаграмм на языке моделирования, описывающий масштабируемый вычислительный процесс решения тестовой задачи.

В заключении рассматриваются возможные сферы применения языка моделирования. Дополнительную информацию по проекту разработки языка моделирования и инструментальных средств автоматизации параллельного программирования на его основе можно найти по адресу http://graphplus.ssau.ru.

РАСПРЕДЕЛЕНИЕ РЕСУРСОВ МНОГОПРОЦЕССОРНЫХ СИСТЕМ ПРИ ВЫЧИСЛЕНИИ СОГЛАСОВАННЫХ ОЦЕНОК ПО МАЛОМУ ЧИСЛУ НАБЛЮДЕНИЙ

А.В. Гаврилов, В.А. Фурсов

Самарский государственный аэрокосмический университет, г. Самара,
Институт систем обработки изображений, г. Самара

Введение

Часто задачу идентификации параметров систем приходится решать по малому числу наблюдений [1], например, при определении изменяющихся параметров модели управляемого объекта [2].

Основное условие предельных теорем теории вероятностей (существование большого числа случайных явлений) при этом не выполняется, поэтому основанная на них теория статистического оценивания и рассматриваемые в рамках этой теории методы построения оценок являются в таком случае недостаточно обоснованными.

В работе [3] было высказано предположение, что оценивание целесообразно проводить, используя только наиболее свободные от шума наблюдения. Эта идея получила свое развитие в виде принципа согласованных оценок [1], представляющего собой методологию получения нестатистических оценок параметров $\hat{c}$ для уравнения вида

$$y = Xc + \xi, \tag{1}$$

где матрица X размера $N \times M$ и вектор y размера $N \times 1$ доступны для непосредственного наблюдения, а вектор ошибок ξ размера $N \times 1$ неизвестен. К виду (1) сводится большое количество задач.

Одной из проблем реализации этого подхода является существенное увеличение объема вычислений, необходимых для получения оценок, вследствие переборного характера задачи поиска таких наблюдений. Для решения задач оценивания больших размерностей неизбежным становится применение параллельных вычислений. В данной работе рассматриваются вопросы построения параллельного алгоритма вычисления согласованных оценок по малому числу измерений, а также вопросы распределения ресурсов гетерогенных кластеров при выполнении этого алгоритма.

1. Последовательный алгоритм

Исходными данными для решения задачи служат матрица X, вектор y, а также размерность подсистем верхнего уровня P [4].

На первом этапе строятся всевозможные подсистемы верхнего уровня H_i^P ($i=\overline{1,C_N^P}$), в каждую из которых входит P исходных наблюдений системы (1).

На втором этапе для каждой из подсистем верхнего уровня строятся всевозможные подсистемы нижнего уровня $L_{i,j}^P$ ($j=\overline{1,C_P^M}$) размерности M. Для каждой из них находится оценка $\hat{c}_{i,j}^P$. Способ нахождения оценки может различаться в зависимости от требований по точности и ограничений по вычислительной сложности.

На третьем этапе вычисляются значения функции взаимной близости [5] $\Delta\left(H_i^P\right)$ для каждой из подсистем верхнего уровня на основании оценок, полученных для подсистем нижнего уровня, и выбирается подсистема верхнего уровня с минимальным значением этой функции.

В результате происходит полный перебор всех возможных сочетаний наблюдений исходной системы (1). При этом большое количество подсистем нижнего уровня, входящих в различные подсистемы верхнего уровня, будут совпадать. Очевидно, что всего подсистем нижнего уровня будет C_N^M. Поэтому работу программы можно построить в виде следующих шагов:

1) получение всевозможных подсистем нижнего уровня и построение для них оценок;
2) получение всевозможных подсистем верхнего уровня и вычисление значений функции взаимной близости для них;
3) нахождение минимума среди полученных значений и вычисление искомой оценки параметров.

2. Логика построения параллельного алгоритма

Пусть известны числа N, M, P, а также число процессов K.

Рассмотрим сначала изменения, возникающие в первом этапе. Логично распределить между процессами построение подсистем нижнего уровня и нахождение оценок для них. Дальнейшее разделение по функциональности (распараллеливание решения конкретных систем) нецелесообразно в виду резкого возрастания количества передаваемых по сети данных и ощутимого преобладания числа подсистем нижнего уровня над возможным числом процессов.

Для получения комбинаций индексов в (1), характеризующих подсистемы верхнего и нижнего уровней, предлагается использовать рекуррентный алгоритм, позволяющий получить следующую комбинацию на основании предыдущей. В основу его работы положена возможность биекции множества возможных комбинаций индексов и множества чисел, имеющих заданные двоичную разрядность и норму Хэмминга. В

этом случае отношение порядка на множестве комбинаций, а также начальные и конечные условия работы алгоритма тривиальны. Таким образом, для начала работы конкретного процесса необходимо получение им инициализирующего и конечного значений комбинаций. Кроме того, необходимо разослать всем процессам исходные данные, затратив на это некоторое время $T_0^{||}$.

Пусть также известен и набор чисел q_k ($k = \overline{1,K}$) количества задач подсистем нижнего уровня, приходящихся на процесс k при выбранной загрузке кластера. При этом должно выполняться условие

$$\sum_{k=1}^{K} q_k = C_N^M . \qquad (2)$$

В этом случае корневой процесс должен сначала решить задачу получения стартовых значений комбинаций всех процессов (конечные значения получать и пересылать необязательно – достаточно пересылать числа q_k). Решение этой задачи представляется возможным построить на последовательном применении следующей формулы:

$$C_f^g = C_{f-1}^{g-1} + C_{f-2}^{g-1} + \mathbf{h} + C_g^{g-1} + C_{g-1}^{g-1} . \qquad (3)$$

Нетрудно убедиться, что для поиска каждой следующей стартовой комбинации необходимо вычисление не более, чем $M \cdot (N - M)$ биномиальных коэффициентов вычислительной сложности не более, чем $C_N^{[N/2]}$, где [·] – оператор определения целой части.

Таким образом, для запуска работы всех процессов потребуется время

$$T_{предв}^{||} \le (K-1) \cdot M \cdot (N-M) \cdot \tau_{ap}\left(C_N^{[N/2]}\right) + (K-1) \cdot \tau_к(комбинация, q_k) , \qquad (4)$$

где τ_{ap} и $\tau_к$ – функции времени вычисления и пересылки соответственно. Время выполнения первого этапа при этом будет

$$T_I^{||} = \max_{k=1,K} \Omega(k, q_k, N, M) , \qquad (5)$$

где $\Omega(k, n, N, M)$ – время, необходимое k-му процессу для поиска комбинаций и решения n подсистем нижнего уровня при заданных N и M. Можно считать, что время поиска следующей комбинации мало по сравнению с временем решения системы, тогда нелинейностью вычислительной сложности данной операции можно пренебречь:

$$\Omega(k, n, N, M) = n \cdot \omega(k, N, M) , \qquad (6)$$

где $\omega(k, N, M)$ – время, необходимое k-му процессу для поиска одной комбинации и решения подсистемы нижнего уровня при заданных N и M.

Распараллеливание второго этапа работы последовательного алгоритма связано с рядом трудностей. Основной из них является образование на первом этапе большого количества числовых данных, которые используются на втором этапе. Попытка обеспечения равномерной загрузки на втором этапе приводит к необходимости пересылки почти всех этих данных всем процессам, что может стать причиной резкого возрастания времени выполнения программы.

В то же время пересылка всех данных корневому процессу неизбежна, т.к. существуют подсистемы верхнего уровня, для которых при вычислении значения функции взаимной близости необходимы данные с множества других процессов. Попытка агломерации без множественного реплицирования данных от процессов на некоторых локальных центрах, вообще говоря, не решает этой проблемы.

Предлагается следующий порядок действий:

1) вычисление характеристик подсистем верхнего уровня на отдельных процессах, если процессы обладают необходимыми для этого данными;
2) пересылка всех данных, полученных на первом этапе, одному из процессов;
3) вычисление на нем недостающих характеристик подсистем верхнего уровня.

Оценить временные характеристики работы данной части алгоритма в общем виде не представляется возможным, но можно утверждать, что для времени выполнения второго этапа будет справедлива оценка:

$$T_{II}^{\|} \le T_{II} + \sum_{k=2}^{K} \tau_{\kappa}\left(q_k \times \hat{c}, q_k \times \textit{комбинация}\right). \qquad (7)$$

Очевидно, что распараллеливать имеет смысл только первые два этапа работы, как наиболее трудоемкие. При этом выигрыш в скорости вычислений в основном будет достигаться на первом этапе. Рассмотрим более подробно его характеристики.

3. Оценка ускорения

Рассмотрим гомогенный кластер из K компьютеров. Пусть τ_L – латентность используемой сети, а τ_b – добавочное время на пересылку одного байта. Будем считать, что время передачи в сети линейно зависит от объема передаваемой информации, и все целочисленные значения двубайтовые, а дробные – восьмибайтовые. Далее приведем оценки времени, необходимого для выполнения различного рода операций:

$$T_0^{\|} = (K-1)\cdot\left(\tau_L + N\cdot(M+1)\cdot 8\cdot\tau_b\right), \qquad (8)$$

$$\tau_{ap}\left(C_N^{[N/2]}\right) = [N/2]\cdot\left(\tau_{\times} + \tau_{\div}\right), \qquad (9)$$

$$\tau_к(\textit{комбинация}, q_k) = \tau_L + 2\cdot(M+1)\cdot\tau_b, \tag{10}$$

$$\omega(k,N,M) = \omega(N,M) \le (2+2M)\cdot\tau_{=} + (4+6M)\cdot\tau_{+} + \tau_P, \tag{11}$$

$$T^{||}_{расс} = \sum_{k=2}^{K}\tau_к(q_k \times \hat{c}, q_k \times \textit{комбинация}) \le C_N^M(\tau_L + 8M\cdot\tau_b + 2M\cdot\tau_b). \tag{12}$$

Здесь $\tau_{\times}$, $\tau_{\div}$, $\tau_{=}$, τ_{+} соответственно время выполнения умножения, деления, проверки условия и сложения, а τ_P – время, необходимое для решения системы выбранным методом. Соответственно,

$$T^{||}_0 + T^{||}_{предв} \le (K-1)\cdot\left(2\tau_L + (8N+2)(M+1)\tau_b + N(N-M)\left[\frac{N}{2}\right](\tau_{\times} + \tau_{\div})\right). \tag{13}$$

В случае гомогенного кластера

$$T^{||}_I = \max_{k=\overline{1,K}} \Omega(k, q_k, N, M) \le \left([C_N^M / K] + 1\right)\cdot\omega(N,M). \tag{14}$$

Тогда ускорение [6] с учетом рассылки полученных результатов составит:

$$S = \frac{T^1}{T^{||}_0 + T^{||}_{предв} + T^{||}_I + T^{||}_{расс}}, \tag{15}$$

$$T^1 = C_N^M\cdot\omega(N,M) > C_N^M\cdot\tau_P, \tag{16}$$

где T^1 – время работы последовательного алгоритма. Далее предположим, что $C_N^M \equiv 0(\mathrm{mod}\ \ K)$ и все арифметические операции имеют одинаковую длительность τ_a, тогда будет справедлива следующая оценка:

$$S > \frac{K}{\dfrac{K(K-1)\cdot O\left(N^3\tau_a\right)}{C_N^M\tau_P} + \left(1 + \dfrac{K}{C_N^M}\right)\dfrac{((6+8M)\tau_a + \tau_P)}{\tau_P} + \dfrac{K(\tau_L + 10M\tau_b)}{\tau_P}}. \tag{17}$$

Таким образом, ускорение может приближаться к K, особенно при N и M существенно больших K.

4. Распределение ресурсов

В рассматриваемом выше алгоритме считались известными числа q_k. Собственно, к их определению и сводится задача оптимального распределения ресурсов кластера для изложенного алгоритма.

Пусть кластер состоит из K компьютеров, для каждого из которых эмпирическим путем получена функция $\Omega(k,n,N,M)$. Тогда при фиксированных значениях N и M определение q_k примет вид задачи целочисленной оптимизации с нелинейной целевой функцией (18).

$$\begin{cases} \max\limits_{k=\overline{1,K}} \Omega(k, q_k, N, M) \to \min, \\ \sum\limits_{k=1}^{K} q_k = C_N^M, \\ q_k \in N, k = \overline{1,K}. \end{cases} \tag{18}$$

В такой постановке она может быть решена, например, полным перебором, поскольку решение проводится один раз для дальнейшего многократного использования основного алгоритма.

Таким образом, основной проблемой является построение зависимостей $\Omega(k, n, N, M)$ для конкретного кластера, на котором будут производиться вычисления. При этом необходимо ограничить диапазон изменения значений N и M в зависимости от типа задач оценивания, которые будут решаться.

Следует отметить, что вычислительные затраты на определение $\Omega(k, n, N, M)$ для $n = \overline{1, C_N^M}$ соизмеримы с затратами на решение задачи с использованием последовательного алгоритма. Поэтому полезно использовать априорную информацию о вычислительных мощностях кластера для выбора верхней границы порядка C_N^M / K. Для гетерогенного кластера эта граница может заметно колебаться.

Благодарности

Работа выполнена при поддержке российско-американской программы «Фундаментальные исследования и высшее образование» и РФФИ (грант №03-01-00109)

Литература

1. *Фурсов В.А.* Проблемы вычисления оценок по малому числу наблюдений // Современные методы математического моделирования (Сб. лекций). Самара, 2001.

2. *Fursov V.A., Gavrilov A.V.* Conforming Identification of the Controlled Object // Computing, Communications and Control Technologies – CCCT 2004: International Conference Austin, Texas, USA, Proceedings.

3. *Калман Р.Е.* Идентификация систем с шумами // Успехи математических наук, 1985. Т. 40. Вып. 4 (244).

4. *Фурсов В.А.* Построение оценок по нестатистическим критериям // Труды международной конференции «ТВП-2001». Самара, 2001.

5. *Гаврилов А.В.* Сравнительный анализ критериев идентификации по малому числу наблюдений в рамках принципа согласованности оценок // Сборник трудов участников II летней школы по дифракционной оптике и обработке изображений. Самара, 2004.

6. *Головашкин Д.Л., Головашкина С.П.* Методы параллельных вычислений (Часть 2) // Самара: СГАУ, 2003.

АЛГОРИТМЫ ПАРАЛЛЕЛЬНЫХ ВЫЧИСЛЕНИЙ ДЛЯ ЗАДАЧ МОДЕЛИРОВАНИЯ СЛОЖНЫХ СИСТЕМ

Ф.Г. Гаращенко, Г.И. Ниссенбаум

Киевский национальный университет им. Тараса Шевченко, г. Киев

Введение

Использование алгоритмов параллельных вычислений позволяет эффективно решать задачи управления динамическими объектами и моделирования сложных систем. В данной работе рассмотрены аспекты разработки соответствующего программного обеспечения для решения задач управления пучками, анализа практической устойчивости и структурно-параметрической оптимизации. Изложена технология и алгоритмы решения задач на параллельных вычислительных системах. В работе используется вычислительная система типа MIMD. Как язык программирования используется язык C++ с вызовом коммуникационных примитивов из системной библиотеки. В качестве коммуникационной библиотеки используется интерфейс MPI (Message Passing Interface), какой в последнее время de-facto стал общепризнанным стандартом передачи сообщений. При выполнении работы использовались компиляторы Microsoft Visual C++ 6.0, gcc-3.0.2 и разработанный на кафедре МСС комплекс программ для проведения вычислительных экспериментов «Моделирование динамических систем» (MDS).

Постановка задачи

Рассматривается трудоемкая с вычислительной точки зрения задача нахождения оптимальных параметров ускоряюще-фокусирующей системы при отсутствующих кулоновских взаимодействиях. Полную постановку задачи можно найти в [1], [2].

Использование параллельной вычислительной техники для данного класса задач проиллюстрируем на примере решения задачи Коши. Пусть имеем систему дифференциальных уравнений с начальными условиями. Использован стиль «одна программа – много данных» (SPMD): каждый процесс выполняет один и тот же код, но обрабатывает разные части данных, т.е. вычисляется одна система дифференциальных уравнений, но для разных начальных условий. Возникает проблема балансирования нагрузки на каждый из процессорных элементов. Необходимо также минимизировать временные затраты на взаимодействие между процессами.

Решение задачи

Для решения задачи использовался вычислительный кластер Киевского национального университета имени Тараса Шевченко. Кластер построен на базе двухпроцессорных узлов с процессорами PentiumIII-933Mhz и 1Ghz. В роли служебной сетки применяется Gigabit Ethernet.

Были проведены вычислительные эксперименты над матрицами больших размеров. Рассматривались алгоритмы распределенного умножения матриц с помощью парадигм взаимодействия между процессами «управляющие - рабочие» (распределенный портфель задач) и конвейер.

Важным условием распараллеливания программы является наличие независимых вычислений, т.е. вычислений с множествами записи, которые не пересекаются.

Результаты моделирования динамики пучков заряженных частиц в ускоряюще-фокусирующих системах.

При разработке алгоритмов эффективного моделирования были использованы результаты распараллеливания градиентного метода и распределенного умножения матриц.

Будем рассматривать уравнение движения без учета кулоновских сил в системе резонансного ускорения. $E=(E_x\ E_y\ E_z)^T$ - вектор напряженности ускоряющего поля. Внешнее поле описывается соотношением $E=(E_x\ E_y\ E_z)^T\ cos\ \varphi$, где $\varphi=\frac{2\pi c}{\lambda}(t-\ t^*)-\varphi(0)$ – фаза ускоряющей волны, которая действует на частицу в момент t, t^*-время прохождения волны от начальной точки $z=0$ в точку z, $\varphi(0)$–фаза ускоряющей волны, при которой ион входит в процесс ускорения, λ – длина волны ускоряющего поля, E – вектор, что определяет амплитуду его напряженности. Если z-независимая координата, то уравнение движения будут иметь вид:

$$\begin{cases} \frac{d\gamma}{dz}=\frac{\eta}{c^2}E_z\cos\varphi, \\ \frac{d\varphi}{dz}=\frac{2\pi}{\lambda}\frac{\gamma}{\sqrt{\gamma^2-1}}; \end{cases} \tag{4.1}$$

$$\begin{cases} \frac{d^2x}{dz^2}=\frac{\eta}{c^2}\frac{\gamma}{\gamma^2-1}\left[\left(-0.5\cdot\frac{\partial E_z}{\partial z}+g(z)\right)x-E_z\frac{dx}{dz}\right]\cos\varphi, \\ \frac{d^2y}{dz^2}=\frac{\eta}{c^2}\frac{\gamma}{\gamma^2-1}\left[\left(-0.5\cdot\frac{\partial E_z}{\partial z}-g(z)\right)x-E_z\frac{dx}{dz}\right]\cos\varphi, \end{cases} \tag{4.2}$$

где $g(z)=0.5\left(\frac{\partial E_x(0,0,z)}{\partial x}-\frac{\partial E_y(0,0,z)}{\partial y}\right),\ z\in[0,T]$.

Система (4.1) описывает продольное движение, а система (4.2) радиальное движение частиц.

Ниже приведена упрощенная модель ускоряюще-фокусирующей системы при отсутствующих кулоновских взаимодействиях [1].

$$\begin{cases} \dfrac{d\gamma}{d\xi} = u(\xi)\cos(\varphi), \\ \dfrac{d\varphi}{d\xi} = \dfrac{2\pi\gamma}{\sqrt{\gamma^2 - 1}}. \end{cases} \tag{4.3}$$

На функцию управления u(ξ) наложим одно из следующих ограничений [1]:

$$u(\xi) \in \Omega_u = \{u(\xi) : 0 \le u(\xi) \le c, \xi \in [0,T]\}, \tag{4.4}$$

$$u(\xi) \in \Omega_u = \{u(\xi) : |u(\xi)| \le c, \xi \in [0,T]\}.$$

Моделирование было проведено при следующих значениях параметров

a=1,15; b=1,17 (для φ_0);

γ_0 =1,5; c=-0,03, T=3,54.

Анализ полученных рисунков подтверждают устойчивость системы.

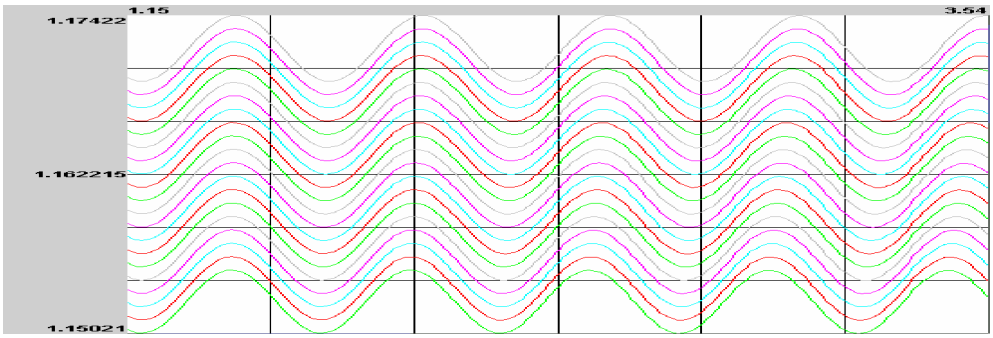


Рис. 1. Пучок заряженных частиц, который состоит из 20 траекторий (координата φ)

Использование параллельной вычислительной техники и разработка соответствующих алгоритмов разрешит эффективно моделировать и решать сложные задачи управления пучками, практической стойкости и структурно-параметрической оптимизации. Важной является также оценка зависимости времени выполнения программы от количества процессов.

Таким образом, главным в работе является: реализация подхода крупноблочного распараллеливания алгоритмов на кластерной архитектуре, описание численного эксперимента по решению оптимизационных задач и задач численного моделирования на примере задач моделирования траекторий заряженных частиц, исследование практической стойкости и структурно-параметрической оптимизации.

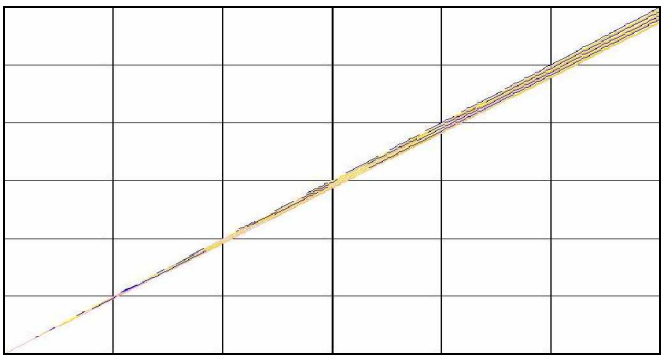


Рис. 2. Пучок заряженных частиц,
который состоит из 20 траекторий (координата γ)

Литература

1. *Бублик Б.Н., Гаращенко Ф.Г., Кириченко Н.Ф.* Структурно-параметрическая оптимизация и устойчивость динамики пучков // К.: Научная мысль, 1985. - 304 с.
2. *Башняков О.М., Гаращенко Ф.Г., Лежебока В.В.* Практическая стойкость и структурная оптимизация динамических систем // К.: Издательско-полиграфический центр «Киевский университет», 2000. - 197 с.
3. *Воеводин Вл.В.* Курс лекций «Параллельная обработка данных» // http://kiev.parallel.ru/.
4. *Gregory R. Andrews.* Foundations of Multithreaded, Parallel, and Distributed Programming-University of Arizona: Addison-Wesley, 2000, ISBN: 0-201-35752-6

РАЗРАБОТКА ИНТЕГРИРОВАННОЙ СРЕДЫ ВЫСОКОПРОИЗВОДИТЕЛЬНЫХ ВЫЧИСЛЕНИЙ ДЛЯ КЛАСТЕРА НИЖЕГОРОДСКОГО УНИВЕРСИТЕТА

В.П. Гергель, А.Н. Свистунов

Нижегородский государственный университет им. Н.И. Лобачевского, г. Нижний Новгород

В работе рассматриваются проблемы создания интегрированной среды высокопроизводительных вычислений для кластера Нижегородского университета, который был предоставлен ННГУ в рамках академической программы Интел в 2001 г. [3]. Важной отличительной особенностью кластера является его неоднородность (гетерогенность). В состав кластера входят рабочие места, оснащенные процессорами Intel Pentium 4 и соединенные относительно медленной сетью (100 Мбит), а также вычислительные 2- и 4-процессорные серверы, обмен данными между которыми выполняется при помощи быстрых каналов передачи данных (1000 Мбит). Основной операционной системой, установленной на узлах кластера явля-

ется ОС Windows (на рабочих станциях установлен Windows 2000 Professional, на серверах установлена Windows 2000 Advanced Server).

Дополнительная информация о структуре кластера ННГУ доступна в сети Интернет по адресу http://www.software.unn.ac.ru/cluster.htm.

Эффективное использование быстродействующего компьютерного оборудования предполагает решение двух основных проблем, возникающих при эксплуатации кластера – проблему предоставления удаленного доступа к вычислительной среде кластера и проблему эффективного управления и мониторинга вычислительных задач, выполняющихся на кластере [1].

На сегодняшний день известно довольно много различных программных систем, позволяющих решать отмеченные проблемы. Большинство подобных систем традиционно разрабатывались для ОС UNIX, однако, в последнее время появились подобные системы и для ОС семейства Windows. К числу таких систем относится, например LSF (Load Sharing Facility, http://www.platform.com), или Cluster CoNTroller (http://www.mpi-softtech.com/).

Однако использование готовых систем управления кластером затруднено рядом обстоятельств. Прежде всего, это высокая стоимость подобных систем, достигающая, для кластера подобного кластеру Нижегородского университета, десятков тысяч долларов. Вторым, не менее важным обстоятельством, является закрытость подобных систем, что осложняет проведение работ по исследованию различных алгоритмов управления кластерных систем. Для кластера Нижегородского университета, в силу его неоднородности, планирование распределения вычислительной нагрузки по узлам кластера является практически важной задачей. Как результат, отмеченные факторы приводят к необходимости создания собственных средств поддержки организации высокопроизводительных вычислений на кластере.

В целом, для организации высокопроизводительных кластерных вычислений соответствующая программная среда поддержки должна обеспечивать:

– реализацию достаточного набора операций для выполнения вычислительных заданий на кластере (загрузка задачи, добавление задачи в очередь задач, получение текущего статуса очереди задач и текущей выполняемой задачи, получение результатов вычислений);

– наличие простого и удобного способа удаленного доступа к кластеру, не требующего установки на рабочих станциях пользователей какого-либо специального программного обеспечения;

– создание собственной системы авторизации пользователей, не связанной напрямую с системой авторизации операционной системы;

– организацию системы очередей задач;

– сохранение задач пользователя после их выполнения и возможность их повторного запуска;

– запоминание результатов выполнения вычислительных заданий.

При построении программной системы за основу была взята сложившаяся архитектура системы мониторинга и управления [2], включающая, как правило, следующие составные части:

1) компонент для взаимодействия с пользователем (Менеджер доступа), позволяющий ставить задачи в очередь, удалять задачи из очереди, просматривать статус задач и т.д., а также ведущий базу данных пользователей и базу данных результатов;
2) компонент для организации очереди заданий (Диспетчер заданий), обеспечивающий выполнение работ по управлению потоком заданий (накопление, упорядочение, просмотр и выборку);
3) компонент для управления процессом выполнения заданий на кластере (Супервизор кластера), осуществляющий его мониторинг, выборку заданий из очереди и распределение вычислительных ресурсов кластера.

В настоящее время разработан и внедрен опытный вариант системы, обладающий следующими возможностями:

- обеспечение доступа к системе с любого компьютера, подключенного к сети Интернет;
- использование в качестве средств доступа Web-браузера, telnet-клиента, различных специализированных программ (что позволяет, в частности, не устанавливать на компьютерах пользователей какого-либо дополнительного программного обеспечения);
- организацию очереди заданий (как ждущих своей очереди на выполнение, так и уже завершившихся, сформировавших результат) во внешней базе данных, что позволяет обеспечить устойчивость системы в случае сбоя;
- замену (при необходимости) планировщика и изменение стратегии распределения вычислительных ресурсов кластера;
- ведение статистики использования задачами пользователей вычислительных ресурсов кластера.

В настоящее время система активно используется широким кругом пользователей и сотрудников Центра компьютерного моделирования Нижегородского университета. Возможности, предоставляемые системой, используются при разработке и эксплуатации учебных и исследовательских программных комплексов - таких, например, как программная система для изучения и исследования параллельных методов решения сложных вычислительных задач (ПараЛаб) и система параллельной многоэкстремальной оптимизации Абсолют Эксперт.

Литература

1. *Rajkumar Buyya*. High Performance Cluster Computing // Volume1: Architectures and Systems. Volume 2: Programming and Applications. Prentice-Hall Inc., 1999.

2. *Saphir W., Tanner L.A., Traversat B.* Job Management Requirements for NAS Parallel Systems and Clusters // NAS Technical Report NAS-95-006 February 95.

3. *Гергель В.П., Стронгин Р.Г.* Высокопроизводительный вычислительный кластер Нижегородского университета // Материалы конференции Relarn. - Н. Новгород, 2002.

ПАРАЛЛЕЛЬНЫЕ МЕТОДЫ ВЫЧИСЛЕНИЯ ДЛЯ ПОИСКА ГЛОБАЛЬНО ОПТИМАЛЬНЫХ РЕШЕНИЙ

В.П. Гергель, Р.Г. Стронгин

Нижегородский государственный университет им. Н.И. Лобачевского, г. Нижний Новгород

Исследование многих математических моделей, рождаемых потребностями приложений, часто включает построение оценок некоторой величины, характеризующей заданную область Q в многомерном евклидовом пространстве R^N. В качестве типового примера таких задач может рассматриваться, например, *задача нелинейного программирования*

$$\varphi^* = \varphi(y^*) = \min\{\varphi(y) : y \in Q\},\ Q = \{y \in D : g_i(y) \le 0, 1 \le i \le m\}$$

где *область поиска* определяется в некотором N-мерном гиперпараллелепипеде

$$D = \{y \in R^N : a_j \le y_j \le b_j, 1 \le j \le N\}$$

системой нелинейных ограничений $g_i(y) \le 0,\ 1 \le i \le m$. В случае многоэкстремальности функции $\varphi(y)$ рассмотренная постановка называется *задачей глобальной оптимизации*, для которой искомые оценки $(y^*, \varphi^* = \varphi(y^*))$ являются *интегральными характеристиками* минимизируемой функции $\varphi(y)$ во всей области D.

1. Рассмотренный пример может быть расширен аналогичным описанием задач интегрирования, решения систем нелинейных уравнений, восстановления зависимостей и т.п. Как результат такого перечисления можно говорить о **существовании широкого класса важных прикладных задач**, требующих оценки некоторой величины (интеграла, глобального минимума, множества не доминируемых решений и т.п.) путем анализа поведения заданной функции $F(y)$ в некотором гиперпараллелепипеде D (в общем случае, функция $F(y)$ может быть век-

торной для учета, например, многокритериальности или ограничений задачи оптимизации).

2. Задача построения искомой оценки могла бы быть решена на основе анализа вычисленных в узлах некоторой сетки $Y_T = \{y^t : 1 \le t \le T\}$ области D множества значений:

$$Z^t = F(y^t), \quad y^t \in D, \; 1 \le t \le T.$$

Возможный (и широко используемый в практических приложениях) способ построения таких сеток состоит в равномерном расположении узлов (*равномерность* расположения узлов позволяет уменьшить их общее число T и, следовательно, сократить вычислительные затраты). Однако проблема состоит в том, что число узлов этой сети экспоненциально увеличивается с ростом размерности N области D. Действительно, для равномерной сетки в области D, имеющей шаг $\Delta > 0$, справедлива оценка:

$$T = T(\Delta) \approx \prod_{j=1}^{N} \left[(b_j - a_j) / \Delta \right].$$

В результате для многих актуальных прикладных задач реализация описанной схемы полного перебора на равномерной сетке оказывается невозможной в силу ее высокой потребности в вычислительных ресурсах.

Дело усложняется еще и тем, что во многих приложениях выполнение операции оценки вектора $F(y)$ для выбранной точки $y \in D$ связано с вычислительным анализом математических моделей, отличающихся высокой степенью сложности. Причем практика последних десятилетий показывает, что впечатляющий воображение быстрый рост производительности компьютеров сопровождается столь же быстрым усложнением рассматриваемых прикладных задач и описывающих их моделей. Разумеется, существуют и многие более простые задачи, решение которых вполне осуществимо описанным выше методом перебора узлов равномерной сетки. Основное предложение, направленное на повышение эффективности анализа сложных задач, связано с переходом к целенаправленному анализу вариантов, в ходе которого вычисления, осуществленные в одних узлах сетки, позволяют исключить необходимость вычислений во многих других узлах, т.е. речь идет об использовании существенно не равномерных сеток. Так, например, применительно к рассмотренной выше задаче глобальной минимизации это означает, что сетка должна быть более плотной в окрестности искомого глобального минимума и – заметно менее плотной вдали от точки минимума. По существу это означает, что такая сетка не может быть задана априори (ибо расположение глобального минимума является не известным). Ее необходимо строить в процессе анализа.

3. Основная идея сокращения объема вычислений, необходимых для оценки характеристик многомерной области, состоит в использовании неравномерных сеток, способных обеспечивать ту же точность решения, что и при равномерных сетках. При этом несколько узлов $y^1, \mathbf{h}, y^t, t \geq 1,$ сетки Y_t могут быть заданы априори, однако, все остальные узлы $y^k, k \geq t,$ последовательно определяются некоторым решающим правилом

$$y^{k+1} = G_k\left(y^1, \mathbf{h}, y^k; Z^1, \mathbf{h}, Z^k\right), \ k \geq t,$$

где величины $Z^l, 1 \leq l \leq k,$ являющееся аргументами решающей функции G_k, есть значения функции $F(y)$, вычисленные в узлах $y^l, 1 \leq l \leq k.$ Каждая конкретная система решающих функций, предлагаемая для определенного класса задач, основывается на некоторых априорных предположениях о вектор–функции $F(y)$, что и позволяет использовать накапливаемую в процессе решения задачи информацию

$$\omega_k = \left\{ \ y^1, \mathbf{h}, y^k; Z^1, \mathbf{h}, Z^k \ \right\}$$

для построения неравномерных сеток.

4. Многопроцессорные системы кластерного типа [1] позволяют ускорить решение задач рассмотренного типа. Для задач, в которых определение значений функции $F(y)$ основано на требующем значительных компьютерных ресурсов численном анализе сложных математических моделей, наиболее целесообразным подходом для распараллеливания вычислений является одновременное вычисление значений функции $F(y)$ в нескольких узлах используемой сетки Y_t. При этом каждому из имеющихся $p > 1$ процессоров задается конкретная точка $y^l \in D, 1 \leq l \leq p,$ в которой этот процессор (будучи загружен необходимым комплектом программ) определяет значение всех или части координатных функций $F_i(y^l), 1 \leq i \leq s$. Для реализации такого подхода решающее правило алгоритма должно определять одновременно $p > 1$ точек следующих итераций

$$(y^{k+1}, \mathbf{h}, y^{k+p}) = G_{k,p}(\omega_k), \quad k \geq t, \quad y^{k+l} \in D, 1 \leq l \leq p,$$

передаваемых отдельным процессорам, (алгоритмы подобного типа, обобщающие эффективные последовательные схемы и характеризуемые низкой избыточностью, изложены в [2]).

5. Время выполнения испытаний в различных точках области поиска, в общем случае, является неодинаковым. Поскольку рассмотренная схема распараллеливания определяет точки следующих p испытаний после получения результатов *всех* предшествующих испытаний, то часть процессоров будет простаивать, ожидая завершения работы дру-

гих процессоров. Этот недостаток может быть преодолен введением следующей *асинхронной схемы*.

Введем обозначение $y^{k+1}(j)$ для точки, в которой процессор с номером $j, 1 \le j \le p,$ осуществляет $(k+1)$–е испытание. При этом $k = k(j), 1 \le j \le p$. В целях обеспечения компактности последующих записей, введем также единую нумерацию всех точек гиперинтервала D, в которых испытания уже завершены. Используя верхние индексы, определим множество

$$Y_\theta = \{y^1, \ldots, y^\theta\} = \bigcup_{j=1}^{p} \bigcup_{i=1}^{k(j)} y^k(j),$$

в точках которого вычислены значения функции $F(y)$, составляющие массив результатов

$$\omega_\theta = \{\omega^1, \ldots, \omega^\theta\} = \{(y^1, Z^1), \ldots, (y^\theta, Z^\theta)\}, \quad \theta = \sum_{j=1}^{p} k(j),$$

полученных всеми процессорами в результате θ завершившихся испытаний.

При этом выбор точки $y^{k+1}(j), k = k(j),$ очередного испытания, которое должно быть реализовано на j–м процессоре, может быть осуществлен с использованием информации ω_θ. Дополнительно может быть учтена также информация о точках $y^{k+1}(i), k = k(i),$ в которых осуществляют испытания другие процессоры $(1 \le i \le p, i \ne j)$. Множество этих точек обозначим

$$Y_\theta(j) = \bigcup_{i=1, i \ne j}^{p} y^{k(i)+1}(i).$$

В результате, решающее правило для j–го процессора может быть построено в форме функции

$$y^{k+1}(j) = G_{\theta, j}(\omega_\theta, Y_\theta(j)), \quad k = k(j), \quad 1 \le j \le p.$$

Заметим, что, согласно описанной схеме, информация о числе испытаний, осуществленных другими процессорами, анализируется j–м процессором в момент, после завершения очередного испытания. Поэтому, в силу допущения, что моменты выбора точек очередных испытаний различными процессорами не синхронизованы, возможны различия значений θ, зафиксированных различными процессорами в один и тот же момент времени (т.е. $\theta = \theta(j)$). Фактически, предложенная схема интерпретирует результаты испытаний, проведенных другими процессорами, как полученные данным процессором.

6. Возможный способ организации параллельных испытаний в соответствии с предложенной схемой состоит в том, что каждый процессор независимо от других реализует свое решающее правило для выбора точек испытаний. При этом каждым процессором используется об-

щий информационный массив ω_θ и множество $Y_\theta(j)$, что предполагает обмен информацией между процессорами. Подразумевается, что каждый процессор, завершив испытание, посылает сообщение с результатами вычислений, всем другим процессорам. Кроме того, выбрав некоторую точку для очередного испытания, процессор информирует об этом выборе все остальные процессоры. При наличии указанных обменов *каждый* процессор рождает решение исходной задачи во всей области D (или Q). Следовательно, предлагаемая схема не включает какого-либо выделенного (управляющего) процессора, что повышает надежность функционирования при возможных аппаратных или программных сбоях отдельных процессоров, а также в случае передачи части процессоров для обеспечения других клиентов или других нужд вычислительной системы. В последнем случае (т.е. при уменьшении числа p используемых процессоров) не требуется какого-либо специального информирования об этом изменении значения p.

7. Для иллюстрации изложенного подхода приведем пример решения 5–мерной задачи с 5 ограничениями вида [3]:

$$g_1(w) = -(x + y + z + u + v) \le 0,$$

$$g_2(w) = (y/3)^2 + (u/10)^2 - 1.4 \le 0,$$

$$g_3(w) = 3 - (x+1)^2 - (y+2)^2 - (z-2)^2 - (v+5)^2 \le 0,$$

$$g_4(w) = 4x^2 \sin x + y^2 \cos(y+u) + z^2\left[\sin(z+v) + \sin(10(z-u)/3)\right] - 4 \le 0,$$

$$g_5(w) = x^2 + y^2\left[\sin((x+u)/3 + 6.6) + \sin((y+v)/2 + 9) + 0.9\right]^2 -$$

$$-17\cos^2(z + x + 1) + 16 \le 0,$$

где вектор параметров $w = (x, y, z, u, v)$ принадлежит гиперпараллелепипеду:

$$-3 \le x, y, z \le 3, \quad -10 \le u, v \le 10.$$

Минимизируемая функция является многоэкстремальной и определяется выражением:

$$\varphi(w) = \sin(xz) - (yv + zu)\cos(xy).$$

Равномерная сетка с шагом 0.01 (по каждой координате) содержит порядка $8 \cdot 10^{14}$ узлов. Далее задача была редуцирована к 10 связанным задачам оптимизации; для их одновременного решения был использован параллельный многомерный индексный метод [2] для вычислений в многопроцессорной системе с 10 процессорами. Общее число итераций составило 59697. Полученная оценка

$$w'' = (-0.068, 1.962, 3.431, 9.833, 9.833)$$

характеризуется значением $\varphi(w'') = -42.992$, которое улучшилось (после локального уточнения решения) до значения $\varphi(w^{**}) = -43.2985$.

Для проведения рассмотренных экспериментов использовалось оборудование компаний Intel и Hewlett Packard, переданное в Нижегородский госуниверситет в качестве грантов для образовательных и научно-исследовательских целей.

Исследования поддержаны грантом РФФИ № 04-01-00455-а.

Литература

1. *Pfister G.P.* (1995). *In Search of Clusters.* Prentice Hall PTR, Upper Saddle River, NJ (2^{nd} edn., 1998).

2. *Strongin R.G., Sergeyev Ya.D.* (2000). *Global optimization with non-convex constraints: Sequential and parallel algorithms.* Kluwer Academic Publishers, Dordrecht.

3. *Гергель В.П., Стронгин Р.Г.* Параллельные вычисления в задачах выбора глобально-оптимальных решений для многопроцессорных кластерных систем // Материалы третьего Международного научно-практического семинара «Высокопроизводительные параллельные вычисления на кластерных системах». Н. Новгород: Изд-во Нижегородского университета. 2003. С. 169-191.

ПАРАЛЛЕЛЬНЫЕ АЛГОРИТМЫ МЕТОДА ВСТРЕЧНЫХ ПРОГОНОК

Д.Л. Головашкин

Институт систем обработки изображений РАН, г. Самара
Самарский государственный аэрокосмический университет имени академика С.П. Королева, г. Самара

Введение

Моделирование физических процессов посредством численного решения дифференциальных уравнений находит все более широкое применение в различных отраслях науки. Этому способствует развитие вычислительной техники и численных методов, ориентированных на решение таких уравнений. Наиболее распространенные методы (разностные и проекционные) сводят дифференциальную задачу к системе линейных алгебраических уравнений (СЛАУ) вида $Ax=b$, где матрица A – ленточная.

С появлением вычислительных систем, допускающих параллельные вычисления, связано создание параллельных алгоритмов решения таких систем, основанных как на ранее известных последовательных алгоритмах (методы прогонки [1], циклической редукции [2]), так и алгоритмов, изначально создававшихся как параллельные. По мнению автора в данном случае уместна классификация, разделяющая параллельные алгоритмы на алгоритмы с функциональной декомпозицией и с декомпозицией данных [3].

В [4] опубликован один из первых алгоритмов, основанных на декомпозиции данных, при которой производится распределение матрицы системы между задачами (в терминах модели канал/задача [3]), позволяющее каждой задаче алгоритма решать свою подсистему, выражая решение через значения на границах области данных, входящих в задачу. Для вычислительных систем с различной архитектурой синтезировано целое семейство параллельных алгоритмов, основанных на этом принципе. К недостаткам данного подхода следует отнести высокие требования к сетевой конфигурации вычислительной системы (если вычисления производятся на кластере, то необходимо «бинарное дерево») и отсутствие учета особенностей сеточной области, на которой решается исходная дифференциальная задача.

Функциональная декомпозиция, применительно к данной задаче, была использована в работе [5]. Автор алгоритма, рассматривая разностное решение двумерного эллиптического уравнения на прямоугольной области, отказался от идеи одновременного решения одной СЛАУ несколькими задачами. Все задачи реализуют прямой или обратный проходы скалярной прогонки для своих участков различных СЛАУ. В случае вытянутой в одном направлении сеточной области ускорение данного алгоритма превышает ускорение алгоритма из [4], при этом используется простейшая сетевая конфигурация «линия». Недостатком функциональной декомпозиции для алгоритма скалярной прогонки является простои задач, ограничивающий рост масштабируемости [3] алгоритма при сгущении сеточной области вдоль выбранного направления. Представляемая работа является развитием данного подхода. Применив его к методу встречных прогонок [1] автор вдвое сократил длительность простоев параллельного алгоритма, сохранив достоинства функциональной декомпозиции для вытянутых прямоугольных сеточных областей. Такие области встречаются, например, в задачах дифракционной оптики и теплопроводности при моделировании работы дифракционных оптических элементов.

1. Алгоритм для одномерной сеточной области

Очевидно, при разностном решении дифференциального уравнения (пусть эллиптического) на одномерной сеточной области ω_1 следует говорить об одной СЛАУ трехдиагонального вида (применяется простейшая неявная разностная схема из [6]). В этом случае наиболее уместны параллельные алгоритмы с декомпозицией данных, но для иллюстрации основной идеи данной работы построим параллельный алгоритм с функциональной декомпозицией. Непосредственное использование алгоритма из [3] невозможно в силу единственности решаемой СЛАУ. Однако, составление двухзадачного алгоритма, основанного на методе встречных прогонок возможно.

Функциональная декомпозиция задается спецификой метода встречных прогонок, в котором прогоночные коэффициенты (α, β и η, ζ) вычисляются с двух сторон по направлению к центру матрицы A. В последовательном алгоритме этот прием используется, когда желают ограничиться нахождением не всего вектора x, а лишь его части. Так как вычисления двух пар коэффициентов прогонки независимы (информационно, логически и конкуренционно [7]), то их можно находить параллельно. Произведем разбиение одномерной сеточной области ω_1 на ω_1^1 и ω_1^2 (рис. 1). Вычисления разделим на три этапа, соответствующие прямому (рис. 1а), обратному ходу (рис. 1в) встречной прогонки и обмену данными между задачами (рис. 1б).

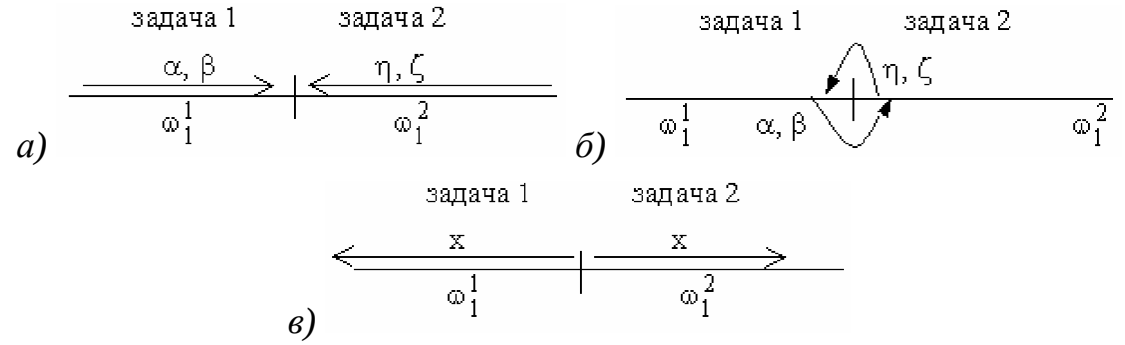


Рис. 1. Этапы вычислений по двухзадачному параллельному алгоритму, реализующему метод встречных прогонок на одномерной сеточной области: а – прямой ход прогонок; б – обмен данными между задачами; в – обратный ход прогонок

В течение прямого хода прогонки первая задача находит прогоночные коэффициенты α, β, вторая задача η, ζ. Далее, первая задача передает второй пару чисел α, β, необходимых для запуска обратного хода прогонки во второй задаче, которая в это время производит аналогичную передачу, необходимую первой задаче. На третьем этапе обе задачи одновременно реализуют обратный ход прогонки, находя решение СЛАУ. Ускорение такого алгоритма можно оценить как:

$$s = \frac{\tau_a(N)}{½\tau_a(N) + 2\tau_k},$$

где N – размерность СЛАУ, $\tau_a(N)=3N\tau_\times+(2N+1)\tau_\div+3N\tau_+$ – время расчета по последовательному алгоритму (τ_+ – длительность одной операции сложения и вычитания), $\tau_\times$ – длительность одной операции умножения, $\tau_\div$ – длительность одной операции деления), $\tau_к$ – время передачи или приема (считаем их равными) одного пакета по сети. При пакетной модели передачи данных время передачи разного объема информации будет оставаться константой до тех пор, пока объем передаваемого массива не

превысит размер пакета. Здесь это условие соблюдается (передается всего пара чисел), в дальнейшем также будем полагать его верным.

Очевидно, при $N\to\infty$ $s\to 2$. К сожалению, этот подход нельзя развить для большего количества задач, если сеточная область ω одномерна.

2. Алгоритм для двумерной сеточной области

Для двумерной области ω_2 это ограничение снимается. Применительно к ней будем говорить о прогонке по строкам (продольной прогонке) и столбцам (поперечной прогонке) сеточной области, как это принято в методах расщепления и переменных направлений [6].

Если алгоритм состоит из двух задач, то модификации, по сравнению с предыдущим алгоритмом, не требуется, каждая СЛАУ в продольном направлении решается одновременно двумя задачами. Пусть размер сеточной области ω_2 – $N\times M$ узлов (разбиение производится по M), тогда ускорения такого алгоритма s_2 и известного алгоритма из [5] $\tilde{s}_2$ составят

$$s_2=\frac{N\tau_a(N)+M\tau_a(M)}{\frac{N}{2}\tau_a(N)+\frac{M}{2}\tau_a(M)+2N\tau_k} \text{ и } \tilde{s}_2=\frac{N\tau_a(N)+M\tau_a(M)}{\frac{N}{2}\tau_a(N)+\frac{M}{2}\tau_a(M)+2N\tau_k+\tfrac{1}{2}\tau_a(M)}.$$

Четвертое слагаемое в знаменателе последней формулы есть время ожидания для задачи в известном алгоритме. Двухзадачный алгоритм, основанный на методе встречных прогонок, свободен от задержек вычислений, связанных с ожиданиями.

Для алгоритма из четырех задач производится следующее разбиение сеточной области (рис. 2).

На рис. 2а-2в представлены начальные этапы вычислений по алгоритму. Задачи 1, 4 (рис. 2а) начинают прямой ход прогонки для строки 1 и передают прогоночные коэффициенты задачам 2, 3, которые на первом этапе простаивают.

Далее не будем специально говорить об обмене данными между задачами, подразумевая, что перед прямым ходом они принимают прогоночные коэффициенты, после прямого хода передают их. Перед обратным ходом принимают значение сеточной функции, после – передают его. Первая и последняя задачи начинают прямой ход без принятия прогоночных коэффициентов, а задачи под номерами $L/2$ и $L/2+1$ (L – общее число задач, в данном алгоритме $L=4$) начинают обратный ход принятием прогоночных коэффициентов (друг от друга).

На втором этапе (рис. 2б) задачи 2, 3 продолжают прямой ход для строки 1, задачи 1, 4 начинают прямой ход для строки 2. Третий этап (рис. 2в) характеризуется простоем задач 1, 4 и началом обратного хода прогонки для строки 1 задачами 2, 3. Четвертый этап (рис. 2г) – прямой ход, задачи 1, 4 осуществляют его для строки 3, задачи 2, 3 для строки 2.

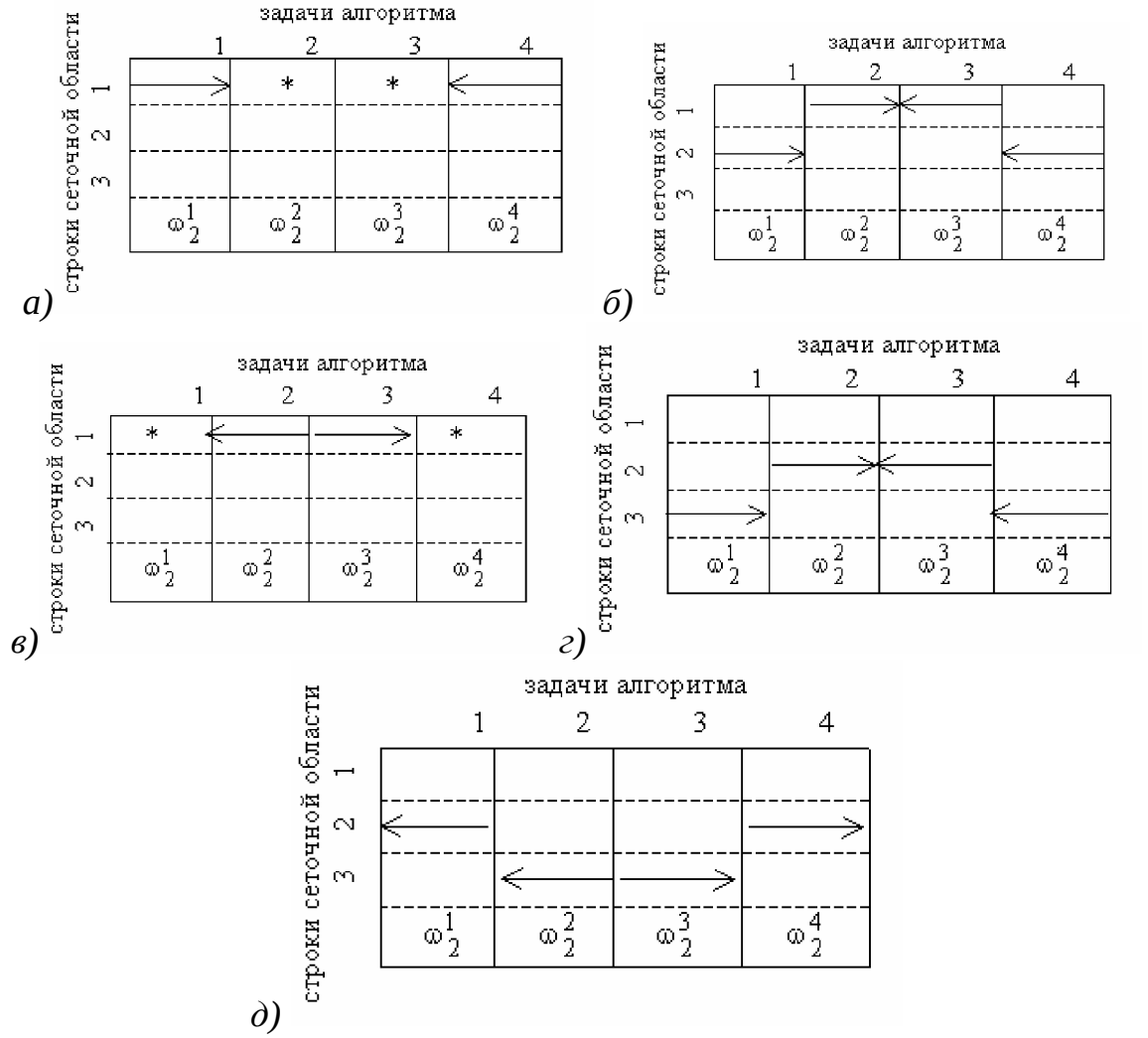


Рис. 2. Этапы вычислений по четырехзадачному параллельному алгоритму для двумерной сеточной области. а, б, г – прямые ходы прогонок; в, д – обратные ходы. Задачи, помеченные символом «», простаивают*

Пятый этап (рис. 2д) – обратный ход, задачи 1, 4 осуществляют его для строки 2, задачи 2, 3 для строки 3. Далее прямой и обратный ходы чередуются, пока задачи 2, 3 не произведут прямой ход прогонки для последней строки. На следующем этапе задачи 1, 4 будут простаивать, задачи 2, 3 начнут обратный ход. Заключительный этап алгоритма характеризуется простоем задач 2, 3, в то время, как задачи 1, 4 завершат обратный ход прогонки для последней строки сеточной области. Прогонка по столбцам осуществляется каждой задачей самостоятельно, эти вычисления независимы.

Ускорения данного алгоритма и известного из [5] оценивается величинами

$$s_4 = \frac{N\tau_a(N) + M\tau_a(M)}{\frac{N}{4}\tau_a(N) + \frac{M}{4}\tau_a(M) + 4N\tau_k + 0{,}25\tau_a(M)}$$

и $\hat{s}_4 = \dfrac{N\tau_a(N)+M\tau_a(M)}{\frac{\mathrm{N}}{4}\tau_a(N)+\frac{\mathrm{M}}{4}\tau_a(M)+4N\tau_k+0{,}75\tau_a(M)+4\tau_k}$,

где $0{,}25\tau_a(M)$ – время ожидания в разработанном алгоритме, $0{,}75\tau_a(M)+4\tau_к$ – время ожидания в известном алгоритме.

Распространим данный подход на произвольное число задач L. На начальном этапе такого алгоритма в течении первых $L/2$ шагов производится только прямой ход прогонки. Задача l (пусть $l \le L/2$) простаивает l-1 шаг, затем производит прямой ход прогонки для строк 1,2, …, $L/2$-l+1. Затем происходит чередование прямого и обратного хода, причем задача l простаивает шаги, связанные с обратным ходом в течении $L/2$-l шагов обратного хода. Далее происходит чередование без простоев. Если в течение прямого хода задача *1* производит прямой ход прогонки для строки i, то задача l производит прямой ход для строки i-l+1. Когда первая задача начинает обратный ход для строки i, задача l начинает обратный ход для i+l-1. Последние шаги алгоритма опять связаны с простоями, когда во время прямого хода простаивают задачи (для $l \le L/2$) с меньшими номерами, во время обратного – с большими номерами. Аналогично проводятся рассуждения для $l > L/2$.

Ускорения такого алгоритма и традиционного из [5] составят:

$$s_L = \frac{N\tau_a(N)+M\tau_a(M)}{\frac{\mathrm{N}}{L}\tau_a(N)+\frac{\mathrm{M}}{L}\tau_a(M)+4N\tau_k+\frac{L-2}{2L}\tau_a(M)+(L-4)\tau_k}$$

$$\text{и } \hat{s}_L = \frac{N\tau_a(N)+M\tau_a(M)}{\frac{\mathrm{N}}{L}\tau_a(N)+\frac{\mathrm{M}}{L}\tau_a(M)+4N\tau_k+\frac{L-1}{L}\tau_a(M)+2(L-2)\tau_k}.$$

По сравнению с известным алгоритмом, основанном на функциональной декомпозиции длительность простоев сокращена в два раза.

Оценка сверху для ускорения лучшего алгоритма, основанного на декомпозиции данных составит

$$\tilde{s}_L = \frac{N\tau_a(N)+M\tau_a(M)}{\frac{\mathrm{N}}{L}\tau_a(N)+\frac{\mathrm{M}}{L}\tau_a(M)+2N(\log_2 L)\tau_k}.$$

Определим пороговую величину $f = \dfrac{\log_2 L}{2+\frac{L-2}{4LN}\lambda(M)+\frac{L-4}{2N}}$, где $\lambda(M) = \dfrac{\tau_a(M)}{\tau_k}$. При $f>1$ можно утверждать, что ускорение представленного алгоритма превзойдет ускорение лучшего, основанного на декомпозиции данных. Чем больше значения принимают величины L, N и чем меньше λ (лучше быстродействие процессора и медленнее передача данных по сети), тем уместней использование разработанного алгоритма.

Например, положив L=16 и $\zeta(M)$=200, исследуем зависимость $f(N)$ (рис. 3). При N>24 представленный алгоритм характеризуется большим ускорением, чем известные.

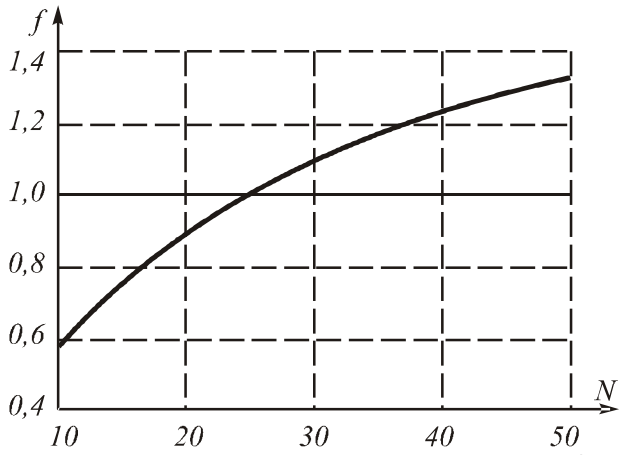


Рис. 3. Зависимость параметра f от количества узлов сеточной области N

Отметим, однако, что как только увеличение параметра N позволит предпочесть двумерную декомпозицию области ω_2 одномерной (так как область более нельзя будет считать вытянутой в выбранном направлении), алгоритм с декомпозицией данных станет оптимальнее в смысле ускорения. Вопрос о применении к исследуемой сеточной области того или иного алгоритма в каждом конкретном случае решается с учетом особенностей имеющейся в распоряжении исследователя вычислительной системы (задающей λ).

К достоинству представленного подхода следует отнести более мягкие требования к конфигурации сети, соединяющей процессоры вычислительной системы. Если организация вычислений по параллельному алгоритму с декомпозицией области потребует сетевую конфигурацию «бинарное дерево», то использование предложенного алгоритма позволяет ограничиться «процессорной линией».

Выводы

Применение функциональной декомпозиции к синтезу параллельных алгоритмов, основанных на методе встречных прогонок, для решения сеточных уравнений трехдиагонального вида оправдано для вытянутых прямоугольных сеточных областей и для реализации на кластерах, процессоры которых соединены в линию. Представляется целесообразным развитие данного подхода для алгоритма циклической прогонки и ленточных систем с большим количеством диагоналей.

Благодарность

Работа выполнена при поддержке Министерства образования РФ, Администрации Самарской области и Американского фонда гражданских исследований и развития (CRDF) в рамках Российско-американской программы «Фундаментальные исследования и высшее образование» (BRHE), а также Фонда содействию отечественной науке.

Литература

1. *Самарский А.А., Николаев Е.С.* Методы решения сеточных уравнений // М.: Наука. 1978. - 561 с.

2. *Ортега Джеймс М.* Введение в параллельные и векторные методы решения линейных систем // Перевод с англ. Х.Д. Икрамова, И.Е. Капорина; под ред. Х.Д. Икрамова. М.: Мир, 1991. – 364 с.

3. *Braunl T.* The art of parallel programming // Prentice Hall International (UK) Limited, 1993. 378 p.

4. *Яненко Н.Н., Коновалов А.Н., Бугров А.Н., Шустов Г.В.* Об организации параллельных вычислений и «распараллеливание» прогонки // Численные методы механики сплошной среды / ИТПМ СО АН СССР. Новосибирск, 1978. Т. 9. С. 139-146.

5. *Миренков Н.Н.* Параллельные алгоритмы для решения задач на однородных вычислительных системах // Вычислительные системы. ИМ СО АН СССР. Новосибирск, 1973. Вып. 57. С. 3-32.

6. *Самарский А.А.* Теория разностных схем // М.: Наука, 1989. - 614 с.

7. *Вальковский В.А., Котов В.Е., Марчук А.Г., Миренков Н.Н.* Элементы параллельного программирования // М.: Радио и связь, 1983. - 239с.

ПОВЫШЕНИЕ ПРОИЗВОДИТЕЛЬНОСТИ КОЛЛЕКТИВНЫХ ОПЕРАЦИЙ MPICH-2

А.В. Гришагин

Нижегородский государственный университет, г. Нижний Новгород

Введение

Распространенность кластерных установок в мире параллельного аппаратного обеспечения, а так же невысокая стоимость и простота построения приводят к огромному количеству разнообразных конфигураций. Фактически, каждый кластер в некотором роде «уникален», кроме тех, которые специально разрабатываются и продаются единым комплексом. Ещё большее разнообразие вносит программное обеспечение: известно, например, что смена версий ядра Linux может изменить коммуникационные задержки вплоть до 2-х раз. Такое разнообразие в производительности, и, более того, отсутствие стабильности в производительности должно учитываться при создании средств параллельного взаимодействия, ведь

разные режимы передачи для «больших» и «маленьких» сообщений появляются почти повсеместно. При определении того, что есть «большое» и «маленькое», обычно господствует голая эмпирика (показательна в этом смысле работа [1]). Ясно, что проблема это достаточно общая, и необходимо вырабатывать общие подходы к её решению. В данной работе указанные проблемы рассматриваются на примере коллективных операций в библиотеке MPICH-2 (http://www-unix.mcs.anl.gov/mpi/mpich2/).

Алгоритмы коллективных операций в MPICH-2

Библиотека MPICH-2 использует несколько алгоритмов для каждой коллективной операции. Например, в MPI_Bcast(…) есть три способа разослать «всем» сообщение. Это бинарное дерево, рассылка частей сообщения и последующий сбор, а так же кольцевая рассылка (подробнее см. [1]). При этом в исходный код MPI_Bcast жестко зашит механизм выбора алгоритма для рассылки: для размера сообщения менее 12К всегда используется бинарное дерево (независимо от размера коммуникатора), для размера коммуникатора более 8 и объема данных менее 512К используется рассылка частей и последующий сбор, а для данных больше 512К – кольцевая рассылка.

Естественно, раз и навсегда запрограммированная схема никогда не покажет наилучшей производительности. Поэтому было принято решение добавить в библиотеку MPICH-2 механизмы выбора наилучшего алгоритма для каждого вызова коллективной операции.

Технология измерения производительности

Первый шаг – изменение исходного кода MPI-библиотеки, здесь и далее речь идет именно о MPICH-2, разбирается пример MPI_Bcast, хотя схема работает для любой коллективной операции. В библиотеку добавляется измененная коллективная операция, с тем же кодом, что и измеряемая, но с одним дополнительным параметром – номером алгоритма для пересылки.

int MPI_Bcast(void, int, MPI_Datatype, int, MPI_Comm);*
int MPI_Bcast_bnchmrk(void, int, MPI_Datatype, int, MPI_Comm, int alg).*

Далее, создание параллельного приложения выбора лучшего алгоритма. Идея следующая: как и любая MPI-программа, приложение собирается с библиотекой MPICH-2 (уже измененной), и запускается на всем кластере. Размер MPI_COMM_WORLD в этом случае будет количество доступных узлов.

Далее, в приложении перебираются все возможные размеры коммуникаторов, начиная с 2, и заканчивая MPI_Comm_size (MPI_COMM_WORLD, …) с единичным шагом. Для каждого размера коммуникатора перебираются размеры буферов пересылаемых сообще-

ний, начиная с 0 и заканчивая определенным пользователем значением (как правило, несколько мегабайт) с некоторым промежутком. Реализованы линейная и логарифмическая шкалы изменения размеров буферов. Теперь последовательно вызываются все алгоритмы из имеющихся (в MPICH-2 для MPI_Bcast их три) с выбранными размерами коммуникатора и буфера. Каждый алгоритм вызывается подряд по несколько раз, и измеряется время всех вызовов. Несколько одних и тех же вызовов делается того, что бы получить усредненное время.

В итоге, для каждого размера коммуникатора и размера буфера получается три (MPICH-2, MPI_Bcast) времени выполнения.

Для каждого размера коммуникатора после перебора всех размеров буферов вычисляются лучшие алгоритмы, а так же размер буфера, начиная с которого этот алгоритм нужно применять. Например:

0:Btree 200000:Ring 900000:ScGather, т.е., начиная с 0 байт выгоднее использовать алгоритм «бинарное дерево», с 200000 байт «кольцевую рассылку», а с 900000 – «рассылка и сбор».

Такая информация для каждого размера коммуникатора сохраняется в результирующий файл.

В вызов MPI_Init добавляются процедуры чтения сгенерированных файлов и заполнения внутренних структур данными из этого файла. Вызов MPI_Bcast никак не меняется, однако внутри устроен совсем иначе – на основе размера коммуникатора и буфера сначала из внутренних данных выбирается номер лучшего способа передачи, а затем вызывается алгоритм именно с этим номером.

Предложенная схема работает только на гомогенных кластерах по следующей причине: все узлы кластера имеют одинаковую конфигурацию, и запуск алгоритма, выбранного на этапе измерения, покажет такую же производительность на некотором подмножестве компьютеров исходной конфигурации. При этом порядок и количество узлов в файле запуска реального параллельного приложения не будут иметь никакого значения, т.к. все узлы одинаковы. Здесь же нужно отметить, что узлы должны иметь по одному процессору, поскольку, очевидно, результаты запуска, например, конфигурации, 2 узла по 2 процесса и 4 узла по одному, даст совершенно разные результаты, т.к. в этом случае латентность пересылок будет играть самую важную роль.

Можно было бы организовать подобную схему и для кластеров разнородных компьютеров, но придется изменить алгоритм перебора узлов, а именно, придется для каждого возможного размера коммуникатора строить перестановки попавших в него узлов и запускать вычисление на каждом таком варианте. Время работы такой схемы имеет почти факториальное значение от количества узлов кластера. Кроме того, если бы даже удалось провести такие эксперименты, при запуске приложе-

ний пришлось бы анализировать файл запуска с именами узлов, что тоже не ускорит выполнение параллельной программы.

Так же, нельзя перенести сгенерированные файлы на совершенно другой кластер (с отличной аппаратной конфигурацией), т.к. измеренные величины корректны только для того суперкомпьютера, на котором были получены, на других же установках можно получить даже замедление по сравнению со стандартной схемой, а вовсе не выигрыш.

Тестирование кластеров

Реализованная схема для MPI_Bcast была протестирована на 2-х кластерах: кластер Нижегородского государственного университета (12 узлов Pentium III 1GHz, 256 Mb RAM, Gigabit Ethernet, OS Win2000AS) и установка МВС-1000/32, установленная в ИММ УрО РАН (16 узлов 2xPentium 4 2GHz, OS Linux Fedora Core 1, HyperThreading включен).

