ОДНОРОДНЫЕ ВЫЧИСЛИТЕЛЬНЫЕ СИСТЕМЫ ИЗ МИКРО-ЭВМ
(Вычислительные системы)
1983 год Выпуск 97

УДК 681.32:519.68

ПАРАЛЛЕЛЬНЫЕ АЛГОРИТМЫ В ЛИНЕЙНОМ ПРОГРАММИРОВАНИИ

Л.А. Седухина

Введение

Методы линейного программирования широко применяются в экономике, технике и других областях. Для этих задач характерен большой объем вычислений, что часто приводит к невозможности их решения на ЭВМ за приемлемое время. Существенное сокращение времени решения при том же объеме вычислений можно получить при параллельной реализации алгоритма на вычислительной системе с программируемой структурой [1]. Параллельное исполнение алгоритма требует, кроме реализации арифметических операций, проведения операций обмена данными между ветвями алгоритма, что приводит к дополнительным накладным расходам.

В отличие от известных работ по параллельным алгоритмам задач линейного программирования [2,3] в работе предлагаются параллельные алгоритмы, основанные на локальных взаимодействиях одновременно исполняемых частей. Использование локальных взаимодействий существенно минимизирует накладные расходы по обмену данными в системе.

Рассматривается параллельно-поточная интерпретация методов решения задач линейного программирования: метода улучшения плана и симплекс-метода с использованием треугольного разложения базисной матрицы.

1. Параллельная реализация метода последовательного улучшения плана

Задача линейного программирования заключается в нахождении неотрицательного вектора $X \geq 0$, обращающего в максимум линейную форму $L(X) = CX^T$ при условиях $AX^T = B$, т.е. $\max\{(C,X) : AX=B, X \geq 0\}$,

где $A=(A_1,\dots,A_n)=\|a_{ij}\|_{m\times n}$ - матрица условий, A_j и B - соответственно вектор условий и вектор ограничений. Численное решение задачи можно осуществить итерационными методами, представляющими собой модификации симплекс-метода. Эксперименты показывают [4], что число итераций l , требуемых для решения задачи, обычно колеблется от 2m до 3m при $m<n$.

J	C_J	B	A_1	A_2	...	A_k	...	A_n	θ
s_1	c_{s_1}	x^1_{10}	x^1_{11}	x^1_{12}	...	x^1_{1k}	...	x^1_{1n}	θ_1
s_2	c_{s_2}	x^1_{20}	x^1_{21}	x^1_{22}	...	x^1_{2k}	...	x^1_{2n}	θ_2
⋮									
s_r	c_{s_r}	x^1_{r0}	x^1_{r1}	x^1_{r2}	...	x^1_{rk}	...	x^1_{rn}	θ_r
⋮									
s_m	c_{s_m}	x^1_{m0}	x^1_{m1}	x^1_{m2}	...	x^1_{mk}	...	x^1_{mn}	θ_m
		L^1	Δ^1_1	Δ^1_2	...	Δ^1_k	...	Δ^1_n	

Рис. I

Рассмотрим метод последовательного улучшения плана [5]. Этот метод представляет собой процесс решения системы линейных уравнений - условий задачи по схеме полного исключения с особым правилом выбора опорного элемента, обеспечивающим движение по базисам (правило выбора разрешающей строки r) и монотонный рост линейной формы (правило выбора разрешающего столбца k). Особенность, превращающая метод исключения решения системы линейных уравнений в метод решения соответствующей задачи линейного программирования - это специальный выбор направляющего элемента x_{rk} (выбор индексов r и k). Данные, имеющие отношение к l-итерации, удобно структурировать в симплекс-таблице (рис.I).

В алгоритмах, позволяющих решать задачу линейного программирования, сначала выбирается начальный базис, определяемый индексным множеством $J^0=\{s_1,\dots,s_k,\dots,s_m\}$, после чего осуществляется последовательный переход от базиса, определяемого J^0, к так называемому смежному базису, определяемому множеством J вплоть до получения окончательного базиса. Два базиса, определяемые множествами J и J', называют смежными, если множества J и J' отличаются лишь одним индексом, т.е. существуют только два индекса k и r , удовлетворяющие условиям $k\notin J$, $k\in J'$, $r\in J$, $r\notin J'$, которые заменяются один на другой. Тогда говорят, что переменная x_k включается в базис, а x_r исключается из базиса. Переход от базиса J^l к другому базису $J^{(l+1)}$, т.е. преобразование l-й таблицы в (l+1)-ю

таблицу производится по рекуррентным формулам:

$$x_{ij}^{(l+1)} = \begin{cases} x_{ij}^{(l)} - \dfrac{x_{rj}^{(l)}}{x_{rk}^{(l)}} x_{ik}^{(l)} & \text{при} \quad i \neq r, \\ \dfrac{x_{rj}^{(l)}}{x_{rk}^{(l)}} & \text{при} \quad i = r, \end{cases} \qquad (1)$$

$$i = 1,2,\ldots,m+1; \quad j = 0,1,\ldots,n .$$

Выбор $r \in J$ удовлетворяет условиям

$$\theta_r = \min_i \left\{ \frac{x_{i0}^{(l)}}{x_{ik}^{(l)}} : x_{ik} > 0 \right\},$$

индекс $k \in J'$, для которого $x_{m+1,k}^{(l)} = \Delta_k^{(l)} < 0$.

