ЖУРНАЛ ВЫЧИСЛИТЕЛЬНОЙ МАТЕМАТИКИ И МАТЕМАТИЧЕСКОЙ ФИЗИКИ, 2009, том 49, № 8, с. 1369–1384

УДК 519.658

ПАРАЛЛЕЛЬНАЯ РЕАЛИЗАЦИЯ МЕТОДА НЬЮТОНА ДЛЯ РЕШЕНИЯ БОЛЬШИХ ЗАДАЧ ЛИНЕЙНОГО ПРОГРАММИРОВАНИЯ [1]

©2009 г. В.А. Гаранжа, А.И. Голиков, Ю.Г. Евтушенко, М.Х. Нгуен

(119333 Москва, ул. Вавилова, 40, ВЦ РАН)

e-mail garan@ccas.ru

Поступила в редакцию 24.02.2009 г.

Для решения задач линейного программирования разработаны параллельные версии метода, основанного на редукции к задаче безусловной максимизации вогнутой дифференцируемой кусочно-квадратичной функции. Задача максимизации решается обобщенным методом Ньютона. Параллельный метод реализован на языке C с использованием библиотеки MPI для межпроцессорных обменов. Вычисления проводились на параллельном кластере МВС-6000IM. Решались задачи линейного программирования большой размерности с несколькими миллионами неизвестных и несколькими сотнями тысяч ограничений. Приведены результаты однопроцессорных и многопроцессорных расчетов. Библ. 14. Фиг. 5. Табл. 5.

Ключевые слова: линейное программирование, обобщенный метод Ньютона, безусловная оптимизация, параллельные вычисления.

ВВЕДЕНИЕ

Многие прикладные задачи могут быть представлены в виде задач линейного программирования (ЛП), для решения которых давно существуют численные методы и программное обеспечение, и, как могло бы показаться, не представляет никакого труда найти оптимальное решение. Однако размерности современных задач (миллионы переменных и сотни тысяч ограничений) порою не позволяют решить их традиционными методами. Поэтому требуется разработка новых подходов для решения таких задач с привлечением мощных вычислительных ресурсов.

Большие задачи ЛП, как правило, имеют неединственное решение. Методы решения задач ЛП, такие как симплекс-метод, метод внутренних точек, метод квадратичной штрафной функции (см., например, [1], [2]), дают возможность получать различные решения в случае неединственности. Так, симплекс-метод дает решение, которое принадлежит вершине многогранного множества. Методы внутренней точки сходятся к решению, в котором выполнено условие строгой дополняющей нежесткости. Метод внешней квадратичной штрафной функции дает возможность найти точное нормальное решение.

В работах [3], [4] предложен новый метод решения задачи ЛП, близкий к методу квадратичной штрафной функции вида, введенного в [5], [6], и модифицированной функции Лагранжа. Его применение к двойственной задаче ЛП дает возможность получить точную проекцию заданной точки на множество решений прямой задачи ЛП в результате однократной безусловной максимизации вспомогательной кусочно-квадратичной функции при конечном значении коэффициента штрафа. Применение

[1]Работа выполнена при финансовой поддержке РФФИ (код проекта 08-01-00619), по программе поддержки ведущих научных школ (код проекта НШ-5073.2008.1) и по программе Президиума РАН П-2.

обобщенного метода Ньютона для максимизации введенной вспомогательной функции дает возможность решать на однопроцессорных персональных компьютерах типа Pentium-IV задачи ЛП с очень большим числом неотрицательных переменных (несколько десятков миллионов) при умеренном числе ограничений (несколько тысяч). Данная работа посвящена распараллеливанию этого метода, что дает возможность решать задачи ЛП с числом ограничений, увеличенным до нескольких сотен тысяч.

В разд. 1 кратко изложены теоретические основы метода решения задачи ЛП, при котором возможно получение проекции заданной точки на множество решений прямой задачи ЛП. С помощью теории двойственности получена вспомогательная задача максимизации вогнутой кусочно-квадратичной функции переменных, число которых равно числу ограничений прямой задачи ЛП. Приводятся формулы для определения порогового значения параметра вспомогательной функции, начиная с которого за одну безусловную максимизацию этой функции по простым формулам вычисляется проекция точки на множество решений прямой задачи ЛП. Повторная максимизация этой функции, в которую подставлено решение прямой задачи, позволяет найти решение двойственной задачи ЛП.

В разд. 2 рекомендуется применять обобщенный метод Ньютона, который, как показано в [6], [7], для решения задачи максимизации вспомогательной вогнутой кусочно-квадратичной функции сходится за конечное число шагов. Результаты решения тестовых задач ЛП на однопроцессорном компьютере предлагаемым методом приводятся в разд. 3. Сравнение реализованного в системе MATLAB метода с некоторыми известными коммерческими и исследовательскими пакетами показали его конкурентоспособность по сравнению с симплекс-методом и методом внутренней точки.

В разд. 4 предлагается несколько возможных вариантов распараллеливания обобщенного метода Ньютона для решения задач ЛП.

В разд. 5 приведены некоторые результаты численных экспериментов на параллельном кластере. Из этих результатов следует, что предлагаемый подход для решения задачи ЛП с использованием обобщенного метода Ньютона хорошо распараллеливается и позволяет решать задачи ЛП с числом переменных до нескольких миллионов и числом ограничений до двух сотен тысяч. Так, например, при решении задач ЛП с одним миллионом неизвестных и при десяти тысячах ограничений с использованием одной из схем распараллеливания на 144 процессорах кластера МВС-6000IM было достигнуто ускорение расчетов примерно в 50 раз, время счета составляло 28 с. В качестве второго примера укажем на задачу ЛП с двумя миллионами переменных при двухстах тысячах ограничений, которая на 80 процессорах была решена примерно за 40 мин.

1. СВЕДЕНИЕ ЗАДАЧИ ЛП К БЕЗУСЛОВНОЙ ОПТИМИЗАЦИИ

Пусть задана прямая задача ЛП в стандартной форме

$$f^* = \min_{x \in X} \; c^T x, \quad X = \{x \in \mathbb{R}^n \; : \; Ax = b, \; x \geq 0_n\}. \tag{P}$$

Двойственная к ней имеет вид

$$f^* = \max_{u \in U} \; b^T u, \quad U = \{u \in \mathbb{R}^m \; : \; A^T u \leq c\}. \tag{D}$$

Здесь $A \in \mathbb{R}^{m \times n}$, $c \in \mathbb{R}^n$ и $b \in \mathbb{R}^m$ заданы, x — вектор прямых переменных, а u — двойственных, через 0_i обозначен i-мерный нулевой вектор.

Предположим, что множество решений X^* прямой задачи (P) непусто, следовательно, множество решений U^* двойственной задачи (D) также непусто. Необходимые и достаточные условия оптимальности (условия Куна–Таккера) для задач (P) и (D) запишем в виде

$$Ax^* - b = 0_m, \quad x^* \geq 0_n, \quad D(x^*)v^* = 0_n, \tag{1}$$

$$v^* = c - A^T u^* \geq 0_n. \tag{2}$$

Здесь в ограничения двойственной задачи (D) введен неотрицательный вектор дополнительных переменных $v = c - A^T u \geq 0_n$. Через $D(z)$ обозначается диагональная матрица, у которой i-й диагональный элемент есть i-я компонента вектора z.

Рассмотрим задачу нахождения проекции заданной точки $\hat{x}$ на множество решений X^* прямой задачи (P)

$$\frac{1}{2}\|\hat{x}^* - \hat{x}\|^2 = \min_{x \in X^*} \frac{1}{2}\|x - \hat{x}\|^2, \quad X^* = \{x \in \mathbb{R}^n \ : \ Ax = b, \ c^T x = f^*, \ x \geq 0_n\}. \tag{3}$$

Здесь и всюду ниже используется евклидова норма векторов, f^* — оптимальное значение целевой функции исходной задачи ЛП.

