РОССИЙСКАЯ АКАДЕМИЯ НАУК
ВЫЧИСЛИТЕЛЬНЫЙ ЦЕНТР ИМ. А.А. ДОРОДНИЦЫНА

СООБЩЕНИЯ ПО ПРИКЛАДНОЙ МАТЕМАТИКЕ

Н.Н. ОЛЕНЕВ

ОСНОВЫ ПАРАЛЛЕЛЬНОГО ПРОГРАММИРОВАНИЯ В СИСТЕМЕ MPI

ВЫЧИСЛИТЕЛЬНЫЙ ЦЕНТР ИМ. А.А.ДОРОДНИЦЫНА
РОССИЙСКОЙ АКАДЕМИИ НАУК
МОСКВА 2005

УДК 681.3.06+519.86

Ответственный редактор
действительный член РАН А.А. Петров

Здесь представлены основы параллельного программирования в интерфейсе передачи сообщений. Изложение дано в виде лабораторной работы № 1 из курса практических занятий "Параллельное программирование в MPI" для студентов базовой кафедры МФТИ в ВЦ РАН. Первая часть работы содержит теоретические сведения, вторая - контрольные вопросы, третья – упражнения. Последнее упражнение представляет собой домашнее задание и состоит в разработке параллельной версии программы, идентифицирующей параметры производственного блока модели экономики России. Предполагается знание основ программирования на языке С и знание операционных систем Linux и Windows NT на уровне пользователя.

Basics of MPI Parallel Programming

Basics of message passing interface are presented here. This is lab no. 1 of labs' collection from the course "Parallel programming in MPI" intended for 5-year students of MIPT. The first part of the lab is a theory, the second part is a quiz and the last part contains exercises. The last exercise is a homework that assumes building of parallel version of program that identifies parameters of a model of Russian transitional economy. The prerequisites cover basics of C, Linux, and Windows NT.

Работа выполнена при финансовой поддержке Российского фонда фундаментальных исследований (код проекта 04-07-90346, 04-01-00606); при поддержке программы фундаментальных исследований ОМН РАН №3 «Вычислительные и информационные проблемы решения больших задач»; по программе государственной поддержки ведущих научных школ (код проекта НШ -1843.2003.1); при поддержке программы фундаментальных исследований РАН № 16 «Математическое моделирование и интеллектуальные системы».

Рецензенты: Ю.Н. Павловский,
И.И. Поспелова

Научное издание

© Вычислительный центр им. А.А. Дородницына
Российской академии наук, 2005

Предисловие автора

Книга рассчитана на начинающего пользователя Message Passing Interface - интерфейса передачи сообщений MPI - студента или специалиста, не знакомого с методами программирования для параллельных ЭВМ, однако владеющего последовательным программированием на языке программирования *C* в среде Windows или Linux. Далее в предисловии указано место данной работы в годовом курсе «Параллельное программирование в интерфейсе MPI», который автор излагает студентам 5 курса Московского физико-технического института, обучающимся на базовой кафедре «Математическое моделирование сложных процессов и систем» в Вычислительном центре имени А.А. Дородницына Российской академии наук (ВЦ РАН) и даны рекомендации студентам.

В настоящее время имеется рабочий вариант полной электронной версии курса (http://www.ccas.ru/mmes/educat/lab04k/), который непрерывно совершенствуется, а некоторые из работ появляются в открытом доступе.

Зачем нужна еще одна книга по MPI? Дело в том, что на русском языке пока нет настолько же ясного и в то же время достаточно полного изложения MPI, которые даны в западных учебных курсах, например, в курсе Cornell Theory Center (CTC) [1]. Кроме того, данный курс MPI специально предназначен для специалистов в математическом моделировании сложных процессов и систем. Поэтому к стандартным упражнениям CTC здесь добавлены упражнения, связанные с применением параллельных вычислений в математическом моделировании.

В настоящей книге представлена первая лабораторная работа из годового курса MPI. Данная работа открывает первую часть курса, в основе которой лежит курс CTC «MPI для начинающих». Начиная с момента основания курса в 2003/2004 учебном году, он эволюционно совершенствуется на основе работ ВЦ РАН по мере получения новых результатов в области параллельных вычислений. В частности, домашнее задание лабораторной работы № 1 включает составление параллельной

версии программы идентификации параметров блока «Производство» в модели экономики России переходного периода.

Изложение курса MPI в виде цикла лабораторных работ идеально подходит для студентов, обучающихся по системе Физтеха, в которой свободное посещение занятий сочетается с обязательным выполнением заданий для самостоятельной работы и сдачей заданий к назначенному сроку. Кроме того, электронную версию курса можно будет использовать в системе открытого дистанционного образования. Необходимые предварительные требования к подготовке студентов включают знание основ программирования на языке C или C++ , а также знание операционных систем Linux и Windows NT на уровне пользователя.

Структура изложения лабораторных работ в курсе совпадает со структурой первой работы, которая представляет собой взаимосвязанный модуль, состоящий из трех частей. В первой части лабораторной работы излагаются необходимые для выполнения заданий теоретические сведения, во второй части приведены контрольные вопросы, проверяющие усвоение теоретического материала, а в третьей, заключительной части, предложены упражнения. Часть упражнений предлагается выполнить в качестве домашнего задания. Предполагается последовательное освоение каждого модуля: вначале следует изучить теорию, затем с помощью контрольных вопросов проверить ваше понимание и, наконец, выполнить практические задания - упражнения. Студентам МФТИ в осеннем семестре требуется изучить первые восемь лабораторных работ, выполнив и сдав практические задания, помещенные в заключительной части каждого модуля. На изучение каждого модуля, как правило, отводится две недели. До конца второй недели следует по электронной почте выслать преподавателю решение домашних заданий лабораторной работы и сдать в очном режиме классные задания. При этом решение заданий по каждой лабораторной работе следует отправить отдельным электронным письмом, а в начале темы письма следует указать номер группы. Крайний срок отправки решений, после которого могут возникнуть трудности с получением зачета в срок: начало девятой, восьмой, седьмой, шестой, пятой, четвертой, третьей и второй недели до Нового года соответственно для 1-8 модуля.

Студентам МФТИ в весеннем семестре требуется сдать шесть лабораторных работ и одну индивидуальную курсовую работу. Для выполнения индивидуальной курсовой работы студенту в начале семестра необходимо получить у своего научного руководителя тему работы, которая могла бы быть использована как часть магистерской диссертации. Требования к теме: должно существовать или быть близко к завершению решение задачи в виде последовательной версии программы на языке C или C++. При отсутствии задачи на распараллеливание у научного руководителя можно взять тему курсовой работы у преподавателя. Крайний срок отправки решений, после которого будут трудности с получением в срок дифференцированного зачета: начало восьмой, седьмой, шестой, пятой, четвертой, третьей и второй недели до 1 июня, соответственно, для 9-14 модуля и курсовой работы.

Распечатки ответов на контрольные вопросы по всем модулям пригодятся при получении дифференцированного зачета. Итоговая оценка складывается из трех частей: 30% - оценка за осенний семестр, 30% - оценка за весенний семестр и 40% - оценка за курсовую работу.

Выполнение лабораторной работы № 1 «Основы программирования в MPI», на изучение и сдачу которой студентам предоставлено четыре недели, дает допуск к выполнению всех последующих работ.

Изложение курса MPI ограничено использованием языка C. Пользователи Фортрана могут изучить его самостоятельно, основываясь на пособиях [2-5] или на знании MPI для C/C++.

1. Введение: обзор MPI

1.1. Что такое MPI?

MPI - это библиотека передачи сообщений, собрание функций на C/C++ (или подпрограмм в Фортране), облегчающих коммуникацию (обмен данными и синхронизацию задач) между процессами параллельной программы с распределенной памятью. Акроним MPI установлен для Message Passing Interface (интерфейс передачи сообщений). MPI на данный момент является

фактическим стандартом и самой развитой переносимой библиотекой параллельного программирования с передачей сообщений.

Формально MPI не является стандартом, подобным тем, что выпускают организации стандартизации, такие как Госкомстандарт РФ, ANSI или ISO. Он является "стандартом по консенсусу", спроектированным на открытом форуме. В форуме принимали участие крупные поставщики компьютеров, исследователи, разработчики библиотек программ и пользователи, представляющие более 40 известных организаций. Широкое участие в развитии MPI гарантировало его быстрое превращение в широко используемый стандарт для написания параллельных программ передачи сообщений.

"Стандарт" MPI был введен MPI-форумом в мае 1994 и обновлен в июне 1995. Определяющий его документ назван "MPI: A Message-Passing Standard". Он опубликован университетом Тэннеси и доступен по World Wide Web в Argonne National Lab (http://www.mpi-forum.org/docs/). Тем, кто впервые знакомится с MPI, будет полезно распечатать русский перевод этого документа (http://www.cluster.bsu.by/MPI_ALL.htm). Этот документ можно использовать для получения справок о синтаксисе функций MPI, которые в данном курсе не могут быть охвачены полностью за исключением нескольких частных случаев.

Для распараллеливания ваших программ нет необходимости знать все функции MPI. На практике для ваших целей в различных приложениях будет достаточно использовать около дюжины функций из трех десятков наиболее распространенных. Поэтому, поняв из настоящей работы, как данные перемещаются между процессами в MPI-окружении, вам, возможно, покажется легче написать программу, используя MPI, чем TCP/IP.

Вторая версия (MPI-2) была разработана форумом в 1997 г. для улучшения стандарта MPI передачи сообщений. Эти усилия не изменили MPI; они расширили его применение на новые сферы:

- управление динамическими процессами,
- ассиметричные операции,
- параллельный ввод/вывод (I/O),
- привязка к C++ и ФОРТРАН 90,
- внешние интерфейсы,

- расширенные коллективные коммуникации,
- расширения реального времени,
- некоторые другие сферы.

1.2. Что предлагает MPI?

MPI имеет несколько высококачественных реализаций, которые поддерживают переносимость, стандартизацию, эффективность работы и функциональность.

Стандартизация

Интерфейс MPI стандартизован во многих аспектах. Например, в силу стандартности синтаксиса вы можете положиться на ваш MPI-код при запуске в любой реализации MPI, работающей на вашей архитектуре. В силу стандартности функционального поведения вызовов MPI вам не нужно беспокоиться о том, какая реализация MPI установлена сейчас на вашей машине: исполнение функций MPI не зависит от реализации. Эффективность работы, однако, может зависеть от реализации.

Переносимость

В быстро изменяющемся окружении высоко производительных компьютеров и технологий коммуникации почти всегда важно иметь переносимость параллельных программ. Кто захочет развивать программу, которая может выполняться только на одной машине, а на других машинах только при дополнительных затратах труда? Все системы массивной параллельной обработки обеспечивают своего рода библиотеку передачи сообщений, которая точно определена аппаратными средствами используемой ЭВМ. Эти системы обеспечивают отличную эффективность работы, но прикладной код, написанный для одной платформы, нельзя легко перенести на другую платформу.

MPI позволяет вам писать портативные программы, которые хотя и используют в своих интересах спецификации аппаратных средств ЭВМ и программного обеспечения, предлагаемого поставщиками, но, к счастью, эти заботы в основном берут на себя функции MPI, ибо конструкторы настроили эти вызовы на основные аппаратные средства ЭВМ и окружающую программную среду.

Эффективность работы

Множество внешних инструментов, включая PVM, Express и P4 (см. краткий глоссарий терминов в приложении 1), пыталось обеспечить стандартизованное окружение для параллельных вычислений, но, тем не менее, ни одна из этих попыток не показала такой высокой эффективности работы, как MPI.

Богатство возможностей

MPI имеет целый набор качественных реализаций. Эти реализации обеспечивают асинхронную коммуникацию, эффективное управление буфером сообщения, эффективные группы, и богатые функциональные возможности. MPI включает большой набор коллективных операций коммуникации, виртуальных топологий и различных способов коммуникации, и, кроме того, MPI поддерживает библиотеки и неоднородные сети.

Имеющиеся в настоящее время реализации включают
MPICH: реализация Argonne National Lab [6]
LAM: реализация Ohio Supercomputer Center
MPI/Pro: реализация MPI Software Technology
IBM MPI: реализация IBM для кластерных систем SP и RS/6000 [5]
CHIMP: реализация Edinburgh Parallel Computing Centre
UNIFY: реализация Mississippi State University

MPI имеется в наличии на многих массивно параллельных системах. ВЦ РАН и МСЦ используют MPICH-реализацию MPI, последнюю версию которой можно свободно скачать с сайта Argonne National Lab (http://www-unix.mcs.anl.gov/mpi/mpich/download.html).

1.3. Как использовать MPI?

Если у вас уже есть последовательная версия программы, и вы собираетесь ее модифицировать, используя MPI, перед распараллеливанием убедитесь, что ваша последовательная версия безукоризненно отлажена. После этого добавьте вызовы функций MPI в соответствующие места вашей программы.

Если Вы пишете программу MPI с чистого листа, и написать сначала последовательную программу (без вызовов MPI) не составляет большого труда, сделайте это. Повторим, идентификация непараллельных ошибок вначале и их удаление намного облегчает отладку параллельной программы. Проектируйте ваш параллельный алгоритм, используя в своих интересах любой параллелизм, свойственный вашему последовательному коду, например, большие массивы можно разбить на подзадачи и обрабатывать независимо.

Отлаживая параллельную версию программы, удостоверьтесь сначала, что запуск вашей программы успешно проходит на небольшом числе узлов. Затем постепенно увеличивайте число узлов, например, от 2 до 4, затем до 8, и т.д. Используя такой способ, вы не будете впустую тратить много машинного времени на поиск дополнительных ошибок.

2. Лабораторная работа № 1: Теория

2.1. Программы MPI

В этом разделе рассмотрена простейшая программа с MPI. Такое рассмотрение нужно, чтобы дать наглядное представление об относящихся к делу вопросах, к которым потом, при желании, можно вернуться, если у вас возникнут вопросы типа:

- в каком порядке следует делать представленные вызовы?
- как выглядит список параметров?

Данная программа сама по себе не является чем-то большим, чем широко известная программа «Hello, world», поэтому мы не касаемся целей алгоритма, наша задача раскрыть механизм осуществления параллельной версии этой чрезвычайно простой задачи.

2.1.1. Формат функций MPI

Во-первых, рассмотрим форматы фактических вызовов, используемых MPI.

Для C общий формат имеет вид

```
rc = MPI_Xxxxx(parameter, ... )
```

Заметим, что, как и в любой команде С, регистр здесь важен. Например, MPI должно быть записано заглавными буквами, то же относится и к первой букве после подчеркивания. Все последующие символы должны быть в нижнем регистре. Переменная rc - это некий код возврата, имеющий целый тип. В случае успеха, он принимает значение MPI_SUCCESS.