Можно записать алгоритм $(l+1)$-й итерации метода улучшения плана в следующем виде:

Шаг I. Если $\forall j \in \{1,n\}$: $x_{m+1,j}^{(l)} = \Delta_j \geq 0$, то план оптимален и вычисления заканчиваются. Иначе, если $\exists j \in \{1,n\}$: $x_{m+1,j}^{(l)} < 0 \wedge \forall i \in \{1,m\}$: $x_{ij}^{(l)} < 0$, то задача неразрешима, т.е. линейная форма не ограничена сверху и вычисления также заканчиваются. В противном случае переходим к шагу 2.

Шаг 2. Выбираем разрешающий столбец k из условия $x_{m+1,k}^{(l)} = \min_{j \in \{1,n\}} x_{m+1,j}^{(l)}$ и разрешающую строку r из условия

$$\theta_r = \min_{i \in \{1,m\}} \left\{ \frac{x_{i0}^{(l)}}{x_{ik}^{(l)}} : x_{ik} > 0 \right\} .$$

Переходим к шагу 3.

Шаг 3. Производим замену базиса, т.е. преобразуем l-ю таблицу в $(l+1)$-ю по формулам (I). Переходим к следующей итерации и возвращаемся к шагу I.

Вычислительные схемы определения исходного опорного плана сводятся к решению вспомогательной или расширенной задачи по тому же алгоритму [4].

Вспомогательная задача формулируется следующим образом: $\max\{L^* = -ex^a : Ax + x^a = b,\ x \geq 0,\ x^a \geq 0\}$, где $x^{aT} = (x_1^a,\ldots, x_m^a)$ -

вектор искусственных переменных, $e^T = (1,1,\ldots,1)$- m-компонентный вектор. Начальный план вспомогательной задачи составляется из компонент $x = 0$, $x^a = b$. В случае, когда оптимальное значение $L^* = 0$, оптимальный план вспомогательной задачи оказывается опорным планом исходной задачи, при $L^* < 0$ исходная задача не имеет ни одного плана, т.е. система условий задачи несовместна. Задачу линейного программирования в этом случае решают в два этапа: вычисление начального опорного плана, затем определение оптимального плана.

Расширенная задача позволяет объединить эти два этапа и формулируется следующим образом: $\max\{L_M = L(x) - Mx^a : Ax + x^a = b,\ x \geq 0,\ x^a \geq 0\}$, где $M > 0$ - достаточно большое число. Начальный опорный план расширенной задачи $x = 0$, $x^a = b$. Если при решении расширенной задачи $x^a = 0$, то $x^* = (x_1^*, \ldots, x_n^*)$ является оптимальным планом исходной задачи, если $\sum_{i=n+1}^{n+m} x_i^a > 0$, то исходная задача является неразрешимой.

Рассмотрим алгоритм одной итерации и оценим операционную сложность реализации метода в зависимости от параметров задач: m - числа ограничений и n - числа переменных. Когда задача линейного программирования содержит еще m дополнительных искусственных переменных, то числом переменных является $n' = n + m$. Первые два шага алгоритма представляют собой комбинаторные операции: просматривается $(m+1)$-я строка симплекс-таблицы, находится минимальный отрицательный элемент и его индекс k, поиск в n-векторе потребует не больше n операций. На втором шаге просматривается k-й столбец, находится индекс r, на поиск в m-векторе потребуется не более m операций. Таким образом, число операций, требуемых для первых двух шагов, линейно зависит от параметров задачи. На третьем шаге $\forall i \in \{1, m+1\}$: $i \neq r$ нужно r-ю строку ($n+1$ - вектор), умноженную на x_{ik}, вычесть из i-й строки, т.е. симплекс-шаг представляет собой матричное преобразование l-таблицы ($(n+1) \times (m+1)$-матрицы), требующее осуществления порядка $(n+1)\cdot(m+1)$ операций. Симплекс-шаг тем самым вносит в реализацию метода наиболее существенные операционные затраты. Очевидно, что время решения задачи на последовательной ЭВМ будет пропорциональным числу операций, требуемых для ее выполнения. Тогда общее время решения при больших значениях n, m определится величиной $T_1 = l[(n+1)(m+1) + n + m] = l[nm + O(m+n)]$, где l - число итераций.

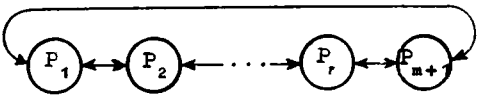


Рис. 2

Рассмотрим теперь параллельную интерпретацию метода улучшения плана на циклической системе, показанной на рис. 2. Распределение данных симплекс-таблицы между (m+1)-м вычислителем системы произведем по строкам так, что каждый вычислитель будет хранить строку симплекс-таблицы и программу обработки. Каждый вычислитель обрабатывает поэлементно свою строку, согласованно обмениваясь результатами или общими данными с соседними вычислителями в соответствии со своей программой обработки.

Первый шаг алгоритма осуществляется в (m+1)-м вычислителе, который находит индекс k в (m+1)-й строке за время порядка n .Чтобы определить индекс r на втором шаге в k-м столбце, элементы которого находятся во всех вычислителях, необходимо кольцо вычислителей пройти дважды. В первый раз каждый i-й вычислитель сравнивает два значения Θ_i и Θ_{i-1}, минимальное из них посылает соседнему вычислителю. Посылаемый вторично индекс r доходит до r-го вычислителя и активизирует его (см. рис.3, где ☒ - состояние работы, ☐ - состояние ожидания). Таким образом, эта операция осуществляется в системе вычислителей за время порядка 2m в худшем случае, т.е. когда r = I.