Нижегородский кластер показал полное превосходство бинарного дерева, лишь для больших буферов (более 1-х Мб) лучше оказалась кольцевая рассылка.

МВС-1000/32 показывает наилучшее время для бинарного дерева при данных в среднем менее 800Кб, и кольцевую рассылку в противном случае. Алгоритм «рассылка-сбор» и на этом кластере лучшим не оказался ни разу, хотя в стандартной схеме используется часто. Видимо, это связано с очень высокой латентностью кластера, а «рассылка-сбор» использует много маленьких пересылок, тогда как другие алгоритмы выполняют по одной пересылке всего буфера целиком.

Заключение

Реализованная схема определения лучших алгоритмов коллективных операций была протестирована на двух кластерах, и в результате экспериментов новая схема операции MPI_Bcast показывает выигрыш по времени до 1,6 раза для большинства вызовов по сравнению со стандартной схемой.

Сейчас ведется подключение других коллективных операций к приложению и MPICH-2 и проведение соответствующих экспериментов. Как видно из результатов для MPI_Bcast, предложенный подход позволяет во многих случаях существенно сократить время вызовов коллективных операций.

Автор выражает благодарность Александру Коновалову и Антону Пегушину (ЗАО «Интел АО») за помощь в разработке, а также ИММ УрО РАН за предоставленные вычислительные ресурсы.

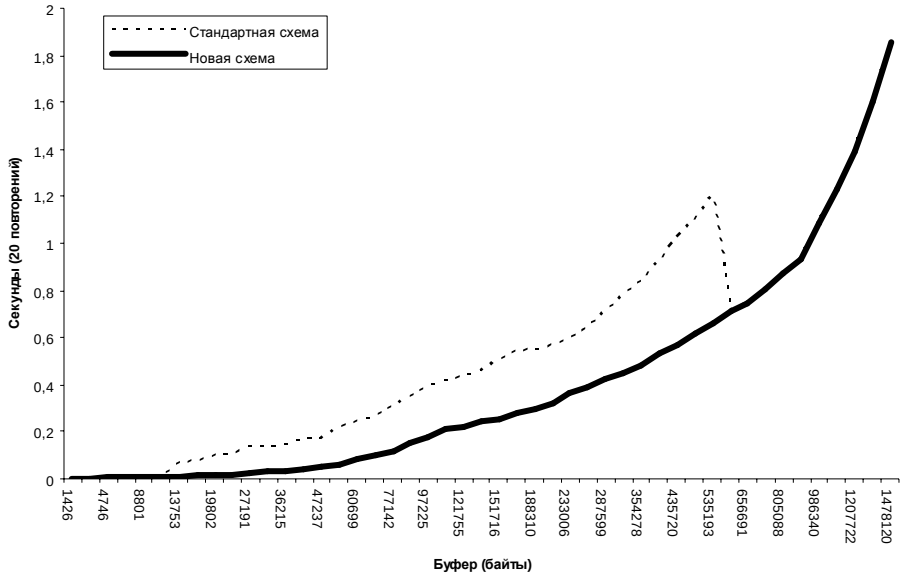


Рис. 1. Время работы MPI_Bcast на 10 узлах кластера ИММ УрО РАН

Литература

1. *Thakur R., Gropp W.* Improving the Performance of Collective Operations in MPICH.

2. *Лацис А.* Как построить и использовать суперкомпьютер // М.: Бестселлер, 2003.

ЭФФЕКТИВНОСТЬ РАСПАРАЛЛЕЛИВАНИЯ ХАРАКТЕРИСТИЧЕСКИХ АЛГОРИТМОВ ГЛОБАЛЬНОЙ ОПТИМИЗАЦИИ В МНОГОШАГОВОЙ СХЕМЕ РЕДУКЦИИ РАЗМЕРНОСТИ

В.А. Гришагин, Я.Д. Сергеев

Нижегородский государственный университет им. Н.И. Лобачевского, г. Нижний Новгород

Задачи многоэкстремальной многомерной оптимизации, в которых требуется найти глобальный оптимум функции многих переменных, имеющей несколько локальных экстремумов, обладают высокой вычислительной сложностью и требуют разработки новых подходов к их эффективному решению. Одним из современных подходов такого плана в настоящее время является подход, основанный на редукции размерности решаемой многомерной задачи и применении параллельных методов оптимизации для решения редуцированных задач [1, 2].

Идея редукции размерности состоит в сведе́нии исходной многомерной задачи к решению одной или нескольких одномерных задач. Ре-

ально разработанными и обоснованными являются две схемы такой редукции, а именно, редукция на основе кривых Пеано [1, 4] и многошаговая схема вложенной оптимизации [3, 4], позволяющая заменить решение исходной многомерной задачи решением семейства рекурсивно связанных одномерных подзадач.

Именно многошаговая схема редукции в сочетании с характеристическими параллельными алгоритмами оптимизации [5] является предметом интереса в данном исследовании.

Рассмотрим задачу поиска глобального минимума функции N переменных $f(x)$ в многомерном гиперпараллелепипеде

$$D = \{x \in R^N : a_i \le x_i \le b_i, 1 \le i \le N\}, \quad (1)$$

предполагая, что целевая функция $f(x)$ является многоэкстремальной и удовлетворяет в D условию Липшица с конечной константой $0 < L < \infty$. Данную задачу будем записывать в символической форме

$$f(x) \Rightarrow \min,\ x \in D \subset R^N. \quad (2)$$

Схема вложенной оптимизации основана на известном соотношении (см., например, работы [3, 4])

$$\min_{x \in D} f(x) = \min_{a_1 \le x_1 \le b_1} \min_{a_2 \le x_2 \le b_2} \mathbf{h} \min_{a_N \le x_N \le b_N} f(x) \quad (3)$$

Введем семейство функций, определенных рекурсивно, как

$$f_N(x_1, x_2, \mathbf{h}, x_N) \equiv f(x_1, x_2, \mathbf{h}, x_N) \quad (4)$$

$$f_i(x_1, x_2, \mathbf{h}, x_i) = \min_{a_{i+1} \le x_{i+1} \le b_{i+1}} f(x_1, x_2, \mathbf{h}, x_{i+1}) \quad (5)$$

Тогда, в соответствии с (3), для того, чтобы найти решение задачи (2), достаточно решить одномерную задачу

$$f_1(x_1) \Rightarrow \min,\ x_1 \in [a_1, b_1] \quad (6)$$

В этой задаче, однако, для вычисления значения $f(x_1)$ в некоторой фиксированной точке x_1 требуется решить одномерную задачу

$$f_2(x_1, x_2) \Rightarrow \min,\ x_2 \in [a_2, b_2] \quad (7)$$

т.е. задачу минимизации функции $f_2(x_1, x_2)$ по координате x_2 при фиксированном x_1.

Продолжая этот процесс, мы на последнем уровне придем к минимизации функции $f_N(x) = f(x)$ по переменной x_N при фиксированных значениях предшествующих координат.

Таким образом, для решения многомерной задачи (2) нам необходимо решить семейство «вложенных» одномерных подзадач

$$f_i(x_1, \mathbf{h}, x_i) \Rightarrow \min,\ x_i \in [a_i, b_i] \tag{8}$$

для $1 \le i \le N$, где номер i обозначает уровень рекурсии в (5) или просто «уровень оптимизации». Предполагается, что первый уровень (6) является наивысшим.

Включение параллельных одномерных методов в многошаговую схему для решения задач (8) позволяет распараллелить решение многомерной задачи (2). Действительно, параллельный метод, решая, например, задачу (6), вычисляет значения функции $f_1(x_1)$ в нескольких точках одновременно и, следовательно, решает параллельно несколько задач (7). Подобные рассуждения справедливы и в общем случае (8).

Для описания процесса параллелизма многошаговой схемы введем вектор степеней распараллеливания

$$\pi = (\pi_1, \pi_2, \mathbf{h}, \pi_N), \tag{9}$$

где $\pi_i, 1 \le i \le N,$ обозначает число параллельно решаемых одномерных подзадач оптимизации $(i+1)$-го уровня рекурсии, возникающих в результате выполнения параллельных итераций на i-м уровне рекурсии. Для координаты x_N число π_N означает количество параллельных испытаний в процессе минимизации функции $f_N(x_1, \mathbf{h}, x_N) = f(x_1, \mathbf{h}, x_N)$ по x_N при фиксированных значениях $x_1, \mathbf{h}, x_{N-1}$, т.е. количество параллельно вычисляемых значений целевой функции $f(x)$.

В общем случае величины $\pi_i, 1 \le i \le N,$ могут зависеть от различных параметров, например, при $i \ge 2$ - от координат $x_1, \mathbf{h}, x_i$, но мы ограничимся случаем, когда все компоненты вектора (9) постоянны.

Применение одномерного параллельного алгоритма в сочетании со схемой (6) - (8) и вектором (9) позволяет использовать для решения задачи (2) вплоть до

$$\Pi = \prod_{i=1}^{N} \pi_i \tag{10}$$

параллельно работающих процессоров. При этом многошаговая схема генерирует до Π / π_N одномерных подзадач оптимизации, решаемых одновременно.

Если мы комбинируем многошаговую схему с одномерным параллельным алгоритмом, в котором условие остановки отлично от выполнения фиксированного одинакового числа итераций, тогда времена выполнения испытаний (вычислений минимизируемой функции) в параллельной итерации на i-м уровне рекурсии $(1 \le i \le N-1)$ будут неизбежно различны, даже если время расчета целевой функция $f(x)$ не зависит от аргументов.

Напомним, что вычисление целевой функции на всех уровнях, кроме последнего, состоит в решении новой оптимизационной подзадачи.

Пусть на i-м уровне рекурсии применяется синхронная схема организации параллельной итерации одномерного алгоритма, когда итерация считается завершенной только после проведения всех π_i испытаний. В этом случае по меньшей мере $\prod_{j=i+1}^{N}\pi_i$ процессоров, которые уже завершили решение своих подзадач, будут простаивать до тех пор, пока не завершится решение остальных подзадач i-го уровня. Это означает, что для повышения производительности процесса решения в многошаговой схеме необходимо использовать асинхронные параллельные алгоритмы глобальной оптимизации, в которых вычислительные ресурсы используются сразу же по их освобождении.

В работе [5] предложен широкий класс параллельных методов одномерной глобальной оптимизации, в котором решающие правила алгоритмов обладают так называемой характеристической структурой. Данные алгоритмы могут реализовывать как синхронные, так и асинхронные механизмы распараллеливания. В ряде работ (см., например, работы [2], [6], [8]) проведено экспериментальное тестирование конкретных параллельных характеристических алгоритмов в сочетании со схемой вложенной оптимизации, в том числе на классах существенно многоэкстремальных функций нескольких переменных, включая тестовый класс [7].

Соединяя одномерные характеристические параллельные алгоритмы с многошаговой схемой редукции размерности, мы конструируем новый класс параллельных алгоритмов многомерной оптимизации. Для данного класса проведено обобщение теоретических результатов работы [5], справедливых для одномерной оптимизации, на многомерный случай. Получены оценки эффективности многошаговых характеристических алгоритмов, связанные с понятиями безызбыточности распараллеливания и ускорения поиска. В частности, установлены условия, при которых многомерные характеристические алгоритмы являются безызбыточными и обеспечивают ускорение поиска при решении задачи (2) до 2^N раз. Полученные ранее экспериментальные результаты показали хорошее согласование с теоретическими оценками.

Работа выполнена при поддержке РФФИ (грант № 04-01-00455-а).

Литература

1. *Strongin R.G., Sergeyev Ya.D.* Global optimization with non-convex constraints: Sequential and parallel algorithms // Kluwer Academic Publishers, Dordrecht, Netherlands. 2000.

2. *Sergeyev Ya.D., Grishagin V.A.* Parallel asynchronous global search and the nested optimization scheme // Journal of Computational Analysis & Applications, 3(2), 2001. P. 123–145.

3. *Carr C.R., Howe C.W.* Quantitative Decision Procedures in Management and Economics // Deterministic Theory and Applications. McGraw Hill, New York, 1964.

4. *Стронгин Р.Г.* Численные методы в многоэкстремальных задача // Информационно- статистический подход. М.: Наука, 1978.

5. *Strongin R.G., Sergeyev Ya.D., Grishagin V.A.* Parallel Characteristical Algorithms for Solving Problems of Global Optimization // Journal of Global Optimization, 10, 1997. P. 185–206.

6. *Гришагин В.А., Филатов А.А.* Параллельные рекурсивные алгоритмы многоэкстремальной оптимизации // Материалы второго Международного научно-практического семинара «Высокопроизводительные параллельные вычисления на кластерных системах» / Под ред. проф. Р.Г. Стронгина. Н. Новгород: Изд-во Нижегородского госуниверситета, 2002. С. 88–90.

7. *Gaviano M., Kvasov D.E., Lera D., Sergeyev Ya.D.* Software for Generation of Classes of Test Functions with Known Local and Global Minima for Global Optimization // (Принято к опубликованию в ACM Transactions on Mathematical Software).

8. *Гришагин В.А., Песков В.В.* Повышение эффективности параллельных рекурсивных схем редукции размерности // Высокопроизводительные параллельные вычисления на кластерных системах: Материалы Международного научно-практического семинара / Н. Новгород: Изд-во ННГУ. 2002. С. 56–60.

ПАРАЛЛЕЛЬНОЕ МОДЕЛИРОВАНИЕ ЛИНЕЙНЫХ ДИНАМИЧЕСКИХ СИСТЕМ С АППРОКСИМАЦИЕЙ ПРАВОЙ ЧАСТИ

О.А. Дмитриева

Донецкий национальный технический университет, г. Донецк

Данная статья является продолжением работ [1-3], которые посвящены проблемам создания и исследования параллельных алгоритмов численного решения систем ОДУ, используемых для моделирования сложных динамических систем с сосредоточенными параметрами. Предлагаемые алгоритмы ориентированы на использование в многопроцессорных вычислительных системах SIMD (*single instruction stream - multiple data stream*) структуры с решеткой или линейкой процессорных элементов. Набор процессоров известен до начала вычислений и не меняется в процессе счета, при этом каждый процессорный элемент может выполнить любую арифметическую операцию за один такт, временные затраты, связанные с обращением к запоминающему устройству, отсутствуют.

Пусть математическую модель динамической системы можно представить в виде системы ОДУ с постоянными коэффициентами и начальными условиями:

$$\frac{d\bar{x}}{dt} = A\bar{x} + \bar{f}(t), \quad \bar{x}(t_0) = \bar{x}^0 = (x_1^0, x_2^0, ..., x_m^0)^t, \tag{1}$$

где $\bar{x}$ – вектор неизвестных сигналов, $\bar{f}(t)$ – вектор воздействий, $t \in [0,T]$, A – матрица коэффициентов системы.

В этом случае решение можно получить последовательно по шагам с помощью численных методов заданного порядка точности.

Здесь вычисление значения вектора неизвестных $\bar{x}^{n+1}$ на очередном шаге требует предварительного определения значений $\bar{x}^n$. В [1] рассмотрены вопросы, связанные с возможностью параллельной реализации таких алгоритмов. В частности, если система (1) является однородной, т.е. $f_i(t)=0$, $i=1,2,...m$, тогда, в зависимости от выбранного метода интегрирования, можно искать решение в виде:

$$\bar{\mathbf{x}}^{n+1} = G\bar{\mathbf{x}}^n, \tag{2}$$

где G- оператор (матрица) переходов.

Полученный оператор перехода G, который необходимо определить один раз до начала вычислений, позволяет вычислять значения вектора неизвестных параллельно [2, 3]. Для методов Рунге-Кутты, например, этот оператор может быть представлен, в зависимости от точности метода, как

$$G = E + \tau A(E + \frac{\tau A}{2}(E + \frac{\tau A}{3}(E + \frac{\tau A}{4}))), \tag{3}$$

который обеспечивает точность 4-го порядка, или, например, как

$$G = E + \tau A(E + \frac{\tau A}{2}(E + \frac{\tau A}{3}(E + \frac{\tau A}{4}(E - \frac{\tau A}{5}(E - \frac{\tau A}{4}))))), \tag{4}$$

точность которого оценивается 6-м порядком.

При решении неоднородной системы необходимо дополнительно вычислить на каждом шаге значения всех функций $f_i(t)$, $i=\overline{1,m}$ в нескольких промежуточных точках. Поскольку все эти функции могут быть различными, одновременное вычисление их на SIMD компьютере невозможно.

В связи с этим можно предложить два основных подхода, первый из которых состоит в том, что все промежуточные значения функций $\bar{f}_i(t)$ могут быть вычислены заранее. При этом если количество промежуточных точек метода определяется как r, то можно оценить количество тактов рас-

чета на топологических структурах линейке из m процессоров и решетке из $m \times m$ процессоров (размерность процессорного поля выбрана совпадающей с размерностью системы уравнений исключительно для удобства изложения). Определим для этого трудоемкость реализации правых частей системы, как Θ_{f_i}, и при расчете будем оперировать с максимальным значением, которое обозначим, как $\Theta_f = \max_i \{\Theta_{f_i}\}$. Если расчет осуществляется для общего количества узлов, равного N, то общее число тактов работы на линейке процессоров составит для одного уравнения ближайшее целое сверху соотношения $\frac{r * \Theta_f * N}{m}$ или $\left[\frac{r * \Theta_f * N}{m}\right] + 1$. Тогда вся система может быть рассчитана за $r * \Theta_f * N + m$ тактов. На решетке процессоров одно уравнение будет решаться за ближайшее целое сверху к $\frac{r * \Theta_f * N}{m^2}$ тактов, а время, которое потребуется для расчета всей системы составит $\left[\frac{r * \Theta_f * N}{m}\right] + 1$. По каждому уравнению системы придется хранить двумерные массивы размерностью $r \times N$ и использовать коэффициенты с нужными индексами при расчете.

Второй подход, который позволит избежать последовательных участков при параллельной реализации системы, основывается на предварительном интерполировании правых частей (1). В вычислительной практике с таким подходом сталкиваются, если приходиться заменять одну функцию $f(t)$ (известную, неизвестную или частично известную) некоторой функцией $\varphi(t)$, близкой к $f(t)$ и обладающей определенными свойствами, позволяющими производить над нею те или иные аналитические или вычислительные операции. Такую замену называют *приближением* функции $f(t)$. Тогда при решении задачи вместо функции $f(t)$ оперируют с функцией $\varphi(t)$, а задача построения функции $\varphi(t)$ называется задачей приближения. Исходя из проблематики задачи, т.е. принимая во внимание большое количество узлов, которые будут участвовать в расчете, и, задаваясь требуемой точностью приближения функций, можно утверждать, что наиболее перспективным является случай, когда используются кусочно-полиномиальная аппроксимация, или сплайны, так как при этом интерполяционный многочлен строится не на весь интервал решения задачи, а на подынтервалах, что позволит избежать накопления ошибок приближения.

Основная идея такого подхода заключается в следующем: исходный отрезок решения для (1) $[0,T]$ разбивается на несколько подынтервалов V с шагом, определяющимся из соотношения точности методов численного интерполирования и интегрирования, а затем на каждом таком интервале строится интерполяционный многочлен. Поскольку в ка-

честве интерполяционной функции обычно выбирают многочлены степени не выше 3-4-й, что соответственным образом влияет на точность интерполяции, то необходимо предварительно согласовать порядки точности методов численного интегрирования и предварительного интерполирования.

Если порядок метода численного интегрирования (1) определяется, как $O(\tau^{\nu})$, а порядок сплайна, как $O(h^4)$, то между шагами двух решающихся задач должно выполняться соотношение $\tau^{\nu} = h^4$. Если порядок точности метода интегрирования $\nu=4$ или более, т.е. между количеством узлов задач интерполирования и интегрирования выполняется соотношение $V \geq N$, то использование интерполирования для восстановления значений правых частей является нерациональным. Проще заранее вычислить значения правых частей на промежутке $[0,T]$. Если же речь идет о методах интегрирования, которые имеют порядок погрешности ниже 4, то тогда $V<N$ и при этом значения V и N можно связать с помощью некоторого коэффициента β, т.е.

$$N = \beta * V, \text{ где } \beta >> 1. \quad (5)$$

При этом желательно выбирать множитель β целым, т.к. предпочтительнее, чтобы на одном такте расчета использовались коэффициенты одного интервала сплайна, что значительно упростит алгоритм вычисления и выбор нужного интервала по заданному аргументу.

Если исходить из (5), то узлов интерполирования будет в β раз меньше, чем узлов интегрирования. К тому же оценку погрешности для сплайна порядка $O(h^4)$ можно считать завышенной. Тогда исходная задача может быть сведена к двум подзадачам, каждая из которых легко распараллеливается.

Первая подзадача будет заключаться в нахождении коэффициентов сплайна. Замену исходных функций $f_i(t)$ в (1) на сплайн-функции будем осуществлять в виде:

$$a_{il} + b_{il}t + c_{il}t^2 + d_{il}t^3 + e_{il}t^4, \; i = \overline{1,m}, l = \overline{1,V} \quad (6)$$

Тогда, во время реализации второй подзадачи, вместо разнородных операций (которые на SIMD-структурах выполняются последовательно), все правые части системы (1) будут считаться параллельно по одним и тем же аргументам t, но с разными коэффициентами сплайн-функций $a_{il}, b_{il}, c_{il}, d_{il}, e_{il}, \; i = \overline{1,m}, l = \overline{1,V}$. Для нахождения неизвестных коэффициентов по каждому уравнению системы (1) придется решать систему линейных алгебраических уравнений размерностью $4V \times 4V$. Формирование системы линейных уравнений будет исходить из принципов совпадения

значений функции, ее первых, вторых и третьих производных на соседних подынтервалах слева и справа от узла интерполяции.

Пусть $\{ p_l \}$ – множество узлов интерполяции, $l = \overline{0,V}$. Причем, для любой правой части системы (1) это множество узлов будет постоянным.

Между узлами p_{l-1} и p_l будем представлять функцию i-го уравнения системы в виде:

$$S_i(p) = a_{il} + b_{il}(p - p_{l-1}) + c_{il}(p - p_{l-1})^2 + d_{il}(p - p_{l-1})^3 + e_{il}(p - p_{l-1})^4 \quad (7)$$

$$p_{l-1} \le p \le p_l, \quad i = \overline{1,m}, l = \overline{1,V}$$

Исходя из постановки задачи интерполирования, в узлах интерполяции значения исходной функции и интерполяционного многочлена должны совпадать, т.е.:

$$S_i(p_l) = f_{il} \quad i = \overline{1,m}, l = \overline{0,V} \quad (8)$$

С другой стороны, если аргумент сплайна совпадает с узлом интерполяции, тогда из (7) и (8)

$$S_i(p_{l-1}) = a_{il} = f_{il-1}, \quad i = \overline{1,m}, l = \overline{1,V}. \quad (9)$$

Поскольку интерполяционный многочлен строится для равностоящих узлов, т.е.:

$$p_l - p_{l-1} = h, \quad (10)$$

то, используя полученные соотношения, можно переписать формулу для сплайна в l–ом узле в следующем виде:

$$f_{il} = f_{il-1} + b_{il}h + c_{il}h^2 + d_{il}h^3 + e_{il}h^4. \quad (11)$$

Для построения оставшихся соотношений воспользуемся соглашением о совпадении 1-ой, 2-ой и 3-ей производных справа и слева от узлов интерполяции, т.е. для первых производных

$$\begin{aligned} &S_i'(p) = b_{il} + 2c_{il}(p - p_{l-1}) + 3d_{il}(p - p_{l-1})^2 + 4e_{il}(p - p_{l-1})^3 \\ &p_{l-1} \le p \le p_l \\ &S_i'(p) = b_{il+1} + 2c_{il+1}(p - p_l) + 3d_{il+1}(p - p_l)^2 + 4e_{il}(p - p_l)^3 \\ &p_l \le p \le p_{l+1}. \end{aligned} \quad (12)$$

Для каждого уравнения системы (1) будем приравнивать (12) в точке p_l, тогда:

$$b_{il+1} = b_{il} + 2c_{il}h + 3d_{il}h^2 + 4e_{il}h^3 \quad (13)$$

Аналогично приравнивая значения полученных выражений для 2-ой и 3-ей производных в узлах интерполяции и добавив граничные условия, получаем m систем линейных уравнений.

Сформированные системы относительно вектора неизвестных коэффициентов можно представить в общем виде, как:

$$Q_i \bar{y}_i = \bar{g}_i, \quad i = \overline{1,m}. \tag{14}$$

Векторы неизвестных будут иметь вид:

$$\bar{y}_i = (a_{i1}, b_{i1}, c_{i1}, d_{i1}, e_{i1}, \ldots, a_{iV}, b_{iV}, c_{iV}, d_{iV}, e_{iV}), \quad i = \overline{1,m}. \tag{15}$$

Особенностью полученных систем является ленточный вид матрицы Q, т.к. каждое уравнение системы (за исключением первого и последнего) будет содержать только 4 неизвестных. В этом случае систему можно преобразовать так, чтобы ее можно было решать методом встречной прогонки [4]. Трудоемкость решения таких систем на параллельных SIMD структурах линейно зависит от размерности решаемой системы и для системы размерностью k оценивается, как $O(k)$. Тогда для нашего случая трудоемкость нахождения коэффициентов сплайн-функции для одного уравнения системы будет оцениваться на уровне $O(4V)$. А для исходной задачи (1) число операций приближенно будет оцениваться на уровне $4*V*m$.

Еще один возможный подход к решению полученных систем (14), которые имеют большую размерность и разреженную матрицу коэффициентов, заключается в приведении ее к блочно-диагональной форме с обрамлением [5, 6] и формировании вспомогательной системы значительно меньшей размерности, которая определит вектор определяющих величин, или переменных связи. Трудоемкость реализации такого подхода на параллельных вычислительных структурах будет, как и в предыдущем случае, линейно зависеть от размерности системы.

Кроме того, для интерполирования необходимо предварительно вычислить значения правых частей в V точках, которые будут использоваться в качестве исходных данных для построения интерполяционного многочлена, тогда для системы из m уравнений потребуется $m*(4V + V*\Theta_f)$ тактов. Также возникает необходимость в восстановлении значений правых частей по полученным коэффициентам интерполяционных многочленов в N основных и r вспомогательных узлах интегрирования. Последовательность выполняемых операций при этом составит для многочлена общего вида $a + bt + ct^2 + dt^3 + et^4$ на SIMD-структуре:

умножения

1-й такт – $t*t$

2-й такт – t^2*t, t^2*t^2 – параллельно

3-й такт – $b*t, c*t^2, d*t^3, e*t^4$ – параллельно

сложения

4-й такт $-a+b*t, c*t^2+d*t^3$

5-й такт $-\ a+b*t+c*t^2+d*t^3+e*t^4$

Таким образом, на определение одного значения правой части необходимо 5 временных тактов. Если учесть, что число точек, в которых необходимо восстанавливать значение правой части каждого уравнения определяется как *r*N*, то всего на восстановление значений функции по интерполяционному многочлену для системы потребуется 5**m*r*N* временных тактов.

Сведем полученные приближенные результаты для 2-х описанных способов реализации правых частей в следующую таблицу.

Таблица 1

	Предварительный расчет	Интерполирование
Общее число операций	$r*\Theta_f*N*m$	$m*(4V+V*\Theta_f+5*r*N)$
Число операций на линейке процессоров	$\lfloor r*\Theta_f*N \rfloor+1$	$4V+V*\Theta_f+5*r*N$
Число операций на решетке процессоров	$\left\lceil \frac{r*\Theta_f*N}{m} \right\rceil+1$	$(4V+V*\Theta_f+5*r*N)/m$

Поскольку изначально предполагалось, что трудоемкости вычисления правых частей *Θf* являются высокими [7], то оценка трудоемкости всего алгоритма может осуществляться относительно этих значений. Тогда очевидно, что в случае выполнения соотношения (5), предпочтительнее интерполировать правые части, хотя этот подход и сопряжен с алгоритмическими сложностями, но имеет безусловные преимущества.

Таким образом, в представляемой работе исследованы возможности распараллеливания известных последовательных алгоритмов численного решения систем обыкновенных дифференциальных уравнений и переноса их на параллельные вычислительные структуры с целью получения максимальной реальной производительности. Предложены подходы, позволяющие избегать последовательных участков работы многопроцессорных вычислительных SIMD систем, один из этих подходов основан на предварительном вычислении правых частей неоднородной линейной системы ОДУ, который предпочтительнее использовать, если точность метода интегрирования, с помощью которого решается задача, высока. Разработан также подход, исключающий последовательные вычисления, который основывается на предварительном интерполировании правых частей системы (1) с помощью сплайнов. Выявлены соотношения между порядками погрешностей методов численного интегрирования системы (1) и метода-

ми интерполирования, которые позволяют определить оптимальный выбор метода параллельной реализации правых частей.

Литература

1. *Дмитриева О.А*. Анализ параллельных алгоритмов численного решения систем обыкновенных дифференциальных уравнений методами Адамса-Башфорта и Адамса-Моултона // Математическое моделирование, 2000. Том 12. № 5. С. 81-86.

2. *Фельдман Л.П., Дмитриева О.А*. Эффективные методы распараллеливания численного решения задачи Коши для обыкновенных дифференциальных уравнений // Математическое моделирование, 2001. Том 13. № 7. С. 66-72.

3. *Дмитриева О.А*. Параллельное моделирование динамических объектов с сосредоточенными параметрами // Тезисы докладов XII Юбилейной международной конференции по вычислительной механике и современным прикладным программным средствам. М.:МГИУ, 2003.

4. *Гаранжа В.А., Коньшин В.Н.* Прикладные аспекты параллельных высокоточных алгоритмов решения задач вычислительной гидродинамики // Тезисы докладов Всероссийской научной конференции «Фундаментальные и прикладные аспекты разработки больших распределенных программных комплексов». М.: Изд-во МГУ. 1998. С. 34-38.

5. *Джорт А., Лю Д.* Численное решение больших разреженных систем уравнений // М.: Мир, 1984. – 333 с.

6. *Дмитриева О.А*. Анализ параллельных алгоритмов численного решения систем линейных уравнений итерационными методами // Научн. тр. ДонГТУ. Серия: Проблемы моделирования и автоматизации проектирования динамических систем, Донецк. 2000. Вып. 10. С. 15-22.

7. *Feldman L., Dmitrieva O., Gerber S.* Abbildung der blockartigen Algoritmen auf die Parallelrechnerarchitekture // 16 Symposium Simulationstechnik ASIM 2002, Rostock, 10.09 bis 13.09 2002. Erlangen: Gruner Druck, 2002. P. 359-364.

РАСПАРАЛЛЕЛИВАНИЕ ПРОГРАММ ДЛЯ МНОГОПРОЦЕССОРНЫХ ВЫЧИСЛИТЕЛЬНЫХ СИСТЕМ С ПОМОЩЬЮ ЭКСПЕРИМЕНТАЛЬНОЙ МНОГОЦЕЛЕВОЙ СИСТЕМЫ ТРАНСФОРМАЦИЙ ПРОГРАММ

О.А. Жегуло

Ростовский государственный университет, г. Ростов-на-Дону

Введение

Разработка систем эффективного распараллеливания для многопроцессорных суперкомпьютеров с распределенной памятью является предметом активных исследований, хотя для векторных и некоторых других архитектур построены приемлемые решения [1].

Один из способов распараллеливания программ – систематическое выполнение трансформаций программ, представленных в виде схемных правил трансформации [2] некоторого расширенного вида [3]; применение этих правил к распараллеливаемой программе должно обеспечиваться разрабатываемой экспериментальной многоцелевой системой трансформаций программ (МСТП) [3]. Выгода от представления распараллеливающих преобразований программ в виде схемных правил заключается в упрощении модификации и добавления преобразований.

В настоящей работе рассматривается экспериментальная трансформационная система распараллеливания на основе макетной реализации МСТП на языке Пролог и набора правил распараллеливания [4]. Система предназначена для автоматического, полуавтоматического и ручного распараллеливания.

1. Схемные правила трансформации

В общем случае [2] схемное правило представляет собой пару из входного и выходного образцов (схем заменяемой и заменяющей программных конструкций), снабженных логическим условием применимости. Схема получается путем параметризации программы или программной конструкции по вложенным компонентам.

Применение правила заключается в поиске некоторой конструкции программы, сопоставимой с входным образцом, при этом параметры входного образца получают значения; затем, если условие применимости удовлетворено, вместо найденного фрагмента вставляется конструкция, полученная из выходного образца подстановкой параметров.

2. Схемные правила трансформации для распараллеливания

Расширенный вид правил трансформации [3] допускает использование многокомпонентных образцов, которые позволяют записывать нелокальные преобразования.

В случае необходимости допускается использование не схемных правил, а процедурных трансформаций на языке реализации трансформационной системы, в данном макете – на Прологе.

Параметр образца задается своим именем; также должен быть указан его синтаксический тип.

В состав простого образца, кроме параметризованного текста программы, могут входить идентифицирующие выражения [3]. С помощью идентифицирующего выражения можно кратко сослаться на сопоставленную с ним синтаксическую конструкцию, задать ее синтаксический тип или указать ее положение относительно других синтаксических конструкций.

Схемное правило может содержать вызовы оператора языка задания правил apply применения заданного правила к множеству программных конструкций.

Условия применимости правил представляют собой логические выражения с использованием предикатов, зависящих от входных параметров правила, переменных среды МСТП, логических переменных, квантора всеобщности all, а также присваивания выходным параметрам правила значений. В основном в условиях применимости участвуют базовые предикаты и функции [4]; большинство этих предикатов и функций задано на графовых и других вспомогательных представлениях программы. Пользователь может также определять собственные служебные предикаты на языке реализации макета МСТП, Прологе.

3. Общая структура макета МСТП

Макет МСТП состоит из следующих подсистем.

1) Синтаксический анализатор входного языка и языка схемных правил со сканером, и соответствующая грамматика.

Шаблоны правил транслируются в семантические деревья, полностью совместимые с деревом программы. Параметры правил могут занимать место любого поддерева или деревьев нескольких последовательно идущих операторов. Логическое условие применимости транслируется в предикат на Прологе.

Пока в качестве входного языка возможен только Фортран.

2) Административный модуль доступа к абстрактным семантическим деревьям программы и образцов правил, обеспечивающий базовые операции с деревьями и скрывающий реализующую их структуру данных.

3) База правил трансформации.

Таблица имен правил, их исходные тексты и совокупности семантических деревьев входного и выходного шаблона и предиката условия применимости.

Набор имеющихся схемных правил включает традиционные преобразования нормализации, оптимизации и распараллеливания [4].

4) Синтаксический анализатор со сканером языка сценариев и соответствующая грамматика.

Макет МСТП может функционировать в автоматическом режиме: выполнение трансформаций происходит согласно сценарию применения правил [3]. В интерактивном режиме пользователю предлагаются возможные варианты преобразований и по его команде применяются указанные им правила.

Язык сценариев является подмножеством языка С и включает операторы циклов, условные операторы, описания переменных и операторы применения правил трансформации. Оператор apply_once означает применение правила к первой сопоставимой конструкции внутри программы или программной конструкции-аргумента, apply – применение правила до исчерпания возможности, apply_set – асинхронное применение независимого набора правил [5].

Сценарий транслируется в предикат на Прологе с тем же именем, что и сценарий.

5) Хранилище сценариев.

К каждому сценарию прилагается список вызываемых правил.

6) Переменные среды системы.

Определяют среду выполнения программы (целевую архитектуру, класс задач и другие требования к выходной программе).

7) Синтаксический анализатор со сканером и грамматика для фильтров правил и сценариев их применения.

Для гибкой настройки на внешние условия набор преобразований и сценариев их применения может быть сужен с помощью логических *условий фильтрации правил трансформации* [6].

Фильтры содержатся в базе Пролога в виде ограничений (constraints).

8) Машина вывода, формирующая множество допустимых правил и сценариев их применения.

9) Модуль вычисления зависимостей по данным.

10) Модуль со встроенными предикатами

11) Трансформационная машина, реализующая процесс поиска сопоставимых фрагментов и замены в семантических деревьях программы.

12) Генератор текста на выходном языке по абстрактному дереву программы.

13) Модуль для диалогового применения трансформаций.

Демонстрирует допускающие преобразование фрагменты программы и принимает от пользователя инструкции по запуску трансформаций для указанных им синтаксических конструкций.

Пользователь вызывает функции системы из оболочки.

В качестве редактора исходных текстов правил, сценариев их применения и фильтров правил и сценариев предполагается использование внешнего простейшего текстового редактора.

4. Особенности работы трансформационной машины

Для унификации данных различных типов (семантических деревьев, арифметических выражений и отдельных вхождений переменных) система использует нестандартные алгоритмы, учитывающие специфику данных и обладающие большей функциональностью и эффективностью, чем встроенный в систему универсальный алгоритм унификации.

5. Пример работы экспериментальной системы распараллеливания

Будем считать, что построены внутренние представлений правил и программы, и выбран набор классических преобразований распараллеливания и оптимизации в Fortran77 с параллельными конструкциями. В данном примере рассмотрим автоматическое преобразование примера по следующему сценарию:

```
void classic () {
int i;
apply normalize; /* нормализация (стандартизация) циклов */
apply induct; /* удаление индуктивных переменных */
apply wraparound; /* удаление охватывающих переменных */
cr_dep; /* построение графа зависимостей по данным */
apply dead_elimination /* удаление мертвых операторов */
apply_set(loop_fusion, loop_distribution); /* слияние и разбиение циклов*/
apply interchange; /* перестановка циклов */
apply unfold; /* развертка циклов*/
for (i=1; i< MaxNest; i++) apply parallelize(M); /* распараллеливание циклов разной вложенности, начиная с самых внешних и кончая самым глубоким уровнем вложенности в программе */ }.
```

Преобразования на самом деле выполняются над внутренним представлением, но здесь мы продемонстрируем текст, сгенерированный по дереву программы. Первый этап, нормализация цикла, нужна для приведения цикла к стандартному виду, чтобы его счетчик изменялся от 1 до целой верхней границы.

```
jj=0                                    jj=0
do 10 j=1,10,2                          do 10 j_new_1=1,5
jj= jj + 1                              jj= jj + 1
ii=0                                    ii=0
do 10 i=1,20,2                          do 10 i_new_5=1,10
ii= ii + 1                              ii= ii + 1
xx(ii,jj,1)= x(ii,j,1)                  xx(ii,jj,1)= x(ii,2 * j_new_1 - 1,1)
```

```
 xx(ii,jj,2)= x(ii,j,2)                    xx(ii,jj,2)= x(ii,2 * j_new_1 - 1,2)
10 continue                               10 continue
```

Теперь избавимся от индуктивных переменных [4]. Для нахождения зависимостей по данным индексные выражения массивов должны зависеть только от счетчиков циклов. Рассмотрим подробнее процесс преобразования первого цикла: текст правила преобразования, найденные значения его параметров и результат.

```
rule induct1:
var <sts> sts1, sts2, <var> i, id,
<num_expr> Ub, Inc_expr;                  $id = jj
<loop: L> ( do $i=1 to $Ub
 $sts1                                    $i=j_new_1
<assign: S2> $id=$inc_expr                $sts1=[]
 $sts2
 continue )|                              L
<before(L): S1> ($id =$expr1)
&& full_reachable_c(S1, loop) & lin-
ear(inc_expr, $id, 1, $b) & un-           $b=1
changed_between($id, S1, S2) &
no_jumps(L) & $id_v=$b*($Ub)+             =1*5+0
($expr1)
=> ( <S1>() |
do $label $i =1 to $Ub
 apply (($id => $b*($i-1) + ($expr1)),
$sts1)
 apply (($id => $b*$i + ($expr1)),        jj =>
$sts2)                                    1*j_new_1
$label: continue )                        +0
$id = $id_v
end
```

```
 jj=0 =$expr1
 do 10 j_new_1=1,5 =$Ub
 jj= jj + 1 = $inc_expr
 ii=0
 do 10 i_new_5=1,10
 i= i_new_5 * 2 – 1 $sts2
 ii= ii + 1
 xx(ii,jj,1)= x(ii,2*j_new_1 - 1,1)
 xx(ii,jj,2)= x(ii,2*j_new_1 - 1,2)
10 continue
```

Данное преобразование применяется, пока не останется индуктивных переменных.

```
 do 10 j_new_1=1,5                        do 10 j_new_1=1,5
 ii=0                                     do 10 i_new_5=1,10
 do 10 i_new_5=1,10                       xx(i_new_5, j_new_1,1)=
 ii= ii + 1                              =x(i_new_5, 2*j_new_1 -1,1)
 xx(ii, j_new_1,1)=                       xx(i_new_5, j_new_1,2)=
=x(i_new_5, 2*j_new_1 -1,1)              =x(i_new_5, 2*j_new_1 -1,2)
 xx(ii, j_new_1,2)=                      10 continue
=x(i_new_5, 2*j_new_1 -1,2)              jj=5
10 continue                              ii=10
 jj=5
```

Теперь строим граф зависимостей по данным. Если значения ii и jj не используются, можно избавиться от лишних присваиваний. Итерации внутреннего цикла независимы и могут исполняться параллельно. Но,

поскольку нет дуг зависимостей противоположных направлений по разным счетчикам, можно выполнить перестановку внутреннего цикла наружу [4], а параллельное исполнение более длинного цикла выгоднее. В итоге получим следующий распараллеленный код:

```
doall 10 i_new_5=1,10
do 10 j_new_1=1,5
xx(i_new_5, j_new_1,1)= x(i_new_5, 2*j_new_1 -1,1)
xx(i_new_5, j_new_1,2)= x(i_new_5, 2*j_new_1 -1,2)
10 continue
```

Макетная реализация МСТП осуществляется в Пролог-системе Eclipse. Частично используется (после портирования) код модулей распараллеливающего компилятора на Прологе [7], обеспечивающих построение внутренних представлений программы.

Развитие экспериментальной системы распараллеливания предполагается в двух направлениях: во-первых, внедрение новых схематических преобразований и методик распараллеливания, во-вторых, развитие средств визуализации и диалога с пользователем макета МСТП.

Литература

1. *Евстигнеев В.А., Спрогис С.В.* Векторизация программ (обзор) // В сб. Векторизация программ: теория, методы, реализация. М., Мир, 1991. С.246-271.

2. *Partsch H., Steinbruggen R.* Program transformation systems // ACM Computer Survey, 1983. V. 15, № 3. P. 199-236.

3. *Bukatov A.A.* Building the Program Parallelization System Based on a Very Wide Spectrum Program Transformation System // Slot P.M.A. at al (Eds.) International Conference on Computer Science 2003, Proceedings, Part II: Lecture Notes on Computer Science, vol. 2658, Berlin, Heidelberg, New York, Hong Kong, London, Milan, Paris, Tokyo: Springer, 2003. P. 945-954.

4. *Жегуло О.А.* Непроцедурное представление преобразований программ в системе поддержки распараллеливания // В сб. «Компьютерное моделирование. Вычислительные технологии», Ростов-на-Дону: изд-во ООО «ЦВВР», 2003. С. 27-40.

5. *Букатов А.А., Коваль В.В.* Методы реализации трансформационной машины в многоцелевой системе преобразований программ // «Искусственный интеллект», журнал Института проблем искусственного интеллекта Национальной академии наук Украины, Донецк: Наука і освіта, 2003. №3. С. 6-14.

6. *Жегуло О.А.* Методы настройки трансформационной системы автоматического распараллеливания программ на параметры условий применения // «Искусственный интеллект», журнал Института проблем искусственного интеллекта Национальной академии наук Украины, Донецк: Наука і освіта, 2003. №3. С. 88-94.

7. *Адигеев М.Г., Дубров Д.В., Лазарева С.А., Штейнберг Б.Я.* Экспериментальный распараллеливающий компилятор на Супер-ЭВМ со структурной реализацией вычислений // Фундаментальные и прикладные аспекты разра-

ботки больших распределенных программных комплексов. Тезисы докладов всероссийской научной конференции. Новороссийск. 21-26.09.98. Москва: МГУ, 1998. С. 101-108.

О ПОДГОТОВКЕ СПЕЦИАЛИСТОВ ПО ПАРАЛЛЕЛЬНОМУ ПРОГРАММИРОВАНИЮ НА КАФЕДРЕ МАТЕМАТИЧЕСКОГО ОБЕСПЕЧЕНИЯ ВС ПГУ

Е.Б. Замятина, К.А. Осмехин

Пермский государственный университет, г. Пермь

Введение

В настоящее время роль многопроцессорных и распределенных вычислительных систем при обработке, хранении и передаче информации существенно выросла. Поэтому подготовка специалистов в области системного программного обеспечения предполагает введение дисциплин, охватывающих область параллельных вычислений, а также параллельных и распределенных архитектур вычислительных систем. На кафедре математического обеспечения вычислительных систем механико-математического факультета Пермского государственного университета читается ряд специальных курсов, посвященных этой тематике. К этим дисциплинам относятся:

- Параллельные вычисления.
- Параллельные архитектуры.
- GRID.
- Распределенные алгоритмы.
- Современные компьютерные технологии.

Содержание курсов, связанных с параллельным программированием

Первые знания, связанные с построением и организацией вычислительных систем с параллельной архитектурой студенты приобретают ещё на первом курсе обучения(1, 2 семестр), изучая курс «Информатика». В этом курсе студенты получают сведения об архитектуре ВС, в том числе сведения о ВС с параллельной архитектурой, классификацией ЭВМ по Флинну, с принципами параллельной и конвейерной обработки информации.

На втором курсе, изучая дисциплину «Системное прикладное программное обеспечение», студенты рассматривают материал, связанный с синхронизацией параллельных процессов, вопросы обнаружения, предотвращения тупиков и т.д.

Студенты приступают к изучению специального курса «Параллельное программирование» в восьмом семестре, прослушав к этому

времени такие общие и специальные курсы, как «Дискретная математика», «Теория графов», «Архитектура ЭВМ», «Языки программирования и методы трансляции», «Операционная система Unix», «Автоматизация проектирования ВС» и др.

Материал этих курсов позволяет студентам лучше усвоить новый материал, связанный с параллельным программированием.

Специальный курс «Параллельное программирование» включает три раздела:

Архитектурные принципы построения вычислительных систем с параллельной обработкой.

Математические схемы и методы анализа, отладки и тестирования параллельных и распределенных программ.

Языки и системы параллельного программирования.

Специальный курс «Параллельное программирование» начинается с изложения теоретических и прикладных проблем параллельного программирования. В материале излагаются вопросы, в которых рассматривается необходимость использования параллельного программирования и пути увеличения эффективности его использования. Кроме того, рассматриваются проблемы, препятствующие широкому использованию параллельного программирования. Это-закон Амдаля, законы Мура и Гроша, потери, связанные с передачей данных, потери, зависящие от взаимодействия процессов, зависимость эффективности параллельных вычислений от используемой аппаратуры, сложность разработки параллельных алгоритмов, проблемы отладки параллельных алгоритмов.

Вслед за этим студенты изучают принципы построения ЭВМ с параллельной архитектурой. На лекциях рассматриваются архитектуры ЭВМ с общей и распределенной памятью, различные схемы коммутации, приводятся сведения о различных классификациях многопроцессорных ВС (классификация по Флинну, Шору и т.д.), кратко излагаются вопросы организации транспьютеров, кластеров, приводятся примеры конкретных ВС. Внимание уделяется и потоковым ЭВМ. Здесь же рассматриваются различные схемы коммутации, типовые топологии (кольцо, шина, гиперкуб и т.д.), их преимущества и недостатки.

Затем студентам предлагается познакомиться с двумя основными парадигмами параллельного программирования – с параллелизмом данных и параллелизмом задач.

Далее рассматриваются такие вопросы, как декомпозиция алгоритма на параллельно исполняемы фрагменты, распределение заданий по процессорам, вопросы балансировки, синхронизации и взаимоисключения, организация взаимодействия параллельных процессов. Как уже говорилось ранее, с некоторым механизмами синхронизации (критические секции, семафоры) и проблемами тупиков, студенты знакомятся ещё в

курсе «Системное прикладное программное обеспечение» (3, 4 семестры). В этом курсе их знания углубляются, в частности, они получают представления о таких механизмах, как мониторы, рандеву, барьеры.

При изложении этого материала студенты знакомятся с конкретными примерами создания параллельных алгоритмов при решении таких простых задач, как умножение матрицы на вектор, перемножение двух матриц и т.д.

В курсе рассматриваются вопросы реализации этих и более сложных параллельных алгоритмов на многопроцессорной ВС той или иной топологии, анализируется эффективность выполнения параллельных алгоритмов с учетом топологии вычислительной среды.

В разделе «Математические схемы и методы анализа, отладки и тестирования параллельных и распределенных программ» студенты знакомятся с такими известными математическими схемами и методами анализа параллельных и распределенных программ, как схемы Карпа-Миллера, сети Петри , и т.д. При изложении материала, связанного с сетями Петри, студенты рассматривают проблемы решения задач «Читатели-писатели», «О пяти философах» и т.д..

В этом же разделе изучаются вопросы, связанные с характерными особенностями отладки и тестирования параллельных программ.

В разделе языки и системы параллельного программирования студенты изучают язык АДА, язык ОККАМ. Далее студенты знакомятся с потоковыми языками, расширениями языков и, наконец, приступают к изучению системы программирования MPI. Кроме того, излагается материал, связанный с другими системами параллельного программирования (PVM, например, Т-система, система программирования НОРМА). При изучении языков программирования АДА и системы параллельного программирования MPI студентам предлагается разработать параллельный алгоритм для решения поставленной перед ним задачи, оценить его эффективность и реализовать алгоритм: написать программу на языке АДА и программу, с использованием системы параллельного программирования MPI.

В этом же разделе рассматривается вопросы построения оптимизирующих компиляторов, вопросы векторизации и конкуррентизации циклов.

На семинарских занятиях обсуждаются вопросы, которые не вошли в лекционный материал, студенты выступают с докладами (примером может служить вопрос организации параллельных баз данных и др.). Материалами для обсуждения являются материалы из Internet (в основном это материалы портала www.parallel.ru, журнала «Программирование»).

Специальный курс «Параллельные архитектуры» введен с 2003 года для студентов-магистрантов первого года обучения. В этом курсе

студенты изучают в основном вычислительной системы кластерной архитектуры.

В курсе приводится классификация ВС с кластерной архитектурой, рассматриваются архитектурные принципы организации МВС-1000.

Далее студенты изучают вопросы, связанные с аппаратным и программным обеспечением ВС с кластерной архитектурой, а именно, вопросы, связанные с выбором рабочей станции, коммуникационного компонента, операционной системы и т.д..

В специальном курсе «GRID» (магистры 1 года обучения) студенты знакомятся с современным положением дел, связанных с развитием Grid-технологий и метакомпьютинга, рассматривают различные стратегии построения метакомпьютера и организации вычислений в этой среде.

Специальный курс «Распределённые алгоритмы» направлен на углубление знаний распределенной и параллельной обработки информации, подробно изучаются распределённые алгоритмы, в том числе, алгоритмы маршрутизации, волновых алгоритмы, алгоритмы обхода, выбора и т.д.

В общем курсе «Современные компьютерные технологии», материал которого предназначен для изучения студентами-магистрами 2 года обучения, студенты подробно знакомятся ещё с одной системой параллельного программирования. А именно, с системой программирования PVM. Курс включает четыре раздела, одним из которых является материал, связанный с распознаванием образов. В частности, в этом разделе затрагивается вопрос распознавания образов с помощью нейронных сетей. Студентам предлагается реализовать трехслойный персептрон (с использованием системы параллельного программирования MPI), предназначенный для распознавания символов. При этом оценивается эффективность разработанного алгоритма.

Заключение

В настоящее время на механико-математическом факультете Пермского государственного университета функционирует кластер, используя вычислительные средства которого студенты выполняют свои индивидуальные задания. Наряду с этим, студентами выполняются курсовые, дипломные и магистерские работы. Так несколько студентов заняты созданием программных средств для отладки параллельных программ, а ряд других участвует в разработке параллельной системы имитационного моделирования. Ведутся также исследования в области моделирования поведения параллельных программ с целью обнаружения в них ошибок.

РЕГИОНАЛЬНЫЙ НАУЧНО-ОБРАЗОВАТЕЛЬНЫЙ КОМПЛЕКС ВЫСОКОПРОИЗВОДИТЕЛЬНЫХ ВЫЧИСЛЕНИЙ

С.А. Запрягаев, С.Д. Кургалин

Воронежский государственный университет, г. Воронеж

В 2002 г. на кафедре цифровых технологий Воронежского государственного университета (ВГУ) при поддержке гранта *VZ-010* Фонда *CRDF* создан 20-процессорный высокопроизводительный вычислительный кластер с пиковой производительностью 28 Gflops.

Работа кластера существенно увеличила возможности компьютерного моделирования сложных процессов и систем при проведении ресурсоемких расчетов. Оборудование кластера и телекоммуникационные устройства сформировали лабораторию высокопроизводительных вычислений ВГУ, которая включена в региональную информационно-образовательную среду, созданную на базе Воронежского университета, и открывшую новые перспективы для проведения научных исследований.

Лаборатория высокопроизводительных вычислений интегрирована с лабораторией дистанционного образования ВГУ высокоскоростными оптоволоконными каналами доступа к их ресурсам. На их совместной базе организован универсальный научно-образовательный комплекс, в котором ведутся учебные занятия, проходят научные семинары, а также разрабатывается программное обеспечение для моделирования с использованием возможностей компьютерного кластера.

В последние годы высокопроизводительные вычислительные кластеры получают все большее распространение и используются для компьютерного моделирования при решении задач на приоритетных научных направлениях. Решение ряда актуальных задач не всегда может быть проведено на традиционных последовательных компьютерах из-за экспоненциально большого времени выполнения программ, поэтому использование компьютерных кластеров часто становиться единственной возможностью получения новых научных результатов. Построение и дальнейшее изучение математических моделей дает возможность, с одной стороны, исследовать сложные физические, химические и биологические процессы и явления, а, с другой стороны, широкое применение современных методов математического моделирования позволяет по-новому подойти и к решению многих традиционных задач.

Комплекс последовательных программ для анализа сложных реально протекающих процессов на традиционных компьютерах не обеспечивает необходимых объемов моделирования, и проведения полного цикла расчетов из-за существенного возрастания времени вычислений с ростом числа входных параметров или данных и не позволяет получить желаемый результат за приемлемое время. Этим вызывается необходимость приме-

нения вычислительных систем с использованием алгоритмов распараллеливания процессов. При реализации конкретных вычислений распараллеливание осуществляется как на основе известных алгоритмов, так и на базе вновь разрабатываемых процедур для обеспечения эффективной работы конкретной параллельной программы в среде *MPICH 1.2.5*.

Для дистанционного доступа по сети Интернет к вычислительным ресурсам кластера на базе программного обеспечения *Microsoft Internet Information Service* создан специализированный *Web*-сервер. Его интерфейс разработан на основе технологии *Active Server Pages* на языке *Java Script*. Зарегистрированным пользователям предоставляется возможность размещать через *Web*-браузер для выполнения на кластере свои программы, созданные в среде *Microsoft Visual Fortran*, а также дистанционно компилировать и исполнять их. Результаты работы программы переносятся на компьютер пользователя средствами *Web*-браузера.

Объединение параллельного кластера и образовательного Интернет-портала *«Voronezh.OpeNet.ru»* создало условия для проведения сложных вычислительных экспериментов в учебных целях. Осуществляемые вычислительные эксперименты ориентируются не только на проведение расчетов по существующим методикам и компьютерное моделирование, но и на поиск пределов применимости расчетных методов, проверку эффективности алгоритмов и достоверности полученных результатов.

Использование высокопроизводительного вычислительного кластера позволяет продвинуться вперед в решении ряда задач и приобрести опыт разработки параллельных алгоритмов и программ, необходимых для применения новых методов компьютерного моделирования в различных направлениях исследований, а также включить комплекс в европейский проект *EGEE* создания глобальной вычислительной инфраструктуры для науки и высоких технологий.

ПАРАЛЛЕЛЬНАЯ СОРТИРОВКА НА МОДЕЛЯХ КЛЕТОЧНЫХ АВТОМАТОВ

И.И. Захарчук

Военно-космическая академия им. А.Ф. Можайского

Известные модели параллельной сортировки на линейных сетях [1] основаны на PRAM-модели (parallel random access memory) вычислений. Особенностью такой модели является общая память параллельных процессоров, доступ к которой происходит в конкурентном и (или исключительном режимах чтения) записи. Такая модель не отвечает требованиям предельной параллельности в работе (наличие общего ресурса), локальности связей и однородности структуры, которые характерны для кластерных систем.

В качестве вычислительной модели, лишенной указанных недостатков, может быть клеточный автомат – бесконечная сеть одинаковых автоматов Мура, расположенных в точках пространства с целочисленными координатами, связанных одинаковым образом друг с другом и изменяющих состояние в зависимости от состояния соседей и своего собственного.

Формально клеточный автомат K есть упорядоченное множество из четырех компонент

$$K = \langle Z^d, N, Q, \varphi \rangle,$$

где Z^d – множество d-мерных векторов с целочисленными координатами - клеточное пространство;

N – конечное множество мощности m векторов из Z^d:

$$N = \{n_i \,|\, n_i = (x_{1i}, ..., x_{di}), \exists n_i = 0\}, i = 1, ..., m,$$

с нулевым вектором - шаблон соседства клеточного автомата;

Q - конечное множество мощности k состояний клетки с выделенным состоянием покоя $\varnothing$ - алфавит клеточного автомата;

φ - локальная функция переходов, определенная на множестве элементов окрестности в дискретные моменты времени

$$\varphi : Q^m \to Q,$$

$$\varphi : (\varnothing_0, \varnothing_1, ..., \varnothing_{m-1}) = \varnothing.$$

Состояния всех клеток на момент времени t образуют текущую конфигурацию c_t:

$$c_t : Z^d \to Q.$$

Применение локальной функции переходов к текущей конфигурации c_j задает глобальную функцию переходов $g(c)$:

$$g : c_j \to c_{j+1}.$$

Упорядоченная совокупность конфигураций, получаемая из начальной последовательным применением глобальной функции переходов, образует эволюцию e клеточного автомата

$$e = \langle c_0, c_1, ..., c_\tau \rangle, e \in E.$$

В процессе исследований клеточных автоматов сформировались две задачи. В первой задаче задан клеточный автомат и законы его функционирования. Требуется определить состояние каждого элемента после определенного количества тактов (под одним тактом подразуме-

вается один процесс перехода элементов из старого состояния в новое согласно законам данного клеточного автомата). Вторая задача является обратной первой. Есть клеточный автомат и задано первоначальное и конечное состояние системы, к которому необходимо прийти в результате выполнения конечного числа тактов. Требуется найти законы, по которым элементы системы будут изменять свое состояние.

Примером здесь может служить ставшая уже классической известная задача о синхронизации цепи стрелков (firing squad synchronization problem - FSSP), предложенная Дж. Майхиллом. В терминах клеточных автоматов задача может быть формализована так: синтезировать одномерный клеточный автомат с тремя соседними клетками, осуществляющий одновременный переход всех клеток в заключительную однородную конфигурацию (самосинхронизация). Эта задача являются частным примером общей задачи организации информационного обмена внутри однородной сети с локальными связями и децентрализованным управлением.

В настоящем докладе представлено одно из решений другой частной задачи. Эта задача в биологической постановке и без ограничений в локальности связей и однородности структуры известна как модель клеточной дифференциации Вольперта – задача о французском флаге (French flag problem -FFP), или как задача о сортировке Дайкстры – задача о датском флаге (Dutch national flag problem) [2]. Отличие от задачи Майхилла заключается в том, что начальное состояние клетки - одно из множества красок: «красной», «синей», «белой». Распределение красок в начальный момент – произвольное. Требуется установить в финальном состоянии конфигурацию французского флага.

Клеточный автомат, в данной постановке задачи, представляет собой строку – одномерный клеточный автомат, элементы которого расположены последовательно один за другим. Задача решена для любого количества логических состояний элемента системы. Время решения линейно зависит от числа непустых элементов клеточного автомата (длины активной зоны клеточного автомата) и числа возможных состояний.

Ниже приводится RAM-модель (программа на языке C++), которая моделирует работу клеточного автомата, используя разработанный алгоритм.

Так как в клеточном автомате все состояния меняются одновременно (клетки работают параллельно), а процессор в RAM-модели может выполнять команды программы только последовательно, то были созданы два массива a[n] и $b[n]$, где $a[n]$ – старые состояния клеток, $b[n]$ – новые, n – число клеток. Также был введен массив флагов и двух буферных массива forleft[n] и forright[n].

Конкретизируем задачу: пусть существует пять типов клеток (1,2,3,4,5). Первоначально состояния каждой клетки клеточного были за-

даны случайным образом. Требуется отсортировать клетки таким образом, чтобы они располагались слева направо по возрастанию своих номеров.

Действие алгоритма основано на том, что если *a*[*i*-1] > *a*[*i*], то *a*[*i*-1] присваивается значение *a*[*i*] и наоборот *a*[*i*] присваивается значение a[i-1]. Таким образом, состояния клеток «бегут» так, что «большие» состояния «бегут» вправо, а «меньшие – влево».

Трудности возникают, когда состояния непрерывно и последовательно расположены в обратном порядке, например 5-4-3-1 или 5-3-2-1. В этом случае невозможно определить, руководствуясь вышеизложенным правилом, состояния элементов 4 и 3 в первом приведенном примере и 3 и 2 – во втором.

Для преодоления этой трудности, весь «проблемный» блок клеток изолируется от других элементов клеточного автомата посредством флагов (flag[n]). Блок делится на логические элементы по три клетки, состояния крайних двух клеток запоминаются в буфере forleft[n] и forright[n] и в следующем такте происходит обмен «крайними» состояниями, после чего «изоляция» снимается.

```
#include <iostream.h>
int a[n],b[n],flag[n],forleft[n],forright[n],i;
void main()
{
for (i=1,i=n,i++) cin>>a[i];
st: for (i=1,i=n,i++)
{
if a[i-1]>a[i] & a[i]>a[i+1] then
{
forleft[i]=a[i+1];
forright[i]=a[i-1];
flag[i]=2;
}
}for (i=1,i=n,i++)
{
if a[i-1]>a[i] & a[i]<a[i+1] & flag[i-2]=0 & flag[i+1]=0 & flag[i-1]=0
& flag[i] =0 then b[i]=a[i-1];
if a[i-1]<a[i] & a[i]>a[i+1] & flag[i-2]=0 & flag[i+1]=0 & flag[i-1]=0
& flag[i] =0 then b[i]=a[i+1];
}
for (i=1,i=n,i++)
{
if flag[i]=2 & flag[i-2]!=0 then
{
flag[i]=1;
```

```
forleft[i]=0;
forright[i]=0;
}
}
for (i=1,i=n,i++)
{
if flag[i+1]=2 then
{
b[i]=forleft[i+1];
forleft[i+1]=0;
}
if flag[i-1]=2 then
{
b[i]=forright[i-1];
forright[i-1]=0;
}
}
for (i=1,i=n,i++) a[i]=b[i];
for (i=1,i=n,i++) cout<<a[i];
goto st;
```

Литература

1. *Akl S.* Parallel sorting. Academic Press, Orlando, FL, 1985.
2. *Dijkstra E.W.* A discipline of programming. Prentice Hall, Englewood Cliffs, NY, 1976.

ИТЕРАЦИОННОЕ ПЛАНИРОВАНИЕ РАСПРЕДЕЛЕНИЯ РЕСУРСОВ МНОГОПРОЦЕССОРНЫХ СИСТЕМ

Д.И. Зимин, В.А. Фурсов

Институт систем обработки изображений, г. Самара

Введение

При организации обработки большеформатных изображений [1] в распоряжении пользователя часто имеется достаточное число вычислительных ресурсов, которые используются лишь часть времени. Естественно желание использовать мощности простаивающих ресурсов для существенного ускорения процесса обработки изображений.

Идея использования для указанных задач существующей инфраструктуры из персональных компьютеров и коммуникационной среды активно развивается в последние годы. Вместе с тем, оказалось, что при использовании многопроцессорной системы возникают серьезные труд-

ности, связанные с разнородностью и неравномерной загрузкой частично свободных ресурсов отдельных узлов многопроцессорной системы.

В работе [2] решалась задача использования многопроцессорных систем без учета изменения во времени степени загрузки компьютеров. В настоящей работе приводится технология итерационного планирования распределения ресурсов с эпизодическим изменением плана включения и исключения узлов многопроцессорной системы при выполнении параллельной программы. Приводится пример реализации GRID-технологии итерационного планирования ресурсов.

1. Постановка задачи распределения данных

С вычислительной точки зрения основные алгоритмы обработки изображений в пространственной области допускают декомпозицию по данным и с этой точки зрения могут быть разбиты на три группы:

1) декомпозиция по данным, не требующая обменов между процессорами;
2) необходим обмен данными после завершения обработки всех фрагментов;
3) процесс обработки должен осуществляться итерационно, при этом необходимо осуществлять обмен данными после каждой итерации.

Последний из указанных класс алгоритмов является наиболее сложным с точки зрения организации вычислительного процесса в распределенных (в особенности гетерогенных) системах. В настоящей работе рассматривается только этот класс алгоритмов, поскольку первые два могут рассматриваться как его частный случай.

В данном случае важное значение имеет балансировка загрузки процессоров на каждой итерации. Для уменьшения расходов на межузловые коммуникации в процессе обработки изображение разбивается на квадратные области [3] (рис. 1). При этом обмен данными целесообразно осуществлять в последовательности, указанной на рис. 2 а), б), в).

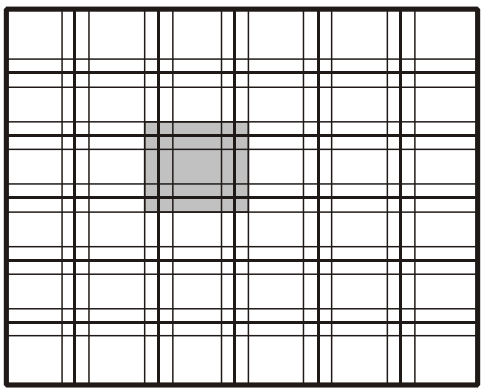


Рис. 1. Двумерная декомпозиция

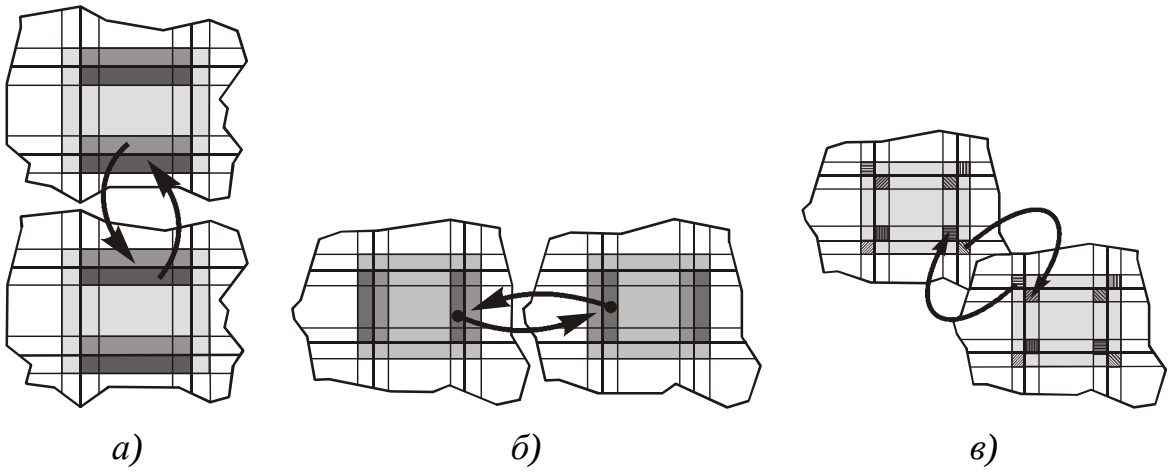


Рис. 2. Этапы обмена данными:
а) первый этап, б) второй этап, в) третий этап

Вследствие неоднородности многопроцессорной вычислительной системы, время обработки одинаковых фрагментов изображения в узлах будет различным. На время обработки фрагмента влияют многие параметры узла, в частности, частота процессора, частота системной шины и памяти, размер КЭШа, параметры коммуникационной среды и т.д.

Определить явную зависимость времени обработки изображения от его размера не представляется возможным, вследствие того, что для различных алгоритмов перечисленные параметры случайным образом взаимодействуют.

Поскольку на используемых для обработки фрагментов изображения узлах одновременно могут выполняться другие виды работ (редактирование текстов, написание программ и др.) производительность вычислительных узлов изменяется также и во времени. Если использовать простаивание узлов в «рабочее время» и полностью использовать «ночное время», то часто можно обойтись без использования дорогостоящих кластеров и суперкомпьютеров.

2. Общая схема итерационного распределения ресурсов

Для распределения ресурсов многопроцессорной системы с характеристиками узлов, зависящими от времени, целесообразно применить процедуру итерационного планирования, обсуждавшуюся в работе [3]. Идея заключается в том, чтобы улучшать качество плана распределения ресурсов по мере увеличения числа реальных запусков. Общая схема итерационного планирования ресурсов приведена на рис. 3. Процедура планирования анализирует эффективность выполнения параллельной программы на предыдущих запусках и формирует новый план распределения.