Программа на С должна включать файл "mpi.h". Он содержит определения для констант и функций MPI. Заметим, что константы в MPI, такие как MPI_SUCCESS, MPI_INT, записываются заглавными буквами.

В случае языка Фортран общий формат выглядит так

```
Call MPI_XXXXX(parameter,..., ierror)
```

Заметим, что здесь регистр не важен. Поэтому, эквивалентной формой будет

```
call mpi_xxxxx(parameter,..., ierror)
```

В отличие от С, в которой функции MPI возвращают код ошибки, Фортран-версия подпрограмм MPI обычно имеет один дополнительный параметр в списке вызова, ierror, который равен коду возврата. В случае успешного вызова, ierror принимает значение MPI_SUCCESS.

Программы Фортран должны включать 'mpif.h' (для компилятора Compaq 'mpif90.h') вместо 'mpi.h'. Это файл определения для констант и функций MPI.

Для обеих привязок и С, и Фортрана исключением к вышеприведенным функциям являются функции времени (MPI_Wtime и MPI_Wtick), которые являются функциями как в С, так и в Фортране, и возвращают действительные числа с двойной точность. В дальнейшем изложении мы ограничимся привязкой только к языку С. Изучив привязку MPI к С, желающие, как уже говорилось, могут самостоятельно освоить соответствующие подпрограммы Фортрана.

2.1.2. Функции MPI

Основная схема программы MPI состоит из следующих общих этапов:

1. Инициализация для коммуникаций
2. Коммуникации распределения данных по процессам
3. Выход "чистым" способом из системы передачи сообщений по завершении коммуникаций

MPI имеет свыше 125 функций. Тем не менее, начинающему программисту обычно достаточно иметь дело только с шестью функциями, которые иллюстрируют нашу простейшую программу и обсуждаются ниже:

Инициализация для коммуникаций

MPI_Init инициализирует окружение MPI

MPI_Comm_size возвращает число процессов

MPI_Comm_rank возвращает номер текущего процесса (ранг = номер по-порядку)

Коммуникации распределения данных по процессам

MPI_Send отправляет сообщение

MPI_Recv получает сообщение

Выход из системы передачи сообщений

MPI_Finalize

2.1.3. Пример MPI-программы

Вызов этих шести базовых функций MPI показан в следующем ниже коде на языке C.

```
#include <stddef.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "mpi.h"
main(int argc, char **argv)
{
  char message[20];
  int  i, rank, size, type = 99;
```

```
 MPI_Status status;
 MPI_Init(&argc, &argv);
 MPI_Comm_size(MPI_COMM_WORLD, &size);
 MPI_Comm_rank(MPI_COMM_WORLD, &rank);

 if (rank == 0)
 {
   strcpy(message, "Hello, world!");
   for (i = 1; i < size; i++)
      MPI_Send(message, 14, MPI_CHAR, i, type,
                    MPI_COMM_WORLD);
 }
 else
    MPI_Recv(message, 20, MPI_CHAR, 0, type,
                   MPI_COMM_WORLD, &status);
 printf( "Message from process = %d : %.14s\n",
          rank,message);
 MPI_Finalize();
}
```

В электронной версии листинга простейшей программы (http://www.ccas.ru/mmes/educat/lab04k/01/csample.html), нажав на название любой функции MPI можно прочесть детальное описание ее предназначения и синтаксиса. Здесь же описание этих шести основных функций MPI помещено в приложении 2. Резюмируя эту программу можно сказать: это код SPMD (Single Program Multiple Data = одна программа множество данных, см. приложение 1), так что копии этой программы выполняются на множестве процессов. Каждый процесс инициализирует себя в MPI (MPI_Init), определяет число процессов (MPI_Comm_size) и узнает его ранг (MPI_Comm_rank). Затем один процесс (с рангом 0) отправляет сообщения в цикле (MPI_Send), устанавливает целевой аргумент (предназначение) равным индексу цикла, чтобы быть уверенным, что в каждый из оставшихся процессов получит одно сообщение. Каждый из оставшихся процессов получает одно предназначенное ему сообщение (MPI_Recv). Затем все процессы печатают сообщение и выходят из MPI (MPI_FInalize).

Здесь мы не заботимся о том, что произойдет в этой программе. Нет функций, которые вызывают дополнительные копии программы на выполнение. Для запуска программы на выполнение на кластере используют команду mpirun (см. приложение 3). В упражнении 1 лабораторной работы, приведенном в третьей части данного модуля, от вас потребуется расширить эту программу некоторыми дополнительными вызовами функций MPI.

2.2. Сообщения MPI

Сообщения MPI состоят из двух основных частей: отправляемые или получаемые данные (содержимое), и сопроводительная информация (записи на оболочке - конверте). Сопроводительная информация позволяет отправить данные по определенному маршруту. В вызовах передачи сообщений MPI обычно существуют три вызываемых параметра, которые описывают данные и три других параметра, которые определяют маршрут:

Сообщение = данные (3 параметра) + оболочка (3 параметра)

Типичный состав сообщения в MPI:

ДАННЫЕ	ОБОЛОЧКА
начало буфера, число, тип данных	цель, тег, коммуникатор

Ниже будет дано более детальное обсуждение каждого параметра в данных и оболочке, включая информацию о том, когда отправителю и получателю следует скоординировать эти параметры.

2.2.1. Данные

Буфер в вызовах MPI есть место в компьютерной памяти, из которого сообщения должны быть отправлены или где они накапливаются при приеме. В этом смысле буфер - это просто память, которую компилятор выделил для переменной (часто, массива) в вашей программе. Следующие три параметра вызова MPI необходимы, чтобы определить буфер:

Начало буфера

Адрес начала данных. Например, начало массива в вашей программе.

Число

Число элементов данных в сообщении. Заметим, что это число элементов, а не байт. Это делается для переносимости кода, чтобы не беспокоиться о различных представлениях типов данных на различных компьютерах. Реализация математического обеспечения MPI определяет число байт автоматически. Число, определенное при получении, должно быть не меньше числа, определенного при отправке. Если отправляется больше данных, чем имеется места в хранилище принимающего буфера, то произойдет ошибка.

Тип данных

Тип данных, которые будут передаваться - с плавающей точкой, например. Этот тип данных должен быть тем же самым в вызовах функций отправки и получения. Исключением из этого правила является тип данных MPI_PACKED, который является одним из способов обработки сообщений со смешанным типом данных (предпочтительнее использовать метод с производными типами данных). Проверка типов в этом случае не нужна.

Предопределенные в MPI типы данных называют "основными типами данных", они перечислены ниже.

Основные типы данных MPI для C

Типы данных MPI	Типы данных C
MPI_CHAR	signed char
MPI_SHORT	signed short int
MPI_INT	signed int
MPI_LONG	signed long int
MPI_UNSIGNED_CHAR	unsigned char
MPI_UNSIGNED_SHORT	unsigned short int
MPI_UNSIGNED	unsigned int
MPI_UNSIGNED_LONG	unsigned long int
MPI_FLOAT	float
MPI_DOUBLE	double
MPI_LONG_DOUBLE	long double

MPI_BYTE	
MPI_PACKED	

Производные типы данных

Могут быть также определены дополнительные типы данных, называемые **производными типами данных**. Вы можете определить тип данных для несмежных данных или для последовательности смешанных основных типов данных. Это может облегчить программирование и часто обеспечивает более быстрое выполнение кода. Производные типы данных будут описаны в модуле «Производные типы данных», одной из последующих лабораторных работ.

Некоторые производные типы данных:

- Contiguous
- Vector
- Hvector
- Indexed
- Hindexed
- Struct

2.2.2. Оболочка

Напомним, что сообщение состоит из данных (содержимого) и оболочки (сопровождения) сообщения. Оболочка содержит информацию о том, как связаны отправления с получениями. Три параметра используют для определения оболочки сообщения:

Цель (назначение или источник)

Этот аргумент устанавливают равным рангу процесса в комммуникаторе (см. ниже). Ранг меняется от 0 до (size-1), где size - это число процессов в коммуникаторе. Назначение определяется отправкой и используется, чтобы определить маршрут сообщения к соответствующему процессу. Источник определяется получением. Только сообщения, идущие от этого источника, могут быть приняты при вызове получения, но получение может установить источник в MPI_ANY_SOURCE, чтобы указать, что любой источник приемлем.

Тег

Тег (метка) – это произвольное число, которое помогает различать сообщения. Теги, определяемые отправителем и получателем, должны совпадать, но получатель может определить его как MPI_ANY_TAG, чтобы показать, что любой тег приемлем.

Коммуникатор

Коммуникатор, определенный при отправке должен равняться коммуникатору, определенному при получении. Несколько шире коммуникаторы обсуждаются далее в п. 2.3, а сейчас будет достаточно знать, что коммуникатор определяет коммуникационную "вселенную", и то, что процессы могут принадлежать более чем к одному коммуникатору. В этом модуле мы будем иметь дело только с предопределенным коммуникатором MPI_COMM_WORLD, который включает все процессы приложения.

Аналогия

Для лучшего понимания смысла параметров окружения сообщения рассмотрим аналогию с агентством, выпускающим иски на разные потребности. Отправляя иск, агентство должно указать:

1. Лицо, получающее иск (точнее, его идентификационный номер). Это - назначение.
2. Какой месяц охватывает этот иск. Так как лицо получает двенадцать исков в год, ему необходимо знать за какой месяц приходит этот иск. Это - тег.
3. На какую потребность выпускается иск. Лицу надо знать, за что получает он этот иск: иск ли это за электричество или иск за телефон. Это - коммуникатор.

2.3. Коммуникаторы

2.3.1. Зачем нужны коммуникаторы?

Раскроем теперь смысл понятия коммуникатора. Сейчас обсудим только детали, касающиеся использования коммуникаторов. Более глубокое описание коммуникаторов будет дано в модуле «Управление группами и коммуникаторами MPI».

Сообщение будет воспринято при вызове функции приема, если будут точно определены источник, тег и коммуникатор. Тег позволяет программе различать типы сообщений. Источник упрощает программирование. Вместо того, чтобы иметь уникальный тег для каждого сообщения, каждый процесс, отправляющий одну и ту же информацию, может использовать тот же тег. Но зачем нужен коммуникатор?

Рассмотрим пример

Предположим, вы передаете сообщения между вашими процессами, и, кроме того, используете ряд откуда-либо полученных библиотек, которые также порождают процессы, выполняемые на различных узлах, взаимодействующих друг с другом с помощью MPI. В этом случае вам нужно быть уверенным, что отправленные вами сообщения придут к вашим процессам и не будут смешаны с сообщениями, отправленными к процессам из библиотечных функций.

Допустим, вы имеете три процесса, взаимодействующих друг с другом. Кроме того, каждый процесс вызывает библиотечную функцию, и три параллельные части библиотеки взаимодействуют друг с другом. Вам хочется иметь два различных "пространства" сообщений, один для ваших сообщений и один для библиотечных сообщений. Вам не хотелось бы получить какое-либо перемешивание этих сообщений.

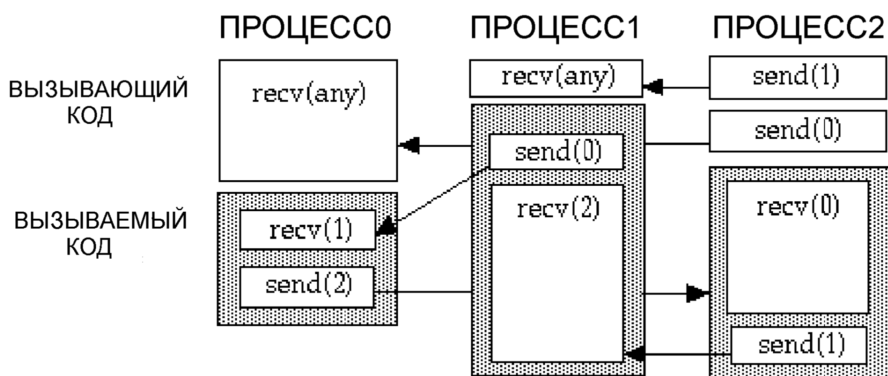

Рис. 1 Намеченное поведение коммуникаций между процессами

Блоки на рис.1 представляют собой части параллельных процессов. Время растет сверху вниз на каждой диаграмме. Цифры в круглых скобках НЕ параметры, а номера процессов. Например, send(1) означает отправку сообщения процессу 1, recv(any) означает получение сообщения от любого процесса. Пользовательский (вызывающий) код находится в белом (незатененном) блоке. Затененный блок (вызываемый) представляет собой пакет (параллельной) библиотеки, вызванной пользователем. Наконец, стрелки представляют собой перемещение сообщения от отправителя к получателю. Тем не менее, [illegible] никакой гарантии, что события произойдут в нужном порядке, так как относительное расписание процессов на различных узлах может меняться от исполнения к исполнению. Предположим, вы изменили третий процесс, добавив некоторые вычисления вначале. Последовательность событий может оказаться такой, что представлена на рис. 2.

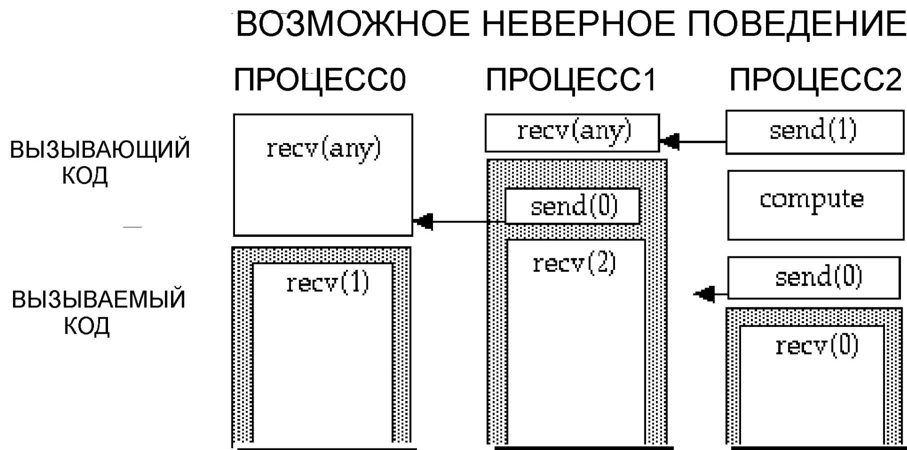

Рис. 2 Возможное неверное поведение коммуникаций

В этом случае коммуникации не происходят, как было намечено. Первый "receive" в процессе 0 теперь получает "send" из библиотечной функции в процессе 1, а не намеченный (и теперь задержанный) "send" из процесса 2. В результате все три процесса зависают.

Проблема решается разработчиком библиотеки за счет того, что библиотека запрашивает новый и уникальный коммуникатор и определяет этот коммуникатор во всех вызовах функций отправки и получения, которые делаются библиотекой. Это создает библиотечное ("вызываемое") пространство сообщений отделенное от пользовательского ("вызывающего") пространства сообщений.