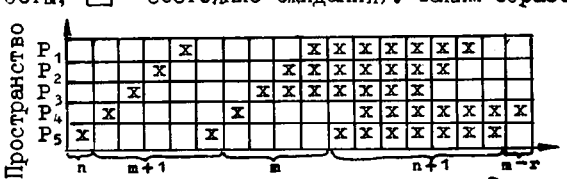


Рис. 3

При выполнении третьего шага активизируется r-й вычислитель, как показано на рис.4, который иллюстрирует динамику вычислений в системе при m = 4, n = 5, r = 3. Каждая клетка на рисунке показывает состояние работы i-го вычислителя на j-м временном шаге, которое заключается в выполнении обобщенной операции: прием операнда $rec_{i\oplus 1}$, вычислении по рекуррентным формулам операнда x_{ij} , передаче операнда $sen_{i\oplus 1}$ ($\oplus$ - сложение по модулю m+1). Определенные в r-м вычислителе элементы r-й строки протягиваются по циклической системе, вычисляя при этом элементы остальных строк. Таким образом, замена базиса в системе осуществится за n временных шагов определения r-м вычислителем элементов разрешающей строки плюс (m+1) шаг задержки распростра -

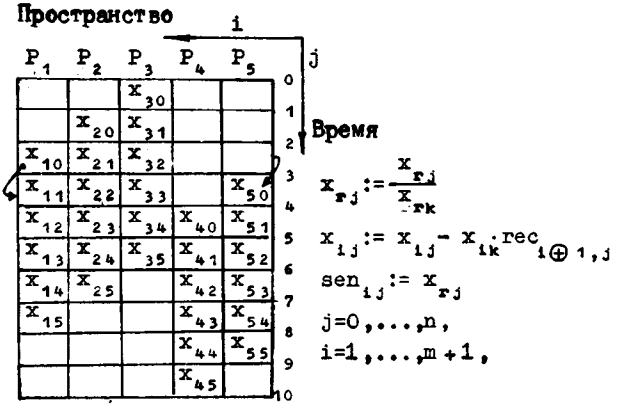


Рис. 4

нения элементов r-й строки и вычисления остальных элементов в (m+1)-м вычислителе системы. В этом случае время параллельного исполнения симплекс-шага определяется величиной (n+m+1) .Общее время параллельной реализации алгоритма на системе определяется величиной порядка $T_m = 2n+3m+O(1)$ (рис.3), на котором показана диаграмма занятости вычислителей при реализации одной итерации.

Степень совмещения операций или ускорение параллельной интерпретации алгоритма на системе из m циклически связанных вычислителей по сравнению с последовательной реализацией определится отношением $S_m = T_1/T_m = nm/(2n+3m) = m\frac{n}{2n+3m}$. Если принять, что $n = cm$, то $S_m = m\frac{c}{3+2c}$. Эффективность использования m вычислителей в системе (см. рис. 3) можно оценить величиной $E_m = S_m/m = \frac{c}{3+2c}$ при $c \gg 1$ $E_m \approx \frac{1}{2}$.В случае, когда $c \ll 1$, т.е. $m \gg n$ целесообразно решать двойственную задачу.

Заметим, что для контроля вычислений через определенное число итераций нужно вычислять условные оценки Δ_j , которые хранятся в (m+1)-м вычислителе, не только по рекуррентным формулам (I), но и непосредственно из соотношений:

$$\Delta_j^{(1)} = z_j - c_j = \sum_{i=1}^{m} c_{s_i}^{(1)} x_{ij}^{(1)} - c_j, \quad j \in \{1,n\},$$

где c_{s_i} - коэффициенты линейной формы при базисных компонентах. Эта операция равносильна умножению транспонированной (mxn)-матрицы условий $A_{(1)}$ на вектор C_s, т.е. $\Delta^{(1)} = A^T_{(1)} C_s^{(1)}$, и требует порядка mn временных шагов. На рис.5 показана параллельная интерпретация этой операции при m = 4, n = 5.

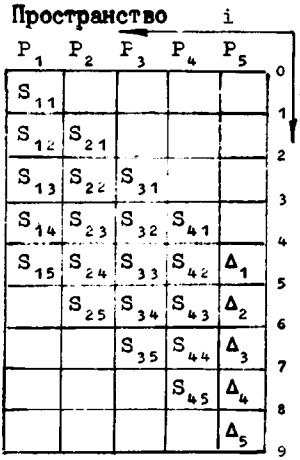


Рис. 5

Каждый вычислитель, начиная с первого, определяет частичную сумму $S^{(1)}_{ij}$ и посылает ее соседнему вычислителю в кольце до (m+1)-го, в котором накапливаются промежуточные значения условных оценок Δ_j. Очевидно, что параллельное исполнение этой операции можно осуществить за время порядка $T_m = n+m$, на рис.5 для n = 5, m = 4, $T_m = 9$.

Таким образом, при параллельной интерпретации метода улучшения плана мы получили линейные временные оценки одной итерации.

Отметим, что во всех вычислительных схемах симплекс-метода используют рекуррентные формулы (I). Применение этих рекуррентных формул эквивалентно умножению матриц $A_{(1)}$, представленных 1-таблицей слева на матрицу Фробениуса F^r:

$$F^r = \begin{bmatrix} 1 & & 0 & -\frac{x_{1k}}{x_{rk}} & & \\ & \ddots & & \vdots & & 0 \\ 0 & & 1 & & & \\ 0 & \cdots & & \frac{1}{x_{rk}} & \cdots & 0 \\ & & & \vdots & 1 & 0 \\ & 0 & & & \ddots & \\ & & & -\frac{x_{mk}}{x_{rk}} & 0 & 1 \end{bmatrix} \begin{matrix} r_1 \\ \\ \\ r_1 \\ \\ \\ \end{matrix}.$$

Например, в модифицированном симплекс-методе на каждой итерации находят обратную базисную матрицу $A_{J(1)}^{-1}$, начиная с единичной, по тем же рекуррентным соотношениям. В некоторых конкретных применениях [6] решение задачи линейного программирования таким способом является неудовлетворительным. Поскольку пара индексов (k,r) и опорный элемент матрицы Фробениуса обычно выбираются так, чтобы обеспечить выполнение условий монотонного роста линейной формы и движения по планам за конечное число итераций, то главный элемент x_{rk} может оказаться очень малым по сравнению с другими x_{ik} и необязательно наилучшим с точки зрения численной реализации метода. С другой стороны, при наличии ошибок округлений ошибки вычисления смежных базисов от итерации к итерации накапливаются.