Для этой задачи введем функцию Лагранжа

$$L(x, p, \beta) = \frac{1}{2}\|x - \hat{x}\|^2 + p^T(b - Ax) + \beta(c^T x - f^*).$$

Здесь $p \in \mathbb{R}^m$, $\beta \in \mathbb{R}^1$ суть множители Лагранжа для задачи (3). Двойственная к (3) задача имеет вид

$$\max_{p \in \mathbb{R}^m} \max_{\beta \in \mathbb{R}^1} \min_{x \in \mathbb{R}^n_+} L(x, p, \beta). \tag{4}$$

Запишем условия Куна–Таккера для задачи (3)

$$x - \hat{x} - A^T p + \beta c \geq 0_n, \qquad D(x)(x - \hat{x} - A^T p + \beta c) = 0_n, \qquad x \geq 0_n, \tag{5}$$

$$Ax = b, \ c^T x = f^*. \tag{6}$$

Легко проверить, что формулы (5) эквивалентны выражению

$$x = (\hat{x} + A^T p - \beta c)_+. \tag{7}$$

Эта формула дает решение внутренней задачи минимизации в (4).

Подставляя (7) в функцию Лагранжа $L(x, p, \beta)$, получаем двойственную функцию для задачи (4)

$$\hat{L}(p, \beta) = b^T p - \frac{1}{2}\|(\hat{x} + A^T p - \beta c)_+\|^2 - \beta f^* + \frac{1}{2}\|\hat{x}\|^2.$$

Функция $\hat{L}(p, \beta)$ вогнута, кусочно-квадратична и непрерывно дифференцируема по своим переменным p и β.

Двойственная задача (4) сводится к решению внешней задачи максимизации

$$\max_{p \in \mathbb{R}^m} \max_{\beta \in \mathbb{R}^1} \hat{L}(p, \beta). \tag{8}$$

Решив задачу (8), найдем оптимальные p и β. После их подстановки в (7) получаем решение $\hat{x}^*$ задачи (3), т.е. проекцию точки $\hat{x}$ на множество решений прямой задачи ЛП (P). Необходимые и достаточные условия оптимальности для задачи (8) имеют вид

$$\hat{L}_p(p, \beta) = b - A(\hat{x} + A^T p - \beta c)_+ = b - Ax = 0_m,$$

$$\hat{L}_\beta(p, \beta) = c^T(\hat{x} + A^T p - \beta c)_+ - f^* = c^T x - f^* = 0,$$

где x определен формулой (7). Эти условия выполнены тогда и только тогда, когда $x \in X^*$ и $x = \hat{x}^*$.

К сожалению, задача безусловной оптимизации (8) содержит неизвестную априори величину f^* — оптимальное значение целевой функции задачи ЛП. Однако эту задачу можно упростить, избавившись от этого недостатка. Для этого вместо (8) предлагается решать следующую упрощенную задачу безусловной максимизации:

$$I = \max_{p \in \mathbb{R}^m} S(p, \beta), \tag{9}$$

где скаляр β фиксирован, а функция $S(p, \beta)$ определена следующим образом:

$$S(p, \beta) = b^T p - \frac{1}{2}\|(\hat{x} + A^T p - \beta c)_+\|^2. \tag{10}$$

Рассмотрим задачу (9) и ее связь с прямой задачей ЛП (P). Заметим, что, в отличие от задачи (3), двойственная к ней задача (8) имеет неединственное решение. Естественно возникает вопрос о нахождении среди всех решений задачи (8) минимального значения β_* множителя Лагранжа β. Тогда в двойственной задаче (8), как следует из приведенной ниже теоремы 1, можно зафиксировать $\beta \geq \beta_*$ и решать задачу максимизации двойственной функции $\hat{L}(p, \beta)$ только по переменным p, т.е. решать задачу (9). При этом пара $[p, \beta]$ является решением задачи (8), тройка $[\hat{x}^*, p, \beta]$ — седловая точка задачи (3), в которой проекция $\hat{x}^*$ — решение задачи (P) — определяется по формуле (7).

Для нахождения минимального β обратимся к условиям Куна–Таккера для задачи (3), которые для этой задачи являются необходимыми и достаточными условиями оптимальности. Без потери общности предположим, что первые l компонент вектора $\hat{x}^*$ строго больше нуля. В соответствии с этим предположением представим векторы $\hat{x}^*$, $\hat{x}$, c и матрицу A в виде

$$\hat{x}^{*T} = [\hat{x}_l^{*T}, \hat{x}_d^{*T}], \quad \hat{x}^T = [\hat{x}_l{}^T, \hat{x}_d{}^T], \quad c^T = [c_l{}^T, c_d{}^T], \quad A = [A_l \mid A_d], \tag{11}$$

где $\hat{x}_l^* > 0_l$, $\hat{x}_d^* = 0_d$, $d = n - l$.

В соответствии с разбиением (11) оптимальный вектор дополнительных переменных v^* из условий Куна–Таккера (1), (2) для задач (P) и (D) представим в виде $v^{*T} = [v_l^{*T}, v_d^{*T}]$. Тогда, согласно условию дополняющей нежесткости $x^{*T} v^* = 0$, $x^* \geq 0_n$, $v^* \geq 0_n$, выражение (2) запишется в виде

$$v_l^* = c_l - A_l^T u^* = 0_l, \tag{12}$$

$$v_d^* = c_d - A_d^T u^* \geq 0_d. \tag{13}$$

С использованием обозначений из (11) необходимые и достаточные условия оптимальности (5), (6) для задачи (3) можно переписать в развернутом виде

$$\hat{x}_l^* = \hat{x}_l + A_l^T p - \beta c_l > 0_l, \tag{14}$$

$$\hat{x}_d^* = 0_d, \quad \hat{x}_d + A_d^T p - \beta c_d \leq 0_d, \tag{15}$$

$$A_l \hat{x}_l^* = b, \quad c_l{}^T \hat{x}_l^* = f^*. \tag{16}$$

Среди решений системы (14)–(16) найдем такие множители Лагранжа $[p, \beta]$, что β является минимальным, т.е. приходим к задаче ЛП

$$\beta_* = \inf_{\beta \in \mathbb{R}^1} \inf_{p \in \mathbb{R}^m} \{\beta \ : \ A_l^T p - \beta c_l = \hat{x}_l^* - \hat{x}_l, \ A_d^T p - \beta c_d \leq -\hat{x}_d\}. \tag{17}$$

Ограничения в этой задаче совместны, но целевая функция может быть не ограничена снизу. В этом случае будем полагать $\beta_* = \gamma$, где γ — некоторое число.

В теореме 1 (см. [3], [4]) установлено, что если система уравнений в (17) однозначно разрешима относительно p, то величина β_* представима в виде

$$\beta_* = \begin{cases} \max\limits_{i\in\sigma} \dfrac{(\hat{x}_d + A_d^T(A_l A_l^T)^{-1}A_l(\hat{x}_l^* - \hat{x}_l))^i}{(v_d^*)^i}, & \sigma \neq \emptyset, \\ \gamma > -\infty, & \sigma = \emptyset. \end{cases} \tag{18}$$

Здесь введено индексное множество $\sigma = \{l+1 \le i \le n \ : \ (v_d^*)^i > 0\}$ и γ — произвольное число.

Теорема 1. *Пусть множество решений X^* задачи* (P) *непусто. Тогда при любом $\beta \ge \beta_*$, где β_* определяется формулой* (17)*, пара $[p(\beta), \beta]$, где $p(\beta)$ — решение задачи безусловной максимизации* (9) *или, что то же самое, решение системы $A(\hat{x} + A^T p - \beta c)_+ = b$, определяет проекцию $\hat{x}^*$ заданной точки $\hat{x}$ на множество решений X^* прямой задачи* (P) *по формуле*

$$\hat{x}^* = (\hat{x} + A^T p(\beta) - \beta c)_+. \tag{19}$$

Если дополнительно ранг матрицы A_l, соответствующий ненулевым компонентам вектора $\hat{x}^*$, равен m, то β_* определяется по формуле (18)*, а точное решение двойственной задачи* (D) *находится в результате решения задачи безусловной максимизации* (9) *по формуле*

$$u^* = \frac{1}{\beta}\left[p(\beta) - (A_l A_l^T)^{-1}A_l(\hat{x}^* - \hat{x}_l)\right].$$

Теорема 1 позволяет заменить задачу (8), содержащую априори неизвестное число f^*, на задачу (9), в которой вместо этого числа фигурирует полуинтервал $[\beta_*, +\infty)$, что существенно проще с вычислительной точки зрения. Отметим, что значение β_*, найденное из решения задачи ЛП (17) или формулы (18), может быть отрицательным.

Следующая теорема утверждает, что если известна какая-нибудь точка $x^* \in X^*$, то можно получить решение двойственной задачи (D) после однократного решения задачи безусловной максимизации (9).

