Сравнение параллельных реализаций симплекс-метода для безошибочного решения задач линейного программирования *

А.В. Панюков, В.В. Горбик

Южно-Уральский государственный университет

В работе рассмотрены подходы к решению задачи линейного программирования с произвольно заданной точностью. Абсолютная точность вычислений достигается применением в алгоритмах симплекс-метода дробно-рациональных вычислений без округления. Если при этом m – минимальная из размерностей задачи, l – число бит, необходимых под один численный элемент исходных данных, то пространственная сложность алгоритма не превосходит $4lm^4 + o(m^3)$, при этом вычислительная сложность одной итерации симплекс-метода не превосходит $O(lm^4)$, а эффективность распараллеливания (т.е. отношение ускорения к числу процессоров) в предложенной реализации параллельного алгоритма составляет в асимптотике 100%. Известные полиномиальные алгоритмы в линейном программировании предполагают приближенные вычисления с наперед заданной точностью, которые могут быть реализованы с помощью представления данных в формате с плавающей точкой необходимой длины. Представлены результаты вычислительных экспериментов на основе реализаций параллельных версий алгоритмов решения задач линейного программирования.

1. Введение

В настоящее время широко распространены предрассудки, не основанные на доказательствах и порождающие ошибки в расчетах: (1) распространение свойства ассоциативности операций сложения и умножения в поле действительных чисел на конечное множество машинных “действительных” чисел; (2) распространение свойства непрерывной зависимости от параметров решения системы, полученной после «эквивалентных» преобразований, на исходную систему имеют популярные коммерческие пакеты MatLab, MathCad и т.п., а также свободно распространяемый пакет SciLab. Использование в вычислениях разного числа процессоров во многих случаях дает существенно различающиеся результаты, демонстрируя необходимость доказательных вычислений (см. работы [1], [2], [3], [4] [5]).

Потенциал имеющихся пакетов, поддерживающих символические вычисления, не позволяет решать реальные проблемы математического и имитационного моделирования. Возможность обеспечения вычислений с произвольной точностью в программах пользователя дает библиотека GMP [6]. Однако, ее использование требует от пользователя разработки собственного интерфейса для организации распределенных и параллельных вычислений [7]. Развитием библиотеки GMP является библиотека ExactComputational [8], которая предоставляет своим объектам возможность их использования в параллельных вычислениях.

Ориентация на применение многопроцессорных вычислительных систем в составе персональных компьютеров или рабочих станций (параллельные вычисления) и на применение сетевых технологий (распределенные вычисления) требует разработки новых параллельных методов их решения. Они должны быть лишены недостатков «традиционных» методов таких, например, как последовательный характер вычислений. Анализ способов распараллеливания показывает эффективность распараллеливания «по информации». Поэтому, весьма перспективной становится SPMD-технология программирования (Single Program - Multiple Data). При этой технологии вычислительный процесс строится на основе единственной программы, запускаемой на всех процессорах вычислительной системы или на

*Работа выполнена при поддержке РФФИ, проект № 10-07-96003-р_урал_а

многих станциях локальной сети. Копии программы могут выполняться по разным ветвям алгоритма, обрабатывая подмножества данных. Неизбежна синхронизация во времени и при обработке общих данных. Данная идеология используется в стандарте MPI (Message Passing Interface [9]). Такая технология параллельного программирования и обусловила разработку соответствующих методов. В то же время не отрицаются известные традиционные методы, сокращающие общее число операций и исключающие перебор. Иной методологический подход открывает дорогу к решению задач большой размерности и эффективной параллельной работе многих процессоров.

Ранее авторами разработаны алгоритмы и программное обеспечение для абсолютно точного решения систем линейных алгебраических уравнений [10] и вычисления обобщенной обратной матрицы Мура-Пенроуза [11] на многопроцессорных вычислительных системах с использованием классов overlong и rational из библиотеки ExactComputational [8]. Теоретическое и практическое исследование данного программного обеспечения демонстрирует высокую эффективность использования многопроцессорных вычислительных систем.

Целью работы является развитие программного обеспечения безошибочных дробно-рациональных и приближенных вычислений, обеспечивающих заданную гарантированную оценку погрешности, для параллельных и распределенных вычислительных систем и его применение для решения задач линейного программирования.

2. Техника реализации симплекс-метода

Применение симплекс-метода для практических задач линейного программирования остается вне конкуренции, несмотря на появившиеся полиномиальные алгоритмы. В настоящее время используются две техники реализации симплекс-метода:

- метод симплекс таблиц;
- метод обратной матрицы (модифицированный симплекс метод).