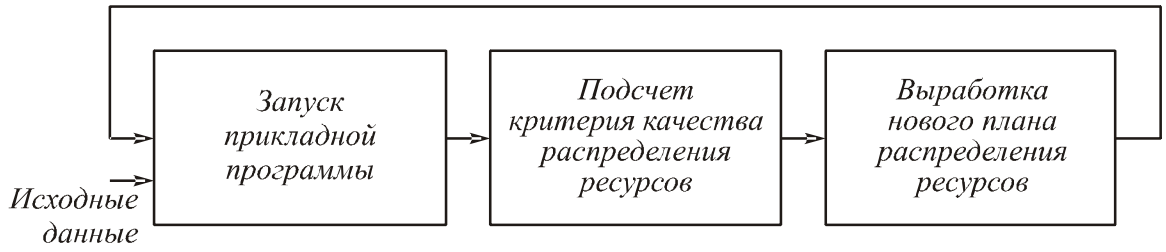


Рис. 3. Схема итерационного планирования распределения ресурсов

Для обеспечения равномерной нагрузки на узлы необходимо эпизодически уточнять план с учетом априорных моделей, характеризующих производительность узлов в различные периоды времени при различных размерах обрабатываемых фрагментов большеформатного изображения. Предлагается следующая модель временных затрат τ на обработку фрагмента отдельным узлом:

$$\tau(x,t) = K(t) \cdot T(x), \tag{1}$$

где $K(t)$ коэффициент, характеризующий зависимость степени загрузки компьютеров от текущего времени, $T(x)$ зависимость времени обработки фрагмента от его размера.

Задача определения зависимости (1) решается для каждого узла по результатам тестовых запусков параллельной программы обработки изображения для различных размеров фрагментов в разное время. Коэффициент $K(t)$ обычно оказывается периодической функцией, с периодом 24 часа, но в некоторых ситуациях период может изменяться, либо не проявляться вовсе.

В работе [2] показано, что для многих алгоритмов обработки зависимость времени обработки от размеров фрагмента изображения хорошо описывается экспоненциальной функцией:

$$T(x) = a_0 e^{a_1 x}, \tag{2}$$

где параметры a_0 и a_1 определяются путем решения соответствующей задачи идентификации [1].

После определения параметров модели (1) для каждого узла задача распределения фрагментов изображения по узлам сводится к определению параметров $(x_1,...,x_n)$, удовлетворяющих системе уравнений (условиям балансировки процессоров):

$$\begin{cases} \tau_1(x_1,t) = \tau^*, \\ \quad ... \\ \tau_n(x_n,t) = \tau^*, \\ \sum_i x_i = N, \end{cases} \tag{3}$$

где N размер всего изображения, τ^* время обработки фрагмента на каждом узле.

При итерационном распределении ресурсов на каждом шаге планирования обычно необходимо решать вопрос о необходимости включения или исключения некоторого узла. В некоторых случаях включение дополнительного узла не повышает быстродействия обработки изображения. Это возможно, например, в ситуации, когда включение узла сопровождается существенным возрастанием коммуникационных расходов в многопроцессорной системе в целом.

Для принятия решения необходимо решать систему (3) для нескольких вариантов числа узлов и сравнивать прогнозируемое время обработки изображения. Если оцененное время, при выключенных из расчета узлах не меньше, чем при использовании этих узлов, такой план распределения ресурсов принимается на текущем шаге.

2. Результаты экспериментов

Были проведены вычислительные эксперименты по распределению данных между узлами при решении задачи восстановления изображения. Для различных размеров фрагмента получены зависимости $T(x)$ (см. рис.4).

Исследованиями также установлена зависимость коэффициента $K(t)$ от времени, она приближенно описывается как:

$$K(t)=\begin{cases} 0.7; t\in[9,18], \\ 1; t\in[0,9)\cup(18,24]. \end{cases} \quad (4)$$

Указанная зависимость отражает тот факт, что в «рабочее время» нагрузка на процессор и коммуникационную сеть возрастает.

Исходя из этих данных, найдено разбиение изображения размером 8192×8192 отсчетов. Обработка изображения происходила на 3-х различных по производительности узлах. Время параллельной обработки изображения составило $405,6$ секунд. Результаты эксперимента приведены в таблице 1.

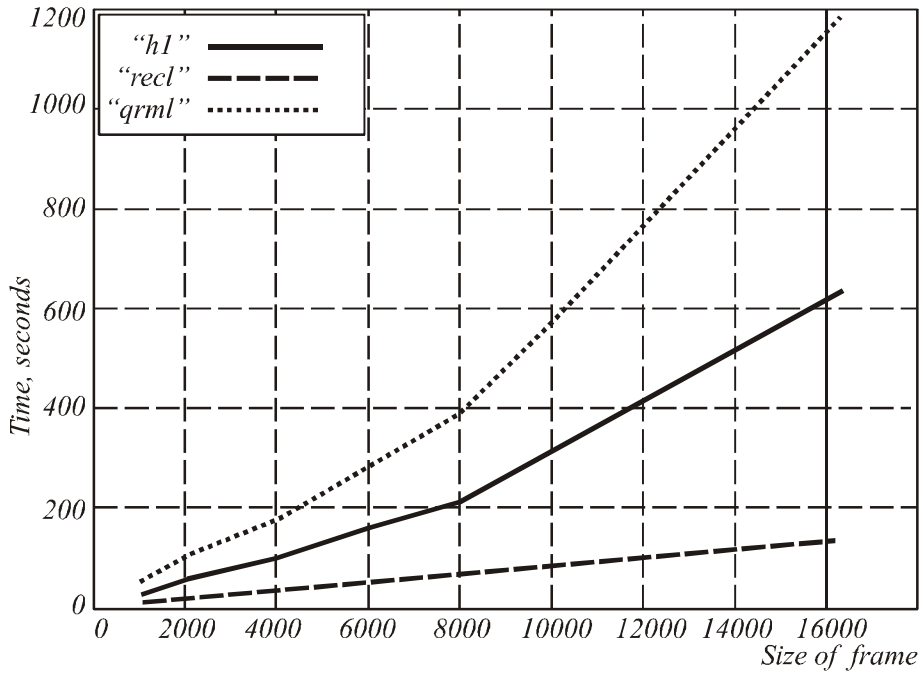


Рис. 4

Таблица 1

Узел	Размер обработанного фрагмента (пиксел)	Время последовательной обработки изображения (с)	Ускорение
N 1	45334528	592,8	1,46
N 2	13385728	1937,6	4,77
N 3	8388608	3158,4	7,82

Из таблицы 1 видно значительное уменьшение времени обработки изображения, при использовании описанного способа использовании свободных ресурсов.

Заключение

Проведенные вычислительные эксперименты подтверждают, что предлагаемые рекомендации по схеме декомпозиции и распределению ресурсов многопроцессорных гетерогенных систем могут эффективно использоваться в задачах обработки изображений.

Работа выполнена при поддержке российско-американской программы Фундаментальные исследования и высшее образование и РФФИ (грант №03-01-00109).

Литература

1. Методы компьютерной обработки изображений / Под ред. Сойфера В.А., Москва, Физматлит, 2001.

2. *Зимин Д.И., Фурсов В.А.* Распределенная обработка большеформатных изображений на многопроцессорных системах // Международная конференция «Распределенные вычисления и Грид-технологии в науке и образовании», Дубна, 2004.

3. *Фурсов В.А., Шустов В.А., Скуратов С.А.* Технология итерационного планирования распределения ресурсов гетерогенного кластера // Труды Всероссийской научной конференции «Высокопроизводительные вычисления и их приложения», Черноголовка, 2000.

МЕТОДЫ И ИНСТРУМЕНТАЛЬНЫЕ СРЕДЫ ПОСТРОЕНИЯ ПРОГРАММНЫХ СРЕДСТВ ДЛЯ УПРАВЛЕНИЯ РАСПРЕДЕЛЁННЫМИ СИСТЕМАМИ

А. Кардашич

Московский энергетический институт (ТУ), г. Москва

Введение

Распределенная система (РС) – это комплекс вычислительных и программных ресурсов, способных совместно выполнять информационно-вычислительную работу.

Примерами РС являются информационно-вычислительные сети, кластеры и т.п. В отличие от аппаратных ресурсов, к которым мы относим компьютеры, коммуникации, память и др., программные ресурсы обладают большей степенью свободы, они легко могут изменять место своего хранения, реплицироваться (т.е. копироваться) и модифицироваться в соответствии со спецификой решаемых задач.

Подобно социальным системам (фирмы, университеты, армия, города и др.), РС могут иметь различные организационные структуры и управление, подверженные постоянным изменениям, вызванными требованиями повышения их эффективности, быстрой реакции на катастрофического характера события и естественную деградацию со временем.

Основные проблемы, которые встают перед создателями РС следующие:

– поиск моделей и методов оптимального управления и распределения ресурсов,

– обеспечение их высокой надёжности,

– создание технологий построения различных систем управления РС,

– разработка инструментария, средствами которого можно быстро проектировать РС.

Организационные схемы управления РС

Будем предполагать, что с организационной точки зрения РС представляет собой распределенное в пространстве множество подсистем, предназначенных в общем случае для выполнения определенных функций, которые могут объединяться для решения общих задач.

В каждой такой подсистеме можно выделить относительно самостоятельные две части: исполнительную и управление. Роль исполнительной части – выполнение возлагаемых управлением задач, роль управления – координация процесса функционирования элементов исполнительной части, и взаимодействие с другими подсистемами. В зависимости от того, каким образом построено взаимодействие между управлением подсистем (у каждой системы в общем случае свое, отличное от других подсистем управление), их классифицируют следующим образом:

– децентрализованное, когда управления подсистемами полностью самостоятельны и координация их работы происходит путем коллективного согласования с целью принятия общих решений по управлению подчиненными подсистемами. В этом случае требуется высокая «квалификация» для управляющих узлов и обеспечение высокоскоростных коммуникаций для согласующих взаимодействий;

– децентрализованное управление, когда управление подсистемами делегирует основные функции по управлению подсистемами и координации их взаимодействия новому более высокого уровня узлу управления. В этом случае требуется высокая квалификация для центрального узла управления, способность им быстро принимать решения (большое быстродействие в реализации) и высокая скорость обмена данными с подчиненными узлами;

– смешанное, в частности, иерархическое, управление, которое предполагает большую свободу в выборе структуры управления, специализации функций управления и схемы подчинения между управляющими узлами.

Мультиагентный подход к управлению[1] предполагает еще одну, более универсальную стратегию организации управления, когда нет жесткой связи между подсистемой и ее управлением, сами блоки управления распределены по различным компьютерам РС. В зависимости от выполняемых задач подсистема «заимствует» наиболее подходящее ей управление на определенное время, а затем меняет его на другое, соответствующее другим обстоятельствам.

Ясно, что на уровне программных ресурсов естественно и достаточно просто реализуется мультиагентный подход, ибо нет никаких проблем, чтобы программный ресурс сделать подвижным (перемещаемым) и реплицируемым, более того, настраиваемым на условия соответствующей задачи.

Существенно, и из этого надо исходить, что РС является системой с переменной структурой, в ней могут происходить отказы и восстановления компонентов, масштабирование как на уровне подсистем, так и в самих подсистемах.

О ресурсах и их характеристиках

Мы уже сказали, что ресурсы целесообразно разделить на физические (компьютеры, память, коммуникации и т. п.) и программные. Каждый ресурс характеризуется:

– типом, относящим его к конкретному виду физических или программных ресурсов,

– множеством функций, которые ресурс может выполнять,

– местом своего размещения X (X может, например, быть точкой 3-х мерного пространства, если мы рассматриваем вычислительную систему, а ресурс R – есть конкретный компьютер в этой точке, или это место задается структурой именования, связанного с принадлежностью ресурса какой-то подсистеме, какому-то конкретному компьютеру в ней (к примеру, если речь идет о размещении базы данных),

– дескриптором, характеризующим характеристики ресурса (для памяти – время обращения, для компьютера – быстродействие и объем оперативной и дисковой памяти и др., для программы, предназначенной для решения систем линейных уравнений – это метод, его точность, вычислительная сложность, требуемая память и др.), возможность его адаптации к задаче (пример генетических алгоритмов), возможность репликации и др.,

– разрешенным доступом к ресурсу, определяющим способ обращения к нему и круг других ресурсов, которым разрешено ресурс использовать.

О состоянии и использовании ресурса

Ресурс с позиций его использования помимо технических или «программных» характеристик, отраженных в его описании, характеризуется:

– способностью выполнять свои функции (компьютер работоспособен или отказал, находится в процессе диагностики, в программном ресурсе обнаружена ошибка и т.п.),

– состоянием (ресурс занят или свободен),

– загруженностью (у компьютера n задач, свободная память $V_{св.}$, интенсивность обмена между дисковой и оперативной памятью $\lambda_д$, интенсивность обмена с другими элементами РС, например, с другими компьютерами $\lambda_{об}$ и др.). Кстати, у ОС Windows все эти характеристики измеряются, их можно получать автоматически,

– использованным процессами процессорным временем, общим временем выполнения и др., что важно для программных ресурсов.

Для разделяемых программных ресурсов особенно важны данные о количестве (интенсивности) запросов к ним, приоритетах и времени их обслуживания, времени доступа к ним и др.

Естественно, что идеально, если ресурс сам может контролировать свое состояние и использование. Компьютер может многое делать сам, используя средства ОС. С программным ресурсом сложнее, эти характеристики приходится получать косвенно, как правило, обращаясь к управлению, которое через соответствующие «службы» выполняет эту работу.

Об основных функциях управления РС

Очевидно, что для того, чтобы управлять РС, необходимо иметь статическую информацию обо всех её ресурсах, содержащую данные о местоположении и характеристики, а также динамическую информацию, касающуюся состояния и использования. Помимо всего, рассматривая управление в общем случае как децентрализованное или смешанное (т.е. также распределенное, как и сама система), необходимо определить общие и частные (в зависимости от схемы управления) правила, по которым узлы управления взаимодействуют между собой при совместном решении задачи.

Рассмотри функции управления РС более детально, выделив в нём наиболее общего значения блоки.

Эти блоки целесообразно разделить на два подмножества: блоки управления системного уровня и блоки управления процессного уровня.

Первые ответственны за управление и поддержание целостности РС в целом, вторые – за управление «рабочими» процессами, протекающими в РС.

К системным блокам мы отнесём следующие блоки.

Блок администрирования РС, выполняющий функции по «ручному» конфигурированию РС и инсталляции на ней необходимых программных средств.

Блок динамической реконфигурации, который подобно администратору РС выполняет функции по изменению конфигурации РС (из-за отказов и восстановлений её компонентов, из-за необходимости масштабирования в процессе функционирования и т.п.) автоматически.

Блок отказоустойчивости, непосредственно взаимодействующий с блоком реконфигурации, который ответственен за поддержание работоспособности РС в случае отказов её компонентов. Для этой цели обычно применяются схемы резервирования, периодического запоминания на выделенных носителях памяти состояний (контекстов) программных компонентов РС, которые передаются для восстановления выполняемого процесса в случае отказа компьютера. Возможны другие, более эффективные методы обеспечения отказоустойчивости, например, осно-

ванные на статистически или динамически устанавливаемой договорённости между компьютерами и подсистемами РС с целью взаимной реакции на отказы и восстановления.

Блок регулирования загрузки компонентов РС, основной целью которого является её равномерное распределение. Этот блок взаимодействует с блоком определения загруженности каждого компьютера РС, получая периодически от него контролируемые параметры загрузки. Наиболее важными из них являются: объём свободной оперативной памяти, частота обменов страницами между дисковой и оперативной памятью, величина трафика сообщений, которыми компьютеры РС обмениваются в процессе её работы, количество процессов, которые выполняет компьютер и др.

Блок регулирования загрузки (он рассматривается как часть управления некоторой подсистемы РС), получая эти данные, определяет загрузку подчинённой ему подсистемы, возможно, её девиацию, более точно характеризующую степень неравномерности загрузки компьютеров, а также определяет некоторый прогноз, как будет изменяться загрузка в ближайшее время [8]. По этой информации управление принимает решение, какие компьютеры должны передать часть своих процессов другим компьютерам с тем, чтобы попытаться их загрузку сделать одинаковой. При иерархической организации общего управления РС может существовать множество специализированных подсистем управления: управление компьютером, конкретной частью (подсистемой) РС, управление более высокого уровня, осуществляющее взаимодействие между подсистемами управления более низкого уровня и одновременно взаимодействующее с управлением более высокого уровня, снабжая его необходимой информацией о загруженности подчинённых компонентов.

Интерфейсный блок, который обеспечивает согласование сообщений, передаваемых между компьютерами (сообщения могут быть на разных языках), собственно их приём и передачу.

Блок информационной безопасности, который ответственен за выполнение всех установленных правил использования ресурсов и их защиты от несанкционированных попыток обращения к ним, намеренного разрушения, внесения вирусов и т.п. Если блок отказоустойчивости в основном реагирует на неполадки в аппаратуре, то данный блок имеет дело с программными ресурсами.

Информационный блок РС, который подобно адресной книге (или порталу), актуализирует все данные о ресурсах РС, их местонахождении, использовании и др.

Блоки управления РС процессного уровня выполняют функции, непосредственно связанные с выполнением и синхронизацией «рабочих» процессов, индуцируемых при выполнении пользовательских программ, а также запросов к базам данных, файловым системам и т.п.

Многие из этих функций, такие как сохранение и восстановление контекста при прерывании и активизации процесса, реакция на команды взаимодействия между процессами, собственно выполнение процесса по сложившейся схеме в компьютере и др., выполняются непосредственно ОС компьютера или сетевой ОС, если у процесса возникает необходимость обращаться, например, к удалённым web-службам, базам данных.

Сложнее обстоит дело, если пользователь хочет организовать параллельное выполнение своей программы, причём стандартные средства ОС (процессы – нити, и т.п.) неадекватны его цели. В этом случае обычно необходим интерпретатор, который выявляет в программе могущие одновременно выполняться фрагменты, необходим планировщик, который, взаимодействуя с интерпретатором, устанавливает приоритеты для одновременно выполняемых фрагментов, а через блок назначения ресурсов процессам (этот блок в отличие от системного блока регулирования загрузки решает не стратегические задачи в разделении ресурсов, а практические, назначая процессам необходимые ресурсы и соблюдая заданные им приоритеты).

Таким образом, и на процессном уровне появляется множество нестандартных блоков, вовлечённых в процесс управления РС, которые приходится создавать как определённое расширение ОС.

О реализации управления РС

В предыдущем пункте мы попытались вычленить основные (очевидно, не все) имеющее самостоятельное значение блоки, предназначенные для организации управления РС. Разделение этих блоков на системные и процессные вносит очевидное соподчинение между ними.

В РС должны поддерживаться различные, могущие динамически изменяться, схемы управления, и должна бать разработана технология их построения на основе определённой функциональной декомпозиции управления в целом.

Другая задача – это создание среды проектирования, которая позволяет, используя в основном стандартные программные средства быстро реализовать различные схемы управления РС.

Многоагентный подход к построению управления [1] позволяет объединить различные модели управления. Естественно, что перечисленные выше блоки управления РС могут рассматриваться как специализированные агенты, предназначенные для этой цели.

Вполне очевиден и путь построения агентов, поскольку объектно-ориентированное программирование в большой степени упрощает решение этой задачи [2]. Более того, сегодня уже существуют инструментальные среды, которые упрощают реализацию многоагентного подхода [3-6].

Средства программирования многоагентных систем, основанные на понятии агента и на языке Java [4], сменяют более мощные среды с более проработанной технологией использованием объектно-ориентированных языков для этой цели (к примеру, назовём IBM aglets work bench [6]).

Комплексное решение проблемы создания эффективных средств для управления РС, сетями и т.п. потребует большей интеграции с работами, которые активно ведутся в области создания web-сервисов [7], распределённых баз данных и др. Остаются актуальными и чисто математические проблемы оптимизации управления распределения ресурсов, распараллеливания программ, обеспечения надёжности РС [8].

Заключение

В настоящее время мы разрабатываем технологию построения программных средств для систем управления РС, которая нацелена на существенное сокращение времени их проектирования, которая учитывает естественную эволюцию РС, изменение их структуры и схем управления и которая базируется на указанных инструментальных средах для их создания.

Работа выполнена при поддержке РФФИ, проект 03-01-00588

Литература

1. *Городецкий В.И.* и др. Многоагентные системы (обзор) // Новости искусственного интеллекта. 1998. №2.

2. *Буч Г.* Объектно-ориентированный анализ и проектирование с примерами приложений на C++ // М.: Издательство «Бином», 1998.

3. *Puliafito A.* et all. MAP: Design and implementation of a mobile agent's platform // Journal of System Architecture, 2000. №46.

4. *Lange D.B., Oshima M.* Mobile agents with Java. IBM Tokyo research laboratory, 1998.

5. *Lange D.B., Aridor Y.* Agent design patterns: elements of agent application design. ACM Press, 1998.

6. *Oshima M., Karjoth G.* Aglets specification 1.1 Draft. IBM Tokyo research laboratory, 1998.

7. *Скотт Ш.* Разработка XML web – сервисов средствами Microsoft.Net // СПБ.: БХВ – 2003.

8. *Кутепов В.П.* Об интеллектуальных компьютерах и больших компьютерных системах // Теория и системы управления, 1996. №5.

МОДИФИЦИРОВАННЫЙ ВЕЙВЛЕТ-АНАЛИЗ ИЗОБРАЖЕНИЙ С ПОМОЩЬЮ КОЛЬЦЕВОГО ПРЕОБРАЗОВАНИЯ РАДОНА

А.А. Ковалев[1], В.В. Котляр[2]

[1] *Самарский государственный аэрокосмический университет, г. Самара,*
[2] *Институт систем обработки изображений РАН, г. Самара,*

Рассмотрено направленное вейвлет-преобразование двумерных функций. Для более эффективного его выполнения предложено использовать кольцевое преобразование Радона как среднее по всем окружностям фиксированного радиуса на плоскости. Предложена схема вычислений для набора углов и масштабов направленного вейвлет-преобразования.

Введение

Вейвлет-анализ возник при обработке записей сейсмодатчиков в нефтеразведке и с самого начала был ориентирован на локализацию разномасштабных деталей. Выросшую из этих идей технику теперь обычно называют непрерывным вейвлет-анализом. Ее основные приложения: локализация и классификация особых точек сигнала, вычисление его различных фрактальных характеристик, частотно-временной анализ нестационарных сигналов. Например, у таких сигналов, как музыка и речь, спектр радикально меняется во времени, а характер этих изменений – очень важная информация. Вейвлет-анализ также применяется в сжатии данных, фильтрации, системах распознавания образов, синтезаторах речи, медицине, метеорологии.

Вейвлет-преобразование является разложением функции по базису, образованному вейвлетами – солитонообразными функциями двух аргументов – масштаба и сдвига [1]. В отличие от преобразования Фурье, множество базисных функций имеет размерность 2, а не 1. Тогда при анализе двумерных функций множество базисных функций должно иметь размерность 4, что приводит к вычислительной сложности. Чтобы устранить этот недостаток, существует несколько подходов.

В [2] предлагается использовать тензорное произведение одномерных вейвлетов. То есть, по аналогии с традиционными спектральными преобразованиями, двумерное вейвлет-преобразование выполняется как последовательность одномерных преобразований над строками и столбцами изображения.

В [3] предлагается «шахматная» схема вейвлет-преобразования изображений, базис которой образован функциями с высокой степенью изотропии. Такой подход позволяет избежать крестообразного характера базисных функций и появления на плоскости четырех выделенных направлений (по обеим осям и по диагоналям). Эта схема хорошо подходит для изображений, на которых преобладают изотропные элементы («пятна»).

Существует класс изображений с отсутствующими выделенными направлениями, и при этом объекты на изображении неизотропны. К такому классу относятся, например, медицинские изображения сосудистых систем. Объекты (сосуды) на таких изображениях могут иметь произвольное направление и для анализа их сечений можно применять одномерное вейвлет-преобразование под разными углами.

Направленное вейвлет-преобразование

Традиционное вейвлет-преобразование имеет вид:

$$F(a,b)=\frac{1}{\sqrt{a}}\int_{-\infty}^{+\infty} f(x)\Psi\left(\frac{x-b}{a}\right)dx. \quad (1)$$

Заменой переменной получим следующую форму вейвлет-преобразования:

$$F(a,b)=\sqrt{a}\int_{-\infty}^{+\infty} f(b+at)\Psi(t)dt \quad . \quad (2)$$

Определим направленное вейвлет-преобразование следующим образом:

$$F(a,\theta,\xi,\eta)=\sqrt{a}\int_{-\infty}^{+\infty} f(\xi+at\cos\theta,\eta+at\sin\theta)\Psi(t)dt. \quad (3)$$

Очевидно, при $\theta=0$ и $\theta=\pi/2$ выражение (3) будет являться соответственно постолбцовым и построчным одномерным вейвлет-преобразованием двумерной функции.

Рассмотрим свертку двумерной функции *f(x,y)* со следующим ядром:

$$h(x,y)=\frac{1}{\sqrt{a}}\delta(x\sin\theta-y\cos\theta)\Psi\left(-\frac{x\cos\theta+y\sin\theta}{a}\right). \quad (4)$$

В результате свертки будет получена следующая функция (при $\theta\neq\pi k/2, k\in Z$):

$$g(\xi,\eta)=\frac{1}{\sqrt{a}}\iint_{R^2} f(\xi-x,\eta-y)\delta(x\sin\theta-y\cos\theta)\Psi\left(-\frac{x\cos\theta+y\sin\theta}{a}\right)dxdy=$$

$$=\{x=u\cos\theta,\ y=v\sin\theta\}=$$

$$=\frac{\sin\theta\cos\theta}{\sqrt{a}}\iint_{R^2} f(\xi-u\cos\theta,\eta-v\sin\theta)\delta\left[(u-v)\sin\theta\cos\theta\right]\Psi\left(-\frac{u\cos^2\theta+v\sin^2\theta}{a}\right)dudv=$$

$$=\frac{1}{\sqrt{a}}\int_{-\infty}^{+\infty} f(\xi-u\cos\theta,\eta-u\sin\theta)\Psi\left(-\frac{u}{a}\right)du=\{u=-at\}=$$

$$=\sqrt{a}\int_{-\infty}^{+\infty} f(\xi+at\cos\theta,\eta+at\sin\theta)\Psi(t)dt \quad . \quad (5)$$

То есть для фиксированного масштаба a и угла θ направленное вейвлет-преобразование является сверткой с импульсной характеристи-

кой (4) и может быть эффективно вычислено с помощью Фурье-коррелятора.

Однако если используется целый набор углов θ и масштабов a, для каждого угла и масштаба нужен свой пространственный фильтр и, следовательно, требуется отдельное выполнение преобразования Фурье. В данной работе предлагается вычислительно более эффективный подход, основанный на кольцевом преобразовании Радона (КПР) [4].

Кольцевое преобразование Радона

В данной работе под КПР будем понимать суммирование (усреднение) по всем окружностям с фиксированным радиусом γ и с центром в точке (ξ,η):

$$R_\gamma(\xi,\eta) = \gamma \int_0^{2\pi} f(\xi + \gamma\cos\varphi, \eta + \gamma\sin\varphi)\, d\varphi . \quad (6)$$

В [4] показано, что КПР можно представить в виде Фурье-коррелятора, функция комплексного пропускания пространственного фильтра которого равна:

$$H_\gamma(\xi,\eta) = 2\pi\gamma\, J_0\left(\gamma\sqrt{\xi^2+\eta^2}\right). \quad (7)$$

Можно показать, что кольцевым импульсным откликом обладают также Фурье-корреляторы с функцией комплексного пропускания пространственного фильтра, имеющей следующий вид (в полярных координатах):

$$H_\gamma^n(\rho,\theta) = \exp(-in\theta)\, J_n(\gamma\rho). \quad (8)$$

Преобразование в таком Фурье-корреляторе имеет вид:

$$R_\gamma^n(\xi,\eta) = \int_0^{2\pi} f(\xi + \gamma\cos\varphi, \eta + \gamma\sin\varphi)\exp(in\varphi)\, d\varphi . \quad (9)$$

Преобразование (9) будем называть кольцевым преобразованием Радона n-го порядка. Видно, что преобразование (6) соответствует нулевому порядку.

Взвешенное кольцевое преобразование Радона

В [5] предлагается обобщение традиционного преобразования Радона – так называемое экспоненциальное преобразование Радона, имеющее следующий вид:

$$R_\mu(p,\alpha) = \int_{-\infty}^{+\infty} f(p\cos\alpha + t\sin\alpha,\, p\sin\alpha - t\cos\alpha)\exp(\mu t)\, dt . \quad (10)$$

По аналогии, можно рассмотреть дальнейшее обобщение преобразования (6) – взвешенное кольцевое преобразование Радона (ВКПР), интегрирование по окружности в котором происходит с некоторой весовой функцией $\mu(\varphi)$ (периодичной с периодом 2π):

$$R_\gamma^{\mu(\varphi)}(\xi,\eta) = \gamma \int_0^{2\pi} f(\xi + \gamma\cos\varphi, \eta + \gamma\sin\varphi)\,\mu(\varphi)\,d\varphi \quad , \tag{11}$$

ВКПР может быть выражено через КПР разных порядков в виде ряда:

$$R_\gamma^{\mu(\varphi)}(\xi,\eta) = 2\pi\gamma \cdot \sum_{-\infty}^{+\infty} (-i)^n \mu_n R_\gamma^n(\xi,\eta) \quad , \tag{12}$$

где $R_\gamma^n(\xi,\eta)$ определяются из (9), μ_n – коэффициенты разложения $\mu(\varphi)$ в ряд Фурье:

$$\mu_n = \frac{1}{2\pi}\int_0^{2\pi} \mu(\varphi)\exp(in\varphi)\,d\varphi \,. \tag{13}$$

Выполнение направленного вейвлет-преобразования с помощью взвешенного кольцевого преобразования Радона

Выражение (3) можно преобразовать следующим образом:

$$F(a,\theta,\xi,\eta) = \sqrt{a}\int_{-\infty}^{+\infty} f(\mathbf{p}+at\boldsymbol{\theta})\Psi(t)\,dt = \sqrt{a}\int_{-\infty}^{+\infty} f\left(\mathbf{p}+\gamma\boldsymbol{\theta}^\perp + at\boldsymbol{\theta} - \gamma\boldsymbol{\theta}^\perp\right)\Psi(t)\,dt =$$

$$= \sqrt{a}\int_{-\infty}^{+\infty} f\left(\mathbf{p}+\gamma\boldsymbol{\theta}^\perp + \sqrt{a^2t^2+\gamma^2}\left(\frac{at}{\sqrt{a^2t^2+\gamma^2}}\boldsymbol{\theta} - \frac{\gamma}{\sqrt{a^2t^2+\gamma^2}}\boldsymbol{\theta}^\perp\right)\right)\Psi(t)\,dt =$$

$$= \sqrt{a}\int_{-\infty}^{+\infty} f\left(\mathbf{p}+\gamma\boldsymbol{\theta}^\perp + \sqrt{a^2t^2+\gamma^2}\left(\sin\varphi\cdot\boldsymbol{\theta} - \cos\varphi\cdot\boldsymbol{\theta}^\perp\right)\right)\Psi(t)\,dt =$$

$$= \sqrt{a}\int_{-\infty}^{+\infty} f\left(\mathbf{p}+\gamma\boldsymbol{\theta}^\perp + \sqrt{a^2t^2+\gamma^2}\begin{bmatrix}\sin(\varphi-\theta)\\ \cos(\varphi-\theta)\end{bmatrix}\right)\Psi(t)\,dt =$$

$$= \sqrt{a}\int_{-\infty}^{+\infty} f\left(\mathbf{p}+\gamma\boldsymbol{\theta}^\perp + \sqrt{a^2t^2+\gamma^2}\begin{bmatrix}\cos(\pi/2-\varphi+\theta)\\ \sin(\pi/2-\varphi+\theta)\end{bmatrix}\right)\Psi(t)\,dt \,, \tag{14}$$

где $\mathbf{p} = \begin{bmatrix}\xi\\ \eta\end{bmatrix}$, $\boldsymbol{\theta} = \begin{bmatrix}\cos\theta\\ \sin\theta\end{bmatrix}$, $\boldsymbol{\theta}^\perp = \begin{bmatrix}\sin\theta\\ -\cos\theta\end{bmatrix}$, $\gamma > 0$, $\sin\varphi = \frac{at}{\sqrt{a^2t^2+\gamma^2}}$,

$\cos\varphi = \frac{\gamma}{\sqrt{a^2t^2+\gamma^2}}$.

При $t \ll \gamma$ и $\gamma \gg 0$ $\sqrt{a^2t^2+\gamma^2} \approx \gamma$, $\sin\varphi \approx \varphi$ и $\frac{at}{\sqrt{a^2t^2+\gamma^2}} \approx \frac{at}{\gamma}$. Тогда имеет место приближенное равенство:

$$t \approx \frac{\gamma}{a}\varphi. \tag{15}$$

Тогда (14) может быть переписано в следующем виде:

$$\begin{aligned} F(a,\theta,\xi,\eta) &= \frac{\gamma}{\sqrt{a}}\int_{-\infty}^{+\infty} f\left(\mathbf{p}+\gamma\boldsymbol{\theta}^{\perp}+\gamma\begin{bmatrix}\cos(\pi/2+\theta-\varphi)\\ \sin(\pi/2+\theta-\varphi)\end{bmatrix}\right)\Psi\left(\frac{\gamma}{a}\varphi\right)d\varphi = \\ &= \{\pi/2+\theta-\varphi=\varphi'\} = \frac{\gamma}{\sqrt{a}}\int_{-\infty}^{+\infty} f\left(\mathbf{p}+\gamma\boldsymbol{\theta}^{\perp}+\gamma\begin{bmatrix}\cos\varphi'\\ \sin\varphi'\end{bmatrix}\right)\Psi\left(\frac{\gamma}{a}(\pi/2+\theta-\varphi')\right)d\varphi' \end{aligned}. \tag{16}$$

Если функция $\Psi(t)$ имеет компактный носитель, то при достаточно большом γ можно считать $\Psi\left(\frac{\gamma}{a}\left(\pi/2+\theta-\varphi'\right)\right)=0$ при $\varphi' \notin [-\pi,\pi]$. Тогда

$$F(a,\theta,\xi,\eta) = \frac{\gamma}{\sqrt{a}}\int_{-\pi}^{+\pi} f(\xi+\gamma\sin\theta+\gamma\cos\varphi',\eta-\gamma\cos\theta+\gamma\sin\varphi')\Psi\left(\frac{\gamma}{a}\left(\pi/2+\theta-\varphi'\right)\right)d\varphi'. \tag{17}$$

То есть направленное вейвлет-преобразование может быть представлено в виде ВКПР, причем для разных углов θ и масштабов a меняется только весовая функция Ψ.

Для выполнения направленного вейвлет-преобразования для одного угла θ и масштаба a использование ВКПР не оправдано. Действительно, при использовании Фурье-коррелятора требуется N^2 умножений и одно БПФ (порядка $N^2\log_2 N$ операций). При использовании ВКПР с весом используется линейная комбинация из P КПР разных порядков (то есть около $2PN^2$ операций), где P – число используемых КПР-образов, по которым разложено исходное изображение. Например, при обработке изображения 512x512 отсчетов и использовании 21 КПР-образов (с -10 по 10 порядки) использование коррелятора требует порядка $10N^2$ операций, а использование ВКПР требует около $42N^2$.

Однако при использовании набора углов θ можно использовать более быстрые алгоритмы. Заметим, что при использовании ВКПР с весом $\mu'(\varphi)=\mu(\varphi-\alpha)$ коэффициенты разложения в формуле (12) будут иметь вид:

$$\mu'_n = \frac{1}{2\pi}\int_0^{2\pi}\mu'(\varphi)e^{in\varphi}d\varphi = \frac{1}{2\pi}\int_0^{2\pi}\mu(\varphi-\alpha)e^{in\varphi}d\varphi = e^{in\alpha}\mu_n. \tag{18}$$

Легко заметить, что при $\alpha=\pi$ $\mu'_n=(-1)^n\mu_n$, то есть можно реализовать вычислительно более эффективную схему. Рассмотрим множества

чисел, делящиеся на d с остатком r: $N_{dr} = \{n \in Z : n = dk + r, k \in Z\}$. Обозначим

$$S_{dr}(\xi,\eta) = 2\pi\gamma \sum_{n \in N_{dr}} (-i)^n \mu_n R_\gamma^{(n)}(\xi,\eta). \tag{19}$$

Тогда

$$\begin{bmatrix} R_\gamma^{\mu(\varphi)}(\xi,\eta) \\ R_\gamma^{\mu(\varphi-\pi)}(\xi,\eta) \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} S_{20}(\xi,\eta) \\ S_{21}(\xi,\eta) \end{bmatrix}. \tag{20}$$

Аналогично, умножение на степень числа i выполняется тривиально, поэтому можно проводить вычисления с расщеплением «по основанию 4»:

$$\begin{bmatrix} R_\gamma^{\mu(\varphi)}(\xi,\eta) \\ R_\gamma^{\mu(\varphi-\pi/2)}(\xi,\eta) \\ R_\gamma^{\mu(\varphi-\pi)}(\xi,\eta) \\ R_\gamma^{\mu(\varphi-3\pi/2)}(\xi,\eta) \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & i & -1 & -i \\ 1 & -1 & 1 & -1 \\ 1 & -i & -1 & i \end{bmatrix} \begin{bmatrix} S_{40}(\xi,\eta) \\ S_{41}(\xi,\eta) \\ S_{42}(\xi,\eta) \\ S_{43}(\xi,\eta) \end{bmatrix}. \tag{21}$$

Данный алгоритм более эффективен по сравнению с Фурье-корреляторами для каждого угла и масштаба. Например, при использовании алгоритма с расщеплением «по основанию 4», когда изображение имеет размеры 512x512 отсчетов, используются КПР-образы с -10-го по 10-й порядки и число углов θ равно 32, мультипликативная сложность составляет $168N^2$ операций, в то время как при использовании Фурье-корреляторов около $32\left(N^2 + \frac{2}{3} N^2 \log_2 N\right) = 224N^2$.

Параллельное выполнение направленного вейвлет-преобразования с помощью взвешенного кольцевого преобразования Радона

Использование Фурье-коррелятора для каждого угла θ и масштаба a может легко выполнятся параллельно. Можно, например, разделить вычисления для разных углов и масштабов по разным машинам. Выполнение направленного вейвлет-преобразования с помощью ВКПР также может вычисляться параллельно. Однако распределение вычислений по машинам должно быть не произвольным. Наиболее трудоемким этапом является выполнение КПР разных порядков $R_\gamma^n(\xi,\eta)$. Логично производить вычисления для разных порядков на разных машинах, но тогда для избежания избыточной пересылки данных при вычислении сумм (19) для всех $n \in N_{dr}$ $R_\gamma^n(\xi,\eta)$ должны вычисляться на одной машине. Например, в случае четырех машин алгоритм с расщеплением «по основанию 4» может быть следующим: k-я машина вычисляет $R_\gamma^n(\xi,\eta)$ для всех $n \in N_{4k}$. Затем та же машина вычисляет сумму $S_{4k}(\xi,\eta)$. Далее

все четыре суммы $S_{4k}(\xi,\eta)$, $k=\overline{0,3}$ пересылаются на одну машину, на которой по формуле (21) вычисляются $R_\gamma^{\mu(\varphi-k\pi/2)}(\xi,\eta)$.

Заключение

В данной работе рассмотрено направленное вейвлет-преобразование двумерных функций. Для более эффективного его выполнения предложено использовать кольцевое преобразование Радона как среднее по всем окружностям фиксированного радиуса на плоскости. Введены кольцевое преобразование Радона произвольного порядка и взвешенное кольцевое преобразование Радона. Предложена схема вычислений для набора углов и масштабов направленного вейвлет-преобразования.

Благодарность

Работа выполнена при поддержке международного гранта CRDF REC-SA-O14-02 и президентского гранта РФ НШ-1007.2003.1.

Литература

1. *Новиков Л.В.* Основы вейвлет-анализа сигналов // Учебное пособие. 1999.
2. *Lemarie P.-G., Meyer Y.* Ondelettes et bases helbertiennes // Rev. Mat. Iberoamericana, 1986. V. 1. P. 1-17.
3. *Толкова Е.И.* Wavelet-анализ изображений // Оптический журнал, 2001. Т. 68. №3. С. 49-59.
4. *Котляр В.В., Ковалев А.А.* Кольцевое преобразование Радона // Компьютерная оптика. 2003. №25. С. 126-131.
5. *Rullgard H.* An explicit inversion formula for the exponential Radon transform using data from 180 // Research Reports in Mathematics, Stockholm University, 2002.

КЛЕТОЧНАЯ МАШИНА

И.Н. Кожин, В.А. Воробьёв, Г.В. Лозинская

Поморский государственный университет, г. Архангельск

Введение

В статье даётся введение в клеточные автоматы, затем понятие клеточного автомата расширяется до клеточной машины, и описывается разрабатываемая авторами программа создания собственных клеточных машин.

Мало кто знает, что основоположник архитектуры современных ЭВМ, Джон Фон Нейман, помимо классической ЭВМ (куда входит вычислительное устройство – процессор и оперативная память, в которой располагается программа для работы процессора, и данные, которые он обрабатывает; программа же определяет, как будут обрабатываться

данные), разрабатывал альтернативную, параллельную архитектуру, соединив понятие «вычислительное устройство» и данные, с которыми система оперирует. При этом вычислители и данные пространственно распределены и образуют собой решающее множество различной конфигурации – плоскость, кольцо, тор, трёх- и более мерные кубы. Данные и вычислительные устройства собираются из одних и тех же структурных элементов. Такая архитектура эволюционировала в то, что мы теперь называем «Клеточные автоматы» (КА).

Чтобы представить себе КА, мысленно нарисуйте прямоугольную решётку, в каждой ячейке которой располагается *конечный автомат* (автомат с определённым количеством внутренних состояний и конкретным набором входных и выходных сигналов, причём в каждый момент времени выходные сигналы однозначно определяются входными и внутренними состояниями). Выходы каждого из этих автоматов напрямую соединены со входами своих соседей. То, какие ячейки соединены между собой, определяет *шаблон соседства*.

Более формально определение КА звучит так: клеточные автоматы являются дискретными динамическими системами, поведение которых полностью определяется в терминах локальных зависимостей.

Рассмотрим основные преимущества клеточного подхода в решении научных и практических задач.

Очевидное преимущество КА перед классической архитектурой – это *параллельность* выполняемых операций. При этом не все известные на сегодняшний день параллельные алгоритмы обладают свойством *масштабируемости* (когда увеличение количества процессоров ведёт к пропорциональному уменьшению времени расчёта задачи в целом). Клеточные же алгоритмы по определению обладают этим свойством.

Очередное достоинство клеточных алгоритмов в том, что создав, например, поле 100x100, и отладив наш алгоритм, мы можем дробить это поле сколь угодно много (пока у нас хватает процессоров или пока не достигнем любой необходимой погрешности), т.е. алгоритм и время вычислений не изменяются, а точность расчётов увеличивается.

Мелкозернистость. Мелкозернистость означает, что количество вычислений в каждой клетке не зависит от размера задачи. Действительно, для автомата каждой клетки не важно, сколько всего имеется ячеек в клеточном множестве, при увеличении размера клеточного множества количество вычислений в автомате не возрастёт.

КА по своей структуре близки к моделированию полей различной природы, жидкостей, газов. Действительно, рассмотрим, например любую жидкость – в ней каждая молекула (ячейка КА) сама «знает» (лишь на основе информации от ближайших молекул), как ей взаимодействовать с соседями – отталкиваться или притягиваться, диффундировать и

т.д. Таким образом, клеточная модель более *естественна* для близкодействующих динамических явлений.

Правила, по которым действует КА, значительно проще дифференциальных уравнений, необходимых для описания того же явления обычным непараллельным алгоритмом, что даёт возможность упрощать и удешевлять процессоры в клетках.

Наглядность. Следующее преимущество КА – лёгкость визуализации. Всегда можно сопоставить значения внутренних состояний клеток определённому цветовому диапазону. В этом случае человек одним взглядом может охватить всю картину в целом, например распределение температур при нагреве металла, зоны турбулентности, распределение плотности в пространстве, кроме того, запустив КА на выполнение, можно видеть, как меняется картина в динамике.

Применять клеточные алгоритмы в своей научной и профессиональной деятельности могут учёные и инженеры самых разных профессий. Рассмотрим основные сферы применения.

Моделирование физических, биологических, экологических, социальных и прочих процессов, в т.ч. *непрерывных* динамических систем, определенных уравнениями в частных производных. Сюда относятся, например, взаимодействие молекул вещества, газа, распространение вещества в среде, диффузия, моделирование лесных пожаров, распространения загрязнения в атмосфере, эрозия почвы.

В компьютерной графике и задачах искусственного интеллекта клеточные автоматы нашли своё применение в распознавании и обработке изображений. В частности, к обработке изображений относится, например, морфинг – плавное преобразование одной картинки в другую. На основе клеточных автоматов строятся *клеточные нейронные сети*.

Применяются клеточные автоматы и в области кодирования информации. Здесь их используют для помехоустойчивого кодирования, для шифрования, (когда на основе данной картинки, зная только шифрующее правило, можно раскодировать изображение) и для сжатия (если из менее заполненной картинки можно получить за некоторое количество тактов времени искомую, то можно хранить лишь первую картинку и правила её преобразования).

В математике уже существует большое количество мелкозернистых локально-параллельных алгоритмов, применимых для различных областей наук.

Итак, применение КА довольно широко, но существуют некоторые свойства, присущие КА по определению, которые сужают возможные сферы использования КА. К таким свойствам относятся:

- КА может иметь конечное количество внутренних состояний;
- шаблон соседства одинаковый по всему полю;

Ÿ алгоритм автомата жёстко задан и неизменен;
Ÿ изменения переменных у соседей.

В связи с вышеописанными недостатками КА авторами предлагается новая архитектура ЭВМ – *клеточная машина* (КМ). Основное отличие КМ от КА – то, что в каждом узле решающего поля расположен не автомат, а процессор, оперативная память и коммутационное устройство для связи с соседями и управляющим устройством. В оперативной памяти находится *изменяемая программа*, по которой работает процессор. Причём процессор может отправлять запросы на чтение и запись соседним узлам.

Основное преимущество КМ перед КА в том, что первая работает по *программе*, (алгоритмические возможности программы зависят от набора команд конкретного процессора – это может быть привычная программа с циклами, ветвлениями, подпрограммами и классами), в то время как КА работает по строго заданному *неизменяемому* алгоритму. В связи с присутствием оперативной памяти количество внутренних состояний теперь ограничено лишь её размером, и в принципе, о внутренних состояниях уже говорить не приходиться – теперь можно пользоваться привычными для программистов переменными.

Если клеточный метод моделирования и решения задач подходит для вашей профессиональной или научной деятельности, то следующий вопрос – как создать КА или КМ. Здесь есть два варианта – либо реализовать КА аппаратно, либо использовать его в откомпилированном виде на ПК для расчёта и для просмотра результатов визуализации. Рассмотрим второй путь, как наиболее простой в реализации. Для программирования поточным способом существуют средства разработки программ. Задачи для клеточного подхода приходится решать с помощью универсальных языков, т.е. для решения конкретной задачи пишется новая программа на каком-либо языке, что ставит определённые барьеры в использовании клеточного подхода – человек должен разбираться не только в изучаемом явлении, но и уметь довольно хорошо программировать. Поэтому авторы поставили перед собой задачу создать универсальный инструментарий для разработки, отладки, прогона и анализа результатов программ для КМ.

Создав и отладив подходящий для решения конкретной задачи КА или КМ (определив правила для клеток, топологию и пр.), в дальнейшем можно реализовать его аппаратно или использовать в откомпилированном виде на обычном компьютере.

В заключение кратко рассмотрим возможности нашей программы.

Данный программный продукт предназначен для учёных, инженеров, программистов, в деятельности которых может использоваться модель клеточного автомата.

Возможности:

- возможность задавать первоначальные конфигурации клеточного множества (параметры множества)
- топология
- тор, кольцо (вертикальное, горизонтальное), плоскость
- задание любого шаблона соседства
- отображение множества
- выбор отображения для клеток класса
- выбор отображаемых переменных и их количества
- масштабирование.

Выбор языка для создания клеточной программы, с которым вы привыкли работать[1]. Планируется также поддержка DVM-технологии.

Возможности программы для клетки:

- может быть синхронной и асинхронной с алгоритмом перебора, задаваемым пользователем;
- возможность обращаться к переменным из клеток-соседей в программе для данной клетки;
- запись рассчитанных множеств в файл с возможностью просмотра в любом направлении.

Литература

1. *Тоффоли Т., Марголус Н.* Машина клеточных автоматов // Москва: Мир, 1991.

2. *Воробьёв В.А.* Эффективность параллельных вычислений // Автометрия, 2001. №5.

3. *Воробьёв В.А., Лаходынова Н.В., Ерёмина Н.Л.* Отказоустойчивость однородных процессорных матриц // Томск: Изд. ТГАСУ, 2002.

4. *Ачасова С.М., Бандман О.Л.* Корректность параллельных вычислительных процессов // Нововсибирск: Наука, 1990.

АВТОМАТИЗИРОВАННЫЙ АНАЛИЗ ПАРАЛЛЕЛИЗМА ПРОГРАММ

Н.Е. Козин, В.А. Фурсов

Институт систем обработки информации, г. Самара

[1] На данный момент к компилятору предъявляются следующие требования: он должен генерировать объектный код формата Win32 PE с экспортируемыми функциями (проще говоря, файлы Win32 dll), но и это ограничение, в принципе преодолимо – наша программа тестировалась и с клеточными функциями, написанными на Pascal для ДОС и VBScript.

Введение

В последние годы усиливается внимание к использованию высокопроизводительной вычислительной техники в научных исследованиях и образовании. Однако построение параллельных вычислительных процессов, обеспечивающих достижение высокой производительности, является серьезной проблемой. Перенос обычной программы на многопроцессорную вычислительную систему связан со значительными трудностями, связанными с нахождением параллельных участков программы.

В настоящей работе рассматривается учебная программная система автоматизированного анализа, предназначенная для использования в учебных целях при проведении лабораторных и учебно-исследовательских работ.

Программная система разработана на основе методик формального анализа алгоритмов, описанных в работе [1]. Краткое описание этих методик приведено в разделе 2 настоящей работы.

1. Методики анализа параллелизма программ

В любой программе можно выделить два типа операторов: логические и арифметические. Логические определяют связи по управлению. Они задают состав и порядок выполнения арифметических операторов. Арифметические же операторы могут выполняться только в порядке, определенном информационными связями между ними. Возможность распараллеливания алгоритмов зависит не только от логической схемы алгоритма, но и от степени детализации операторов.

В любом случае степень детализации операторов должна учитывать архитектуру ВС. Определяющими при распараллеливании являются два фактора:

– информационная и логическая взаимосвязь операторов (использование одними операторами выходной информации других операторов);

– объем работ в каждом операторе.

Связанные с объёмом работ временные характеристики операторов t_i, i=1, N (где N – число операторов) называют их весом. Если ВС однородная, достаточно оценить время t_i, выполнения каждого оператора на одном процессоре.

При составлении информационно-логической граф-схемы алгоритма необходимо руководствоваться действительной зависимостью между операторами, обусловленной преемственностью информации.

Переход от обычной схемы алгоритма или программы к модели в виде граф-схемы дает более ясное представление о структуре алгоритма, его свойствах и возможности направленных преобразований.

Для удобства исследования и преобразования графов в рассмотрение вводят матрицу следования *S*. Пример матрицы следования представлен на рис. 1.

1										
2										
3	•									
4	•									
5				**1**						
6		**1**								
7			**1**							
8					•					
9					•					
10						**1**	**1**	**1**	**1**	

Рис. 1. Пример матрицы следования S

Вершине (оператору) *i* графа ставят в соответствие *i*-ю строку и *i*-й столбец матрицы. Элемент (*i*, *j*) этой матрицы будем отмечать знаком • или цифрой 1, если между операторами с этими номерами существует какая-либо связь (по управлению или информации). Для различения типа операторов там, где это необходимо, связи по управлению будем отмечать знаком •, а связи по информации – единицей (1) (см. рис. 1). Можно показать, что по графу без контуров может быть построена треугольная матрица следования с нулевыми элементами на главной диагонали, что упрощает некоторые задачи их анализа.

Для исследования матриц *S* необходимо отразить в них неявные связи между операторами, которые реально существуют и определяются задающими связями. Их введение необходимо, во-первых, для выявления контуров в графе. Кроме того, решение задач распараллеливания как раз и заключается в выявлении дополнительных, в явном виде не существовавших связей между операторами. Рассмотрим способы установления этих связей.

Если существуют задающие связи $\beta \to \gamma$ и $\gamma \to \delta$, но нет задающей связи $\beta \to \delta$, то дополнение вычислительной схемы такими связями не меняет результата решения задачи, а лишь подтверждает недопустимость определенного порядка следования операций. Связи, которые введены направленно внутри всех пар операторов, принадлежащих одному пути в графе G и не связанные задающими связями, образуют множество транзитивных связей. Транзитивные связи могут быть введены путем формальных преобразований матрицы *S* по следующему алгоритму: строки матрицы просматриваются последовательно сверху вниз. В каждой (*i*-й) строке просмотр элементов производится в порядке увеличения номеров столбцов. Если в очередном (*j*-м) столбце имеется

знак •, указывающий на существование связи по информации или управлению, одноименные элементы строк с номерами i и j складываются по правилу дизъюнкции:

$$\bullet \cup \bullet = \bullet,\ \bullet \cup 0 = \bullet,\ 0 \cup \bullet = \bullet,\ 0 \cup 0 = 0.$$

Возможности алгоритма по распараллеливанию можно также выявить, если использовать только информационные связи. Дело в том, что эти связи прямо указывают на то, что образующие их операторы не могут выполняться одновременно (параллельно). Такие операторы называются логически несовместимыми. Для выявления таких операторов используется матрица L – логической несовместимости, которая формируется следующим образом.

Вначале по исходной треугольной матрице S, в которой связи по управлению отмечены знаком •, а связи по информации – 1, формируются задающие связи логической несовместимости операторов по следующим правилам: последовательно просматривают столбцы j=1, M, (где M – число вершин в графе) матрицы S. Если очередной элемент (i_μ, j)=•, и ранее в этом столбце также были зафиксированы элементы (i_k, j)=•, k=1,μ-1, то все элементы в i_μ-й строке с номерами (по столбцам) совпадающими с номерами строк, в которых ранее встречался знак •, заполняются единицами до i_μ-1 -го столбца включительно.

Далее на основе заданных связей логической несовместимости определяется несовместимость для всех операторов схемы. Для этого вводятся транзитивные связи логической несовместимости по следующим правилам.

Последовательно просматривают строки треугольной матрицы следования S, дополненной транзитивными связями (нулевые строки пропускают). В очередной ненулевой i-й строке матрицы S анализируют множество ненулевых элементов. Из этого множества исключают номера операторов, образующих входы матрицы S, т.е. обнуляют элементы в столбцах, номера которых совпадают с номерами нулевых строк. С использованием оставшегося множества номеров операторов формируют строки i^*. Для этого объединяют по операции конъюнкции строки матрицы L с номерами, соответствующими номерам столбцов, в которых остались ненулевые элементы. Если ненулевой элемент в строке один, то строку i^* полагают равной строке матрицы L с номером равным номеру столбца ненулевого элемента в анализируемой строке. Далее формируют *новую* i-ю строку в матрице L на основе ее старого значения и вновь сформированной строки i^* с помощью операции дизъюнкции $i=i \cup i^*$. Наконец, i-й столбец матрицы L полагают равным вновь сформированной i-й строке.

При распределении работ между процессорами важно установить множество тех операторов, внутри которого имеет смысл решать задачу распараллеливания выполнения операторов. Для этого необходимо объединить информацию о логической несовместимости операторов и их информационно-логической связи по следующим правилам.

Вначале по матрице *S*, дополненной транзитивными связями, строят матрицу следования *S*' путем симметричного отображения элементов матрицы *S*, относительно главной диагонали (или сложением матрицы с ее транспонированной). Далее на сформированную матрицу S' накладывают матрицу *L*, дополненную транзитивными связями. Каждый элемент новой матрицы *M* – матрицы независимости, формируется из одноименных элементов матриц *S*' и *L* по правилу дизъюнкции. Нулевые элементы полученной матрицы *M* указывают множество тех операторов, каждый из которых при выполнении некоторых условий может быть выполнен одновременно с данным, т.е. он информационно или по управлению не зависит от этого оператора и не является с ним логически несовместимым.

Матрица независимости M (см. рис. 2) отражает информационно-логические связи между операторами без учета их ориентации, а также связи логической несовместимости операторов с учетом всех транзитивных связей.

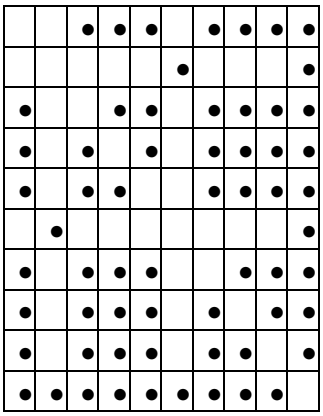


Рис. 2. Матрица независимости M
для матрицы следования S

При решении задач распараллеливания необходимо перебирать все возможные полные множества ВНО и находить максимально полное множество ВНО, содержащее максимальное число операторов. Для его нахождения используется тот факт, что если два оператора μ и ν взаимно независимы, то при сложении по правилам дизъюнкции μ-й и ν-й строк матрицы независимости M образуется строка, в которой элементы в μ-м и ν-м столбцах обязательно нулевые.

Ранние и поздние сроки окончания выполнения операторов удобно определять с использованием матрицы следования. Общая процедура анализа графа со скалярными весами вершин t_j строится в виде следующей последовательности шагов. Положив $\tau_{11} = \tau_{12} = ... = \tau_{1m} = 0$, последовательно обрабатывают строки матрицы *S*. Если очередная (*j*-я) строка не содержит единичных элементов, полагают $\tau_{1j} = t_j$. Если *j*-я строка содержит единичные элементы, из множества $\{\tau_{11},...,\tau_{1m}\}$ выбирают подмножество элементов $\{\tau_{1j_\nu}\}$, $\nu = \overline{1,k_j}$, соответствующих номерам единичных элементов *j*-й строки. В случае, когда все они отличны от нуля, полагают:

$$\tau_{1j} = \max_{\nu=1,...,k_j} \tau_{1j_\nu} + t_j .$$

Если же среди элементов множества $\{\tau_{1j_\nu}\}$, $\nu = \overline{1,k_j}$ есть хотя бы один нулевой (т.е. данное значение раннего срока окончания выполнения оператора еще не определено), переходят к следующей необработанной строке.

2. Описание программной системы автоматизированного анализа

При разработке программы авторы преследовали, прежде всего, учебные цели. Тем не менее, описываемая методика и программный комплекс (который далее мы будем называть «анализатор») могут оказаться полезными и при анализе параллелизма программ, используемых при проведении научных исследований.

Нетрудно заметить, что описанные выше правила формирования матриц следования и независимости сами являются алгоритмами и, следовательно, могут быть реализованы программно. Использование инструментальных средств автоматизированного анализа последовательных программ позволяет существенно ускорить написание параллельной программы. Ниже приводится описание программы и методики автоматизированного анализа параллелизма алгоритмов.

Анализатор обрабатывает алгоритмы, написанные на языке С. На основе текста программы нумеруются отдельные блоки операторов, а также выявляются их явные и неявные связи по управлению и информации. После этого выводятся матрицы следования, включающие и не включающие транзитивные связи, матрица информационной зависимости и итоговая матрица зависимостей.

Рабочая область программы состоит из областей для просмотра исходного текста алгоритма, области отображения пронумерованных блоков операторов, а также области отображения матриц. Диалоговое окно программы представлено на рис. 3.

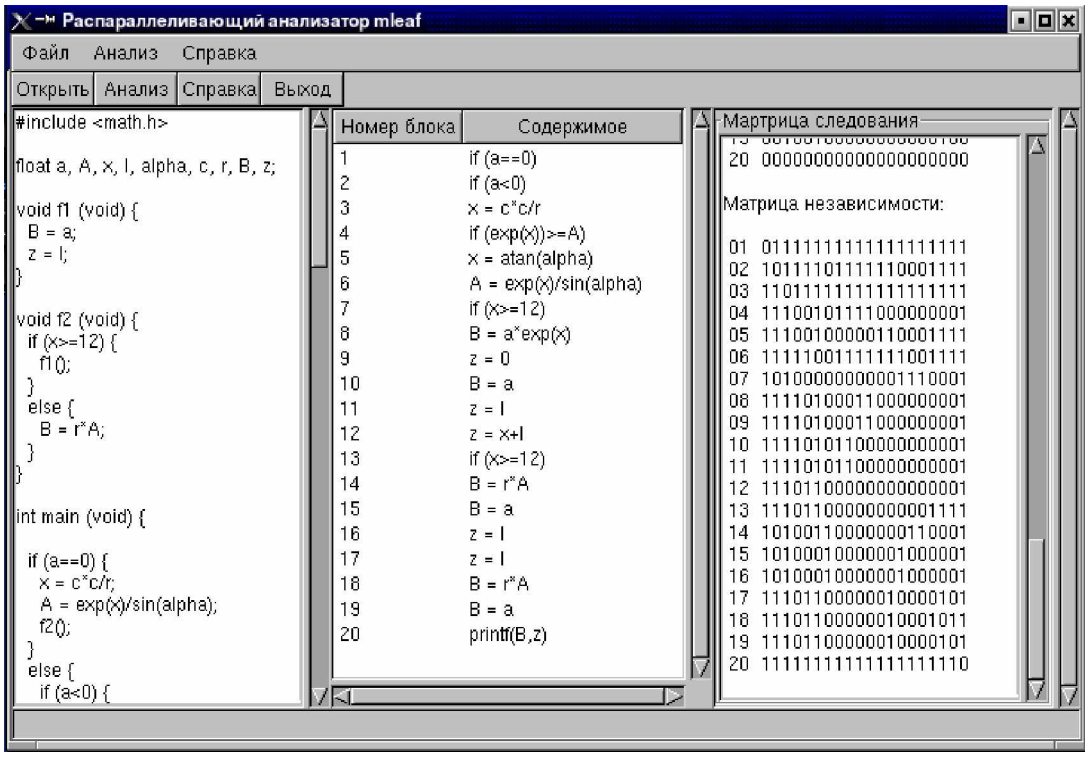


Рис. 3. Диалоговое окно

После построения матриц следования пользователь может получить информацию о получившихся полных множествах и сохранить её в файле. В настоящее время анализатор пока не поддерживает автоматическое создание исходного текста параллельной программы. Тем не менее, его использование существенно ускоряет процесс написания параллельной программы в автоматизированном режиме.

На первом этапе анализатор выявляет связи по информации между блоками операторов программы и после этого нумерует их таким образом, что получившаяся матрица следования *S* принимает треугольный вид. Далее она дополняется транзитивными связями. После этого программа нисходящим движением по графу выявляет несовместимость блоков операторов и строит матрицу логической несовместимости *L*, которая также дополняется транзитивными связями и симметрично отображается относительно главной диагонали. Итоговая матрица независимости *M* получается сложением предыдущих двух матриц по правилу дизъюнкции, предварительно матрица следования также симметрично отображается относительно главной диагонали.

В результате реализации описанной технологии анализа последовательной программы осуществляется автоматическая нумерация неделимых блоков операторов.

Далее формируются матрица следования *S'*, дополненная транзитивными связями, матрица логической несовместимости *L* и итоговая матрица независимости M, являющаяся объединением двух предыду-

щих. С использованием полученной матрицы независимости, путём сложения строк по правилу дизъюнкции, можно выделить множества независимых операторов. Любой блок из каждого множества независимых операторов может выполняться параллельно с другими блоками из данного множества.

С теоретической точки зрения целесообразно выделять множества с наибольшим количеством элементов (по возможности полные множества). Однако иногда использование оптимального решения может быть нецелесообразным. Например, небольшое дробление полных множеств значительно упрощает написание алгоритма и улучшает наглядность исходного текста.

На основе матрицы независимости легко строится ярусно-параллельная форма оптимизированного алгоритма. Число процессов в нём будет равно размерности полного множества, имеющего максимальное число элементов.

Особо следует отметить правила обработки вызовов функций. В случае, когда функция является библиотечной, её вызов считается отдельным неделимым оператором. Если же анализатор находит тело функции, то на место её вызова вставляется программный код данной функции. Таким образом, функция будет «развёрнута» столько раз, сколько имеется её вызовов.

Описанный анализатор пока имеет ограниченные возможности. В настоящей версии отсутствует возможность анализа конструкций, содержащих следующие операторы: цикла ***for***, безусловного перехода ***goto*** и безусловного выхода из блока ***break***. Распараллеливание циклических конструкций представляет собой отдельную задачу. Использование же оператора ***goto*** приводит к появлению ненулевых элементов на главной диагонали описанных матриц, которые в настоящей работе не рассматриваются.

В докладе будут подробно рассмотрены примеры распараллеливания конкретных задач с помощью описанного программного комплекса.

Благодарности

Работа выполнена при поддержке российско-американской программы «Фундаментальные исследования и высшее образование» и РФФИ (грант № 03-01-00109).

Литература

1. *Барский А.Б.* Параллельные процессы в вычислительных системах // Москва: Радио и связь, 1990.

2. *Тоффоли Т., Марголус Н.* Машины клеточных автоматов // М.: Мир,1991.

3. *Воеводин В.В., Воеводин Вл.В.* Параллельные вычисления // СПб.: БХВ-Петербург, 2002.

4. *Фурсов В.А., Шустов В.А., Скуратов С.А.* Технология итерационного планирования распределения ресурсов гетерогенного кластера // Труды Всероссийской научной конференции.

MPI: СТАНДАРТ И РЕАЛИЗАЦИОННАЯ ПРАКТИКА

А. Коновалов[*], А. Курылёв[**], А. Пегушин[*]

[*]ЗАО «Интел АО»,

[**]Нижегородский Государственный Университет им. Лобачевского, г. Нижний Новгород

Версия 1.0 стандарта MPI была принята в 1994-м году. Это был значительный шаг от «ни с чем не совместимых, и никуда не переносимых» коммуникационных интерфейсов первого поколения масспараллельных суперЭВМ. За прошедшие 10 лет стандарт стал «средством по умолчанию» индустрии высокопроизводительных вычислений: по-видимому, продать параллельную машину с неработающим MPI невозможно, и, с другой стороны, пользователь, MPI не использующий, обычно объясняет, почему он выбрал другое средство распараллеливания. Большой опыт, накопленный как в прикладном программировании, так и в реализации MPI, позволяет указать некоторые нетривиальные пункты, где намерения разработчиков стандарта разошлись с реальностью. Необходимо отметить, что зачастую это не ошибки проектирования, а проявляющиеся таким неожиданным образом ограничения современных вычислительных архитектур.

Совмещение счёта и обменов

Прикладной программист может бороться с высокими коммуникационными задержками на нескольких уровнях. Самое замечательное, если к задержкам нечувствителен алгоритм, например, число операций на один обмен достаточно велико. Общеизвестно, что совмещение счёта и обменов - стандартный «технический» способ создания параллельных задач, терпимых к высоким коммуникационным задержкам. Однако реализационная реальность существенно расходится здесь с лозунгом «совмещайте счёт и обмены - и ваши каналы станут бесконечно быстрыми». Как легко увидеть, 3 популярные свободные реализации MPI (MPICH, MPICH-2 и LAM) устроены одинаково и совсем не так. А именно, для достаточно больших сообщений применяется так называемый механизм progress engine, когда фоновые обмены происходят в результате синхронных вызовов процедур доставки. Таким образом, если

счётная часть программы не вызывает функций MPI вовсе, никакого совмещения счёта и обменов не произойдёт.

В работе [Лацис] проблему предлагается решать за счёт введения специальной нити, выделенной для обеспечения коммуникаций. По нашему мнению, окончательное решение этой проблемы возможно лишь если будет задействован интеллектуальный потенциал устройств ввода-вывода, ведь зачастую даже гигабитная EtherNet карточка может использоваться как универсальный процессор [EMP]. Значительным шагом здесь представляется стандартизация интерфейсов таких устройств (см. http://www.datcollaborative.org/). К сожалению, положенная проектировщиками в основу метафора удалённой очереди уже сейчас демонстрирует свою ограниченность при реализации таких важных операций, как широковещание или редукция.

Ещё одной иллюстраций неоднозначности положения с совмещением счёта и обменов является область коллективных операций. Как известно, коллективные операции в стандарте MPI-1 - только блокирующие. MPI-2 попытался снять это искусственное ограничение за счёт введения «обобщённых запросов», что, по мысли авторов, позволяет реализовать такие операции на пользовательском уровне. Легко видеть, однако, что для случая, например, аппаратуры, поддерживающей широковещание, никакого решения проблемы не произошло.

Параллельный ввод-вывод

Практика показывает, что существуют 3 группы задач, для которых значим высокопроизводительный ввод-вывод:

- задачи, создающие контрольные точки для счёта с продолжением,
- задачи, которым требуется большой объём временной памяти,
- источники больших объёмов данных для дальнейшей визуализации.