Теорема 2. *Пусть множество решений X^* задачи ЛП* (P) *непусто. Тогда для любых $\beta > 0$ и $\hat{x} = x^* \in X^*$ точное решение двойственной задачи* (D) *находится по формуле $u^* = p(\beta)/\beta$, где $p(\beta)$ — решение задачи безусловной максимизации* (9)*.*

Для иллюстрации работы этой теоремы обратимся к методу внешнего квадратичного штрафа, примененному к двойственной задаче (D), т.е. рассмотрим задачу

$$\max_{p\in\mathbb{R}^m}\left\{b^T p - \frac{1}{2}\left\|(A^T p - \beta c)_+\right\|^2\right\}. \tag{20}$$

Оказывается, что можно получить точное решение u^* двойственной задачи (D), не устремляя в (20) коэффициент штрафа β к $+\infty$. Если в задаче (20) коэффициент штрафа $\beta \ge \beta_*$, то, согласно теореме 1, по формуле (19), в которой $\hat{x} = 0_n$, находим нормальное решение $\tilde{x}^*$ прямой задачи (P). Согласно теореме 2 далее следует решить задачу безусловной максимизации при любом $\beta > 0$:

$$\max_{p\in\mathbb{R}^m}\left\{b^T p - \frac{1}{2}\left\|(\tilde{x}^* + \beta(A^T p - \beta c))_+\right\|^2\right\}. \tag{21}$$

Используя ее решение $p(\beta)$, получаем решение $p(\beta)/\beta = u^* \in U^*$ двойственной задачи (D). Отметим, что задача (21) не сложнее, чем задача (20). Таким образом, решая только две задачи безусловной максимизации, можно получить точную проекцию на множество решений прямой и некоторое решение двойственной задач ЛП, если в задаче (20) взять коэффициент штрафа $\beta \geq \beta_*$, а в задаче (21) взять любой положительный коэффициент β.

Для одновременного решения прямой и двойственной задач ЛП предлагается использовать следующий итерационный процесс:

$$x_{s+1} = (x_s + A^T p_{s+1} - \beta c)_+, \tag{22}$$

где произвольный параметр $\beta > 0$ фиксирован, а вектор p_{s+1} определяется из решения следующей задачи безусловной максимизации:

$$p_{s+1} \in \arg\max_{p \in \mathbb{R}^m} \left\{ b^T p - \frac{1}{2} \left\| (x_s + A^T p - \beta c)_+ \right\|^2 \right\}. \tag{23}$$

Теорема 3. *Пусть множество решений X^* прямой задачи* (P) *непусто. Тогда при любом $\beta > 0$ и при любой начальной точке x_0 итерационный процесс* (22), (23) *сходится к $x^* \in X^*$ за конечное число шагов ω. Формула $u^* = p_{\omega+1}/\beta$ дает точное решение двойственной задачи* (D).

Этот итерационный процесс является конечным и дает точное решение прямой задачи (P) и точное решение двойственной задачи (D). Отметим, что в этом методе не требуется знать пороговое значение коэффициента штрафа. Но если выбранное значение коэффициента меньше порогового значения, то метод за конечное число шагов находит некоторое решение прямой задачи, а не проекцию начальной точки на множество решений прямой задачи ЛП. Заметим, что $x_\omega = x^* \in X^*$ является проекцией точки $x_{\omega-1}$ на множество решений X^* задачи (P).

2. ОБОБЩЕННЫЙ МЕТОД НЬЮТОНА ДЛЯ БЕЗУСЛОВНОЙ МАКСИМИЗАЦИИ КУСОЧНО-КВАДРАТИЧНОЙ ФУНКЦИИ

Решение задачи о безусловной максимизации (23) может выполняться любым методом, например, методом сопряженного градиента. Но, как показал Мангасарьян, для безусловной оптимизации кусочно-квадратичной функции особенно эффективен обобщенный метод Ньютона (см. [7]). Приведем краткое описание этого метода.

Максимизируемая функция $S(p, \beta, \hat{x})$ (см. (10) в задаче (9) или (23) является вогнутой кусочно-квадратичной и дифференцируемой. Для этой функции обычная матрица Гессе не существует. Действительно, градиент

$$\frac{\partial}{\partial p} S(p, \beta, \hat{x}) = b - A(\hat{x} + A^T p - \beta c)_+$$

функции $S(p, \beta, \hat{x})$ не дифференцируем. Но для этой функции можно определить обобщенную матрицу Гессе, которая является $m \times m$-симметричной отрицательно полуопределенной матрицей следующего вида:

$$\frac{\partial^2}{\partial p^2} S(p, \beta, \hat{x}) = -AD^\sharp(z)A^T,$$

где через $D^\sharp(z)$ обозначена $n \times n$ диагональная матрица с i-м диагональным элементом z_i, равным 1, если $(\hat{x} + A^T p - \beta c)_i > 0$, и равным 0, если $(\hat{x} + A^T p - \beta c)_i \le 0$, $i = 1, 2, \ldots, n$. Так как обобщенная матрица Гессе может быть вырожденной, используется следующее модифицированное ньютоновское направление:

$$-\left(\frac{\partial^2}{\partial p^2} S(p, \beta, \hat{x}) - \delta I_m\right)^{-1} \frac{\partial}{\partial p} S(p, \beta, \hat{x}),$$

где δ — малая положительная величина (обычно при расчетах полагалось $\delta = 10^{-4}$) и I_m — единичная матрица порядка m.

В этом случае модифицированный метод Ньютона имеет вид

$$p_{k+1} = p_k - \left(\frac{\partial^2}{\partial p^2} S(p_k, \beta, \hat{x}) - \delta I_m\right)^{-1} \frac{\partial}{\partial p} S(p_k, \beta, \hat{x}).$$

Критерий окончания его работы полагался следующим:

$$\|p_{k+1} - p_k\| \le \mathrm{tol}\,.$$

О. Мангасарьян исследовал сходимость обобщенного метода Ньютона для безусловной оптимизации подобной вогнутой кусочно-квадратичной функции с выбором шага по правилу Армихо. Доказательство конечной глобальной сходимости обобщенного метода Ньютона для безусловной оптимизации кусочно-квадратичной функции можно найти в [7].

3. ВЫЧИСЛИТЕЛЬНЫЕ ЭКСПЕРИМЕНТЫ НА ОДНОПРОЦЕССОРНОМ КОМПЬЮТЕРЕ

Решались сгенерированные случайным образом задачи ЛП с большим числом неотрицательных переменных (до нескольких десятков миллионов) и существенно меньшим числом ограничений-равенств, т.е. имело место $n \gg m$.

Итак, задавались числа m и n, определяющие количество строк и столбцов матрицы A, и ρ — плотность заполнения матрицы A ненулевыми элементами. В частности, $\rho = 1$ означает, что случайным образом генерировались все элементы матрицы A, а значение $\rho = 0.01$ указывает, что в матрице A генерировался только 1% элементов, а остальные полагались равными нулю. Элементы матрицы A определялись случайным образом из интервала [-50,+50]. Решение x^* прямой задачи (P) и решение u^* двойственной задачи (D) генерировались следующим образом. Полагалось, что в векторе x^* содержится $n - 3m$ нулевых компонент, а остальные компоненты выбирались случайным образом из интервала [0,10]. Половина компонент вектора u^* полагались равными нулю, а остальные выбирались случайным образом из интервала [-10,10]. Решения x^* и u^* использовались для вычисления коэффициентов целевой функции c и правых частей b задачи ЛП (P). Векторы b и c определялись по формулам

$$b = Ax^*, \quad c = A^T u^* + \xi,$$

если $x_i^* > 0$, то $\xi_i = 0$, если $x_i^* = 0$, то компонента ξ_i выбиралась случайным образом из интервала

$$0 \le \gamma_i \le \xi_i \le \theta_i.$$

В приведенных ниже результатах расчетов считалось, что все $\gamma_i = 1$ и $\theta_i = 10$. Заметим, что при γ_i, близких к нулю, величина $\xi_i = (c - A^T u^*)_i = (v_d^*)_i$ может также оказаться очень малой величиной. Согласно формуле (18), априори неизвестная величина β_* может быть очень большой. Тогда сгенерированная задача ЛП может оказаться трудно решаемой.