Можно ли было использовать теги, чтобы осуществить разделение пространства сообщений? Проблема с тегами состоит в том, что их значения задаются программистом, а он может использовать тот же тег, что и параллельная библиотека, использующая MPI. С коммуникаторами система, а не программист, осуществляет идентификацию - система назначает коммуникатор пользователю и он назначает отличный

коммуникатор библиотеке - так что возможность перекрытия не возникает.

В электронной версии работы можно выполнить Java апплет http://www.ccas.ru/mmes/educat/lab04/01/CommApplet.html для закрепления понимания.

2.3.2. Группы коммуникаторов и процессов

В дополнение к работе с параллельными библиотеками, коммуникаторы полезны также в организации коммуникаций внутри приложения. Мы описали коммуникатор, который включает все процессы в приложении. Но программист может также определить подмножество процессов, называемое группой процессов, и прикрепить один или больше коммуникаторов к этой группе процессов. Коммуникатор определяет, что коммуникация будет теперь ограничиваться этими процессами.

В следующем ниже примере коммуникационным шаблоном является 2-мерная решетка (2D-mesh). Схема коммуникаций по двумерной решетке используется, например, для задач с геометрической структурой, в которой значения в соседних точках необходимы для уточнения значения в точке. Лучшим способом распараллеливания этой задачи является блочная декомпозиция данных по двум размерностям, и, затем, отображение блоков на 2-мерной решетке процессов. Решеточная топология - это абстракция, указывающая, каким процессам принадлежат соседние данные. Для точек внутри каждого процессного блока вся необходимая информация для уточнения точки содержится в этом процессе. Чтобы уточнить граничные точки процесса, необходимо знать строки и столбцы точек, содержащихся у соседей.

Пусть, например, каждый из шести блоков представляет собой процесс. Каждый процесс должен обмениваться данными с соседями сверху и снизу, справа и слева. Кодировать эту коммуникацию будет проще, если процессы сгруппировать по столбцам (для коммуникаций выше/ниже) и строкам (для коммуникаций направо/налево). Итак, каждый процесс принадлежит трем коммуникаторам, что указано словами в блоке этого процесса: один коммуникатор на все процессы (мировой

коммуникатор, который дан по умолчанию), один коммуникатор на его строку и один коммуникатор на его столбец. Эти коммуникаторы указаны ниже.

world	коммуникатор для	всех процессов	не помечен
comm1	коммуникатор для	строки 1	## помечен ##
comm2	коммуникатор для	строки 2	** помечен **
comm3	коммуникатор для	столбца 1	$$ помечен $$
comm4	коммуникатор для	столбца 2	!! помечен !!
comm5	коммуникатор для	столбца 3	?? помечен ??

world, rank0	world, rank1	world, rank2
## comm1, rank0 ##	## comm1, rank1 ##	## comm1, rank2 ##
$$ comm3, rank0 $$	!! comm4, rank0 !!	?? comm5, rank0 ??

world, rank3	world, rank4	world, rank5
** comm2, rank0 **	** comm2, rank1 **	** comm2, rank2 **
$$ comm3, rank1 $$	!! comm4, rank1 !!	?? comm5, rank1 ??

Здесь уместно снова вспомнить аналогию с выпуском исков, когда одно лицо может иметь счет от электрической и телефонной компаний (2 коммуникатора), но ни одного от водопроводной компании. Коммуникатор по электричеству может содержать список людей, отличающихся от списка людей в телефонном коммуникаторе. Персональный идентификационный номер (ранг) может изменяться с потребностью (коммуникатором). Итак, чрезвычайно важно заметить, что ранг, заданный как источник или назначение сообщения, есть ранг в точно определенном коммуникаторе.

2.4. Запуск MPI программ

2.4.1. Компиляция

Для создания параллельной исполняемой программы вам следует включить при активизации компилятора директорию, содержащую MPI и библиотеки MPI. Подразумевается, что компилятор и реализация MPI установлены на машине, где вы компилируете исполняющую программу.

Компиляция С-программы из командной строки:

mpicc –o prog prog.c -lm

Здесь компилируется С-программа из файла prog.c. При этом используется библиотека математических функций и создается исполняемая программа с именем prog.

2.4.2. Запуск на исполнение из командной строки

Запуск на исполнение MPI-программы производится с помощью команды:

mpirun [параметры_mpirun...] <имя_программы> [параметры_программы...] [-host <host>]

Параметры команды mpirun (см. приложение 3) следующие:

-h | -help
показывает аргументы командной строки для mpirun

-maxtime <максимальное_время>
определяет максимальное время счета. От этого времени зависит положение задачи в очереди. По истечении этого времени задача принудительно заканчивается. Обязательный параметр в МСЦ.

-np <число_процессоров>
определяет число процессоров, требуемое программе.

-wd <рабочая директория>
определяет рабочую директорию для использования процессом MPI. Это необходимо делать только тогда, когда текущая директория и желаемая рабочая директория не совпадают.

-mpi_debug
включает дополнительную проверку на входные параметры и дополнительный выход, если ошибки возникнут при выполнении.

Пример запуска на исполнение из командной строки

```
mpirun –np 4 prog –maxtime 5
```

Здесь запускается на исполнение программа prog на 4 процессорах с ограничением по времени счета до 5 минут.

2.4.3. Запуск в пакете

Процедура запуска параллельной работы в пакете немного сложнее, чем запуск последовательной работы:

- Необходимо позаботиться о переключении между узлами. Помните, что собираемые в наше время узлы содержат, как правило, по 2 процессора каждый, а некоторые ускоренные машины имеют и по 4 процессора на каждом узле.
- mpirun следует использовать для выполнения исполняемой программы.
- Файлы необходимо скопировать на все узлы работы и из всех узлов работы.

Пример рабочего файла, используемого для запуска программы MPI в пакете в системе PBS на кластере ВЦ РАН, приведен ниже:

```
#!/bin/sh
#PBS –l walltime=00:30:00,nodes=4:ppn=2
#PBS –q @broody
##P#BS –q shortest
#PBS –V
```

```
#PBS –j oe
#PBS –N job2

echo $PBS_NODEFILE
# here are list of used nodes in $PBS_NODEFILE

cd /home/nicko
# you can put any commands here

/usr/local/bin/mpiexec –n 8 ./qqq
# here is command-line for running parallel mpi program
# parameter –n is equal to number of processes,
# then follows name of program.
# Machine file is automatically used from
# $PBS_NODEFILE
```

2.5. Резюме

Хотя MPI обеспечивает расширенное множество функций, функциональная программа на MPI может быть записана с помощью всего шести базовых функций:

- MPI_Init
- MPI_Comm_rank
- MPI_Comm_size
- MPI_Send
- MPI_Recv
- MPI_Finalize

Для удобства программирования и оптимизации кода, вам следует рассмотреть использование других вызовов, например, таких, которые описаны в последующих лабораторных работах, посвященных изучению попарных и коллективных коммуникаций.

Сообщения MPI

Сообщения MPI состоят из двух частей:

- данные (начало буфера, число, тип данных)
- оболочка (назначение/источник, тег, коммуникатор)

Данные определяют информацию, которая будет отослана или получена. Оболочка используется в маршрутизации сообщения к получателю и связывает вызовы отправки с вызовами получения.

Коммуникаторы

Коммуникаторы гарантируют уникальные пространства сообщений. В соединении с группами процессов их можно использовать, чтобы ограничить коммуникацию на подмножество процессов.

3. Лабораторная работа № 1: Вопросы по основам программирования в MPI

1. Что вы всегда можете ожидать от MPI?
   a. Верная программа MPI должна выполняться на любой машине, которая поддерживает MPI
   b. Верная программа MPI должна давать сопоставимое представление на любой машине, которая поддерживает MPI

2. Отметьте каждый верный пункт о MPI:
   a. MPI - библиотека передачи сообщений
   b. MPI - официальный стандарт
   c. Программы MPI переносимы
   d. Функции, начинающиеся с "MPE" являются частью MPI

3. Что неправильно в следующем вызове функции MPI на Fortran (ответ следует дать без использования man-страницы руководства)?

```
count = 12
dest = 0
tag = 100
call MPI_Send (buffer, count, MPI_CHARACTER, dest, tag,
  MPI_COMM_WORLD)
```

   a. В этом случае неверно имя подпрограммы MPI

b. Пропущен последний аргумент, который должен быть кодом ошибки

4. Какие из следующих предложений о ранге не верны?
   a. Ранг есть целое число между 0 и nproc - 1, где nproc равно числу процессов в приложении
   b. Каждый ранг уникален внутри коммуникатора
   c. Ранг возвращается посредством обращения к MPI_Comm_rank

5. "Буфер" в MPI является:
   a. Временным размещением выхода
   b. Коммуникационным путем
   c. Пространством в памяти

6. Параметр "число" в вызовах отправки и получения в MPI измеряется в единицах:
   a. В световых единицах
   b. В байтах данных
   c. Элементов данных
   d. Пакетов данных

7. Правда или нет: Уникальный тег должен быть точно определен при каждом вызове функции получения
   a. Правда
   b. Ложь

8. Как много байт в данных сообщения передается в данном вызове из C?

```
MPI_Send(buffer, 1024, MPI_INT, dest, tag, MPI_COMM_WORLD)
```

   a. 1024 байт
   b. 1024 * число байт, используемых для записи целого числа

9. Проверьте все, что подходит: Что должно быть верно для того, чтобы сообщение было "маршрутизировано" к точно определенному вызову получения?
   a. Коммуникатор, определенный при отправке, должен совпадать с коммуникатором, определенным при получении
   b. Тег сообщения, определенный при отправке, должен равняться тегу сообщения, определенному при получении
   c. Тег сообщения, определенный при отправке, должен соответствовать тегу сообщения, определенному при получении

10. Основная цель коммуникатора состоит в том, чтобы

a. Определить размер группы процессов
b. Помочь в маршрутизации сообщений
c. Определить число задач в функции параллельной библиотеки

16. С каким числом функций MPI обычно может иметь дело программа для начинающих?

a. 2
b. 4
c. 6
d. 8

17. Для каждого из следующих вызовов MPI из C программы, проверьте когда этот вызов *НЕ* работает. Предполагается, что все другие необходимые команды представлены верно.

a. MPI_Send(msg, 12, MPI_CHARACTER, i, tag, MPI_COMM_WORLD)
   не работает
b. MPI_Comm_size(MPI_COMM_WORLD, &size)
   не работает
c. rc=MPI_Comm_rank(MPI_COMM_WORLD,&rank,&ierror);
   не работает

4. Лабораторная работа № 1: Упражнения по основам программирования в MPI

4.1. Предварительные требования

Практические упражнения могут быть выполнены не только на кластере ВЦ РАН, но и на виртуальной параллельной машине, устанавливаемой в Windows NT/2000 на вашем компьютере. Для этого перед началом работы вам следует установить переносимую реализацию MPI - MPICH для Windows - на ваш компьютер (http://www-unix.mcs.anl.gov/mpi/mpich/). Предварительно следует изучить руководство по установке MPICH (http://www.cluster.bsu.by/mpich_install.pdf) и руководство пользователя MPICH (http://www.cluster.bsu.by/mpich_userguide.pdf). Кроме того, полезно изучить руководство пользователя МВС 1000М (http://www.jscc.ru/informat/1000MUsrGuide.zip) Межведомственного суперкомпьютерного центра (МСЦ), с тем, чтобы вы знали ответы на следующие вопросы:

- Компиляция при использовании библиотек MPI
- Использование команды mpirun
- Создание файла
- Понимание механизма выполнения MPI на двух узлах

Именно необходимость предварительной подготовки к курсу MPI увеличивает общее время выполнения лабораторной работы № 1 примерно в два раза.

4.2. Обзор

В двух первых упражнениях данной лабораторной работы требуется модифицировать очень простую программу MPI, используя только шесть базовых вызовов MPI.

В третьем упражнении дана простая последовательная программа, названная karp, которая вычисляет PI, используя цикл for для вычисления аппроксимации интеграла. Требуется

преобразовать ее в параллельную программу, используя представление SPMD (одна программа множество данных).

Четвертое упражнение, предлагаемое в качестве домашнего задания, имеет не только учебный характер, но и показывает пример приложения полученных знаний для решения реальной научной задачи – задачи идентификации параметров в сложной математической модели. Здесь требуется распараллелить цикл перебора значений параметров программы, идентифицирующей параметры производственного блока в модели экономики России переходного периода.

Ориентировочное время выполнения этих упражнений для новичков в использовании MPI:

- Упражнение 1: 20-30 минут
- Упражнение 2: 20-30 минут
- Упражнение 3: 45-60 минут
- Упражнение 4 (домашнее задание): 180-240 минут.

Синтаксис и описание шести самых распространенных MPI-функций содержится в Приложении 2, при необходимости их можно найти также в руководстве программиста суперкомпьютера МВС 1000М (http://www.jscc.ru/informat/1000MPrgGuide.zip).

Заметим, что в электронной версии работы (http://www.ccas.ru/mmes/educat/lab04k/01/lab04_01.html) исходные файлы в упражнениях можно свободно скачать, что освобождает значительное время от рутинной работы. Однако, для полноты описания, листинги всех исходных файлов даны и в печатной версии.

4.3. Упражнение 1:

Добавить отправку сообщений от рабочих мастеру

исходный С файл: hello.c

файл решения на С записать в: helloex1.c

Листинг параллельной программы hello.c

```
#include <stddef.h>
#include <stdio.h>
```

```
#include <stdlib.h>
#include <string.h>
#include "mpi.h"
main(int argc, char **argv)
{
  char message[20];
  int i, rank, size, type = 99;
  MPI_Status status;

  MPI_Init(&argc, &argv);
  MPI_Comm_size(MPI_COMM_WORLD, &size);
  MPI_Comm_rank(MPI_COMM_WORLD, &rank);

  if (rank == 0)
  {
    strcpy(message, "Hello, world!");
    for (i = 1; i < size; i++)
       MPI_Send(message, 14, MPI_CHAR, i, type,
                 MPI_COMM_WORLD);
  }
  else
     MPI_Recv(message, 20, MPI_CHAR, 0, type,
                 MPI_COMM_WORLD, &status);

  printf( "Message from process = %d : %.14s\n", rank,message);
  MPI_Finalize();
  return 0;
}
```

Hello является программой SPMD (Single Program Multiple Data = одна программа множество данных), то есть, одна и та же программа выполняется и как процесс "мастер", и как процессы "рабочие". Программа определяет, является ли она мастером (ранг 0) или рабочим (ранг 1 или выше) посрдством предложения if, и затем разветвляется по соответствующим сегментам программы.