Понятно, что, особенно для 1 и 3, препятствием для совмещения записи и счёта будет только наличие свободных буферов, т.е. условия близки к идеальным. Неплохо смотрится здесь и MPI-2 – в отличие от, например, односторонних коммуникаций, работающая реализация стандарта (ROMIO) появилась одновременно со стандартом. К сожалению, ситуация в области совмещения счёта и обменов вновь нерадостная. А именно, ставшая стандартом в мире свободного ПО ROMIO, никакого совмещения счёта и обменов не представляет, совершенно синхронно и API популярной параллельной файловой системы PVFS. Здесь необходимо добавить, что развитие ROMIO фактически заморожено.

Управление процессами

Важным новшеством MPI-2 стала возможность динамического порождения процессов. Реализации этой возможности, по крайней мере,

свободные, пока ещё не достигли уровня, когда их стоит критиковать. Но, на наш взгляд, важная проблема содержится в самом стандарте. И действительно, если обмен данными – внутреннее дело, в основном, параллельной задачи, то управление процессами требует существенного участия операционного окружения. Понятно, что его стандартизация – не дело MPI, но унифицированный язык запросов оказался бы очень кстати. Многообещающей в этом контексте выглядит работа [Gropp] по интерфейсам менеджеров процессов.

На эту же проблему можно посмотреть и с другой стороны, если обратиться к вопросам безопасности. В самом деле, любая развитая система коллективного использования суперЭВМ должна исчерпывающе решать эти вопросы. Нет никакого сомнения, что соответствующие компоненты реализаций MPI-2 нуждаются в серьёзной доработке в этой области. Но даже после такой доработки останется проблема универсальности, ведь система прохождения задач должна обеспечивать работу не только конкретной реализации MPI-2.

Тестирование асинхронных реализаций

Известно, что полноценный стандарт образует не только собственно текст стандарта, немаловажную часть составляет базовая реализация (reference implementation) и общедоступные наборы тестов, которым доверяет соответствующее профессиональное сообщество. К сожалению, анализ доступных для MPI тестовых наборов (см., например, http://www-unix.mcs.anl.gov/mpi/mpi-test/tsuite.html) показывает, что именно в области тестирования корректности фоновых операций проверяется лишь малая часть проблемных мест. В то же время, ошибки такого сорта являются крайне коварными, т.к. проявляются на крупномасштабных реальных задачах. Стандарт MPI известен своим большим объёмом, поэтому ручная реализация необходимых проверочных средств не выглядит привлекательной. Суть системного программирования – в ориентации на создание средств решения проблем, а не на непосредственное их решение; в данном случае это означает, что существует потребность в программном средстве, обеспечивающем автоматическую генерацию требуемых тестов. Это – также одно из направлений, где приложение сил оказывается востребованным.

Литература

1. *Лацис* Как построить и использовать суперкомпьютер // М.: изд-во Бестселлер, 2003. - 274 с.

2. *Butler, Gropp, Lusk* Components and Interfaces of a Process Management System for Parallel Programs // Workshop on Clusters and Computational Grids for Scientific Computing, Sept. 24-27, 2000, Le Chateau de Faverges de la Tour, France.

3. *Shivam, Wyckoff,* Panda EMP: Zero-Copy OS-Bypass NIC-Driven Gigabit Ethernet Message Passing // Proc. of the ACM/IEEE SC2001 Conference (SC'01), Nov. 10-16, 2001, Denver, Colorado.

УПРАВЛЕНИЕ КОНФИГУРАЦИЯМИ И ЗАГРУЗКОЙ ВЫЧИСЛИТЕЛЬНЫХ СИСТЕМ

Д.В. Котляров

МЭИ(ТУ), г. Москва

Введение

Как показано в [1] следующие две задачи имеют принципиальное значение при построении эффективных стратегий управления параллельными процессами на вычислительных системах (ВС): задача управления конфигурацией ВС и задача управления загрузкой ВС.

Описывается подход к решению этих задач, и разрабатываемые для этой цели программные средства, которые являются составной частью проекта «Разработка системы граф – схемного потокового параллельного программирования для кластерных ВС», реализуемого на кафедре прикладной математики МЭИ (научный руководитель проф. В.П. Кутепов) [2, 3].

Управление конфигурацией ВС

Определим конфигурацию в узком смысле как структуру ВС, отражающую имеющие самостоятельное значение элементы ВС (компьютеры, каналы связи, организация памяти и т.п.) и связи между ними.

Будем структуру ВС обозначать *S(t)*, где переменная *t* – момент времени, в который структура имеет место.

Конфигурация в широком смысле это пара:

$K(t)=<Y(t), S(t)>$,

где *S(t)* – структура ВС или конфигурация в узком смысле, а *Y(t)* – схема управления, которая реализуется в *S(t)* в момент времени *t*.

Управление конфигурациями – это функция *F*, осуществляющая отображение множества возможных (допустимых для рассматриваемых структур) схем управления *Y* в множество возможных структур *S BC*:

$F: Y -> S$.

Рассмотрим множество допустимых (возможных) структур *S*. *S* может включать только структуры полученные добавлением или удалением одного или нескольких компьютеров, при сохранении общей организации ВС, в частности её коммуникаций (обычно подобные изменения связывают с понятием масштабирование). При более общих допущениях

структурные изменения могут включать не только количественные изменения компьютеров в ВС, но и коммуникационных узлов, коммуникационных связей, блоков памяти и её организационной структуры и т.д.

Множество схем управления *Y* может включать помимо традиционных моделей управления – централизованных, децентрализованных, иерархических и другие модели, например, основанные на многоагентной парадигме построения управления. Выбор той или иной схемы управления зависит от класса рассматриваемых на ВС задач и критериев оценки эффективности работы ВС.

Управление конфигурированием ставится как задача нахождения такой функции *K*, которая обеспечивает максимальный эффект при выполнении заданного класса задач на ВС (например, времени выполнения задач этого класса, использования ресурсов ВС, возможно, и того и другого, также, возможно, ещё других дополнительных критериев).

В практическом плане решение этой задачи предполагает разработку соответствующих алгоритмов и программных средств, которые позволят учитывать изменения, естественно возникающие в процессе функционирования ВС из-за неизбежных отказов и восстановлений или, в более общем случае, из-за необходимости динамического управления загруженностью ВС в процессе решения задачи (в частности, их параллельной работы). Так что алгоритмы управления конфигурациями в определённые моменты времени варьируют и структуру системы, и схему управления вычислительными процессами в ней.

В рамках выполняемого проекта разработаны программные средства, которые позволяют динамически управлять конфигурацией кластерных ВС [4].

В реализации эти программные средства представляют собой в определённом смысле надстройку имеющимися стандартными средствами ОС (в частности, сетевых ОС), предназначенными для этой цели.

В них разработанные программные средства взаимодействуют, с одной стороны, с блоками управления загрузкой кластера и блоком отказоустойчивости [1, 2]. С другой стороны, они «реагируют» на все события, обнаруживаемые средствами ОС (отказы и восстановления), в том числе, той её части, которая ответственна за администрирование кластера.

Программные средства позволяют визуализировать процессы изменения конфигурации кластера, что важно с позиции «человеческого» вмешательства в этот процесс.

Управление загрузкой кластера

Одной из целевых функций, определяющих эффективность работы ВС, является достижение равномерной загрузки ее компонентов, в частности компьютеров.

Для решения этой задачи в системе управления кластерными ВС [1, 2, 3] предусмотрен программный блок, который по данным, измеряемым компьютерами ВС, и данным об их загруженности пытается перераспределить выполняемые процессы таким образом, чтобы достичь ее равномерного распределения.

Три проблемы возникают при этом:

- как измерять и какие параметры загруженности компьютеров;
- каким образом определять необходимость перемещения процессов между компьютерами или, иначе, какие компьютеры считать перегруженными, а какие недогруженными;
- какие процессы стоит перемещать между компьютерами, если в этом возникает необходимость.

Третья проблема требует тонкого учета «поведения» процессов, использования ими ресурсов компьютера. Эту проблему решает планировщик во взаимодействии с блоком выполнения – интерпретатором параллельных программ [2].

Остановимся на рассмотрении первых двух проблем. Начнем с параметров, по которым можно достаточно объективно судить о загруженности компьютера. Наиболее важными из них являются:

– количество процессов,

– объем свободной памяти,

– интенсивность обменов между дисковой и оперативной памятью,

– интенсивность обмена данными с другими компьютерами.

Второй и третий параметры взаимосвязаны, по ним можно судить (особенно по третьему параметру) о непроизводительном времени, которое процессор компьютера тратит на обмен между оперативной и дисковой памятью (напомним, что среднее обращение к дисковой памяти в современных компьютерах при страничном обмене равно 1,5 мс – среднему времени полуоборота дискового носителя).

В то же время измерение свободной памяти компьютера реализуется гораздо проще и требует гораздо меньших накладных расходов по сравнению с измерением интенсивности обменов между уровнями памяти. Известно, что в ранних компьютерах со страничной памятью задача управления страничным обменом была центральной.

Интенсивность обмена между компьютерами позволяет судить о загруженности каналов сетевого обмена ВС и косвенно показывают насколько часто взаимодействуют процессы, размещённые на различных компьютерах.

Параметр – количество процессов на компьютере полезен, когда решается вопрос о простаивающих компонентах ВС и возможности перемещения части процессов. Хотя он менее информативен с позиций

регулирования загрузки по сравнению с другими перечисленными параметрами.

Заметим, что в ОС (например, Windows) есть специальные программные средства для измерения указанных параметров.

Для оценки своей загруженности локальное управление компьютером строит некоторый функционал (интегральную оценку, используя , например, линейный функционал, в котором параметры загруженности взвешены подобранными коэффициентами их важности). Одновременно вычисляется загруженность компьютера по памяти и по обмену с другими компьютерами, применяя нечёткую шкалу: недогружен, нормально загружен, перегружен.

На основе этих данных о загруженности компьютеров ВС управление загрузкой кластера пытается её сделать равномерной, сравнивая по достаточно сложной системе эвристических правил степень загруженности подчиненных компьютеров.

Схема на рис. 1 иллюстрирует эту модель управления загрузкой кластера и представляет самостоятельный блок в реализации управления, предназначенного для эффективного выполнения граф – схемных потоковых параллельных программ на кластерах [1].

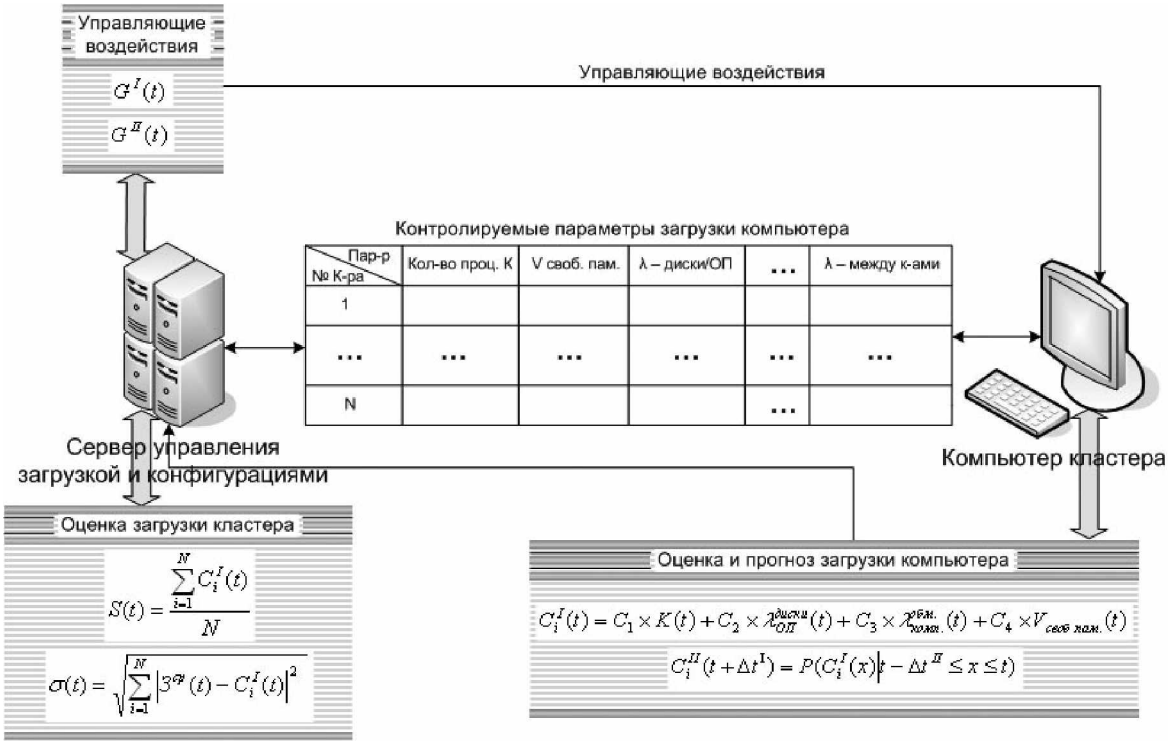


Рис. 1. Схема управления загрузкой кластера

Сервер управления загрузкой и конфигурациями кластера получает от каждого *i*-го компьютера оценку его интегральной загруженности $C_i^I(t)$ в момент времени t и её прогноз $C_i^{II}(t+\Delta t^1)$ в следующий момент времени $t+\Delta t^I$, вычисляемый функцией P по изменению загрузки компьютера на интервале $[t-\Delta t^{II}, t]$: $C_i^{II}(t+\Delta t^1) = P(C_i^I(x) \mid t-\Delta t^{II} \le x \le t)$.

По этим данным, которые сервер периодически получает от компьютеров кластера, определяется средняя загрузка $S(t)$ и девиация $\sigma(t)$, которая часто позволяет достаточно точно судить степени разброса в загруженности компьютеров. В принципе сервер, также как и компьютеры, по их прогнозу о загруженности пытается предсказать изменения в загрузке кластера с тем, чтобы более точно управлять ею. Сервер вырабатывает для управления загрузкой и конфигурацией кластера управляющие воздействия, которыми являются две функции: $G^I(t)$ и $G^{II}(t)$, первая из которых определяет текущую конфигурацию кластера, а $G^{II}(t)$ - распределение процессов по его компьютерам.

Отметим, что в реализации сервера параметры, по которым компьютер определяет свою загрузку, также передаются серверу для того, чтобы администратор имел возможность их контролировать. Для упрощения контроля эти данные представлены в табличной форме с использованием стандартных средств визуализации ОС.

Заключение

В настоящее время завершены работы по разработке указанных программных средств управления конфигурациями и загрузкой кластера и начата их апробация на практике в рамках указанного во введении проекта.

Работа выполнена при поддержке РФФИ, проект 03-01-00588

Литература

1. *Кутепов В.П.* Об интеллектуальных компьютерах и больших компьютерных системах нового поколения // М: Теория и системы управления, 1996. №5.

2. *Кутепов В.П., Котляров Д. В., Лазуткин В.А., Осипов М.А.* Граф-схемное потоковое параллельное программирование и его реализация на кластерных системах // М: Программирование, 2004.

3. *Кутепов В.П.* Методы и инструментальные средства построения управления распределёнными системами. М.: 2004.

4. *Котляров Д.В.* Разработка программных средств контроля загрузки и управления конфигурациями кластерных систем // М.: 2004.

ИНТЕРПРЕТАЦИЯ ФУНКЦИОНАЛЬНО-ПАРАЛЛЕЛЬНЫХ ПРОГРАММ С ИСПОЛЬЗОВАНИЕМ КЛАСТЕРНЫХ СИСТЕМ

Д.А. Кузьмин, А.И. Легалов

Государственный технический университет, г. Красноярск

Введение

В работах [1, 2] предложен язык программирования «Пифагор», реализующий поддержку функциональной модели вычислений с управлением по готовности данных, обрабатываемых в бесконечных, динамически выделяемых ресурсах [3]. Для выполнения подобных программ на реально существующих параллельных вычислительных системах существует два основных подхода, каждый из которых обладает своими достоинствами и недостатками:

Преобразование функциональных параллельных программ (ФПП) в параллельные программы, соответствующие конкретной архитектуре. В этом случае возможно достижение производительности, близкой к характеристикам параллельной системы, если удастся обеспечить эффективность процесса трансформации. Однако, для получения результата необходимо тщательное исследование принципов перехода от одного вида параллелизма к другому. При этом не всегда удается получить эффективные эквивалентные представления для результирующих программ и приходится заниматься динамической эмуляцией в ходе вычислений.

Непосредственная интерпретация ФПП с использованием специально разработанного эмулятора. Несмотря на падение производительности, такой подход допускает более простую реализацию и позволяет достаточно быстро приступить к разработке и отладке параллельных программ. Он является более удобным при решении задач исследовательского характера. Помимо этого потеря производительности будет невысокой, если операциями будут являться достаточно крупные программные модули.

В рамках данной работы акцент сделан на разработку интерпретатора, обеспечивающего выполнение программ на кластерных архитектурах. Это связано с предполагаемым первоначальным использованием языка как средства динамического управления длительными по времени вычислительными процессами. Создание интерпретатора обусловлено решением следующих задач:

– исследование общих принципов построения и использования интерпретирующей среды для языка функционально-параллельного программирования «Пифагор»;

– анализ методов повышения производительности в зависимости от состава и конфигурации используемой кластерной системы.

В настоящее время реализован и используется интерпретатор функционального языка, обеспечивающий последовательное выполнение написанных программ [4]. Однако развитие системы требует дальнейших исследований принципов построения и использования средств, поддерживающих реальный, а не виртуальный параллелизм. Для проведения подобных работ имеются определенные предпосылки. В настоящее время широко используются кластерные системы, реализованные на основе обычных персональных компьютеров, взаимодействующих в рамках локальной вычислительной сети. Они легко создаются и масштабируются. Для организации кластерных систем разработаны программные пакеты, функционирующие под управлением различных операционных систем.

Среди существующих средств, обеспечивающих поддержку кластерных и распределенных вычислений можно выделить системе динамического распараллеливания Mosix [5], которая обеспечивает динамическое распределение процессов по узлам кластера. Порождение процессов осуществляется стандартными средствами ОС Linux, что позволяет запускать интерпретатор, как на кластере, так и на однопроцессорных системах, не использующих данный пакет. Помимо этого Mosix обеспечивает поддержку статического управления, что, при необходимости, позволяет явно накладывать ограничения на модель вычислений с целью достижения максимальной эффективности конкретной кластерной архитектуры.

Организация процесса интерпретации

Язык «Пифагор» позволяет описывать параллелизм задачи на уровне элементарных операций. Базовой операцией языка, является операция интерпретации, объединяющая функцию и обрабатываемые данные в единый операционный пакет, исполнение которого начинается по готовности того и другого. В идеальной ресурсно-независимой системе каждый такой пакет выполняется на собственном вычислителе. Реальное выполнение под управлением ОС Linux, с поддержкой динамического распараллеливания посредством Mosix осуществляется моделированием идеальных условий посредством использования механизма порождения процессов.

Отсутствие в современных архитектурах эффективной аппаратной поддержки параллельных вычислений на уровне элементарных операций определяет иерархическое построения интерпретатора функциональных программ. Таким образом, архитектура виртуальной машины, должна иметь многоуровневую структуру в рамках которой:

– низкоуровневые операции выполняются последовательно;

– крупные информационные единицы выполняются параллельно на уровне отдельных процессов, а в общем случае, на уровне потоков и процессов (реализация крупноблочного распараллеливания);

– возможно последовательное выполнение независимых процессов при наличии ресурсных ограничений.

Выполнение ФП программ осуществляется на множестве виртуальных машин, каждая из которых выполняет собственную функцию.

Использование процессов для параллельного выполнения функций требует решения ряда задач, связанных с передачей данных и получением результатов. Проблема заключается в том, что функция и данные, определяющие операционный пакет становятся известными только перед запуском операции интерпретации. Необходимо решить задачи, связанные с передачей составляющих операционного пакета от родительского процесса дочернему, динамически порождаемому во время вычислений. Вариант, связанный с пересылкой функций и обрабатываемых данных в моменты их формирования, особенно при рекурсивных вызовах, ведет к дополнительным временным задержкам. Альтернативой является явная передача процессу всех функций до начала его выполнения. Это решение может быть непосредственно поддержано моделью управления процессами ОС Linux. Каждый процесс наследует окружение процесса-родителя.

Наряду с проблемой передачей данных в интерпретируемый процесс, необходимо определиться с механизмами их доставки от потомков к родителям, обеспечивающими сборку результатов. В Linux системах имеются различные средства взаимодействия между процессами, которые поддерживаются Mosix. Сборка результатов вычислений осуществляется с использованием любого из них. Наиболее удобным представляется использование очередей, обладающих развитыми средствами синхронизации, на уровне операционной системы. Поимо этого, данные в очереди типизированы, что позволяет родительскому процессу идентифицировать: от кого из потомков они получены. Для предотвращения конфликта по доступу к общему ресурсу можно использовать семафоры.

Наличие ресурсных ограничений ведет к необходимости введения дополнительного управления, обеспечивающего видимость бесконечных виртуальных ресурсов. Это управление подчиняется определенному набору следующих правил:

– интерпретация начинается в момент создания конкретного процесса;

– интерпретирующий процесс может порождать новые процессы-потомки, неявно передавая им параметры, через механизм наследования, поддерживаемый на уровне операционной системы;

– процесс-потомок может порождать другие процессы интерпретации, тогда для передачи родителю результата вычислений, ему необходимо дождаться результатов от своих потомков;

– если процесс-потомок, при попытке выполнить следующую операцию интерпретации получает отказ, то выполнение функциональной программы продолжается в текущем процессе;

– если некоторый процесс-потомок начал передачу данных родительскому процессу, то ни один из потомков последнего не может начать передачу данных, до окончания передачи данных этим процессом.

Реализация интерпретатора

Интерпретатор реализован в виде исполняемого файла, запускаемого из командной строки, в которой передаются все необходимые параметры:

– Файл на языке XML с промежуточным представлением, созданный транслятором.

– Имя функции, запускаемой на интерпретацию.

– Параметры, определяющие уровень параллелизма.

Процесс интерпретации начинается с разбора xml-файла и создания объектного представления. В ходе выполнения для каждой операции интерпретации происходит анализ данных и функций на наличии параллельного списка. Если аргумент или функция являются таковыми, то управление передается модулю интерпретации, который принимает решение о параллельном запуске.

Интерпретация функциональной программы начинается с порождения необходимого количества процессов. В каждом из процессов запускается интерпретатор со своим аргументом (или функцией) из параллельного списка. Процессы получают сегмент данных от родительского процесса. Поэтому отпадает необходимость явной передачи функциональной программы порожденному процессу при отсутствии внутренних вычислений. Родительский процесс «засыпает» в ожидании результатов. Каждый дочерний процесс обрабатывает свою часть и передает результат родительскому процессу, после чего завершает свою работу. Для обмена результатами между родительским и дочерними процессами используется модуль сборки результатов. После получения всех необходимых результатов родительский процесс продолжает свою работу.

Повышение эффективности работы интерпретатора

Поддержка интерпретатором модели с бесконечными ресурсами на уровне элементарных операций обеспечила выполнение функциональных программ и позволила выявить узкие места, связанные с ресурсными ограничениями исполнительной среды (фиксированным чис-

лом порождаемых процессов, очередей сообщений, семафоров и др.). Это, в свою очередь, помогло осуществить тестирование и отладку интерпретатора, а также улучшить алгоритмы функционирования, что повысило эффективность и надежность системы в целом.

Вместе с тем, рассмотренная реализация не обеспечивает эффективное выполнение функциональных программ. Это также связанно с тем, что время создания параллельного процесса для элементарных операций много больше времени ее обычного последовательного выполнения. Для повышения эффективности необходимо обеспечить автоматическое ограничение числа процессов, связав их не с количеством операций, а с особенностями архитектуры ПВС.

Отсутствие дополнительных ограничений обеспечивает параллелизм на уровне, определяемым языком, который включает параллельное выполнение всех элементарных операций (арифметических, операций сравнения и др.). Если выбирается этот вариант распараллеливания, то процессы будут создаваться для каждой из элементарных операций. Возможны следующие варианты дополнительных ограничений:

Использование распараллеливания только на уровне независимых данных, являющихся аргументами операций интерпретации. Они должны быть параллельными списками. Функция, осуществляющая обработку этих данных, не должна быть элементарной.

Распараллеливание проводится только на уровне функций поступающих на функциональные входы операций интерпретации. Они должны быть представлены в виде параллельных списков, и не являться элементарными операциями.

Еще одним фактором, влияющим на выполнение функциональных программ, является глубина распараллеливания рекурсивных функций. Параллельная рекурсия будет эффективна в том случае, если число процессов, порожденных в ходе интерпретации, не будут превышать количества узлов кластера, а время выполнения каждого из этих процессов будет больше времени миграции порожденного процесса на свободный узел кластера. Параметр, ограничивающий глубину распараллеливания, может накладывать дополнительные ограничения на уровень распараллеливания.

Ввод параметров, ограничивающих распараллеливание и глубину параллельного выполнения рекурсивных функций до определенного уровня, позволяет настраивать интерпретатор на требуемый режим работы. Это обеспечивает более эффективное выполнение функциональных параллельных программ.

Заключение

Разработка интерпретатора обеспечила практическую поддержку экспериментов, связанных с исследованием параллельного выполнения

функциональных программ, написанных на языке «Пифагор». Проведенные эксперименты показывают, что даже при простых алгоритмах динамического распределения ресурсов использование кластерных систем позволяет повысить производительность вычислений. Вместе с тем, полученные данные позволяют определить направления дальнейших работ по повышению эффективности интерпретатора, улучшению транслятора и модификации языка программирования. Работа выполнена при поддержке РФФИ № 02-07-90135.

Литература

1. *Kuzmin D.A., Kazakov F.A., Legalov A.I.* Description of parallel-functional programming language // Advances in Modeling & Analysis, A, AMSE Press, 1995. Vol.28. №3. P. 1-17.

2. *Легалов А.И., Кузьмин Д.А., Казаков Ф.А., Привалихин Д.В.* На пути к переносимым параллельным программам // Открытые системы, 2003. № 5 (май). С. 36-42.

3. *Легалов А.И., Казаков Ф.А., Кузьмин Д.А. Водяхо А.И.* Модель параллельных вычислений функционального языка // Известия ГЭТУ. Вып. 500. Структуры и математическое обеспечение специализированных средств. С.-Петербург, 1996. С. 56-63.

4. *Легалов А.И.* Инструментальная поддержка процесса разработки эволюционно расширяемых параллельных программ // Проблемы информатизации региона. ПИР-2003/ Материалы 8-й Всероссийской научно-практической конференции. Красноярск, 2003. С. 132-136.

5. *Barak A., La'adan O.* The MOSIX Multicomputer Operating System for High Performance Cluster Computing // Journal of Future Generation Computer System, 1998. №13.

РАСПАРАЛЛЕЛИВАНИЕ В КЛАСТЕРЕ ПОЛУЭМПИРИЧЕСКИХ КВАНТОВО-ХИМИЧЕСКИХ МЕТОДОВ ПРИ ПРЯМОМ ВЫЧИСЛЕНИИ МАТРИЦЫ ПЛОТНОСТИ ДЛЯ БОЛЬШИХ МОЛЕКУЛЯРНЫХ СИСТЕМ

М.Б. Кузьминский, В.В. Бобриков, А.М. Чернецов, О.Ю. Шамаева

Институт органической химии РАН, г. Москва
Московский энергетический институт, г. Москва

Введение

Задачей работы было распараллеливание в кластере программы прямого построения матрицы плотности для полуэмпирических расчетов больших молекул методами нулевого дифференциального перекрывания (НДП). Актуальность работы определяется потребностями в теоретическом исследовании электронной структуры больших молекуляр-

ных систем, в т.ч. биомолекул (включая протеины и ДНК) и наноструктур. Практическое значение таких исследований связано, в частности, с задачами конструирования лекарств.

Применение неэмпирических методов расчета даже для небольших протеинов сегодня практически нереально. Например, расчет молекулы цитохром-С, содержащей 1738 атомов, по программе, специализированной для расчетов больших молекул, методом DFT в гетерогенном кластере с использованием 15 микропроцессоров Alpha 21x64, требует около суток на итерацию и свыше месяца – на весь расчет без оптимизации геометрии [1].

Не только строгие неэмпирические методы, но и даже более простые полуэмпирические методы при расчете сверхбольших молекул требуют неприемлемо высоких затрат вычислительных ресурсов. Для методов НДП время расчета растет как N^3, где N-размерность базиса, пропорциональная размеру молекулы. Это обусловлено необходимостью решения симметричной проблемы на собственные значения с нахождением всех собственных векторов, из которых затем строится матрица плотности.

Распараллеливание расчета матрицы плотности

В работе в рамках полуэмпирической схемы AM/1 использован альтернативный диагонализации численный метод прямого вычисления матрицы плотности с применением полиномов Чебышева [2]. При использовании технологии разреженных матриц этот метод обеспечивает линейное масштабирование времени расчета с ростом N. До настоящего времени распараллеленных программ, реализующих данный метод, в мире не имелось.

Авторами создана распараллеленная версия экспериментальной программы на языке Фортран-95, реализующей данный метод, с использованием средств MPI. При оценке минимального и максимального собственных значений использован метод Гершгорина. Для нахождения корня нелинейного уравнения, определяющего химический потенциал, применен метод Ньютона, а для расчета коэффициентов разложения матрицы плотности по полиномам Чебышева – подпрограмма dfcost из свободно доступной библиотеки FFTPACK [3], реализующая дискретное преобразование Фурье. Для умножения матриц (в первой версии программы технология разреженных матриц не использована) применяется BLAS-подпрограмма dgemm из библиотеки Atlas-3.6.0, также свободно доступной. Выбор данных программных средств обусловлен потребностями в разработке мультиплатформенных программных средств с открытым кодом.

Созданные программные средства работают в среде операционных систем Windows и Linux для x86 (использованы компиляторы Compaq Visual Fortran-6.1 и Intel ifc-7.1, соответственно). Однако тестирование было осуществлено в Linux-кластере на базе Myrinet 2000 и SMP-узлов, использующих процессоры Xeon/2.6 ГГц и материнские платы с набором микросхем iE7500 с оперативной памятью DDR266.

Для профилирования при измерениях процессорного времени выполнения применялся высокоточный таймер на основе счетчика RDTSC [4]. Найдено, что лимитирующей стадией расчетов является вычисление матричных полиномов Чебышева. Матрица плотности целиком хранится в памяти узлов, поскольку исходная полуэмпирическая программа не предполагает распределенного хранения. В программе реализовано поблочное умножение матриц, так что каждый процесс рассчитывает только свой блок, а затем осуществляется обмен рассчитанными блоками.

Результаты работы программы иллюстрирует пример молекулы полипептида, содержащей около 1800 атомов (N=4500), см. рис. 1. Достигнутое нами ускорение расчета матрицы плотности (без построения фокиана) по сравнению с последовательной программой составляет при 3 процессорах 2,3 раза, при 6 процессорах – 4,1 раза.

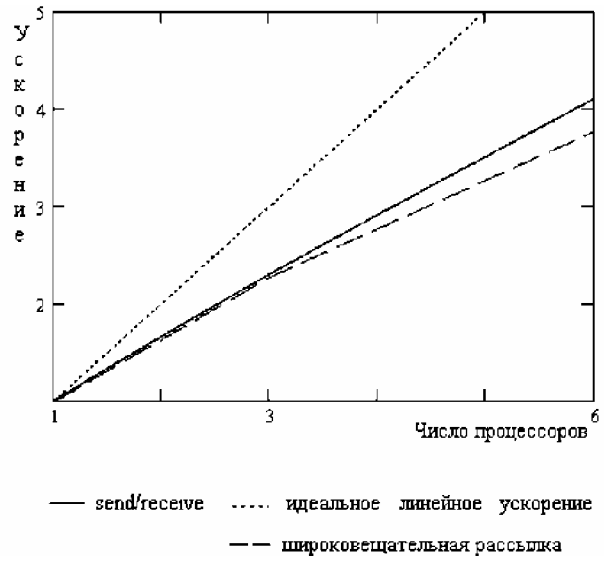


Рис. 1

Оценка предельного ускорения по закону Амдала при бесконечном числе процессоров близка к 11 (9% последовательных кодов).

Для сравнения, в лучшей на сегодня коммерческой полуэмпирической программе MOPAC2002, использующей другую линейно масштабируемую с ростом N численную схему на основе локализованных орбиталей, на многопроцессорном сервере SGI Origin 300 с процессорами R14000/500 МГц для молекулы $C_{960,}$ с близким размером базиса

(N=3840) оценки достигнутого ускорения близки к полученным нами. Они составляют 2,5 и 4,1 для 3 и 6 процессоров соответственно [5]. При этом процессоры в Origin 300 медленнее, чем используемые нами, а межсоединение узлов – наоборот, существенно быстрее, что повышает достигаемое в MOPAC2002 ускорение.

Приведенные результаты относятся к запуску на SMP-узлах кластера по 1 процессу. При запуске по 2 процесса на узел время расчета существенно возрастает, что связано, вероятно, с ограничениями пропускной способности системной шины и конфликтами процессоров по доступу в оперативную память.

Нами были протестированы также схемы обмена данными с использованием широковещательных рассылок, вместо применявшихся в указанных выше расчетах MPI_SEND/MPI_RECEIVE. Несмотря на довольно эффективную реализацию, MPI_BCAST в использованной в работе MPICH-GM v.1.2.4-8, суммарное время обменов в этом случае возрастает на 30% (для 6 процессоров).

Использованная схема распараллеливания отвечает крупнозернистому параллелизму, поэтому от межсоединения узлов кластера требуется в первую очередь высокая пропускная способность. Достигнутая нами в MPI_SEND/MPI_RECEIVE пропускная способность при передаче блоков матрицы составила около 210 Мбайт/с, что уже достаточно близко к пиковой величине 250 Мбайт/с.

Литература

1. *Sato F., Yoshihiro T., Era M., Kashiwagi H.* Chem. Phys. Letters, 2001. V. 341. P. 645.

2. *Goedecker S., Colombo L.* Phys. Rev. Letters // 1994. V. 73. P. 722.

3. http://netlib.org/fftpack

4. *Кузьминский М.Б., Мендкович А.С., Аникин Н.А., Чернецов А.М.* «Пути модернизации программных и аппаратных кластерных ресурсов для задач вычислительной химии» // Высокопроизводительные параллельные вычисления на кластерных системах. Материалы второго международного научно-практического семинара. Изд-во Нижегородского госуниверситета, Н. Новгород, 2002. С. 169.

5. SGI Computational Chemistry Applications Performance Report. Spring 2002, Silicon Graphics, Inc., 2002.

ФУНКЦИОНАЛЬНОЕ ПАРАЛЛЕЛЬНОЕ ПРОГРАММИРОВАНИЕ: ЯЗЫК, ЕГО РЕАЛИЗАЦИЯ И ИНСТРУМЕНТАЛЬНАЯ СРЕДА РАЗРАБОТКИ ПРОГРАММ

В.П. Кутепов, С.Е. Бажанов

Энергетический Институт (ТУ), г. Москва

Введение

Доклад посвящен описанию реализуемого на кафедре прикладной математики МЭИ проекту создания системы функционального параллельного программирования для кластерных систем [1, 2].

Проект содержит три независимые части: высокоуровневый язык композиционного функционального параллельного программирования, инструментальные средства программирования на нем и систему управления процессом параллельного выполнения функциональных программ на кластерных системах.

1. Язык FPTL

Язык FPTL (Functional Parallel Typified Language) – это язык функционального композиционного параллельного программирования [1]. Его отличительными особенностями являются:

- композиционный стиль программирования, в основе которого лежит универсальный набор операций композиции функций и их определение через системы функциональных уравнений;
- схемный характер задания функций, структурированность определений;
- возможность определения различных типов данных, строгая типизация и статический контроль типов;
- полиязычность, заключающаяся в возможности использования в нем функций, определяемых на других языках программирования;
- асинхронная вычислительная семантика, основанная на формальной модели свертывания и развертывания деревьев – состояний вычислительного процесса, индуцируемого при применении функции к аргументам.

Теоретическую базу языка FPTL составляют исследования по функциональной схемотологии и функциональным системам [3], которые обобщены в теории направленных отношений, объединяющей функциональный и логический стили программирования [4, 5].

Основными семантическими объектами, представляемыми в языке, являются данные, функции, функционалы и реляционалы – параметризованные функции и типы данных соответственно.

Функции в FPTL, следуя [1, 3], рассматриваются как типизированные соответствия между множествами (данными). В общем случае, в

Работа выполнена при поддержке РФФИ, проект 03-01-00588

качестве аргументов и значений функций в языке FPTL выступают кортежи произвольной длины $m \geq 0$, элементами которых могут быть синтаксически определенные объекты различной структуры.

Данные и функции в языке FPTL определяются в общем случае посредством системы реляционных или функциональных уравнений [1], которые трактуются как операторы «взятия» минимальной фиксированной точки или минимального решения (для соответствующих интерпретаций) для этих систем реляционных и функциональных уравнений.

В языке FPTL для построения функций используются следующие четыре простые операции композиции (интерпретация операции композиции задается как график функции, определяемой через эту операцию):

операция последовательной композиции •

$$f^{(m,n)} = f_1^{(m,k)} \bullet f_2^{(k,n)} \cong \left\{ (\alpha,\beta) \middle| \exists\gamma \left((\alpha,\gamma) \in f_1^{(m,k)} \wedge (\gamma,\beta) \in f_2^{(k,n)} \right) \right\};$$

операция конкатенации *

$$f^{(m,n)} = f_1^{(m,n_1)} * f_2^{(m,n_2)} \cong \left\{ (\alpha,\beta_1\beta_2) \middle| (\alpha,\beta_1) \in f_1^{(m,n_1)} \wedge (\alpha,\beta_2) \in f_2^{(m,n_2)} \right\};$$

операция условной композиции →

$$f^{(m,n)} = f_1^{(m,k)} \to f_2^{(m,n)} \cong \left\{ (\alpha,\beta) \middle| (\alpha,\beta) \in f_2^{(m,n)} \wedge \exists\gamma \left((\alpha,\gamma) \in f_1^{(m,k)} \right) \right\};$$

операция объединения графиков ортогональных функций ⊕

$$f^{(m,n)} = f_1^{(m,n)} \oplus f_2^{(m,n)} \cong \left\{ (\alpha,\beta) \middle| (\alpha,\beta) \in f_1^{(m,n)} \vee (\alpha,\beta) \in f_2^{(m,n)} \right\}.$$

Запись $f^{(m,\ n)}$ означает, что функция f имеет арность (m, n), $m \geq 0$, $n \geq 0$, m и n – длина кортежей входных и выходных значений функции.

Для задания рекурсивных функций используется специальный оператор минимальной фиксированной точки или минимального решения определенного вида системы функциональных уравнений. Параметризованные функции (функционалы) определяются системой функциональных уравнений, содержащей свободные переменные.

В качестве примера приведем программу вычисления факториала:

```
Functional Program Factorial
Scheme Factorial
{
Factorial = (id*<0>).eq -> <1> +
 (id*<0>).ne -> (id * (id*<1>).minus.Factorial). mult;
}

Application %factorial (6)
```

В этом примере функция вычисления факториала определена с помощью одного функционального уравнения, содержащего только встроенные в язык функции (*id* – тождественная функция, *eq* – функция

равенства и т.д.). Программа также содержит задание на вычисление функции (от аргумента 6).

1.1. Модель вычисления значений функций

Модель асинхронного вычисления значений функций определяет процесс вычисления как процесс преобразования деревьев [3]. На каждом шаге состояние вычисления представляется бинарным размеченным деревом особого вида. Вычисление значения функции представляется в виде последовательности состояний. Переходы из состояния в состояние определяются правилами преобразования деревьев, разделенными на два подмножества: правила развертывания и правила свертывания деревьев.

Данная модель является недетерминированной, поскольку в общем случае возможно применение нескольких правил к дереву – состоянию вычислений, и поэтому в зависимости от применяемого правила будут получаться различные последовательности вычислений. Также модель является параллельной, т.к. возможно одновременное применение нескольких правил к несвязным кустам дерева состояния. Источником параллелизма являются свойства операций $*$, $\rightarrow$, $\oplus$.

Важно отметить, что не всякий порядок применения правил преобразования состояний приводит к корректному вычислению значения функции. Например, такой случай имеет место, если при вычислении значения $(t_1 \oplus t_2)(d)$ сначала делается попытка вычисления значения $t_1(d)$, которое не определено и процесс вычислений продолжается бесконечно, а значение $t_2(d)$ определено. Таким образом, условием корректности реализации модели является параллельное (или квазипараллельное) вычисление значений функций, соединяемых операцией $\oplus$.

Кроме этого, модель предоставляет возможность повышения эффективности вычислений за счет вычислений с упреждением. Если имеется достаточное количество ресурсов, то при вычислении значения $(t_1 \rightarrow t_2)(d)$ можно значение $t_1(d)$ вычислять одновременно с t_2(d), стремясь максимально распараллелить процесс вычислений.

Эти особенности модели вычислений являются принципиальными при ее реализации на вычислительных системах и при разработке эффективных алгоритмов планирования параллельного выполнения функциональных программ.

2. Реализация языка FPTL

В реализации любого современного языка программирования непременно присутствуют два основных компонента: инструментальная среда, предназначенная для автоматизации процесса программирования на языке и операционные средства управления, предназначенные для эффективного управления процессом выполнения программы на соот-

ветствующих вычислительных установках (компьютерах, компьютерных системах, кластерах и т. п.) [1].

Важными аспектами в реализации языка FPTL, отличающими ее от реализации широко известных языков программирования (Лисп, Пролог, Паскаль, Си и др.), является включение в нее ряда механизмов разработки функциональных программ и управления процессом их выполнения, в частности механизмы планирования, которые позволяют существенно повысить эффективность этих процессов.

2.1. Инструментальная среда

Инструментальная среда состоит из двух частей. Одна часть отвечает за правильность программы: синтаксический и типовой контроль, верификация программы. Другая часть включает технологические средства, упрощающие процесс разработки программ: редактор программ, систему управления проектом, систему поддержки декомпозиции программ и др. Среди этих средств следует выделить средства анализа структуры программы и ее вычислительной сложности и средства осуществления целенаправленных эквивалентных преобразований программ. Эти два элемента, а также система верификации программ основаны на теоретических результатах работ [3, 5]. Опишем их немного подробнее.

Задачей системы верификации является поиск ошибок и доказательство правильности функциональных программ. Данная задача включает в себя следующие подзадачи:

- поиск противоречий в спецификациях,
- доказательство тотальности и функциональности программных функций,
- поиск ошибок в объекте верификации,
- доказательство соответствия объекта верификации и его спецификаций.

В указанных работах предложено исчисление эквивалентности функциональных программ, что дает возможность осуществлять эквивалентные преобразования программ. Целью таких преобразований может быть улучшение некоторых качеств программы, например:

- преобразование к форме с минимальным временем параллельного вычисления,
- преобразование к форме без повторных вычислений значений функций,
- преобразование к форме без вычислений с упреждением.

Также были предложены методы оценки вычислительной и структурной сложности программы. Методы оценки вычислительной сложности позволяют получить числовые характеристики сложности выполнения функциональных программ по сложностным оценкам элементарных функций, используемых при их написании. Методы оценки структурной сложности рассматривают функциональную программу с точки

зрения таких понятий как рекурсивность, цикличность, взаимная рекурсивность, вложимость. Существует также метод графического представления структурных характеристик схем функциональных программ.

2.2. Средства управления выполнением программ

Средства управления должны обеспечить эффективное выполнение функциональных программ, как на кластерных установках, так и на одном компьютере [1, 4]. Основой для реализации управляющих средств является модель асинхронного вычисления значений функций (см. выше). Сами управляющие механизмы должны обеспечить эффективную реализацию этой модели (управление индуцируемыми при выполнении функциональных программ параллельными процессами) на кластерных установках различной конфигурации и с различным количеством компьютеров в них, в том числе и на отдельном компьютере.

Предполагается, что вычислительная система, на которой должны выполняться функциональные программы, строится как масштабируемая кластерная система, основным строительным блоком которой является узел кластера. Узел кластера – хорошо сбалансированная по быстродействию компьютеров и пропускной способности каналов вычислительная система, в частности это может быть локальная сеть. Именно по такой схеме сегодня строятся все большие масштабируемые компьютерные системы, например, наиболее интересные из них SPP, SP, ASCI WHITE и др.

Задача сервера каждого узла – определение его загрузки в процессе выполнения функциональной программы и управление ею путем перемещения процессов между узлами вычислительной системы. Эта задача известна как задача межузлового планирования процессов с целью достижения баланса в их загрузке. С другой стороны, эту же задачу сервер решает, но уже применительно к компьютерам узла, которые ему подчинены. Все компьютеры узла при выполнении функциональной программы работают независимо по одной и той же схеме, реализуя процессы свертывания и развертывания деревьев, лежащие в основе модели асинхронного вычисления значений функций. При этом каждый компьютер узла имеет свой планировщик, алгоритм работы которого существенно учитывает семантику задания функциональных программ на FPTL, в частности семантику операций композиции функций.

Интерпретатор, работающий на отдельном компьютере, является базовым блоком системы управления выполнением программ. В одномашинном варианте реализации, которая выполнена на данный момент, интерпретатор включает в себя транслятор, систему типового контроля и систему выполнения (рис. 1).

Наиболее важная часть интерпретатора – система выполнения. Система выполнения непосредственно управляет процессом выполне-

ния функциональных программ и осуществляет моделирование процессов свертывания и развертывания деревьев.

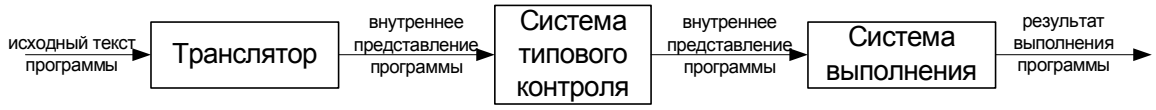


Рис. 1. Основные блоки интерпретатора

Модель вычисления по своей сути является параллельной и одной из основных задач реализации интерпретатора является обеспечение корректности реализации модели.

На каждом шаге выполнения программы дерево состояния вычисления содержит в общем случае множество кустов (или редексов), к которым можно применить правила преобразования. В силу последовательного характера процесса выполнения программы возникает задача планирования вычислений, т.е. выбора редекса для преобразования на каждом шаге. Эта задача непосредственно связана с обеспечением корректности и эффективности реализации модели.

Для решения задачи планирования вводится список редексов, динамически изменяемый в процессе выполнения программы. Планирование вычислений сводится к выбору одного из редексов из этого списка и добавлению новых редексов, появившихся в результате преобразования деревьев. Можно доказать, что организация работы с данным списком по принципу FIFO обеспечивает корректность реализации модели вычислений.

Для повышения эффективности вычислений используется два подхода.

Разбиение множества всех вычислений на подмножество вычислений, заведомо необходимых для получения конечного результата и подмножество вычислений, необходимость результатов которых зависит от результатов других вычислений. Сделав вычисления из первой группы более приоритетными можно заметно повысить эффективность выполнения программы.

Второй подход основан на соображении о том, что простые вычисления более приоритетны по отношению к сложным. Априорно можно предположить, что некоторые из преобразований деревьев являются сложными. Выполняя простые преобразования без откладываний можно быстрее осуществить свертывание дерева.

Заключение

Описанная выше версия интерпретатора реализована и успешно опробована. При сравнении с интерпретаторами других языков (Haskell, SML, Lisp, Prolog) он показал на одном компьютере схожие, а в отдель-

ном случае (в случае сложной рекурсивной программы) и лучшие результаты по времени.

Система управления параллельным выполнением функциональных программ для кластеров и находятся в стадии отладки. Результаты предварительных экспериментов, касающиеся эффективности работы системы управления, обсуждаются в докладе [7].

Литература

1. *Бажанов С.Е., Кутепов В.П., Шестаков Д.А.* Язык функционального параллельного программирования и его реализация на кластерных системах // Теория и системы управления (в печати).
2. *Бажанов С.Е., Кутепов В.П., Шестаков Д.А.* Разработка и реализация системы функционального параллельного программирования на вычислительных системах // Доклад на четвертом Международном научно-практическом семинаре «Высокопроизводительные параллельные вычисления на кластерных системах».
3. *Кутепов В.П., Фальк В.Н.* Функциональные системы: теоретический и практический аспекты // Кибернетика, 1979. №1. С. 46-58.
4. *Кутепов В.П.* Об интеллектуальных компьютерах и больших компьютерных системах нового поколения // Изв. РАН. Теория и системы управления, 1996. №5. С. 97-114.
5. *Кутепов В.П., Фальк В.Н.* Направленные отношения: теория и приложения // Изв. РАН. Техническая кибернетика, 1994. №4, 5.
6. *Кутепов В.П., Фальк В.Н.* Теория направленных отношений и логика // Изв. РАН. Теория и системы управления, 2000. №5.
7. *Кутепов В.П., Шестаков Д.А.* Анализ структурной сложности функциональных программ и его применение для планирования их параллельного выполнения на вычислительных системах // Доклад на четвертом Международном научно-практическом семинаре «Высокопроизводительные параллельные вычисления на кластерных системах».

ГРАФ-СХЕМНОЕ ПОТОКОВОЕ ПАРАЛЛЕЛЬНОЕ ПРОГРАММИРОВАНИЕ И ЕГО РЕАЛИЗАЦИЯ НА КЛАСТЕРНЫХ СИСТЕМАХ

В.П. Кутепов, Д.В. Котляров

МЭИ(ТУ) Кафедра ПМ, г. Москва

Введение

В докладе обсуждается реализуемый на кафедре прикладной математики проект создания системы граф – схемного потокового параллельного программирования для кластерных вычислительных систем [1].

Первая версия языка граф-схем, которая была создана в начале 70-х годов на волне активных исследований по параллелизму и параллель-

ным системам, задумывалась как «мягкое» развитие структурных (блочно-схемных) форм описания последовательных программ с предполагаемой реализацией на многомашинных и (или) многопроцессорных системах [1-3].

Следующие принципиального характера требования ставились при формулировке языка:

– модульный принцип построения параллельной программы, причем с возможностью программирования модулей как многовходовых-многовыходовых процедур на последовательных языках программирования;

– наглядное граф-схемное описание структуры параллельной программы в виде двух компонентов: граф-схемы и интерпретации, однозначно сопоставляющей каждому модулю (блоку граф-схемы) подпрограмму (или процедуру) на соответствующем языке программирования;

– интерпретация связей между модулями как связей по данным (а не по управлению), правда, с таким ограничением, чтобы у корректных, т.е. однозначных граф-схем [4] на каждом входе модуля не накапливалось более одного данного (естественное наследование принципа выполнения любой команды или оператора последовательной программы);

– экспликация параллелизма как информационной независимости различных модулей граф-схемы.

Уже первая попытка реализации такого граф-схемного модульного языка параллельного программирования [5] на многомашинных комплексах М6000/М7000 по принципу децентрализованного управления оказалась вполне успешной, особенно в сравнении с популярными в то время системами параллельных вычислений Сумма и Минимакс [6], позволявшими описывать параллелизм более примитивными и ограниченными средствами.

Затем эта же версия языка граф-схем была реализована в централизованном варианте на многомашинных комплексах СМ1/СМ2, причем в этой реализации существенное развитие получила система машинной поддержки процесса конструирования граф-схемных параллельных программ, их модификации и отладки.

В [3] в язык граф-схем было введено простое, но весьма важное по практическим соображениям расширение, касающееся векторного динамического параллелизма и возможности его компактного параметрического схемного отражения.

В работе [7] был сделан радикальный шаг в развитии граф-схемного языка, состоящий в интерпретации информационных связей между модулями как буферов с FIFO (первое поступившее данное считывается первым) дисциплиной их обслуживания. Как следствие, пространственная (информационно-независимая) и основанная на времен-

ном совмещении выполнения различных частей программы по принципу конвейера, формы параллелизма могли быть одинаково представлены в граф-схемной модели. Утвердившийся затем термин «потоковые вычисления» [8] объединяет эти различные формы параллелизма, а классификационное разделение принципов функционирования параллельных систем по типу SIMD (один поток команд и много потоков данных), MIMD (много потоков команд и много потоков данных) и др. отражает особенности реализации потоковых схем.

Эта версия граф-схемного языка интересна еще и тем, что она базируется на строгом понятии асинхронной вычислительной сети и описании процесса функционирования сети как композиции процессов, индуцируемых конечно-автономной интерпретацией процессов выполнения модулей граф-схемы. Естественно, что формализм сетей Петри оказался наиболее подходящей моделью для формального описания процесса выполнения граф-схем [7].

Реализация этой версии граф-схемного языка, выполненная на многопроцессорных и многомашинных комплексах ЕС ЭВМ [9], – первая профессионально выполненная разработка системы параллельного программирования для подобной универсальной организации вычислительных систем на базе серийных ЭВМ. Во-первых, в этой реализации максимально использовались средства операционной системы ЕС ЭВМ для выполнения программ модулей граф-схемы, для организации межмашинных обменов и др. Управление параллельным выполнением граф-схемных программ в этой реализации построено по строго децентрализованному принципу, причем предварительно граф-схема разрезается на n подсхем, где n – количество машин в системе, таким образом, чтобы после «назначения» подсхем на n ЭВМ достигался определенный баланс их загрузки и загрузки каналов обмена между машинами. Подобное статическое планирование параллельных вычислений, не допускающее динамического перераспределения работ между ЭВМ в процессе их функционирования, является ограничительным, однако оказалось достаточно простым в реализации.

В [10] средства управления параллельным выполнением граф-схем были расширены введением специальных программ реакции на отказы и сбои в системе и управления реконфигурированием системы, это существенно повысило ее значение, особенно в сфере военных приложений. Вместо традиционно применяемого механизма резервирования параллельная работа обеспечивала большее суммарное быстродействие и одновременно способность продолжения работы, если в ней оставалась работоспособной хотя бы одна ЭВМ.

Данный доклад посвящён описанию реализуемого научной группой под руководством проф. В.П. Кутепова на кафедре прикладной ма-

тематики МЭИ проекта создания системы граф-схемного потокового параллельного программирования для кластерных систем.

Широкие возможности, которые сегодня предоставляет вычислительная техника в части организации вычислительных систем на базе стандартных аппаратных средств – персональных компьютеров и сетевых коммуникаций актуализировали работы в области параллельных и распределенных вычислений [11]. Активно ведутся работы по созданию программных средств для эффективной организации параллельных и распределенных вычислений на кластерах. Кроме известных расширений последовательных языков программирования (High performance C и C++, mpC, DVM, параллельный ФОРТРАН, CHARM++, Mosix и др.), сегодня широко используются стандартизированные API-средства (Application Parallel Interface) и библиотеки: MPI, PVM и др. (см. сайт parallel.ru и др.).

Внимательный анализ этого уже достаточно обширного арсенала программных средств для кластеров показывает [12], что они обычно базируются на расширениях языков последовательного программирования (Фортрана, C, C++ и др.), а в реализации для организации параллельного выполнения программ используются специальные библиотеки функций, позволяющие описывать межпроцессное (и межкомпьютерное) взаимодействие. При этом за программистом остается основная работа не только в написании корректной и эффективной программы, но также и ее реализация (распределение процессов на компьютеры кластера). Из программных систем для кластеров, возможно, только в проекте Mosix задача управления параллельными процессами рассматривается как центральная, от решения которой прямо зависит эффективность работы кластера [13] (см. также сайт mosix.tcs.huji.ac.il).

В нашем проекте три главных составляющих, определяющих эффективность параллельных вычислений на кластере: язык программирования, инструментальная среда разработки параллельных программ и программные средства управления параллельным выполнением программ рассматриваются и реализуются комплексно.

Язык граф-схемного потокового параллельного программирования (ЯГСПП)

ЯГСПП ориентирован на крупноблочное (модульное) потоковое программирование задач, он также может эффективно применяться для программного моделирования распределенных систем, систем массового обслуживания и др., информационные связи между компонентами которых структурированы и управляются потоками данных, передаваемых по этим связям.

Язык позволяет эффективно и единообразно представлять в программах три вида параллелизма:

– параллелизм информационно-независимых фрагментов;

– потоковый параллелизм, обязанный своим происхождением конвейерному принципу обработки данных;

– параллелизм множества данных, реализуемый в ЯГСПП через механизм тегирования, когда одна и та же программа или ее фрагмент применяются к различным данным;

Другими, важными с позиции программирования, особенностями ЯГСПП являются:

– возможность визуального графического и текстового представлений программ;

– возможность простого структурирования программы и отражения декомпозиционной иерархии при ее построении путем использования отношения «схема-подсхема»;

– использование традиционных последовательных языков при программировании модулей.

Особенности «устройства» языка и программирование на нём детально обсуждается в [16].

Для программирования на ЯГСПП разработан комплекс программных средств (названный инструментальной средой [16]), который предназначен для повышения эффективности их разработки.

Реализация ЯГСПП на кластерных системах

Мы исходим из того, что локальная сеть (в том числе сеть персональных компьютеров) является основным «строительным блоком» или узлом вычислительной системы или кластера (рис. 1)

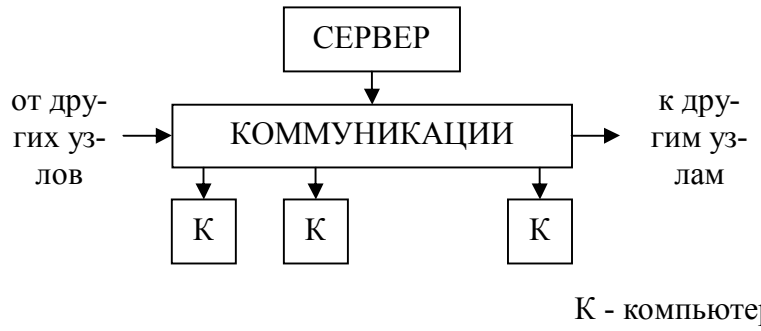


Рис. 1. Узел кластера

Узел кластера – хорошо сбалансированная по отношению, к быстродействию компьютеров, их количеству и пропускной способности коммуникации вычислительная система, так что на ней можно эффектно осуществлять реализацию параллельных вычислений в достаточно широком диапазоне вариаций степени распараллеливания программ [11] от «крупнозернистого» до «мелкозернистого» параллелизма.

Задача сервера на рис. 1, как важного логического элемента узла кластера, – обеспечение на программном уровне механизма масштабирования кластера, реализация интерактивных взаимодействий между узлами, управление их загрузкой и др. (см. далее). Путем введения соответствующей стратегии соподчинения серверов кластера можно легко построить различные схемы управления в системе: от строго централизованной, до иерархической, полностью децентрализованной и др.

Расширение системы достигается путем объединения кластерных узлов, как правило, через более медленные коммуникации. По сути, эта идеология построения больших масштабируемых вычислительных структур, продиктованная также техническими и технологическими соображениями, прослеживается в архитектуре всех самых мощных компьютерных систем: SPP, Hewlett Packard и Convex [14], SP1 и SP2, ASCI WHITE IBM [15] и др.

В них в качестве узла обычно выступает 8, 16 или 32 – процессорная подсистема, в которой используются самые быстрые коммуникации (коммутаторы, переключательные матрицы и т.п.). Для межузловых соединений применяются менее дорогие и скоростные коммуникации. Эта особенность в организации вычислительной системы требует, чтобы при вычислениях более частые обменные взаимодействия происходили внутри узла, так как в нем можно планировать вычисления с большей степенью распараллеливания, т.е. реализовывать мелкозернистый параллелизм. В то же время на межузловом уровне планирование загрузки (ее реализует сервер узла, рис. 1) должно осуществляться на более высоком уровне сложности частей программы, пересылаемых между узлами, с целью уменьшения времени, затрачиваемого на обмены [11].

Отметим также, что компьютеры узла могут быть многопроцессорными, иметь векторные сопроцессоры, что так же должно учитываться как на программном уровне, так и при планировании параллельных вычислений на кластере.

Управление параллельными вычислениями на кластере

Собственно задача управления параллельными вычислениями формулируется, как задача минимизации времени выполнения параллельной программы, и используемых при этом ресурсов (обычно количества используемых компьютеров или процессорных элементов в вычислительной системе). Ее практическое решение осуществляется через механизмы планирования и требует тонкого учета специфики задания выполняемой параллельной программы, степени ее распараллеливания, загруженности различных элементов вычислительной системы (компьютеров, коммуникаций и др.), времени их безотказной работы и др. [1, 10, 11].

Эта проблема требует специального анализа, поэтому далее рассматриваются, в основном, организационные аспекты реализации управления параллельным выполнением ГСПП на кластерах.

В организационной структуре управления можно выделить два относительно самостоятельных уровня: общесистемный, в задачу которого входит управление конфигурированием кластера, контроль и планирование загрузки (узлов, компьютеров), реакция на отказы компонентов кластера и др., и уровень собственно управления параллельными процессами, индуцируемыми при выполнении ГСПП.

На рис. 2 представлена структура и основные блоки управления процессом выполнения ГСПП на кластерных системах.

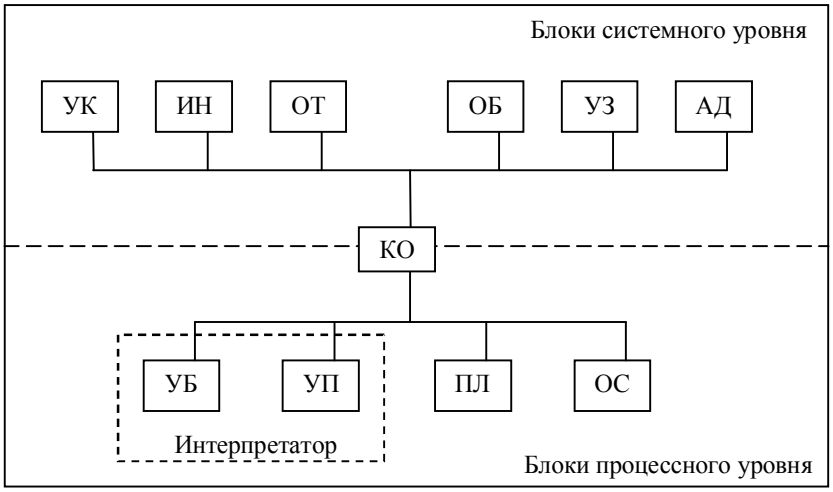


Рис. 2. Структура и основные блоки управления процессом выполнения ГСПП

Кратко опишем функции указанных на этом рисунке блоков.

УК – блок управления конфигурациями, в задачу которого входит конфигурирование узла кластера (или кластера в целом), и его реконфигурирование в процессе функционирования в связи с отказами и восстановлениями компонентов кластера (компьютеров, коммуникаций и др.).

ИН – блок инициализации выполнения ГСПП.

ОТ – блок отказоустойчивости, его функции: реакция на отказы компонентов кластера и реализация принятой стратегии периодического сохранения состояний компонентов кластера на случай их отказа. Стратегия может быть или простейшей, предусматривающей периодическое сохранение состояния компьютера или другого контролируемого компонента кластера на общем носителе памяти (например, дисковой памяти сервера узла), или более сложной, например, основанной на той или иной схеме «договоренности» между компьютерами о периодическом сохранении компьютерами состояний других компьютеров на случай отказа последних [10]. Отказы или восстановления компонентов класте-

ра, фиксируемые блоком ОТ, также передаются блоку УК в качестве информации для изменения конфигурации кластера.

ОБ – блок реализации по соответствующему протоколу интерфейсных взаимодействий между компьютерами кластера.

УЗ – блок контроля загрузки компонентов кластера, ее прогнозирования и перераспределения параллельных процессов между компьютерами узла или между узлами кластера с целью достижения максимальной эффективности его работы.

АД – блок администрирования, в функции которого входят инсталляция программных средств для управления на кластере, оперативное получение информации о функционировании (например, загрузке) кластера, «вмешательство» в его работу администратора, если это оказывается необходимым.

Блок координации (КО) представляет из себя в программном смысле монитор и осуществляет все взаимодействия друг с другом перечисленных блоков.

Блоки процессного уровня непосредственно связаны с определением готовности и идентификацией индуцируемых при выполнении ГСПП процессов (блок управления буферами – УБ), их организацией и контролем состояний (блок управления процессами УП) и планированием выполнения (блок планирования – ПЛ).

Блок УП также реализует, путем обращения к операционной системе (ОС) функции, связанные с постановкой процессов в очередь для выполнения на процессоре компьютера и идентификацией команд обращения к блоку управления буферами.

Блоки, реализующие эти действия, образуют интерпретатор ЯГСПП.

Заключение

В настоящее время на языке Java разработан комплекс программных средств управления параллельным выполнением граф – схемных программ на кластере, который базируется на описанной схеме построения управления кластером и выделенных в ней блоках.

В настоящее время идут работы по отладке разработанных программных средств на кластере кафедры.

Работа выполнена при поддержке РФФИ, проект 03-01-00588

Литература

1. *Кутепов В.П.* Организация параллельных вычислений на системах // М:, МЭИ, 1988.

2. *Kutepov V.P., Falk V.* Integrated tools for functional logical and data flow parallel programming and controlling parallel computations on computer systems //

Proceedings of International conference on parallel computing technologies, Novosibirsk, 1991.

3. *Арефьев А.А.* и др. Язык граф-схем параллельных алгоритмов и его расширения // Программирование, 1981, №4.

4. *Кораблин Ю.П.* Проблема корректности граф-схем параллельных алгоритмов // Программирование, 1978, №5.

5. *Кораблин Ю.П.* Языки параллельных алгоритмов и принципы их реализации // Автореферат канд. диссертации. М.: МЭИ, 1977.

6. *Дмитриев Ю.К., Хорошевский В.Г.* Вычислительные системы из мини ЭВМ // М:, «Радио и связь», 1982.

7. *Строева Т.М., Фальк В.Н.* Асинхронные вычислительные сети (АВС): АВС-модель и АВС система программирования // Кибернетика, 1981. №3.

8. *Dennis J.B.* First version of data flow language // In Proc. Colloque sur la Programmation, vol.19 (Lecture notes in computer science), 1974.

9. *Строева Т.М.* Асинхронные вычислительные сети и их применение в системах параллельного программирования // Автореферат кандидатской диссертации. М.: МЭИ, 1981.

10. *Лобанов В.П.* Разработка алгоритмов и программных средств обеспечения надежности параллельных вычислений на вычислительных комплексах // Автореферат кандидатской диссертации. М.: МЭИ, 1985.

11. *Кутепов В.П.* Об интеллектуальных компьютерах и больших компьютерных системах нового поколения // Теория и системы управления, 1996. №5.

12. *Воеводин В.В., Воеводин Вл.В.* Параллельные вычисления // Санкт-Петербург, «БХВ-Петербург», 2002.

13. *Amnon Barak* et. A Scalable cluster computing with Mosix for Linux: Institute of Computer Science // The Hebrew University of Jerusalem, 1999.

14. *Шнитман В.* Системы Exemplar SPP1200 // Открытые системы, 1991. №6.

15. *Шмидт В.* Системы IBM SP2 // Открытые системы, 1995. №6.

16. *Кутепов В.П., Котляров Д.В.* и др. Граф-схемное потоковое параллельное программирование и его реализация на кластерных системах // Программирование (в печати).

СИСТЕМА ГРАФ-СХЕМНОГО ПОТОКОВОГО ПАРАЛЛЕЛЬНОГО ПРОГРАММИРОВАНИЯ: ЯЗЫК И ИНСТРУМЕНТАЛЬНАЯ СРЕДА ПОСТРОЕНИЯ ПРОГРАММ

В.П. Кутепов, Д.В. Котляров, В.А. Лазуткин

Московский Энергетический Институт (ТУ), г. Москва

Введение

В докладе описаны языковые и инструментальные средства граф-схемного потокового программирования, разработанные в процессе выполнения проекта создания системы граф-схемного параллельного пото-

кового программирования, осуществляемого на кафедре прикладной математики МЭИ [1, 2].

1. Язык граф-схемного потокового параллельного программирования

Язык граф-схемного параллельного программирования (ЯГСПП) ориентирован на крупноблочное (модульное) потоковое программирование задач, он также может эффективно применяться для программного моделирования распределенных систем, систем массового обслуживания и др., информационные связи, между компонентами которых структурированы и управляются потоками данных, передаваемых по этим связям.

Язык позволяет эффективно и единообразно представлять в программах следующие виды параллелизма:

– параллелизм информационно-независимых фрагментов (пространственный);

– потоковый параллелизм, обязанный своим происхождением конвейерному принципу обработки данных;

– параллелизм множества данных (SIMD), реализуемый в языке граф-схемных параллельных программ через механизм тегирования, когда одна и та же программа или ее фрагмент применяются к различным данным;

– «внутренний» параллелизм, описываемый в самих подпрограммах модулей, например, средствами порождения нитей в подпрограммах, задания векторного параллелизма и др. [1].

С другой стороны, важными с позиции программирования, особенностями ЯГСПП являются:

– возможность визуального графического и текстового способов разработки программ, возможность взаимнооднозначного перехода между ними;

– возможность простого структурирования программы и отражения декомпозиционной иерархии при ее построении путем использования отношения «схема-подсхема»;

– использование традиционных последовательных языков (Pascal, C/C++, Java и др.) при программировании модулей.

Граф-схемная параллельная программа (ГСПП) представляется в виде пары <ГС,I>, где ГС – граф-схема, I – интерпретация.