Чтобы избежать этого, вместо умножения на матрицу Фробениуса на каждой итерации используют LU-разложение базисной матрицы [6,7]. В этом случае процедура решения задачи линейного программирования является одной из разновидностей симплекс-метода, основанного на приведении к треугольному виду текущего базиса A_J согласно уравнению $LA_J = R$, где R - невырожденная верхняя треугольная матрица. Известно, что каждый шаг симплекс-метода требует решения трех систем линейных уравнений с базисной матрицей A_J. При использовании треугольного разложения базиса процесс решения систем линейных уравнений сводится к решению треугольных систем уравнений численно устойчивыми методами [8,9]. Этот метод более трудоемкий, но часто бывает необходим для практического решения задач с плохо обусловленными базисными матрицами [6].

2. Параллельная реализация симплекс-метода с использованием треугольного разложения

Рассмотрим задачу линейного программирования в следующей постановке [7]: найти $\min c_0+c_{-m}x_{-m}+\ldots+c_{-1}x_{-1}+c_1x_1+\ldots+c_nx_n$ при условии, что $x_{-i}+\sum_{j=1}^{n} a_{ij}x_j = b_i$, $i\in\{1,\ldots,m\}$, $x_i \geq 0$, где x_i - неотрицательная переменная, если $i\in I^+$, x_i - нулевая переменная, если $i\in I^0$, x_i - свободная переменная, если $i\in I^{\pm}$, $I^+\cup I^0\cup I^{\pm} = \{i: -m\leq i\leq -1,\ 1\leq i\leq n\}$. Множество индексов текущего базиса обозначим $J=\{j_1,j_2,\ldots,j_m\}$. В общем случае $B=(b^{(1)},\ldots,b^{(rs)})$ является (mxrs)-матрицей правых частей.

Задача решается в два этапа. На первом этапе определяется допустимое базисное решение путем минимизации вспомогательной целевой функции: $\bar{c}^Tx=\bar{c}_{-m}x_{-m}+\ldots+\bar{c}_{-1}x_{-1}+\ldots+\bar{c}_nx_n$, где

$$\bar{c}_j = \begin{cases} -1, \text{ если } j\in J\cap I^+ \text{ и } \bar{x}_j<0, \\ 0 \text{ - в противном случае.} \end{cases}$$

На втором этапе определяется оптимальное базисное решение, причем в качестве начального базиса выбирается базис, для которого базисное решение допустимо. Рассмотрим алгоритм перехода от базиса J к базису J'.

1. Вычисляются симплексные множители $v^T=(v_1,\ldots,v_m)$ из решения системы линейных уравнений $v^TA_J=c_J^T$ и определяется вектор условных оценок $u=(u_{-m},\ldots,u_{-1},u_j,\ldots,u_n)^T$ как $u=c-A^Tv$.

2. Определяется индекс $k\in I^0$ вводимого в базис вектора $u_k=\min_i\{u_i: i\in I^0\}$, если $u_k\geq 0$, то вычисления заканчиваются, в противном случае перейти к следующему шагу.

3. Для определения индекса r исключаемого из базиса вектора определяется вектор y из решения системы линейных уравнений $A_Jy=A_k$ и вектор $\hat{b}=\bar{x}_J$ из системы $A_J\bar{x}_J=b$ ($\bar{x}_J$ - базисное решение). Затем определяются величины p и ma:

$$ma=\frac{y_p}{\hat{b}}=\max_{i\in\{1,m\}}\left\{\frac{y_i}{\hat{b}_i}: y_i>0,\ b_i>0,\ j_i\in I^+\cap J\right\},$$

если ma не определяется, то задача является неограниченной, в противном случае $r=j_p$, вернуться к п.1.

Таким образом, для введения в базис новой переменной нужно решить на каждой итерации три системы линейных уравнений с базисной матрицей A_J:

$$\left.\begin{aligned} &A_J\bar{x}_J^{(i)} = b^{(i)}, \quad i\in\{1,rs\},\\ &A_J y = A_k,\\ &A_J^T v = c_J\ . \end{aligned}\right\} \qquad (2)$$

Так как решение систем (2) на основе формирования обратной базисной матрицы A_J^{-1} с использованием формул (I) может оказаться численно неустойчивым, используют метод, основанный на приведении к треугольному виду текущего базиса A_J [7].