Для сохранения целостности изложения приведем их особенности на примере задачи линейного программирования

$$\max\left\{c^T x \ : Ax = b \geq 0,\ x \geq 0;\ c, x \in \mathbf{R}^n;\ b \in \mathbf{R}^m\right\}. \tag{1}$$

2.1. Метод симплекс-таблиц

Данный метод на итерации k пересчитывает симплекс-таблицу

$$S^{(k)} = \left|\begin{array}{c|c} Z^{(k)} = c_{B(k)}^T B(k)^{-1} b & z^{(k)} = -c^T + c_{B(k)}^T B(k)^{-1} A \\ \hline X_{B(k)} = B(k)^{-1} b & B(k)^{-1} A \end{array}\right|$$

где $B(k)$ – базисная матрица, содержащая все относящиеся к базисным переменным k-й итерации столбцы матрицы A (базисные столбцы); $c_{B(k)}$ – вектор коэффициентов целевой функции, относящихся к базисным переменным k-й итерации. При этом левый столбец симплекс-таблицы содержит вектор $X_{B(k)} = B(k)^{-1}b$ значений базисных переменных k-й итерации и значение $Z^{(k)}$ целевой функции на этом решении. Столбцы матрицы S, соответствующие базисным переменным, являются ортами (т.е. из них может быть составлена единичная матрица). Верхняя строка содержит вектор $z^{(k)} = -c^T + c_{B(k)}^T B(k)^{-1} A$ невязок двойственной задачи. Значения элементов строки $z^{(k)}$, относящиеся к базисным столбцам являются нулевыми.

Критерием оптимальности текущего базисного решения является неотрицательность вектора z. Если же существует небазисная переменная $x_i:\ z_i^{(k)} < 0$, то ее введение в число базисных приведет к увеличению целевой функции задачи. В этом случае если множество $L = \{l:\ S_{li}^{(k)} > 0\} = \emptyset$, то целевая функция задачи неограничена, иначе введение переменной x_i в число базисных приведет к увеличению целевой функции на величину

$$\Delta_i = -\frac{X_{B(k)_{l^*}} \cdot z_i^{(k)}}{S_{l^*i}^{(k)}}, \quad \text{где } l^* = \arg\min_{l \in L} \frac{X_{B(k)_{l^*}}}{S_{li}^{(k)}}. \tag{2}$$

Переход от таблицы k-й итерации к таблице $(k+1)$-й итерации осуществляется применением процедуры исключения Жордана-Гаусса для столбца i (ведущего столбца), используя в качестве ведущей строку l^*

$$(\forall l = 0,1,2,\ldots,m;\ j = 0,1,2,\ldots,n)\left(S_{lj}^{(k+1)} = \begin{cases} S_{lj}^{(k)} - \dfrac{S_{l^*j}^{(k)}}{S_{l^*i}^{(k)}} S_{li}^{(k)}, & \text{если } l \neq l^*, \\ \dfrac{S_{l^*j}^{(k)}}{S_{l^*i}^{(k)}}, & \text{если } l = l^* \end{cases} \right). \tag{3}$$

Легко заметить, что выполнение итерации, включая пересчет симплекс-таблицы, потребует не более $(m+n)$ операций деления и сравнения, а также не менее $m(n+1)$ операций сложения и умножения. Алгебраическая пространственная сложность табличного симплекс-метода в основном определяется числом операндов в симплекс-таблице, т.е. равна $mn + O(m)$.

2.2. Метод обратной матрицы

В данном методе, в отличие от табличного, на каждой итерации вместо пересчета симплекс-таблицы пересчитывается матрица $B(k)^{-1}$. Наличие обратной матрицы позволяет для текущего базиса легко находить $y(k)^T = c_{B(k)}^T B(k)^{-1}$ – соответствующее решение двойственной задачи. Текущий базис является оптимальным если соответствующее ему двойственное решение допустимо: $y(k)^T A \geq c^T$. Если же в матрице A найдется столбец $A_{i(k)}:\ y(k)^T A_{i(k)} < c_{i(k)}$, то введение его в число базисных приведет к увеличению целевой функции.

Образом столбца $A_{i(k)}$ в симплекс-таблице $S(k)$ будет вектор $g = B(k)^{-1} A_{i(k)}$. Поэтому ведущей строкой, определяющей выводимый из базиса столбец, будет

$$r = \arg \min_{l:\ g_l > 0} \frac{X_{B(k)_l}}{g_l}, \tag{4}$$

а новые значения базисных переменных равны

$$(\forall l = 1,2,\ldots,m)\left(X_{B(k+1)_l} = \begin{cases} X_{B(k)_l} - \dfrac{g_l}{g_r} X_{B(k)_r}, & \text{если } l \neq r, \\ \dfrac{X_{B(k)_l}}{g_r}, & \text{если } l = r; \end{cases} \right). \tag{5}$$