Граф-схема или просто схема позволяет визуально представлять состоящую из модулей программу решения задачи; интерпретация сопоставляет модулям множество подпрограмм, а связям между модулями – типы данных, передаваемых между подпрограммами модулей в процессе выполнения ГСПП. Основным «строительным» блоком ГСПП яв-

ляется модуль. Все входы и выходы модулей строго типизированы и разделены на группы (отдавая дань предыстории ЯГСПП, они называются конъюнктивными группами входов (КГВх) и конъюнктивными группами выходов (КГВых) соответственно) и отражают структуру потоков данных, передаваемых между подпрограммами модулей. Каждой КГВх модуля однозначно сопоставляется подпрограмма на одном из последовательных языков программирования (C/C++, Pascal, Java и др.), причем перечисленные формальные параметры и их типы в задании подпрограммы должны совпадать с порядком изображения (слева направо) входов КГВх и их типами соответственно. ГС представляет блочную структуру, построенную из модулей, путем соединения выходов КГВых модуля с входами КГВх другого или этого же модуля, причем типы соединяемых входов и выходов должны совпадать. В интерпретации такое соединение называется информационной связью (ИС), по которой данные соответствующего типа передаются от одного модуля к другому [1, 3].

Параллельное выполнение ГСПП представляется как последовательность смены состояний, каждое из которых характеризуется множеством процессов, индуцируемых при выполнении подпрограмм модулей ГСПП, сопоставляемых в интерпретации их КГВх.

Модуль ГСПП считается готовым для выполнения по любой из своих КГВх, если на всех входах этой КГВх (в соответствующих входам буферах) есть данные, помеченные одним и тем же тегом [1].

При выполнении процесса в его подпрограмме могут использоваться специальные системные команды, реализуемые посредством обычного обращения к специальным функциям: **WRITE** (запись), **READ** (чтение), **OUT** (выход), позволяющие осуществлять межмодульное взаимодействие, т.е. строить различные схемы взаимодействия по данным между подпрограммами различных модулей путем чтения данных из буферов или записи данных в буферы, сопоставляемые входам КГВх модулей [1].

Для того чтобы ГСПП могла одновременно оперировать с различными, не связанными друг с другом экземплярами данных, последние снабжаются специальными метками, иначе тегами, позволяющими не смешивать данные из различных экземпляров. Тегирование позволяет одновременно применять подпрограммы модуля к экземплярам данных, обеспечивая при этом однозначность зависимости между входными данными и результатами параллельного потокового выполнения ГСПП. С каждым значением, которое передается по ИС, в процессе выполнения ГСПП, должен быть связан уникальный тег, идентифицирующий только этот набор данных, и наследуемый при передаче результата применения подпрограммы модуля к этим данным на входах следующих модулей.

Рассмотрим пример построения двух граф-схем по разному реализующих задачу вычисления двух сумм: произведений четных элементов

и частных (соответствующий элемент первого вектора делим на соответствующий элемент второго вектора) нечетных элементов векторов $\boldsymbol{a}$ и $\boldsymbol{b}$ длины N, причем длина N и сами векторы заданы не статически, а генерируются случайным образом.

Разработка граф-схемы задачи, определение задач модулей:

а) Первое решение – с использованием механизма тегирования. С помощью тега будем различать, какое именно (четное или нечетное) значение поступило.

ГС данной задачи построим следующим образом (см. рис.1): модуль GenN вырабатывает случайное целое число – количество компонент вектора и передает на вход модулей GenA и GenB, которые вырабатывают сами компоненты вектора. Четные компоненты вектора помечаем одним тегом, нечетные – другим и передаем их модулю EvalExpr, который в зависимости от значения тега либо перемножает компоненты векторов, либо делит их. А результат затем отправляет модулю Sum. На другой вход модуля Sum поступают два значения: количество четных и нечетных компонент вектора, с соответствующими тегами. Это необходимо для того, чтоб просуммировать четные (нечетные) компоненты вектора (будет одновременно запущено два экземпляра суммирующей функции). После суммирования результат передается модулю Print для вывода его на экран.

б) Второе решение – без использования механизма тегирования (см. рис. 2). В данном случае задачу определения четности поступившего значения наложим на дополнительный модуль (CheckValue), который, в зависимости от четности значений, отсылает их либо на первую КГВых, либо на вторую. Модуль EvalSum занимается вычислением суммы, если данные поступили на первую КГВх модуля, то суммируются произведения поступающих значений, если на вторую, то – частные. Модуль GetData читает начальные данные из файла и передает модулю CheckValue, который, потоковым образом, отправляет полученные данные на различные КГВых в зависимости от четности текущего индекса. КГВх модуля EvalSum также обрабатывают данные потоковым образом, а именно суммируют произведения (частные) компонент вектора. После вычисления обоих сумм результат передаются модулю Print для вывода его на экран.

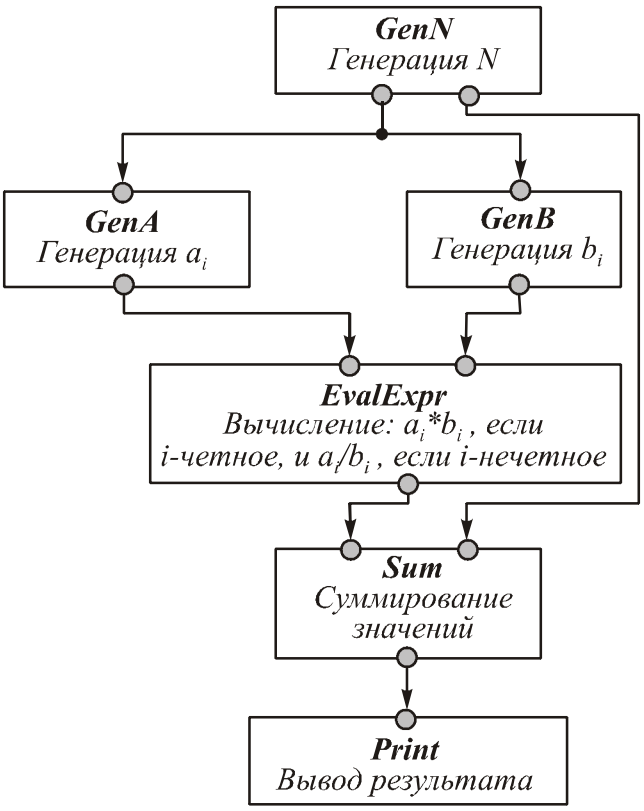


Рис. 1. ГС программы а)

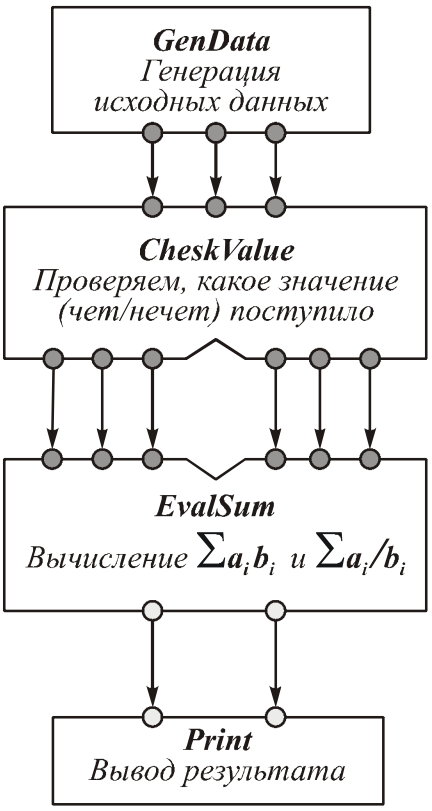


Рис. 2. ГС программы б)

Для первого решения примера рассмотрим описание подпрограмм каждого модуля на языке С:

Модуль GenN:

```
void startup()
{
 int N;
 int place[1]={1},data[1];
 N=rand();//Генерируем случайное целое число N
 data[0]=N;
 write(1,0,place,data);//Пересылаем количество генери-
руемых элементов
 place[0]=2;
 data[0]=N - N/2;
 write(1,1,place,data);//Количество четных элементов
(тег = 1)
 data[0]=N/2;
 out(1,2,place,data);//Количество нечетных элементов
(тег = 2)
}
```

Модули GenA и GenB:

```
void GenVector(int tag,int x1)
{
 int place[1]={1},data[1];
 for(int i=0;i<x1;i++)
 {
```

```
      data[0]=rand();//Генерируем случайное целое число –
элемент вектора
      if(i%2==0)//если i-четное, то тег задаем равный 1
      {
      write(1,1,place,data);
      }
      else//иначе тег устанавливаем в 2
      {
      write(1,2,place,data);
      }
      }
     }
```

Модуль EvalExpr:

```
     void evalution(int tag,int x1,int x2)
     {
      int place[1]={1};
      float data[1];
      if(tag==1)//если четный индекс
      {
      data[0]=(float)(x1*x2);//вычисляем произведение
      }
      else//если нечетный индекс
      {
      data[0]=(float)x1/(float)x2; //вычисляем частное
      }
      out(1,tag,place,data);
     }
```

Модуль Sum:

```
     void sum(int tag,float x1,int x2)
     {
      int place[1]={1};
      float data[1];
      float S=x1;//Начальное значение суммы
      float value;
      for(int i=0;i<x2-1;i++)
      {
      //Читаем новое значение и добавляем к сумме
      read(1,tag,place,&value);
      S+=value;
      }
      data[0]=S;
      out(1,tag,place,data)
     }
```

Модуль Print:

```
     void outres(int tag,float x1)
     {
      if(tag==1)
      {
```

```
      printf("Сумма произведений четных компонент вектора:
%f",x1);
      }
      else
      {
      printf("Сумма частных нечетных компонент вектора:
%f",x1);
      }
     }
```

Для второго решения примера ограничимся рассмотрением подпрограмм модулей CheckValue и EvalSum:

Модуль CheckValue:

```
     void checking(int tag,int x1,int x2,int x3)
     {
      int place1[2]={1,2},place2[1]={3},place3[2]={1,3};
      int data1[2],data2[1];
      data2[0]=x2-x2/2;
      data1[0]=x1; data1[1]=x2;
      write(1,0,place1,data1);//Отсылаем модулю EvalSum1
первые четные значения векторов: a0 и b0
      write(1,0,place2,data2);//Отсылаем модулю EvalSum1 ко-
личество четных значений
      data2[0]=x2/2;
     write(2,0,place2,data2);//Отсылаем модулю EvalSum2 ко-
личество нечетных значений
      int CurrentVal=1;//Переменная, которая будет отвечать
за четность текущего значения (1-нечетное,0-четное)
      for(int i=0;i<x2-1;i++)
      {
      read(1,0,place3,data1);//Читаем поступающие значения
      if(CurrentVal==0)//если текущий индекс четный
      {
      write(1,0,place1,data1);//передаем значения модулю
EvalSum1
      }
      else//если текущий индекс нечетный
      {
      write(2,0,place1,data1);//передаем значения модулю
EvalSum2
      }
      CurrentVal=1-CurrentVal;//меняем четность
      }
     }
```

Модуль EvalSum:

```
     void EvalSum1(int tag,int x1,int x2,int x3)
     {
      int place1[2]={1,2},place2[1]={1};
```

```
      float data[1];
      float S=(float)(x1*x2);//Начальное значение суммы
      int value[2];
      for(int i=0;i<x3-1;i++)
      {
      //Читаем новые значение и их произведение добавляем к
сумме
      read(1,0,place1,value);
      S+=(float)(value[0]*value[1]);
      }
      data[0]=S;
      out(1,0,place2,data);
     }
     void EvalSum2(int tag,int x1,int x2,int x3)
     {
      int place1[2]={1,2},place2[1]={1};
      float data[1];
      float S=(float)x1/(float)x2;//Начальное значение суммы
      int value[2];
      for(int i=0;i<x3-1;i++)
      {
      //Читаем новые значение и их частное добавляем к сумме
      read(1,0,place1,value);
S+=(float)value[0]/(float)value[1];
      }
      data[0]=S;
      out(1,0,place2,data);
```

2. Инструментальная среда разработки программ на ЯГСПП

Инструментальная среда предназначена для разработки граф-схемных параллельных программ и целиком согласована с ЯГСПП. В инструментальной среде поддерживается графическая разработка ГСПП и разработка ГСПП в текстовом виде, используя многооконную организацию. Между графическим и текстовым представлением ГСПП существует взаимнооднозначный переход. Инструментальная среда соединена с реляционной базой данных, которая играет роль единого хранилища ГСПП всех пользователей. При разработке программы пользователю предоставляется возможность сконцентрировать свое основное внимание на интересующем его элементе: подсхеме (ее содержимое открывается в новом окне), подпрограмме, сопоставленной некоторой КГВх (текст подпрограммы будет открыт в новом окне, где можно его отредактировать, настроить различные параметры подпрограммы, проверить корректность типов данных) и др.

На рис. 3 приведены основные блоки инструментальной среды:

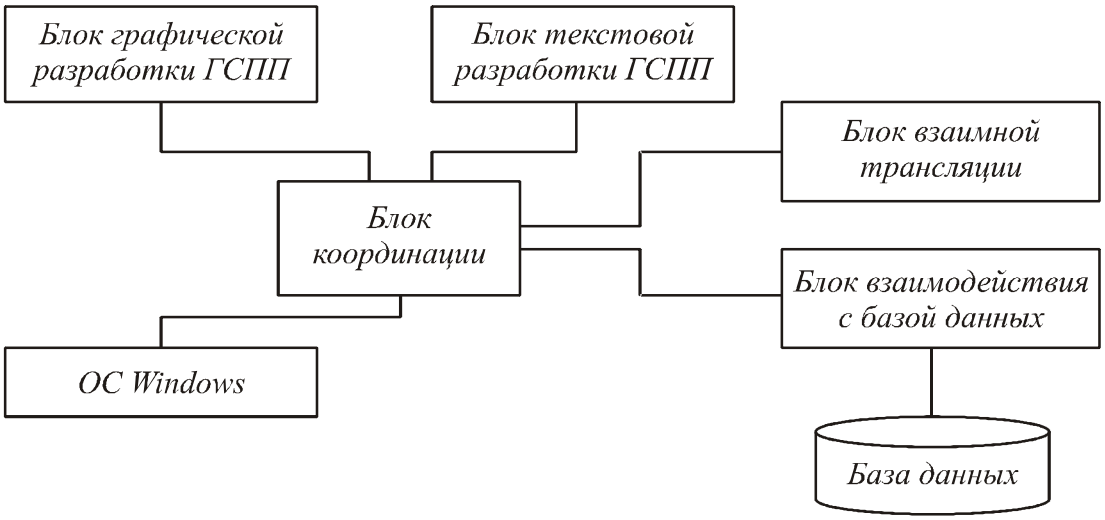


Рис. 3. Архитектура инструментальной среды

Рассмотрим назначение каждого блока:

– Блок графической разработки ГСПП предназначен для визуального проектирования ГС, позволяя добавлять, удалять, редактировать элементы ГСПП и др. Данный блок также осуществляет синтаксический контроль создаваемой ГС.

– Блок текстовой разработки ГСПП управляет ходом построения текстового представления и по своему назначению представляет собой аналог блока разработки графического представления, но работает с текстом ГСПП. Текстовое представление ГСПП представляет собой набор XML – документов, DTD которых описано в [1].

– Блоки графической и текстовой разработки ГСПП помимо редактирования, синтаксического анализа и выявления ошибок в ГС создаваемых программ имеют средства для поддержки процессов разработки программ «сверху – вниз» и «снизу – вверх». Декомпозиция и структурирование, присущие процессу разработки программ «сверху – вниз», легко реализуются в инструментальной среде, благодаря модульной организации ГСПП и возможности выделения подсхем.

Механизм задания интерпретации ГС требует обращения к конкретному языку (или языкам) последовательного программирования. Как уже было сказано ранее, в ЯГСПП в качестве подпрограмм, сопоставляемых КГВх модулей, могут использоваться широко известные последовательные языки (C/C++, Pascal, Java и др.). Поскольку инструментальная среда разработана под ОС Windows, в задании интерпретации можно применять все стандартные средства этой ОС для написания подпрограмм, сопоставляемых КГВх модулей.

Все необходимые для задания интерпретации ГС подпрограммы образуют специальную библиотеку (или набор библиотек), а доступ к ним осуществляется, как это определено синтаксисом ЯГСПП.

Блок взаимной трансляции занимается переводом графического представления ГСПП в текстовое и наоборот. Промежуточным звеном на этапе трансляции является объектно-ориентированное внутреннее представление ГСПП.

В инструментальной среде используется реляционная база данных (MS SQL Server 2000) как средство для длительного хранения всей информации о разрабатываемых пользователями ГСПП. Блок взаимодействия с базой данных осуществляет перенос информации о ГСПП из внутреннего представления в базу данных и обратно.

Блок координации управляет работой других блоков.

Заключение

В настоящее время уже накоплен определенный опыт программирования на ЯГСПП различных задач [3], как вычислительного характера (матричных задач, задач решения систем линейных уравнений и др.), так и задач, которые непосредственно связаны с потоковой обработкой [4]. Как показывает этот опыт, визуальная схемная разработка параллельных потоковых программ на ЯГСПП с использованием созданных инструментальных средств существенно упрощает дело. К примеру, декомпозиционное описание технологических процессов сборки автомобилей, других автоматизированных производств позволяет быстро и просто строить прямые их аналоги в виде программ на ЯГСПП. В стадии завершения находится работа по объединению объектно-ориентированного и граф-схемного подходов, которые должны обеспечить более широкую технологическую базу для повышения эффективности и унификации методов проектирования программ на ЯГСПП.

Литература

1. *Кутепов В.П., Котляров Д. В., Лазуткин В.А., Осипов М.А.* Граф-схемное модульное потоковое программирование и его реализация на кластерных системах // Программирование (в печати).

2. *Кутепов В.П., Котляров Д. В., Лазуткин В.А., Осипов М.А.* Разработка системы граф-схемного потокового параллельного программирования для кластеров // Доклад на четвертом Международном научно-практическом семинаре «Высокопроизводительные параллельные вычисления на кластерных системах».

3. *Лазуткин В.А.* Реализация языка граф-схемного параллельного программирования // Бакалаврская работа.

4. *Грегори Р. Эндрюс* Основы многопоточного, параллельного и распределенного программирования // Пер. с англ. – М.: Издательский дом «Вильямс», 2003. – 512 с.

АНАЛИЗ СТРУКТУРНОЙ СЛОЖНОСТИ ФУНКЦИОНАЛЬНЫХ ПРОГРАММ И ЕГО ПРИМЕНЕНИЕ ДЛЯ ПЛАНИРОВАНИЯ ИХ ПАРАЛЛЕЛЬНОГО ВЫПОЛНЕНИЯ НА ВЫЧИСЛИТЕЛЬНЫХ СИСТЕМАХ

В.П. Кутепов, Д.А. Шестаков

Московский Энергетический Институт (ТУ), г. Москва

Введение

Задача определения характеристик программ, по которым можно судить об их организационной (структурной) и вычислительной сложности, представляют большой теоретический и практический интерес.

Ниже описывается решение этой задачи, которая возникла при реализации на кафедре прикладной математики проекта создания системы функционального параллельного программирования для кластерных систем (научный руководитель – проф. В.П. Кутепов). В рамках этого проекта был разработан язык композиционного функционального параллельного программирования FPTL [1], программы на котором в математическом смысле представляются в виде специального вида функциональных уравнений:

(*) $X_i = \tau_i(X_1,..,X_k, i=1,2,..,k)$,

где X_i – определяемая функция, а τ_i – композиционные термы, построенные путем применения операций композиции функций к функциональным переменным X_i, $i=1,2,..,k$, и базисным функциям, которые являются либо встроенными командами компьютера, либо функциями, определяемыми посредством конструкторов, либо импортируемыми из других языков (C, Pascal и т.д.).

Операции композиции функций в созданном языке таковы, что только операция последовательной композиции имеет последовательную вычислительную семантику; другие операции являются параллельными [2].

Это создает предпосылки для построения достаточно эффективных механизмов автоматического выявления параллелизма в программе и реализации процессов параллельного выполнения функциональных программ на вычислительных системах (ВС). Однако на этом пути возникает проблема определения фрагментов функциональной программы, которые целесообразно перемещать между компьютерами ВС в процессе ее параллельного выполнения. Допуская возможность перемещения несложных фрагментов (например, базисных функций), обнаруживается известный эффект резкого увеличения интенсивности обменных взаимодействий между компьютера-

ми ВС и следующей за этим ее деградацией [3]. Поэтому алгоритм планирования параллельных процессов должен обладать возможностью достаточной тонкой дифференциации по сложности фрагментов выполняемой программы, которые могут выполняться одновременно [3].

Предлагаемое решение основано на анализе системы уравнений (*), представляющих функцию, а также на анализе отношений между транзитивными классами определяемых функций с целью выявления в ней различных по своей природе рекурсивных определений функций (простых циклов, взаимно рекурсивных определений и др.), независимых определений функций и др. [4]. Это позволяет достаточно точно упорядочивать одновременно выполняемые фрагменты параллельной программы по вычислительной сложности и сложности их взаимных связей по данным, эффективно решать задачу определения функций, которые планировщик перемещает между компьютерами с целью достижения их равномерной загрузки и уменьшения обменных взаимодействий.

Заметим, что анализ природы рекурсивных определений в программе позволяет также достаточно строго судить о ее структурной сложности.

Описываемая ниже система оценок структурной сложности основывается на понятиях «рекурсивность», «цикличность», «вложенность».

1. Характеристики структурной сложности функциональных программ

Будем исходить из того, что функция определена в виде системы уравнений (*), где τ_i – терм, в котором могут содержаться функциональные переменные $X_1,..,X_k$ и базисные функции $f_1,.., f_m$.

Введем отношение непосредственной зависимости между функциональными переменными: если X_j используется в терме τ_i, то будем говорить, что X_i непосредственно зависит от X_j (обозначение $X_i \rightarrow X_j$).

Обозначим через $[X_i]$ транзитивное замыкание отношения непосредственной зависимости для X_i.

Множество $[X_i]$ будем транзитивным классом X_i. Очевидно, если $[X_i] = \varnothing$, то в определении X_i не используется функциональных переменных.

Определение 1: Функциональная переменная X_i определена рекурсивно, если $X_i \in [X_i]$.

Определение 2: Определения X_i и X_j взаимно рекурсивны, если $X_i \in [X_j]$ и $X_j \in [X_i]$.

Легко показать, что в этом случае $[X_i]=[X_j]$

Определение 3: Определение X_j- простое рекурсивное или простое циклическое определение, если

$X_j \in [X_j] \wedge \forall X_i (X_i \in [X_j] \wedge X_i \neq X_j \supset X_j \notin [X_i] \wedge X_i \notin [X_i])$.

Простое циклическое определение не содержит вложенных циклических определений.

Определение 4: Определение X_j назовем циклическим, если

$X_j \in [X_j] \wedge \forall X_i (X_i \in [X_j] \wedge X_i \neq X_j \supset X_j \notin [X_i])$.

Определение 5: Определение X_j – простое, если X_j не является рекурсивным и $[X_i]$ не содержит рекурсивных определений.

Определение 6: Назовем классом взаимной рекурсивности для X_i множество всех взаимно рекурсивных с X_i определений.

Будем обозначать класс взаимной рекурсивности для X_i через $<X_i>$. Очевидно, что для взаимно рекурсивных определений классы взаимной рекурсивности будут эквивалентны. Легко показать, что неэквивалентные классы взаимной рекурсивности не пересекаются. Очевидно, что если $<X_i>$ вложено в X_j, то каждый элемент $X_k \in <X_i>$ также вложен в X_j.

Определение 7: Определение X_i вложено в определение X_j, если $X_i \in [X_j]$, X_i строго вложено в определение X_j, если $X_i \in [X_j]$ и $X_j \notin [X_i]$.

Очевидно, взаимно рекурсивные определения строго вложены друг в друга.

Если определение X_i – циклическое определение (простое циклическое определение), то будем также говорить, что определение X_i – циклическое вложение в определение X_j (простое циклическое вложение в X_j).

Определение 8: Определения X_i и X_j независимы, если пересечение их транзитивных классов пусто.

Если X_i и X_j независимы, то ясно, что при параллельном вычислении значений $X_i(X')$ и $X_j(X'')$ на разных компьютерах ВС, между этим процессами не будут происходить обменные взаимодействия.

2. Применение к задаче планирования процессов параллельного вычисления функциональных программ на ВС

Описанные результаты структурного анализа функциональных программ используются в двух аспектах в рамках реализуемого проекта [1]. С одной стороны, в инструментальной среде разработки функциональных программ создан программный блок, который позволяет программисту проводить структурный анализ разрабатываемой им программы и оценивать ее организационную и логическую сложность.

С другой стороны, в исполнительной части разработан планировщик, который использует результаты структурного анализа при опреде-

лении фрагментов функциональной программы, которые назначаются для выполнения на различные компьютеры ВС.

При этом планировщик пытается решить две основные задачи: поддерживать равномерную загрузку компьютеров ВС, следуя эвристическому правилу – не допускать простоев и перегрузок компьютеров, и минимизировать обменные взаимодействия между компьютерами ВС.

Используя результаты структурного анализа функциональной программы, планировщик при необходимости перемещения фрагментов программы с одного компьютера (например, перегруженного) на другой компьютер (простаивающий или недогруженный) выбирает наиболее сложный с вычислительной точки зрения фрагмент. При этом в качестве перемещаемых фрагментов рассматриваются только рекурсивно определенные функциональные переменные, что позволяет существенно уменьшить обменные взаимодействия между компьютерами ВС и исключить ее деградацию, которая возникает, если перемещать между компьютерами базисные функции.

Заключение

Были проведены предварительные эксперименты с целью проверки эффективности применения описанных алгоритмов для планирования параллельного выполнения функциональных программ на ВС [5]. Эксперименты проводились на кластере из 6 машин (локальная сеть). Были заданы две функциональных программы – линейная и сильнорекурсивная. Для сравнения помимо эвристического алгоритма планирования, основанного на описанных выше правилах, было использовано еще три алгоритма планирования:

- случайный – назначение процессов на компьютеры для вычисления проводилось в случайном порядке,
- с приоритетами – то же, что и случайный, но выбор функций для вычисления производится в соответствии с их структурными характеристиками,
- «равномерный» – на все узлы ЛВС назначалось одинаковое количество процессов.

В результате произведенных экспериментов было выявлено, что время выполнения линейной программы для эвристического алгоритма планирования одинаково для 2, 4 и 6 машин. Это связано со стратегией назначения функциональных процессов. В результате вся программа выполнялась на одной машине, поэтому время вычислений не зависит от числа машин в кластере.

Наибольшая производительность достигается при использовании эвристического алгоритма планирования для вычисления рекурсивной программы. Фактически, на 6 машинах время выполнения при использо-

вании этого алгоритма в 2-4 раза меньше времени выполнения с использованием случайного алгоритма.

Проведенные эксперименты, хотя и являются ограниченными (небольшое количество компьютеров в кластере, небольшое разнообразие моделируемых программ), тем не менее, показывают, что предложенный алгоритм упорядочивания может оказаться эффективным при организации параллельного выполнения сложных рекурсивных программ на ВС и в настоящее время реализуется в создаваемой системе [1].

Литература

1. *Бажанов С.Е., Кутепов В.П., Шестаков Д.А.* Язык функционального параллельного программирования и его реализация на кластерных системах. Теория и системы управления (в печати).
2. *Кутепов В.П., Фальк В.Н.* Модель асинхронных вычислений значений функций в языке функциональных схем // Программирование, 1978, № 3. С.3-15.
3. *Кутепов В.П.* Об интеллектуальных компьютерах и больших компьютерных системах нового поколения // Известия академии наук, Теория и системы управления, 1996. №5.
4. *Шестаков Д.А.* Разработка алгоритмов и инструментальных программных средств для структурного анализа программ и оценки их сложности. Дипломная работы на соискания звания бакалавр математики, Москва, МЭИ, 2000.
5. *Шестаков Д.А.* Структурный и семантический анализ функциональных программ и создание на его основе эффективных алгоритмов планирования процессов параллельных вычислений значений функций на вычислительных системах // Диссертационная работы на соискание звания магистр математики, Москва, МЭИ, 2002.

ОСОБЕННОСТИ ФУНКЦИОНАЛЬНОГО ЯЗЫКА ПАРАЛЛЕЛЬНОГО ПРОГРАММИРОВАНИЯ «ПИФАГОР»

А.И. Легалов, Д.В. Привалихин

Государственный технический университет, г. Красноярск

Введение

Язык программирования Пифагор [1] предназначен для разработки параллельных программ, управление вычислениями в которых осуществляется по готовности данных. Несмотря на то, что непосредственная реализация подобных средств в настоящее время неэффективна, уже сейчас можно разрабатывать параллельные программы, выполняемые как на традиционных последовательных компьютерах, так и на кластерных системах. Существующие на данный момент версии языка и испол-

нительной системы [2] поддерживают только динамическую типизацию. Формирование новых типов данных осуществляется неявно в ходе выполнения функций. Подобный механизм использовался в ранних языках функционального программирования, например, LISP [3]. Большинство же современных языков данной группы поддерживают строгую типизацию.

Вместе с тем, механизмы динамической типизации могут развиваться по специфичным для них направлениям, что позволяет получать интересные эффекты, расширяющие возможности языка. В частности, сочетание механизма перегрузки функций и динамических типов, определяемых пользователем, обеспечивает написание эволюционно расширяемых параллельных программ. Ниже эти возможности языка «Пифагор» рассматриваются более подробно.

Перегрузка функция с одинаковой сигнатурой

В языке отсутствует строгая типизация, поэтому все функции с одинаковыми именами становятся неразличимы (имеют одинаковую сигнатуру). Вместо выбора одной из перегруженных функций, осуществляется их одновременное выполнение. Результат возвращается в виде параллельного списка. В отличие от обычных функций, перегруженные функции задаются добавлением к обозначению имени квадратных скобок, в которых может стоять любое действительное число, задающее ранг.

Ранг используется для упорядочения функций в параллельном списке по возрастанию. При отсутствии числа ранг считается равным нулю. Функции с одинаковым рангом могут располагаться в списке в произвольном порядке. Обычно они размещаются в порядке их обработки транслятором. Пример использования рангов:

OverFunc[2.5] << funcdef Param { // Тело функции }
OverFunc[] << funcdef Param { // Тело функции }
OverFunc[-10] << funcdef { // Тело функции }

Вызов параллельной функции синтаксически ничем не отличается от обычного вызова. В постфиксной записи он будет выглядеть следующим образом:

X:OverFunc;

Перегрузка функций позволяет гибко добавлять новые возможности, обуславливаемые появлением дополнительной информацией о разрабатываемом алгоритме. В частности можно использовать методы, обеспечивающие децентрализованную разработку отдельных фрагментов функций, связанных с вычислением при альтернативных условиях.

В качестве простейшего примера использования перегрузки функций с одинаковой сигнатурой можно рассмотреть вычисление факториала. Пусть, факториал отдельно вычисляется для значений аргумента, равного

0, 1, диапазона от 2 до ***n*** (где ***n*** – константа, определяющая максимально допустимую величину аргумента). При значении аргумента, превышающем *n*, выводится сообщение о переполнении. При отрицательном значении аргумента выводится сообщение об ошибке. Константа ***n*** определена в программе следующим образом:

n<<const 10.

Функция, осуществляющая вычисление факториала, состоит из нескольких перегруженных функций, каждая из которых осуществляет необходимые вычисления для аргумента, расположенного в ее диапазоне. Если значение аргумента не попадает в рассматриваемый диапазон, перегруженная функция выдает пустое значение.

Сообщение об ошибке, выдаваемое при отрицательном значении аргумента, порождается функцией:

```
overfact[]<< funcdef x {
«Ошибка! Отрицательный аргумент»:[(x,0):<]
>>return;
}
```

В случае переполнения значащий результат выдает другая перегруженная функция, сигнализирующая об ошибке переполнения:

```
overfact[]<< funcdef x {
«Ошибка! Слишком большой аргумент»:[(x,n):>]
>>return;
}
```

При аргументе равном 0 или 1 вычисления осуществляются одной из следующих функций:

```
overfact[]<< funcdef x {
1:[(x,0):=] >>return;
}
overfact[]<< funcdef x {
1:[(x,1):=] >>return;
}
```

И, наконец, при попадании аргумента в диапазон от 2-х до максимально допустимого числа, получаемый результат возвращается следующей функцией:

```
overfact[]<< funcdef x {
 ({(x, (x,1):-:overfact):*})
:[((x,1):>, (x,n):<=):*]:[]:. >> return
}
```

Одновременный вызов всех функций с одинаковой сигнатурой осуществляется внутри функции, осуществляющей окончательное формирование и вывод факториала, синхронизацию альтернативных значений в общем списке данных:

```
fact << funcdef x {
```

```
 (x:overfact):[] >> return
}
```

Применение пользовательских типов данных

В языке реализована инструментальная поддержка механизма строгой динамической типизации на основе пользовательских типов. Для этого введен ряд дополнительных конструкций и расширена семантика уже существующих понятий. Возможны:

– определение пользовательского типа;

– сравнение пользовательских типов на равенство и неравенство;

– проверка на принадлежность некоторого значения величине, допустимой для заданного пользовательского типа;

– преобразование в пользовательский тип;

– разыменование пользовательского типа.

Определение пользовательского типа задается соответствующим предикатом, сопоставляющим проверяемый элемент с некоторым выражением. Если результат проверки является истиной, то элемент принадлежит проверяемому типу. Предикат оформляется в виде специальной функции **typedef**, возвращающей булевское значение. Ее обозначение регистрируется в таблице пользовательских типов. В качестве примера можно рассмотреть, как задаются геометрические фигуры треугольник и круг:

```
Triangle << typedef X {
// Аргумент – список из трех целочисленных элементов
 [(((X:type,datalist):=,(X:|,3):=):*:int,1):+]^
 (
false,
{([(X:1:type,int),
(X:2:type,int),
(X:3:type,int)]:=):*}
):. >> return
};
Circle << typedef X {
// Аргумент – целочисленный атом
(X:type,int):= >> return;
};
```

Сравнение пользовательских типов осуществляется точно также как и сравнение базовых типов языка: выделяется тип элемента функцией **type**, проверяется совпадение имен выделенного и проверяемого типа. Результат сравнения является истиной при совпадении имен типов. Ниже приводится пример использования сравнения пользовательских типов для описания обобщенной геометрической фигуры.

```
Figure << typedef X {
```

// Аргумент – треугольник или круг
X:type >> t
([(t, Triangle), (t, Circle)]:=):+ >> return;
};

Проверка на принадлежность позволяет выяснить возможность соответствия между динамически формируемыми данными и **typedef**. Для этого используется функция **in**, которая возвращает значение, полученное в результате выполнения предиката, заданного в описании пользовательского типа. Принадлежность позволяет в дальнейшем осуществить преобразование проверяемого аргумента в элемент пользовательского типа. Ниже представлены примеры использования функции принадлежности:

((10,20,15),Triangle):in⇒true

((10,20,15),Circle):in⇒false

(10,Circle):in ⇒true

Преобразование в пользовательский тип используется для формирования требуемых абстракций по принципу «обертки» преобразуемых данных. Является расширением операции преобразования базовых типов. Суть заключается в получении нового значения элемента, следующей структуры:

Элемент пользовательского типа =
<пользовательский тип, преобразуемый элемент>

Само преобразование задается указанием пользовательского типа в качестве функции и осуществляется в зависимости от значения аргумента:

– если тип аргумента совпадает с типов в операции преобразования, то возвращается значение исходного аргумента;

– преобразование осуществляется только в том случае, если проверка аргумента на принадлежность функцией **in**, осуществляемая неявно, дает «истину»;

– во всех остальных случаях функция преобразования в пользовательский тип возвращает ошибку **TYPEERROR**.

Использование данной операции позволяет формировать необходимые абстракции при выполнении программы:

(10,20,15):Triangle ⇒ Треугольник со сторонами (10,20,15)

Описанная операция не обеспечивает автоматического преобразования пользовательских типов друг в друга, даже если их значения принадлежать единому подмножеству. Данное ограничение введено для более строгого контроля. Зачастую подобные преобразования бывают необходимы. В этом случае можно воспользоваться **разыменованием**

пользовательского типа, заключающемся в выделении «обернутого» значение функцией **value**. Данная функция «отбрасывает» пользовательский тип, тем самым «обезличивая» преобразуемый элемент:

(10,20,15):Triangle:value ⇒ (10,20,15)

(10,20,15):Triangle:value:1:Circle ⇒ Круг радиусом 10

Следует отметить, что попытка применить операцию разыменования к базовым типам ведет к генерации ошибки **VALUEERROR**:

10:value ⇒ VALUEERROR

Ниже приводится функция, вычисляющая периметры различных геометрических фигур, расположенных в некотором списке:

```
// Список из пяти фигур
Figures << const ((3,4,5): Triangle, 10:Circle,
7:Circle, 1:Circle, (13,14,15): Triangle);
// Нахождение периметра обобщенной фигуры
pi << 3.1415;
fig_perimeter << funcdef figure {
 fig << figure:value; // выделение параметров фигуры
 // Формирование селектора по типу фигуры
 tag << ([(figure:type,Triangle),
 (figure:type,Circle)]:=):?;
 // Использование признака для выбора формулы
 tag^(
 // суммирование сторон треугольника
 {((fig:1,fig:2):+,fig:3):+},
 // формула для периметра круга
 {((2,pi):*,fig):*}
 ):.
 >> return;
};
// Получение списка периметров по списку фигур
all_perimeter << funcdef fig_list {
 // Вычисление одновременно для всех элементов списка
 (fig_list:[]:fig_perimeter) >> return;
};
```

Заключение

Предлагаемые механизмы предоставляют инструментальную поддержку для гибкой разработки функциональных параллельных программ. Возможно не только добавление новых обработчиков специализаций, но и изменение свойств ранее разработанных типов. Это позволяет достаточно гибко наращивать функциональные возможности параллельной программы. Работа выполнена при поддержке РФФИ № 02-07-90135.

Литература

1. *Легалов А.И., Кузьмин Д.А., Казаков Ф.А., Привалихин Д.В.* На пути к переносимым параллельным программам // Открытые системы, 2003. № 5 (май). С. 36-42.

2. *Легалов А.И.* Инструментальная поддержка процесса разработки эволюционно расширяемых параллельных программ // Проблемы информатизации региона. ПИР-2003/ Материалы 8-й Всероссийской научно-практической конференции. Красноярск, 2003. С. 132-136.

3. *Маурер У.* Введение в программирование на языке ЛИСП // М.: Мир, 1976. - 104 с.

РЕАЛИЗАЦИЯ ФУНКЦИЙ СТАНДАРТА MPI ДЛЯ ЭМУЛЯЦИИ ОБМЕНОВ СООБЩЕНИЯМИ МЕЖДУ УЗЛАМИ МНОГОПРОЦЕССОРНОЙ ВЫЧИСЛИТЕЛЬНОЙ СИСТЕМЫ

А.В. Лепихов

Челябинский государственный университет, г. Челябинск

В настоящее время вычислительные комплексы с массовым параллелизмом широко применяются для решения большого класса задач. Однако при использовании данных систем возникает сложная проблема отладки параллельных программ, не решенная в полной мере до настоящего момента.

Как известно отладка программы отнимает порядка 50% времени создания программы, а зачастую оказывается очень продолжительной. Сложность параллельных программ как таковых и их недетерминированное поведение превращают отладку параллельных программ в очень сложный для разработчика процесс.

Помимо приобретения коммерческого отладчика (например, TotalView [http://www.etnus.com] и PGDBG [http://www.pgroup.com]) существуют другие способы отслеживания событий программы при помощи вывода сообщений трассировки, распечатка значений переменных в заданном процессе и т. д. – весьма трудоемкий процесс. Предложенное решение проблемы выбора средств отладки основано на *эмуляции* параллельных процессов и обменов данными между ними с помощью стандартных средств ОС Windows.

Разработанный *эмулятор обменов сообщениями* представляет собой динамически компонуемую библиотеку, интерфейс которой есть подмножество функций стандарта MPI, реализующих асинхронный обмен сообщениями между процессами. Таким образом, эмулятор позволяет запускать параллельные программы непосредственно из среды

MS Visual C++ (без загрузчика) и использовать весь спектр средств встроенного отладчика.

Эмулятор обеспечивает представление процессов, запускаемых на процессорных узлах, в виде процессов ОС Windows, каждый из которых представляет собой совокупность следующих *потоков* (нитей): мастер, отправитель, получатель и терминатор. *Поток-мастер* выполняет собственно код процесса. *Поток-отправитель* обрабатывает массив, элементами которого являются очереди сообщений, передаваемых другим процессам программы. *Поток-получатель* обрабатывает очередь сообщений, поступающих от потоков-отправителей других процессов программы. *Поток-терминатор* выполняет аварийное завершение всех процессов программы, если поток-мастер одного из процессов выполнил функцию MPI_Abort.

Прием-передача сообщения от процесса *S* процессу *R* выглядит следующим образом (далее мы будем именовать потоки как *Master*, *Sender* и *Receiver*, а индексы имен будут указывать на принадлежность к процессу).

При выполнении функции посылки сообщения поток $Master_S$ добавляет в соответствующую очередь потока $Sender_S$ запись о данном сообщении и открывает ему семафор для начала передачи сообщения.

Поток $Sender_S$ создает в оперативной памяти процесса *S* область, доступную для потока $Receiver_R$, записывает в нее передаваемое сообщение и генерирует для $Receiver_R$ событие о необходимости начать прием. После этого поток $Sender_S$ переходит к ожиданию подтверждения от потока $Receiver_R$ о завершении приема. При получении подтверждения $Sender_S$ уничтожает ранее созданную область и памяти и закрывает семафор передачи сообщения.

Поток $Receiver_R$ выполняет перманентное ожидание события о необходимости начать прием. При наступлении такого события он обращается к области памяти, которую создал $Sender_S$, и считывает переданное им сообщение. После этого $Receiver_R$ высылает потоку $Sender_S$ подтверждение о завершении приема сообщения.

Разработанный эмулятор может быть использован для отладки параллельных программ, выполняющих обмен сообщениями на основе стандарта MPI. Переход от отладки к тестированию и обратно требует внесения минимальных изменений в исходные тексты, не влияющие на её работу с использованием стандартных реализаций MPI и изменения параметров их компоновки.

АВТОМАТНАЯ МОДЕЛЬ ПАРАЛЛЕЛЬНЫХ ВЫЧИСЛЕНИЙ

В.С. Любченко

ООО Специальное конструкторское бюро «Локальные автоматизированные системы», г. Новочеркасск

Введение

Рассматривая вопросы проектирования сложных дискретных систем, структурно представимых и/или разлагаемых на параллельно функционирующие подсистемы, можно обратить внимание, что в основе методологии их проектирования в основном лежат сети Петри [1-3]. Модель конечного автомата, если и рассматривается, то на заключительной стадии проектирования для реализации управления дискретными процессами.

Мнение, различающее и разделяющее по своим возможностям сети Петри и автоматы, обосновано лишь в силу бытующих представлений об ограниченных возможностях конечных автоматов. Но, как можно убедиться, уже незначительное усовершенствование автоматной модели придает свойства автоматам, которые могут поколебать главенствующее положение сетей Петри даже в такой традиционной для них области, как описание и реализация параллельных систем.

Проектирование

Формальная модель конечного автомата (КА) и сеть из множества КА – модель СМКА[4] предоставляют разработчику гибкие структурные средства для отображения параллельного взаимодействия, как на уровне отдельных компонентов, так и на уровне их множества. Другие технологии разработки программ «сверху вниз», «снизу вверх», структурный подход и т.п. таких возможностей, как правило, не имеют. В том же объектно-ориентированном программировании (ООП) вопросы параллельной работы объектов вынесены за рамки ООП.

Предлагаемая конечно-автоматная технология (КА-технология) использует модель со средствами описания параллелизма, которые по своим возможностям не уступают другим параллельным моделям. В то же время реализация базовой модели вычислений много проще. Кроме того, при необходимости, применяя стандартные приемы, несложно перейти от параллельного конечно-автоматного описания процессов к их тривиальному последовательному представлению в форме модели блок-схем.

Таким образом, в рамках КА-технологии можно легко и наглядно представлять структуру задач любой сложности в параллельном виде. Пример из [5] это демонстрирует на решении довольно простой задачи.

Более сложные примеры применения конечных автоматов можно найти в [6-8].

Кодирование

После описания структуры системы и алгоритмов работы ее частей в терминах формальной модели наступает момент ее реализации. Сейчас это сводится к программированию алгоритмов, представленных моделью блок-схем программ. В КА-технологии алгоритмы представлены формой КА-описания. И в ней для их (автоматов) реализации, во-первых, существуют формальные методы перехода от КА к эквивалентной блок-схеме (см. [9]), во-вторых, можно использовать давно известные программные методы реализации автоматов.

В то же время современные программные средства позволяют реализовать автоматы в форме, которая приближена к исходному описанию автоматов. В [5] показано, как это осуществимо в рамках КА-технологии. Там же имеется ссылка на DLL-библиотеку, реализующую автоматную модель и параллельную среду, в которою они «погружены».

При выбранном в рамках КА-технологии методе интерпретации автоматов является принципиальным отделение множества операторов программы от ее управления. Это полностью соответствует формальному определению программы в виде так называемой схемы программы[10], где программа представлена тройкой - память, операторы, управление. Такое представление удобно и для реализации выбранного способа интерпретации автоматов. Оно же оказывает сильное влияние на содержание дальнейших этапов разработки программ.

Кроме того, применение ООП и метода интерпретации автоматов позволяют достичь уровня абстракции, когда кодирование требует знаний, связанных только с самой автоматной моделью, но не с особенностью ее реализации.

Таким образом, **кодирование автоматов в рамках КА-технологии** основано на введении понятия динамичного объекта, когда любой объект может быть наделен алгоритмом поведения во времени. Алгоритм поведения объекта задается моделью конечного автомата. Язык описания автомата основан на табличной форме представления автоматов. Логика поведения объекта (таблица переходов автомата) отделена от методов автоматного объекта (предикатов и действий), связанных с реализацией его поведения во времени. Любые динамичные объекты могут выполняться параллельно.

Тестирование

После этапа кодирования приступают к процедуре тестирования отдельной программы и проекта в целом. Считается, что стоимость вы-

явления ошибки на этапе кодирования в два раза выше, чем на этапе проектирования, а ее обнаружение при тестировании обходится уже в 10 раз дороже.

Ошибки, к сожалению, есть всегда. Теория конечных автоматов определяет более высокий уровень защиты от ошибок уже на стадии проектирования. Существуют математические приемы, позволяющие не только проверить корректность алгоритма, но и применить к нему формальные операции - умножение, деление и т.п. для синтеза и анализа новых алгоритмов из уже проверенных.

В рамках КА-технологии применимы приемы тестирования и отладки, которые зачастую невозможны при других подходах. Отделение множества операторов от управления КА-программы позволяет проводить их независимый анализ. Можно легко строить и модифицировать логику работы программы на множестве ее операторов, проводя эксперименты чисто с логикой программы. Можно даже использовать одну уже проверенную и отлаженную логику (управление) в разных программах/объектах, т.е. *наследовать алгоритм* по аналогии с наследованием методов или данных в ООП.

Работу КА-программы легко отслеживать с помощью понятия *состояния* конечно-автоматного управления КА-программы. Множество таких состояний отражают состояние всей системы в целом. Тем самым легко выявляются точки зацикливания и/или некорректной работы, как отдельной подсистемы/подсистем, так и всей системы. При этом ошибки КА-программы сосредоточены в ее четко определенных структурных компонентах – предикатах, действиях или управлении.

В расширение методов проектирования программных систем свойства базовой автоматной модели позволяют использовать зарекомендовавшие себя формальные математические методы и другие эффективные методы проектирования и тестирования из теории и практики проектирования аппаратных средств.

Эксплуатация и сопровождение

Удачно разработанные проекты «живут» долго. При этом их сопровождением и эксплуатацией занимаются часто специалисты, не участвовавшие в их разработке. Для этого им нужна эксплутационная документация. Но если через некоторое время даже сам программист может с трудом разобраться в своей программе, то понятно, насколько сложен процесс сопровождения другими лицами.

Самым точным источником информации о программе является листинг. С листингом КА-программ работать очень легко, т.к. **отделение логики программы (управления) от ее операторов делает ее «самодокументируемой».** Это свойство позволяет быстро и точно вос-

становить алгоритм работы программы/объекта. Тот, кто хоть раз пытался построить по листингу программы ее блок-схему, знает о сложности такой процедуры.

Модель КА обобщает такие методы разработки программ, как таблицы решений и программирование с использованием переменной состояния. Эти методы удобны и эффективны не только для разработки программ, но и для последующего их сопровождения. Они часто предлагаются в качестве подходов, расширяющих возможности блок-схем.

Примеры применения

Автоматная модель и ее параллельный вариант – это универсальная алгоритмическая модель, пригодная для применения в любой прикладной области. На текущий момент она реализована в форме DLL библиотеки (см. [4]). Микроядро или, другими словами, виртуальная КА-машина, выполняет функции интерпретации модели КА, реализует параллельную работу компонент, управляя их запуском, синхронизацией и т.п. Разработанные на языке C++ средства описания автоматов расширяют объектные возможности языка, реализуя динамическую работу классов.

Прикладные области, где проходила «обкатку» данная параллельная модель и ее технология, достаточно разнообразны. Их диапазон - от простых в алгоритмическом плане бухгалтерских программ типа расчета заработной платы до сложных проектов типа системы управления технологическим процессом выращивания кристаллов с множеством динамически порождаемых параллельно функционирующих объектов. Последние реализуют процессы съема данных с датчиков, выдачу управляющих воздействий на объект и автоматные алгоритмы работы драйверов с разнообразной аппаратурой, а также процессы отображения и расчета.

Одна из сложных областей эффективного применения КА – игры. Другая предметная область такого же класса – системы управления и/или функционирования в реальном времени. В этих областях в полной мере присутствуют элементы, которые сложно реализовать при обычных подходах. Это, прежде всего, параллельный характер работы составных частей системы и их синхронизация в реальном времени. В основу решения данных задач положена концепция микроядра, которое реализует параллельные механизмы среды функционирования процессов.

Еще раз подчеркнем, что КА-технология хорошо подходит для работы в реальном времени. Реализованный проект системы управления технологическим процессом обеспечивает реакцию системы в пределах 0.01 сек (можно обеспечить и лучшую) в среде Windows. Это не хуже, чем гарантируют специализированные системы такого же типа.

Выводы

Чем сложнее проект, тем эффективнее применение конечно-автоматной модели. КА-технология решает проблемы структурного и алгоритмического проектирования систем. Она органична для любой прикладной области и потому может быть легко внедрена всюду. Это аналогично распространению ООП на достаточно специализированные прикладные области и системы программирования.

КА-технология позволяет внедрить весьма эффективные математические методы разработки, тестирования и отладки программ. Автоматная модель стандартизирует процесс разработки, соединяя в себе наиболее общие подходы проектирования программ, которые в малой степени зависят от способа их дальнейшей реализации.

Табличное представление КА легко реализовать аппаратно, что позволяет говорить уже не об интерпретации, а о прямой реализации автоматов. И хотя метод интерпретации снижает скорость работы программы, современный уровень развития аппаратных средств практически снимает эту проблему. В то же время аппаратная реализация - перспективное направление, которое определяет *новые архитектурные принципы построения вычислительных систем*. Реально значительное увеличение быстродействия аппаратных средств.

КА-технология – параллельная технология. Параллельное решение по структуре проще последовательного. Такую структуру легче изменять, т.к. затрагиваются компоненты, которые в достаточной степени независимы. Сопровождение и эксплуатация параллельной системы также гораздо проще и дешевле.

Автоматная модель наделяет объекты динамическими свойствами. Это расширяет возможности технологии ООП, устраняя один из основных ее недостатков. Включение в описание класса алгоритма его работы во времени делает это описание действительно *всеобъемлющим*. Пока же вопросы динамики работы объектов в ООП выходят за рамки определения классов.

Отделение управления от операторов программы позволяет создавать совершенно уникальный тип библиотек - библиотеки алгоритмов. Создание коллекций, будь то библиотеки программ, объектов, шаблонов и т.д. и т.п., обычно создает хорошие предпосылки для развития технологий. По крайней мере, об этой возможности нужно знать, а реализация и механизмы ее применения во многом могут быть аналогичны уже существующим.

Литература

1. *Питерсон Дж.* Теория сетей Петри и моделирование систем // Пер. с англ. – М.: Мир, 1984. – 264с.

2. *Юдицкий С.А., Магергут В.З.* Логическое управление дискретными процессами // Модели, анализ, синтез: - М.: Машиностроение, 1987. – 176.: ил.

3. *Рамбо Дж., Якобсон А., Буч Г.* UML: специальный справочник // – СПб.: Питер, 2002. – 656 с.

4. *Любченко В.С.* От машины Тьюринга к машине Мили // «Мир ПК», 2002. №8. http://www.osp.ru/pcworld/2002/08/130.htm

5. *Любченко В.С.* О бильярде с Microsoft C++ 5.0 // «Мир ПК», 1998. № 1. С. 202-206.

6. *Любченко В.С.* Новые песни о главном (римейк для программистов). «Мир ПК», 1998. № 6. С. 114-119.

7. *Буч Г.* Объектно-ориентированное проектирование с примерами применения // Пер. с англ. - М.: Конкорд, 1992. – 519 с.

8. *Шлеер С., Меллор С.* Объектно-ориентированный анализ: моделирование мира в состояниях:// Пер. с англ. - Киев: Диалектика, 1993. - 240 с.

9. *Баранов С.И.* Синтез микропрограммных автоматов // Л.: Энергия, 1979. – 232 с.

10. *Любченко В.С.* Автоматные схемы программ (с исторической ремаркой автора). http://www.softcraft.ru/auto/ka/ash/ash.shtml

К РЕШЕНИЮ ПРОБЛЕМЫ ОБЕДАЮЩИХ ФИЛОСОФОВ ДЕЙКСТРЫ

В.С. Любченко

ООО Специальное конструкторское бюро «Локальные автоматизированные системы», г. Новочеркасск

Введение

В [1] приведено решение задачи Дейкстры и описана модель философа. Чтобы в ней исключить тупики, нужно попытаться или совсем устранить переходы, ведущие в соответствующее состояния, или хотя бы попытаться уменьшить их число, а циклы, порождающие тупики, желательно «разорвать». В этих целях мы далее 1) внесем изменения в поведение компонентов системы и в их связи между собой, 2) после каждого изменения «вычислим» новое поведение системы, 3) проведем анализ нового поведения системы.

Цель данной работы – демонстрация математического аппарата точного расчета поведения сложных систем. Приложение его к задаче Дейкстры, как мы увидим, позволяет строго доказать наличие тупиков в ней, выявить их причину, а уж затем внести коррективы в поведение элементов системы.

Подобные результаты для данной задачи получены, пожалуй, впервые, т.к. известные решения часто ограничиваются лишь общими рассуждениями о возможных тупиках и подходах к их устранению.

Не спеши, если сам не готов, или … модель «умного» слуги!

Из анализа модели слуги (см. рис. 8, 9 в [1]) и эквивалентной модели философа (рис. 10 в [1]) следует, что он может попросить вилку у соседа даже в том случае, когда его вилка отдана другому. Логичнее же просить вилку когда, когда своя вилка не занята. Признак ее свободы – состояние ps (см. рис.7 [1]). И если ранее было: слуга переходит из состояний *f*1 в состояние *f*2 сразу при поступлении сигнала «хочу есть» (*x*5=1) от философа, то теперь синхронизируем этот переход с состоянием *ps* модели вилки.

Дополним модель слуги входным каналом – *x*8. По нему будет поступать информация о внутреннем состоянии *ps* вилки. Теперь аналитическая форма автомата слуги может иметь следующий вид:

F:

$$f1 = \{f2(x5x8/-)\} \quad (1)$$

$$f2 = \{f1(x6x7/y1)\}.$$

При этом дополнение автомата F - автомат ^F, который нам будет необходим для вычисления эквивалентного автомата системы, будет следующим:

^*F*:

$$f1 = \{ f1(\hat{}x5x8/-), f1(x5\hat{}x8/-), f1(\hat{}x5\hat{}x8/-)\} = \{ f1(\hat{}x5/-), f\,1(x5\hat{}x8/-)\}, \quad (2)$$

$$f\,2 = \{ f\,2(\hat{}x6/-), f\,2(\hat{}x7/-)\}.$$

Сетевая модель блока «философ/вилка» – первое решение

Сетевая модель философа, отражающая новое поведение слуги, будет следующей:

P:

$$ps = \{pl(f\,2/-), \mathrm{pr}(f\,1x2/-)\},$$
$$pl = \{ps(f\,1/-)\}, \quad (3)$$
$$pr = \{ps(x1/-)\}.$$

F:

$$f\,1 = \{ f\,2(psx5/-)\}, \quad (4)$$
$$f2 = \{ f\,1(plx6/y1)\}.$$

Здесь переменной *x*8 соответствует состояние *ps*, а ее отрицанию – состояние *pl*. При этом дополнения компонентных автоматов *P* и *F* будут следующими:

^*P*:

ps = {ps(f1^x2/-)}, (5)

pl = {*pl*(*f* 2/-)},
pr = {*pr*(^x1/-)}.
^*F*:

f 1 = { f 1(ps^x5/-), f 1(plx5/-), f 1(prx5/-)}, (6)

f 2 = { *f* 2(*pr*/-), *f* 2(*ps*/-), *f* 2(^x6/-)}.

Вычислим эквивалентный автомат *M*1, определяющий новое поведение системы – отдельного философа, где:

*M*1 = *P*⊗*F* = *P*×^*F*∪^*P*×*F*∪*P*×*F*,

В результате (опустив промежуточные вычисления) автомат *M* будет следующим:

*M*1:

*psf*1 = {*prf*1(x2^x5/-), *psf*2(^x2x5/-), *prf*2(x2x5/-)}, (7)

*psf*2 = {*plf*2(-/-)},
*plf*1 = {*psf*1(^x5/-)},
*plf*2 = {*plf*1(x6/y1)},
*prf*1 = {*psf*1(x1/-)},
*prf*2 = {*psf*2(x1/-)}.

Граф автомата *M*1 показан на рис. 1.

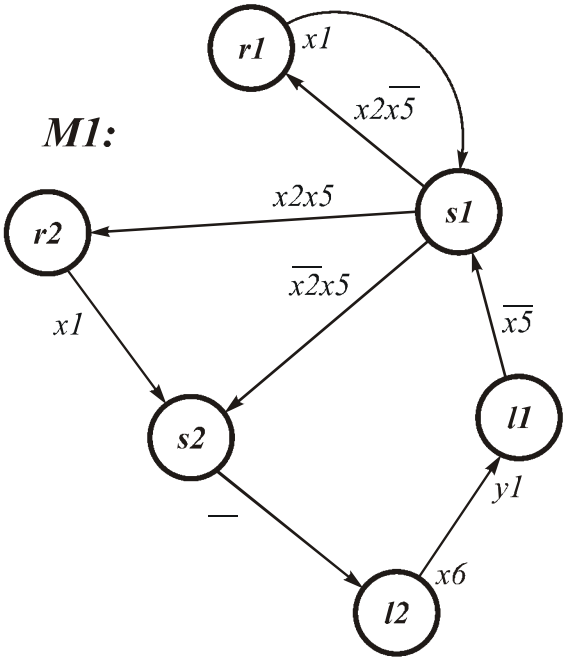


Рис. 1. Граф автомата M1

Видно, что поведение философа изменилось и в автомате стало меньше переходов в состояние «*r*2» (напомним, что оно может быть причиной тупика). Хотя нам и не удалось совсем исключить попадание в него. Исчез переход между состояния «*l*1» в «*s*2»! Это один из перехо-

дов цикла, определяющего поведение очень жадного до еды философа. Причина также не устранена совсем, но длина цикла увеличилась. Теперь «жадность» философа ограничена тем, что после поглощения очередной порции спагетти он должен отказаться на какое-то время от еды ($x5=0$). За это время его голодный сосед, который ждет освобождения вилки, возможно, успеет ее «умыкнуть».

Поешь сам, а потом помоги соседу

Из анализа сетевой модели следует, что вилка может быть отдана соседу и в тот момент, когда сам философ запросил есть. Это видно из следующего. Пусть слуга находится в состоянии $f1$ и его вилка свободна. Если в какой-то момент приходит запрос от соседа дать ему вилку ($x2=1$), то она будет отдана, если даже если в этот же самый момент появится сигнал «хочу есть» ($x5=1$). Произойдет это потому, что слуге нужен будет один такт, чтобы перейти в состояние $f2$, которое переводит вилку из состояния ps в состояние pl. Но вилка-то будет в этот момент в состоянии pr, т.к. она уже взята соседом на предыдущем такте работы вилки.

В эквивалентной модели (7) этот момент отражает состояние $r2$, в которое автомат перейдет из состояния $s1$. Но состояние $r2$ – это состояние возможного тупика: из него философ не выйдет до тех пор, пока сосед не вернет ему вилку (пока $x1$ не примет значение истинно). Но сосед в это время тоже может ждать уже свою вилку, которую он точно также до этого в такой же ситуации мог опрометчиво отдать.

Такую ситуацию можно исключить, если разрешить отдавать свою вилку только в том, случае, когда ее владелец не голоден, т.е. только при $x5=0$. Но для этого нужно добавить в модель вилки еще один входной канал. А раз уж мы ввели в модель вилки сигнал «хочу есть» – $x5$, то это повлечет за собой изменение условия перехода из состояния ps в pl. Уберем синхронизацию с состоянием $f2$ слуги ($x4$), а разрешим взять вилку, если $x5=1$. С одной стороны это равносильно, т.к. и состояние $f2$ философа и сигнал $x5$ соответствуют желанию философа поесть. Но есть и отличие: сигнал $x5$ примет раньше истинное значение, чем сигнал $x4$ (переход слуги в состояние $f2$). В результате вилка будет выделена быстрее.

Модель вилки, которая больше любит своего философа, следующая:

P:

$$ps = \{pl(x5/-), pr(x2\hat{}x4\hat{}x5/-)\}, \quad (8)$$

$$pl = \{ps(x3/-)\}, pr = \{ps(x1/-)\}.$$

Теперь вилка перейдет из состояния ps в состояние pl сразу при получении сигнала от своего философа «хочу есть». Это более «независимая» вилка, т.к. меньше связана с поведением слуги. Сосед не сможет забрать вилку до тех пор, пока «свой» философ не будет накормлен.

Сетевая модель блока «философ/вилка»

Сетевая модель, отражающая еще одно поведение вилки будет следующей:

P:

ps = {*pl*(*x*5/-), *pr*(*x*2^x5*f*1/-)},

pl = {*ps*(*f*1/-)}, (9)

pr = {*ps*(*x*1/-)}.

F:

f1 = {f2(x5ps/-)}, (10)

f2 = {f1(x6pl/y1)}.

Дополнения компонентных автоматов сети примут следующий вид:

^P:

ps = {ps(^x2^x5/-), ps(x2^x5f2/-)}, (11)

pl = {*pl*(*f*2/-)},

pr = {*pr*(^*x*1/-)}.

^*F*:

f1 = {f1(pl/-), f1(pr/-), f1(^x5/-)}, (12)

f2 = {f2(ps/-), f2(pr/-), f2(^x6/-)}.

Эквивалентный автомат M2, определяющий новое поведение системы, будет следующим:

M2 = P⊗F = P×^F ∪ ^P×F ∪ P×F:

psf1 = {prf1(x2^x5/-), plf2(x5/-)}, (13)

psf2 = {plf2(x5/-)},
plf1 = {psf1(-/-)},
plf2 = {plf1(x6/y1)},
prf1 = {psf1(x1/-)},
prf2 = {psf2(x1/-)}.

Графическая форма эквивалентного автомата представлена на рис. 2.

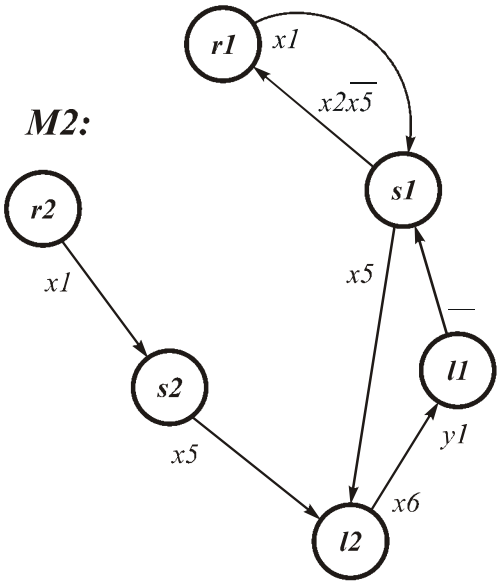


Рис. 2. Граф автомата M2

Введенные ограничения еще больше упростили поведение системы: исключен переход из состояния s1 в r2, а переход из состояния s1 направлен сразу в состояние l2, а не в состояние s2, как было до этого (см. (7).

Теперь, во-первых, мы, наконец-то, избавились от перехода в одно из тупиковых состояний *r*2. И это существенное достижение. Вообще-то из результирующего автомата можно исключить переходы, связанные не только с состоянием *r*2, но и с состоянием *s*2, т.к. в него возможен переход только из состояния *r*2, которое мы собираемся исключить, т.е. в окончательном виде эквивалентный автомат будет следующим:

*M*3:

psf1 = {prf1(x2^x5/-), plf2(x5/-)}, (14)

*plf*1 = {*psf*1(-/-)},

*plf*2 = {*plf*1(*x*6/*y*1)},

*prf*1 = {*psf*1(*x*1/-)},

Формально состояния *r*2, *s*2 в системе все же есть, но в них, если модели элементов системы функционируют в соответствии с заданным поведением, система не должна попасть. Но если по каким-то внешним причинам это все же случится, то из *r*2 система перейдет в *s*2, далее в *l*2 (см. граф *M*2), а уж потом все пойдет обычным путем. Главное, что и в этом случае будет выход из состояния *r*2.

Сравнение решений

Мы имеем три решения задачи. Первое решение представлено в [1], вторую модель представляет эквивалентный автомат (7), третью – (13). Первое решение дает большую свободу для компонентных моделей философа. Но оно же и наименее надежно, т.к. система имеет боль-

ше шансов попасть в тупиковые состояния или в циклы. И если строить эквивалентный автоматный граф для всей задачи, то он будет содержать в данном случае 6^5=7776 внутренних состояний, из которых какое-то число будут тупиковых.

Как минимум одно тупиковое состояние мы можем сразу назвать. Оно соответствует состояниям *l*2 всех пяти философов (его можно обозначить как – *l*2*l*2*l*2*l*2*l*2). Какие другие существуют комбинации тупиковых состояния среди всего множества состояний системы дать ответ сложно. Можно лишь сказать, что их количество ограничено числом 3^5=243. В принципе можно разработать процедуру, которая, анализируя общий граф, определит, есть ли в нем состояния, соответствующие комбинации состояний из множества тупиковых состояний, и есть ли в них и из них переходы в другие состояния.

Эквивалентный граф для системы из автоматов (7) имеет то же число состояний. Единственное отличие – меньшее число переходов. Существенные отличия от двух предыдущих автоматов иметь автомат (13). Он формально тоже имеет это же число состояний, но среди них рабочих – 4^5=1024, из которых тупиковых состояний – одно! Это упомянутое выше *l*2*l*2*l*2*l*2*l*2. Все философы не могут находиться одновременно в состоянии *r1*, а в остальных случаях в системе тупики возникать не должны. Эта модель сразу определяет множество состояний, в которые она, просто не должна попасть. Их число – 7776-1024=6752. Из этого видно соотношение между рабочими состояниями системы и ее возможными тупиковыми состояниями. Среди них истинно тупиковые ограничены числом – 243 (см. выше), а в остальные состояния система либо не попадет, либо найдет из них выход.

Выводы

Известно множество разных решений данной задачи. Из использованных базовых моделей, возможно, самые популярные сети Петри. Примером решения на базе другой модели служит решение, рассмотренное в [2]. Приведенное автоматное решение позволит с иных позиций взглянуть на проблемы реализации параллельных систем вообще и на вопросы применения автоматов для реализации параллелизма в частности. Как правило, понятие конечного автомата связывается только с последовательными процессами и мало используется при переходе к параллельным системам.

Действительно, конечный автомат, как модель отдельной компоненты, больше приспособлен для описания последовательного процесса, хотя и на этом уровне у него есть возможности для задания параллелизма. Но, как видно, множество автоматов моделирует системы параллельного типа не хуже, чем специально разработанные для таких систем

модели (например, те же сети Петри). Представленное решение задачи Дейкстры служит достаточно убедительным аргументом в пользу данного высказывания.

Литература

1. *Любченко В.С.* Обедающие философы Дейкстры (о модели параллельных процессов) // http://www.softcraft.ru/auto/ka/fil/fil.shtml

ВЫЧИСЛИТЕЛЬНЫЙ КЛАСТЕР ВЦ РАН

Г.М. Михайлов[1], М.А. Копытов[1], Ю.П. Рогов[1], А.М. Чернецов[1], А.И. Аветисян[2], О.И. Самоваров[2]

[1]Вычислительный центр имени А. А. Дородницына Российской академии наук (ВЦ РАН), г. Москва
[2]Институт Системного Программирования Российской академии наук (ИСП РАН)

Введение

Высокопроизводительный вычислительный кластер ВЦ РАН [1] был введен в эксплуатацию в 2003 г. Кластер построен на базе процессоров Intel Xeon с использованием высокопроизводительной сетевой технологии Myrinet 2000. Кластер представляет собой единый вычислительный комплекс коллективного пользования и предназначен для выполнения широкого класса параллельных программ.