Текущий базис A_J приводят к треугольному виду согласно уравнению $LA_J = R$, $\bar{B} = LB$, где L - невырожденная (mxm)-матрица, R - невырожденная верхняя треугольная (m x n)- матрица, B - матрица (mxrs). В этом случае решение (2) с полной матрицей A_J сводится к решению систем линейных уравнений с треугольными матрицами R:

$$\left.\begin{aligned} &R\bar{x}_J^{(i)} = \bar{b}^{(i)}, \quad i\in\{1,rs\},\\ &R_y = LA_k\ ,\\ &R^T z = c_J, \quad v = L^T z\ . \end{aligned}\right\} \qquad (3)$$

Пусть для базиса J определена матрица $\{J,R,L,\bar{B}\}$, тогда при переходе к смежному базису J' матрица $\{J', R', L', B'\}$ строится следующим образом. Если x_r $(r = j_s \in J)$ выводится из базиса, а x_k $(k \notin J)$ вводится, то

$$J' = \{j_1,\dots,j_{s-1},j_{s+1},\dots,j_m,k\}\ ,$$

$$R' = \{R_1,\dots,R_{s-1},R_{s+1},\dots,R_m,x\}\ ,$$

где R_i - i-й столбец матрицы R, $x = LA_k$. Матрица R', например, для m = 5, s = 2 имеет вид

$$R' = \begin{matrix} \\ \\ \\ \\ \\ \end{matrix}\overset{\begin{matrix}(s-1)\ \ (s)\end{matrix}}{\begin{bmatrix} x & x & x & x & x\\ & x & x & x & x\\ & x & x & x & x\\ & & x & x & x\\ & & & x & x \end{bmatrix}}\begin{matrix}\\ (s-1)\\ (s)\\ \\ \\ \end{matrix}\ ,$$

Чтобы привести матрицу $E' = (R', L', \bar{B}')$ к треугольному виду, можно $(m-s)$ ненулевых поддиагональных элементов обратить в нуль либо по схеме исключения с перестановками строк, либо плоскими вращениями Гивенса, оба эти метода являются численно устойчивыми.

С учетом изложенного рассмотрим алгоритм перехода от базиса $J^{(i)}$ к базису $J^{(i+1)}$ на i-й итерации.

Шаг 0. Вначале устанавливают $J = \{-1, -2, \ldots, -n\}$, $R = I$; $L = I$; $\bar{b} = b$; $\bar{x}_J = \bar{b}$; $v = c_J$.

Шаг I. Вычисляют вектор $u = c - A^T v$, определяют $u_k = \min_{i \in \bar{J}} u_i$ и индекс $k \notin J$ вводимого в базис вектора.

Шаг 2. Вычисляют вектор $x = L^{(i)} A_k$, решают системы уравнений $R^{(i)} y = x$, $R^{(i)} x_J = \bar{b}$, определяют индекс выводимого вектора r:

$$\max = \frac{y_p}{\hat{b}_p} = \max \left\{ \frac{y_i}{\hat{b}_i} : y_i > 0,\ \hat{b}_i > 0,\ j_i \in \{ I^+ \cap J \} \right.,$$

отсюда $r = j_p$, здесь $\hat{b} = \bar{x}_J$.

Шаг 3. Формируют матрицу $\bar{R}^{(i+1)}$: из $\bar{R}^{(i)}$ исключается столбец $j_s = r$, все столбцы, следующие за ним, сдвигаются влево, справа присоединяется столбец x.

Шаг 4. Преобразуют матрицу $E^{(i)} = (\bar{R}^{(i+1)}, L^{(i)}, \bar{B}^{(i)})$ по правилам $\forall i = s, s+1, \ldots, m-1$:

а) если $|r_{i+1,i}| > |r_{i,i}|$, то i и (i+1) строки переставляются;

б) определяется $p = r_{i+1,i} / r_{i,i}$, $|p| \le 1$, после чего из (i+1)-й строки матрицы $E_i^{(i)}$ вычитается i-я строка, умноженная на p, в результате получают матрицу $E_{i+1}^{(i)}$. Применив $(m-s)$ таких преобразований, получают матрицу $E^{(i+1)}$.

Шаг 5. Определяют симплексные множители из решения системы линейных уравнений $R^T z = c_J$, $v = L^T z$. Возвратиться к шагу I.

Оценим операционную сложность реализации алгоритма в зависимости от параметров задачи: m - числа ограничений и n - числа переменных. Очевидно, что для реализации первого шага, на котором осуществляется умножение (mxm)-матрицы на вектор и поиск минимального элемента в строке, требуется не больше $P_1 = mn + n$ операций. На втором шаге умножение (mxm)-матрицы на вектор требует m^2 операций, решение двух систем линейных уравнений с треугольными матрицами требует порядка $m(m+1)$ операций (так как требуется

только обратный ход вычисления), определение индекса - не более m операций. Общее число операций определится тогда величиной $P_2 = 2(m^2+m)$. Третий шаг при формировании матрицы R требует $(m-s)$ сдвигов столбцов переменной длины и при исключении s-го столбца составляет величину $P_3 = m + \frac{m(m+1)}{2} - \frac{s(s+1)}{2}$ и в худшем случае при исключении первого столбца $P_3 = m + \frac{m(m+1)}{2} - 1 = \frac{m^2+3m-2}{2}$. На четвертом шаге нужно провести в худшем случае $(m-1)$ преобразований $(m \times (2m+1))$ - матрицы E по схеме исключения. Чтобы обратить в нуль $(m-1)$ поддиагональных элементов потребуется $P_4 = 2\left[\frac{(m+1)m}{2} - 2\right] + 2(m+1)(m-1) = 3m^2+m-4$ операций. Пятый шаг для решения системы линейных уравнений с треугольной матрицей требует проведения $\frac{m(m+1)}{2}$ операций, для умножения m-матрицы на вектор - m^2 операций и общее число операций определяется величиной $P_5 = \frac{3m^2+m}{2}$. Таким образом, операционная сложность одной итерации определяется величиной $P = \sum_{i=1}^{5} P_i = 7m^2 + mn + 5m + n - 5$. Так как при последовательном исполнении время реализации пропорционально числу операций, то $T_1 = 7m^2 + mn + O(m+n)$.