Предлагаемый метод решения прямой и двойственной задач ЛП, сочетающий итеративный процесс (22), (23) и обобщенный метод Ньютона, реализован в системе MATLAB (см. [3],[4]). Для вычислений использовался компьютер с процессором Pentium-4, тактовой частотой 2.6 ГГц, оперативной памятью 1 Гб. Численные эксперименты со случайно сгенерированными задачами ЛП показали высокую эффективность метода при решении задач ЛП с большим числом неотрицательных переменных (решались задачи до 50 млн. переменных) и средним числом ограничений-равенств (до 5 тысяч). Время решения таких задач составляло от нескольких десятков до полутора часов. Высокая эффективность этих расчетов объясняется тем, что основная вычислительная трудность предлагаемого метода приходится на решение вспомогательной задачи безусловной максимизации, которая решается обобщенным методом Ньютона. Ее размерность определяется количеством ограничений типа равенств, число которых существенно меньше, чем число неотрицательных переменных в исходной задаче ЛП.

В табл. 1 даны результаты расчетов тестовых задач с помощью программы EGM, (см. [4]), реализующей на MATLAB метод (22),(23), и с помощью других зарубежных коммерческих и исследовательских пакетов. Все задачи решались на компьютере Celeron 2.02 GHz с оперативной памятью 1.0 Gb. Для сравнения использовались пакеты BPMPD v.2.3 (метод внутренней точки, см. [8]), MOSEK v.2.0 (метод внутренней точки, см. [9]) и широко распространенный коммерческий пакет CPLEX v.6.0.1 (метод внутренней точки и симплекс-метод).

В табл. 1 указаны размерности m, n и ρ — плотность ненулевых элементов матрицы A, T — время решения задачи ЛП в секундах, в столбце Iter — количество итераций (для программы EGM указано общее число решенных систем линейных уравнений в методе Ньютона при решении задач (23)). Везде полагалось $\beta = 1$. Вычислялись чебышевские нормы векторов невязок:

$$\Delta_1 = \|Ax - b\|_\infty, \qquad \Delta_2 = \|(A^T u - c)_+\|_\infty, \qquad \Delta_3 = |c^T x - b^T u|. \tag{24}$$

Таблица 1

$m \times n \times \rho$	Программа	T, с	Iter	Δ_1	Δ_2	Δ_3
$500 \times 10^4 \times 1$	EGM (NEWTON)	55.0	12	1.5×10^{-8}	1.8×10^{-12}	1.2×10^{-7}
	BPMPD (Interior point)	37.4	23	2.3×10^{-10}	1.8×10^{-11}	1.1×10^{-10}
	MOSEK (interior point)	87.2	6	9.7×10^{-8}	3.8×10^{-9}	1.6×10^{-6}
	CPLEX (Interior point)	80.3	11	1.8×10^{-8}	1.1×10^{-7}	0.0
	CPLEX (Simplex)	61.8	8308	8.6×10^{-4}	1.9×10^{-10}	7.2×10^{-3}
$3000 \times 10^4 \times 0.01$	EGM (NEWTON)	155.4	11	6.1×10^{-10}	3.4×10^{-13}	3.6×10^{-8}
	BPMPD (Interior point)	223.5	14	4.6×10^{-9}	2.9×10^{-10}	3.9×10^{-9}
	MOSEK (interior point)	42.6	4	3.1×10^{-8}	1.2×10^{-8}	3.7×10^{-8}
	CPLEX (Interior point)	69.9	5	1.1×10^{-6}	1.3×10^{-7}	0.0
	CPLEX (Simplex)	1764.9	6904	3.0×10^{-3}	8.1×10^{-9}	9.3×10^{-2}
$1000 \times (5 \times 10^6) \times 0.01$	EGM (NEWTON)	1007.5	10	3.9×10^{-8}	1.4×10^{-13}	6.1×10^{-7}
$1000 \times 10^5 \times 1$	EGM (NEWTON)	2660.8	8	2.1×10^{-7}	1.4×10^{-12}	7.1×10^{-7}

Представленные в табл. 1 результаты решения первых двух задач небольшой размерности показали, что программа EGM для MATLAB оказалась по эффективности близкой к известным пакетам, реализующим метод внутренней точки и симплекс-метод.

В третьей строке таблицы даны результаты расчетов задач ЛП с пятью миллионами неотрицательных переменных, тысячей ограничений, однопроцентной заполненностью матрицы A ненулевыми элементами. Время расчетов по программе EGM1 составило 16 мин. В четвертой строке приведены результаты в случае, когда задача имела 1000 ограничений, матрица A была полностью заполнена и $n = 10^5$. Время расчетов было 44 мин. Обе задачи были решены с высокой точностью (нормы невязок не превосходили 7.1×10^{-7}). Обе эти задачи не удалось решить другими пакетами. Таким образом, для больших задач ЛП программа EGM, реализованная в MATLAB для однопроцессорного компьютера, оказалась существенно эффективней по сравнению с рассмотренными пакетами ЛП. Отметим, что реализация этого же алгоритма на языке C позволила уменьшить время счета примерно в 8 раз.

С целью увеличения числа ограничений и возможности доведения его до сотен тысяч целесообразно перейти к параллельной реализации предложенного метода.

4. ПАРАЛЛЕЛЬНЫЕ ИТЕРАЦИОННЫЕ АЛГОРИТМЫ

Будем рассматривать параллельные реализации метода (22), (23) с использованием обобщенного метода Ньютона в рамках модели распределенной памяти. При этом у каждого процесса (исполняемой программы) имеется свое адресное пространство, а обмены данными между процессами осуществляются при помощи библиотеки MPI. Предполагается, что каждый процесс исполняется на своем процессоре (ядре).

4.1. Расчетные формулы

Описание параллельной реализации начнем с расчетных формул итерационного метода (22), (23) и обобщенного метода Ньютона.

Шаг 1. Задать $\beta > 0$, точности, соответственно, для внешних и внутренних итераций tol1 и tol, начальные приближения x_0 и p_0.

Шаг 2. Вычислить значение функции

$$S(p_k, \beta, x_s) = b^T p_k - \frac{1}{2} ||(x_s + A^T p_k - \beta c)_+||^2. \tag{25}$$

Здесь k — номер внутренней итерации метода Ньютона для решения задачи безусловной максимизации (23), s — номер внешней итерации.

Шаг 3. Вычислить градиент функции (25) по переменной p:

$$G_k = \frac{\partial S}{\partial p}(p_k, \beta, x_s) = b - A(x_s + A^T p_k - \beta c)_+. \tag{26}$$

Шаг 4. Используя обобщенную матрицу Гессе функции (25), сформировать матрицу $H_k \in \mathbb{R}^{m \times m}$:

$$H_k = \delta I + A D_k A^T. \tag{27}$$

Здесь диагональная матрица $D_k \in \mathbb{R}^{n \times n}$ задается равенствами

$$(D_k)_{ii} = \begin{cases} 1, & (x_s + A^T p_k - \beta c)^i > 0 \\ 0, & (x_s + A^T p_k - \beta c)^i \leqslant 0 \end{cases} \tag{28}$$

Шаг 5. Найти направление максимизации δp как решение линейной системы

$$H_k \delta p = -G_k \tag{29}$$

с помощью предобусловленного метода сопряженных градиентов. В качестве предобусловливателя используется диагональная часть матрицы H_k.

Шаг 6. Определить p_{k+1} по формуле

$$p_{k+1} = p_k - \tau_k \delta p,$$

где итерационный параметр τ_k находится из решения одномерной задачи максимизации методом деления пополам или методом Армихо

$$\tau_k = \max_{\tau} S(p_k - \tau \delta p, \beta, x_s).$$

На практике, во всех приведенных ниже расчетах, значение $\tau_k = 1$ задавало искомое локальное решение задачи максимизации, так что перебор оказывался не нужен.

Шаг 7. Если выполнен критерий останова для внутренних итераций по k

$$\|p_{k+1} - p_k\| \leqslant \mathrm{tol},$$

то положить $\tilde{p} = p_{k+1}$ и вычислить x_{s+1} по формуле

$$x_{s+1} = (x_s + A^T \tilde{p} - \beta c)_+. \tag{30}$$

Иначе перейти к шагу 2, положив $k = k + 1$.

Шаг 8. Если выполнен критерий останова для внешних итераций по s

$$\|x_{s+1} - x_s\| \leqslant \mathrm{tol1},$$

то вычислить решение двойственной задачи (D)

$$u^* = \tilde{p}/\beta.$$

Решение прямой задачи (P) есть $x^* = x_{s+1}$. Иначе положить $p_0 = \tilde{p}$ и перейти к шагу 2, положив $s = s + 1$.