Базисные матрицы $B(k) = \left(b_{lj}^{(k)}\right)_{l,j=1}^{m}$ и $B(k+1) = \left(b_{lj}^{(k+1)}\right)_{l,j=1}^{m}$ отличаются только r-м столбцом, поэтому элементы обратной матрицы $B(k+1)^{-1} = \left(\beta_{lj}^{(k+1)}\right)_{l,j=1}^{m}$ следующим образом вычисляются через элементы матрицы $B(k)^{-1} = \left(\beta_{lj}^{(k)}\right)_{l,j=1}^{m}$

$$\beta_{lj}^{(k+1)} = \begin{cases} \beta_{lj}^{(k)} - \dfrac{g_l}{g_r} \beta_{rj}^{(k)}, & \text{если } l \neq r, \\ \dfrac{1}{g_r} \beta_{rj}^{(k)}, & \text{если } l = r. \end{cases} \tag{6}$$

Оценим алгебраическую вычислительную сложность рассматриваемого метода. Очевидно, что в общем случае на заключительной итерации потребуется проверка всех n ограничений двойственной задачи, для чего потребуется не более mn операций умножения и $m(n-1)$ операций сложения. На промежуточных итерациях для установления недопустимости двойственного решения эта величина как правило является существенно меньшей. Пересчет обратной матрицы потребует не более m операций деления, не более m^2 операций умножения и не более (m^2-1) операций сложения/вычитания. Поскольку $m < n$,то затраты вычислительных ресурсов на пересчет обратной матрицы могут быть существенно ниже затрат на пересчет симплекс-таблицы. Алгебраическая пространственная сложность метода обратной матрицы определяется числом элементов в исходных данных и в обратной матрице, т.е. равна $mn + m^2 + O(m)$, т.е. больше чем в табличном симплекс-методе.

2.3. Сопоставление методов

Подводя итог изложенному, делаем вывод, что применение метода обратной матрицы оправдано когда

- требуется найти решение двойственной задачи;
- число n существенно превосходит m;
- матрица A разрежена;
- возможна оптимизация проверки допустимости двойственного решения.

Отметим, что алгебраическая сложность дает адекватные оценки используемого вычислительного ресурса только когда все операнды имеют одинаковую длину, например, при использовании стандартных типов данных. При выполнении дробно-рациональных вычислений без округления память необходимая для операндов динамически изменяется (как правило возрастает), поэтому в этом случае более адекватными являются оценки фактически достаточного объема памяти и числа элементарных операций с битами. Эти величины принято называть битовой сложностью (пространственной и вычислительной соответственно).

2.4. Битовая сложность абсолютно точной реализации симплекс-метода

Практическая реализуемость вычислений без округления, в частности, безошибочного решения задачи линейного программирования, определяется требуемыми для вычислений ресурсами: числом бит оперативной памяти и количеством операций с битами. Далее через $S(\lambda)$ будем обозначать число бит, требуемое для представления объекта λ, а через $C(\lambda)$ – число битовых операций, выполненных при нахождении представления объекта λ. Например, число бит, требуемое для представления целых чисел будет равно $S(0) = 1$, $(\forall z \in \mathbf{Z} \setminus \{0\})S(z) = \lceil \log_2 |z| \rceil$. Число бит, требуемое для представления рационального числа $r = p/q$, $p, q \in \mathbf{Z}$ имеет верхнюю оценку $S(r) \le O(S(p) + S(q))$.

Легко проверить, что если $S(p)$, $S(q)$ – память, требуемая для представления рациональных чисел p, q, то память, требуемая для представления результата арифметической операции $\circ \in \{+, -, /, \times\}$ над данными числами будет $S(p \circ q) \le S(p) + S(q)$. Для *битовой* вычислительной сложности выполнения операции $\circ \in \{+, -, /, \times\}$ с помощью классических арифметических алгоритмов (умножение/деление столбиком) справедлива оценка $C(p \circ q) \le S(p)S(q)$. Использование алгоритмов быстрого умножения дает оценку $C(p \circ q) \le S(p) + S(q)$, которая будет использована в работе.

Оценим число бит оперативной памяти достаточное для решения задачи линейного программирования с применением вычислений без округления. Поскольку как элементы симплекс-таблицы, так и элементы обратной матрицы являются решениями систем линейных алгебраических уравнений, то сначала найдем число бит, требуемое для представления

определителя матрицы с элементами, имеющими заданную верхнюю оценку пространственной сложности.

Утверждение 1 *Пусть $B = (b_{ij})$ – целочисленная $m \times m$ матрица, $l = \max\limits_{i,j=1,2,\ldots,m} S(a_{ij})$. Тогда*

$$S(\det A) \le n(\log_2 m + l).$$

Доказательство. Рассмотрим верхнюю оценку абсолютной величины определителя

$$|\det B| = \left| \sum_{\sigma} \prod_{k=1}^{m} b_{k\sigma(k)} \right| \le \sum_{\sigma} \prod_{k=1}^{m} \left| b_{k\sigma(k)} \right| \le m! L^m \le (mL)^m,$$

где $L = \max\{|b_{ij}| : \ i, j = 1, 2, \ldots, m\}$.