Кластер ВЦ РАН спроектирован и построен в соответствии с требованиями, предъявляемыми к архитектуре высокопроизводительных кластерных вычислительных систем [2]. Вычислительные системы данной архитектуры позволяют эффективно решать параллельные задачи, характеризующиеся наличием интенсивных обменов данными в процессе вычислений.

В статье описывается системное программное обеспечение, установленное на кластере ВЦ РАН, представлены результаты тестирования кластера, а также результаты и опыт эксплуатации кластера при решении реальных прикладных задач.

Системное программное обеспечение

Системное программное обеспечение кластера строится на базе операционной системы Linux RedHat и системы управления кластерами OSCAR [3].

В состав OSCAR входят средства, позволяющие инсталлировать узлы кластера с общего, заранее настроенного образа операционной сис-

темы, что обеспечивает однородность узлов кластера. Это так же позволяет легко модифицировать конфигурацию кластера, добавляя новые вычислительные узлы, сохраняя при этом однородность настроек системного программного обеспечения. В случаях аппаратных или программных сбоев вычислительные узлы могут быть легко восстановлены.

В состав OSCAR также входит набор утилит, обеспечивающих управление и администрирование кластера.

Так же в состав OSCAR включены система пакетной обработки заданий OpenPBS и планировщик очередей MAUI.

OpenPBS и MAUI обеспечивают эффективную эксплуатацию кластера, как единого вычислительного ресурса коллективного пользования и позволяют решать следующие задачи:

- формирование очередей заданий пользователей;
- гибкое управление очередями заданий;
- выделение ресурсов кластера по запросам;
- освобождение ресурсов кластера после выполнения заданий;
- поддержка гибкой системы приоритетов, позволяющей распределять ресурсы по заданным правилам;
- поддержка алгоритмов, обеспечивающих равномерную и максимально полную загрузку вычислительной системы в целом.

Удаленный доступ на кластер через защищенный протокол SSH (Secure Shell) может быть организован с любой рабочей станции (в том числе и WINDOWS), имеющей доступ в ту же сеть, что и управляющий узел кластера.

В качестве базового параллельного окружения на кластер установлена специально оптимизированная для работы с Myrinet 2000 реализация MPI – mpich-gm.

В состав mpich-gm входят утилиты и библиотеки, которые совместно со свободно распространяемыми компиляторами gcc 3.2.2 (C/C++, Fortran 77), а также коммерческим компилятором Intel Fortran 77/90 7.1 позволяют разрабатывать параллельные MPI программы.

Результаты тестирования

На кластере ВЦ РАН был проведен комплекс тестов определяющих характеристики, как его отдельных систем: коммуникационных сред, вычислительных узлов, так и всего кластера в целом.

При тестировании коммуникационных сред кластера определялись такие характеристики, как пропускная способность и латентность. Наибольший интерес представляют характеристики вычислительной сети кластера, которые определялись как для низкоуровневых коммуникаций, так и для коммуникаций уровня MPI. В качестве тестовых программ использовались PMB (Pallas MPI Benchmark) [4], а также тесты,

предоставленные производителем Myrinet 2000. На рис.1 и 2 представлены результаты тестирования пропускной способности вычислительной сети кластера для случая односторонней передачи данных.

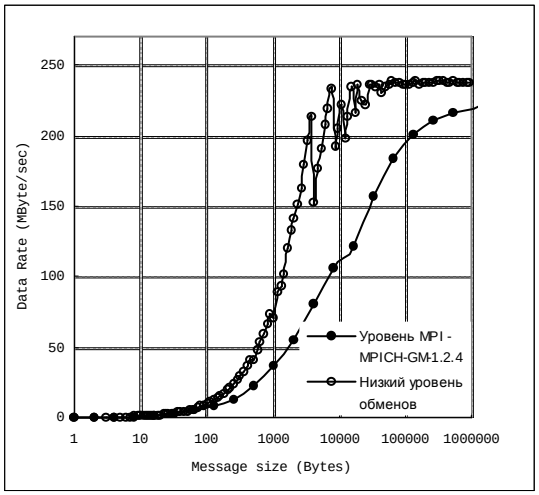


Рис. 1. Пропускная способность вычислительной сети кластера ВЦ РАН для односторонних обменов низкого уровня (уровень GM) и уровня MPI

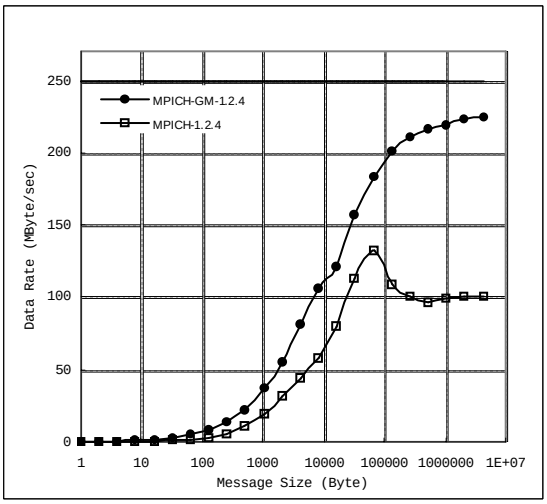


Рис. 2. Пропускная способность вычислительной сети кластера ВЦ РАН для разных реализаций MPI – mpich-gm-1.2.4 и mpich-1.2.4

На рис. 3 и 4 представлены результаты тестирования латентности вычислительной сети кластера для случая односторонней передачи данных.

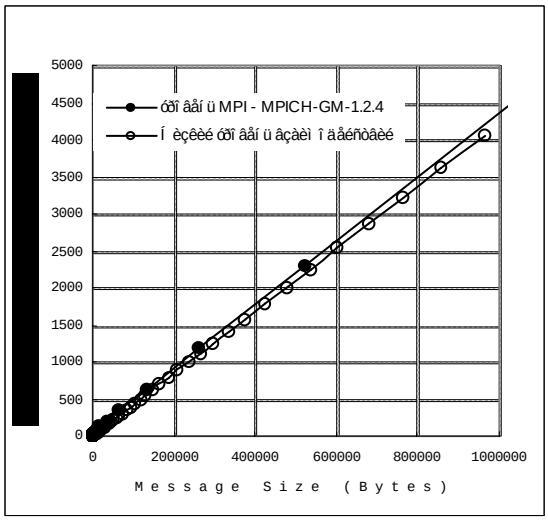


Рис. 3. Латентность вычислительной сети кластера ВЦ РАН для односторонних обменов низкого уровня (уровень GM) и уровня MPI

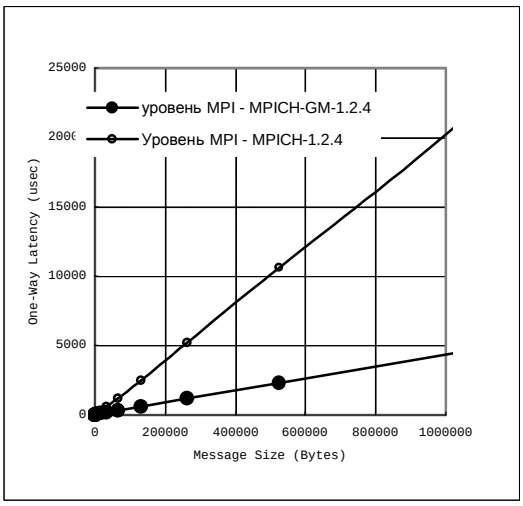


Рис. 4. Латентность вычислительной сети кластера ВЦ РАН для разных реализаций MPI – mpich-gm-1.2.4 и mpich-1.2.4

Результаты тестирования показали, что:

- пропускная способность вычислительной сети кластера ВЦ РАН для коммуникаций низкого уровня (уровня GM) равна не менее 237.5 MB/sec и латентность не более 10.2 usec. В тоже время по результатам тестирования представленным на Web-сайте производителя Myrinet 2000 [5] для узлов построенных на базе процессоров Intel P4 и материнской платы Supermicro P4DL6 (64bit/66Mhz PCI) были получены следующие характеристики: пропускная способность 240 MB/sec, латентность 8.5 usec;
- пропускная способность вычислительной сети кластера ВЦ РАН для базовых коммуникаций уровня MPI в случае реализации оптимизированной для работы с Myrinet 2000 составляет не менее 224.4 MB/sec, латентность не более 11.06 usec;
- пропускная способность вычислительной сети кластера ВЦ РАН для базовых коммуникаций уровня MPI в случае реализации использующей стек протоколов TCP/IP составляет не менее 101.09 MB/sec, латентность не более 43.18 usec.

Анализируя полученные результаты, следует подчеркнуть, что:

- параметры (пропускная способность, латентность) вычислительной сети кластера ВЦ РАН, полученные экспериментально соответствуют эталонным параметрам, представленным производителем Myrinet 2000;
- эффективность оптимизированной под Myrinet 2000 (mpich-gm) реализации MPI близка эффективности коммуникаций низкого уровня (уровня GM);
- накладные расходы на стек протоколов TCP/IP серьезно снижают эффективность коммуникаций уровня MPI. Следовательно, эффективные межузловые взаимодействия могут быть обеспечены только с использованием специально оптимизированной для работы с Myrinet 2000 реализации MPI.

Необходимо отметить, что в данной статье проводится анализ параметров вычислительной сети для базовых коммуникаций типа «точка-точка», однако аналогичные результаты были получены также для групповых и коллективных коммуникаций.

Также проводилось тестирование кластера, которое предполагало определение таких характеристик, как его производительность и эффективность. Данные характеристики были получены для задачи уровня MPI приложения – теста HPL (High Performance Linpack) [6]. Отметим, что на основе теста HPL строится список Top500[7], а также список Top50 России [8].

Максимальная производительность (*R*max) на тесте HPL была получена при размерности задачи (*N*max) 57000x57000 и равнялась 59.4 Gflops. При этом эффективность, то есть отношение полученной на тес-

те HPL производительности (*R*max) к пиковой производительности кластера (*R*pic), равна 71,4%.

Анализ первых 100 кластеров из Top500 показал, что эффективность кластеров на базе платформы *X*eon с использованием сетевой технологии Myrinet находится в диапазоне 50-70%, т.е. эффективность кластера ВЦ РАН близка к максимально возможной. Необходимо отметить, что данный результат получен для увеличенной с 16 до 32-х гигабайт оперативной памяти кластера. В исходной конфигурации кластера (16 гигабайт оперативной памяти) была получена эффективность 67% (см. рис. 5-6). Таким образом, увеличение памяти в два раза позволило поднять общую эффективность кластера на ~5%.

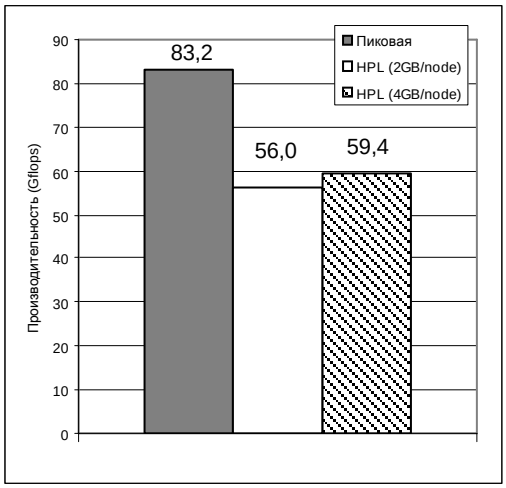


Рис. 5. Производительность кластера ВЦ РАН для разного объема оперативной памяти – 2GB на узел и 4GB на узел

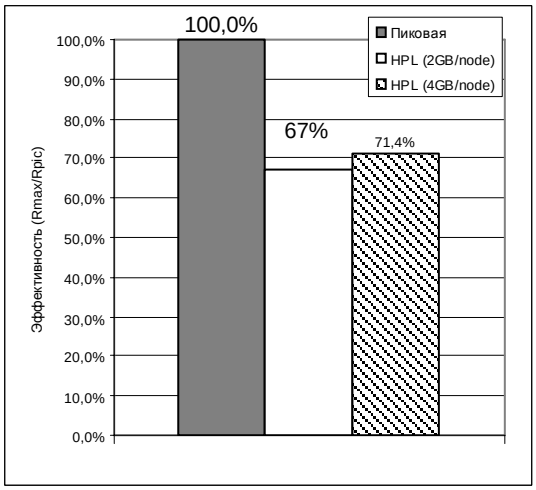


Рис. 6. Эффективность кластера ВЦ РАН для разного объема оперативной памяти – 2GB на узел и 4GB на узел

Важным преимуществом кластерных вычислительных систем является оптимальное соотношение цена/качество. Для данной системы это соотношение составляло чуть менее 1 доллара США за Mflops, что являлось одним из лучших на момент построения системы. В настоящее время соотношение цена/качество не более 0,5 доллара США за Mflops.

Заключение

В процессе эксплуатации кластера на нем решались различные классы задач:

1) длительные задачи-тесты (HPL, PMB, Perftect), требующие загрузки всех узлов кластера;

2) длительные задачи от 3-х до 5-ти суток, от 4-х до 8-ми узлов;

3) средней длительности от 2-х до 4-х суток, от 3-х до 5-ти узлов;

4) отладка - до 2-х узлов, в пределах одних суток.

На базе кластера был организован учебный класс для обучения студентов МФТИ курсу параллельных вычислений.

В настоящее время ведутся работы по модернизации кластера ВЦ РАН, включающие в себя установку последних версий системного программного обеспечения. ИСП РАН, как член ассоциации Gelato [9] ведет работы по интеграции в OSCAR программных продуктов собственной разработки, часть из которых будет также установлена на кластер ВЦ РАН. В частности предполагается установить:

- интегрированный интерфейс системы управления кластера, который обеспечивает безопасный доступ пользователей к ресурсам вычислительного кластера через интерфейс Web-броузера;
- систему активного мониторинга аппаратуры, которая обеспечивает мониторинг жизненно важных параметров аппаратуры кластера. При этом обеспечивается автоматическое завершение работы, как отдельных узлов, так и кластера в целом в случаях возникновения аварийных ситуаций;
- систему управления питанием кластера, обеспечивающую удаленное управление питанием, как отдельных узлов, так и кластера в целом.

Кластер ВЦ РАН был построен, введен в эксплуатацию и поддерживается специалистами ИСП РАН, ВЦ РАН и компании C.I. Technology [10].

С 2004 года работа поддерживается грантом РФФИ 04-07-90346 (руководитель проекта академик РАН А.А. Петров).

Литература

1. *Михайлов Г.М., Копытов М.А., Рогов Ю.П., Самоваров О.И., Чернецов А.М.* Параллельные вычислительные системы в локальной сети ВЦ РАН // Вычислительный Центр им. А.А. Дородницына РАН. Москва. 2003.

2. ISP HPC. http://www.ispras.ru/groups/ctt/clusters.html
3. OSCAR. http://www.gelato.org/software/view.php?id=1_18
4. Pallas MPI Benchmark. http://www.pallas.com/e/products/pmb/
5. Myricom Inc. http://www.myri.com
6. HPL (High Performance Linpack). http://www.netlib.org/benchmark/hpl/
7. Top500. http://www.top500.org
8. Top50. http://www.parallel.ru
9. Gelato. http://www.gelato.org/
10. C.I. Technology. http://www.citech.ru/

РЕГУЛЯРИЗУЮЩИЕ АЛГОРИТМЫ ГРАНИЧНО-ЭЛЕМЕНТНОГО РАСЧЕТА УПРУГИХ ТЕЛ С ТОНКИМИ ЭЛЕМЕНТАМИ СТРУКТУРЫ, РАСПРЕДЕЛЕННОГО НА КЛАСТЕРЕ РАБОЧИХ СТАНЦИЙ

А.И. Олейников, К.С. Бормотин

Комсомольский-на-Амуре государственный технический университет, г. Комсомольск-на-Амуре

Введение

Для описания напряженно-деформированного состояния сред применяется подход, основанный на теории интегральных уравнений, и его численная реализация методом граничных элементов (МГЭ).

Решение задач при наличии малых и тонких областей, в связи с близостью границ тонких элементов структуры и использованием интегральных уравнений для перемещений, сопряжено с очень высоким порядком и плохой обусловленностью соответствующей системы алгебраических уравнений. Эти обстоятельства приводят к потере точности, вычислительной неустойчивости и, в конечном итоге, к некорректности расчета. В этих условиях получение удовлетворительного численного решения представляет собой одну из важных задач вычислительного моделирования.

Эта задача решается различными методами регуляризации и определяется наиболее эффективный.

1. Математическая постановка задачи и гранично-интегральная формулировка

Рассмотрение метода расчёта напряжённого состояния кусочно-однородных тел основывается на непрямом методе граничных элементов, предназначенного для решения двумерных краевых задач в случае однородного изотропного линейно-упругого тела [1].

Рассматривается двумерная линейно-упругая область, ограниченная контуром. При заданных на контуре значениях (т.е. поверхностных нагрузках, перемещениях) требуется определить напряжения (и перемещения) внутри данной области. Напряжения и смещения во внутренней точке плоскости, вызванные действием фиктивной нагрузки (неизвестной, действующей вдоль контура), представляются в интегральном виде, где плотность потенциала представляется функцией Грина. Так как интегральные уравнения представляют суперпозицию фундаментальных решений, то будут удовлетворены все уравнения линейной теории упругости.

Чтобы решить некоторую заданную граничную задачу, необходимо ещё удовлетворить граничным условиям на контуре. Для этого в интегральных уравнениях нужно перейти к пределу при стремлении полевой точки к контуру. Удовлетворение заданным граничным значениям приводит к системе сингулярных уравнений.

Для решения интегрального уравнения применяется квадратурный метод прямоугольников, при котором интегральное уравнение аппроксимируется системой линейных алгебраических уравнений. Граничный контур разбивается на отрезки, в средней точке которого определяются результирующие граничные значения.

Метод граничных элементов, рассмотренный выше, предназначен для решения двумерных краевых задач в случае однородного изотропного линейно-упругого тела.

Задача теории упругости для кусочно-однородных тел основывается на рассмотрении тела, состоящего из фаз. Каждая из фаз считается однородной изотропной и линейно-упругой со своими упругими постоянными. Описание напряжённо-деформированного состояния всего тела осуществляется вектором перемещения, тензорами деформаций и напряжений для каждой фазы.

Краевая задача для кусочно-однородного тела заключается в задании обычных условий для смещений и напряжений на «свободной» части контуров, а также условий непрерывности смещений и усилий на поверхности контакта подобластей.

2. Методы решения интегральных уравнений для тел с тонкими слоями

При численном решении система интегральных уравнений с использованием квадратурной формулы прямоугольников приводится к системе линейных алгебраических уравнений

$$Ax = b.$$

Для определения эффективности методов решалась тестовая задача о тонком покрытии отверстия в пластине в условиях плоской деформации. При толщине кольца $h = 0.01мм$ порядок матрицы системы линейных уравнений достигал 7624, определитель оценивался равным $3{,}189^{-24313}$. Это говорит об ухудшении вычислительных качеств задачи. Вычислительный процесс на основе метода Зейделя или квадратного корня терял устойчивость.

Задача регуляризации сводится к минимизации функционала

$$f_1(x) = \| Ax - B \|^2 \qquad (1)$$

или

$$f_2(x) = \| Ax - B \|^2 + \alpha \| x \|^2, \quad (2)$$

где α - параметр регуляризации.

Градиентный метод с функцией $f_1(x)$ и $f_2(x)$ (регуляризованный градиентный метод) использовался с априорным заданием градиентного шага. Расчеты показали плохую сходимость методов – невязка от итерации к итерации изменяется на очень малую величину и поэтому число итераций должно быть большим.

Метод регуляризации Тихонова заключается в минимизации функционала (1) [2]. Алгоритм минимизации состоит из формирования сходящейся к нулю последовательности $\{\alpha_p\}$ и поиска минимума при закрепленной величине $\alpha = \alpha_p$. Минимум данного функционала при $\alpha = \alpha_p$ обеспечивает корень уравнения:

$$f'(x) = 2A^*(Ax - b) + 2\alpha_p(x - x_0) = 0.$$

Отсюда получим

$$\left(A^*A + \alpha_p E\right)x = A^*b + \alpha_p x_0,$$

где A^* – транспонированная матрица A, E – единичная матрица. После решения этой системы выбираем следующее значение α_p. В качестве вектора x_0 при α_0 используется вектор граничных условий, а в каждой последующей итерации – регуляризованное приближение *x*, полученное в предыдущей.

При решении этим методом использовалась нормировка, иначе численное решение сильно отклоняется от аналитического.

Проксимальный метод заключается в построении последовательности $\{x_k\}$ по следующему правилу:

$$x_{k+1} = pr(x_k, \beta_k), \; k = 0,1, \mathbf{h}$$

На *к*-м шаге процесса для определения очередного приближения x_{k+1} нужно решить следующую задачу минимизации при $x = x_k$, $\alpha = \alpha_k$:

$$\varphi(x, x_k, \beta_k) = \frac{1}{2}\left|x - x_k\right|^2 + \beta_k f(x) \to \min \quad (3)$$

Минимум будем получать из уравнения: $\varphi'(x, x_k, \alpha_k) = (x - x_k) + 2\beta_k A^*(Ax - b) = 0$, т.е.

$$\left(2\beta_p A^*A + E\right)x = 2\beta_p A^*b + x_0.$$

Для данного метода характерно то, что коэффициенты могут быть постоянными – не изменятся от итерации к итерации.

В результате подбора коэффициента после **двух** итераций было получено решение, (в следующих итерациях особых отличий не наблюдается). Результат расчета показан на рис. 1.

Регуляризованный проксимальный метод, подобно проксимальному методу, состоит в минимизации (3), но только функция $f(x)$ имеет вид (2) [3]. Уравнение для нахождения минимума следующее

$$\left(2\beta_p A^* A + 2\beta_p \alpha_p E + E\right)x = 2\beta_p A^* b + x_0,$$

где в качестве x_0 на нулевом шаге используется вектор граничных условий, а на каждой последующей итерации – регуляризованное приближение *x*, полученное на предыдущей.

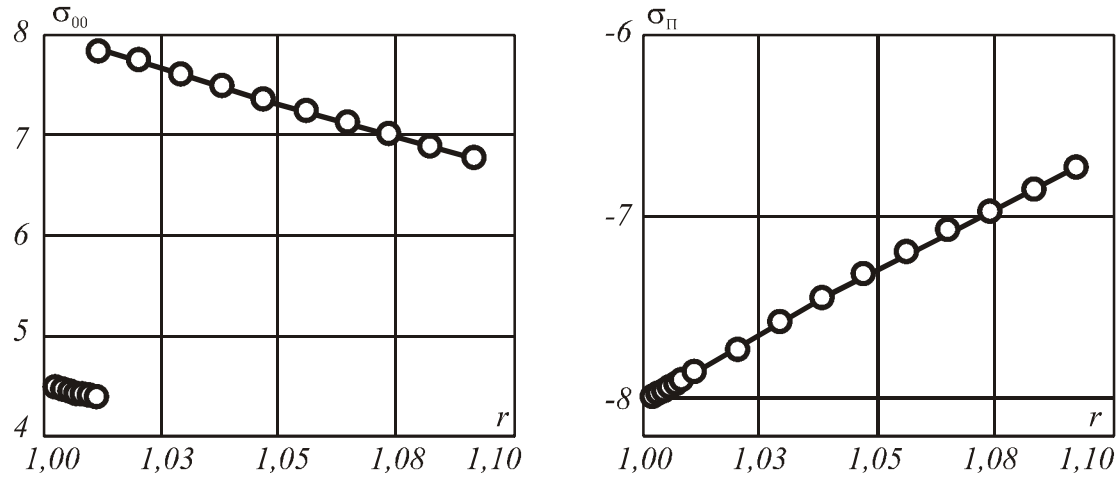


Рис. 1. Сравнение численных результатов с аналитическими. (Сплошная линия – аналитические результаты)

Для данного метода характерно то, что при использовании нормировки невязка имеет большее значение, чем без нормировки. Таким образом, по наблюдению изменения невязки регуляризованный проксимальный метод лучше использовать без нормировки. Хотя решение все равно достигается медленно.

Следовательно, из рассмотренных методов наиболее эффективным является проксимальный.

3. Алгоритм и результаты расчета

Математической основой, позволившей распараллелить вычисления, является принятый в алгоритме способ минимизации функционала. Во-первых, постоянная часть, а именно произведения A^*A и A^*b, могут быть вычислены единовременно. При этом, операция умножения транспонированной матрицы на саму себя может быть легко распараллелена между процессорами (рабочими станциями) по тому же математическому алгоритму, что и на суперкомпьютерах с разделяемой памятью. Данная операция составляет первый этап работы программы.

Оставшаяся часть алгоритма регуляризации, состоящая в последовательном решении линейных систем с разными параметрами, однако, может быть проведена параллельно. Это возможно благодаря использованию для решения таких систем метода квадратного корня – неитера-

ционного метода, прямой ход которого состоит в преобразовании матрицы коэффициентов уравнений, а обратный в последовательном получении вектора-решения. Так как не требуется знать результатов вычислений предыдущей итерации во время прямого хода, то эти вычисления для разных параметров могут проходить параллельно. По окончании же расчета одним из компьютеров *p*-ой итерации результат должен быть передан компьютеру, рассчитывающему *p*+1-ю итерацию и т.д.. Выигрыш по времени следует из того, что время, тратящееся на прямой ход, намного больше времени проведения обратного. Данная операция составляет второй этап работы комплекса.

Были проведены расчеты режущих инструментов с моно и многослойными покрытиями. Решение данной задачи было проведено с использованием программы, вычисляющей на кластере рабочих станций.

Та же задача расчета напряженно-деформированного состояния износостойких покрытий была проведена методом конечных элементов в MSC.Nastran с использованием MSC.Patran.

Результаты, произведенные с помощью разных комплексов программ, полностью совпадают, причем расчет на MSC.Nastran проходил быстрее, но подготовка конечно-элементной сетки с использованием препроцессора MSC.Patran потребовала подбора геометрии задачи для каждой толщины покрытия, что сильно снижает эффективность данных комплексов.

В таблице 1 приведены времена распределённого расчёта решения задачи, а также коэффициенты ускорения $S_m = \frac{T_1}{T_m}$ и эффективности $E_m = \frac{S_m}{m}$ (где T_m – время параллельного алгоритма на кластере из m рабочих станций, T_1 – время выполнения последовательного алгоритма на одной машине; T_m представляет собой совокупность чистого времени счёта и накладных расходов на подготовку и пересылку дынных).

Таблица1. Сравнение коэффициентов ускорения и эффективности параллельного расчёта задачи о НС режущего инструмента с монопокрытием 6мкм

m	T_m, мин	S_m	E_m
1	836	-	-
2	422	1.98	0.99
6	140	5.97	0.99
12	71	11.77	0.98

Как видно из таблицы ускорение S_m растёт практически линейно. Учитывая большую автономность выполняемых каждой машиной задач, малые накладные расходы на поддержку работы параллельных алгорит-

мов и обмены результатами, можно прогнозировать стабильное увеличение производительности при дальнейшем росте числа компьютеров в кластере вплоть до числа, равного количеству итераций внутреннего цикла метода регуляризации.

Литература

1. *Олейников А.И. и др.* Расчет напряжений в породных массивах методом граничных интегральных уравнений // Кривой рог: НИГРИ, 1982. – 24с.

2. *Тихонов А.Н., Арсенин В.Я.* Методы регуляризации некорректных задач // Главная редакция физико-математической литературы изд-ва «Наука», М., 1974.

3. *Васильев Ф.П., Обрадович О.* Регуляризованный проксимальный метод для задач минимизации с неточными исходными данными // ЖВМиМФ, 1993. Т. 33, №2. С. 182-188.

ПАРАЛЛЕЛЬНЫЕ ВЫЧИСЛЕНИЯ ДЛЯ ИДЕНТИФИКАЦИИ ПАРАМЕТРОВ В МОДЕЛЯХ ЭКОНОМИКИ*

Н.Н. Оленев*

Вычислительный центр им. А.А. Дородницына РАН, г. Москва

Введение

Огромный возможный набор сочетаний значений параметров не позволял до появления кластерных суперкомпьютеров точно определять эти значения. Использование высокопроизводительных параллельных вычислений на суперкомпьютере благодаря достаточно полному перебору позволяет точно решить задачу идентификации параметров сложных математических моделей экономических систем.

Дело в том, что в моделях экономики имеется немало параметров, которые не удается найти напрямую из данных экономической статистики. В случае же, когда данных статистики хватает, качество исходных статистических данных, как правило, таково, что их хватает только для определения интервалов, в которые попадают параметры модели. Кроме того, и начальные значения некоторых переменных модели часто

*Работа выполнена при финансовой поддержке Российского фонда фундаментальных исследований (коды проектов 04-07-90346, 04-01-00606), по программе государственной поддержки ведущих научных школ (код проекта НШ-1843.2003.01), при поддержке программы № 3 фундаментальных исследований ОМН РАН «Вычислительные и информационные проблемы решения больших задач» и при поддержке программы фундаментальных исследований РАН № 16 «Математическое моделирование и интеллектуальные системы».

оказываются неизвестными и поэтому должны рассматриваться как такого рода параметры.

Неизвестные параметры экономической модели определяют косвенным образом, сравнивая выходные временные ряды переменных модели с доступными статистическими временными рядами. В качестве критериев близости расчетного X_t и статистического Y_t временных рядов удобно (см. [1, 2]) использовать коэффициент корреляции Пирсона $P \in [-1, 1]$ и индекс несовпадения Тэйла $U \in [0,1]$ [3]. Коэффициент корреляции является мерой силы и направленности линейной связи между сравниваемыми временными рядами и, чем он ближе к +1, тем более схоже поведение этих рядов. При этом следует учитывать, что инфляционная составляющая может преувеличивать линейную связь рядов, поэтому при использовании коэффициента корреляции нужно сравнивать показатели в реальных величинах. Индекс Тэйла U измеряет несовпадение временных рядов X_t и Y_t и чем ближе он к нулю, тем ближе сравниваемые ряды:

$$U=[\Sigma(X_t - Y_t)^2]^{1/2} / [(\Sigma X_t^2)^{1/2}+(\Sigma Y_t^2)^{1/2}].$$

Поскольку параметров довольно много, вначале следует провести естественное распараллеливание процессов, описываемых моделью: разбить модель на отдельные блоки, идентификацию параметров в которых можно производить независимо. Это дает возможность за разумное время определить независимые параметры. При этом временные ряды переменных, определяемые в модели из других блоков и используемые в данном блоке, как внешние переменные, можно задавать либо на основе данных, полученных из уже откалиброванных блоков модели, либо на основе статистических данных.

После декомпозиции модели по блокам, благодаря параллельным вычислениям на суперкомпьютере становится реальной возможность полного перебора параметров модели на заданном интервале их изменения с последовательно уменьшающимся интервалом изменения параметров. Для однозначности выбора оптимального варианта можно использовать ту или иную свертку коэффициентов корреляции и индексов Тэйла, например, если искомые параметры имеют примерно равную важность, можно максимизировать отношение среднегеометрической величины корреляции к среднегеометрическому коэффициенту Тэйла. При этом следует перебирать только те варианты значений параметров, коэффициент корреляции для которых выше некоторой положительной величины, например, 0.75.

В качестве примера в данной работе рассмотрена задача идентификации параметров производственной функции, используемой в новой

модели переходной экономики России, разрабатываемой в ВЦ РАН для оценки динамики теневого оборота в 1995-2003 гг.

Производственная функция

Обычно параметры производственной функции определяют по данным экономической статистики для временных рядов переменных, непосредственно входящих в производственную функцию. Такой подход может быть оправдан отсутствием модели, в которой используется данная производственная функция. Кроме того, найти параметры производственной функции, внутренне согласованные с поведением других переменных модели весьма сложно в силу чрезвычайно большого числа возможных вариантов. Однако высокопроизводительные вычисления на суперкомпьютере позволяют найти параметры производственной функции, согласованные с поведением других переменных математической модели экономики.

Однородная степени 1 производственная функция с постоянной эластичностью замещения (CES-функция) успешно применялась при исследовании экономики СССР (см., например, [4]). В качестве производственной функции экономики России 1995-2003 гг., описывающей в каждый момент времени t зависимость валового внутреннего продукта Y от количеств используемых производственных факторов: труда R и капитала C, – возьмем однородную степени γ CES – функцию.

$$Y(t)=C^{\gamma}(t)f(x),\ f(x)=\delta[\alpha+(1-\alpha)x^{\beta}]^{\gamma/\beta},\ x=R(t)/C(t), \quad (1)$$

где параметры производственной функции удовлетворяют неравенствам

$$0<\alpha<1,\ \beta>0,\ \gamma>1,\ \delta>0. \quad (2)$$

Суммарные производственные фонды (капитал) в постоянных ценах в соответствии со статистическими данными практически не меняются, а занятость даже слегка уменьшается. Чтобы описать рост ВВП с помощью выбранного типа производственной функции сделаем дополнительные предположения об органическом строении капитала. Будем считать, что капитал $C(t)$ состоит из двух частей: «старого капитала» A, который только падает

$$dA/dt=-\mu_A A(t), \quad (3)$$

и «нового капитала» B, который растет за счет введения в строй новых производственных фондов $J(t)$

$$dB/dt=J(t)-\mu_B B(t), \quad (4)$$

так что

$$C(t) = A(t) + B(t). \quad (5)$$

Параметры амортизации $\mu_A>0$, $\mu_B>0$ наряду с параметрами производственной функции предстоит определить косвенным образом.

Статистические данные по ВВП испытывают заметные сезонные колебания от квартала к кварталу. Однако ни капитал, ни реальная зарплата таких колебаний не испытывают. Что касается труда, то если его выражать, как это обычно принято, в миллионах человек занятых, то он также не испытывает сезонных колебаний. Однако если труд выразить в количестве отработанного в каждом квартале времени (в миллиардах человеко-часов), то он испытывает колебания, четко повторяющим колебания ВВП.

Экономический агент «Производитель»

Ответственный за производственный блок экономический агент, которого логично назвать производителем, получает доходы производства и торговли, делает инвестиции и нанимает трудящихся, может брать срочные кредиты у коммерческих банков, выплачивает дивиденды собственникам производственных фирм в соответствии с заданной им политикой накопления капитала.

Считаем, что объем банковских ссуд L ограничен основным капиталом в соответствии с его органическим строением:

$$L(t)=(\alpha A A+\alpha B B)p(t), \qquad (6)$$

где индекс цен на основные фонды $p(t)$ считается заданным в данном блоке, а параметры $\alpha_A > 0$, $\alpha_B > 0$ следует определить.

Изменение банковских ссуд и расчетного счета определяются из баланса доходов и расходов. Для простоты налоговые отчисления в консолидированный бюджет задаются с помощью двух видов налогов: налога на добавленную стоимость n (НДС), включающего налог на прибыль, и единого социального налога m (ЕСН), включающего подоходный налог.

Считается, что производитель функционирует с целью максимизации его капитализации. Это совпадает с максимизацией дисконтированных доходов, причем процент дисконтирования совпадает с соответствующей двойственной оценкой роста активов, внутренне присущей рассматриваемой задаче оптимального управления.

Полученные в результате решения задачи оптимального управления формулы, определяют загрузку производственных фондов трудом, процент дисконтирования, ставку заработной платы, ВВП, дивиденды, объем банковских ссуд, остаток расчетного счета.

Параллельные вычисления

Параллельные вычисления основаны на распараллеливании циклов, осуществляющих полный перебор значений искомых параметров в заданных интервалах их изменения с последовательно уменьшающимся шагом равномерной разбивки интервалов.

В первой серии параллельных вычислений был осуществлен перебор указанных выше 8 параметров модели и, кроме того, начального значения нового капитала $B(0)$. Оптимальные значения параметров определялись косвенным образом, сравнивая по введенной свертке критериев Тэйла и коэффициентов корреляции, рассчитанные по модели временные ряды и статистические временные ряды для следующих показателей: выпуска Y, капитала C и объема банковских ссуд L при заданном труде R, заданном индексе цен p, заданных капиталовложениях J. Параметры производственного блока: δ=130 тыс. руб 2000 г., α=0,91, β=0,68, γ=5,68, μ_A=0,0675, μ_B=0,0273, α_A=0,0005, α_B=0,188. Амортизация нового капитала оказалась ниже амортизации старого, а норма обеспечения банковских ссуд для нового капитала значительно выше соответствующей нормы для старого капитала.

При этом ставка заработной платы получилась примерно в три раза выше статистической, что на первый взгляд не противоречит реальностям нашей жизни. Однако при таком подходе дивиденды оказываются меньше нуля.

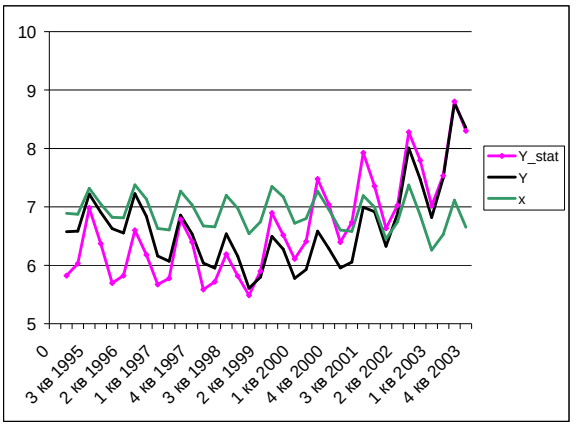


Рис. 1. ВВП расчетный и статистический

В следующей серии параллельных вычислений наряду с указанными выше показателями подгонялось к статистическим значениям и значения для ставки заработной платы s. Такой подход можно оправдать предположением, что статистические органы уже учли теневой доход в заработной плате. В результате, как и следовало ожидать, дивиденды оказались положительными. Параметры производственного блока: δ=16 тыс. руб 2000 г., α=0,93, β=0,08, γ=4,9, μ_A=0,064, μ_B=0,009, α_A=0,001, α_B=0,11. Полученные данные подтверждают необходимость

совокупного согласованного изменения параметров при изменении исходных предположений.

Однако сезонные колебания расчетных значений выпуска стали намного меньше статистических. Это означает, что используемый труд должен колебаться значительно больше, чем показывает статика.

Это можно учесть, если увеличить амплитуду колебаний выпуска предполагая, что фактический труд R (измеряемый в млрд. человеко-часов) имеет большую амплитуду колебаний, чем статистический труд R_s:

$$R=k(Rs–R0). \quad (6)$$

Возможная интерпретация (6):

R_0 – «балласт», труд, учитываемый в статистике, но не приносящий добавленной стоимости (оплачиваемый по статистической зарплате), $R_0<R_s$.

$k>1$ – коэффициент фактической занятости оставшихся. Другими словами, те занятые, что приносят добавленную стоимость, работают не 8 часов в день, а в k раз больше.

Кроме того, в дальнейших расчетах следует учесть, что фактический фонд и фактическая ставка заработной платы не ниже статистической, фактическая ставка единого социального налога не превышает статистическую ставку, а доход консолидированного бюджета от ЕСН совпадает со статистическим доходом. Таким образом, будет получена нижняя оценка теневой составляющей в заработной плате.

Литература

1. *Olenev N.* Production Function of Skilled and Unskilled Labour in a Model of a Non-Growing Russian Economy // International Labour Market Conference Proceedings. - Aberdeen: Robert Gordon University, 1999. P. 560-575.
Web: http://ideas.repec.org/p/wpa/wuwpla/0309005.html.

2. *Оленев Н.Н.* Параллельные вычисления при калибровке моделей экономики // Декомпозиционные методы в математическом моделировании и информатике. Тезисы докладов 2-й Московской конференции. Москва, Россия (21-24 июня 2004 г.). С. 81-82.

3. *Theil H.* Economic Forecasts and Policy. North-Holland Publishing Company, Amsterdam, 1961.

4. *Weitzman M.L.* Soviet Postwar Economic Growth and Capital-Labor Substitution // The American Economic Review, (Sep.). 1970. Vol. 60, No. 4. P. 676-692.

ОСОБЕННОСТИ АВТОМАТИЗИРОВАННОГО РАСПРЕДЕЛЕНИЯ ВЫЧИСЛИТЕЛЬНОГО ПРОЦЕССА ДЛЯ ИМИТАЦИОННОГО МОДЕЛИРОВАНИЯ СИСТЕМ

С.И. Олзоева

Восточно-Сибирский государственный технологический университет, г. Улан-Удэ

Введение

Имитационное моделирование сложных систем в настоящее время является одним из ключевых направлений развития интеллектуальных компьютерных технологий. В последнее время доминирует тенденция внедрения методов имитационного моделирования в различные интерактивные системы оптимизации, экспертные системы. Активное использование систем поддержки принятия решений в задачах планирования, управления и проектирования в реальных производственных структурах ведут к интеграции этих систем с проблемно-ориентированными имитационными моделями (ИМ) на основе объектной ориентированности программных компонентов. Имитационное моделирование позволяет осуществлять анализ динамических процессов в условиях неопределенности действия стохастических факторов различной природы, исследовать большое количество алгоритмов, сценариев развития и является основой для поиска и формирования варианта решения.

Однако, несмотря на большие функциональные возможности у всех ИМ есть одна общая проблема – это проблема производительности. Это обусловлено: во-первых, огромными объемами данных, которые необходимо обработать; во-вторых, получение статистически обоснованных оценок исследуемых характеристик требует больших вычислительных затрат. Стремление ускорить вычисления, либо получить большую точность, либо усложнить модель изучаемого явления требуют более мощную вычислительную технику. Таковыми являются сегодня параллельные вычислительные системы – кластерные многопроцессорные системы. Программы ИМ, предназначенные для выполнения на однопроцессорной ЭВМ, не переносятся эффективно на параллельные системы. Требуется переделка, сопровождаемая дополнительным анализом задачи. Приобретает наибольшую важность и значимость автоматизированное конструирование структуры программного обеспечения ИМ для распределенного моделирования. Это требует расширения и разработки программно-технического инструментария имитационных систем.

Существующие сегодня методы автоматического синтеза имитационных моделей и соответствующие им алгоритмы не позволяют программно и в полном соответствии требованиям разработки реализовать

всю совокупность свойств распределенных имитационных систем в рамках допустимых временных ограничений при имитационном моделировании. Поэтому их недостатки должны компенсироваться программной поддержкой функций распараллеливания алгоритмов имитационного моделирования, измерения параметров качества получаемого проекта распределенного имитационного моделирования (РИМ).

К рассмотрению предлагаются подходы к реализации автоматизированного инструментария для оптимальной по критерию времени организации распределенного вычислительного процесса ИМ.

Функциональное содержание инструментария для распределения вычислительного процесса ИМ

Функциональное содержание такого инструментария определяется тем, что значительные резервы повышения эффективности имитационных систем скрыты в способах организации вычислительного процесса на многопроцессорной платформе, соответствующем построении вычислительных схем моделирующих алгоритмов, управляющей программы имитационной модели, а также в способах организации и представления базы данных модели.

Сложившиеся подходы к имитационному моделированию сложных систем предполагают проведение анализа динамических свойств систем на основе экспериментов с их дискретно-событийными имитационными моделями. При всем многообразии и различии предметных областей имитационного моделирования, все ИМ дискретно-событийных систем состоят из следующих составляющих:

– модулей $S_1, \ldots S_n$, воспроизводящих поведение различных элементов и подсистем моделируемого объекта;

– управляющей программы (УП), реализующей логику поведения модели, т.е. квазипараллельное и коррелированное течение модельных процессов;

– модулей, обслуживающих процесс моделирования (ввод – вывод, планирование машинного эксперимента, статистическая обработка результатов моделирования).

Таким образом, современная концепция построения имитационных моделей сложных систем естественным образом вписывается в модель параллельных вычислений MPMD (Multiple Program – Multiple Data). Для реализации на параллельных вычислительных платформах имитационного моделирования сложных систем наиболее подходит модель вычислений MPMD по следующим причинам:

- *Внутренний параллелизм*. Сложные системы состоят, как правило, из параллельно функционирующих компонент на разных уровнях иерархической структуры системы. Поэтому имитационная модель

сложной системы содержит множество модулей, соответствующих различным элементам и подсистемам моделируемого объекта.

- *Разнородность подсистем и элементов*, составляющих сложную систему, порождает и разнородность математических схем, описывающих функционирование различных элементов. Отдельные элементы и подсистемы могут быть описаны, например, дифференциальными уравнениями, системами массового обслуживания, конечными автоматами и т.д. Отсюда следует разнородность программной реализации модулей ИМ.

На основании изложенного, назначение программного инструментария для организации распределенного имитационного моделирования состоит в отделении аспектов, связанных с разработкой программного обеспечения ИМ, от вопросов организации распределенных (параллельных) вычислений. Что и определяет технологические этапы функционирования программного инструментария:

1. анализ структуры программного обеспечения (ПО) имитационной модели, выбор алгоритма распараллеливания (декомпозиции) ИМ;
2. декомпозиция ИМ;
3. оценка качества вариантов проектов распределенной имитационной модели, определение возможных временных факторов;
4. выбор окончательного проекта РИМ.

Алгоритмы технологических этапов

Алгоритмическим наполнением технологических этапов являются методы декомпозиции имитационной модели систем, а также методы оценки возможных временных факторов РИМ.

Для программ имитации характерно то, что они представляют собой сильно-связный код. Это обусловлено традиционной организацией управляющей программы имитации и централизацией базы данных моделирования**,** с которой работают и управляющая программа, и программы имитации процессов, т.е. взаимодействие моделируемых процессов происходит через обращение к общим наборам данных. Поэтому простое распределение модулей моделирующей программы по процессорам не будет способствовать эффективному ускорению процесса имитации из-за многочисленных передач информации между компонентами имитационной модели. Следовательно, задача декомпозиции ПО ИМ сложных систем на блоки, по критерию минимизации обмена данными между блоками ИМ, состоит именно в оптимальном распараллеливании алгоритмов управления имитационной моделью. Что автоматически определит вариант декомпозиции ПО ИМ и базы данных модели.

Задача декомпозиции управляющей программы требует такого формального описания логики управления имитационной моделью, ко-

торое можно подвергать математическому анализу. Чтобы математическое представление логики управления можно было сравнивать между собой, подвергать декомпозиции и перестраивать разными способами. Этот формализм должен отвечать на вопрос об эквивалентности моделей управления: централизованной и распределенной.

С этой целью рассмотрим математические модели представления дискретно-событийных систем (ДСС) каковыми являются ИМ. Логическое поведение ДСС может моделироваться с помощью конечных автоматов и сетей Петри, которые подчеркивают лишь последовательность состояний дискретных систем и полностью опускают описание времени пребывания в каждом из состояний. С их помощью можно исследовать вопросы лишь качественного или логического характера. Временные же и стохастические дискретно-событийные модели обеспечивают более полное описание поведения систем, поэтому больше приспособлены для ответа на вопросы, связанные с характеристиками функционирования. Кроме того, полнота описания достигается за счет уменьшения общности. На практике менее сложный формализм, как более ограниченный, может оказаться полезнее.

В нашем случае преследуется цель такой декомпозиции ИМ, чтобы сохранялась глобальная логика поведения модели. Поэтому предлагается управляющую программу описать математической моделью расширенного конечного автомата. Работу УП можно представить в виде многополюсника, который коммутирует сигналы от группы входов $\boldsymbol{w}$ к группе выходов $\boldsymbol{z}$ под управлением сигналов $\boldsymbol{x}$. Работа такого автомата описывается специальными рекурсивными функциями. Они определяют переход системы из предыдущего состояния в последующее и выходной сигнал. Математически это выражается с помощью следующей системы уравнений:

$$S(t+1)=f(S(t),X(t)),$$

$$Z(t+1)=f(S(t),X(t)),$$

где $X(t)=[x_1(t),\dots x_p(t)]$ – значения сигналов на входе автомата, $S(t)=[S_1(t),\dots S_n(t)]$ – внутренние состояния, $Z(t+1)= [z_1(t),\dots,z_k(t)]$ – значения сигналов на выходе автомата. При получении внешнего сигнала считывается состояние системы. После этого паре «входной сигнал – состояние» сопоставляются действия перехода и корректируется состояние. При этом внутренним состояниям $S_i(t)$ соответствуют модули ИМ, воспроизводящие поведение подсистем моделируемого объекта.

При таком описании логики поведения задача декомпозиции УП сводится к задаче разложения автоматного описания управляющей программы на ряд функционально несвязных друг с другом автоматов меньшей размерности, обеспечивающих реализацию необходимых воздействий на объект управления, т.е. ИМ. Задача декомпозиции автомата сво-

дится к разложению графа переходов исходного автомата в частичное декартово произведение более простых по числу вершин графов переходов функционально несвязных друг с другом подавтоматов. Эту задачу предлагается решить методом многокомпонентной раскраски графов с использованием характеризационного метода [1], удобного для реализации на ЭВМ и позволяющего существенного сократить перебор вариантов.

На основании проведенных преобразований формируется децентрализованная УП имитационной модели дискретно-событийных систем, обеспечивающая синхронизацию работы модельных блоков, выполняющихся на разных процессорах и работающая с подчиненными диспетчерами модельных блоков. В соответствии с полученной структурой УП автоматически формируется распределенная база данных моделирования.

Для оценки качества полученного варианта декомпозиции ИМ предложен способ оценки потенциального параллелизма для заданного варианта декомпозиции ИМ, позволяющий определить возможный временной фактор. Предлагаемый способ основан на применении математического аппарата теории случайных импульсных потоков и позволяет учесть характеристики неоднородности вычислительной платформы, на которой предполагается реализация РИМ. Определение возможного временного фактора до начала процесса моделирования позволит эффективно организовать вычислительный процесс. Описание способа изложено в [2].

Заключение

Таким образом, разработка методов и программных средств распределенного имитационного моделирования, сочетающих сложившиеся приемы имитационного моделирования систем и методов обработки информации на многопроцессорных системах, создает благоприятную основу для решения задач ускорения имитационного моделирования систем большой сложности и размерности и придает современным имитационным системам новые системные свойства. Последние заключаются в возможности получения статистически обоснованных результатов моделирования с требуемой точностью и детальностью исследования, без существенных ограничений на размерность модели и моделируемого объекта и, как следствие, возможность широкого использования в автоматизированных системах реального времени.

Литература

1. *Горбатов В.А.* Теория частично-упорядоченных систем // М.: Советское радио, 1976, - 336 с.

2. *Kutuzov O. I., Olzoeva S. I.* Some questions of the simulation of complex systems onto message-passing parallel architectures // In: Proceedings of International Conference on Soft Computing and Measurements, 2003, St. Petersburg, Russia, 2003. Vol. 1. P. 187-120.

ПРОГРАММЫЕ СРЕДСТВА ДЛЯ УПРАВЛЕНИЯ ПАРАЛЛЕЛЬНЫМ ВЫПОЛНЕНИЕМ ГРАФ-СХЕМНЫХ ПОТОКОВЫХ ПРОГРАММ НА КЛАСТЕРНЫХ ВЫЧИСЛИТЕЛЬНЫХ СИСТЕМАХ

М.А. Осипов

Московский энергетический институт (ТУ), г. Москва

Введение

В докладе описаны программные средства, реализованные в процессе выполнения проекта[2], нацеленного на создание системы граф-схемного потокового программирования для кластерных вычислительных систем (ВС) (руководитель проекта профессор Кутепов В.П.) [1]

Следующие три основные проблемы были поставлены и решались в рамках этого проекта:

- создание визуального граф-схемного языка крупно-блочного потокового параллельного программирования,
- разработка инструментальной среды, предназначенной для программирования параллельных программ на языке граф-схем,
- создание операционных средств, осуществляющих управление параллельным выполнением граф-схемных потоковых программ (ГСПП) на различных конфигурациях кластерных ВС.

Созданный язык и разработанная для него инструментальная среда программирования описаны в [2].

Ниже мы остановимся на основных решениях, которые были положены в основу создания программных средств, предназначенных для эффективного управления процессом выполнения граф-схемных параллельных программ (ГСПП) на кластерных ВС.

1. Операционная семантика параллельного выполнения ГСПП

ГСПП представляется в виде пары <ГС,I>, где ГС – граф-схема, I – интерпретация. Граф-схема или просто схема позволяет визуально представлять строящуюся из модулей программу решения задачи; интерпретация сопоставляет каждому модулю множество подпрограмм, а связям между модулями – типы данных, передаваемых между подпрограммами модулей в процессе выполнения ГСПП.

Основным «строительным» блоком ГСПП является модуль, графическое представление которого приведено на рис. 1. Все входы и выходы модулей строго типизированы и разделены на группы (они называются конъюнктивными группами входов (КГВх) и конъюнктивными

[2] Работа выполнена при поддержке РФФИ, проект 03-01-00588

группами выходов (КГВых), соответственно) и отражают структуру потоков данных, передаваемых между подпрограммами модулей. Каждой КГВх модуля однозначно сопоставляется подпрограмма на одном из последовательных языков программирования (C/C++, Pascal, Java и др.). Кроме того, в качестве формального параметра в списке параметров подпрограмме добавляется параметр *tag*, который указывается первым. На ГС этот параметр не изображается, поскольку он идентифицирует данные, передаваемых между модулями ГСПП в процессе его выполнения (рис. 1).

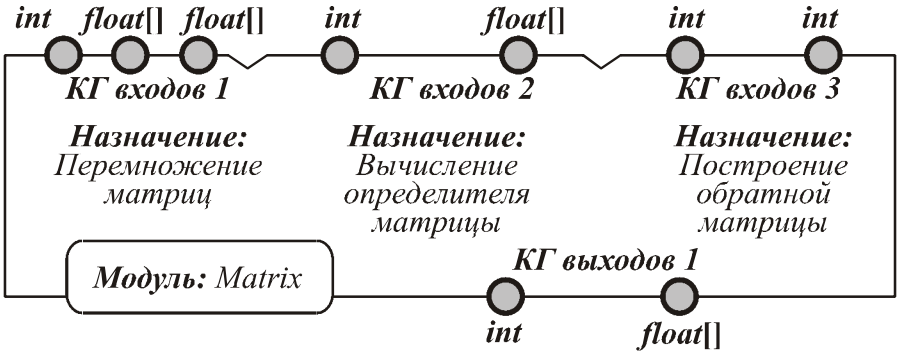


Рис. 1. Графическое изображение модуля

ГС представляет блочную структуру, построенную из модулей путем соединения выходов КГВых модуля с входами КГВх другого или этого же модуля, причем типы соединяемых входов и выходов должны совпадать.

В модели параллельного выполнения ГСПП реализуются три вида параллелизма:

- пространственный, являющийся следствием информационной независимости модулей ГСПП и, как следствие, процессов, сопоставляемых их КГВх,
- потоковый, параллелизм множества данных (SIMD – один поток команд, множество потоков данных), т.е. тот случай параллелизма, когда одна и та же программа (или подпрограмма модуля в случае языка ГСПП) одновременно выполняется для разных данных на ее входе.

Для того чтобы различать последние два вида параллелизма при выполнении ГСПП, используется механизм тегирования. В операционной семантике языка ГСПП это выражается в том, что при наличии на всех входах конъюнктивной группы входов модуля данных, помеченных соответствующими тегами, параллельно запускаются на выполнение несколько процессов, каждый из которых однозначно идентифицируется тегом и сопоставленными ему данными.

Модуль ГСПП считается готовым для выполнения по любой из своих КГВх, если на всех входах этой КГВх (в соответствующих входам буферах) есть данные, помеченные одним и тем же тегом.

Различные теги идентифицируют различные данные, к которым одновременно применяется ГСПП, точнее, подпрограммы, отнесенные к соответствующим КГВх ее модулей (таким образом, реализуется параллелизм множества данных).

При выполнении процесса в его подпрограмме могут использоваться специальные системные команды (реализуемые посредством обычного обращения к специальным функциям) **WRITE** (запись), **READ** (чтение), **OUT** (выход) – позволяющие осуществлять межмодульное взаимодействие, т.е. строить различные схемы взаимодействия по данным между подпрограммами различных модулей путем чтения данных из буферов или записи данных в буферы, сопоставляемые входам КГВх модулей.

Команда **WRITE** имеет формат: WRITE(<номер КГВых>,<тег>,<список выходов><список переменных>), где список выходов есть перечисление номеров выходов КГВых, на которые передаются значения.

Команда **OUT** имеет аналогичный формат аргументов, однако после ее выполнения процесс, в котором она возникла, считается завершенным и его контекст может быть уничтожен.

Команда **READ** имеет формат: READ (<номер КГВх>, <тег>, <список входов>, <список переменных>). При ее выполнении также сохраняется контекст процесса, а после ее завершения процесс продолжает свое выполнение в старом контексте. Команда READ позволяет процессу читать данные с указанным тегом из буферов, сопоставленных КГВх, по которой процесс был инициирован.

Для более «тонкой» работы с поступающими на КГВх модулей данными, в частности работы с сопоставляемыми им буферами, предусмотрена команда **CHECK**(<номер КГВх>.<тег>,<список входов>, <переменная>), которая проверяет наличие данных с указанными тегами на указанных входах КГВх.

Для того, чтобы уничтожить процессы (удалить их контексты), которые оказываются ненужными, в подпрограммах модулей может использоваться команда **KILL**(<список имен уничтожаемых процессов>).

Отметим еще ряд существенных элементов операционной семантики языка ГСПП, которые также важны при его реализации на параллельных системах:

- порядок выполнения множества процессов, существующих на каждом шаге выполнения ГСПП, не существенен,
- определение готовых для выполнения процессов следует дисциплине FIFO: проверка на наличие в буферной памяти КГВх данных с одним и тем же тегом на всех ее входах осуществляется в порядке занесения этих данных в буферную память.

Ÿ поскольку с одним и тем же буфером данных могут одновременно работать несколько процессов (чтения или записи), поэтому в реализации нужно обеспечить взаимное исключение одновременных подобных действий.

2. Реализация языка граф-схем параллельных алгоритмов на кластерных ВС

Локальная сеть (в том числе сеть персональных компьютеров) является основным «строительным блоком» или узлом вычислительной системы или кластера. Узел кластера – хорошо сбалансированная по отношению, к быстродействию компьютеров, их количеству и пропускной способности коммуникации вычислительная система, так что на ней можно эффектно осуществлять реализацию параллельных вычислений в достаточно широком диапазоне вариаций степени распараллеливания программ от «крупнозернистого» до «мелкозернистого» параллелизма.

Для параллельного выполнения ГСПП на ВС разработана система программных средств – система управления. Эта система обеспечивает корректность параллельного выполнения программ на кластерной ВС, возможность отладки этих программ, хранение граф-схем и модулей, надёжную передачу данных между процессами модулей и пр.

Помимо перечисленных выше основных задач, связанных с выполнением граф-схем, система должна выполнять и другие функции. Следующий список содержит основные функциональные требования к реализации системы управления параллельным выполнением граф-схемных программ на кластерах:

Ÿ инициализация выполнения граф-схем,
Ÿ определение готовности процессов для выполнения,
Ÿ назначение процессов на выполнение, контроль их состояния,
Ÿ реализация команд языка: Read, Write, Out, Kill и Check,
Ÿ контроль состояния компьютеров кластера,
Ÿ планирование процессов с целью оптимизации использования ресурсов и уменьшения времени выполнения граф-схемы,
Ÿ реализация интерфейса взаимодействия компьютеров кластера,
Ÿ параллельное выполнение нескольких процессов,
Ÿ общая (для всех узлов ВС) система хранения данных,
Ÿ возможность мониторинга процессов системы,
Ÿ возможность выполнения подпрограмм модулей в различных ОС.
Ÿ Система управления также должна обеспечить:
Ÿ устойчивость к изменениям конфигурации кластера,
Ÿ надёжный протокол обмена данными между узлами кластера,
Ÿ высокую производительность.

Система управления параллельным выполнением ГСПП представляет собой набор программных средств, устанавливаемых на каждом компьютере или узле кластера. Состав и организация устанавливаемых программных средств приведена на рис. 2:

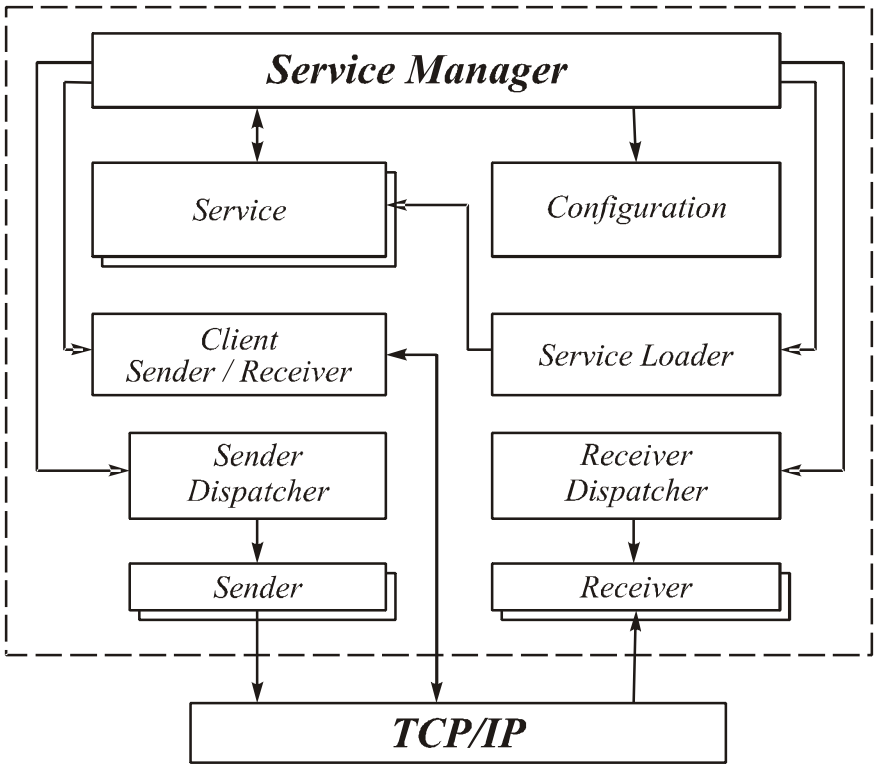


Рис. 2. Архитектура программных средств управления процессом выполнения ГСПП

Центральным элементом данной архитектуры является сервис. Сервис – это набор функций системы, выполняющих одну общую задачу. Сервис выполнения содержит все функции, необходимые для выполнения ГСПП на кластере. Именно сервисы реализуют функции, непосредственно связанные с процессом выполнения ГСПП, описанном в предыдущем разделе. Все остальные элементы архитектуры выполняют системные функции:

- инициализация сервисов,
- управление конфигурацией кластера,
- передача данных между узлами кластера,
- взаимодействие с внешними системами,
- управление сервисами.

Передача данных между узлами кластера осуществляется компонентом Sender, а получение – компонентом Reciever. Для каждого внешнего (по отношению к данному) узла сети в локальной версии системы существуют свои Sender и Reciever. Управление этими компонентами осуществляют соответственно Sender Dispatcher и Reciever Dis-

patcher. Компонент Client Sender/Reciever осуществляет обмен данными с внешними системами. При запуске системы создаётся компонент Service Manager, который при помощи Service Loader'а загружает сервисы и инициализирует их.

В сущности, предустановленный на компьютере набор программных компонентов является контейнером сервисов. Контейнер организует вызов необходимых сервисов, предоставляет сервисам функции высокого уровня, создаёт окружение. На различных узлах кластера в принципе могут функционировать различные наборы сервисов. Набор сервисов узла кластера определяет роль данного узла в системе. Назовём некоторые сервисы из системы управления:

- сервис конфигурации контролирует конфигурацию узла кластера и предоставляет её остальным узлам,
- сервис реестра предоставляет другим узлам интерфейс реестра системы. Реестр – это база данных системы, которая имеет иерархическую структуру. Реестр содержит информацию, необходимую для работы системы: программы граф-схем, конфигурацию кластера, информацию о пользователях и т.д. Также реестр может использоваться граф-схемами для хранения данных в ходе выполнения,
- сервисы управления выполнением – это набор сервисов, которые выполняют функции по запуску, контролю выполнения граф-схем, управлению буферами и планированию.

Сервис выполнения – это основной сервис системы, отвечающий за выполнение ГСПП, соответственно он должен работать на всех узлах кластера, которые участвуют в параллельных вычислениях. Во время инициализации данный сервис получает модули и граф-схемы, инсталлированные на данном узле. Инсталляция производится при помощи специального программного средства, входящего в состав системы, и заключается в копировании файлов модулей, а также описаний модулей и граф-схем и регистрацию данных сущностей в системе.

Описание модуля содержит идентификатор адаптера для данного модуля. Адаптер – это подключаемый компонент системы, осуществляющий запуск и остановку процесса модуля. Таким образом, один адаптер может использоваться для определённого класса модулей. Классификация определяется сценарием запуска. Например, для модулей, написанных на языке C++ в виде консольных приложений, будет применяться один адаптер, а для модулей, представляющих собой динамическую библиотеку DLL – другой. Также есть специальный регламент поведения модуля и спецификация интерфейса взаимодействия процесса модуля с системой. Таким образом, если программисту необходимо реализовать модуль на каком-нибудь языке программирования, он должен создать библиотеку для работы с системой на этом языке программирования и

придерживаться установленного регламента. Данная библиотека может далее использоваться для реализации любого модуля на этом языке программирования. Библиотека предоставляет программисту модулей функции высокого уровня, такие как запись данных на конъюнктивную группу выходов, завершение процесса граф-схемы и т.д. Подобная стратегия (рис. 3) позволяет программировать модули на разных языках программирования и даёт возможность выбора наиболее подходящего языка.

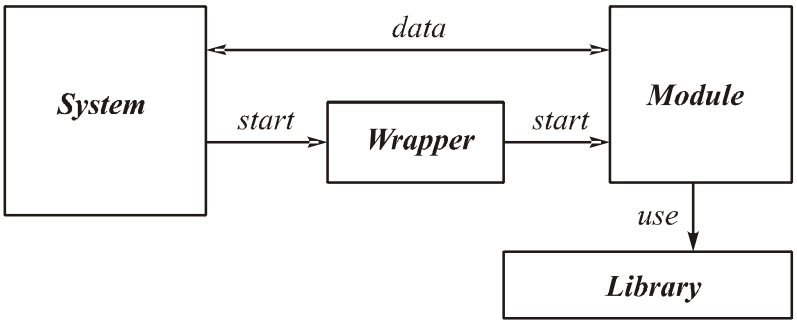


Рис. 3. Взаимодействие процесса модуля с системой

Взаимодействие между локальными компонентами системы управления осуществляется при помощи протокола TCP/IP посредством передачи сообщений определённого формата. Каждое сообщение имеет имя и тело сообщения. При получении сообщения компонент системы Service Manager по имени сообщения определяет сервис, которому предназначено это сообщение и передаёт его найденному сервису, который выполняет обработку данного события. Список сообщений, обрабатываемых сервисом, предоставляется системе самим сервисом. По такому же сценарию проходит взаимодействие системы с модулями и внешними системами. У каждого типа сообщений есть набор метаданных, который включает вид возможных отправителей для данного сообщения: локальный компонент системы, модуль или внешняя система.

Функции системы по управлению выполнением граф-схем распределены между несколькими сервисами. Набор этих сервисов должен присутствовать на каждом узле кластера, участвующем в вычислениях. Список данных сервисов следующий:

- сервис выполнения. Данный сервис отвечает за запуск граф-схем, управление процессами, индуцируемыми при выполнении. Этот сервис осуществляет также все взаимодействия между процессами модулей, получает запросы на выполнение функций WRITE, READ, CHECK, KILL, OUT, KILL,
- сервис управления буферами. Данный сервис реализует функции WRITE, READ, CHECK и OUT,
- сервис *планирования*. Осуществляет планирование выполнения граф-схемы, а именно определяет узлы кластера, на которых будут выполняться процессы модулей

Для того чтобы понять принципы взаимодействия данных сервисов, приведём несколько сценариев:

Запуск граф-схемы: Пользователь инициирует команду запуска граф-схемы, которая передаётся на выполнение сервису выполнения. Этот сервис создаёт контекст выполнения граф-схемы. Контекст содержит идентификатор контекста и имя граф-схемы. Контекст необходим для того, чтобы пользователь имел возможность выполнять несколько граф-схем в системе.

Запись данных на висящие входы граф-схемы: Пользователь инициирует команду записи данных на висящие входы граф-схемы, которая перенаправляется сервису выполнения. Сервис выполнения определяет контекст выполнения и запрашивает у сервиса планирования узел кластера или компьютер, на котором должен располагаться буфер для данного модуля. На указанном узле или компьютере создаётся буфер, и в него записываются данные. Данные действия выполняет сервис управления буферами на этом узле.

Запись данных в буфер: При записи данных сервис управления буферами проверяет наличие в буфере готовых кортежей данных. Если в буфере есть кортеж, данный сервис сообщает об этом сервису управления. После получения уведомления от сервиса управления буферами о наличии готового кортежа данных, сервис выполнения запрашивает у сервиса планирования компьютер, на котором должен выполняться процесс для этого набора данных. Далее сети сервис выполнения запускает процесс модуля по соответствующей конъюнктивной группе входов.