4.2. Основные операции параллельного алгоритма

В алгоритме 1–8 основными трудоемкими операциями являются формирование системы линейных уравнений (29) и ее решение. Параллельная реализация алгоритма требует распараллеливания следующих основных операций:

- умножение матриц на вектор: Ax и $A^T p$;
- вычисление скалярных произведений вида $x^T y,\ x, y \in \mathbb{R}^n$ и $p^T q,\ p, q \in \mathbb{R}^m$;
- формирование обобщенной матрицы Гессе (27);
- умножение обобщенной матрицы Гессе на вектор.

Заметим, что все оставшиеся вычисления на шагах алгоритма 1–8 являются локальными. Например, метод сопряженных градиентов требует вычисления распределенных скалярных произведений и распределенного умножения матрицы на вектор. Остальные вычисления локальны и не требуют обменов. При вычислении функции (25) необходимо распределенное умножение матрицы на вектор для вычисления вектора

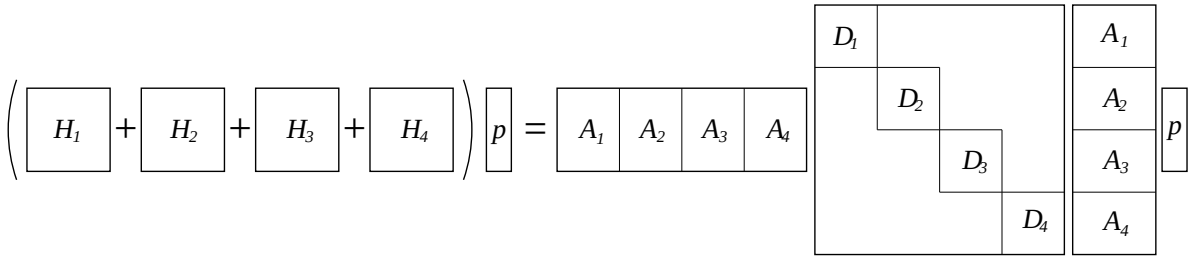
$$z_k = (x_s + A^T p_k - \beta c)_+$$

и для вычисления скалярного произведения $b^T p_k$. Для вычисления градиента (26) нужна дополнительная операция умножения матрицы на вектор.

С точки зрения структуры параллельного алгоритма можно полагать, что решается недоопределенная линейная система с матрицей A методом наименьших квадратов, поскольку все основные трудности параллельного алгоритма возникают уже на этом этапе. Варианты разбиения данных для построения параллельных алгоритмов обсуждались, в частности, в [10] и [11].

4.3. Разбиение данных: столбцовая схема

Поскольку число строк m в матрице A значительно меньше, чем число столбцов n, то при небольшом числе процессоров n_p можно использовать простую столбцовую схему разбиения данных, в которой матрица A разбивается на блочные столбцы A_i примерно равного размера, как показано на фиг. 1 на примере 4 процессоров.



Фиг. 1.

Для простоты изложения будем полагать, что n нацело делится на n_p. Тогда $N_c = n/n_p$. На процессоре с номером i хранится подвектор $x_i \in \mathbb{R}^{N_c}$ и подматрица $A_i \in \mathbb{R}^{m \times N_c}$. При описании разбиения данных удобно опускать номер локальной и глобальной итерации. В этой столбцовой схеме все векторы размерности m, т.е. p, b и т.д., дублируются на каждом из процессоров. В результате операция умножения на транспонированную матрицу, т.е. вычисление выражения $A^T p$, является локальной. С другой стороны, умножение матрицы A на вектор можно записать в виде

$$Ax = \sum_{i=1}^{n_p} A_i x_i$$

При этом умножение подматриц на подвекторы x_i производится независимо и n_p полученных векторов размерности m складываются друг с другом при помощи функции MPI_Allreduce из библиотеки MPI.

Матрицу Гессе H представим в виде

$$H = \sum_{i=1}^{n_p} H_i, \qquad H_i = A_i D_i A_i^T,$$

где $D_i \in \mathbb{R}^{N_c \times N_c}$ — диагональные блоки матрицы D. В столбцовой схеме матрица H не формируется. На i-м процессоре вычисляется локальная матрица H_i. Для умножения матрицы на некоторый вектор q

$$Hq = \sum_{i=1}^{n_p} H_i q$$

каждое слагаемое $H_i q$ вычисляется локально и n_p получившихся векторов суммируются при помощи функции MPI_Allreduce так, что результат суммирования оказывается доступен на всех процессорах.

В результате метод сопряженных градиентов для решения линейной системы (29) не является параллельным и его вычислительные затраты растут пропорционально числу процессоров. Обозначим через γ отношение вычислительных затрат на решение линейной системы к оставшимся вычислительным затратам на одной итерации метода Ньютона. Если пренебречь временем на обмены, то параллельное ускорение можно оценить сверху при помощи простой формулы

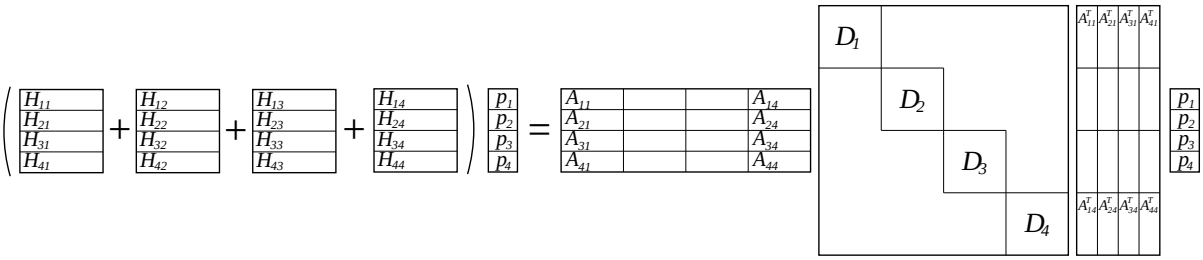
$$s(n_p) = n_p \frac{1+\gamma}{1+\gamma n}. \tag{31}$$

Таким образом, если $\gamma = 0.05$, то $s(4) = 3.5$, а $s(6) = 4.85$. Ясно, что такой простой алгоритм эффективен лишь в том случае, если коэффициент γ достаточно мал.

С учетом реальных коммуникационных затрат ускорение может быть заметно хуже.

4.4. Разбиение данных: клеточная схема

Для того чтобы достичь большей параллельной эффективности, можно использовать "клеточное" разбиение, в котором матрица A разбивается на прямоугольные блоки, как показано на фиг. 2. На этой же фигуре показано соответствующее разбиение векторов.



Фиг. 2.

Для простоты изложения предположим, что полное число процессоров можно представить в виде $n_p = n_r \times n_c$, т.е. достаточно условно можно полагать, что процессоры расположены в узлах решетки размера $n_r \times n_c$, при этом величина n делится нацело на n_c, а m делится нацело на n_r. Таким образом, $n = n_c \times N_c$ и $m = n_r \times N_r$. Матрица A состоит из подматриц $A_{ij} \in \mathbb{R}^{N_r \times N_c}$. В этой схеме вектор $x \in \mathbb{R}^n$ разбивается на n_c подвекторов x_i, а вектор $p \in \mathbb{R}^m$ — на n_r подвекторов p_j, причем подвектор x_i лежит одновременно на n_r процессорах, принадлежащих i-му столбцу решетки процессоров, а j-й подвектор p_j дублируется n_c раз на j-й строке решетки процессоров.

Рассмотрим, как для такого разбиения данных производятся основные распределенные операции.

(а) Операция $x = A^T p$ представляется в виде

$$x_i = \sum_{j=1}^{n_r} A_{ji}^T p_j, \qquad i = 1, 2, \ldots, n_c. \tag{32}$$

На первый взгляд, все вычисления являются локальными и для нахождения x_i необходимо найти сумму из n_r векторов размерности N_c, например, при помощи функции MPI_Allreduce из библиотеки MPI. При этом для суммирования удобно использовать аппарат расщепления коммуникаторов, который предоставляется библиотекой MPI. Также можно независимо вычислять суммы векторов, распределенных по разным строкам решетки процессоров. Однако если величина n очень велика, то время, затрачиваемое на обмены, оказывается неприемлемо большим, что не позволяет получить ускорения.