Рассмотрим параллельную интерпретацию изложенного метода на циклической системе вычислителей, показанной на рис.2. Для удобства осуществления преобразования матрицы E при исключении поддиагональных элементов с перестановками строк было произведено расщепление матриц R и L по столбцам, не оказывая существенного влияния на параллельную интерпретацию остальных шагов алгоритма. Декомпозицию матрицы (J, R, L, B) произведем, как показано на рис. 6,

	J	R^T				L^T				A			x_J	c_J	x	v	y
P_1	j_1	r_{11}	0	0	0	l_{11}	l_{21}	l_{31}	l_{41}	a_{11}	a_{12}	a_{13}	x_{j_1}	c_{j_1}	z_1	v_1	y_1
P_2	j_2	r_{12}	r_{22}	0	0	l_{12}	l_{22}	l_{32}	l_{42}	a_{21}	a_{22}	a_{23}	x_{j_2}	c_{j_2}	z_2	v_2	y_2
P_3	j_3	r_{13}	r_{23}	r_{33}	0	l_{13}	l_{23}	l_{33}	l_{43}	a_{31}	a_{32}	a_{33}	x_{j_3}	c_{j_3}	z_3	v_3	y_3
P_4	j_4	r_{14}	r_{24}	r_{34}	r_{44}	l_{14}	l_{24}	l_{34}	l_{44}	a_{41}	a_{42}	a_{43}	x_{j_4}	c_{j_4}	z_4	v_4	y_4
P_5		b_1	b_2	b_3	b_4	x_1	x_2	x_3	x_4	u_1	u_2	u_3	c_1	c_2	c_3		

Рис. 6

т.е. каждый вычислитель P_i хранит i-й элемент столбцов J, x_J, c_J, v, y, z и i-е столбцы матриц R и L, $i = (1,2,\ldots,m)$. Вычислитель P_{m+1} хранит соответствующие столбцы матрицы правых частей B, векторы x, c, u.

В таблице представлена во времени и пространстве (по вычислителям) параллельная реализация одной итерации алгоритма для $n = 3$, $m = 4$, $k = 1$, $r = s = 2$. Таблица иллюстрирует динамику локальных действий вычислителей при параллельном исполнении алгоритма.

Оценим время $T_m^{(1)}$ параллельной интерпретации i-го шага на системе из $(m+1)$-го вычислителя по таблице.

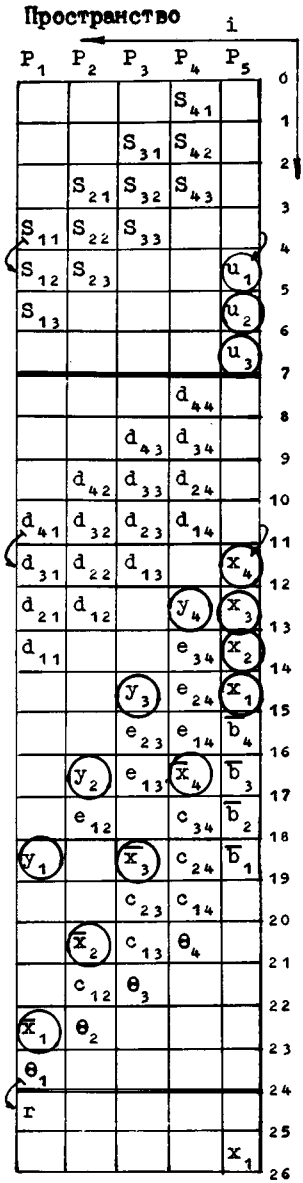
На первом шаге алгоритма каждый вычислитель P_i, начиная с P_m, определяет на j-м такте, $j \in \{1,n\}$, частичную сумму S_{ij} и посылает ее P_{i-1}, который, в свою очередь, доопределяет эту сумму, добавив $a_{i-1,j} v_{i-1}$, посылает ее P_{i-2} и т.д. вплоть до P_{m+1}. Через $T_m^{(1)} = n+m$ в P_{m+1} будет определен вектор u и найден индекс k.

На втором шаге из P_{m+1} транслируется по системе индекс k каждый раз соседнему вычислителю и одновременно каждый P_i на j-м такте, $j \in \{1,m\}$, определяет частичную сумму d_{ij} и посылает ее P_{i-1}. Через m тактов в P_{m+1} будет определен x_m и начинается обратный ход решения системы $Ry = x$. Каждый P_i на соответствующем такте определяет y_i, который затем посылает P_{i-1}. После определения всех x_i, $i \in \{1,m\}$, в P_{m+1} начнется обратный ход решения системы $R\bar{x}_J = \bar{b}$. После определения в каждом P_i элемента $\bar{x}_J$ вычисляется $\theta_i = y_i / \bar{x}_i$, затем находится индекс r, при котором $\theta_r = \max_i \theta_i$: каждое θ_i сравнивается с принятым θ_{i-1}, максимальное из них посылается P_{i-1}. Через m тактов в P_{m+1} будет определен индекс r. Таким образом, $T_m^{(2)} = m+m+m+m+1 = 4m+1$.

На третьем шаге одновременно с трансляцией индекса r вплоть до P_s сдвигаются $(m-s)$ столбцов: из P_{m+1} в P_m передается вектор x, из P_m в P_{m-1} столбец r_m и т.д., так как столбцы сокращающейся длины, то время формирования матрицы R определится временем передачи последних столбцов и определится величиной $T_m^{(3)} = m+2$.