Заключение

В настоящий момент реализован прототип системы на языке программирования Java. Выбор языка Java обусловлен в первую очередь таким требованием к системе, как поддержка различных операционных систем. К плюсам данного языка программирования можно отнести поддержку параллельного программирования и синхронизации, а также простоту документирования системы. Минусом является то, что программы на языке Java работают медленнее, чем программы на таких языках программирования, как C/C++. В дальнейшем предполагается перепрограммирование критических компонентов системы или всей системы на более «быстрых» языках программирования для нескольких популярных операционных систем, в частности для платформ Win32 и Unix/Linux.

Литература

1. *Кутепов В.П., Котляров Д. В., Лазуткин В.А., Осипов М.А.* Граф-схемное потоковое параллельное программирование и его реализация на кластерных системах // Журнал «Программирование» (в печати).

2. *Кутепов В.П., Котляров Д.В., Лазуткин В.А.* Система граф-схемного потокового параллельного программирования: язык и инструментальная среда построения программ // (В материалах конференции).

ОПЫТ ПОСТРОЕНИЯ ВС ПО КЛАСТЕРНОЙ ТЕХНОЛОГИИ НА МЕХАНИКО-МАТЕМАТИЧЕСКОМ ФАКУЛЬТЕТЕ ПГУ

К.А. Осмехин

Пермский государственный университет, г. Пермь

Введение

Сегодня понятие кластерных архитектур становится всё более и более известным.

Под кластером обычно понимают множество взаимосвязанных вычислительных систем, которые, как правило, могут работать независимо, но также в нужное время способны объединятся для решения одной общей задачи.

Наиболее распространённым типом кластеров в университетской среде являются кластеры на основе технологии Fast Ethernet или Gigabit Ethernet. В таких кластерах узлами зачастую являются обычные персональные компьютеры или возможно машины с архитектурой SMP, но построенные на базе процессоров Intel. К таким кластерам относится и кластер созданный у нас в университете. Этот кластер включает в себя обычные персональные ЭВМ, на которых могут работать студенты, а также пять выделенных двухпроцессорных серверов с SMP архитектурой на базе процессоров Intel.

Развитие кластерных архитектур в настоящее время происходит, как и развитие любой технологии связанной с компьютерами, высокими темпами. В данной области нет большого объёма накопленных знаний, и потому решение каждой задачи связано с исследованием, опирающимся скорее на эмпирические данные, нежели на опыт предыдущих исследователей.

После появления в университете кластера возникла проблема исследования возможностей его использования и настройки для работы.

Изначально кластер нельзя было назвать таковым, так как кластер - это программно-аппаратный комплекс, и вот программной части как раз не было.

Поэтому первой проблемой был выбор и тестирование операционных систем для установки на кластере. Обязательным требованием выбора операционной системы была её бесплатность. Поэтому и по ряду других причин все исследования проводились среди операционных систем семейства Unix. При тестировании операционных систем необ-

ходимо было учесть их совместимость с аппаратным обеспечением, а это было не так просто вследствие, во-первых, его разнородной структуры, и, во-вторых, специфичностью аппаратных компонент.

В результате проведённого исследования в области применения кластеров были выделены следующие проблемы, которые могут быть решены с использованием кластерной архитектуры:

– создание систем высокой производительности;

– создание систем высокой доступности;

– более полное использование вычислительных ресурсов.

Ниже описан состав программного и аппаратного обеспечения, которое используется на кластере.

Аппаратное обеспечение

Кластер состоит из набора 12-ти рабочих станций, на базе процессоров Intel Celeron 1.2 МГц, которые расположены как обычные рабочие места. Также в состав кластера входит блок серверов, смонтированный в отдельную стойку и имеющий в своём составе: четыре 2-х процессорных сервера на базе процессоров Intel Pentium III 1.2МГц и один 2-х процессорный сервер на базе процессоров Intel XEON 2.0ГГц.

Программные компоненты

Встала задача создания на базе имеющегося аппаратного обеспечения системы способной решать вышеуказанные проблемы.

Основным и самым широко распространённым способом использования кластеров является их использование в научных областях для решения сложных задач. Этот подход имеет место, так как зачастую создание кластера из "обычных" машин является гораздо более дешёвым решением, нежели покупка специального суперкомпьютера. К тому же во многих организациях такая обычная техника простаивает и потому логично её использовать как часть кластера.

Между тем, для тех же задач в кластер могут быть объединены и более мощные компьютеры, специально приобретённые для этих целей. В таком случае предполагается, что помимо самих вычислителей в состав кластера должно входить более совершенное коммуникационное оборудование. Так как основным стремлением при создании кластера является желание создать некоторый параллельный компьютер, при работе на котором создавалось бы ощущение, что вы работаете с одной машиной, то увеличение производительности средств связи является совершенно необходимым, ибо позволит сократить временные задержки при взаимодействии отдельных узлов кластера.

В рамках создания на кластере высокопроизводительной вычислительной системы способной решать сложные расчётные задачи, были

установлены библиотеки параллельного программирования и средства управления параллельными задачами (PVM, MPI).

Как уже было сказано, использование кластеров для научных расчётов является наиболее широко используемым направлением работы в кластерной архитектуре, но естественно не единственным.

Огромное значение для современных прикладных задач имеет создание отказоустойчивых систем, и здесь на помощь снова приходят кластеры. Современные отечественные разработки в области создания кластеров в университетских центрах, в основном, направлены на создание систем высокой производительности. Исследованию же создания систем высокой доступности посвящено очень мало работ, поскольку в них могут быть заинтересованы крупные компании, использующие тяжеловесные системы управления предприятием, системы поддержки принятия решений и тому подобное. В этой области пока ещё не сформирована большая потребность.

Естественно, что, заменяя единичную систему многозвенной, мы увеличиваем её сложность и это может привести к недостаткам, но несомненны и преимущества, даваемые кластерными системами в системах ориентированных на поддержание корпоративных решений (распределённые базы данных компании, интранет порталы, системы поддержки принятия решений). Кластеры помогают решить множество проблем возникающих в таких системах. Среди них такие как: балансировка загрузки, создание распределённых хранилищ данных, создание систем работающих в режиме 7x24. Поэтому создание подобных систем сейчас так широко поддерживается ведущими производителями программного обеспечения программных систем, такими как Microsoft, IBM, SUN Microsystems, Oracle и др. Создание с помощью кластера системы высокой надёжности реализовано на основе продукта Oracle Real Application Clusters. Данный продукт позволяет создавать базы данных высокой доступности за счёт распределения загрузки по нескольким узлам кластера. Основным замечанием является то, что в списке совместимых операционных систем компании Oracle нет ни одной свободно доступной системы, поэтому, делая попытку использовать программный продукт Oracle, мы понимали, что гарантий успеха нет. Тем не менее, после испытаний на кластере девяти операционных систем удалось установить Oracle Real Application Clusters. Следует отметить, что это позволит использовать кластер не только для научных расчётов, но и для реализации прикладных проектов, в которых требуется использование высоконадёжной и производительной базы данных.

Следующим направлением, в котором проводятся сейчас исследования в области кластерных архитектур, является метакомпьютинг или «технология GRID». Данная технология позволяет использовать простаивающие вычислительные ресурсы, которых достаточно много в со-

временных организациях. Для этого в кластер объединяются все вычислительные ресурсы организации, предприятия, а зачастую узлами кластера становятся машины находящиеся на большом расстоянии друг от друга и соединённые глобальной сетью Internet.

Технология GRID применяется ещё менее интенсивно. Сама идея GRID родилась около пяти лет назад, но её полезность очевидна. Использование GRID на предприятиях и в организациях позволит увеличить полезность использования вычислительных ресурсов, а в глобальном масштабе, если GRID- сети будут объединены также как Internet, позволит получать доступ к распределённой информации, находящейся в информационном пространстве. Данную технологию можно сравнить с созданием электросети, когда конечный пользователь не знает, из какого источника получает электроэнергию, и точно также запрос к GRID должен автоматически порождать механизмы поиска нужного сервиса и конечный пользователь не будет знать, кто ему это сервис предоставил.

В рамках исследования технологии GRID на кластере была установлена система Sun One Grid Engine, которая позволяет распределять нагрузку в сети, планируя выполнение отдельных заданий на компьютерах кластера.

Заключение

Итак, в Пермском государственном университете на механико-математическом факультете была создана вычислительная система с кластерной архитектурой. В настоящее время её ресурсами пользуются студенты, в рамках учебных курсов. Следует отметить, что пока нет «серьёзных» задач, успешно реализованных средствами этой системы.

РЕШЕНИЕ ЗАДАЧИ МОДЕЛИРОВАНИЯ ДВИЖЕНИЯ КОСМИЧЕСКОГО АППАРАТА НА ПАРАЛЛЕЛЬНЫХ ВЫЧИСЛИТЕЛЬНЫХ СИСТЕМАХ

Н.Д. Пиза, Р.К. Кудерметов

Запорожский национальный технический университет, г. Запорожье, Украина

Введение

Для обеспечения качественной наземной отработки и испытаний высокоточных систем управления космических аппаратов (КА) в составе полунатурных моделирующих комплексов (ПНМК) постоянно возрастают требования к точности моделей, описывающих движение КА, процессов его взаимодействия с внешней средой, работу элементов системы управления. Это достигается применением все более сложных и

полных моделей, организацией моделирования в реальном времени, что, в свою очередь, приводит к необходимости увеличения вычислительной мощности компьютеров, которые используются в ПНМК. Одним из направлений повышения производительности и сокращения времени решения задач является использование параллельных вычислений.

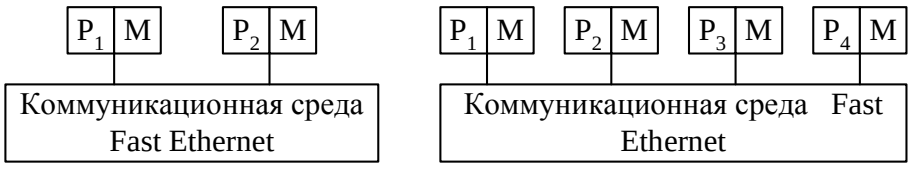
Авторами исследована возможность применения параллельных блочных методов интегрирования [1] и разработаны параллельные программно-алгоритмические средства для моделирования движения КА. В данной статье приведены результаты экспериментальных исследований ускорения и эффективности разработанных параллельных программно-алгоритмических средств, полученные для различных типов вычислительных систем.

Структуры параллельных вычислительных систем для исследования разработанных программно-алгоритмических средств.

Для экспериментальных исследований выбраны четыре параллельных вычислительных системы, которые представляют основные типы современных высокопроизводительных систем:

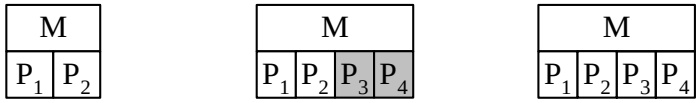
- компьютерная сеть Fast Ethernet;
- SCI-кластер (Петродворцовый телекоммуникационный центр Санкт-Петер-бургского государственного университета (СПбГУ));
- SMP-узлы (symmetric multiprocessors – симметричные мультипроцессоры, кластер Нижегородского государственного университета (ННГУ));
- суперкомпьютер CRAY T3E (Штутгартский вычислительный центр).

Экспериментальные исследования разработанных параллельных программных моделей проводились на конфигурациях параллельных вычислительных систем, представленных на рис. 1–4 (серым цветом обозначены неиспользуемые процессоры). В составе кластера ННГУ (рис. 2) исследовалась реализация параллельной модели движения КА на процессорах в режиме SMP. NN-two и NN-four – обозначения вариантов конфигураций, в которых используются двухпроцессорные и четырехпроцессорные узлы кластера, соответственно.



а) два процессора б) четыре процессора

Рис. 1. Сеть Fast Ethernet (Intel Pentium IV 1.76 ГГц)

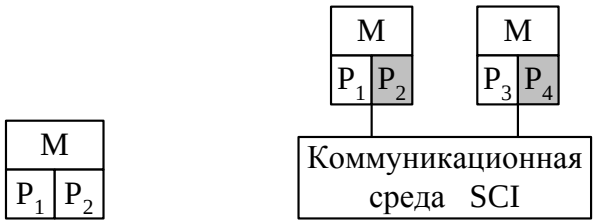


а) два процессора, вариант NN-two

б) два процессора, вариант NN-four

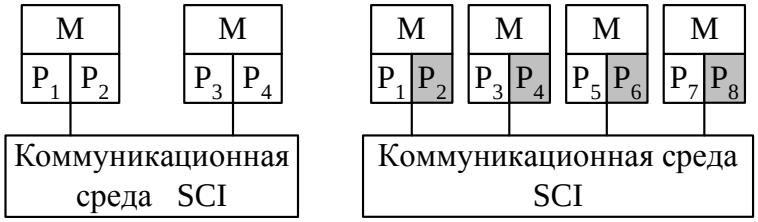
в) четыре процессора, вариант NN-four

Рис. 2. Конфигурации процессоров при экспериментах на кластере ННГУ (двухпроцессорный узел: Intel Pentium III Xeon 1000 МГц, четырехпроцессорный узел: Intel Pentium III Xeon 700 МГц)



а) два процессора, вариант double

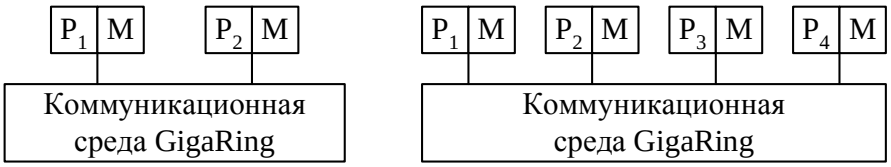
б) два процессора, вариант single



в) четыре процессора, вариант double

г) четыре процессора, вариант single

Рис. 3. Конфигурации процессоров при экспериментах на SCI-кластере (Intel Pentium III Copermine 933 МГц)



а) два процессора

б) четыре процессора

Рис. 4. Конфигурации процессоров в экспериментах на CRAY T3E

Для исследования на SCI-кластере (Scalable Coherent Interface – масштабируемый когерентный интерфейс) были проведены два варианта экспериментов, условно названные single и double (рис. 3). В варианте single на каждом узле кластера использовался только один процессор, т.е. для двухточечного метода интегрирования был задействован один канал SCI, а для четырехточечного – три канала SCI. В варианте double использовались два процессора в узлах кластера, т.е. двухточечный метод интегрирования выполнялся в режиме SMP на одном узле, а четырехточечный – с использованием двух узлов в режиме SMP и одного канала SCI между узлами.

Результаты исследования характеристик быстродействия параллельных средств моделирования движения КА

В качестве меры вычислительной сложности моделируемых систем (уравнений движения КА) принято время выполнения операции умножения вещественных чисел типа double. Сложность типовой программной модели движения КА, описываемой системой дифференциальных уравнений 13 порядка, содержащей в правых частях упрощенные функции моделирования возмущающих и управляющих воздействий, экспериментально оценена как 10 тыс. операций умножения. Путем последовательного увеличения сложности исходной модели и измерения затрат времени на моделирование 10000 с полета КА получены зависимости ускорения и эффективности параллельных вычислений от сложности модели, характеристик процессоров и быстродействия коммуникационных сред перечисленных выше вычислительных систем.

На рис. 5 представлены результаты экспериментальных исследований. В экспериментах измерялись затраты времени при параллельном и последовательном интегрировании движения КА с использованием одношаговых блочных методов и вычислялись ускорение и эффективность параллельных вычислений на вычислительных системах: сети Fast Ethernet (рис. 5 а, б), узлах и каналах связи SCI-кластера СПбГУ (рис. 5 в, г), SMP-узлах кластера ННГУ (рис. 5 д, е), суперкомпьютере CRAY T3E (рис. 5 ж, з). На оси абсцисс представлено количество вычислительных операций.

Для ссылок на результаты экспериментов введены следующие обозначения: NET – результаты, полученные на сети Fast Ethernet; SCI-single и SCI-double – результаты, полученные на SCI-кластере в вариантах single и double, соответственно; NN-two – результаты, полученные на двухпроцессорном узле кластера ННГУ, NN-four – на четырехпроцессорном узле; CRAY – результаты, полученные на CRAY T3E, цифры 2 и 4 указывают на двухточечный и четырехточечный блочные методы интегрирования.

Как видно из рис. 5 а, в, д, ж, параллельное решение типовой задачи моделирования движения КА выполняется быстрее, чем последовательное (ускорение больше 1) на всех параллельных вычислительных системах, кроме сети Fast Ethernet. Это объясняется тем, что затраты времени на вычисления для сети Fast Ethernet значительно ниже по сравнению с затратами времени на обмен данными. Однако, с увеличением сложности модели (начиная, примерно, с 25 тыс. вычислительных операций) решение с использованием параллельных методов выполняется быстрее, чем с использованием последовательных методов и на сети Fast Ethernet. Это справедливо как для двухточечного, так и для четырехточечного блочных методов.

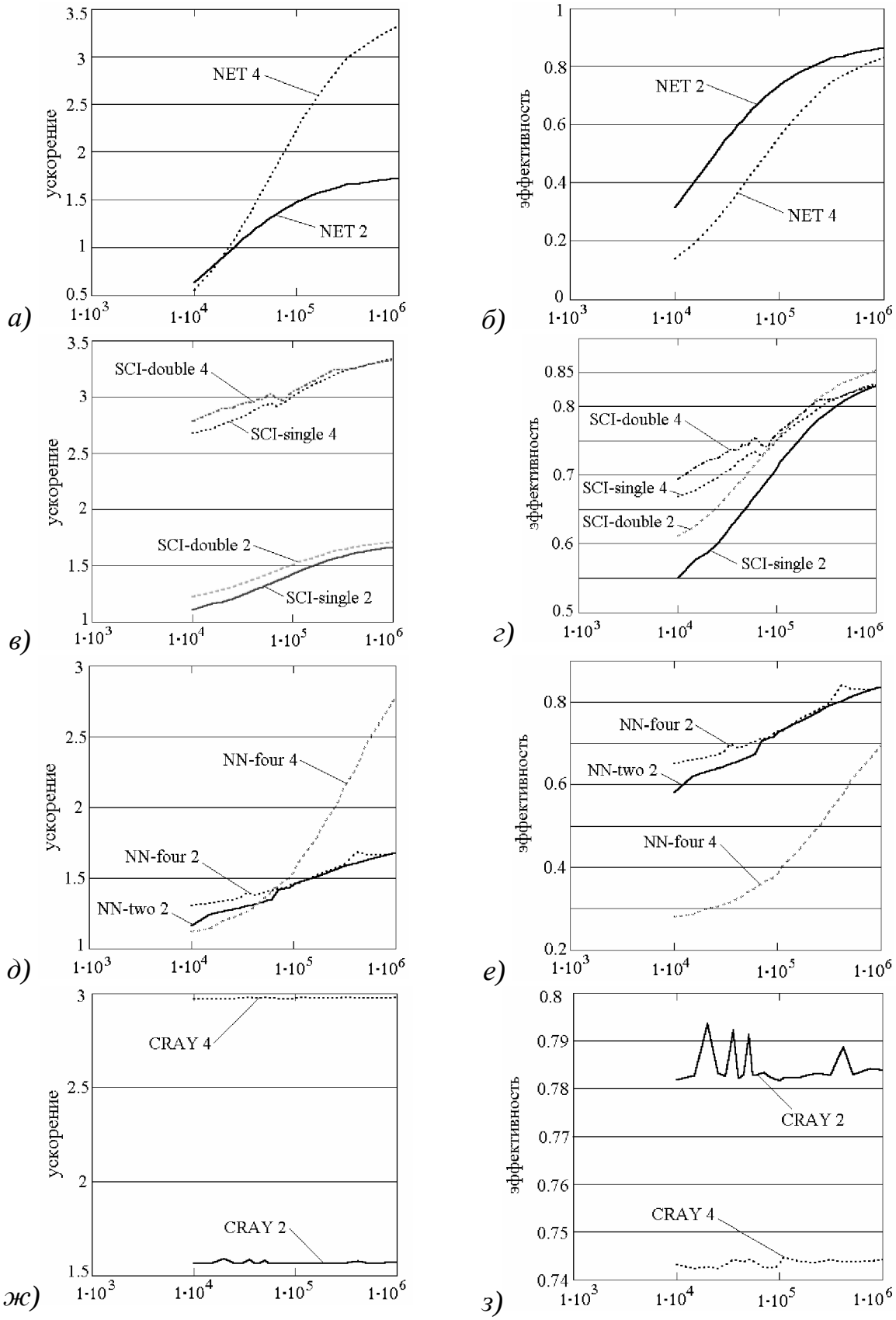


Рис. 5. Ускорение и эффективность параллельного моделирования движения КА

Из рис. 5 а, в, д, ж видно, что с усложнением правых частей уравнений ускорение для всех параллельных вычислительных систем в случае двухточечного метода стремится к двум, в случае четырехточечного метода – к четырем, что согласуется с максимальными теоретически возможными значениями.

На рис. 5 б, г, е, з представлена эффективность – отношение ускорения к количеству процессоров, на которых решается задача.

Используя тесты скорости обменов [2], оценены затраты времени на коммуникации (табл. 1).

Таблица 1

Блочный метод	**Время, секунды**					
	NET	SCI-single	SCI-double	NN-two	NN-four	CRAY
двухточечный	2,139	0,208	0,043	0,404	0,660	0,219
четырехточеный	8,200	1,027	0,926	--------	3,431	0,525

На основании экспериментальных данных получены формулы для оценки эффективности применения параллельных вычислительных систем для моделирования движения КА в зависимости:

- от сложности модели m (коэффициент сложности правых частей уравнений),
- характеристик процессоров и быстродействия коммуникационной среды системы,
- методов блочного интегрирования.

Для двухточечного блочного одношагового метода:

$$m > \frac{t_c + C_p - C_1}{3t_f}, \qquad (1)$$

для четырехточечного блочного одношагового метода:

$$m > \frac{t_c + C_p - C_1}{15t_f}, \qquad (2)$$

где C_1 и C_p – суммарные затраты времени на выполнение арифметических операций для последовательной и параллельной реализаций методов интегрирования, соответственно;

t_c – время, необходимое для обмена данными между процессорами;

t_f – время вычисления правых частей уравнений.

С помощью этих формул можно определить во сколько раз исходная задача должна быть сложнее, чтобы ее решение на параллельной

вычислительной системе стало эффективно (ускорение больше единицы). Так, для используемой сети Fast Ethernet при усложнении правых частей уравнений модели движения в 2.43 раза применение двухточечного метода становится эффективным. Для четырехточечного метода этот коэффициент равен 2.19.

Выводы

Экспериментальные исследования показывают, что моделирование динамических объектов, описывающихся системами обыкновенных дифференциальных уравнений и решаемых с помощью блочных методов интегрирования, может быть реализовано на кластерных и параллельных вычислительных системах. Эффективность такого моделирования зависит как от производительности процессоров, коммуникационных средств, так и от сложности модели. Наиболее эффективными для построения ПНМК являются SMP-системы с узлами из двух и четырех процессоров.

Литература

1. *Пиза Н.Д., Кудерметов Р.К.* Применение кластерных систем для моделирования движения космического аппарата // Высокопроизводительные параллельные вычисления на кластерных системах. Материалы третьего международного научно-практического семинара / Под ред. проф. Р.Г. Стронгина. – Нижний Новгород: Изд-во Нижегородского госуниверситета. 2003. С. 127-135.

2. *Андреев А.Н., Воеводин Вл.В.* Методика измерения основных характеристик программно-аппаратной среды // (www.dvo.ru/bbc/benchmarks.html).

АЛГОРИТМ СОЗДАНИЯ ПОДСИСТЕМ В ВЫЧИСЛИТЕЛЬНЫХ СИСТЕМАХ С ПРОИЗВОЛЬНОЙ СТРУКТУРОЙ

М.С. Седельников

Сибирский государственный университет телекоммуникаций и информатики, г. Новосибирск

Введение

Решение сложных задач требует применения параллельных вычислительных систем (ВС). Эффективность использования ВС определяется тем, как организован процесс их функционирования. Как правило, ВС используются в монопрограммном режиме. В данной работе рассматривается одна из проблем, возникающих при одновременном решении на ВС нескольких параллельных задач.

Постановка задачи

Пусть имеется ВС, на каждую из элементарных машин (ЭМ) которой поступают параллельные задачи. Каждая задача характеризуется рангом (требуемым числом ЭМ) и временем решения. Любая ЭМ содержит расписание для исполнения распределенных на неё задач. Под временем загрузки ВС понимается максимум времени решения всех задач по всем ЭМ. Требуется распределить поступающие задачи по ЭМ системы (т.е. создать для каждой из них подсистему) так, чтобы время загрузки ВС было минимально.

Такая задача является *NP*-полной [1], что является достаточным основанием для отказа от поиска точного алгоритма для её решения. Интерес представляют приближенные и эвристические алгоритмы, которые позволяют получить субминимальное значение времени загрузки [2, 3]. Среди них можно выделить централизованные и децентрализованные алгоритмы. Тенденции к повышению производительности ВС путем увеличения числа ЭМ [4] заставляют отдавать предпочтение вторым. В данной работе предлагается эвристический децентрализованный алгоритм диспетчера распределения параллельных задач по вычислительной системе.

Схема алгоритма

Пусть всем ЭМ присвоены уникальные номера. На ЭМ с номером k имеется очередь Q^k поступающих задач. Диспетчер ЭМ извлекает задачу ранга r из своей очереди и создаёт подсистему размера r путем, по возможности, равномерного распределения требуемого числа ЭМ среди своих соседей. Другими словами, диспетчер строит дерево подсистемы, в котором корнем является ЭМ k, инициирующая создание подсистемы, а ЭМ каждого следующего уровня являются соседями одной или нескольких ЭМ с предыдущего. Листьями дерева являются ЭМ, на которые распределена одна ЭМ.

Работа алгоритма диспетчера основана на событиях, обмене сообщениями между ЭМ и их состояниях. Каждая ЭМ может находиться в одном из 3-х состояний. ***I*** (***I***dle) – ЭМ не участвует в создании подсистемы. ***PD*** (***P***rimary ***D***istribution) – ЭМ распределяет поступившую на нее задачу и создает подсистему для неё. ***SD*** (***S***econdary ***D***istribution) – ЭМ создает подсистему по запросу от другой ЭМ. Имеются 6 видов сообщений: запрос, согласие, отказ, подтверждение, отмена и информационное.

Далее примем следующие условные обозначения:

M_q^k – множество, состоящее из ЭМ k и множества её соседей, среди которых распределено требуемое число ЭМ.

STATE – состояние текущей ЭМ;

$M^s = \{j\}$, $T^s = \{T_j\}$ – множество соседей ЭМ s и их загрузка соответственно;

M_q^s – множество, состоящее из ЭМ s и множества опрашиваемых соседей;

$R_q^s = \{r_j\}$ – число ЭМ, которые распределены по соседям ЭМ s;

$L_q^s = \{l_j\}$ – множество предельных соседей (l_j = 1, если r_j максимально, иначе l_j =0);

$M_+^s \subseteq M_q^s$ – подмножество соседних с s ЭМ, приславших отказ или согласие;

k_q^s – номер ЭМ, от которой ЭМ s был получен запрос на создание подсистемы;

N^*, r^*, t^* – параметры обрабатываемой задачи.

КОНЕЦ – окончание работы алгоритма обработки события.

Решение о включении соседей в множество M_q^s, принимаемое ЭМ s, основано на критерии ***F*** (T_s, T_k , r, Δ), где T_s – загрузка ЭМ s, T_k – загрузка ЭМ k, инициирующей создание подсистемы, r – требуемое число ЭМ, Δ – параметр (отклонение от загрузки ЭМ k).

Приведем алгоритмы обработки каждого события.

Событие 1: на ЭМ k из очереди Q^k поступила задача вида {N, r, t}.

Шаг 1. *Если* STATE ≠ ***I***, *то* задача возвращается в очередь КОНЕЦ.

Шаг 2. *Если* r=1, *то* $T_k := T_k + t$, подсистема для задачи {N, r, t} создана, всем ЭМ из множества M^s отправляется информационное сообщение {T_k}, STATE:= ***I***, КОНЕЦ.

Шаг 3. STATE := ***PD***, N^* := N, r^*:= r, t^*:=t.

Шаг 4. Согласно критерию **F** (T_s, T_k, r, Δ) из множества соседей M^k формируется множества M_q^k, R_q^k, L_q^k, $M_+^k := \{k\}$.

Шаг 5. Всем ЭМ s из множества M_q^k отправляется запрос {N, r_s, t, T_s, T_k}, $r_s \in R_q^k$.

Событие 2: на ЭМ s пришёл запрос от ЭМ i, вида {N, r, t, T, T_k}.

Шаг 1. *Если* STATE ≠ SD *и* STATE ≠ I, *то* ЭМ i отправляется отказ {N, 0, T}, КОНЕЦ.

Шаг 2. *Если* STATE = SD *и* $k_q^s \neq i$, *то* ЭМ i отправляется отказ {N, 0, T}, КОНЕЦ.

Шаг 3. *Если* $T_s \neq T$, *то* ЭМ i отправляется отказ {N, 0, T_s }, КОНЕЦ.

Шаг 4. *Если* STATE = ***I***, *то* переход на шаг5, *иначе* на шаг11.

Шаг 5. N^* := N, r^* := r, t^* := t, $k_q^s := i$, STATE = ***SD***.

Шаг 6. Согласно критерию **F** (T_j, T_k, r, Δ) из множества соседей M^s формируется множество M_q^s, R_q^s, L_q^s, $M_+^s := \{s\}$.

Шаг 7. *Если* $|M_q^s| = 1$ *и* $r > 1$, *то* ЭМ *i* отправляется отказ {*N*, 1, *T*}, КОНЕЦ.

Шаг 8. *Если* $|M_q^s| = 1$, *то* ЭМ *i* отправляется согласие {*N* }, КОНЕЦ.

Шаг 9 Всем ЭМ *j* из множества M_q^s отправляется запрос вида {*N*, r_j, *t*, T_j, T_k}, $r_j \in R_q^s$, КОНЕЦ.

Шаг 10. Согласно критерию **F** (T_j, T_k, r, Δ) из множества соседей M^s переформируются множества M_q^s, R_q^s, L_q^s и M_+^s. *Если* невозможно сформировать указанные множества, *то* ЭМ *i* отправляется отказ {*N*, *r**, *T*}, КОНЕЦ.

Шаг 11. *r**:= *r*.

Шаг 12. Всем ЭМ *j* из множества M_q^s отправляется запрос {*N*, r_j, *t*, T_j, T_k}, $r_j \in R_q^s$.

Событие 3: на ЭМ *s* пришло согласие от ЭМ *i*, вида {*N*, l}.

Шаг 1. *Если* STATE ≠ ***PD*** *и* STATE ≠ ***SD***, *то* ЭМ *i* отправляется отказ {*N*, 0, *T*}, КОНЕЦ.

Шаг 2. $M_+^s := M_+^s \setminus \{i\}$

Шаг 3. *Если* $M_+^s \neq M_q^s$, *то* КОНЕЦ.

Шаг 4. *Если* $\sum_{r_s \in R_q^s} r_s = r*$, *то* переход на шаг5, *иначе* на шаг8.

Шаг 5. *Если* STATE = ***PD*** *то* переход на шаг6, *иначе* на шаг7.

Шаг 6. Всем ЭМ *j* из множества M_q^s отправляется подтверждение вида {*N**}, загрузка ЭМ $j \in M_q^s$ $T_j := T_k + t*$. Подсистема для задачи {*N**, *r**, *t**} создана. Всем ЭМ из множества M^s отправляется информационное сообщение {T_s}. STATE := I. КОНЕЦ.

Шаг 7. ЭМ k_q^s отправляется согласие вида {*N**, $\prod_{l_j \in L_q^s} l_j$ }, КОНЕЦ.

Шаг 8. *Если* $\prod_{l_j \in L_q^s} l_j = 1$ *и* $|M_q^s| = |M^s|$, *то* переход на шаг9, *иначе* на шаг12.

Шаг 9. *Если* STATE=***PD***, *то* переход на шаг10, *иначе* на шаг11.

Шаг 10. Подсистему под задачу {*N**, *r**, *t**} создать невозможно. Всем ЭМ *j* из множества M_q^s отправляется отмена вида {*N**}, STATE=***I***, КОНЕЦ.

Шаг 11. ЭМ k_q^s отправляется отказ вида {*N**, $\sum_{r_s \in R_q^s} r_s$, T_s}, КОНЕЦ.

Шаг 12. Согласно критерию **F** (T_j, T_k, *r**, Δ) из множества соседей M^s переформируются множества M_q^s, R_q^s, L_q^s и M_+^s. *Если* невозможно

сформировать указанные множества, *то* подсистему под задачу {*N**, *r**, *t**} создать невозможно, всем ЭМ *j* из множества M_q^s отправляется отмена вида {*N**}, STATE := I, КОНЕЦ.

Шаг 13. Всем ЭМ *j* из множества M_q^s отправляется запрос {*N*, r_j, *t*, T_j, T_k}, $r_j \in R_q^s$.

Событие 4: на ЭМ *s* пришел отказ от ЭМ *i*, вида {*N*, *r*, *T*}.

Шаг 1. *Если* $T_i \neq T$, *то* $T_i := T^*$, $M_q^s := M_q^s \setminus \{i\}$, $R_q^s := R_q^s \setminus \{r_i\}$, $L_q^s := L_q^s \setminus \{l_i\}$, *иначе* $r_i := r$, $l_i := 1$.

Шаг 2. *Если* $M_+^s \neq M_q^s$, *то* КОНЕЦ, *иначе* на шаг 3.

Шаг 3. *Если* STATE=***PD***, *то* переход на шаг4, *иначе* на шаг 8.

Шаг 4. *Если* $\prod_{l_j \in L_q^s} l_j = 1$, *то* переход на шаг5, *иначе* на шаг 6.

Шаг 5. Подсистему под задачу {*N**, *r**, *t**} создать невозможно. Всем ЭМ *j* из множества M_q^s отправляется отмена вида {*N**}, STATE = ***I***, КОНЕЦ.

Шаг 6. Согласно критерию **F** (T_j, T_k, *r*, Δ) из множества соседей M^s переформируются множества M_q^s, R_q^s, L_q^s и M_+^s. *Если* невозможно сформировать указанные множества, *то* подсистему под задачу {*N**, *r**, *t**} создать невозможно, всем ЭМ *j* из множества M_q^s отправляется отмена вида {*N**}, STATE=***I***, КОНЕЦ.

Шаг 7. Всем ЭМ *j* из множества M_q^s отправляется запрос {*N*, r_j, *t*, T_j, T_k}, $r_j \in R_q^s$.

Шаг 8. *Если* STATE = ***SD***, *то* переход на шаг 9, *иначе* КОНЕЦ.

Шаг 9. *Если* $\prod_{l_j \in L_q^s} l_j = 1$, *то* переход на шаг 10, *иначе* на шаг 11.

Шаг 10. ЭМ k_q^s отправляется отказ вида {*N**, $\sum_{r_s \in R_q^s} r_s$, T_s}, КОНЕЦ.

Шаг 11. Согласно критерию **F** (T_j, T_k, *r*, Δ) из множества соседей M^s переформируются множества M_q^s, R_q^s, L_q^s и M_+^s. *Если* невозможно сформировать указанные множества, *то* ЭМ k_q^s отправляется отказ {*N*, $\sum_{r_s \in R_q^s} r_s$, T_s}, КОНЕЦ.

Шаг 12. Всем ЭМ *j* из множества M_q^s отправляется запрос {*N*, r_j, *t*, T_j, T_k}, $r_j \in R_q^s$.

Событие 5: на ЭМ *s* пришло подтверждение от ЭМ *i*, вида {*N*}.

Шаг 1. Всем ЭМ *j* из множества M_q^s отправляется подтверждение вида {*N**}, загрузка ЭМ $j \in M_q^s$ $T_j := T_k + t^*$, подсистема для задачи {*N**, *r**, *t**} создана.

Шаг 2. Всем ЭМ из множества M^s отправляется информационное сообщение вида $\{T_s\}$. STATE:=***I***.

Событие 6: на ЭМ *s* пришла отмена от ЭМ *i*, вида $\{N\}$.

Шаг 1. Всем ЭМ из множества M_q^s отправляется отмена вида $\{N^*\}$. STATE:=***I***.

Событие 7: на ЭМ *s* пришло информационное сообщение от ЭМ *i*, вида $\{T\}$.

Шаг 1. *Если* $i \in M^s$, *то* T_i:=*T*, *иначе* $M^s := M^s \mathbf{r} \{i\}$, $T^s := T^s \mathbf{r} \{T_i := T\}$.

Событие 8: на ЭМ *s* пришло сообщение приветствия от ЭМ *i*, вида $\{T\}$.

Шаг 1. *Если* $i \in M^s$, *то* $T_i := T$, *иначе* $M^s := M^s \mathbf{r} \{i\}$, $T^s := T^s \mathbf{r} \{T_i := T\}$.

Шаг 2. ЭМ *i* отправить информационное сообщение вида $\{T_s\}$.

Алгоритм формирования множеств M_q^s, R_q^s, L_q^s по критерию **F** (T_s, T_k, *r*, Δ).

Шаг 1. ЭМ $j \in M^s$ включается во множество M_q^s, если $T_k - T_j \le \Delta$.

$\forall l_j \in L_q^s \; l_j := 0$, $\forall r_j \in R_q^s \; r_j := \left[\frac{r}{|M_q^s|}\right] \vee r_j := \left[\frac{r}{|M_q^s|}\right] + 1$ и $\sum_{r_j \in R_q^s} r_j = r$.

Алгоритм переформирования множеств M_q^s, R_q^s, L_q^s по критерию **F** (T_s, T_k, *r*, Δ).

Шаг 0. $r_{top} := 0$, $r_{\max} := \max_{l_j = 0} r_j$.

Шаг 1. *Если* $\sum_{l_j = 0} (r_{\max} - r_j + r_{top}) \ge r - \sum_{j \in R_q^s} r_j$, *то* переход на шаг 2, *иначе* на шаг 3.

Шаг 2. Просматриваем r_j в порядке возрастания $\forall j \; l_j = 0 \; r_j := r_j + \Delta_j \; l_j = 1$, где $r_j \le r_{\max}$ и $\sum_{j \in R_q^s} r_j = r$. КОНЕЦ.

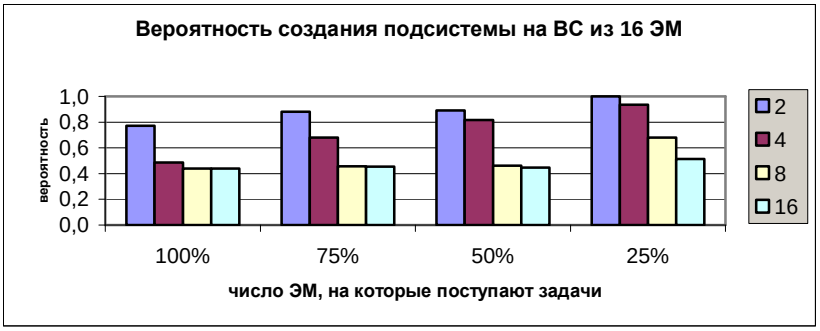
Шаг 3. *Если* $\exists j \in M^s \; j \notin M_q^s$ и $T_k - T_j \le \Delta$, *то* $M_q^s := M_q^s \mathbf{r} \{j\}$, $R_q^s := R_q^s \mathbf{r} \{r_j := 0\}$, $L_q^s := L_q^s \mathbf{r} \{l_j := 0\}$, переход на шаг1, *иначе* на шаг4.

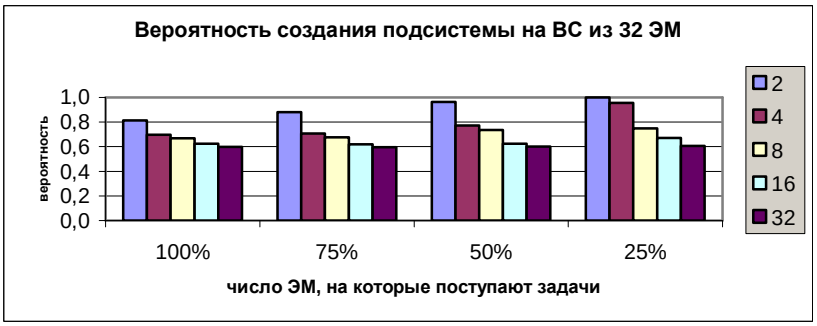
Шаг 4. *Если* $\forall j \; l_j = 1$, *то* переформировать множества невозможно, КОНЕЦ, *иначе* $r_{top} := r_{top} + 1$, на шаг 1.

Моделирование

При моделировании алгоритма был использован вычислительный кластер ЦПВТ СибГУТИ [5]. Рассматривались две ВС, содержащие 16 и 32 ЭМ со структурой сети связи, представленной оптимальным D_2-графом [2]. Число ЭМ, на которые поступали задачи, составляло 25, 50, 75, 100 % от общего числа машин. Рассматривались задачи случайного

ранга в интервале от 1 до 2, 4, 8, 16, 32. Целью моделирования являлось получение эмпирической вероятности распределения алгоритмом параллельных задач.





Полученные результаты показывают, что предложенный алгоритм позволяет распределить большую часть задач, даже в случае одновременного поступления их на все ЭМ системы.

Литература

1. *Гэри М., Джонсон Д.* Вычислительные машины и труднорешаемые задачи // М.: Мир, 1982.

2. *Евреинов Э.В., Хорошевский В.Г.* Однородные вычислительные системы // Новосибирск: Наука, 1978.

3. *Седельников М.С.* Параллельный алгоритм распределения набора задач по машинам вычислительной системы // Материалы всероссийской научной конференции «Наука. Техника. Инновации. – 2003», 2003. Ч. 2. С. 24-25.

4. Top500 Список 500 самых мощных компьютеров мира. 22-я редакция [Электронный ресурс] / Информационно-аналитический центр по параллельным вычислениям ; Электрон. дан. – М.: Лаборатория Параллельных информационных технологий НИВЦ МГУ, 2004. Режим доступа: http://www.parallel.ru/computers/top500.list.html свободный. – Загл. с экрана. – Яз. рус.

5. *Хорошевский В.Г., Майданов Ю.С., Мамойленко С.Н., Павский К.В., Моренкова О.И.* Живучая кластерная вычислительная система // Труды школы-семинара «Распределенные кластерные вычисления», Красноярск, изд-во Красноярского государственного университета, 2001.

СЕРВИС МЕЖКЛАСТЕРНЫХ КОММУНИКАЦИЙ ПАРАЛЛЕЛЬНЫХ ПРОГРАММ

А.В. Селихов

Институт вычислительной математики и математической геофизики СО РАН, г. Новосибирск

Введение

Рост потребности в высокопроизводительных вычислительных ресурсах привел к необходимости объединения имеющихся вычислительных систем на разных уровнях и с разной степенью интеграции. Результатом стало появление SMP-систем, мультикомпьютеров, кластеров и сетей рабочих станций. Следующий шаг в увеличении доступной вычислительной мощности ведет к созданию распределенных информационно-вычислительных сетей. Работы в направлении создания единого представления глобальных информационно-вычислительных ресурсов привели к созданию концепции Grid [1,2], одной из реализаций которой стала система Globus [3].

Несмотря на большое количество работ в области организации параллельных вычислений с использованием Grid-систем, до сих пор не решены все проблемы, связанные с объединением вычислительных кластеров с целью запуска на них параллельного приложения, использующего библиотеку MPI. В частности, такие проблемы существуют для кластеров, все рабочие узлы которых имеют только локальные IP-адреса и поэтому не могут быть получателями или отправителями информации при взаимодействии с узлами внешней сети по протоколу TCP/IP, используемому для низкоуровневой передачи сообщений в MPICH-реализации библиотеки MPI. Единственным узлом такого кластера, имеющим доступ во внешнюю сеть, является хост-узел. Такая ситуация является следствием различных причин, в частности, установленной политики сетевой безопасности или ограничений на количество доступных IP-адресов. Доступные реализации библиотеки MPI для организации вычислений на Grid, такие как MPICH-G2 [4] для системы Globus и PACX-MPI [5], работоспособны только в случае наличия у всех узлов кластера глобальных IP-адресов и возможности обмена сообщениями между любыми двумя узлами.

Целью создания Сервиса межкластерных коммуникаций (СМКК) параллельных программ является преодоление этих трудностей путем организации передачи сообщений между узлами различных кластеров, объединенных в Grid-систему, через хост-узлы этих кластеров. Разработка СМКК имеет также целью создание примера *низкоуровневого*

сервиса, интегрируемого в программные системы организации параллельных вычислений на Grid-системах.

Основные понятия и принципы

Рассматриваемые вычислительные кластеры, отличительные свойства которых были указаны во Введении, могут быть названы «*закрытыми*», а все вычислительные узлы таких кластеров, кроме хост-узла, – *внутренними* узлами кластера.

В этой работе под *сервисом* понимается совокупность функций, предоставляемых посредством интерфейсов, и реализуемых множеством взаимосвязанных программных компонент. СМКК обеспечивает передачу сообщений между внутренними узлами различных закрытых кластеров через имеющие непосредственную связь между собой хост-узлы этих кластеров.

Разрабатываемый Сервис межкластерных коммуникаций параллельных программ основан на следующих основных принципах.

Прозрачность. Этот принцип определяет неизменность исходного кода параллельного приложения, гарантируя, что параллельное приложение будет перенесено на Grid-систему без изменения (требуется только перекомпиляция исходного кода для каждой архитектуры), сохраняя его исходную работоспособность.

Атомарность. СМКК реализует только те функции, которые необходимы для обеспечения передачи MPI-сообщений от одного внутреннего узла другому через хост-узлы. Обеспечение распределения процессов приложения по узлам кластеров, запуск параллельных процессов, обеспечение других функций, в том числе на основе СМКК, реализуется другими сервисами и программными компонентами, в том числе путем взаимодействия с СМКК на основе общих интерфейсов, встраиваемых в сервис при необходимости.

Динамизм. СМКК должен обеспечивать работу параллельного приложения в условиях изменяющейся конфигурации Grid-системы, при подключении и отключении кластеров, представляющих общий вычислительный ресурс. Этот принцип может быть удовлетворен частично, путем реконфигурации сервиса только во время простоя, или полностью, обеспечивая реконфигурацию во время выполнения приложений.

Кроме этих общих принципов, в качестве отличительной особенности СМКК можно отметить осуществление всех коммуникаций, в том числе и между внутренними узлами кластера, через шлюз межкластерных коммуникаций. Такой выбор заведомо ухудшает производительность коммуникаций «точка-точка», однако может существенно улучшить производительность групповых коммуникаций. Кроме того, такой подход не требует разработки системы поддержки очередей сообщений

на вычислительных узлах, так как все сообщения буферизируются на шлюзовом сервере. Использование высокопроизводительного многопоточного шлюзового сервера может существенно снизить дополнительные накладные расходы на передачу *синхронных* сообщений, в то время как асинхронные пересылки могут сохранить свою производительность. Передача всех сообщений через шлюзовой сервер позволяет также реализовать программную отказоустойчивость параллельного приложения по протоколу MPICH-V[6], основанному на временном хранении всех передаваемых между процессами сообщений.

Структура сервиса

Сервис межкластерных коммуникаций состоит из двух программных компонент: коммуникационного драйвера библиотеки передачи сообщений MPICH (ch_iccs) и шлюза межкластерных коммуникаций (рис.1).

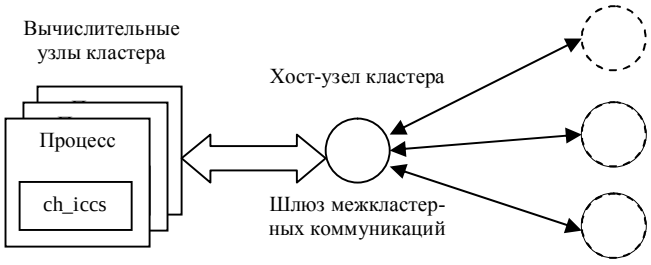


Рис.1. Структура сервиса межкластерных коммуникаций параллельных программ

Коммуникационный драйвер библиотеки MPICH предназначен для реализации высокоуровневых коммуникационных функций MPI. На основе реализации базовых функций, таких как инициализация и завершение работы библиотеки, а также функции передачи, приема и проверки наличия сообщений, коммуникационный драйвер осуществляет подключение процесса параллельной программы к шлюзу межкластерных коммуникаций и прием/передачу сообщений через шлюз. Как уже было указано ранее, передача всех сообщений осуществляется через шлюз, что исключает необходимость в организации очереди входящих сообщений, а также подключения процесса ко всем остальным процессам данного приложения.

Шлюз межкластерных коммуникаций является компонентом, обеспечивающим прием и буферизацию сообщений от процессов-источников в рамках своего кластера и от шлюзов других кластеров, передачу этих сообщений *по запросу* процессам-приемникам своего кластера, а также передачу сообщений, адресованных процессам других кластеров, шлюзам этих кластеров. Для выполнения этих функций ШМК поддерживает Коммуникационную таблицу приложения (КТП), содержащую распределение ранков процессов параллельного приложе-

ния по кластерам. Каждая строка с номером *i* содержит адрес локального узла, если *i*-й процесс параллельного приложения выполняется на данном кластере, иначе эта строка содержит адрес хост-узла того кластера, на котором этот процесс выполняется. Каждый ШМК имеет собственную КТП. Пример коммуникационных таблиц приложения, использующего 8 процессов для трех ШМК, показан в Таблице 1.

Таблица 1. Коммуникационная таблица параллельного приложения, использующего 4 узла трех кластеров

Шлюз 1(194.226.195.11)		Шлюз 2(212.192.181.16)		Шлюз 3(194.226.185.99)	
Ранк	Адрес	Ранк	Адрес	Ранк	Адрес
0	127.0.0.5	0	194.226.195.11	0	194.226.195.11
1	212.192.181.16	1	192.168.0.3	1	212.192.181.16
2	127.0.0.8	2	194.226.195.11	2	194.226.195.11
3	194.226.185.99	3	194.226.185.99	3	127.0.0.12

Любое сообщение, поступающее как от процесса-источника с локального узла, так и от другого шлюза-источника, пересылается соответствующему локальному узлу, если его ранк равен ранку процесса-приемника, иначе оно пересылается следующему шлюзу-приемнику.

На основе описанных свойств СММК, объединяемые кластеры образуют структуру, один из примеров которой показан на рис. 2.

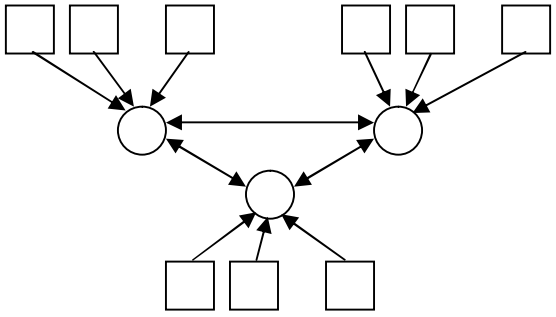


Рис. 2. Пример объединения трех кластеров на основе Сервиса межкластерных коммуникаций

Связи между процессами кластера и шлюзом этого кластера образуют топологию «звезда», в то время как шлюзы различных кластеров связаны между собой по топологии «полный граф».

Функционирование сервиса

Так же как и сервисы, определенные в OGSI [7], сервис межкластерных коммуникаций имеет три этапа функционирования: инициализация, обеспечение сервисных функций и завершение работы. Каждый этап может содержать различное количество шагов в зависимости от выбранной конфигурации сервиса.

Инициализация сервиса межкластерных коммуникаций включает в себя, в первую очередь, запуск шлюзов межкластерных коммуникаций на хост-узлах двух или более кластеров. Интерфейсом запуска ШМК является конфигурационный файл, содержащий адреса всех остальных хост-узлов кластеров. Используя эти адреса, каждый ШМК подключается к соответствующим узлам, формируя полный граф коммуникаций. После окончания инициализации ШМК может быть выполнена инициализация коммуникационного устройства библиотеки MPICH для каждого процесса параллельного приложения. Интерфейсом запуска коммуникационного устройства является командная строка приложения, содержащая адрес ШМК, уникальный идентификатор приложения, общее количество процессов приложения и ранк данного процесса в этом приложении. Используя адрес, коммуникационное устройство подключается к ШМК и передает ему остальные три параметра, на основе которых ШМК формирует КТП.

Коммуникации «точка-точка» между любыми двумя процессами одного приложения осуществляются через ШМК. Отправка сообщения от процесса с ранком i кластера A процессу с ранком j кластера B состоит из передачи этого сообщения на ШМК кластера A, пересылки этого сообщения на ШМК кластера B и, наконец, доставки этого сообщения от ШМК кластера B процессу с ранком j. В соответствии с такой схемой, для любой конфигурации Grid-системы на основе СМКК передача сообщения между любой парой процессов требует ровно три пересылки.

Групповые коммуникации с использованием СМКК могут быть оптимизированы с учетом особенности формируемой структуры связей. Например, рассылка сообщений «один-всем» состоит из передачи на ШМК сообщения с дополнительным флагом, указывающим специфику данной коммуникационной операции, рассылки этим ШМК данного сообщения всем остальным ШМК, адреса которых указаны в КТП, и доставки сообщения всем процессам. Передача сообщения «все-одному» также может быть оптимизирована путем сборки и буферизации всех сообщений в ШМК кластера процесса-приемника, исключая ненужную синхронизацию и ожидание процессов-отправителей. Особенностью реализации групповых операций является необходимость переноса некоторых высокоуровневых операций библиотеки MPICH на уровень функций драйвера низкоуровневых коммуникаций.

В отличие от обычной коммуникационной среды, формируемой при использовании MPICH, процессы параллельного приложения связаны только с ШМК своего кластера и поэтому нечувствительны к отключениям или подключениям новых ресурсов к Grid-системе. Обеспечение динамизма вычислительных ресурсов в процессе выполнения приложения с использованием СМКК основывается на динамическом

обновлении КТП, в случае подключения/отключения других ШМК, а также на использовании буферов сообщений, поддерживаемых Шлюзами межкластерных коммуникаций. Эти буферы позволяют реализовать протокол MPICH-V обеспечения отказоустойчивости параллельного приложения, используя Память Каналов и асинхронный откат процесса на последнюю контрольную точку. СМКК реализует Память каналов, обеспечивая восстановление сообщений, принятых после последней контрольной точки, однако реализация всего протокола MPICH-V требует привлечения дополнительных компонент и реализации дополнительных протоколов, как это сделано, например, в системе CMDE [8].

Заключение

Разработка средств объединения вычислительных кластеров в Grid-системы представляется в настоящее время актуальной задачей. Представленный Сервис межкластерных коммуникаций параллельных программ нацелен на решение проблемы передачи сообщений библиотеки MPI между узлами кластеров, имеющих связь с глобальной сетью только через единственный хост-узел каждого кластера. Реализация компонентов СМКК основывается на опыте разработки системы CMDE, в частности, на использовании Серверов Памяти Каналов, позволяющих осуществлять конвейеризацию синхронной передачи больших сообщений, что дает производительность, близкую к передаче этих сообщений по непосредственному соединению между процессами. Использование СМКК предполагает наличие некоторого дополнительного программного окружения, осуществляющего распределение множества ранков процессов параллельного приложения по различным кластерам, распределение кода приложения по узлам кластеров, формирование входных параметров процессов для использования СМКК, подключение узлов Шлюзов межкластерных коммуникаций в общую сеть и другие. На начальном этапе для этих целей предполагается использовать компоненты системы CMDE, модифицированные для реализации интерфейсов с СМКК. Дальнейшее развитие сервиса может включать в себя как реализацию интерфейсов в соответствии со спецификацией OGSI с целью включения этого сервиса в систему Globus, так и развитие набора сервисов, включающих СМКК и являющихся более узко специализированными для организации метавычислений.

Литература

1. Foster, What is the Grid? A Three Point Checklist. GRIDToday, July 20, 2002.
2. *Foster, Kesselman C., Nick J., Tuecke S.* The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration. Open Grid Service Infrastructure WG, Global Grid Forum, June 22, 2002.

3. *Foster, Kesselman C,* Globus: A Metacomputing Infrastructure Toolkit. Intl J. Supercomputer Applications, 1997. 11(2). P. 115-128.
4. *Karonis N., Toonen B. and I. Foster* MPICH-G2: A Grid-Enabled Implementation of the Message Passing Interface. Journal of Parallel and Distributed Computing, 2003.
5. *Beisel T., Gabriel E., Resch M.* An Extension to MPI for Distributed Computing on MPPs'. In Marian Bubak, Jack Dongarra, Jerzy Wasniewski (Eds.) 'Recent Advances in Parallel Virtual Machine and Message Passing Interface', Lecture Notes in Computer Science, Springer, 1997. P. 75-83.
6. *Bosilca G., Bouteiller A., Cappello F., Djilali S., Fedak G., Germain C.,. Herault T, Lemarinier P., Lodygensky O., Magniette F., Neri V., Selikhov A.* MPICHV: Toward a Scalable Fault Tolerant MPI for Volatile Nodes, In Proceedings of SuperComputing 2002. IEEE, Nov., 2002.
7. *Tuecke S., Czajkowski K., Foster I., Frey J., Graham S., Kesselman C., Maguire T., Sandholm T., Vanderbilt P., Snelling D.* Open Grid Services Infrastructure (OGSI) Version 1.0. Global Grid Forum Draft Recommendation, 6/27/2003.
8. *Selikhov A., Germain C.* CMDE: a channel memory based dynamic environment for fault-tolerant message passing based on MPICH-V architecture // Proc. of 7-th Int. Conference PaCT-2003, Springer Verlag, LNCS 2763, 2003. P. 528-537

ЭФФЕКТИВНОЕ РАСПАРАЛЛЕЛИВАНИЕ ВЫЧИСЛЕНИЙ ДЛЯ ЗАДАЧ ФИЗИКИ АТМОСФЕРЫ

Ю.М. Тырчак

Киевский национальный университет им. Тараса Шевченко, г. Киев, Украина

Математическая модель задачи прогноза региональных атмосферных процессов

Современное развитие экологической науки ставит все более сложные вычислительные задачи, решение которых требует применение сложного математического аппарата при построении модели физических процессов и большого количества вычислений. Решение таких задач становится возможной благодаря развитию мультипроцессорных систем и параллельных вычислений [6].

Сегодня моделирование региональных атмосферных процессов реализуется с учетом того, что поля метеорологических величин в ограниченной области формируются под влиянием макромасштабных циркуляций атмосферы. Поэтому ограниченная область решения рассматривается как часть некоторого целого, и нестационарные краевые условия на его боковых границах формируются на основе данных, полученных для области, которая окаймляет. Кроме этого, при численном решении задач прогноза состояния атмосферы для ограниченной террито-

рии, появляется необходимость сгущать сетку для достижения необходимой точности решения задачи в местах больших градиентов зависимых функций.

Предполагая, что поля метеорологических величин на ограниченной территории формируются на фоне общей циркуляции атмосферы, задачу прогноза погоды для ограниченной области можно сформулировать путем комбинации двух моделей разной полноты: 1) глобальной модели общей циркуляции атмосферы, которая включает упрощенные уравнения, численно решаемые на грубой сетке; и 2) региональной модели, которая включает полные уравнения гидродинамики и тепло- массопереноса, численно решаемые на мелкой сетке. Необходимые для региональной модели предельные условия отождествляются с решением глобальной модели, уравнения которой могут интегрироваться одновременно с уравнениями региональной модели или заранее. Этот метод прогноза на вложенных сетках получил название метода «одностороннего воздействия», поскольку численные результаты внутренней модели не влияют на интегрирование уравнений внешней модели [3].

Широкое применение в существующих моделях циркуляции атмосферы и прогноза погоды нашли фундаментальные уравнения динамики вяжущей сплошной среды, которые базируются на универсальных законах, таких, как сохранение массы, сохранение количества движения, сохранение энергии, сохранение скалярных величин, состояния среды.

Прогноз значений метеорологических величин над ограниченной территорией $\overline{G}$ осуществляется на основе метода «одностороннего воздействия». Другими словами, как предельные условия для региональной модели используются результаты анализа и прогноза, полученные с помощью макромасштабной модели (полушара или глобальной).

Пусть состояние атмосферы в пространстве $r=(\lambda,\varphi,\sigma)$ макромасштабной области $G(r)\subset\overline{G}(r)$ определяется вектором $\Re(r,t)=(u,v,w,\pi,T,q,q_L,q_W,k,\varepsilon)$ дискретных значений анализа и прогноза $\Re(r,t^m)=\Re^m(r)$, полученных на основе макромасштабной модели в моменты времени $t=t^m$ $(m=0,1,...,M)$ с шагом $\tau=t^m-t^{m-1}$. Тогда для определения состояния атмосферы на ограниченной территории $\overline{G}$ при условии $\forall t\in[t^{m-1},t^m]$ решается задача, которая в векторном представлении имеет вид:

$$\frac{\partial\Re}{\partial t}=D\Re,\ \forall t\in\left[t^{m-1},t^m\right],\ \forall r\in\overline{G}$$

$$\Re(r,t^m)=\Re^m(r),\ m=0,\ 1,...,M$$

В дальнейшем схема интерполируется с помощью интерполяции с кратными узлами полиномом Эрмита при кратности три (используется предыстория значения для каждой точки, которая вычисляется)

Схема распараллеливания вычислений

Сложность написания параллельных программ состоит в том, что необходимо в явном виде согласовывать структуру алгоритма решаемой задачи с архитектурой и структурой вычислительной системы. Как платформа для реализации, выбрана кластерная система благодаря своей дешевизне по сравнению с системами другой архитектуры, масштабируемости и простоте в эксплуатации. Ситуация осложнена еще и тем, что средств отладки параллельных программ практически нет.

В основе работы лежит метод крупноблочного распараллеливания [2], исследование эффективности применения которого к выбранному классу задач проводится. Приведем основные принципы, которыми следует руководствоваться при распараллеливании последовательных алгоритмов. Эффективность работы будущей параллельной программы прямо зависит от соблюдения этих правил:

Важнейшим принципом есть декомпозиция последовательной программы на информационно независимые и сбалансированные по нагрузке блоки вычислений. То есть программу следует поделить на максимальное количество блоков, одновременное выполнение которых не является взаимоисключающим (необходимым и достаточным условием этого есть непересечение входных данных одного процесса с выходными данными другого) таким образом, чтобы объемы вычислений в блоках оставались приблизительно одинаковыми.

При написании параллельной программы следует обращать внимание на то, чтоб в процессе выполнения не было простоев вычислительных узлов, или свести их количество к минимуму. Одним из вариантов избежания таких ситуаций является организация программного буфера данных, необходимых для вычисления: во время выполнения вычислений над одним пакетом данных происходит запрос и пересылка данных, необходимых для следующего блока вычисления. Такой способ организации работы параллельной вычислительной системы напоминает конвейер обычного процессора.

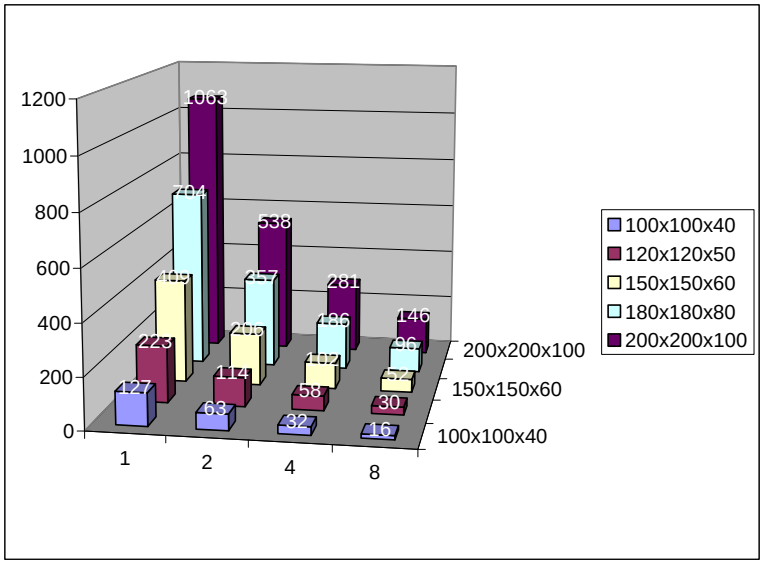
В вычислительных узлах есть много уровней памяти: регистровые блоки процессора, буфер данных процессора, кэш-память процессора, оперативная память, память на дисковой подсистеме, общая память всей системы параллельной обработки (в разных архитектурах последняя по скорости может находиться выше, чем дисковая подсистема) Поэтому следует уменьшать количество обращений к памяти низших уровней,

которые замедляют общую скорость обработки. Решением этого служит фоновая подкачка данных.

Результаты экспериментального программирования

В ходе выполнении работы были проведенные эксперименты, использовавшие метод крупноблочного распараллеливания той части задачи, которая относится к вычислению правых частей системы уравнений – следующего этапа после интерполяции данных, описание распараллеливания которого приводится в работе [1]. Согласно этому методу, основной целью при распараллеливании задачи должно быть уменьшение количества межпроцессного взаимодействия и достижение того, чтобы время работы одной задачи намного превышало время запуска процесса и время, которое теряется на передачу сообщений между процессами. Для достижения этих результатов в алгоритме интерполяции исходная сетка распределялась между вычислительными узлами поровну. После вычислений все результаты собираются на главном узле.

Ниже приведенная диаграмма времени вычислений (в сек., ось *Z*) на разном количестве процессоров (от 1 до 8, ось *X*) при разных размерах мезомасштабной сетки (ось *Y*). Результаты показывают высокую степень распараллеливаемости и масштабируемости прогнозных вычислений, которые в дальнейшем предполагается выполнить полностью на параллельной системе.



Эксперименты проводились на локальном кластере из восьми компьютеров следующей конфигурации: Celeron 1.3G 256MB 10Mbit LAN, OS: RadHat Linux 7.3 kernel ~2.4.20, OS: Windows 2000 Professional, MPICH 1.2.5.2, gcc 2.95

Выводы

При анализе результатов эксперимента оказалось, что имеет место явление суперлинейного ускорения времени выполнения. Это явление объясняется уменьшением количества памяти, которая необходимое для одного процесса задачи с увеличением количества процессоров (уменьшением размера задачи которая приходится на один вычислительный узел). С уменьшением необходимого количества оперативной памяти уменьшается использование файла подкачки и переключение страниц памяти.

Такой результат говорит о перспективности дальнейших работ в направлении крупноблочного распараллеливания остальных частей этой модели и целесообразность применения метода при вычислении других задач этого класса.

ЛИТЕРАТУРА

1. *Прусов В.А., Дорошенко А.Ю., Приходько С.В., Тырчак Ю.М., Черныш Р.И.* Методы эффективного решения задач моделирования и прогнозирования региональных атмосферных процессов // Проблемы программирования. Сб. науч. тр. Киев-2004. №2-3. С. 556-569.
2. *Дорошенко А.Э.* Математические модели и методы организации высокопроизводительных параллельных вычислений. Алгебродинамический подход. // Киев: Научная Мысль, 2000. – 177 с.
3. *Довгий С.А., Прусов В.А., Копейка О.В.* Математическое моделирование техногенных загрязнений окружающей среды // Киев: Научная мысль, 2000. – 247 с.
4. *Воеводин В.В., Воеводин Вл.В.* Параллельные вычисления // Спб.: БХВ-Петербург, 2002. 608 с.
5. *Вальковский В.А.* Распараллеливание алгоритмов и программ. Структурный подход // М.: Радио и связь, 1989. – 176 с.
6. Параллельные вычисления /Под ред. *Г. Родрига*: пер. с англ. // Под ред. Ю.Г. Дадаева. – Г.: Наука, 1986. – 376 с.

ПАРАЛЛЕЛЬНЫЙ АЛГОРИТМ ПОСТРОЕНИЯ ДИСКРЕТНОЙ МАРКОВСКОЙ МОДЕЛИ

Л.П. Фельдман, Т.В. Михайлова

Донецкий Национальный Технический Университет, г. Донецк

Введение

Кластеры обладают высокой производительностью и «живучестью» вычислительных ресурсов и являются одним из примеров вычислительных систем с распределенной обработкой. Одной из основных проблем, возникающих при проектировании, эксплуатации и оптимизации ВС с распределенной обработкой является выработка рекомендации

для рационального использования ресурсов этой вычислительной среды. Для решения этой задачи можно использовать непрерывные [1] и дискретные [2] аналитические модели кластеров.

Дискретная модель кластера с совместным разделением дискового пространства [3] с использованием методики построения дискретных Марковских моделей [4] представлена в [5].

Однако анализ кластерных систем с помощью дискретных Марковских моделей затруднен при большом количестве решаемых задач (M) на ПЭВМ. Количество состояний (L) дискретной Марковской модели резко возрастает при увеличении количества задач. Одним из вариантов решения такой задачи в течение реального времени является распараллеливание алгоритма построения дискретной модели, предварительно оценив трудоемкость последовательного алгоритма.