Из данной оценки следует $S(\det B) = \log_2 |\det B| \le m(\log_2 m + l)$ ■

Утверждение 2 *Пусть $B = (b_{ij})$ – $m \times m$ рациональная матрица, $l = \max\limits_{i,j=1,2,\ldots,m} S(b_{ij})$. Тогда $S(\det B) \le m(\log_2 m + (2m+1)l)$.*

Доказательство. Пусть K_r равно наименьшему общему всех кратному всех знаменателей строки $r = 1, 2, \ldots, m$ матрицы B. Очевидно, что $S(K_r) \le lm$. Пусть K представляет диагональную матрицу: $\mathtt{diag}(K) = \{K_r : \ r = 1, 2, \ldots, m\}$. Рассмотрим матрицу $\tilde{B} = KB$. Данная матрица является целочисленной. Верхняя оценка числа бит, необходимых для одного элемента матрицы $\tilde{B}$, имеет вид

$$\tilde{l} = \max_{i,j=1,2,\ldots,m} S(\tilde{b}_{ij}) = \max_{i,j=1,2,\ldots,m} S(K_i b_{ij}) \le l(m+1).$$

Принимая во внимание утверждение 1, а также $\log_2 |\det K^{-1}| \le lm$, получим

$$S(\det B) = S\left(\det K^{-1} \cdot \det(\tilde{B})\right) \le m(\log_2 m + (2m+1)l) \ \blacksquare$$

Из формул Крамера для систем линейных алгебраических уравнений и доказанных утверждений вытекает

Утверждение 3 *Если один численный элемент исходных данных имеет пространственную сложность не более l, то*

- *элементы симплекс-таблицы и обратной матрицы имеют пространственную сложность не более $4lm^2 + lm + m\log_2 m + 1$;*
- *столбец симплекс-таблицы и обратной матрицы имеют пространственную сложность не более $4lm^3 + lm^2 + m^2 \log_2 m + m$;*
- *для представления симплекс-таблицы достаточно $4lm^3 n + o(lm^2 n)$ бит;*
- *для представления обратной матрицы достаточно $4lm^4 + o(lm^3)$ бит.*

Поскольку $m < n$, а наиболее критическим ресурсом при использовании точных дробно-рациональных вычислений является память, то целесообразным является использование метода обратной матрицы.

3. Параллельные версии симплекс метода

3.1. Метод симплекс-таблиц

Для реализации метода симплекс-таблиц наиболее подходящей является декомпозиция симплекс-таблицы по столбцам на число блоков равное числу процессоров N. Все столбцы

Процесс $K = 1, 2, \ldots, N$				
$S_{00} = Z$	$z_{\lceil \frac{(K-1)n}{N} \rceil + 1}$	$z_{\lceil \frac{(K-1)n}{N} \rceil + 2}$	$\cdots$	$z_{\lceil \frac{Kn}{N} \rceil}$
$S_{10} = X_{B1}$	$S_{1\lceil \frac{(K-1)n}{N} \rceil + 1}$	$S_{1\lceil \frac{(K-1)n}{N} \rceil + 2}$	$\cdots$	$S_{1\lceil \frac{Kn}{N} \rceil}$
$S_{20} = X_{B2}$	$S_{2\lceil \frac{(K-1)n}{N} \rceil + 1}$	$S_{2\lceil \frac{(K-1)n}{N} \rceil + 2}$	$\cdots$	$S_{2\lceil \frac{Kn}{N} \rceil}$
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
$S_{m0} = X_{Bm}$	$S_{m\lceil \frac{(K-1)n}{N} \rceil + 1}$	$S_{m\lceil \frac{(K-1)n}{N} \rceil + 2}$	$\cdots$	$S_{m\lceil \frac{Kn}{N} \rceil}$

Рис. 1. Декомпозиция симплекс-таблицы по процессорам

симплекс-таблицы, за исключением левого столбца, делятся в равных пропорциях между N процессами, левый столбец т.е. вектор значений базисных переменных и значение целевой функции на нем, рассылаются всем процессам и обрабатываются ими независимо. Пример разбиения симплекс-таблицы S на блоки $S(K),\ K = 1, 2, \ldots, N$ представлен на рис. 1.

Принятые соглашения позволяют предложить следующую параллельную реализацию итерации метода симплекс-таблиц.