Мастер посылает сообщение ("Hello world!") всем рабочим, и затем распечатывает сообщение на стандартный вывод stdout.

Каждый рабочий получает его сообщение, затем распечатывает его на stdout.

Ваша миссия заключается в том, чтобы модифицировать программу hello так, чтобы каждый рабочий вместо распечатки подтверждения посылал сообщение обратно мастеру, добавив в него рабочий ранг. Мастер должен получить эти сообщения и распечатать сообщение и соответствующие ранги на stdout.
Запустите эту программу на 4 процессорах.

4.4. Упражнение 2:

Согласовать сообщения, используя теги

исходный C файл: helloex1.c

файл решения на C записать в: helloex2.c

Приложение может использовать параметр тег в вызовах отправки и получения, чтобы отличать сообщения. Модифицируйте программу, полученную в упражнении 1 так, чтобы мастер отправлял сообщения каждому рабочему, используя два различных тега. Используя теги заставьте рабочих получать сообщения в обратном порядке и затем ответьте мастеру, как в упражнении 1. И опять заставьте мастера получить и распечатать каждое сообщение и соответствующий ранг на stdout.

Запустите эту программу на 8 процессорах. Создайте командный файл для запуска ex2.bat и очистки ex2cu.bat.

4.5. Упражнение 3:

Преобразовать последовательный код в параллельный

исходный C файл: karp.c

файл решения на C записать в: karpsoln.c

входной файл данных: values

Листинг последовательной программы karp.c

```
/* Parallelizing for MPI Lab Karp Example karp.c
 * Last revised RYL 2/6/95  */
#include <stdlib.h>
```

```
#include <stdio.h>
#include <math.h>
#define f(x) ((double)(4.0/(1.0+x*x)))
#define pi ((double)(4.0*atan(1.0)))
void startup (void);
int solicit (void);
void collect (double sum);

int main()
{ /* This simple program approximates pi by computing
   * pi = integral from 0 to 1 of 4/(1+x*x)dx which is approximated
   * by sum from k=1 to N of 4 / ((1+[(1/N)*(k-1/2)]**2) and then
   * multiplying the sum by (1/N). (This numerical integration rule
   * is called "Midpoint rule" and can be found in most numerical
   * analysis text books). The only input data required is N.
   */
  double sum, w;
  int i, N;
  /* The startup routine will create parallel tasks
   */
  /* startup(); */
  /* The solicit routine will get and propagate the value of N
   */
  N = solicit();
  while (N > 0) {
    w = 1.0/(double)N;
    sum = 0.0;
    for (i = 1; i <= N; i++)
      sum = sum + f(((double)i-0.5)*w);
    sum = sum * w;
    /* The collect routine will collect and print results
     */
    collect (sum);
    N = solicit ();
  }
  return (0);
}
```

```
/*  ----------------------------------------  */
void startup (void)
{
}

/*  ----------------------------------------  */
int solicit (void)
{
 int N;
 printf ("Enter number of approximation intervals:(0 to exit)\n");
 scanf("%d",&N);
 return (N);
}

/*  ----------------------------------------  */
void collect(double sum)
{
 double err;
 err = sum - pi;
 printf("sum, err = %7.5f, %10e\n", sum, err);
}
```

Листинг входного файла данных values

```
10
100
0
```

Программа karp вычисляет PI, используя интегральную аппроксимацию. Вам предоставлена последовательная версия программы karp, и от вас требуется модифицировать ее в параллельную версию в форме SPMD.

a. Разберитесь с тем, как работает последовательная версия. Скопируйте и запустите последовательную программу, затем найдите ответ на следующие вопросы:

- Как программа считает PI? (Совет: переведите комментарии к программе)

- Как точность вычисления зависит от числа шагов аппроксимации N? (**Совет:** отредактируйте values для различных входных значений от 10 до 10000)
- Как вы думаете, что будет с точностью, с которой мы вычисляем PI, когда мы разобъем работу по узлам?

b. Добавьте вызовы MPI, чтобы создать некую SPMD karp-программу. В этом разделе вы разбиваете работу по параллельным процессам.

Цель: Получить реальную, работающую, SPMD программу, осуществленную на MPI.

Шаги: Отредактируйте karp, чтобы разбить работу по процессам. Используйте только шесть базовых вызовов MPI.

Совет: мастеру нужно передать каждому рабочему полное число итераций и, тогда, каждый рабочий рассчитывает свои индексы цикла с тем, чтобы проделать свою часть работы. Каждый рабочий после окончания отработки отправляет свою частную сумму назад мастеру, который в свою очередь получает эти частные суммы и вычисляет окончательную сумму.
- Рассчитать задачу на одном из вводимых узлов
- Выполнить задачу с 4 процессорами и с 8 процессорами.

Ответьте на вопросы

- Считает ли программа?
- Дает ли она правильный ответ?
- Будет ли вычисленное значение PI для данного N одним и тем же?
- Как может операция «распространения (broadcast)», в которой одна задача отправляет одно и тоже сообщение всем другим задачам, помочь вам? (Операция MPI «broadcast» будет изучаться позже.)

4.6. Упражнение 4:

Идентифицировать параметры сложной модели (домашнее задание)

Требуется преобразовать заданный последовательный код программы pf, решающей задачу идентификации параметров

производственного блока переходной экономики России, в параллельный код.

файл описания переменных: pf.h
исходный *C* файл: pf.c
параллельный код решения на *C* следует записать в файл: parpf.c

Файл описания переменных pf.h содержит исходные статистические данные, используемые в программе идентификации параметров производственного блока модели.

Листинг файла исходных данных pf.h

```
/* pf.h - файл описания переменных: */
#define NW 36
 /* число кварталов для статистических данных */
/***** Описание глобальных переменных: *****/
double dt = 0.25;     /* шаг по времени = 1 квартал */

typedef struct {
        double
                R_stat[NW],    /* Труд, млрд. чел - час в год
                                * (4 квартала, 8 часов в неделю,
                                * / 1000 перевод млн в млрд) */
                J_stat[NW],     /* Инвестиции */
                Y_stat[NW],     /* ВВП в ценах I кв 2000 г. */
                p_stat[NW],     /* дефлятор ВВП = индексу цен */
                L_stat[NW],     /* кредиты */
                C_stat[NW],     /* основные фонды (на конец года по
                                 * полной учетной стоимости)
                                 * трлн руб 2000 г. */
                rl_stat[NW],    /* ставка по кредиту */
                s_stat[NW],     /* ставка зарплаты, тыс руб на чел-час
                                 */
                dLdt_stat[NW]; /* прирост кредитов */
} SRC; /* источник входных данных */
```

```
int workdays[NW]={
        61, 61, 65, 63, 61, 61, 66, 64, 60, 60, 66, 64,
        61, 61, 66, 64, 60, 61, 66, 65, 62, 61, 65, 63,
        61, 60, 65, 64, 59, 61, 66, 63, 59, 61, 66, 64
}; /* рабочих дней в квартале с I кв 1995 по IV кв 2003*/

double employment[NW]={
        66.6, 66.5, 66.5, 66.2, 66.2, 66.0, 65.9, 65.6,
        65.0, 64.63333333, 64.56666667, 64.4,
        64.0, 63.73333333, 63.53333333, 63.3,
        63.23333333, 64.03333333, 64.43333333,
        63.86666667, 62.8, 64.7, 65.06666667, 64.4,
        63.33333333, 64.43333333, 65.26666667, 64.8,
        65.06666667, 66.06666667, 67.13333333,
        65.76666667, 64.46666667, 65.46666667,
        66.46666667, 64.6
}; /* число занятых в экономике. млн чел (‘численность’) */

double J_src[119]={
        2082.171138, 1906.715273, 1370.396029,
        1491.31279, 1681.746529, 1545.371785,
        1505.911462, 1427.435358, 1392.219688,
        1285.89264, 1496.628226, 1189.539056,
        1363.885879, 1695.993296, 1314.770387,
        1275.892978, 1397.314222, 1366.638365,
        1372.003879, 1307.845169, 1357.182612,
        1309.067629, 1397.403925, 1361.432295,
        1377.834705, 1262.460074, 1224.652224,
        1183.414471, 1069.022445, 1028.060546,
        1051.009549, 1012.163342, 1071.727515,
        1055.644552, 1107.616877, 1085.80739,
        1134.296883, 1068.573749, 1108.145213,
        1103.02707, 994.2511165, 1006.914326,
        1036.804602, 1002.563027, 1089.64018,
        1065.842405, 1109.388122, 1133.496311,
        1065.951694, 1004.188599, 1007.60729,
        1005.122045, 928.4800697, 927.4382514,
```

```
        967.1994075, 937.2276545, 879.3852152,
        859.1316934, 890.9605226, 837.8974036,
        1005.430716, 963.6111765, 980.2921424,
        973.7612341, 942.2539392, 976.3017833,
        1010.242111, 977.0189393, 999.4864768,
        1008.986446, 1021.757367, 1050.908776,
        1086.395164, 1120.335966, 1135.803259,
        1136.037419, 1152.120372, 1173.220273,
        1182.64448, 1198.542121, 1197.077677,
        1194.309918, 1212.496122, 1193.417229,
        1200, 1223, 1221, 1245, 1290, 1271, 1290,
        1314, 1316, 1330, 1329, 1350, 1205, 1224,
        1264, 1294, 1335, 1314, 1335, 1334, 1355,
        1367, 1358, 1395, 1300, 1358, 1407, 1459,
        1531, 1475, 1492, 1497, 1531, 1543, 1521
};      /* инвестиции в основной капитал, млрд руб 2000 г.
         * помесячно в пересчете на год с января 1994 по
         * ноябрь 2003 */

double GDPnom[NW] = {
        235.0, 324.3, 421.1, 448.1, 425.3, 468.4,
        548.9, 565.2, 512.4, 555.1, 634.2, 640.8,
        550.9, 602.5, 675.5, 800.7, 901.3,
        1101.5, 1373.1, 1447.3, 1527.4, 1696.6,
        2037.8, 2043.8, 1900.9, 2105.0, 2487.9,
        2449.8, 2269.0, 2528.4, 3021.1, 3015.7,
        2893.1, 3134.9, 3688.3, 3588.2
};      /* Номинальный объем произведенного ВВП по
         * кварталам с 1 кв 1995 по 4 кв 2003,
         * в текущих ценах, млрд.рублей,
         * после пересмотра квартальных данных
         * 2001-02 9.03.2004. Квартальные данные за 2003
         * год не пересматривались и не соответствуют
         * годовым оценкам. */

double GDPre95[NW] = {
        100.0, 103.5, 119.9, 109.4, 97.8, 100.0,
```

```
        113.3, 106.1, 97.4, 99.2, 116.5, 109.9,
        95.9, 98.2, 106.3, 99.9, 94.2, 101.3,
        118.4, 111.9, 104.9, 111.6, 130.9, 121.1,
        110.1, 115.9, 136.4, 126.6, 114.2, 120.9,
        142.5, 134.2, 120.9, 129.7, 151.6
}; /* ВВП реальный в % к 1 кварталу 1995 */

double tmpre00[15] = {
        104.9, 122.4, 115.2, 104.7, 110.2, 129.7,
        120.4, 108.6, 115,   135.5, 127.6, 114.9,
        123.3, 144.1, 135.9
};/* Динамика реального объема произведенного ВВП
   * по кварталам, в % к 1 кварталу 2000 года (квартальные
   * данные за 2001-2002 годы пересмотрены 9.03.2004)  */

double tmpCred[23] = {
        206.949, 196.29, 201.115, 287.761, 300.2,
        329.569, 331.663, 354.529, 445.2, 483.6,
        543.9, 626.5, 763.3, 808.3, 894.5,
        1034.8, 1191.5, 1244.1, 1353, 1481.6,
        1612.7, 1722.8, 1878.9
}; /* Объем кредитов всем предприятиям с IV кв 1997 г
    * по II-й 2003, млрд руб */

double C_proc[NW] = {
        0, 0, 0, 100.1,0, 0, 0, 99.9, 0, 0, 0, 99.6,
        0, 0, 0, 99.6, 0, 0, 0, 100.1, 0, 0, 0, 100.4,
        0, 0, 0, 100.6, 0, 0, 0, 100.7, 0, 0, 0, 0
}; /* Основные  фонды в % к предыдущему периоду */

double r_src[107] = {
        179.7, 175.2, 173.9, 172.5, 148.9, 140.9,
        130.2, 115.3, 123.1, 126.2, 118.8, 119.,
        108., 105., 105., 104.4, 105.4, 94.3, 83.0,
        94.7, 65.0, 61.3, 51.6, 44.2, 46.1, 41.6,
        32.5, 34.0, 28.6, 28.8, 28.3, 24.8, 24.0,
        23.0, 28.6, 29.8, 29.8, 30.2, 38.8, 39.6,
```

```
        40.7, 47.7, 44.2, 48.5, 44.8, 48.2, 45.1,
        40.5, 44.8, 44.0, 47.5, 44.1, 44.7, 32.1,
        39.0, 38.6, 37.9, 37.0, 38.8, 32.1, 33.8,
        31.2, 29.5, 29.1, 25.3, 22.8, 22.5, 21.2,
        20.2, 19.9, 18.2, 18.1,18.5, 19.1, 18.7,
        17.4, 18.1, 18.0, 18.5, 18.0, 17.1, 17.4,
        16.9, 16.3, 18.0, 15.9, 15.7, 18.4, 17.7,
        15.2, 16.1, 14.9, 13.4, 13.6, 14.7, 14.9,
        14.6, 14.0, 13.4, 15.8, 12.6, 11.9, 11.9,
        11.8, 13.2, 12.6, 12.1
}; /* Ставки по кредитам предприятиям и организациям
    * в российских рублях, % годовых
    */

double s_src[108] = {
        302.642, 320.986, 361.491, 386.244,
        429.935, 480.64, 499.532, 520.579,
        564.472, 594.502, 615.656, 735.483,
        654.837, 684.421, 744.962, 746.473,
        779.309, 837.193, 842.78, 831.044,
        848.071, 843.344, 835.02, 1017.05, 812.2,
        821.2, 902.9, 901.1, 919.7, 993.2, 999.1,
        982.3, 1026.2, 1006.1, 997.8, 1214.8, 988,
        1000, 1059, 1040, 1047, 1122, 1110, 1052,
        1112, 1123, 1164, 1482, 1167, 1199, 1385,
        1423, 1472, 1626, 1618, 1608, 1684, 1716,
        1789, 2283, 1830, 1839, 2018, 2039, 2101,
        2294, 2302, 2289, 2367, 2425, 2508, 3025,
        2733, 2655, 2964, 2923, 3054, 3284, 3364,
        3376, 3405, 3515, 3578, 4541,
        3760, 3725, 4031, 4110, 4187, 4460, 4597,
        4511, 4521, 4646, 4785.2, 5868, 4696, 4701,
        4986, 5100, 5221, 5550, 5615, 5491, 5556,
        5864, 5990, 7344}; /* Среднемесячная
                              * начисленная зарплата, руб */
```

Исходные данные, введенные через файл описания pf.h преобразуются в основной программе к единицам измерения, используемым в производственном блоке модели переходной экономики России. Затем осуществляется идентификация параметров.