Простой выход из этой ситуации состоит в том, чтобы отказаться от оптимальной вычислительной сложности при умножении матрицы A^T на вектор. Предположим, что для всех j известно, что j-му процессору в i-строке решетки процессоров принадлежит весь i-й блочный столбец матрицы A, а не просто подматрица A_{ij}, т.е. матрица A дублируется n_r раз. Если предположить, что на каждом из $n_r \times n_c$ процессоров доступен вектор p целиком, то одна и та же расчетная формула (32) повторяется на всех процессорах из одного столбца решетки независимо. Таким образом, вектор x_i оказывается доступен на всех процессорах i-го столбца решетки без дополнительных обменов. Но для этого матрица A^T умножается n_r раз на вектор p так, что число умножений на этом этапе близко к ρmnn_r.

(б) Операция Ax является почти локальной, так как требуется лишь суммировать n_c подвекторов длины N_r независимо на каждой из n_r строк решетки процессоров.

(в) Вычисления на этапе формирования матрицы H оказываются полностью локальными и не требуют обменов. Количество умножений на этом этапе можно примерно оценить как $m^2\rho^2\rho_z n$, где ρ_z — это доля ненулевых элементов в диагональной матрице D. В этих выкладках предполагалось, что ненулевые элементы разбросаны по матрице A случайным образом, т.е. равномерно в среднем.

Результатом сборки является сумма из n_c матриц, причем j-е слагаемое распределено по j-му столбцу решетки процессоров, как видно из фиг. 2.

Пусть коэффициент γ имеет тот же смысл, что и в формуле (31), т.е. γ — отношение затрат на решение линейной системы к остальным затратам в однопроцессорной реализации алгоритма. Коэффициент

$$\alpha = \frac{\rho mn}{m^2 n\rho^2\rho_z} = \frac{1}{m\rho\rho_z} \tag{33}$$

обозначает приближенное отношение затрат на умножение матрицы A на вектор к затратам на формирование матрицы H в однопроцессорной реализации. Тогда получается следующая весьма грубая, но простая и наглядная оценка сверху для ускорения клеточной схемы:

$$s_u(n_r \times n_c) = n_r \frac{1+2\alpha}{1+(n_r+1)\alpha} n_c \frac{1+\gamma}{1+\gamma n_c}. \tag{34}$$

На практике коэффициент α получается существенно меньшим, чем следует из формулы (33). Если в (34) положить $\gamma = 0.05$ и $\alpha = 0.1$, то $s_u(8 \times 8) = 30.3$

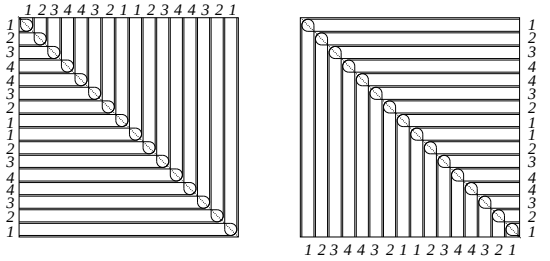
При анализе данной схемы остался за рамками рассмотрения вопрос, каким же образом удается записывать целый j-й блочный столбец матрицы A на каждый процессор в j-м столбце решетки процессоров. Ведь если матрица A формируется локально, т.е. каждая клетка строится на одном процессоре, то для такого перераспределения данных может понадобиться существенно больше времени, чем для решения задачи. Поскольку в данной работе матрицы формировались при помощи явных формул, то эта проблема выпала из поля зрения. Нужно иметь в виду, что в данной задаче оптимальные структуры данных сконструировать очень непросто, поскольку матрица A является весьма разреженной, в то время как матрицу Гессе можно полагать практически заполненной.

4.5. Разбиение данных: зеркальная циклическая схема

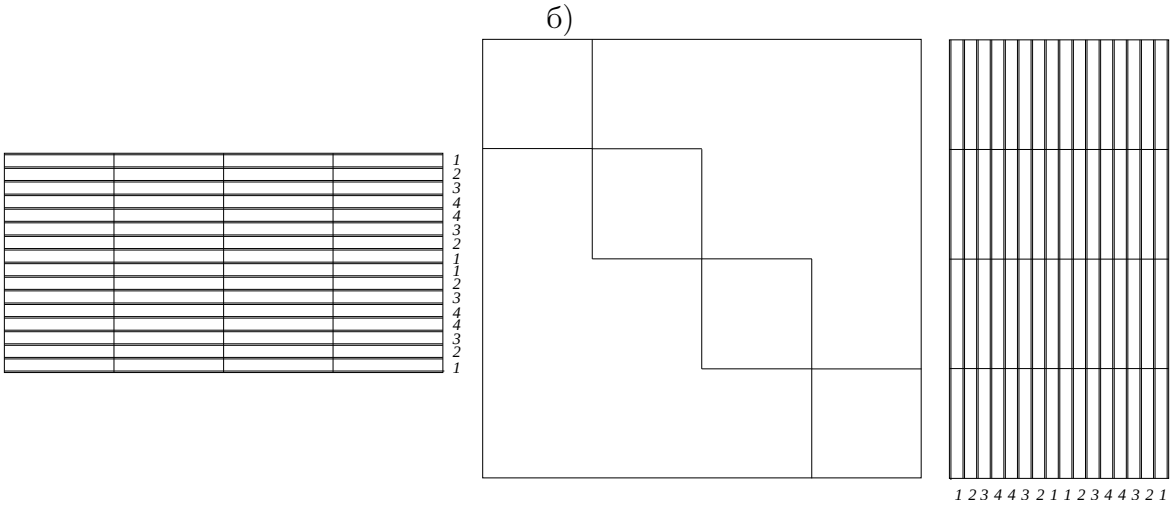
При параллельной реализации клеточной схемы разбиения матрицы A не учитывался тот факт, что обобщенная матрица Гессе H является симметричной, т.е. арифметические затраты на ее заполнение можно уменьшить почти вдвое по сравнению с общей квадратной матрицей. Уменьшить затраты очень легко в рамках столбцовой схемы, но попытка сделать это же в рамках клеточной и строчной схемы приводит к дисбалансу загрузки процессоров как на этапе построения H, так и при умножении матрицы H на вектор.

Для балансировки загрузки процессоров следует использовать известную зеркальную циклическую схему разбиения, которая показана на фиг. 3 на примере решетки процессоров 4×4.

а)



б)



Фиг. 3.

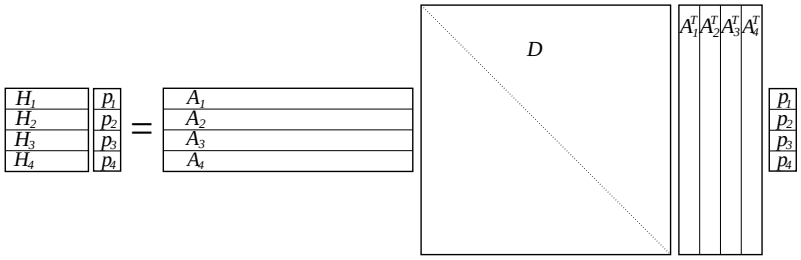
На фиг. 3а показаны варианты разбиения матрицы Гессе, а на фиг. 3б приведено разбиение матриц A, A^T и D и векторов.

В этом примере используется аналог клеточной схемы, но строки матрицы A разбиваются на группы, в которых каждая (блочная) строка присваивается своему процессору, причем при переходе от группы к группе последовательность нумерации блочных строк в группе меняется на противоположную. При этом для представления матрицы H можно использовать две разные схемы хранения, показанные на фиг. 3а. При таком разбиении число операций на один процессор при заполнении матрицы H, а также при ее умножении на вектор варьируется очень слабо. Такая схема хранения не была

реализована, вместо этого она моделировалась схемой, в которой H заполняется как общая несимметричная матрица. Здесь нужно иметь в виду, что такой подход может занижать величину α почти в два раза и реальные значения ускорения при циклической схеме могут быть меньше, чем полученные по клеточной схеме, хотя времена счета циклической схемы могут быть меньше.

4.6. Разбиение данных: строчная схема

Можно рассматривать два достаточно сильно отличающихся варианта реализации строчной схемы разбиения данных, которая показана на фиг. 4 на примере 4 процессоров.



Фиг. 4.