Алгоритм TabularSimplex

k-я итерация

- **Данные:** симплекс-таблица $S(K)$ для каждого процесса $K = 1, 2, \ldots, N$.
- **Шаг 1.** Каждому процессу $K = 1, 2, \ldots, N$ найти столбец $i_K :\ z_{i_K} < 0$. Если столбец i_K не найден, то вернуть "Пусто", иначе найти строку

$$l_K = \arg \min_{l:\ S_{l\, i_K} > 0} \frac{X_{Bl}}{S_{l\, i_K}}.$$

  Если строка l_K не найдена, то завершить выполнение всех процессов и вернуть "Не ограничена". В противном случае вычислить значение $\Delta_{i_K} = -X_{Bl_K} z_{i_K} / S_{l_K i_K}$.
  Комментарий. *При выполнении данного шага будет либо установлена неразрешимость задачи, либо найдены данные для изменения базиса: ведущий столбец i_K – кандидат для ввода в базис, ведущая строка l_K, определяющая столбец выводимый из базиса, и приращение целевой функции Δ_{i_K}, – либо установлено отсутствие таких кандидатов.*

- **Шаг 2.** Если $\{K :\ \exists i_K\} \neq \emptyset$, то определить ведущий процесс

$$K^* = \arg \max_{K:\ \exists i_K} \Delta_{i_K}$$

  и прийти на **Шаг 3**. Иначе каждому процессу $K = 1, 2, \ldots, N$ для

$$k = \lceil \frac{(K-1)n}{N} \rceil + 1,\ \lceil \frac{(K-1)n}{N} \rceil + 2, \ldots, \lceil \frac{Kn}{N} \rceil$$

  положить

$$x_k = \begin{cases} X_{Bl}, & \text{если } (S_{lk} = 1) \bigwedge ((\forall i = 0, 1, \ldots, l-1, l+1, \ldots, m)\, S_{ik} = 0), \\ 0, & \text{в противном случае.} \end{cases}$$

Таблица 1. Алгебраический вычислительный ресурс реализаций метода симплекс-таблиц

Оператор	Один процессор	N процессоров	
	Количество алгебраических операций	Количество пересылаемых операндов	Нагрузка на один процесс
Проверка условия оптимальности	$[1, n]$	–	$[1, n/N]$
Определение ведущей строки	$2m + n + 2$	–	$2m + n/N + 2$
Выбор ведущего процесса	–	N	$\lceil \log_2 N \rceil$
Пересылка ведущего столбца	–	$m + 2$	–
Пересчет симплекс-таблицы	$2(m + 1)(n + 1)$	–	$2m(1 + n/N)$
Итого:	$[1, n] + 2mn +$ $4(m + n + 1)$	$m + N + 2$	$[1, n/N] +$ $2(n/N)(m + 1) +$ $4m + 2 + \log_2 N$

Вернуть x – оптимальное решение задачи, Z – оптимальное значение целевой функции.
Комментарий. *При выполнении данного шага будет либо установлена оптимальность текущего базисного решения задачи и возвращены найденные оптимальные решение и значение целевой функции, либо найден ведущий процесс и продолжен процесс оптимизации.*

- **Шаг 3.** Ведущему процессу K^* передать остальным процессам ведущий столбец

$$S_{i_{K^*}} = (z_{i_{K^*}},\ S_{1i_{K^*}},\ S_{2i_{K^*}},\ \ldots,\ S_{mi_{K^*}})^T$$

и номер ведущей строки l_{K^*}.

- **Шаг 4.** Каждому процессу $K = 1, 2, \ldots, N$ пересчитать по формуле (3) симплекс-таблицу $S(K)$. Перейти к следующей итерации.

- Конец алгоритма

Из изложенного выше видно, что описание параллельной версии табличного симплекс-метода не намного сложнее чем для обычного алгоритма. При этом используется одна широковещательная коммуникация между процессами при нахождении ведущего процесса K^* для обмена значениями Δ_{i_K}, $K = 1, 2, \ldots, N$ и одна широковещательная коммуникация для передачи ведущего столбца $S_{i_{K^*}}$ и числа l_{K^*}. В таблице 3.1 приведена сводка требуемого алгебраического (т.е. количества используемых алгебраических операций) вычислительного ресурса для последовательной и параллельной реализаций метода симплекс-таблиц. Из таблицы и описания алгоритма видно, что процессоры загружены равномерно, а эффективность распараллеливания растет с ростом сложности задачи, достигая в пределе 100%.

3.2. Метод обратной матрицы

Из описания метода обратной матрицы следует, что максимальный эффект от распараллеливания при поиске вводимой в базис переменной достигается при декомпозиции

исходных данных: вектора c^T и матрицы A, – по столбцам в равных пропорциях между процессорами на блоки

$$c(K)^T = \left(\begin{array}{cccc} c_{\lceil \frac{(K-1)n}{N} \rceil+1} & c_{\lceil \frac{(K-1)n}{N} \rceil+2} & \cdots & c_{\lceil \frac{Kn}{N} \rceil} \end{array} \right), \quad K = 1, 2, \ldots, N; \tag{7}$$