На четвертом шаге активизируется P_s ($s=r=2$ в таблице), определяет $p_s = r_{s+1,s} / r_{s,s}$, посылает p_s $(s+1)$-у вычислителю, преобразует элементы s-й и $(s+1)$-й строк матрицы R, L, причем если $|r_{s+1,s}| > |r_{s,s}|$, то элементы s-й и $(s+1)$-й строк R, L переставляются во всех вычислителях на соответствующем такте. Элементы вектора $\bar{b}$ правой части преобразуются в P_{m+1}. На рис.7 показа-

Таблица



Шаг I

1. $u = c - A^T v$
2. $u_k = \min_j u_j$

$S_{ij} := rec_{i \oplus 1, j} + a_{ij} v_i$;

$sen_{m+1,n} = k$; $sen_{ij} = S_{ij}$;

$S_{m+1,j} = 0$

Шаг 2

1. $x = LA_k (k=1)$
2. $Ry = x$
3. $R\bar{x}_J = \bar{b}$
4. $\theta_r = \max_i y_i / \bar{x}_i$

$d_{ij} := rec_{j, i \oplus 1} + l_{ji} a_{jk}$;

$sen_{ij} = d_{ji}$; $d_{j,m+1} = 0$

$e_{ji} := rec_{j,i+1} - y_i r_{ji}$;

$sen_{ij} = e_{ji}$; $e_{j,m+1} = x_j$

$y_i = e_{ii} / r_{ii}$

$c_{ji} := rec_{j,i+1} - \bar{x}_i r_{ji}$;

$sen_{ji} = c_{ji}$;

$c_{j,m+1} = \bar{b}_i$

Шаг 3

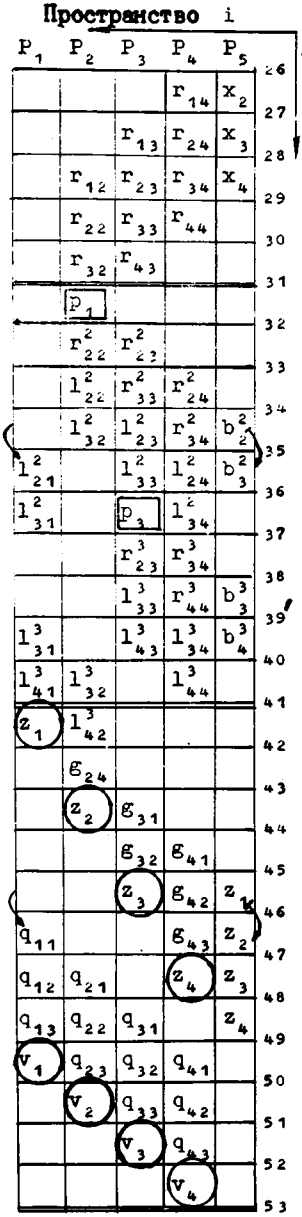
1. **Формирование** $R^{(i)} (r=2)$

$r_{ij}^{(i)} := rec_{i+1}$; $sen_{ij} := r_{ij}^{(i-1)}$;

$sen_{m+1,j} := x_j$;

$i = m+1, \ldots, m-s$; $j = 1, \ldots, i+1$

Продолжение таблицы



Шаг 4

1. **Преобразование** $E = (R, L, \bar{b})$

$p_i := r_{j+1,i} / r_{j,i}$; $sen_i = p_i$;

$r_{ii}^{(i)} := r_{ii} \cdot rec_{i-1}$; $l_{ii}^{(i)} = l_{ii} \cdot rec_{i \ominus 1}$;

$r_{i+1\ i}^{(i)} := r_{i+1,i} - r_{ii}^{(i)}$;

$l_{i+1,i}^{(i)} := l_{i \oplus 1,i} - l_{ii}^{(i)}$;

$b_i^{(i)} := b_i \cdot rec_{i-1}$; $b_{i+1}^{(i)} := b_{i+1}^{(i)} - b_i^{(i)}$

Шаг 5

1. $R^T z = c_J$
2. $v = L^T z$

$g_{ij} := g_{i,j-1} - r_{ji} \cdot rec_{i-1,j}$;

$sen_{ij} := z_j$; $g_{i0} = c_i$;

$z_i := g_{ii} / r_{ii}$;

$q_{ij} := q_{i,j-1} + l_{ji} \cdot rec_{i \ominus 1,j}$;

$q_{i0} = 0$; $sen_{ij} := z_j$;

$q_{im} := v_i$

○ - **результат операции**

ны (m-1) преобразований матрицы E = (R,L,b) в худшем случае при исключении первого столбца. В этом случае $T_m^{(4)} = m + 4(m-1) = 5m-4$. (Причем x^1, $+^1$, $*^1$ - соответственно элементы R,L,B,1 - номер преобразования.)

На пятом шаге P_1 , закончив (m-s) преобразований матрицы L, начинает выполнять обратный ход решения системы уравнений $R^T z = c_J$. Каждая компонента полученного вектора z транслируется из P_{m+1} в P_1 и далее, одновременно определяя произведение столбца матрицы L на вектор z . В результате вектор v будет определен через $T_m^{(5)} = 2m \cdot m + 1$. Общее время реализации одной итерации определяется как $T_m = \sum_{i=1}^{5} T_m^{(i)} = 14m + n + O(1)$.