Первый вариант можно рассматривать как частный случай клеточной схемы, в которой $n_c = 1$, так что грубая оценка сверху для ускорения строчной схемы имеет вид

$$s_u(n_r \times 1) = n_r \frac{1 + 2\alpha}{1 + (n_r + 1)\alpha}. \tag{35}$$

Таким образом, при $\alpha = 0.1$ получаем $s_u(8 \times 1) = 5.05$. Основным недостатком этой схемы является то, что вся матрица A должна храниться на каждом из процессоров, так что требуемая память для этого алгоритма не может быть меньше, чем ρmnn_p. Неоптимальность по памяти сильно ограничивает максимальный размер решаемых задач, так что возникает проблема построения алгоритма, который является оптимальным по памяти, когда на каждом из процессоров хранится блочная строка матрицы A.

Был реализован так называемый "безматричный" алгоритм, в котором матрица H вообще не формируется, находится лишь ее главная диагональ, а для умножения матрицы H на вектор p используется последовательность операций $q = ADA^T p$, или

$$q_i = \sum_{j=1}^{n_p} A_j D A_j^T p_j,$$

где вектор p_j принадлежит j-му процессору. Для того чтобы избежать обмена векторами длины n, используется разреженность диагональной матрицы D, из которой следует, что число ненулевых элементов в векторе $DA_j^T p_j$ не может быть больше, чем в D. Таким образом, после упаковки разреженных векторов длину векторов, которые каждый процессор передает своим соседям, можно оценить как $2\rho_z n$, что может быть существенно меньше n.

К сожалению, на этапе построения матрицы D требуется величина $A^T p$, так что одного суммирования n_p векторов длины n на одну итерацию метода Ньютона избежать не удается. Для оптимизации этого этапа можно использовать наложение вычислений и обменов. От "безматричного" метода трудно ожидать эффективного распараллеливания. Скорее хотелось бы, чтобы параллельная версия алгоритма не очень замедлялась по сравнению с последовательной. Тогда параллельная реализация оказывается намного

эффективнее по сравнению с методами, использующими виртуальную память и дисковое пространство ЭВМ в случае, когда требуемая память превышает размер доступной оперативной памяти.

С другой стороны, если матрица A является сверхразреженной в том смысле, что и матрицу H также можно полагать разреженной, то строчная схема может оказаться весьма эффективной, хотя и должна быть существенно модифицирована. Кроме того, в рамках строчной схемы можно весьма эффективно реализовать параллельный предобусловленный метод сопряженных градиентов для решения линейной системы, причем в качестве предобусловливателя использовать безотказную неполную LU-факторизацию (см. [12]).

5. РЕЗУЛЬТАТЫ ЧИСЛЕННЫХ ЭКСПЕРИМЕНТОВ

Для численных экспериментов использовался генератор случайных тестовых задач ЛП, описанный в разд. 3.

Расчеты проводились на параллельном вычислительном кластере МВС-6000IM, состоящем из двухпроцессорных узлов на основе Intel Itanium 2 с частотой 1.6 GHz, соединенных сетью Myrinet 2000.

Результаты расчетов представлены в таблицах 2–5. В них T_{tot} означает время решения задачи ЛП в секундах, T_{lin} — время, потраченное на решение систем линейных уравнений (29), а T_{rem} — время на оставшиеся вычисления. Через s_{tot} обозначено параллельное ускорение при решении задачи ЛП, s_{lin} обозначает ускорение при решении линейных систем, а s_{rem} — ускорение оставшихся вычислений. В качестве значения β бралось 100, что в данных задачах превосходило β_*. Всюду вектор $\hat{x} = 0_n$, т.е. в задаче (3) находилось нормальное решение.

В табл. 2 представлены результаты расчетов с помощью столбцовой схемы при $m = 10^4$, $n = 10^6$, $\rho = 0.01$.

В табл. 3 приведены результаты расчетов с помощью строчной схемы при $m = 5000$, $n = 10^6$, $\rho = 0.01$.

И наконец, результаты расчетов для клеточной схемы при $m = 10^4$, $n = 10^6$, $\rho = 0.01$ приведены в табл. 4.

Таблица 2

T,	s	n_p						
		1	2	4	8	16	32	64
T_{tot}		1439.32	1311.40	568.79	408.94	236.68	188.17	142.65
	s_{tot}	1	1.10	2.53	3.52	6.08	7.65	10.09
T_{lin}		121.88	124.42	122.09	120.60	116.87	109.80	106.17
	s_{lin}	1	0.98	1.00	1.01	1.04	1.11	1.15
T_{rem}		1317.35	1186.69	446.61	288.25	119.73	78.30	36.40
	s_{rem}	1	1.11	2.95	4.57	11.00	16.82	36.19

Таблица 3

T,	s	n_p						
		1	2	4	8	16	32	64
T_{tot}		406.66	242.78	121.32	62.02	32.13	18.43	15.49
	s_{tot}	1	1.67	3.35	6.56	12.66	22.07	26.24
T_{lin}		31.47	16.45	8.36	4.40	2.28	1.55	2.09
	s_{lin}	1	1.91	3.77	7.16	13.81	20.24	15.04
T_{rem}		375.13	226.24	112.88	57.53	29.76	16.79	13.32
	s_{rem}	1	1.66	3.32	6.52	12.60	22.35	28.16

Таблица 4

T,	s	$n_r \times n_c$					
		1×1	2×2	4×4	8×8	10×10	12×12
T_{tot}		1439.32	502.98	145.13	45.65	36.77	28.21
	s_{tot}	1	2.86	9.92	31.53	39.14	51.03
T_{lin}		121.88	62.17	31.66	15.89	12.94	11.26
	s_{lin}	1	1.96	3.85	7.67	9.42	10.82
T_{rem}		1317.35	440.73	113.43	29.74	23.79	16.90
	s_{rem}	1	2.99	11.61	44.30	55.37	77.94

Из табл. 1–4 видно, что наилучшее параллельное ускорение было получено с использованием строчной и клеточной схем разбиения данных. Максимальное ускорение, равное 51.03 на 144 процессорах, было получено при помощи клеточной схемы. Заметим, что слабое ускорение для столбцовой схемы на двух процессорах, по-видимому, можно объяснить тем, что часть вычислений на одном процессоре на самом деле исполнялась на двух ядрах за счет автоматического распараллеливания.

В табл. 5 представлены времена расчетов для безматричной схемы и чебышёвские нормы невязок критериев (24) для разных задач и разного количества процессоров.