$$A(K) = \left(\begin{array}{cccc} a_1{}_{\left(\lceil \frac{(K-1)n}{N} \rceil+1\right)} & a_1{}_{\left(\lceil \frac{(K-1)n}{N} \rceil+2\right)} & \cdots & a_1{}_{\left(\lceil \frac{Kn}{N} \rceil\right)} \\ a_2{}_{\left(\lceil \frac{(K-1)n}{N} \rceil+1\right)} & a_2{}_{\left(\lceil \frac{(K-1)n}{N} \rceil+2\right)} & \cdots & a_2{}_{\left(\lceil \frac{Kn}{N} \rceil\right)} \\ \vdots & \vdots & \ddots & \vdots \\ a_m{}_{\left(\lceil \frac{(K-1)n}{N} \rceil+1\right)} & a_m{}_{\left(\lceil \frac{(K-1)n}{N} \rceil+2\right)} & \cdots & a_m{}_{\left(\lceil \frac{Kn}{N} \rceil\right)} \end{array} \right), \quad K = 1, 2, \ldots, N. \tag{8}$$

При этом предполагается. что вектор двойственных переменных y размещен в каждом процессе $K = 1, 2, \ldots, N$.

Эффект от распараллеливания при пересчете обратной матрицы B^{-1} будет максимальным при ее декомпозиции по строкам в равных пропорциях на блоки по числу процессов

$$B^{-1}(K) = \left(\begin{array}{cccc} b_{\left(\lceil \frac{(K-1)m}{N} \rceil+1\right)1} & b_{\left(\lceil \frac{(K-1)m}{N} \rceil+1\right)2} & \cdots & b_{\left(\lceil \frac{(K-1)m}{N} \rceil+1\right)m} \\ b_{\left(\lceil \frac{(K-1)m}{N} \rceil+2\right)1} & b_{\left(\lceil \frac{(K-1)m}{N} \rceil+2\right)2} & \cdots & b_{\left(\lceil \frac{(K-1)m}{N} \rceil+2\right)m} \\ \vdots & \vdots & \ddots & \vdots \\ b_{\left(\lceil \frac{Km}{N} \rceil\right)1} & b_{\left(\lceil \frac{Km}{N} \rceil+2\right)2} & \cdots & b_{\left(\lceil \frac{Km}{N} \rceil+2\right)m} \end{array} \right), \quad K = 1, 2, \ldots, N. \tag{9}$$

Из описания метода обратной матрицы следует целесообразность декомпозиции вектора базисных переменных X_B и вектора g по таким же блокам строк

$$X_B(K) = \left(\begin{array}{c} x_{B\lceil \frac{(K-1)m}{N} \rceil+1} \\ x_{B\lceil \frac{(K-1)m}{N} \rceil+2} \\ \vdots \\ x_{B\lceil \frac{Km}{N} \rceil} \end{array} \right); \quad c_B(K) = \left(\begin{array}{c} c_{B\lceil \frac{(K-1)m}{N} \rceil+1} \\ c_{B\lceil \frac{(K-1)m}{N} \rceil+2} \\ \vdots \\ c_{B\lceil \frac{Km}{N} \rceil} \end{array} \right);$$

$$g(K) = \left(\begin{array}{c} g_{\lceil \frac{(K-1)m}{N} \rceil+1} \\ g_{\lceil \frac{(K-1)m}{N} \rceil+2} \\ \vdots \\ g_{\lceil \frac{Km}{N} \rceil} \end{array} \right); \qquad K = 1, 2, \ldots, N. \tag{10}$$

Способ (9) декомпозиции обратной матрицы по процессам позволяет каждому процессу вычислить субвектор двойственных переменных

$$y(K) = \left(B(K)^{-1} \right)^T c_{B(K)}. \tag{11}$$

Очевидно, что вектор действенных переменных

$$y = \sum_{K=1}^{N} y(k).$$

Изложенное выше позволяет предложить следующую параллельную реализацию итерации метода обратной матрицы.

Алгоритм InvBasMatrSimplex

k-я итерация

- **Данные:** в каждом процессе $K = 1, 2, \ldots, N$ размещены
  y – вектор двойственных переменных,
  $M(K)$ – вектор номеров базисных переменных процесса,
  $X_B(K)$ – вектор значений базисных переменных процесса,
  $c_B(K)$ – вектор значений коэффициентов целевой функции при базисных переменных процесса,
  $A(K)$ – блок матрицы ограничений процесса,
  $c(K)^T$ – блок коэффициентов целевой функции процесса,
  $B^{-1}(K)$ – блок обратной матрицы процесса.

- **Шаг 1.** Каждому процессу $K = 1, 2, \ldots, N$ найти столбец

$$i_K : \; z_{i_K} = A(K)^T_{i_K} y - c(K)_{i_K} < 0.$$

  Если столбец i_K не найден, то вернуть "Пусто".