Листинг последовательной версии программы: pf.c

```
/* pf.c Author: N. Olenev 2005 olenev@ccas.ru */
#include "pf.h"
#include <stdio.h>
#include <stddef.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#define TINY 1.0e-20 /* регулировка необычного
                      * случая полной корреляции */

SRC src; /* src - экземпляр объекта SRC данных из pf.h */

/***** прототипы используемых функций: *****/
void Scale(int num, double vec[], int dim1,
int dim0, double res[]);
double LinearCorrelation(double x_x[],
double y_y[], int n_1, int n_2);
double TheilInequality(double x_x[],
double y_y[], int n_1, int n_2);
double Min4(double x_1, double x_2, double x_3, double x_4);
void fun(int iii, int N[], int i[]);

/**** Основная программа: *****/
int main(int argc, char **argv)
{
        FILE *fopt;
        int a;
        double GDPre00[NW];
          /* реальный объем произведенного ВВП по
           * кварталам, в отношении к 1 кварталу 2000 г */
        int j, cond[NW]; /* cond=условие: (1-n)*rl[a]<=rho[a] */
```

```
        double YLC1, CLC1, LLC1, sLC1, LC1;
                /* коэффициенты корреляции */
        double YTI1, CTI1, LTI1, sTI1, TI1;
                /* индексы расхождения Тэйла */
        double CRIT, C_tmp1[8], C_tmp2[8];
                /* критерий CRIT -> max */
        double C0, A0;
/* Начальные значения капитала par[0] = 8.5; (1.0, C0 - 1.0) */
        double C[NW], B[NW], A[NW];
          /* основные фонды: A – старые, B - новые, C - все */
        double J[NW], rl[NW], p[NW], R[NW];
          /* инвестиции, процент по кредиту, цена, труд */
        double x[NW], f[NW], iota[NW], L[NW], dLdt[NW];
        double Dfdx[NW],omega[NW],s[NW],rho[NW],Z[NW];
          /* x(t), f(x(t)), iota - инфляция, L - задолженность,
           * dL/dt, df/dx - прирост, omega и s - зарплата;
           * rho процент, Z прибыль */
        double Y[NW], YY[NW], YY_s[NW];
        double CC[NW], CC_s[NW], LL[NW], LL_s[NW];
        double ss[NW], ss_s[NW], s_re[NW];
        double s_re1[NW], Z_s1[NW], Z_re1[NW];
        double n_re, m_nonpay[NW], n_nonpay[NW];
          /* Y ВВП, LL задолженность */
        double n = 0.17, m = 0.36; /* оценить по статистике */
           /* double C2000= 16.605251 ; оценка основных
            * фондов на начало 2000г: здесь являются
            * параметром par[10] */
        int i[12]={0};
          /* индекс цикла в последовательной программе */
        int i6=0, i7=0, i8=0, i9=0, i10=0, i11=0;
        double par[12]; /* обозначение параметров в цикле */

        int NN, iiii=0, iii=0;
        int N[12] = {8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8};
                /* число шагов в цикле */
        double dwn[12] = {0.119, 5.846, 0.650,
          0.9689, 0.0000000098, 0.0395, 0.01532,
```

```
          0.00279, 0.099, 27.05, 2.89, 95.9};
/* нижние границы параметров */
         double top[12] = {0.123, 5.852, 0.653,
          0.9694, 0.0000000102, 0.0402, 0.01537,
          0.00283, 0.101, 27.11, 2.91, 96.2};
/* верхние границы параметров */
         /* B0, gamma_f, alpha_f, a_f, A_f, mu_A,
          * mu_B, alpha_A, alpha_B, C2000, k, R0 –
          * внешние названия параметров */

double step[12]; /* - шаг по каждому интервалу перебора */
         for (a = 0; a < 12; a++)
                  step[a] = (top[a] - dwn[a]) / (N[a] - 1);

         for (a = 0; a < NW; a++)
                  src.R_stat[a] = workdays[a] * employment[a] * 4
                           * 8 / 1000;
                     /* труд, в млрд чел-час в год
                      * (4 квартала, 8 часов в день, /1000
                      * перевод млн в млрд) */

 /* сложение по 3 элемента из массива J_src
  * размерности 119, начиная с 13 элемента и
  * вывод результата в src.J_stat: */
Scale(3, J_src, 119, 13, src.J_stat);

         for (a = 0; a < NW; a++)
          src.J_stat[a] = src.J_stat[a] / 3 / 1000;
 /* Валовые инвестиции в основной капитал, трлн руб 2000
  * (поквартально с 1 кв 1995, в пересчете на год) */
          GDPre00[20] = 1.0;
          /* реальный ВВП в I кв 2000 = 100% */
         for (a = 0; a < 15; a++)
            GDPre00[21 + a] = tmpre00[a] / 100.0;
             /* использование имеющихся данных по re2000 */

         for (a = 0; a < 20; a++)
```

```
GDPre00[a] = GDPre95[a] / GDPre95[20];
/* пересчет re2000 через re1995 */
         for (a = 0; a < NW; a++) {
          src.Y_stat[a] = GDPre00[a] * GDPnom[20];
            /* Y - реальный ВВП, в ценах I кв 2000г */
          src.p_stat[a] = GDPnom[a] / src.Y_stat[a];
            /* p - индекс цен, приравнен дефлятору ВВП */
         }

         for (a = 11; a < NW-2; a++) {
          src.L_stat[a] = tmpCred[a-11] / 1000.0;
/* Объем кредитов всем предприятиям
 * с IV кв 1997 г по II-й 2003,
 * трлн руб */
          if (a>11)
                  src.dLdt_stat[a] = (src.L_stat[a] –
                          src.L_stat[a-1]) / dt;
         }

         /* сложение по 3 элемента из массива r_src
          * размерности 107, начиная с 1 элемента и
 * вывод результата в src.rl_stat */
Scale(3, r_src, 107, 1, src.rl_stat);
         for (a = 0; a < NW; a++)
src.rl_stat[a] *= 1.0/3.0/100.0;
  /* r[l]stat - ставка по кредиту */

         /* сложение по 3 элемента из массива s_src
          * размерности 108, начиная с 1 элемента и
          * вывод результата в src.s_stat */
Scale(3, s_src, 108, 1, src.s_stat);
         for (a = 0; a < NW; a++) {
          src.s_stat[a] *= (12.0 / 3) / (8.0*4*workdays[a]) / 1000;
            /*  s, тыс руб на человеко-час: 12 месяцев в
             * году, 8 час раб день, 4 квартала в году */
/********************************************/
          for (a = 0; a < NW; a++) {/* заданы по статистике: */
```

```
            J[a] = src.J_stat[a];   /* инвестиции */
            rl[a] = src.rl_stat[a]; /* процент по кредитам */
            p[a] = src.p_stat[a];  /* индекс цен */
            src.Y_stat[a] *= 4.0/1000.0;/* ВВП */
            src.L_stat[a] /=  src.p_stat[a];
                    /* задолженность в постоянных ценах */
            src.s_stat[a] /=  src.p_stat[a];
                    /* зарплата в постоянных ценах */
        }

/* подготовка начальных данных для перебора: */
NN = 1;
for(a = 0; a < 6; a++) NN *= N[a];
        CRIT = 0.0;

        for (iii=0; iii < NN; iii++){
            /* цикл перебора по первым 6 параметрам,
             * который требуется распараллелить */
          fun(iii, N, i);
          /* далее циклы по оставшимся 6 параметрам: */
          for (i[11] = 0; i[11] < N[11]; i[11]++) {
          for (i[10] = 0; i[10] < N[10]; i[10]++) {
          for (i[9] = 0; i[9] < N[9]; i[9]++) {
          for (i[8] = 0; i[8] < N[8]; i[8]++) {
          for (i[7] = 0; i[7] < N[7]; i[7]++) {
          for (i[6] = 0; i[6] < N[6]; i[6]++) {

            for(a = 0; a < 12; a++)
                    par[a] = dwn[a] + step[a] * i[a];
 /* определение параметров по индексам циклов */

/*        основные фонды в трлн руб 2000 г: */
 src.C_stat[19] = par[9] / src.p_stat[19]; /* IV квартал 1999 */
 src.C_stat[23] = src.C_stat[19]*C_proc[23]/100;
 src.C_stat[27] = src.C_stat[23]*C_proc[27]/100;
 src.C_stat[31] = src.C_stat[27]*C_proc[31]/100;
 src.C_stat[15] = src.C_stat[19]*100/C_proc[19];
```

```
src.C_stat[11] = src.C_stat[15]*100/C_proc[15];
src.C_stat[7] = src.C_stat[11]*100/C_proc[11];
src.C_stat[3] = src.C_stat[7]*100/C_proc[7];
C0 = src.C_stat[3] * 100 / C_proc[3];

a = 3;
for (j = 0; j < 8; j++){
        C_tmp1[j] = src.C_stat[a] * p[a]; /* в текущих ценах */
        a+=4;
}

        A0 = C0 - par[0]; /* начальное значение */
        for (a = 0; a < NW; a++) {
          if(a == 0) {
                A[a] = A0 * (1.0 - dt * par[5]);
        /* старый капитал только изнашивается */
                B[a] = par[0]*(1.0-dt*par[6])+dt*J[a];
        /* новый капитал и прирастает и изнашивается */
          } else {
                A[a] = A[a-1] * (1.0 - dt * par[5]);
                B[a] = B[a-1]*(1.0-dt*par[6])+dt*J[a];
          }
          L[a] = (par[8]*B[a]+par[7]*A[a]); /* задолженность */
          R[a] = par[10]*(src.R_stat[a]-par[11]); /* труд */
          C[a] = A[a] + B[a];
                /* капитал - из двух частей: старого и нового */
          x[a] = R[a] / C[a]; /* x(t) */
          f[a] = par[4]*exp((par[1]/par[2])*log(par[3]
                +(1–par[3])*exp(par[2]*log(x[a])))); /* f(x(t))*/
          Y[a] = f[a]*exp(par[1]*log(C[a])); /* ВВП */
          Dfdx[a] = (1-par[3])*par[1]*f[a]/x[a]
                 / (1–par[3]*(1-exp((-par[2])*log(x[a]))));
        /* производная производственной функции */
          omega[a] = (1-n)*Dfdx[a]*exp((par[1]-1)* log(C[a]));
                /* зарплата omega */
          s[a] = omega[a] / (1 + m) ; /* зарплата s */
        }
```

```
YLC1 = LinearCorrelation(src.Y_stat,Y,0,NW-1);
        /* коэффициент корреляции для реального ВВП Y */
YTI1 = TheilInequality(src.Y_stat, Y, 0, NW-1);
        /* индекс Тейла для реального ВВП Y */

a = 3;
for (j=0; j<8; j++){
        C_tmp2[j] = C[a] * p[a];
        a+=4;
}
CLC1 = LinearCorrelation(C_tmp1, C_tmp2, 0, 7);
        /* коэффициент корреляции для капитала C */
CTI1 = TheilInequality(C_tmp1, C_tmp2, 0, 7);
        /* индекс расхождения Тейла для капитала C */

LLC1 = LinearCorrelation(src.L_stat,L,11,NW-3);
        /* коэффициент корреляции для задолженности L */
LTI1 = TheilInequality(src.L_stat,L,11,NW-3);
        /* индекс Тейла для задолженности L */

sLC1 = LinearCorrelation(src.s_stat,s,0,NW-1);
        /* коэффициент корреляции для зарплаты s */
sTI1 = TheilInequality(src.s_stat, s, 0, NW-1);
        /* индекс расхождения Тейла для зарплаты s */

/* Среднегеометрическое значение коэффициентов
 * корреляции и Тейла и свертка критериев = корр / Тейл: */
if ((Min4(YLC1,CLC1,LLC1,sLC1) > 0.7)&&
  ((-Min4(-YTI1,-CTI1,-LTI1,-sTI1)) < 0.15)){
  LC1 = exp(log(YLC1*CLC1*LLC1*sLC1)/4.0);
  TI1 = exp(log(YTI1*CTI1*LTI1*sTI1)/4.0);
  if (LC1 / TI1 > CRIT) {
   CRIT = LC1 / TI1;
   iiii = iii;
   i6 = i[6];
   i7 = i[7];
   i8 = i[8];
```

```
  i9 = i[9];
  i10 = i[10];
  i11 = i[11];
  printf("Y:  %.8f %.8f\n",YLC1,YTI1);
  printf("C:  %.8f %.8f\n",CLC1,CTI1);
  printf("L:  %.8f %.8f\n",LLC1,LTI1);
  printf("s:  %.8f %.8f\n",sLC1,sTI1);
  printf("All %.8f %.8f %.8f \n",LC1,TI1,CRIT);
  printf(" B0=%.8f gamma_f=%.8f alpha_f=%.8f a_f=%.8f
         A_f=%.8f mu_A=%.8f mu_B=%.8f alpha_A=%.8f
         alpha_B=%.8f C2000=%.8f k=%.8f R0=%.8f\n",
         par[0], par[1], par[2], par[3], par[4], par[5], par[6], par[7],
         par[8], par[9], par[10], par[11]);
  printf("i: ");
  printf("%i ", iiii);
  for (a = 0; a < 12; a++) printf("%i ", i[a]);
  printf("\n");
 }
}
 }// i[6]
 }// i[7]
 }// i[8]
 }// i[9]
 }// i[10]
 }// i[11]
}

/* далее идет вывод данных */

fun(iiii, N, i);

 for(a = 0; a < 12; a++)
         par[a] = dwn[a] + step[a] * i[a];

src.C_stat[19] = par[9] / src.p_stat[19]; /* IV квартал 1999 */
src.C_stat[23] = src.C_stat[19]*C_proc[23]/100;
src.C_stat[27] = src.C_stat[23]*C_proc[27]/100;
```

```
src.C_stat[31] = src.C_stat[27]*C_proc[31]/100;
src.C_stat[15] = src.C_stat[19]*100/C_proc[19];
src.C_stat[11] = src.C_stat[15]*100/C_proc[15];
src.C_stat[7] = src.C_stat[11]*100/C_proc[11];
src.C_stat[3] = src.C_stat[7]*100/C_proc[7];
C0 = src.C_stat[3] * 100 / C_proc[3];

a = 3;
for (j=0; j<8; j++){
        C_tmp1[j] = src.C_stat[a] * p[a];
        a+=4;
}

A0 = C0 - par[0]; /* начальное значение */
for (a = 0; a < NW; a++) {
        if(a == 0) {
          A[a] = A0 * (1.0 - dt * par[5]);
          B[a] = par[0]*(1.0-dt*par[6])+dt*J[a];
          L[a] = (par[8]*B[a]+par[7]*A[a])*p[a];
          dLdt[a] = 0.0; /* dL/dt */
          iota[a] = 0.0; /* инфляция */
        } else {
          A[a] = A[a-1] * (1.0 - dt * par[5]);
          B[a] = B[a-1]*(1.0-dt *par[6])+dt*J[a];
          L[a] = (par[8] * B[a] + par[7] * A[a]);
          dLdt[a] = (L[a] - L[a-1]) / dt;
           iota[a] = (p[a] / p[a-1] - 1.0) / dt;
        }
        R[a] = par[10] * (src.R_stat[a] - par[11]);
        C[a] = A[a] + B[a];
        x[a] = R[a] / C[a]; /* x(t) */
        f[a] = par[4] * exp((par[1] / par[2]) * log(par[3]
                + (1 - par[3]) * exp(par[2] * log(x[a]))));
        Y[a] = f[a] * exp(par[1] * log(C[a]));
        Dfdx[a] = (1 - par[3]) * par[1] * f[a] / x[a] / (1 – par[3] *
                 (1 - exp((-par[2]) * log(x[a]))));
        omega[a] = (1 - n) * Dfdx[a] * exp((par[1] - 1)
```

```
                * log(C[a]));
        s[a] = omega[a]  / (1 + m) ;
        rho[a] = -(iota[a] - par[6]) / (-1 + par[8]) - par[8] *
                 (-1 + n) / (-1 + par[8]) * rl[a] +
                omega[a] / (-1 + par[8]) * x[a] +
                par[1] * exp((par[1] - 1) * log(C[a]))
                * (-1 + n) / (-1 + par[8]) * f[a]; /* rho */
        cond[a] = 0 <= (-1 + n) * rl[a] + rho[a];
        Z[a] = dLdt[a] * p[a] - (1 - n) * rl[a] * L[a] * p[a] +
                p[a] * (-J[a] - omega[a] * C[a] * x[a] +
                 (exp(par[1] * log(C[a]))) * (1 - n) * f[a]);
         }

YLC1 = LinearCorrelation(src.Y_stat,Y,0,NW-1);
YTI1 = TheilInequality(src.Y_stat, Y, 0, NW-1);

a = 3;
for (j=0; j<8; j++){
        C_tmp2[j] = C[a] * p[a];
        a+=4;
}
CLC1 = LinearCorrelation(C_tmp1, C_tmp2, 0, 7);
CTI1 = TheilInequality(C_tmp1, C_tmp2, 0, 7);
LLC1 = LinearCorrelation(src.L_stat,L,11,NW-3);
LTI1 = TheilInequality(src.L_stat,L,11,NW-3);
sLC1 = LinearCorrelation(src.s_stat,s,0,NW-1);
sTI1 = TheilInequality(src.s_stat, s, 0, NW-1);

/* Среднегеометрические значения коэффициентов
 * корреляции и Тейла и критерий = корр / Тейл: */
LC1 = exp(log(YLC1*CLC1*LLC1*sLC1) / 4.0);
TI1 = exp(log(YTI1*CTI1*LTI1*sTI1) / 4.0);
CRIT = LC1 / TI1;
printf("Y:  %.8f %.8f\n",YLC1,YTI1);
printf("C:  %.8f %.8f\n",CLC1,CTI1);
printf("L:  %.8f %.8f\n",LLC1,LTI1);
printf("s:  %.8f %.8f\n",sLC1,sTI1);
```

```
 printf("CRIT %.8f %.8f %.8f \n",LC1,TI1,CRIT);
 printf(" B0=%.8f gamma_f=%.8f alpha_f=%.8f
  a_f=%.8f A_f=%.8f mu_A=%.8f mu_B=%.8f
  alpha_A=%.8f alpha_B=%.8f C2000=%.8f k=%.8f
  R0=%.8f\n" , par[0], par[1], par[2], par[3],
  par[4], par[5], par[6], par[7], par[8],
  par[9], par[10], par[11]);
 printf("i: ");
 printf("%i ", iiii);
 for (a = 0; a < 12; a++) printf(" %i", i[a]);
 printf("\n");

 /* Запись выходных данных в файл opt.dat */
if ((fopt = fopen("opt.dat", "w")) == NULL)
  printf("File opt.dat could not be opend\n");
else {
  for (a = 0; a < NW; a++) {/* расчет номинальных величин: */
          YY[a] = Y[a] * src.p_stat[a];
          YY_s[a] = src.Y_stat[a] * src.p_stat[a];
          CC[a] = C[a] * src.p_stat[a];
          CC_s[a] = src.C_stat[a] * src.p_stat[a];
          LL[a] = L[a] * src.p_stat[a];
          LL_s[a] = src.L_stat[a] * src.p_stat[a];
          ss[a] = s[a] * src.p_stat[a];
          ss_s[a] = src.s_stat[a] * src.p_stat[a];
          s_re[a] = s[a] * ((1.0 + m) * (1.0 –
     src.R_stat[a]/R[a]) + src.R_stat[a]/R[a]);
/*факт. зарпл. на 1 млрд чел-час после налог. */
          s_re1[a] = s[a] * (1.0 + (1.0 + m) * (R[a]
                   / src.R_stat[a] - 1.0));
/* факт. зарплата на 1 млрд учтенных в статистике
 * человеко-часов после налогообложения */
          Z_s1[a] = (1.0 - n) * Y[a] - (1.0 + m) *
                            src.s_stat[a] * src.R_stat[a];
/* прибыль Z статистическая под тильдой */
          n_re = n; /* надо бы оценить !!!!! */
          Z_re1[a] = (1.0 - n_re) * Y[a] - (1.0 + m)
```

```
                * src.s_stat[a] * R[a];
/* прибыль Z фактическая под тильдой */
        m_nonpay[a] = s[a] * (1.0 + m) * (R[a] –
                src.R_stat[a]); /* недоплата ЕСН*/
        n_nonpay[a] = (n - n_re) * Y[a];
/* оценка недоплаты НДС: надо оценить Z_stat и n_re !!!!! */

        fprintf(fopt,
"%.9f\t%.9f\t%.9f\t%.9f\t%.9f\t%.9f\t%.9f\t%.9f\
t%.9f\t%.9f\t%.9f\t%.9f\t%.9f\t%.9f\t%.9f\t%.9f\
t%.9f\t%.9f\t%.9f\t%.9f\t%.9f\t%.9f\t%.9f\t%.9f\
t%.9f\t%.9f\t%.9f\t%.9f\t%.9f\t%.9f\t%.9f\t%.9f\
t%.9f\t%.9f\t%.9f\n", src.J_stat[a],
src.rl_stat[a], src.p_stat[a], src.R_stat[a],
src.Y_stat[a], src.L_stat[a], src.C_stat[a],
src.s_stat[a], Y[a], L[a], C[a], A[a], B[a],
x[a], f[a], iota[a], Dfdx[a], omega[a], s[a],
rho[a], Z[a], YY[a], YY_s[a], CC[a], CC_s[a],
LL[a], LL_s[a], ss[a], ss_s[a], s_re[a],
s_re1[a], Z_s1[a], Z_re1[a], m_nonpay[a],
n_nonpay[a]);
 };
        for (a = 0; a < 12; a++)
                fprintf(fopt,"%i\t", i[a]);
        fprintf(fopt,"%i\n", iiii);
        for (a = 0; a < 12; a++)
                fprintf(fopt,"%.8f\t", par[a]);
        fprintf(fopt, "\n");
        fprintf(fopt, "Y CT:\t  %.8f\t %.8f\n", YLC1,YTI1);
        fprintf(fopt, "C CT:\t  %.8f\t %.8f\n", CLC1,CTI1);
        fprintf(fopt, "L CT:\t  %.8f\t %.8f\n", LLC1,LTI1);
        fprintf(fopt, "s CT:\t  %.8f\t %.8f\n", sLC1,sTI1);
        fprintf(fopt, "CRIT\t %.8f\t %.8f\t %.8f\n",
                LC1,TI1,CRIT);
} fclose(fopt); /* 2 opt.dat */
  return 0;
} /* end main */
```

```
/***** Functions:***** /
void Scale(int num, double vec[], int dim1,
  int dim0, double res[]){
   /* функция сложения по num элементов из массива
    * vec размерности dim, начиная с dim0 = 1...dim1
    * и вывод в массив res */

 int i, j; /* n = (dim - dim0) / num;
          * if (dim - dim0 > n*num) {n++;} */
 double tmp1;
 j = 0;
 tmp1 = 0.0;

 for (i = dim0; i <= dim1; i++) {
tmp1 = tmp1 + vec[i - 1];
if ((i + 1 - dim0) / num == j+1) {
                    res[j] = tmp1; j++;
                    tmp1 = 0.0;
          }
 }
 if (dim1 - dim0 > num*j)
  res[j] = num*tmp1/(dim1 + 1 - dim0 - num *j);
} /* end Scale */

double LinearCorrelation(double x_x[],
double y_y[], int n_1, int n_2){
/* Given two arrays x_x[n_1..n_2] and y_y[n_1..n_2],
 * this routine computes their Pearson Correlation Coefficient */
  int j;
  double yt,xt;
  double syy=0.0,sxy=0.0,sxx=0.0,ay=0.0,ax=0.0;

  for (j=n_1;j<=n_2;j++) { /* Find the means: */
          ax += x_x[j];
          ay += y_y[j];
  }
  ax /= n_2 - n_1 + 1;
```

```
  ay /= n_2 - n_1 + 1;
  for (j=n_1;j<=n_2;j++) {
/* Compute the correlation coefficient: */
          xt=x_x[j]-ax;
          yt=y_y[j]-ay;
          sxx += xt*xt;
          syy += yt*yt;
          sxy += xt*yt;
  }
  return sxy / (sqrt(sxx * syy) + TINY);
} /* end LinearCorrelation */

double TheilInequality(double x_x[],
double y_y[], int n_1, int n_2){
 /* Given two arrays x_x[n_1..n_2] and y_y[n_1..n_2],
  * this routine computes their Theil Inequality Index */
int j;
 double xmy2=0.0,x2py2=0.0;
 for (j=n_1;j<=n_2;j++) { /* Find the means: */
  xmy2 += (x_x[j] - y_y[j]) * (x_x[j] - y_y[j]);
  x2py2 += x_x[j] * x_x[j] + y_y[j] * y_y[j];
 }
 return sqrt(xmy2) / (sqrt(x2py2) + TINY);
} /* end TheilInequality */

double Min4(double x_1, double x_2, double x_3, double x_4){
          /* this routine computes minimum of 4 numbers */
          double x = x_1;
          if (x_2 < x) x = x_2;
          if (x_3 < x) x = x_3;
          if (x_4 < x) x = x_4;
          return x;
} /* end Min4 */

void fun(int iii, int N[], int i[]){
  int iii1;
  int j;
```

```
iii1 = iii;
 for(j = 0 ; j < 6; j++){
  i[j] = iii1 % N[j]; /* остаток от деления */
  iii1 = (iii1 - i[j]) / N[j];
 }
} /* end fun */
```