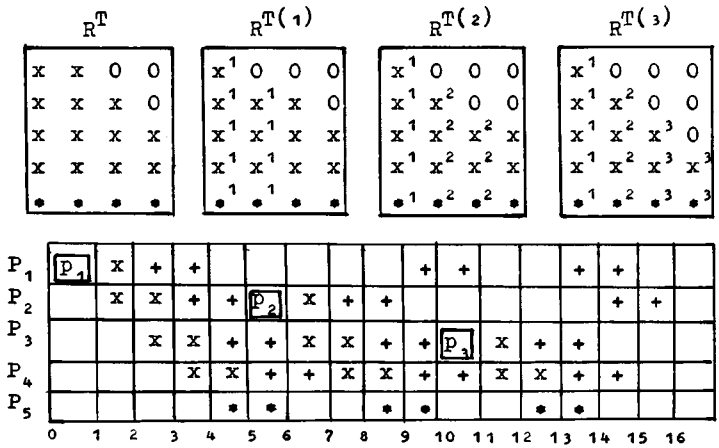


Рис. 7

Как уже упоминалось, матрицу E , полученную на третьем шаге, можно привести к треугольному виду плоскими вращениями Гивенса. Плоское вращение

$$u_{i,i+1} = \begin{bmatrix} c_i & s_i \\ -s_i & c_i \end{bmatrix}$$

преобразует две строки e_i и e_{i+1} в $\bar{e}_i$ и $\bar{e}_{i+1}$ так, чтобы исключить $e_{i+1,1}$. Так как

$$\bar{e}_i = c_i e_i + s_i e_{i+1}, \quad \bar{e}_{i+1} = -s_i e_i + c_i e_{i+1}, \qquad (4)$$

то $c_i = \frac{e_{i1}}{\mu}$, $s_i = \frac{e_{i+1,1}}{\mu}$, где $\mu = (e_{i1}^2 + e_{i+1,1}^2)^{1/2}$. Для исключения $(m-s)$ поддиагональных элементов необходимо провести преобразование $Q = u_{m-s+1,m} \cdot u_{m-s,m-s-1} \cdots u_{m-1,m}$. При параллельной реализации это преобразование применяется к столбцам матрицы E, так как расщепляется транспонированная матрица. Преобразование осуществляется по схеме, показанной на рис.7, с тем отличием, что вместо p_i по системе транслируется $\theta_{i,i+1} = \{c_i, s_i\}$, а элементы преобразуемых столбцов вычисляются по формулам (4). Так как вес операции исключения одного элемента плоского вращения строк больше веса операции исключения с перестановками строк, то временная сложность параллельной реализации четвертого шага по схеме Гивенса будет отличаться на мультипликативную константу.

Отметим, что операционная сложность алгоритма, которую можно определить по таблице последовательным подсчетом заполненных клеток, остается той же, что была определена при анализе алгоритма. Время параллельной реализации алгоритма уменьшается за счет совмещения операций, т.е. одновременного выполнения операций на $(m+1)$-м вычислителе в системе.

Степень совмещения операций или ускорение параллельной реализации алгоритма по сравнению с последовательной с учетом того, что число итераций одно и то же, определяется величиной $S_m = T_1/T_m = \frac{m(7m+n)}{14m+n}$. Если $n = cm$, то $S_m = m\,\frac{c+7}{c+14}$.

Эффективность использования вычислителей в системе определяется величиной $E_m = \frac{S_m}{m} = \frac{c+7}{c+14}$ и для разных значений c находится в пределах $\frac{1}{2} < E_m < 1$.

В заключение отметим, что симплекс-метод с использованием треугольного разложения базисной матрицы, хотя и более трудоемкий, при параллельной реализации является более эффективным. Анализ таблицы показывает, что ветви параллельного алгоритма, выполняемые в разных вычислителях, описываются последовательностью простых рекуррентных формул, т.е. простой программой обработки.

Очевидно, что параллельная реализация алгоритмов решения задач линейного программирования, которые характеризуются большим объемом обрабатываемой информации, позволяет уменьшить не только временную сложность алгоритма, но и уменьшить емкостную нагрузку

на каждый вычислитель за счет распределения данных по вычислителям и уменьшения или даже исключения обращения к внешней памяти.

Таким образом, если последовательное исполнение каждой итерации рассматриваемых алгоритмов требует времени $O(m^2+mn)$, где m - число ограничений, n - число переменных, то параллельная реализация алгоритмов на вычислительной системе из $O(m)$ циклически связанных вычислителей уменьшает время проведения одной итерации до величины $O(m+n)$.

Литература

1. ЕВРЕИНОВ Э.В., ХОРОШЕВСКИЙ В.Г. Однородные вычислительные системы. - Новосибирск: Наука, 1978. - 319 с.

2. ЕВРЕИНОВ Э.В., КОСАРЕВ Ю.Г. Однородные универсальные вычислительные системы высокой производительности. - Новосибирск: Наука, 1966. - 307 с.

3. ТАРНОРУЦКАЯ С.А. Решение задачи блочного программирования на сосредоточенной однородной вычислительной системе.-В кн.: Вопросы теории и построения вычислительных систем (Вычислительные системы, вып. 70). Новосибирск, 1977, с. 98-112.

4. ХУ Т. Целочисленное программирование и потоки в сетях.-М.: Мир, 1974. - 516 с.

5. ЮДИН Д.Б., ГОЛЬШТЕЙН Е.Г. Линейное программирование. -М.: ФМ, 1963. - 775 с.

6. RICHARD H.BARTELS, GENE H.GOLUB. The Simplex Method of Linear Programming Using LU Decomposition. - Communication of the ACM,1969,v.12,N 5,p.266-268.

7. УИЛКИНСОН, РАЙНШ. Справочник алгоритмов на языке АЛГОЛ. Линейная алгебра. -М.: Машиностроение, 1976. - 390 с.

8. SAMEH A.H.,KUCK D.J. On stable Parallel Linear System Solvers.- J.of the Association for Computing Machinery,1978,v.25, N 1,p.81-91.

9. HASSAN M.AHMED, JEAN-MARC DELOSME, MARTIN MORF. Highly Concurrent Computing Structures for Matrix Arithmetic and Signal Processing.- J.Computer,1982,Jan.,p.65-82.

Поступила в ред.-изд.отд.
30 марта 1983 года