- **Шаг 2.** Если $\{K : \; \exists i_K\} \neq \emptyset$, то определить ведущий процесс

$$K^c = \arg \min_{K:\ \exists i_K} z_{i_K}.$$

  и прийти на **Шаг 3**. Иначе сформировать решение задачи: положить $x[1 : n] = 0$, каждому процессу $K = 1, 2, \ldots, N$ для

$$r = \lceil \frac{(K-1)m}{N} \rceil + 1, \; \lceil \frac{(K-1)m}{N} \rceil + 2, \ldots, \lceil \frac{Km}{N} \rceil$$

  положить

$$x_{M(K)_r} = X_{B(K)_r}.$$

  Вернуть x – оптимальное решение задачи, y – оптимальное решение двойственной задачи.

- **Шаг 3.** Процессу K^c разослать всем процессам $K = 1, 2, \ldots, N$ столбец $A_{i_{K^c}}$ и $c_{i_{K^c}}$.

- **Шаг 4.** Каждому процессу $K = 1, 2, \ldots, N$ вычислить $g(K) = B(K)^{-1} A_{i_{K^c}}$ и

$$r(K) = \arg \min_{l:\ g(K)_l > 0} \left[h(K, l) = \frac{X_B(K)_l}{g(K)_l} \right].$$

  Если $r(K)$ не найдено, то вернуть "Не ограничена" и завершить выполнение алгоритма, в противном случае вернуть $r(K), \; h(K, r(K))$.

- **Шаг 5.** Определить ведущий процесс

$$K^r = \arg \min_{K:\ \exists r(K)} h(K, r(K)).$$

- **Шаг 6.** Ведущему процессу K^r разослать всем процессам $K = 1, 2, \ldots, N$ ведущую строку $r^* = r(K^r)$ обратной матрицы

$$\left(B(K)^{-1} \right)^{(r^*)} = \left(\; b_{r^*1}, \quad b_{r^*2}, \quad \cdots, \quad b_{r^*m} \; \right)$$

  и значение $g(K^r)_{r^*}$. Положить $M(K)_{r^*} = i_{K^c}$, $c_B(K)_{r^*} = c_{i_{K^c}}$.

- **Шаг 7.** Каждому процессу $K = 1, 2, \ldots, N$ вычислить новые значения базисных переменных процесса $x_B(K)$ по формулам (5), новый блок обратной матрицы процесса $B^{-1}(K)$ по формулам (6), и двойственное субрешение блока

$$y(K) = \left(B(K)^{-1}\right)^T c_B(K).$$

- **Шаг 8.** Выполнить между процессами глобальный обмен полученными на шаге 8 субрешениями и вычислить значения двойственных переменных

$$y = \sum_{k=1}^{N} y(k).$$

  Перейти к следующей итерации.

- Конец алгоритма

Таким образом, описание параллельной версии метода обратной матрицы оказывается сложнее описания табличного симплекс-метода. В основном это объясняется большей степенью неоднородности используемых структур. При выполнении одной итерации используется пять широковещательных коммуникаций между процессами:

1) широковещательная коммуникация между процессами при нахождении ведущего процесса K^c для обмена значениями z_{i_K}, $K = 1, 2, \ldots, N$,

2) широковещательная коммуникация для передачи вводимого в базис (ведущего) столбца $A_{i_{K^*}}$ и его номера,

3) широковещательная коммуникация между процессами при нахождении ведущего процесса K^r для обмена значениями $x_B(K)_{r(K)}/g(K)_{r(K)}$, $K = 1, 2, \ldots, N$,

4) широковещательная коммуникация для передачи ведущей строки $\left(B(K)^{-1}\right)^{(r^*)}$ и ее номера,

5) широковещательная коммуникация для обмена между процессами двойственными субрешениями и нахождения двойственного решения.

В таблице 3.2 приведена сводка требуемого алгебраического вычислительного ресурса для последовательной и параллельной реализаций метода обратной матрицы. Из таблицы и описания алгоритма видно, что процессоры загружены равномерно, а эффективность распараллеливания растет с ростом сложности задачи, достигая в пределе 100%.

4. Заключение

Положительный эффект от распараллеливания, в случае использования дробно-рациональных вычислений без округления состоит не только в ускорении вычислений, но и возможности решать задачи большей размерности, т.к. достаточно легко достичь границ, когда матрица целиком не будет умещаться в оперативной памяти одного узла.

В основе вычислений всех рассмотренных нами коммерческих программ лежат типы данных с плавающей точкой, поэтому они не могут гарантировать высокую точность решения. Исключением из ряда продуктов, считающих приближенно, являются две открытых реализации симплекс метода, использующих в вычислениях библиотеку точных вычислений GNU MP, что неизбежно ведет к существенному увеличению времени счета. Однако они не используют преимущества параллельного программирования, методы которого, при