Оценка трудоемкости алгоритма расчета характеристик функционирования кластерной системы

Алгоритм построения дискретной Марковской модели состоит из двух частей: вычислений матрицы переходных вероятностей и вектора стационарных вероятностей. Трудоемкость первой части вычисляется как

$$L \times L \times 2(N+1)\, t_{сл} + K_1 * 3N\, t_{сл} + K_1$$
$$\{((2N+1)C_{k_2+N-2}^{N-2} - 1)\, t_{сл} + C_{k_2+N-2}^{N-2}[(4k_2 - 5) + \sum_{s=1}^{N} k_s]\; t_{умн}\}, \quad (1)$$

где $$K_1 = N + \sum_{j=1}^{M-1} C_{j+N-1}^{N-1}(C_{j+N-2}^{N-1} + C_{j+N-1}^{N-1} + C_{j+N}^{N-1}) +$$
$$+ C_{M+N-1}^{N-1}(C_{M+N-1}^{N-1} + C_{M+N-2}^{N-1}),$$

Учитывая, что $t_{сл} \approx t_{умн}$, формула (1) примет вид

$$T1_{посл} = L \times L \times 2(N+1)\, t_{оп} + K_1 * 3N\, t_{оп} +$$
$$+ K_1\, \{(2N + 4k_2 + \sum_{s=1}^{N} k_s - 4) C_{k_2+N-2}^{N-2}\}\; t_{оп} \quad (2)$$

Для определения трудоемкости вычисления вектора стационарных вероятностей. Представим СЛАУ в виде

$$\bar{\pi}(k) = \bar{\pi}(0)\mathrm{P}^K, \quad (3)$$

где $\bar{\pi}(0)$ – начальное распределение вероятностей состояний,

K – степень, при которой матрица не изменяется.

Возведение матрицы P в степень K потребует $(1+\log_2 K)$ операций умножения матрицы. При последовательной реализации расчета векто-

ра стационарных вероятностей общее количество вычислений определяется как

$$T2_{посл}=(L^3*t_{умн} + L^2\ (L\text{-}1)*t_{сл})*\ (1+log_2K)+L*(L*\ t_{умн}+\ (L\text{-}1)*\ t_{сл})$$

или

$$T2\text{посл}=(L3 + L2\ (L\text{-}1)\ )*ton\ (1+log2K)+L*\ 2*(L\text{-}1)*\ ton \qquad (4)$$

Вычислительная сложность последовательного алгоритма построения дискретной Марковской модели кластера с разделяемой дисковой памятью определяется суммой формул (2) и (4) и составляет

$$\begin{aligned} T_{посл} &= L\times L\times 2(N+1)\,t_{on} + K_1 * 3N\ t_{on} + \\ &+ K_1\ \{(2N+4k_2+\sum_{s=1}^{N} k_s - 4)C_{k_2+N-2}^{N-2}\}\ t_{on} + \\ &+ (L^3 + L^2(L-1))(1+loq_2K)\,t_{on} + L*2(L-1)*t_{on} \end{aligned} \qquad (5)$$

Отображение параллельного алгоритма Марковской модели на решетку процессоров

Пусть процессор SIMD-структуры имеет топологию- решетку процессорных элементов (ПЭ) $p=n^2$ типа двумерной прямоугольной решетки-тора. Каждый ПЭ может выполнить любую операцию за один такт; время обращения к запоминающему устройству пренебрежительно мало по сравнению со временем выполнения операции.

На каждом ПЭ с индексами i, j вычисляется один элемент матрицы переходных вероятностей p_{ij}. Алгоритм вычисления переходной вероятности между двумя произвольными состояниями приведен в [6].

Отобразим параллельный алгоритм построения матрицы переходных вероятностей размерности L. Временная сложность вычисления одного элемента этой матрицы определяется как

$$T1_{парал} = 5N\ t_{on} + (2N + 4k_2 + \sum_{s=1}^{N} k_s - 4)C_{k_2+N-2}^{N-2}\ \ t_{on} \qquad (6)$$

Всего возможно ненулевых элементов определяется величиной K_1. В этом случае временная сложность параллельного алгоритма вычисления матрицы переходных вероятностей вычисляется следующим образом

$$T1_{парал} = ([\frac{K_1}{p}]+1)\{5N\ t_{on} + (2N + 4k_2 + \sum_{s=1}^{N} k_s - 4)C_{k_2+N-2}^{N-2}\ \ t_{on}\} \qquad (7)$$

Временная сложность последовательного алгоритма определяется (5). Ускорение параллельного алгоритма вычисления матрицы переход-

ных вероятностей на решетке процессоров, определяемое, как отношение формул (6) и (7) для различных значений количества обрабатываемых на кластере задач представлено на рис.1 для решетки процессоров p=16. При этом изменялась конфигурация кластера, т.е. увеличивалось количество устройств (серверов и/или дисков) в узлах. Эффективность параллельного алгоритма вычисления матрицы переходных вероятностей на решетке процессоров с увеличением количества решаемых задач M незначительно растет (рис. 2.) и мало изменяется (в сотых долях) с ростом количества процессоров.

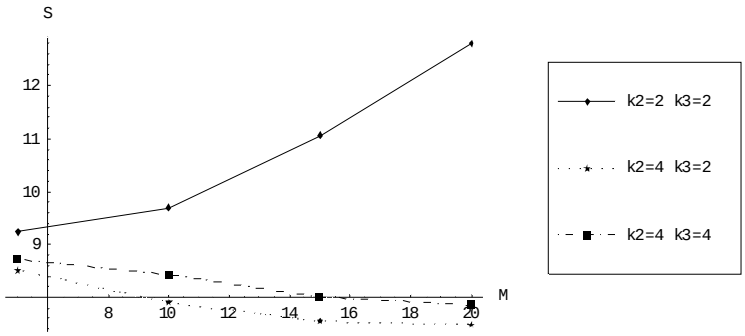


Рис. 1. Ускорение параллельного алгоритма вычисления матрицы переходных вероятностей на решетке процессоров в зависимости от количества решаемых задач M

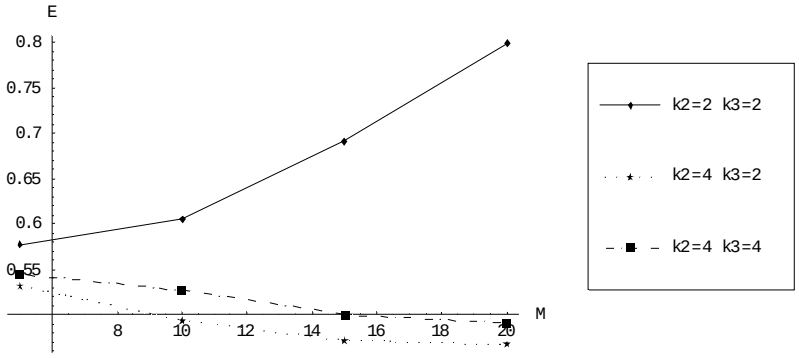


Рис. 2. Эффективность параллельного алгоритма вычисления матрицы переходных вероятностей на решетке процессоров в зависимости от количества решаемых задач M

Эффективность параллельного алгоритма вычисления матрицы переходных вероятностей на решетке процессоров зависит от значений M, k_2,k_3,p.

Распараллелим вычисление вектора стационарных вероятностей на решетку процессоров размерности $p=n^2$ с использованием блочного подхода методом Фокса [5, 6]. Размер блоков матрицы P равен $(L/\sqrt{p})\times(L/\sqrt{p})$.

Временная сложность вычисления вектора стационарных вероятностей определяется как

$$T2_{парал} = \{2L^3 / p * t_{сл} + L^2 / p\sqrt{p}\ t_{сдв}\}(log_2 K + 1) + \\ + L^2 / p * t_{сл} + L / \sqrt{p}\,(\sqrt{p} - 1)\ t_{сдв} \quad (8)$$

где $2L^3 / p$ -количество операций сложения для каждого процессора;

$t_{сдв}$ – максимальная длительность рассылки по строкам решетки;

$L^2 / p\ t_{сдв}$ – длительность рассылки блоков на одной итерации;

$\sqrt{p}$ – количество итераций.

Порядок вычислений последовательного алгоритма – $L^3(1+log_2 K)$.

Ускорение и эффективность параллельного алгоритма вычисления вектора стационарных вероятностей на решетке процессоров представлены на рис. 3 и 4.

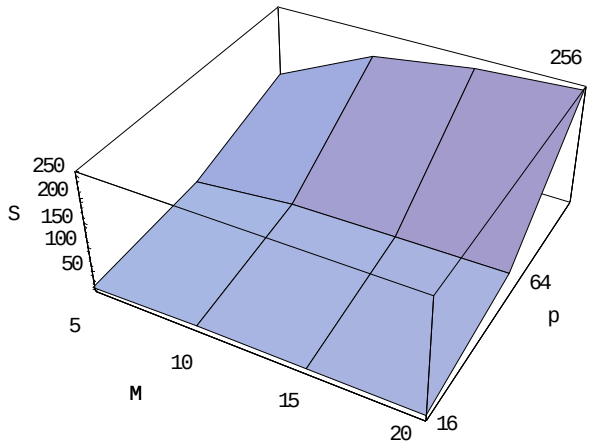


Рис. 3. Ускорение параллельного алгоритма вычисления вектора стационарных вероятностей на решетке процессоров в зависимости от количества решаемых задач и процессоров

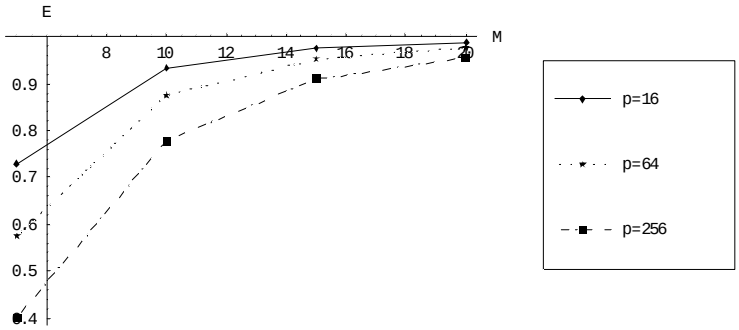


Рис. 4. Эффективность параллельного алгоритма вычисления вектора стационарных вероятностей на решетке процессоров в зависимости от количества решаемых задач и процессоров

Временная сложность алгоритма построения модели Маркова определяется суммой формул (7) и (8).

Ускорение параллельного алгоритма асимптотически приближается к количеству процессоров тора (рис.5) и не зависит от количества устройств (серверов и/или дисков) в узлах кластера. Поэтому в формулах оценки трудоемкости алгоритма вычисления матрицы переходных вероятностей можно отбросить слагаемые, соответствующие количеству устройств в узлах.

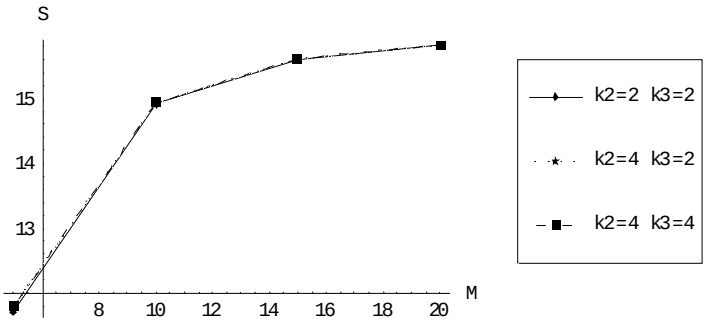


Рис. 5. Ускорение параллельного алгоритма расчета модели Маркова на решетке процессоров в зависимости от количества решаемых задач M

Эффективность параллельного алгоритма расчета модели Маркова на решетке процессоров в зависимости от количества решаемых задач *M* от устройств не изменяется (рис. 6).

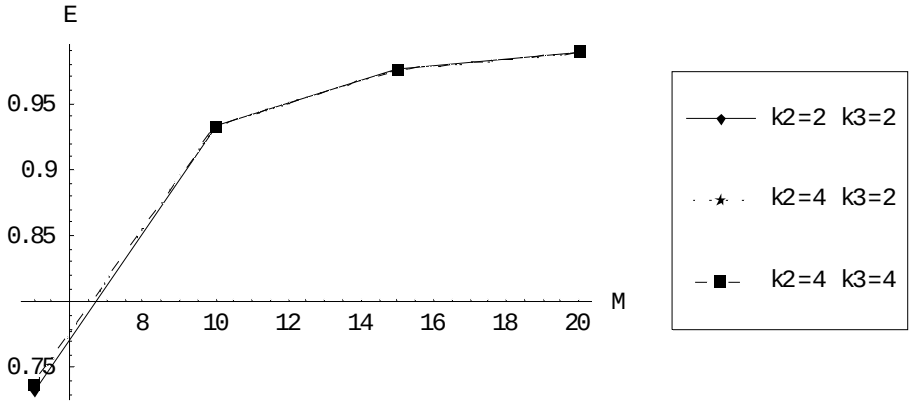


Рис. 6. Эффективность параллельного алгоритма расчета модели Маркова на решетке процессоров в зависимости от количества решаемых задач M

Выводы

Получены оценки трудоемкости алгоритма дискретной Марковской модели кластера с совместным использованием дискового пространства, которые зависят от структуры вычислительной среды, от количества обрабатываемых задач и от особенностей матрицы переходных вероятностей и оценки распараллеливания этого алгоритма на SIMD-структуры, позволяющие считать аналитическую модель на высокопроизводительных ВС.

Литература

1. *Клейнрок Л.* Вычислительные системы с очередями // М.:Мир, 1979, - 600с.
2. Последовательно-параллельные вычисления // Пер. с англ. - М.: Мир, 1985. - 456 с.
3. *Авен О.И.* и др. Оценка качества и оптимизация вычислительных систем // М.: Наука, 1982. – 464 с.
4. *Шнитман В.* Современные высокопроизводительные компьютеры. Информационно-аналитические материалы центра информационных технологий, 1996: http//hardware/app_kis.
5. *Фельдман Л.П., Дедищев В.А.* Математическое обеспечение САПР. Моделирование вычислительных и управляющих систем // Киев: УМК ВО, 1992. – 256 с.
6. *Фельдман Л.П., Михайлова Т.В.* Методы оптимизации состава и структуры высокопроизводительных систем // Научные труды Донецкого государственного технического университета. Серия «Информатика, кибернетика и вычислительная техника» (ИКВТ-2000). Донецк: ДонГТУ. 2000.
7. *Михайлова Т.В.* Параллельная реализация алгоритма построения дискретных моделей Маркова // Научные труды Донецкого государственного технического университета. Серия «Информатика, кибернетика и вычислительная техника»(ИКВТ-2003).-Донецк: ДонГТУ.2003 (в печати).
8. *Гергель В.П., Стронгин Р.Г.* Основы параллельных вычислений для многопроцессорных вычислительных систем // Н.Новгород, ННГУ, 2001.
9. *Корнеев В.В.* Параллельные вычислительные системы // М., 1999. – 312 с.

ЭФФЕКТИВНОСТЬ СПОСОБОВ ОЦЕНКИ АПОСТЕРИОРНОЙ ЛОКАЛЬНОЙ ПОГРЕШНОСТИ ПРИ ПАРАЛЛЕЛЬНОМ РЕШЕНИИ СИСТЕМ ЛИНЕЙНЫХ ОДНОРОДНЫХ ОДУ

Л.П. Фельдман, И.А. Назарова

Донецкий Национальный Технический Университет, г. Донецк

Введение

Моделирование реальных экономических, технических и других процессов, описываемых системами обыкновенных дифференциальных уравнений (СОДУ), представляет собой обширный класс задач, для решения которых применение высокопроизводительной вычислительной техники не только оправдано, но и необходимо. Об этом свидетельствует знаменитый список проблем «большой вызов», в котором такие задачи занимают одно из ведущих мест [1]. Как показала практика, наиболее эксплуатируемым способом создания параллельных методов является распараллеливание хорошо исследованных и многократно апробированных последовательных численных алгоритмов [2]. Современный ка-

чественный численный алгоритм решения СОДУ обязательно должен содержать механизм управления шагом интегрирования на основе информации о погрешности решения на каждом шаге интегрирования [3]. В данной статье рассматривается эффективность альтернативных методов оценки апостериорной локальной погрешности на базе параллельных алгоритмов решения линейных однородных СОДУ с постоянными коэффициентами для многопроцессорных ВС SIMD-архитектуры с распределенной памятью.

1. Общая характеристика способов оценки апостериорной локальной погрешности

Известными методами оценки локальной погрешности решения СОДУ являются: правило Рунге; вложенные методы Рунге-Кутты (ВМРК); метод локальной экстраполяции Ричардсона.

Наиболее простой способ достижения поставленной цели – это удвоение или дублирование шага по правилу Рунге. На основе одной и той же формулы Рунге-Кутты порядка $\boldsymbol{p}$ вычисляют два приближения к решению в одной и той же точке сначала с удвоенным шагом: $2h$, а затем независимо, дважды, с половинным шагом: h. Для точного решения при переходе из точки x_n к точке $x_n + 2\cdot h$, $y(x_n + 2\cdot h)$ получают две аппроксимации y_1(1 шаг $2\cdot h$**)** и y_2(2 шага $\boldsymbol{h}$**)**. Разность между этими значениями используется в качестве оценки локальной погрешности интегрирования, в качестве решения принимается аппроксимация, полученная с половинным шагом, как наиболее точная:

$$
\begin{aligned}
&\boldsymbol{y(x+2h) \cong y_1 + (2h)^p \varphi + O(h^p)} \\
&\boldsymbol{y(x+2h) \cong y_2 + 2(h^p)\varphi + O(h^p)} \\
&\boldsymbol{\Delta y = y_2 - y_1} \\
&\boldsymbol{y(x+2h) \cong y_3 = y_2 + \frac{\Delta y}{2^p - 1} + O(h^{p+1}).}
\end{aligned}
\qquad (1)
$$

Процесс уточнения решения с половинным шагом $\boldsymbol{y_2}$, как видно из (1), повышает точность решения на единицу, и получил название локальной экстраполяции. Данный способ оценки погрешности достаточно прост, однако имеет большие накладные расходы: объем вычислений на один узел сетки возрастает почти втрое.

Второй способ оценки локальной погрешности метода – вложенные методы Рунге-Кутты. Этот способ основан на использовании двух приближенных значений решения в одной точке, но в отличие от правила Рунге приближения вычисляются не по одной, а по двум формулам различных порядков точности $\boldsymbol{p}$ и $\hat{p}$ с одним и тем же шагом [3-4]:

$$\begin{cases} y_{n+1} = y_n + h_n \cdot \sum_{l=1}^{s} b_l \cdot k_l ; \overset{\Gamma}{y}_{n+1} = y_n + h_n \cdot \sum_{l=1}^{s'} \overset{\Gamma}{b}_l \cdot k_l ; \\ k_l = f(x_n + c_l \cdot h_n ; y_l + h_n \cdot \sum_{i=1}^{l-1} a_{li} \cdot k_i ; l = 1,...,s; \\ d_{n+1} = |(\overset{\Gamma}{y}_{n+1} - y_{n+1})| \end{cases} . \quad (2)$$

Для определения локальной погрешности менее точного результата и управления величиной шага интегрирования используется величина d^{n+1}. Вложенный метод Рунге-Кутты $p(\overset{\Gamma}{p})$ – это схема, в которой метод $\overset{\Gamma}{p}$ – го порядка («оценщика погрешности») получается как побочный продукт метода p-го порядка. Порядок аппроксимаций: y_n и $\overset{\Gamma}{y}_n$ обычно отличаются на 1, т.е. $p = \overset{\Gamma}{p} + 1$ или $p = \overset{\Gamma}{p} - 1$. Оба приближения используют практически одни и те же значения шаговых коэффициентов $k_i, i = 1,..,s$, но комбинируют их по-разному. Это позволяет уменьшить количество вычислений функции $f(x,y)$. Для s- стадийных вложенных методов эта величина составит $s + c$, где c – некоторая константа (обычно $c=1$ или $c = 2$).Для сравнения приведем эти оценки для $p= 4$: накладные расходы для метода Рунге-Кутты с правилом Рунге, а именно число вычислений правой части равно 11, а для вложенного метода Фельберга того же порядка точности всего 6.

Метод локальной экстраполяции Ричардсона является обобщением технологии удвоения шага по правилу Рунге [3-5]. Идея этого метода заключается в многократном измельчении шага интегрирования, и также в многократном применении процесса вычисления, названного локальной экстраполяцией. Решение задачи Коши рассматривается при переходе из точки x_n в точку $x_{n+1} = x_n + H$, H - базовая длина шага, $H > 0$. Выбирается ряд натуральных чисел такой, что: $n_1 < n_2 < ... < n_{k-1} < n_k < ...$ и, соответственно, последовательность шагов: $h_1 > h_2 > ... > h_{k-1} > h_k > ...$, где $h_i = H / n_i$. Задается опорный численный метод порядка p и, выполняя n_i шагов интегрирования длиной h_i, вычисляют приближенное решение исходной задачи:

$$T_{i,1} := y_{h_i}(x_0 + H). \quad (3)$$

Выполнив вычисления для ряда последовательных значений i, по рекуррентному соотношению (2) определяют экстраполированные значения $T_{i,j+1}$ для произвольных i,j по формуле (4). Этот процесс получил название локальная полиномиальная кстраполяция:

$$T_{i,j+1} := T_{i,j} + \frac{T_{i,j} - T_{i-1,j}}{(n_i / n_{i-j})^b - 1}. \qquad (4)$$

Здесь величина b равна единице в общем случае, в тоже время для симметричных опорных методов, имеющих разложение погрешности по степеням h^2, b равно двум (каждая экстраполяция исключает две степени h вместо одной).

Таблица 1. Экстраполяционная таблица

p	$p+b$	$p+2b$		$p+(k-2)b$	$p+(k-1)b$
T_{11}					
T_{21}	T_{22}				
T_{31}	T_{32}	T_{33}			
….	……	…	…	T_{k-1k-1}	
T_{k1}	T_{k2}	T_{k3}	…	T_{k-1k}	T_{kk}

В таблице (1) T_{ij} - есть приближенное решение задачи Коши, полученное численным методом порядка $p+(j-1)\cdot b$ с шагом h_i. Величина $T_{k,k}$ соответствует аппроксимации наивысшего порядка, равного $2k$, в случае, если вычислены первые k строк экстраполяционной таблицы, а величина $T_{k-1,k}$ соответствует аппроксимации порядка $2k-2$. Для управления шагом интегрирования естественно использовать выражение:

$$\left\| T_{k-1,k} - T_{k,k} \right\|.$$

Таким образом, экстраполяционная технология Ричардсона включает численный метод решения задачи Коши, последовательность сеток, рекуррентное правило вычисления значений приближенного решения. Эффективность применения технологии локальной экстраполяции напрямую зависит от правильного выбора и сочетания всех трех составляющих этого метода.

2. Анализ вычислительной сложности последовательных методов решения СЛОДУ с постоянными коэффициентами

Исследование эффективности предложенных альтернативных способов оценки локальной погрешности проводилось на следующем множестве методов одного и того же порядка p:

1 – явный метод Рунге-Кутты и правило Рунге;

2 – экспоненциальный метод и правило Рунге;

3 – вложенные методы Рунге-Кутты;

4 – вложенные методы на основе экспоненциального;

5 – метод Грэгга-Булирша-Штера;

6 – экспоненциальный метод с локальной экстраполяцией.

Экспоненциальный метод относится к специальным методам численного интегрирования линейных СОДУ с постоянными коэффициентами, основанными на точном представлении решения в аналитической форме и вычислении матричной экспоненты.

Точным решением задачи Коши вида:

$$\begin{cases} y' = A \cdot y \\ y(x_0) = y_0 \end{cases} \tag{6}$$

является матричная экспонента

$$y(x_0 + h) = e^{Ah} \cdot y_0,\ e^{Ah} = \sum_{k=0}^{\infty} \frac{(hA)^k}{k!}.$$

Приближенное решение (6) можно построить, аппроксимировав матричную экспоненту отрезком ряда Тейлора:

$$e^{Ah} \approx \sum_{k=0}^{p} \frac{(hA)^k}{k!}, \tag{7}$$

причем, (7) определяет численный метод решения (6) порядка $\boldsymbol{p}$ [6].

Наиболее эффективным из известных последовательным методом, реализующим технологию локальной полиномиальной экстраполяции считается алгоритм Грэгга-Булирша-Штера(ГБШ), базирующийся на модифицированном методе средней точки [3]. Сравнение перечисленных методов

Наиболее трудоемким из рассматриваемых алгоритмов является метод ГБШ, число арифметических операций для него имеет порядок $O(p^2 \cdot m^2)$, где $\boldsymbol{m}$ – размерность системы уравнений, а $\boldsymbol{p}$ – порядок метода. Наименее трудоемким является метод вложенных форм Рунге-Кутты: $O([2p+2] \cdot m^2)$ и вложенный метод на основе экспоненты: $O([2p+6] \cdot m^2)$.

3. Анализ эффективности параллельных алгоритмов решения СЛОДУ с постоянными коэффициентами с учетом локальной погрешности

Рассмотрим отображение алгоритмических схем приведенных методов на многопроцессорные вычислительные системы SIMD-структуры с распределенной памятью. Конфигурацию системы считаем фиксированной: число процессорных элементов и схема их соединения не изменяются в процессе счета. Каждый процессор может выполнить

любую арифметическую операцию за один такт, временные затраты, связанные с обращением к памяти отсутствуют. Параллельная реализация перечисленных методов требует распараллеливания следующих базовых операций: матричное и векторное умножение и сложение, а также умножение вектора и матрицы на скаляр. Наиболее оптимальное топологическое решение – это квадратная сетка $\boldsymbol{m \times m}$ или ее замкнутый эквивалент – тор. На такой топологической схеме достаточно эффективно выполняются матричные операции. Для простоты изложения рассмотрим случай, когда количество процессорных элементов в строке или столбце матрицы совпадает с размерностью задачи, ***m.*** Вычисление матричного умножения и умножения матрицы на вектор может быть выполнено по систолическому алгоритму, который является наиболее эффективным для SIMD-систем [7]. Процесс выполнения систолического умножения матриц состоит из предварительного косого сдвига левого сомножителя по строкам, влево и правого – вверх, по столбцам. Затем пошагово выполняются: ***m*** умножений элементов, ***(m-1) –*** одиночный сдвиг и ***(m-1)*** сложение:

$$T_{AxA}^{SYS} = mt_{ум} + (m-1)t_{сл} + 3(m-1)t_{сд}, \qquad (11)$$

где $\boldsymbol{t_{ум}, t_{сл}, t_{сд}}$ – времена выполнения одиночных операций умножения, сложения и сдвига. Вычисление систолического умножения матрицы на вектор на базе алгоритма сдваивания, подробно описано в [7] и требует следующего времени выполнения:

$$T_{AxY}^{SYS} = t_{ум} + (m + \overdash{m} - 2)t_{сд} + log_2 \overdash{m} t_{сл}, \qquad (12)$$

где $\overdash{m} = 2^l - 1$ и $l = \lceil \log_2 m \rceil$.

Вычислительные схемы параллельных алгоритмов с учетом отображения на структуру многопроцессорной ВС получены с помощью аппарата графов влияния[2]. В рамках статьи приведем результаты анализа эффективности использования потенциального параллелизма (без учета обменных операций) и реального (с учетом операций обмена между процессорами). Наименее трудоемкими оказались экспоненциальный метод с оценкой погрешности по правилу Рунге, для которого число арифметических операций имеет порядок: $\boldsymbol{O(3 \cdot Log_2 m)}$ и вложенный экспоненциальный метод: $\boldsymbol{O(2 \cdot Log_2 m)}$. Худшие показатели потенциального параллелизма у метода Рунге-Кутты с правилом Рунге и метода ГБШ.

Общепризнанным приемом повышения эффективности параллельных вычислений служит сокращение межпроцессорных обменов данными. Уменьшение времени выполнения обменных операций возможно, как за счет повышения пропускной способности линков, так и путем целенаправленного программирования решаемых задач, ориенти-

рованного на повышение отношения времени обработки к времени передачи данных в ВС. Эта задача тем более актуальна в связи с тем, что скорость обработки данных в микропроцессорах растет существенно быстрее увеличения пропускной способности интерфейсов микропроцессоров [2]. Оценки трудоемкости обменных операций для исследуемых алгоритмов приведены в таблице 1. Наибольшей коммуникационной сложностью обладает метод ГБШ, наименьшей – вложенный экспоненциальный и экспоненциальный на базе локальной экстраполяции.

Таблица 1. Характеристика алгоритмов по трудоемкости обменных операций

Параллельный алгоритм	Количество обменных операций
1. Метод Рунге-Кутты и правило Рунге	$(2 \cdot m - 2)(3 \cdot p - 1)$
2. Экспоненциальный метод и правило Рунге	$9(m-1)$
3. Вложенные методы Рунге-Кутты	$3(m-1)(p+1)$
4. Вложенный экспоненциальный метод	$6(m-1)$
5. Метод ГБШ	$3(m-1)(p^2+2p+4)/4$
6. Экспоненциальный метод с локальной экстраполяцией Ричардсона	$(m-1)(p-1)$

Анализ зависимости отношения времени обменов ко времени выполнения арифметических операций приведен на рис. 1.

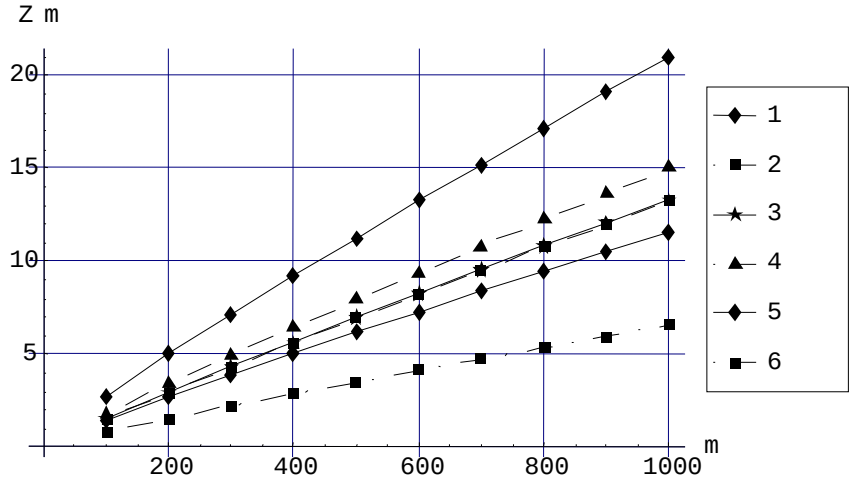


Рис. 1. График зависимости отношения обменных операций к арифметическим от размерности задачи

В целом характеристики реального параллелизма:

1) лучшие методы:

– 6-экспонента и локальная экстраполяция: $O(0.1m)$;

– 4- вложенная экспонента: $O(0.6m)$;

2) худший метод – метод ГБШ: $O(0.3 \cdot [p^2 + 2p + 4] \cdot m)$.

В качестве показателей эффективности параллельных алгоритмов применялись такие характеристики, как коэффициенты ускорения и эффективности. На рисунке 2 приведены коэффициенты ускорения для оценки реального параллелизма с учетом обмена, как функции размерности задачи.

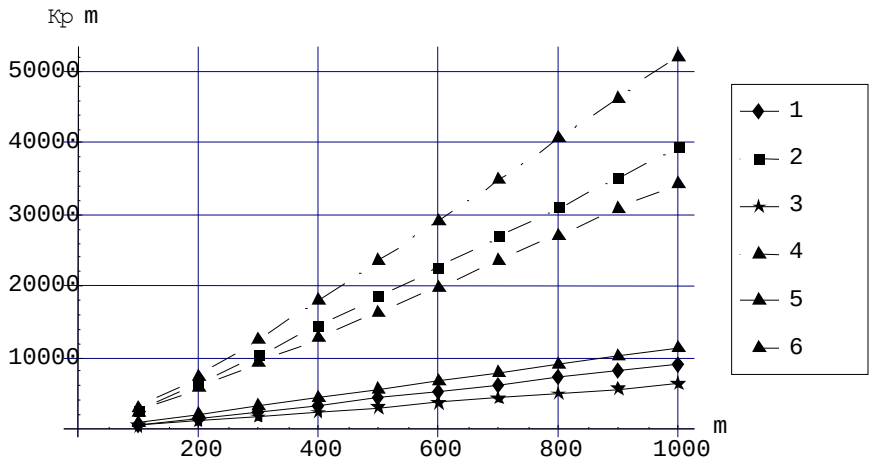


Рис. 2. Зависимость коэффициентов ускорения от порядка системы с учетом операций обмена

Определение характеристик параллелизма осуществлялось с помощью пакета *Mathematica*® (Wolfram Research Inc.), численный эксперимент проводился на базе тестов для СОДУ, разработанных в НИВЦ МГУ [8].

Анализ полученных результатов позволяет сделать следующие выводы:

– самым эффективным из рассмотренных последовательных методов является вложенный метод с контролем погрешности на шаге;

– лучшими параллельными методами с точки зрения трудоемкости являются вложенный экспоненциальный метод и экспоненциальный с локальной экстраполяцией;

– преимущества методов на базе локальной экстраполяции проявляются при получении высокоточных решений ($10^{-15} - 10^{-20}$) и для сложных правых частей.

Заключение

Анализ вычислительной сложности различных технологий определения апостериорной локальной погрешности при решении задачи Коши имел своей целью управление шагом интегрирования в достижение некоторой заданной точности с минимальными вычислительными затратами. Перспективным направлением дальнейших исследований

является оценка влияния различных топологий многопроцессорных ВС на характеристики качества параллельных алгоритмов, исследование устойчивости полученных методов.

Литература

1. *Rajkumar Buyya.* High Performance Cluster Computing. Volume 1: Architectures and Systems. Volume 2: Programming and Applications. Prentice Hall PTR, Prentice-Hall Inc., 1999.
2. *Воеводин В.В., Воеводин Вл.В.* Параллельные вычисления // СПб.: БХВ-Петербург, 2002. – 608с.
3. *Хайрер Э., Нёрсетт С., Ваннер Г.* Решение обыкновенных дифференциальных уравнений. Нежесткие задачи: Пер. с англ. – М.: Мир, 1990. – 512с.
4. *Bergman S., Rauber T., Runger G.* Parallel execution of embedded Runge-Kutta methods.International Journal of Supercomputer Applications, 1996. 10 (1). P. 62-90.
5. *Houwen P.J., Sommeijer B.P.* Parallel ODE solver // Proceedings of the International Conference on Supercomputing. – ACM Press, 1990. P. 71-81.
6. *Арушанян О.Б., Залеткин С.Ф.* Численное решение обыкновенных дифференциальных уравнений на Фортране // М.: МГУ, 1990. – 336 с.
7. *Бройнль Т.* Паралельне програмування: Початковий курс: Навч. посібник/ Вступ. слово А. Ройтера. Пер. з нім. В.А. Святного. – К.: Вища шк.. 1997. – 358 с.
8. *.Арушунян О.Б., Залеткин С.Ф., Калиткин Н.Н.* Тесты для вычислительного практикума по обыкновенным дифференциальным уравнениям // Вычислительные методы и программирование, 2002. Т. 3. С. 11-19.

ПАРАЛЛЕЛЬНОЕ ПРОГРАММИРОВАНИЕ КАК УЧЕБНАЯ ДИСЦИПЛИНА СПЕЦИАЛЬНОСТИ «ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ ВЫЧИСЛИТЕЛЬНОЙ ТЕХНИКИ И АВТОМАТИЗИРОВАННЫХ СИСТЕМ»

Н.П. Фефелов

Томский государственный университет систем управления и радиоэлектроники (ТУСУР), г. Томск

Изучение архитектуры современных супер-ЭВМ освоение принципов составления параллельных программ должно непременно входить в систему образования профессиональных программистов, чтобы сделать их мастерство конкурентоспособным на мировом рынке. По государственному стандарту высшего профессионального образования (ГОС) 1995 года предусмотрена учебная дисциплина «Параллельное программирование», основное назначение которой ознакомить с параллельными вычислительными системами, методами параллельной обра-

ботки информации, языками параллельного программирования, в том числе для транспьютерных систем.

На кафедре АСУ ТУСУРа преподавание этой дисциплины ведется по учебному плану в седьмом семестре с 1998 года в следующем объеме: лекций – 36 часов, практических занятий – 10 часов, лабораторных занятий – 16 часов, самостоятельная работа – 56 часов. Обеспечивал дисциплину доцент с базовым математическим образованием, работавший в 80-х годах на вычислительной системе ПС-2000 в Томском СКБ Геофизики.

В лекциях давались принципы построения многопроцессорных вычислительных систем и их классификация (по Флинну), рассматривались принципы конвейерной и матричной обработки данных, методы передачи данных и коммутации. Как пример разбиралась архитектура вычислительной системы ПС-2000. Основное внимание в лекциях уделялось параллельным алгоритмам: каскадным вычислениям выражений, векторным и матричным вычислениям, решению систем линейных уравнений. В заключительной части давался обзор языков параллельного программирования, основной упор делался на параллельные расширения языка С.

Практические и лабораторные занятия проводились на однопроцессорных персональных компьютерах. Они имели цель научить алгоритмам крупноблочного параллелизма матричных операций для задач линейной алгебры. Студент должен был распределить вычисления по узлам, а затем составить программу на алгоритмическом языке для одного узла. Передача данных между узлами не предусматривалась.

В 2000 году тематика дисциплины была перестроена с учетом современных достижений параллельных вычислений, требования ГОС 1995 к этому времени устарели. За основу были взяты материалы информационно-аналитического центра по параллельным вычислениям (МГУ) [1].

Основные темы лекций (ссылки даны на основные источники):

– принципы параллельной обработки и супер-ЭВМ [2];

– архитектура параллельных вычислительных систем и их классификация. Список TOP-500 [1];

– коммуникации и сети [1-3];

– производительность супер-ЭВМ и программ [4], тесты ее проверки [2];

– парадигмы и модели программирования [3];

– технологии параллельного программирования [2,3];

– системы программирования на основе передачи сообщений. Интерфейс MPI [5].

На практических занятиях разбираются графы параллельных алгоритмов. На примере обобщенной схемы вычисления полинома выявляется возможность проведения каскадных параллельных вычислений над массивами и их реализация на векторных процессорах. Рассматриваются также методы крупноблочного разбиения в параллельных матричных вычислениях. В заключение происходит знакомство с основными функциями MPI путем составления простых программ передачи сообщений.

Для проведения лабораторного практикума на кафедре создан учебный кластер из восьми узлов на основе персональных ЭВМ и локальной сети кафедры. На жестком диске каждого компьютера, входящего в кластер, выделен раздел, на котором размещается программное обеспечение (ОС Linux Red Hat 7, библиотека LAM и студенческие программы). В другое время компьютеры кластера могут использоваться в обычном режиме учебной работы (раздел диска для кластера тогда недоступен).

В первой лабораторной работе с помощью последовательных процедур моделируются векторные операции и на их основе составляются программы по алгоритмам каскадных вычислений, например численное интегрирование. Для освоения технологии MPI параллельную программу вычисления интеграла на языке Фортран предлагается переписать на языке С. Используются различные функции передачи данных. Далее студентам предлагается разобраться в программах, взятых из различных руководств по использованию MPI. Можно составить программу для матричных вычислений.

В ГОС 2000 дисциплина «Параллельное программирование» отсутствует, но в рабочем учебном плане кафедры АСУ она оставлена как дисциплина, установленная вузом. Меняется ее содержание. В число обязательных дисциплин по новому стандарту включены «Теория вычислительных процессов», где можно изложить парадигмы параллельного программирования, а также «Архитектура вычислительных систем», где предусмотрено изучение принципов построения супер-ЭВМ. В дисциплине «Параллельное программирование» решено основное внимание уделить не алгоритмам (это удел прикладных математиков), а технологиям параллельного программирования, в частности MPI как наиболее распространенной.

Кафедра сотрудничает с институтом оптики атмосферы СО РАН. Обеспечен удаленный доступ на кластер института (10 двухпроцессорных узлов) для учебно-исследовательской работы студентов. Кроме того, кафедра АСУ имеет небольшой опыт сотрудничества с японским центром морских наук и технологий (Иокогама), где установлен самый мощный суперкомпьютер Earth-Simulator. Надеемся на продолжение сотрудничества.

Литература

1. http://www.parallel.ru – Информационно-аналитический центр по параллельным вычислениям в сети Интернет.

2. *Воеводин В.В., Воеводин Вл.В.* Параллельные вычисления // СПБ.: БХВ, 2002.

3. *Немнюгин С.А., Стесик О.Л.* Параллельное программирование для многопроцессорных вычислительных систем // СПБ.: БХВ, 2002.

4. *Хокни Р., Джесссхоуп К.* Параллельные ЭВМ // М.: Радио и связь, 1986.

5. *Шпаковский Г.И., Серикова Н.В.* Программирование для многопроцессорных систем в стандарте MPI // Минск: Изд-во БГУ, 2002.

ПРОГРАММНО-АППАРАТНЫЙ КОМПЛЕКС ДЛЯ РЕШЕНИЯ ПРИКЛАДНЫХ ЗАДАЧ МЕХАНИКИ

С.К. Черников, А.Н. Ашихмин

Казанский физико-технический институт им. Завойского, г. Казань

Для многих образовательных и научно-исследовательских организаций России, у которых имеется потребность в мощных вычислительных ресурсах, кластеры, созданные из общедоступных компьютеров на базе процессоров AMD и недорогих Ethernet-сетей, являются естественной альтернативой недоступным из за высокой стоимости суперкомпьютерам. В то же время эффективность используемых алгоритмов в значительной степени будет определяться быстродействием коммуникационного оборудования, обеспечивающего среду передачи сообщений между вычислительными узлами. При создании программ ориентированных на использование на вычислительных кластерах необходимо специальным образом распределять данные между процессорами, чтобы минимизировать число пересылок и объем пересылаемых данных. В этой связи представляется целесообразным проектирование вычислительного кластера под определенный класс задач одновременно с разработкой программного продукта, учитывающего как особенности задач класса, так и конкретную архитектуру кластера. Результатам создания такого программно-аппаратного комплекса для решения задач механики деформирования сложных машиностроительных конструкций методом конечных элементов (МКЭ) и посвящен настоящий доклад.

1. Конструкция и технические характеристики кластера АРКО-9А2200

Кластер АРКО-9А2200 представляет собой 9-процессорный вычислительный комплекс, построенный на базе процессоров AMD ATHLON. Комплекс включает в себя:

- 2-х процессорный системный блок на базе системной платы TYAN Tiger с двумя процессорами ATHLON MP 2200+ и сетевым адаптером Gigabit Ethernet NARDLINK HA-64G, установленным на шине PCI64;
- 7 1-процессорных системных блоков на базе системной платы MICROSTAR MS-6570 (nForce 2 chipset) с процессором ATHLON XP 2200+ и сетевым адаптером Gigabit Ethernet NARDLINK HA-32G, установленным на шине PCI32;
- 8-ми портовый Gigabit Ethernet-коммутатор NARDLINK HS-8G.
- 16-ти портовый переключатель консоли EDIMAX EK-16RO, с возможностью управления с консоли
- Источник бесперебойного питания APC Smart-UPS 3000.
- Пульт управления (Монитор, клавиатура, манипулятор «мышь»)

Все элементы кластера смонтированы в открытой стойке. Кластер работает под управлением операционной системы Windows 2000. В качестве коммуникационной среды используется библиотека MPICH 1.2.5, свободно распространяемая Техническим университетом г. Аахен, Германия. Внешний вид кластера показан на рис. 1.

Рис. 1. Внешний вид кластера «Арко-9А2200»

Для определения вычислительных возможностей кластера и оптимизации его структуры был проведен ряд исследований, в ходе которых оценивались: производительность отдельного вычислительного узла, производительность сетевого оборудования при различных способах коммутации узлов и интегральная производительность кластера при одновременном использовании нескольких узлов.

Производительность отдельного узла кластера (число элементарных арифметических операции с плавающей точкой в секунду - гигаф-

лоп) определялась при помощи модуля Linpack в зависимости от размерности решаемой задачи, точности вычислений и степени оптимизации вычислений под тип центрального процессора. Под точностью вычислений в данном случае понималось использование четырехбайтного или восьмибайтного представления числа с плавающей точкой. Оптимизация вычислений под тип процессора представляла собой использование библиотеки подпрограмм базовых операций линейной алгебры (BLAS), использующей дополнительный набор команд процессора (SSE), предназначенный для векторных арифметических операций. Отметим, что процессор Athlon-XP имеет расширенный набор команд только для обработки чисел с плавающей точкой в четырехбайтовом представлении.

На рис. 2 представлены результаты тестирования узла кластера описанными выше модулями в зависимости от размерности задачи.

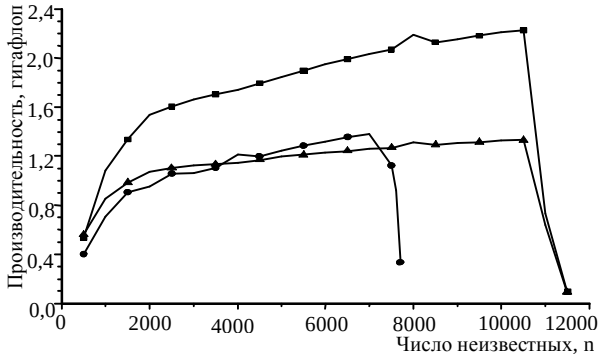


Рис. 2. Производительность узла кластера в зависимости от размерности задачи и точности вычислений (●– REAL(8), ▲– REAL(4), ■ – REAL(4) SSE)

В процессе тестирования сетевого оборудования производилось определение скорости передачи информации (Мб/с) в режиме обмена УЗЕЛ-УЗЕЛ и в режиме коллективного обмена. При тестировании в режиме обмена УЗЕЛ-УЗЕЛ производилось сравнение производительности двух различных сетевых адаптеров, которые предполагалось использовать на рядовых узлах - Gigabit Ethernet 3Com 3C996B-T и Gigabit Ethernet NARDLINK HA-32G. Для двухпроцессорного узла был выбран сетевой адаптер Gigabit Ethernet NARDLINK HA-64G. Тестирование сетевых адаптеров производилось при подключении типа УЗЕЛ-УЗЕЛ (использовалось непосредственное соединение сетевых адаптеров двух узлов сетевым кабелем категории 5e длиной 1,5 м). В качестве инструмента использовалась утилита Netpipe. Оценивалась латентность и скорость передачи данных. Латентность оценивалась на соединении двух однотипных адаптеров временем передачи пакетов минимальной длины и составила: 3C996B-T – 64, HA-64G – 104, HA-32G – 170 микросекунд. Скорость передачи оценивалась в зависимости от величины передаваемого пакета данных для четырех комбинаций пар: 3C996B-T – 3C996B-

T, HA-32G – HA-32G, HA-64G – 3C996B-T и HA-64G – HA-32G. На рис. 3 приведена зависимость скорости передачи величины передаваемого пакета данных для указанных пар адаптеров.

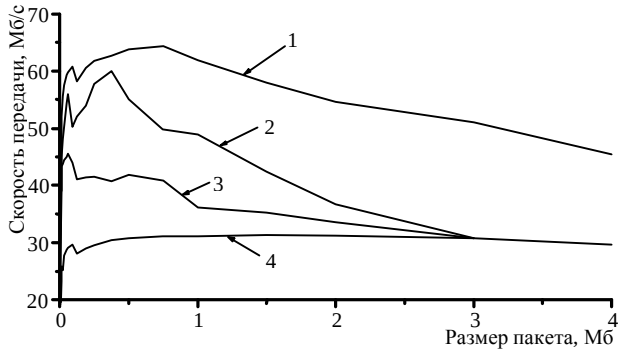


Рис.3. Скорость передачи данных в зависимости от размера пакета (1) 3C996B-T – 3C996B-T, 2) HA-64G – 3C996B-T, 3) HA-64G – HA-32G и 4) HA-32G – HA-32G

Ниже в таблице приводится сравнение скорости обмена при величине пакета 1 КБ кластера АРКО-9А2200 и некоторых кластеров, производительность которых была оценена утилитой Transfer, а полученная информация опубликована в сети Internet.

Кластер	**НИВЦ (г. Нижний новгород)**		**КазНЦ (г. Казань)**	**АРКО-9А2200**
Тип оборудования	SCI	Fast Ethernet	Fast Ethernet	Gigabit Ethernet
Скорость передачи данных	26,08	3,146	3,1370	9,421

В режиме коллективного обмена тестирование сетевого оборудования кластера производилось при соединении через коммутатор всех восьми узлов. Определение производительности сетевого оборудования проводилось для нескольких вариантов коммутации узлов: звезда, кольцо, хаос.

Этот тест производился при помощи модуля Nettest. На рис. 4 приведены зависимости пропускной способности сетевого оборудования от величины передаваемого пакета данных.

Интегральная производительность кластера определялась при решении системы уравнений утилитой Linpack пакета PLAPACK в зависимости от размерности решаемой задачи, точности вычислений и степени оптимизации вычислений под тип используемого процессора. На рис. 5 приведены графики этих зависимостей, полученные в ходе тестирования.

Максимальная производительность кластера достигается при использовании всей доступной оперативной памяти и составляет 15.05 гигафлоп для четырехбайтовой точности решения и 8.11 гигафлоп для

восьмибайтовой точности. Пиковая производительность кластера (сумма производительностей всех узлов кластера) при использовании для вычислений 8 узлов составит 17,84 гигафлоп для четырехбайтовой точности решения и 11.04 гигафлоп для восьмибайтовой.

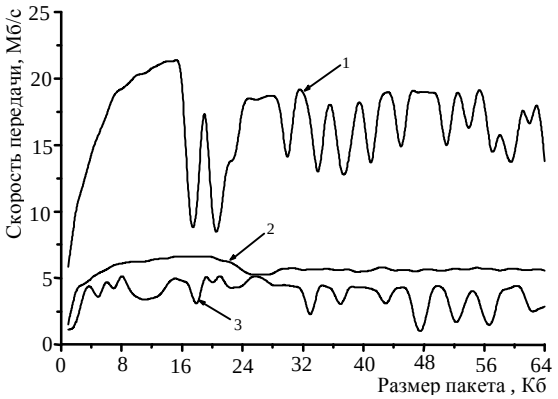


Рис. 4. Скорость передачи данных в зависимости от размера пакета и схемы коммутации узлов кластера (1-кольцо; 2-звезда; 3-хаос)

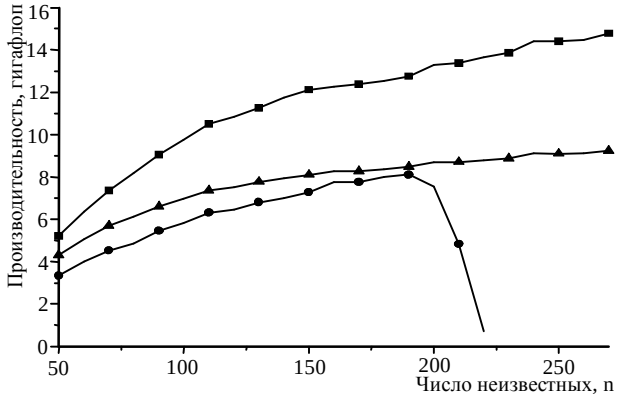


Рис. 5. Производительность кластера в зависимости от размерности задачи и точности вычислений (●– REAL(8),▲– REAL(4), ■– REAL(4) SSE)

2. Некоторые особенности программной реализации

Наиболее трудоемкой процедурой при решении решения задач механики деформирования сложных машиностроительных конструкций МКЭ является решение системы линейных уравнений большой размерности (1000000 и более неизвестных), которое в той или иной форме присутствует в большинстве алгоритмов. На рис. 7 показано время необходимое для определения перемещений в конструкции приведенной на рисунке 6 при использовании различного количества узлов кластера.

Система линейных уравнений при этом содержала 16050 неизвестных при ширине полуленты 594 и решалась методом Гаусса с распараллеливанием. Легко заметить, что уже при 5-6 процессах наступает насыщение, и дальнейшее увеличение процессов становится неэффективным.

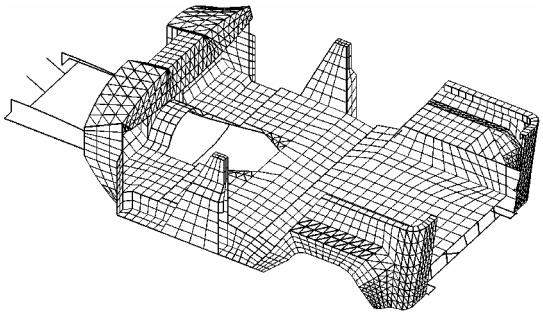


Рис. 6. Расчетная схема конструкции

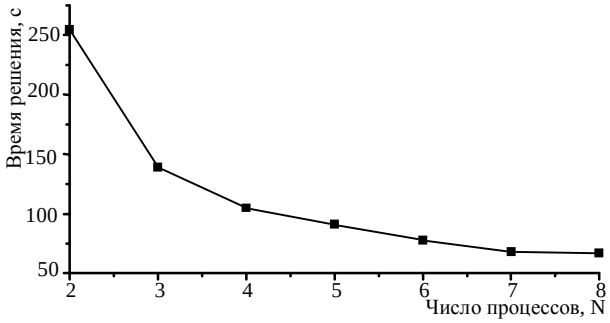


Рис. 7. Время решения в зависимости от различного количества используемых узлов

Главной причиной этого является сравнительно низкая скорость передачи данных по сети и высокая латентность используемого коммуникационного оборудования. Улучшение этих характеристик оборудования привело бы к резкому увеличению стоимости вычислительной системы в целом и не решило бы задачу в принципе. По мнению авторов более перспективным является использование методов и алгоритмов распараллеливания не предъявляющих высоких требований к коммуникационной среде.

Предполагая, что сложная конструкция может быть расчленена на несколько подконструкций, связанных между собой, рассмотрим некоторые схемы распараллеливания вычислений использующих эту технику. В её основе лежит выделение из основных неизвестных *j*-й подконструкции $\boldsymbol{q}_j$ «внешних» неизвестных $\tilde{\boldsymbol{q}}_j$, связанных с узлами сетки, которые стыкуются с другими подконструкциями. Суть методов проиллюстрируем на примере линейной задачи статики хотя аналогичные приемы применимы и к другим задачам механики [1]. При использовании метода перемещений решение линейной задачи статики сводится к отысканию из решения системы алгебраических уравнений:

$$Kq = P, \qquad (1)$$

вектора обобщенных перемещений $\boldsymbol{q}$ и вычисления вторичных неизвестных (напряжений, интенсивностей и т.п.), определяемых этим вектором. При использовании для объединения подконструкций метода штрафных функций, уравнение (1) может быть записано в виде

$$\left[\sum K^i + \sum k\right]q = \sum P^i \qquad (2)$$

В этом выражении k – матрица жесткости «элемента стыковки» подконструкций, которая для каждой пары стыкуемых узлов вычисляется по соотношению $[k]=\alpha\begin{bmatrix} e & -e \\ -e & e \end{bmatrix} = k^D - k^L$.

Вначале рассмотрим алгоритм распараллеливания без конденсации матриц жесткости подконструкций. Полагая $q^T = \{q_1^T q_2^T ....q_m^T\}$, $\tilde{q}^T = \{\tilde{q}_1^T \tilde{q}_2^T ....\tilde{q}_m^T\}$, система (2) может быть заменена на $\boldsymbol{m}$ независимых систем:

$$\left[K^j + \sum_j k^D\right]q_j = P^j - \sum_j k^L \tilde{q}\ , j=1,2...m,$$

которые могут быть решены итерационно. Таким образом, процедура отыскания решения системы (1) сведется к следующей последовательности действий.

1.В отдельном вычислительном процессе для каждой подконструкции формируются матрицы $\hat{K} = \left[K^i + \sum_j k^D\right]$ и $\boldsymbol{P^i}$. После чего матрица $\hat{K}$ факторизуется.

2.В главном процессе формируется начальное приближение вектора $\tilde{\boldsymbol{q}}$, которое рассылается всем процессам.

3. В отдельном вычислительном процессе для каждой подконструкции из системы $\hat{K}^j q_j^{n+1} = P^i - \sum_j k^l \tilde{q}^n$ находится n+1 приближение вектора $\boldsymbol{q_j}$. Из вектора $\boldsymbol{q_j^{n+1}}$ выделяется вектор $\tilde{\boldsymbol{q}}^{\boldsymbol{n+1}}$, который посылается главному процессу.

Вычисления повторяются, начиная с позиции 2 до тех пор, пока не будет достигнута необходимая величина погрешности. После этого в отдельном для каждой подконструкции процессе вычисляются вторичные результанты (напряжения, интенсивности и т.п.). Отметим, что в описанном алгоритме объем информации, передаваемой между вычислительными процессами, определяется только количеством внешних неизвестных.

Схема алгоритма распараллеливания решения с конденсацией, представляет собой следующую последовательность действий.

В отдельном вычислительном процессе для каждой подконструкции формируются матрицы $\boldsymbol{K}$ и $\boldsymbol{P}$.

Для построения коэффициентов конденсированных матриц подкострукции, имеющей n «внешних» степеней свободы, однократно решается система уравнений $[K+\alpha E][Q^1 Q^p] = [\alpha\tilde{E} \mid P]$ с n+1 правой частью. Здесь: $E = diag[1 \quad 0 \quad 1 \quad .. \quad 1]$ - диагональная матрица, в которой «едини-

цы» стоят на местах, соответствующих внешним степеням свободы, $\tilde{\boldsymbol{E}}$ представляет собой матрицу $\boldsymbol{E}$ с вычеркнутыми нулевыми столбцами, $\boldsymbol{Q}^1, \boldsymbol{Q}^p$-матрицы перемещений от единичных загружений и внешней нагрузки соответственно. Коэффициенты $\tilde{\boldsymbol{k}}_{ij}$ и $\tilde{\boldsymbol{p}}_i$ конденсированных матриц подкострукции вычисляются по соотношениям:

$$\tilde{k}_{ij} = \begin{cases} -\alpha\left(Q^1_{ind(i)j} - 1\right) \text{если } i = j \\ -\alpha Q^1_{ind(i)j} \text{ если } i \neq j \end{cases}, \quad \tilde{p}_i = -\alpha Q^p_i,$$

в которых ind - матрица индексов, содержащая глобальные номера внешних неизвестных.

Конденсированные матрицы $\tilde{K}$ и $\tilde{P}$ передаются ведущему процессу для формирования разрешающих уравнений структуры для «внешних» неизвестных $\left[\sum \tilde{K}^i + \sum k\right]\tilde{q}_s = \sum \tilde{P}^i$.

Далее в ведущем процессе определяется вектор «внешних» неизвестных структуры $\tilde{\boldsymbol{q}}$, и его соответствующие компоненты $\tilde{\boldsymbol{q}}_i$ передаются процессам, обрабатывающим информацию о подконструкциях.

Для каждой подконструкции в изолированных процессах вычисляются основные неизвестные в узлах $\{q\} = Q^1\{q_s\} + Q^p$ и вторичные результанты (напряжения, интенсивности и т.п.)

Как и в алгоритме без конденсации, в описанном алгоритме объем информации, передаваемой между вычислительными процессами, определяется только количеством внешних неизвестных.

Литература

1. *Черников С.К.* Метод подконструкций – эффективный инструмент распараллеливания алгоритмов в механике // Труды второго международного научно-практического семинара «Высокопроизводительные параллельные вычисления на кластерных системах», Н. Новгород: Издательство НГУ. 2002. С. 318-326.

ИСПОЛЬЗОВАНИЕ КЛАСТЕРА ДЛЯ ОБУЧЕНИЯ МНОГОСЛОЙНОГО ПЕРСЕПТРОНА

В.А. Шустов

Самарский государственный аэрокосмический университет

Введение

Многослойные нейронные сети получили широкое распространение при решении задач классификации. Вопрос выбора архитектуры нейронной сети (количества слоев и нейронов в слоях) пока, к сожалению, слабо формализован. Нейронные сети, используемые для решения задач распознавания, имеют важную особенность. Решение о распознанном классе прини-

мается за короткое время. В тоже время обучение нейронной сети (подбор ее параметров) – продолжительный процесс, требующий длительных временных затрат. Для получения оптимальной в некотором смысле нейронной сети в общем случае требуется неоднократное обучение сетей с различной архитектурой и различными наборами данных [1]. Таким образом, сокращение времени обучения является актуальной задачей.

В настоящей работе рассматривается использование многопроцессорной вычислительной техники – кластера – для сокращения времени обучения. Нейронная сеть обучается распознаванию изображений печатных, рукописных и стилизованных цифр.

Алгоритмы обучения

Последовательные алгоритмы обучения, подвергнутые распараллеливанию, подробно изложены в работе [2]. Эти алгоритмы предназначены для обучения всем примерам из обучающего набора данных и используют равномерный критерий качества обучения. Идея такого обучения заключается в следующем. На каждой эпохе обучения множество обучающих примеров делится на две части: примеры, для которых нейронная сеть выдает допустимый ответ, и примеры, для которых ответ не является допустимым. Корректировка параметров нейронной сети происходит в соответствии с обратным распространением ошибки только с использованием недопустимых примеров. В алгоритме поэтапного обучения (ПО) обучающее множество делится исходя из заданного порога на значения выходов нейронов последнего слоя. В алгоритме обучения на худших примерах (ОХП) обучающее множество делится исходя из заданного количества примеров, которые должны попасть в группу недопустимых.

Распараллеливание алгоритмов [3] произведено за счет одновременного поиска недопустимых примеров, который заключается в выполнении прямого прохода нейронной сети для каждого примера из обучающего множества. Здесь осуществляется декомпозиция данных – разбиение обучающего множества по задачам. В результате прямых проходов формируется множество примеров, по которым должна производиться корректировка параметров нейронной сети на текущей эпохе обучения в рамках модели задача/канал [4] получены оценки сверху ускорения a параллельных алгоритмов [3]:

$$a_{\text{по}} < \overline{a}_{\text{по}} = \frac{1+1{,}5\gamma}{1/k+2\gamma}; \quad a_{\text{охп}} < \overline{a}_{\text{охп}} = \frac{1+4\gamma}{1/k+4\gamma},$$

где k – количество задач; γ – отношение количества использования обучающих примеров для корректировки параметров нейронной сети к максимально возможному.

Величина γ определяется как отношение количества недопустимых примеров за все время обучения к количеству прямых проходов нейронной сети при поиске недопустимых примеров:

$$\gamma = \frac{\sum_i r_i}{\sum_i Q} = \frac{\sum r_i}{nQ};$$

здесь r_i – число недопустимых примеров на i-й эпохе обучения, Q – число обучающих примеров, n – число эпох обучения. Графики зависимостей оценок сверху ускорения алгоритмов от величин k и γ приведены на рис. 1 (слева – для $\overline{a}_{\text{ПО}}$, справа – для $\overline{a}_{\text{ОХП}}$).

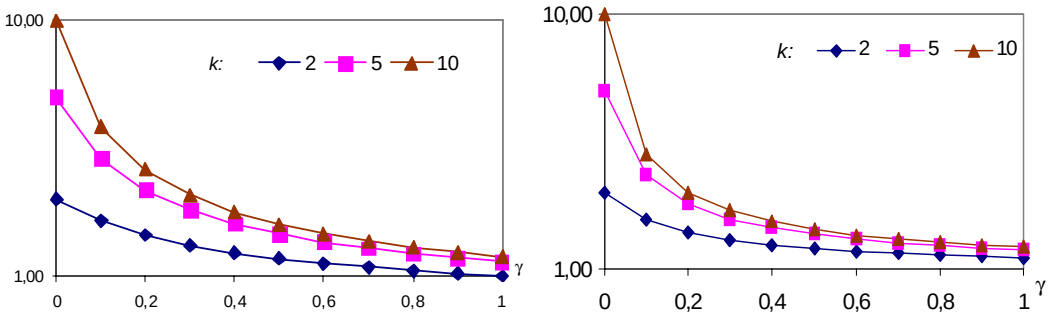


Рис. 1. Оценки сверху ускорения параллельных алгоритмов

Видно, что максимальное достижимое ускорение в значительной степени зависит от параметра γ, причем с увеличением числа задач k зависимость усиливается. Следовательно, ускорение параллельных алгоритмов при большом числе задач определяется величиной, которая не зависит от способа распараллеливания. При увеличении обоих параметров расхождение оценок с ускорением увеличивается.

Результаты экспериментов

Экспериментальные исследования параллельных алгоритмов проводились на кластере Самарского научного центра РАН. Использовались три двухпроцессорных узла P-III 600MHz с сетью Ethernet 100Mbit. Таким образом, максимальное число процессов, для которых получены результаты, равно шести. Нейронная сеть обучалась распознаванию изображений рукописных и стилизованных цифр различного характера написания. Общее число изображений – 10000.

Первый эксперимент заключался в определении величины γ при обучении нейронной сети. Этот процесс носит случайный характер из-за неопределенности в выборе начальных значений настраиваемых параметров. Поэтому в качестве результата использовалось среднее значение, полученное при проведении серии экспериментов. Зависимость величины γ от количества обучающих примеров Q для алгоритма поэтапного обучения приведена на рис. 2. Видно, что увеличение объема

обучающей выборки приводит к уменьшению γ. Это позволяет сделать предположение о том, что распараллеливание обучения нейронной сети будет особенно эффективным при большом числе обучающих примеров. При использовании алгоритма ОХП есть возможность задавать количество примеров, которые будут недопустимыми на каждой эпохе обучения. В связи с этим значение γ для алгоритма ОХП определялось исходя из условий, при которых время обучения минимально.

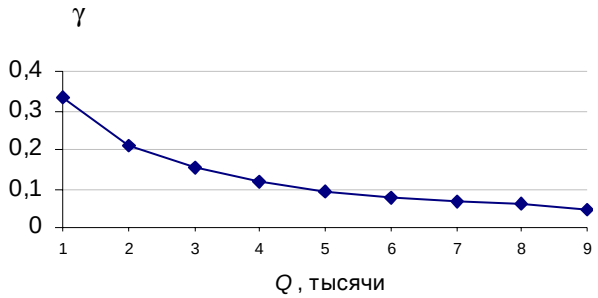
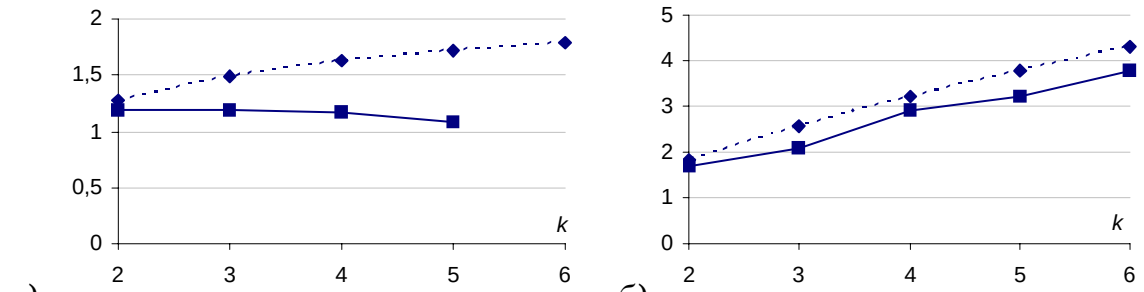


Рис. 2. Зависимость параметра обучения γ от количества обучающих примеров

Следующие эксперименты преследуют цель определения ускорения выполнения параллельных вычислительных процессов (ПВП), порожденных указанными алгоритмами. Для этого нейронная сеть обучалась одной и десяти тысячам изображений. Результаты экспериментов и соответствующие оценки ускорений приведены на рис. 3 и 4. Сплошной линией изображены графики ускорений ПВП, пунктирной – оценка ускорения при соответствующем значении γ. В случае применения алгоритма ОХП (рис. 4) для обоих обучающих множеств γ=0,002.



а) *б)*

Рис. 3. Ускорения ПВП при обучении а)1000 и б)10000 изображениям по алгоритму ПО

Из рисунков видно, что при обучении малому количеству изображений по алгоритму ОХП дает большее ускорение, чем по алгоритму ПО. При обучении большому количеству изображений большее ускорение показал ПВП, порожденный алгоритмом ПО. С ростом числа задач

разница между оценкой ускорения алгоритма и ускорением обучения на кластере увеличивается.

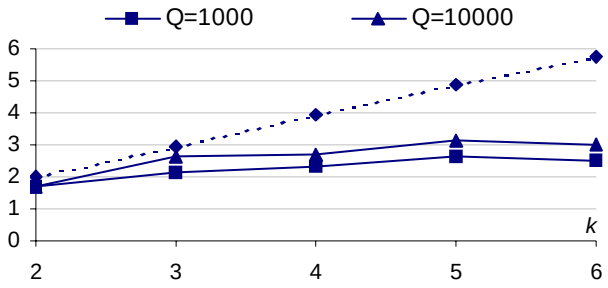


Рис. 4. Ускорения ПВП при обучении по алгоритму ОХП

Заключение

Проведены эксперименты по обучению нейронной сети с использованием высокопроизводительной вычислительной техники. Результаты экспериментов согласуются с предположениями о характере обучения рассматриваемыми параллельными алгоритмами на основании аналитических зависимостей. Выявлены предпочтительные области использования алгоритмов. Ускорение обучения нейронной сети на кластере возрастает при увеличении количества обучающих данных.

Литература

1. *Круглов В.В., Борисов В.В.* Искусственные нейронные сети. Теория и практика. // М.: Горячая линия – Телеком, 2001. - 328 с.

2. *Шустов В.А*. Алгоритмы обучения нейронных сетей распознаванию изображений по равномерному критерию // Компьютерная оптика. 2003. № 25. С.183-189.

3. *Шустов В.*А. Параллельные алгоритмы обучения нейронной сети, использующие равномерный критерий качества обучения // Тр. Всероссийской конференции ММ-2004. Секция 2. Самара, СамГТУ, 27-28 мая 2004.

4. *Корнеев В.В.* Параллельные вычислительные системы // «Нолидж», М.-1999. - 312 с.

СОДЕРЖАНИЕ