Модели экономики страны обычно содержат большое число параметров, которые не удается определить напрямую по данным экономической статистики. Поэтому неизвестные параметры модели экономики определяют косвенным образом, сравнивая временные ряды выходных переменных модели с известными статистическими временными рядами. Процедура определения параметров называется идентификацией.

В качестве критериев близости расчетного X_t и статистического Y_t временных рядов некоторого показателя удобно использовать коэффициент корреляции Пирсона $P \in [-1,1]$ и индекс несовпадения Тэйла $U \in [0,1]$. Коэффициент корреляции P 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 расчетные и статистические временные ряды показателя нужно брать в реальных величинах. Если через $\bar{X}$ и $\bar{Y}$ обозначить средние значения для соответствующих рядов, то коэффициент корреляции Пирсона

$$P = \frac{\sum_t \left(X_t - \bar{X}\right)\left(Y_t - \bar{Y}\right)}{\sqrt{\sum_t \left(X_t - \bar{X}\right)^2 \cdot \sum_t \left(Y_t - \bar{Y}\right)^2}}. \quad (1)$$

Индекс Тэйла U измеряет несовпадение статистических и расчетных временных рядов X_t и Y_t некоего показателя, и чем ближе этот индекс к нулю, тем ближе сравниваемые ряды.

$$U = \frac{\sqrt{\sum_t \left(X_t - Y_t \right)^2}}{\sqrt{\sum_t \left(X_t^2 + Y_t^2 \right)}}. \tag{2}$$

Поскольку параметров 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 идентифицированных и верифицированных блоков модели, либо на основе статистических данных.

Для однозначности выбора оптимального варианта можно использовать ту или иную свертку коэффициентов корреляции и индексов Тэйла, например, если близость расчетных и статистических данных для всех макропоказателей имеет примерно равную важность, можно максимизировать отношение среднегеометрической корреляции к среднегеометрическому коэффициенту Тэйла.

В формальной записи

$$K(\vec{a}) \to \max_{\vec{a} \in D}, \tag{3}$$

где

$$D = \left\{ \vec{a} \in R^N : a_i^- \le a_i \le a_i^+, 1 \le i \le N \right\}, \tag{4}$$

$$K(\vec{a}) = \sqrt[2m]{\prod_{j=1}^{m} \frac{P_j(\vec{a})}{U_j(\vec{a})}}, \quad (5)$$

Здесь

m – число макропоказателей;

j - индекс макропоказателя, $j = 1, ..., m$;

$2m$ - общее число критериев (корреляции и Тейла).

При этом следует перебирать только те варианты значений параметров, при которых коэффициенты корреляции выше некоторой заданной положительной величины, например, $P_j > 0.85$, а индексы Тейла – ниже некоторой другой, например, $U_j < 0.15$ ($j = 1, ..., m$).

В качестве примера задачи идентификации параметров модели экономики рассмотрим задачу идентификации параметров производственного блока в модели экономики России, разрабатываемой в ВЦ РАН для оценки динамики теневого оборота в 1995-2003 гг.

В модели экономики России периода 1995-2003 гг. в качестве производственной функции, описывающей в каждый момент времени t зависимость валового внутреннего продукта Y от количеств используемых производственных факторов: труда R и капитала C, - была взята однородная степени γ CES – функция

$$Y(t) = C^{\gamma}(t)f(x), \quad f(x) = \delta[\alpha + (1 - \alpha)x^{\beta}]^{\gamma/\beta}, \quad x = R(t) / C(t), \quad (6)$$

в которой параметры удовлетворяют неравенствам

$$0 < \alpha < 1, \beta > 0, \gamma > 1, \delta > 0. \quad (7)$$

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 убывает с течением времени

$$dA/dt = -\mu_A A(t), \tag{8}$$

и «нового капитала» *B,* который растет за счет введения в строй новых производственных фондов *J(t)* и также может убывать

$$dB/dt = J(t) - \mu_B B(t), \tag{9}$$

так что

$$C(t) = A(t) + B(t). \tag{10}$$

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 повторяющие колебания ВВП.