Таблица 2. Алгебраический вычислительный ресурс метода обратной матрицы

Оператор	Один процессор	N процессоров	
	Количество алгебраических операций	Количество пересылаемых операндов	Нагрузка на один процесс
Проверка условия оптимальности	$2m[1,n]$	–	$2m[1,n/N]$
Выбор c-ведущего процесса	–	N	$\log_2 N$
Передача ведущего столбца	–	$m+1$	–
Нахождение g	$m(m-1)$	–	$m(m-1)/N$
Нахождение ведущей строки	$2m$	N	$2m/N+\log_2 N$
Модификация X_B	$1+2m$	1	$2m/N$
Модификация B^{-1}	$3m^2$	m	$3(m^2/N)$
Вычисление $y(K)$	–	–	$(m/N)(2(m/N)-1)$
Вычисление y	$m(2m-1)$	N	$\log_2 N$
Итого:	$2m[1,n]+$ $6m^2+2m-1$	$2m+3N+2$	$2m[1,n/N]+$ $6m^2/N+2m/N+$ $3\log_2 N$

умелом использовании, позволяют сократить время расчетов и, следовательно, расширить круг решаемых задач (задачи большей размерности).

Основной задачей данной работы была разработка программы Plinpex решения задачи линейного программирования, основанной на параллельном алгоритме и использующей вычисления без округлений и интервальные вычисления на основе типов данных с плавающей точкой заданной точности. Plinpex основан на предложенных методах для адаптации используемых типов данных к MPI.

Проведенный эксперимент показал, что реализованный метод эффективен на задачах разной размерности и соотношением количества ограничений к количеству переменных. В результате проведенного эксперимента можно сделать выводы о высокой эффективности распараллеленного алгоритма симплекс метода. Согласно проведенным экспериментам, эффективность распараллеливания для типов данных с плавающей точности заданной точности прямо пропорционально зависит от длины мантиссы (количества бит точности) и составляет 70-80%, и показатель эффективности еще выше для точных дробно-рациональных вычислений. Однако, общее время вычислений может быть улучшено некоторыми оптимизациями реализации алгоритма, что является целью для дальнейшей работы.

Список литературы

1. *Ю.Г. Евтушенко, А.И. Голиков.* Параллельные алгоритмы решения задач линейного программирования. // Российская конференция "Дискретная оптимизация и исследование операций": Материалы конференции (Алтай, 27 июня – 3 июля 2010 г.). – Новосибирск: Изд-во Института математики, 2010. – С. 25–29.

2. *В.А. Гаранжа, А.И. Голиков, Ю.Г. Евтушенко.* Параллельная реализация метода Ньютона для решения больших задач линейного программирования. // Журнал вычислительной математики и математической физики. – Т. 49. – No 8. – С. 1369–1384.

3. *V. Y. Pan, J. H. Reif.* Fast and Efficient Parallel Linear Programming and Linear Least Squares Computations //Proceedings of the VLSI Algorithms and Architectures, Aegean Workshop on Computing. – Springer-Verlag. – 1986. P. 283–295.

4. *Ju. Hall.* Towards a practical parallelization of the simplex method // Journal Computational Management Science. – 2010. Vol. 7. – Number 2. – P. 139-170.

5. *G. G. Yarmish.* A Distributed Implementation of the Simplex Method, UMI Dissertations Publishing, 2001.

6. GNU Multiple Precision Arithmetic Library: http://gmplib.org/, 2010.

7. *A. V. Panyukov, V. V. Gorbik* Exact and Guaranteed Accuracy Solutions of Linear Programming Problems by Distributed Computer Systems with MPI // Tambov University REPORTS: A Theoretical and Applied Scientific Journal. Series: Natural and Technical Sciences. – Volume 15, Issue 4, 2010. – P. 1392-1404. http://vestnik.tsutmb.ru/old/index.php?module=subjects&func=viewpage&pageid=165

8. *А.В. Панюков, М.И. Германенко, В.В. Горбик.* Библиотека классов "Exact Computational" / Свидетельство о государственной регистрации программы для ЭВМ № 2009612777 от 29 мая 2009г. // Программы для ЭВМ, базы данных, топологии интегральных микросхем. Официальный бюллетень Российского агентства по патентам и товарным знакам No 3. – 2009. – С. 251.

9. MPI: A Message Passing Interface Standard: http://www.MPI-forum.org/docs/MPI-11-html/MPI-report.html, 1995.

10. *А.В. Панюков, М.И. Германенко, В.В. Горбик.* Параллельные алгоритмы решения систем линейных алгебраических уравнений с применением вычислений без округления // Параллельные вычислительные технологии (ПАВТ–2007): Труды международной научной конференции (Челябинск, 29 января – 2 февраля 2007 г.). – Челябинск: Изд-во ЮУрГУ, 2007, т. 2, с. 238–249.

11. *А.В. Панюков, М.И. Германенко* Приложение для безошибочного нахождения обобщенной обратной матрицы методом Мура-Пенроуза и безошибочное решение систем линейных алгебраических уравнений // Информационные технологии моделирования и управления. – 2009. – No 1 (53). – С. 78–87.

12. Netlib library collection: ftp://netlib2.cs.utk.edu/lp/data, 1996.