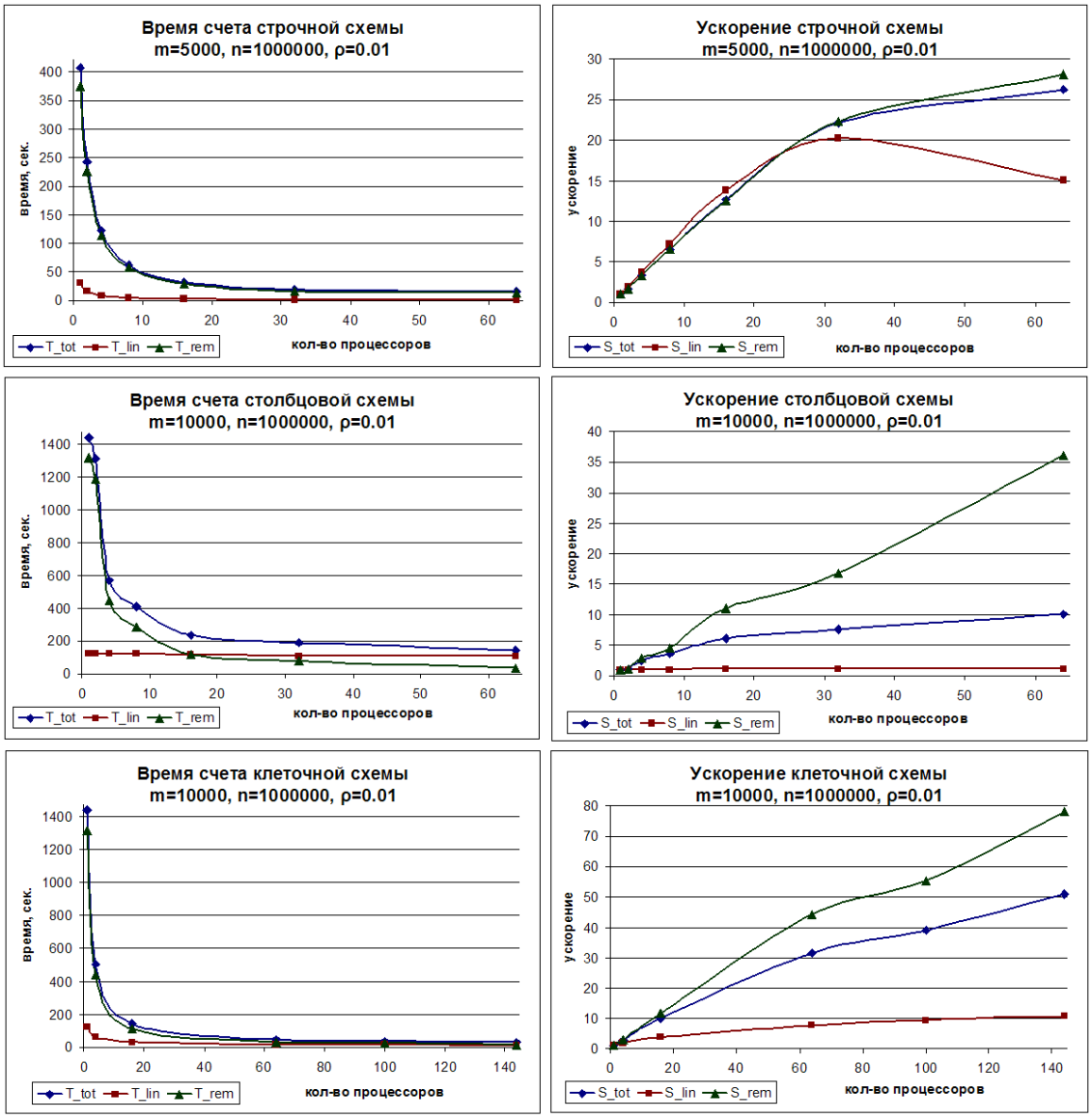
Таблица 5

$m \times n \times \rho$	n_p	T_{tot}	Δ_1	Δ_2	Δ_3
$5 \cdot 10^4 \times 10^6 \times 0.01$	16	400.02	$2.1 \cdot 10^{-6}$	$2.1 \cdot 10^{-7}$	$1.1 \cdot 10^{-10}$
$10^5 \times 10^6 \times 0.01$	20	484.62	$8.1 \cdot 10^{-6}$	$3.6 \cdot 10^{-6}$	$5.6 \cdot 10^{-11}$
$10^5 \times 2 \cdot 10^6 \times 0.01$	40	823.13	$4.5 \cdot 10^{-6}$	$4.2 \cdot 10^{-7}$	$7.2 \cdot 10^{-11}$
$2 \cdot 10^5 \times 2 \cdot 10^6 \times 0.01$	80	2317.42	$4.9 \cdot 10^{-5}$	$6.6 \cdot 10^{-6}$	$5.2 \cdot 10^{-10}$

Использование безматричной схемы позволило решать задачи ЛП с наибольшим числом ограничений m по сравнению с другими схемами. Так, задача ЛП с двумя миллионами переменных при двухстах тысячах ограничений на 80 процессорах была решена менее чем за 40 мин. Эта схема наиболее рационально использует память кластера, поэтому ее целесообразно применять при больших значениях m и n. Ускорение для этой схемы при сравнительно малых m может оказаться меньше единицы. с другой стороны, в наиболее интересном и трудном случае больших m эта схема весьма эффективна

и в некоторых случаях приводила к ускорениям, большим единицы. В любом случае использование безматричной схемы существенно эффективнее по сравнению с методами, использующими медленную дисковую память при нехватке оперативной памяти.

Время счета типичных тестовых задач для различных схем разбиения данных и параллельное ускорения показаны на фиг. 5.



Фиг. 5

6. ВЫВОДЫ И НАПРАВЛЕНИЯ ДАЛЬНЕЙШИХ ИССЛЕДОВАНИЙ

Разработаны и реализованы параллельные версии обобщенного метода ньютона для решения задач линейного программирования, основанные на различных схемах распределения данных, таких как столбцовая, строчная и клеточная схемы. Полученные параллельные алгоритмы успешно применены для решения задач ЛП большой размерности — до нескольких миллионов неизвестных и нескольких сотен тысяч ограничений при достаточно сильно заполненной матрице A.

Как и ожидалось, разработка эффективного решателя оказалось весьма сложной задачей, так что для каждого параллельного варианта алгоритма искался разумный

компромисс между вычислительной масштабируемостью и масштабируемостью по памяти. Наилучшее ускорение было получено для клеточной схемы. Однако, в зависимости от размерности задачи и структуры разреженности матрицы A, оптимальной может оказаться как клеточная, так строчная или столбцовая схема.

Затраты для построения обобщенной матрицы Гессе, в принципе, можно заметно сократить, если на каждой итерации метода Ньютона формировать лишь переменную поправку к матрице Гессе.

Продемонстрированное ускорение является вполне приемлемым (около 30 на 64 процессорах кластера), но нужно иметь в виду, что параллельное задание входных данных для решателя само по себе является нетривиальной задачей и может потребовать время большее, чем на параллельное решение задачи.

В представленной реализации параллельного алгоритма не использовалось распараллеливание на общей памяти. Можно ожидать, что комбинация алгоритмов, основанных на комбинации распределенной и общей памяти с использованием MPI/OpenMP позволит получить более эффективные алгоритмы на современных многоядерных архитектурах вычислительных кластеров.

Важным фактором в ускорении вычислений является оптимизация операций с плотными и разреженными матрицами. Для эффективной работы с плотными матрицами использовались функции из библиотеки Lapack. Однако матрично-векторные операции с разреженными матрицами оставляют дополнительные возможности для оптимизации. В частности, можно использовать специальные способы представления разреженных матриц, сочетающие переносимость и учет характеристик кэш-памяти современных процессоров (см. [13], [14]) и позволяющие существенно уменьшить время на вычисление матрично-векторных умножений.

Литература

1. *Еремин И.И.* Теория линейной оптимизации. Екатеринбург: Изд-во УрО РАН, 1998.

2. *Васильев Ф.П., Иваницкий А.Ю.* Линейное программирование. М.: Факториал, 2003.

3. *Голиков А.И., Евтушенко Ю.Г.* Метод решения задач линейного программирования большой размерности // Докл. РАН. 2004. Т. 397. N° 6. С. 727–732.

4. *Голиков А.И., Евтушенко Ю.Г., Моллаверди Н.* Применение метода Ньютона к решению задач линейного программирования большой размерности // Ж. вычисл. матем. и матем. физ. 2004. Т. 44. N° 9. С. 1564-1573.

5. *Голиков А.И., Евтушенко Ю.Г.* Отыскание нормальных решений в задачах линейного программирования // Ж. вычисл. матем. и матем. физ. 2000. Т. 40. N° 12. С. 1766-1786.

6. *Kanzow C., Qi H., Qi L.* On the Minimum Norm Solution of Linear Programs // J. of Optimizat. Theory and Appl. 2003. V. 116. P. 333–345.

7. *Mangasarian O.L.* A Newton Method for Linear Programming // J. of Optimizat. Theory and Appl. 2004. V. 121. P. 1-18.

8. *Meszaros Cs.* The BPMPD interior point solver for convex quadratic programming problems // Optimizat. Meth. and Software. 1999. V. 11&12. P. 431–449.

9. *Andersen E.D., Andersen K.D.* The MOSEK interior point optimizer for linear programming: an implementation of homogeneous algorithm // High Performance Optimizat. New York: Kluwer, 2000. P. 197–232.

10. *Karypis G., Gupta A., Kumar V.* A parallel formulation of interior point algorithms // Proc. of Supercomputing. 1994. P. 204–213.

11. *Coleman T.F., Czyzyk J., Sun C., Wagner M., Wright S.J.* pPCx: Parallel Software for Linear programming // Proc. Eighth SIAM Conf. Parallel Proc. Sci. Comput., PPSC 1997, March 14–17, 1997. Minneapolis, Minnesota, USA: SIAM, 1997.

12. *Kaporin I.E.* High quality preconditioning of a general symmetric positive definite matrix based on its UTU+UTR+RTU-decomposition // Numer. Linear Algebra Appl. 1998. V. 5. P. 483–509.

13. *Frigo M., Leiserson C.E., Prokop H., Ramachandran S.* Cache-oblivious algorithms
Proc. 40th Ann. Symposium on Foundations of Computer Sci. New York, October, 1999. P. 285–297

14. *Brodal G.S.* Cache-oblivious algorithms and data structures // Proc. 9th Scandinavian Workshop on Algorithm Theory. Vol. 3111 of Lect. Not. in Comput. Sci. Berlin: Springer, 2004. P. 3–13.