В блоке «производство» механизмы регулирования производства описывались моделью поведения экономического агента, которого логично назвать производителем. Агент получает

доходы от 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.

Объем банковских ссуд L ограничивался основным капиталом в соответствии с его органическим строением:

$$L(t) \leq (\alpha_A A + \alpha_B B)p(t), \tag{11}$$

где индекс цен на основные фонды $p(t)$ считался заданным в данном блоке, а параметры $\alpha_A > 0,\ \alpha_B > 0$ определялись.

Изменение банковских ссуд $L(t)$ и расчетного счета $N(t)$ определялись из баланса доходов и расходов.

$$p(t)Y(t) - s(t)R(t) - r(t)L(t) + \frac{dL}{dt} - Z(t) - Tax(t) - p(t)J(t) - \frac{dN}{dt} = 0 \tag{12}$$

$$N(t) \geq 0. \tag{13}$$

Для простоты налоговые отчисления в консолидированный бюджет задавались двумя видами налогов: налогом на добавленную стоимость n (НДС), включающим налог на прибыль, и единым социальным налогом m (ЕСН), включающим подоходный налог.

$$\mathrm{Tax}(t) = n\,(\mathrm{pY}(t) - \mathrm{rL}(t)) + m\,\mathrm{sR}(t) \tag{14}$$

Последовательный вариант программы идентификации параметров производственного блока модели содержится в файле pf.c. Это совпадает с максимизацией дисконтированных доходов, причем коэффициент дисконтирования совпадает с

соответствующей двойственной оценкой роста активов, возникающей в рассматриваемой задаче оптимального управления.

Решение задачи оптимального управления производителя определяет загрузку производственных фондов трудом, коэффициент дисконтирования, ставку заработной платы, ВВП, дивиденды, объем банковских ссуд, остаток расчетного счета.

$$F := (C, R) \to C^{\gamma_f} \mathrm{f}\left(\frac{R}{C}\right)$$

$$\frac{d}{dt} p_y(t) = \iota(t)\, p_y(t)$$

$$s_r(t) = \frac{\omega(t)\, p_y(t)}{1 + m}$$

$$\{ 0 \le (-1 + n)\, r_l(t) + \rho(t) \}$$

$$\mathrm{L}(t) = (\alpha_B\, \mathrm{B}(t) + \alpha_A\, \mathrm{A}(t))\, p_y(t)$$

$$\frac{d}{dt} \mathrm{A}(t) = -\mu_a\, \mathrm{A}(t)$$

$$\frac{d}{dt} \mathrm{B}(t) = \mathrm{J}(t) - \mu_b\, \mathrm{B}(t)$$

$$0 = \mathrm{C}(t) - \mathrm{A}(t) - \mathrm{B}(t)$$

$$\mathrm{Z}(t) = (-\mathrm{J}(t) - \omega(t)\, \mathrm{C}(t)\, \mathrm{x}(t) - \mathrm{C}(t)^{\gamma_f} (-1 + n)\, \mathrm{f}(\mathrm{x}(t)))\, p_y(t) + r_l(t)\, (-1 + n)\, \mathrm{L}(t) + \left(\frac{d}{dt} \mathrm{L}(t)\right)$$

$$\mathrm{D}(f)(\mathrm{x}(t)) = -\frac{\omega(t)\, \mathrm{C}(t)^{(-\gamma_f + 1)}}{-1 + n}$$

$$\rho(t) = -\frac{\iota(t) - \mu_b}{-1 + \alpha_B} - \frac{\alpha_B(-1 + n)\, r_l(t)}{-1 + \alpha_B} + \frac{\omega(t)\, x(t)}{-1 + \alpha_B} + \frac{\gamma_f\, C(t)^{(\gamma_f - 1)}(-1 + n)\, f(x(t))}{-1 + \alpha_B}$$

На основе сравнения показателей, рассчитанных по предложенной модели, и соответствующих статистических показателей в [7] было показано, что фактический труд *R*, измеренный в миллиардах человеко-часов, имеет большую амплитуду колебаний, чем статистический труд R_s, измеренный в тех же единицах. Так что

$$R = k\,(R_s - R_0). \quad (15)$$

Возможная интерпретация (15):

- R_0 - “балласт”, труд, учитываемый в статистике, но не приносящий добавленной стоимости, $R_0 < R_s$.
- $k > 1$ – коэффициент фактической занятости оставшихся. Другими словами, те занятые, что приносят добавленную стоимость, работают не 8 часов в день, а в *k* раз больше.

Требуется определить 12 параметров модели (α, β, γ, δ, μ_A, μ_B, α_A, α_B, $B(0)$, $C(0)$, k, R_0), возможные значения каждого из которых находятся на некотором интервале.

Оптимальные значения параметров определяются косвенным образом, на основе максимизации свертки критериев Тэйла и коэффициентов корреляции (3), в которых сравниваются рассчитанные по модели временные ряды и статистические временные ряды для следующих показателей: выпуска Y, капитала C, объема банковских ссуд L и ставке заработной платы s. В данном блоке считается, что труд R, индекс цен p, капиталовложения J известны: эти величины здесь заданы их статистическими значениями.

Вычисление параметров основано на полном переборе значений искомых параметров в заданных интервалах их изменения. Для уточнения решения полный перебор может повторяться несколько раз с последовательным уменьшением интервалов изменения переменных.

Представленная выше последовательная программа полностью подготовлена к распараллеливанию (сделаны все предварительные шаги). Пользуясь полученными знаниями по основам MPI, требуется только распараллелить объединенный цикл перебора по первым 6 параметрам:

```
for (iii=0; iii < NN; iii++){  …
}
```

4.7. Очистка

После того, как завершится работа ваших программ, а ваши решения упражнений будут предъявлены и сданы преподавателю, не забудьте очистить ваш рабочий каталог на кластерном компьютере ВЦ РАН. Исходные тексты программ, при желании, можно хранить на сервере Fast в вашем каталоге.

Литература

1. Cornell Theory Center. Basics of MPI Programming (http://www.tc.cornell.edu/services/edu/topics/topics.asp?section=mpi).
2. Богачёв К.Ю. Основы параллельного программирования. - М.: БИНОМ. Лаборатория знаний, 2003. - 342 с.
3. Домашняя страница MPI в Argonne National Labs (http:// www-unix.mcs.anl.gov/mpi/)
4. Часто задаваемые вопросы по MPI (http://www.faqs.org/faqs/mpi-faq/)
5. RS/6000 SP: Practical MPI Programming (http://www.redbooks.ibm.com/abstracts/sg245380.html?Open)

6. MPICH документация в прямом доступе
(http://www. cluster.bsu.by/MPI_ALL.htm)
7. Оленев Н.Н. Параллельные вычисления для идентификации параметров в моделях экономики // Высокопроизводительные параллельные вычисления на кластерных системах. Материалы четвертого Международного научно-практического семинара и Всероссийской молодежной школы. / Под ред. чл.-корр. РАН В.А.Сойфера, Самара, 2004. – 280 с. С.204-209. (http://www.ccas.ru/olenev/tezisy.pdf)

Приложение 1. Краткий глоссарий

ANSI:

(American National Standards Institute)
Национальный институт стандартизации США.

Express:

Язык, поддерживающий параллельность посредством передачи сообщений, называвшийся очередями сообщений. Поддерживался корпорацией ParaSoft. См. ftp://ftp.parasoft.com/
express/docs/.

HPF:

(High Performance Fortran)
Высокоскоростной Фортран, расширение к Фортран 77 или 90, которое обеспечивает: возможности параллельного исполнения при автоматическом обнаружении компилятором; различные типы имеющегося параллелизма - MIMD, SIMD, или некоторую их комбинацию; распределение данных в память индивидуальных процессоров и размещение данных внутри единственного процессора.

I/O:

(Input/Output)
Вход/Выход – механизмы вычислительной техники и программного обеспечения, связывающие компьютер с

«внешним миром». Они включают связь компьютера с диском и компьютера с терминалом / сетью / графикой. Стандартный I/O есть специфический пакет программ, используемый языком C.

ISO:

(International Organization for Standardization) Международная организация стандартизации.

MIMD:

(Multiple Instruction stream, Multiple Data stream) - несколько потоков команд и несколько потоков данных - архитектура, в которой несколько потоков команд выполняются одновременно. Каждая единственная команда может распоряжаться несколькими элементами данных (например, одним или более векторами на векторной машине). Хотя одно-процессорный векторный компьютер способен работать по методу MIMD из-за пересекающихся функциональных единиц, терминологию MIMD используют более обще относя ее к многопроцессорным машинам. См. также SIMD, SISD.

P4:

Пакет макро/подпрограмм для параллельного программирования, разработанный Rusty Lusk lusk@anta.mcs.anl.gov. P4 использует мониторы на машинах с разделяемой (коллективной) памятью и передачу сообщений на машинах с распределенной памятью. Он употребляется как библиотека подпрограмм для C и Fortran. Улучшение "Argonne macros" (PARMACS).

PVM:

(Parallel Virtual Machine) - Параллельная виртуальная машина, библиотека передачи сообщений и множество средств, используемых, чтобы создать и исполнить одновременные или параллельные приложения.

SIMD:

(Single Instruction stream, Multiple Data stream) - один поток команд и несколько потоков данных - архитектура, которой характеризуется большая часть векторных компьютеров. Один поток команд запускает процесс, который устанавливает движение потоков данных и результатов. Этот термин также применим к параллельным процессорам, у которых один поток команд вызывает синхронное выполнение одной и той же операции на более чем одном процессоре, даже на различных кусках данных (например, ILLIAC). См. также MIMD, SISD, и SPMD.

SISD:

(Single Instruction stream, Single Data stream) - один поток команд и один поток данных, - архитектура традиционного компьютера, в котором каждая команда (из одного потока команд) имеет дело с точно определенными элементами данных или парой операндов скорее, чем с "потоками" данных. См. также SIMD и MIMD.

SPMD:

(Single Program, Multiple Data stream) - одна программа и несколько потоков данных - обобщение параллельного программирования данных SIMD, SPMD далее ослабляет обязательства по синхронизации, которые ограничивают функционирование различных процессов, и просто явно определяет, какую программу все процессы должны запустить, но не какую команду каждый вынужден выполнять в каждый отдельный момент времени. Распределение данных, тем не менее, является все еще ключевой концепцией; в реальности, другой, обычно используемый для SPMD, термин есть декомпозиция данных - это косвенно указывает на тот факт, что собираются декомпозировать всеобъемлющий набор данных, при выполнении на каждом участвующем процессе одного и того же кода.

Амдала закон (в параллельных вычислениях)

Пусть F - часть вычислений, которая является последовательной, а (1-F) – часть вычислений, которую можно запараллелить. Тогда максимальное ускорение, S, которое можно достичь, используя N параллельных процессоров:

S = (частичная параллельная скорость) /
/ (скорость одного процессора) = 1 / (F + (1 - F) / N).

архитектура:

Конструкция основных компонентов компьютера (hardware) и способов взаимодействия этих компонент, составляющих полную машину. Для параллельного процессора архитектура включает как топологию машины целиком, так и детальную конструкцию каждого узла.

баланс загрузки:

Цель алгоритмов, выполняющихся на параллельных процессах, которая достигается, если все узлы эффективно выполняют примерно равное количество работы, так что никакой узел не простаивает значительное количество времени.

массив:

Последовательность объектов, в которой порядок важен (что противопоставляется множеству, являющемуся группой объектов, в которой порядок не важен). Переменная массива в программе хранит n-мерную матрицу или вектор данных. Термин компьютерный массив также используется, чтобы описать множество узлов в параллельном процессоре, это термин подразумевает, но не требует, что этот процессор имеет геометрическую или матрично-подобную связность.

блокировка:

Действие подпрограмм коммуникации, которые ожидают, пока их функция не завершится до возвращения управления в вызывающую программу. Например, подпрограмма, которая отправляет сообщение, может задержать свое завершение до

тех пор пока она не получит подтверждение, что сообщение получено.

C:

Язык программирования, оригинально основанный на «B», созданный Дэнисом Ритчи. C – это язык низкого уровня, который имеет множество черт, обычно находимых в языках высокого уровня. C и C++ - это два наиболее общих языка программирования, используемых в наши дни.

C++:

Объектно-ориентированное расширение языка программирования C. C++ позволяет пользователю использовать классы абстрактных данных и другие продвинутые методы представления и манипуляции данными.

кластер, кластерная:

Тип архитектуры, состоящей из множества узлов, соединенных в сеть.

мастер-рабочий:

Программистский подход, в котором один процесс, названный "мастер", назначает задачи для других процессов, известных как "рабочие".

коммуникационная накладка:

Мера дополнительной рабочей нагрузки, которой подвержен параллельный алгоритм из-за коммуникации между узлами параллельного процессора. Если коммуникация – единственный источник накладки, то коммуникационная накладка задается формулой: ((число процессоров * параллельное время выполнения) - последовательное время выполнения) / последовательное время выполнения.

ЦПУ:

Центральное Программируемое Устройство, арифметическое и управляющие части последовательного компьютера.

ВЦ РАН:

Вычислительный центр им. А.А. Дородницына Российской академии наук

МСЦ:

Межведомственный суперкомпьютерный центр Российской академии наук

декомпозиция данных:

Способ разделения массивов между связанными ЦПУ, чтобы минимизировать коммуникацию (взаимосвязь).

параллельность данных:

Модель программирования, в которой каждый процессор выполняет одну и туже работу на уникальном сегменте данных. Какая-либо из библиотек передачи сообщений, такая как MPI, или язык высокого уровня, такой как HPF, могут быть использованы для кодирования по этой модели. Альтернативой к параллельности данных является функциональная параллельность.

передача сообщений:

Парадигма коммуникации, в которой процессы поддерживают связь посредством обмена сообщениями через коммуникационные каналы.

процесс:

Задача, исполняемая на данном процессоре в данное время.

процессор:

Часть компьютера, которая реально исполняет ваши команды. Также известна как центральное процессорное устройство или ЦПУ.

тупик (дэдлок):

Ситуация, в которой процессы параллельного процессора ожидают события, которое никогда не произойдет. Простая версия тупика для узко синхронизированного окружения возникает, когда блокирующие чтения и записи не являются корректно связанными. Например, если два узла оба выполняют блокирующие записи друг другу в одно и тоже

время, тупик возникает, так как никакая запись не может завершиться, пока дополнительное чтение не выполнится на другом узле.

пространство локального диска:

Дисковое пространство внутри данного узла. Например, на учетверенном узле один и тот же локальный диск достижим для всех процессоров узла (1-4).

распределенная память:

Память, которая разделена на сегменты, каждый из которых может быть напрямую доступен только одним узлом из параллельных процессоров. Распределенная память и совместная (коллективная) память есть две основных архитектуры, которые требуют очень разных стилей программирования.

распределенная обработка:

Обработка на ряде соединенных в сеть компьютеров, каждый из которых имеет локальную память. Компьютеры могут быть или могут не быть различной относительной мощности и функции.

распределенная совместная память (DSM):

Память, которая физически распределена, но это скрывается операционной системой, так что представляется пользователю как совместная память с одним адресным пространством. Также называют виртуальной совместной памятью.

синхронизация:

Акт приведения известных точек выполнения у двух или более процессов к одному и тому же заданному моменту времени. Явная синхронизация не нужна в программах с архитектурой SIMD (в которой каждый процессор выполняет ту же самую операцию, как и любой другой, или не делает ничего), но часто необходима в программах с архитектурами SPMD и MIMD. Время, затрачиваемое процессами в ожидании других процессов для синхронизации с ними, может быть основным источником неэффективности в параллельных программах.

сборка/размещение:

Операция коллективной коммуникации, в которой (для сборки - gather) один процесс собирает данные от каждого участвующего процесса и запоминает их в порядке номеров процессов или (для размещения - scatter) один процесс разделяет некоторые данные и распределяет кусок каждому участвующему процессу опять же в порядке номеров процессов.

узел:

Один из индивидуальных компьютеров, связанных вместе, чтобы сформировать параллельную систему. Компьютер может иметь множество процессоров, которые разделяют (совместно используют) систему ресурсов, таких как диск, память и сетевой интерфейс.

функции коллективной коммуникации:

Функции обмена сообщениями, которые обмениваются данными среди всех процессов в группе. Эти функции обычно включают функцию барьера для синхронизации, функцию рассылки (broadcast) для отправки данных от одного процессора всем процессорам, и функции сборки/разброса (gather/scatter).

эффективность:

Мера затрат времени параллельной программы на непосредственный счет, а не на коммуникации. Эффективность измеряется как ускорение, деленное на число процессоров. Чем ближе она к 1, тем более совершенно параллельной является задача на том уровне параллелизма, чем ближе к 0, тем менее параллельной.

Приложение 2. Шесть основных функций MPI

Инициализации процесса MPI
MPI_Init должна быть первой функцией MPI, которую Вы вызываете в каждом процессе. Её можно вызвать только один раз. Она устанавливает окружение необходимое для запуска MPI. Это окружение может быть приспособлено для любых флагов запуска MPI, обеспечиваемых реализацией MPI.

- int MPI_Init(int *argc, char ***argv)

Нахождение числа процессов
MPI_Comm_size возвращает число процессов внутри коммуникатора. Коммуникатор - это механизм MPI для создания отдельных коммуникационных областей ("вселенных") (подробнее об этом позже). Наша пробная программа использует предопределенный "мировой коммуникатор" MPI_COMM_WORLD, который включает все ваши процессы. MPI может определить число процессов, поскольку вы определяете это, когда задействуете команду mpirun, используемую для запуска программ MPI.

- int MPI_Comm_size(MPI_Comm comm, int *size)

Нахождение ранга процесса
Ранг (номер по-порядку) используется, чтобы специфицировать отдельный процесс. Ранг является целым в интервале от 0 до size - 1, где size есть число процессов, которое возвращает функция MPI_Comm_size. **MPI_Comm_rank** возвращает ранг вызываемого процесса в точно определенном коммуникаторе.

Часто для процесса необходимо знать его собственный ранг. Например, в случае, когда Вы желаете разбить вычислительную

работу в цикле по всем вашим процессам, так чтобы каждый процесс выполнял подмножество в исходном диапазоне цикла. Один из способов сделать это состоит в том, чтобы для каждого процесса использовать его ранг для указания его диапазона в индексах цикла.

- int MPI_Comm_rank(MPI_Comm comm, int *rank)

Позднее, когда мы изучим коммуникаторы, Вы увидите, что процесс может принадлежать более чем одному коммуникатору, и может иметь различный ранг для каждого коммуникатора. Но сейчас будем полагать, что мы имеем дело только с предопределенным коммуникатором MPI_COMM_WORLD, который состоит из всех ваших процессов, как указано в данном примере.

Отправка сообщения
В MPI существует множество всевозможных функций для отправки и получения. Это одна из причин того, что MPI содержит более чем 125 функциями. При рассмотрении шести основных функций мы рассмотрим только один тип отправки и один тип получения.

MPI_Send является блокирующей отправкой. Это означает, что вызов не возвращает управление в вашу программу до тех пор, пока все данные не будут скопированы из расположения, которое вы точно определили в листе параметров. Из-за этого, Вы можете изменить данные после вызова, что не отразится на оригинальном сообщении. (Существуют неблокирующие отправления, в которых это не так.)

- int MPI_Send(void *buf, int count, MPI_Datatype datatype, int dest, int tag, MPI_Comm comm)

Параметры:

- buf - начало буфера, содержащего данные, которые будут отосланы. Для C это - адрес.
- count - число элементов, которые будут отосланы (не байт)
- datatype - тип данных
- dest - ранг процесса, места назначения, для сообщения
- tag - произвольное число, которое можно использовать для отличия от других сообщений
- comm - (определенный) коммуникатор
- (ierror - возвращаемый функцией код ошибки).

В теоретической части модуля это обсуждается более детально.

Получение сообщения

Передача сообщения в MPI-1 требует двухстороннюю коммуникацию (связь). Одностороняя коммуникация является одной из черт, добавленной в MPI-2. Каждый раз, когда один процесс отправляет сообщение, другой процесс должен явно получить сообщение. Итак, для каждого места в вашем приложении, в котором Вы вызываете функцию MPI отправки, должно быть соответствующее место, где Вы вызываете функцию MPI получения. Следует проявить заботу, чтобы увериться в том, что параметры отправки и получения сочетаются. Это обсуждается более детально в этом модуле.

Подобно MPI_Send, **MPI_Recv** является блокирующей. Это означает, что вызов не возвращает управление в вашу программу до тех пор пока все полученные данные не будут запомнены в переменной (-ых), которые Вы точно определили в листе параметров. Из-за этого, Вы можете использовать эти данные после вызова функции и быть уверенным, что все они имеются. (Существуют неблокирующие получения, в которых это не так.)

- int MPI_Recv(void *buf, int count, MPI_Datatype datatype, int source, int tag, MPI_Comm comm, MPI_Status *status)

Параметры:

- buf - начало буфера, в котором входящие данные должны быть запомнены. Для С это - адрес.
- count - число элементов (не байт) в вашем буфере получателя
- datatype - тип данных
- source - ранг процесса, от которого данные будут приняты (Он может быть любым (джокером, дикой картой) при задании параметром MPI_ANY_SOURCE)
- tag - произвольное число, которое можно использовать для отличия от других сообщений (оно может быть любым (дикой картой) при задании параметром MPI_ANY_TAG)
- comm - (определенный) коммуникатор
- status - массив или структура возвращаемой информации. Например, если Вы определяете дикой картой источник source или тег tag, статус скажет Вам действительный ранг или тег для полученного сообщения
- (ierror - возвращаемый функцией код ошибки).

Выход из MPI

Последним вызовом, который Вам надлежит сделать в каждом процессе, является вызов **MPI_Finalize**. Этот вызов помогает удостовериться в том, что MPI выходит чисто, и что ничего не отложено на узлах, когда вы это делаете. Для того чтобы всё приложение завершить чисто, все ожидающие (неблокирующие) коммуникации следует завершить до вызова MPI_Finalize.

Отметим, что Ваш код может продолжать выполнение после вызова MPI_Finalize, однако он не может больше вызывать функции MPI.

- int MPI_Finalize()

Приложение 3. Запуск программ с MPI

Запуск на исполнение MPI-программы производится с помощью команды:

mpirun [параметры_mpirun...] <имя_программы>
[параметры_программы...] [-host]

Параметры команды mpirun слелующие:

-h

интерактивная подсказка по параметрам команды mpirun.

-maxtime <максимальное_время>

Максимальное время счета. От этого времени зависит положение задачи в очереди. После истечения этого времени задача принудительно заканчивается.

-np <число_процессоров>

Число процессоров, требуемое программе.

-quantum <значение_кванта_времени>

Этот параметр указывает, что задача является фоновой, и задает размер кванта для фоновой задачи.

-restart

Указание этого ключа приведет к тому, что после своего завершения задача будет вновь поставлена в очередь. Для удаления из очереди такой задачи пользуйтесь стандартной командой mqdel, а для ее завершения - командами mkill или mterm.

-stdiodir <имя_директории>

Этот параметр задает имя каталога стандартного ввода/вывода, в который будут записываться протокол запуска задачи, файл стандартного вывода и имена модулей, на которых запускалась задача.
-stdin <имя_файла>

Этот параметр задает имя файла, на который будет перенаправлен стандартный ввод задачи.

-stdout <имя_файла>

Этот параметр задает имя файла, на который будет перенаправлен стандартный вывод задачи.

-stderr <имя_файла>

Этот параметр задает имя файла, на который будет перенаправлен стандартный вывод сообщений об ошибках задачи.

-interactive

Задание этого ключа делает задачу интерактивной, а также отменяет действия ключей stdin, stdout, stderr.

-ldmname <имя_ресурса_ЛДП>

Этот параметр указывает на то, что задача будет использовать ресурс ЛДП с указанным именем. Если на момент запуска задачи ресурс ЛДП с указанным именем не существует, он будет создан временным, о чем пользователь извещается специальным сообщением, например:

Warning! LDM resource ldm does not exists, it will be created temporary

Для временного ресурса пользователь обязан задать размер требуемой дисковой памяти на каждом модуле (параметр ldmspace).

Для любых типов ресурсов (как разовых, создаваемых на время счета задачи, так и постоянных, т.е. уже выделенных на момент запуска задачи) требуется указание каталога монтирования (параметр mountdir).

Если в очереди или в счете у пользователя уже есть задача, использующая (или требующая) временный ресурс с таким же именем, то система выдаст сообщение об ошибке, и данная задача не будет поставлена в очередь.

-ldmspace <размер_ресурса_ЛДП_на_одном_модуле>

Этот параметр имеет действие только, если указано имя ресурса ЛДП (параметр ldmname) и ресурс ЛДП с заданным именем не существует (т.е. для задачи требуется разовый или временный ресурс ЛДП). Параметр ldmspace задает размер локальной дисковой памяти на одном модуле, требуемой под временный ресурс ЛДП. Локальная дисковая память заданного размера будет выделена на каждом модуле. Размер ресурса ЛДП задается в килобайтах.

-mountdir <имя_каталога_монтирования>

Этот параметр имеет действие только, если указано имя ресурса ЛДП (параметр ldmname). Параметр mountdir задает имя каталога монтирования для ресурса ЛДП. Монтирование ресурса ЛДП к указанному пользователем каталогу будет производиться перед стартом задачи на каждом вычислительном модуле. ВНИМАНИЕ! Монтирования ресурса ЛДП на сервер доступа или управляющую ЭВМ при старте задачи не производится!

-termtime <дополнительное_время>

Этот параметр задает дополнительное время для завершения задачи. Время задается в минутах.

-termsignal <сигнал_для_завершения>

Этот параметр имеет действие только, если задано дополнительное время для завершения задачи (параметр termtime). Параметр termsignal задает сигнал, который будет разослан всем процессам задачи в качестве предупреждения о предстоящем завершении. Формат задания сигнала должен соответствовать ко-манде kill (pkill). Пользователь должен самостоятельно определить обработчик сигнала в своей программе. Подробнее о сигналах можно узнать в описаниях системных вызовов kill() и signal().

-transform <имя_командного_файла>

При запуске задачи происходит преобразование списка выделенных задаче вычислительных модулей в формат среды программирования MPICH. По умолчанию системой используется командный файл /common/runmvs/bin/p4togm.sh. Параметр имя_командного_файла задает командный файл, который выполнит указанное преобразование вместо стандартного файла p4togm.sh. Необходимо учесть, что при вызове данный командный файл получит два параметра: файл со списком узлов выделенных задаче и полное имя файла запускаемой программы.

-width

Использовать альтернативный способ нумерации процессоров. По умолчанию процессы задачи распределяются по процессорам выделенных модулей в следующем порядке: 1-й процесс - на 1-й процессор 1-го модуля, 2-й процесс - на 1-й процессор 2-го модуля, 3-й процесс - на 1-й процессор 3-го модуля и т.д. После занятия всех 1-х процессоров всех выделенных модулей занимаются 2-е процессоры в том же порядке.

При указании ключа width используется другой способ нумерации: 1-й процесс - на 1-й процессор 1-го модуля, 2-й процесс - на 2-й процессор 1-го модуля, 3-й процесс - на 1-й процессор 2-го модуля и т.д.

Содержание

Предисловие автора 3
1. Введение: обзор MPI 5
1.1. Что такое MPI? 5
1.2. Что предлагает MPI? 7
1.3. Как использовать MPI? 8
2. Лабораторная работа № 1: Теория 9
2.1. Программы MPI 9
2.1.1. Формат функций MPI 9
2.1.2. Функции MPI 11
2.1.3. Пример MPI-программы 11
2.2. Сообщения MPI 13
2.2.1. Данные 13
2.2.2. Оболочка 15
2.3. Коммуникаторы 17
2.3.1. Зачем нужны коммуникаторы? 17
2.3.2. Группы коммуникаторов и процессов 20
2.4. Запуск MPI программ 22
2.4.1. Компиляция 22
2.4.2. Запуск на исполнение из командной строки 22
2.4.3. Запуск в пакете 23
2.5. Резюме 24
3. Лабораторная работа № 1: Вопросы по основам программирования в MPI 25
4. Лабораторная работа № 1: Упражнения по основам программирования в MPI 29
4.1. Предварительные требования 29
4.2. Обзор 29
4.3. Упражнение 1: 30
4.4. Упражнение 2: 32
4.5. Упражнение 3: 32
4.6. Упражнение 4: 35
4.7. Очистка 62
Литература 62
Приложение 1. Краткий глоссарий 63

Приложение 2. Шесть основных функций MPI 71
Приложение 3. Запуск программ с MPI 75
Содержание... 80

Николай Николаевич Оленев

Основы параллельного программирования
в системе MPI

Подписано в печать 02.06.2005
Формат бумаги 60х 84 1/16
Уч.-изд.л. 4,7. Усл.-печ.л. 5
Тираж 120 экз. Заказ 24

Отпечатано на ротапринтах в Вычислительном центре
им. А.А. Дородницына Российской академии наук
119991, Москва, ул. Вавилова, 40