ПАРАЛЛЕЛЬНЫЙ АЛГОРИТМ ЦЕЛОЧИСЛЕННОГО КВАДРАТИЧНОГО ПРОГРАММИРОВАНИЯ*

Г. И. Забиняко, Е. А. Котельников
Институт вычислительной математики и математической геофизики СО РАН, Новосибирск, Россия
e-mail: zabin@rav.sscc.ru

The parallel algorithm of integer and mixed-integer quadratic programming, based on the branch and bound method. The algorithm was realized in FORTRAN using the MPI system of parallel programming. The efficiency of the parallel and the sequential algorithms are compared for test problems.

Введение

Задача целочисленного и частично целочисленного квадратичного программирования (ЦКП) представляется в следующем виде:

$$\text{минимизировать } f(x) = \frac{1}{2}(x, Qx) + (c, x) \tag{1}$$

при условиях

$$Ax = b, \tag{2}$$

$$\alpha \le x \le \beta, \tag{3}$$

$$x_j - \text{целые числа для } j \in J. \tag{4}$$

Здесь A — матрица размера $m \times n$; Q — симметричная матрица размера $l \times l$ $(l \le n)$; $Q \ge 0$; $c, x, \alpha, \beta \in R^n$; $b \in R^m$; J — список целочисленных переменных.

Алгоритм решения задач ЦКП основывается на методе ветвей и границ с односторонним ветвлением, который представляет собой обобщение алгоритма целочисленного линейного программирования (ЦЛП). Решение исходной задачи сводится к решению последовательности оценочных задач квадратичного программирования.

Распараллеливание осуществляется асинхронным исполнением на каждом из процессоров алгоритма ветвей и границ с односторонним ветвлением. В результате получается

*Работа выполнена при финансовой поддержке Российского фонда фундаментальных исследований (грант № 01-07-90367).

некоторое сочетание алгоритмов с односторонним и одновременным ветвлением [1]. Использование MPI существенно упрощает программирование такого сочетания алгоритмов. Во время исполнения процессы обмениваются короткими сообщениями на фоне выполнения большого числа арифметических операций, что обеспечивает высокую эффективность распараллеливания как на вычислительных системах с общей памятью, так и на кластерах.

1. Решение оценочных задач

Оценочные задачи отличаются от исходной (1)–(4) тем, что в них отсутствует требование целочисленности (4) и в каждой i-й оценочной задаче условия (3) заменяются условиями $\alpha^i \le x \le \beta^i$. Векторы α^i и β^i формируются по правилам метода ветвей и границ, которые рассматриваются далее.

Для решения оценочных задач используется метод приведенного градиента [2], в котором переменные x подразделяются на базисные x_B, супербазисные x_S и небазисные x_N. В матрице A, соответственно, выделяются B-базисная $m \times m$, S-супербазисная $m \times s$ и N-небазисная $m \times (n-(m+s))$ матрицы. Предполагается, что B — невырожденная матрица, а переменные x_N принимают свои граничные значения.

В текущем подпространстве супербазисных переменных минимизация проводится методом сопряженных градиентов. На k-м шаге метода последовательно определяются:

1) вектор двойственных переменных $y^k = (B^{-1})^T \nabla f(x_B^k)$;

2) вектор приведенного градиента $h^k = \nabla f(x_S^k) - S^T y^k$;

3) вектор направления в подпространстве супербазисных переменных

$$p_S^k = -h^k + \frac{||h^k||_2^2}{||h^{k-1}||_2^2} p_S^{k-1};$$

4) вектор направления в подпространстве базисных переменных $p_B^k = -B^{-1} S p_S^k$;

5) оптимальный шаг вдоль p^k $\lambda_k = -\dfrac{(h^k, p_S^k)}{(p^k, Qp^k)}$, где вектор p^k составлен из p_B^k, p_S^k и $p_N^k = 0$. Полагается $x^{k+1} = x^k + \lambda p^k$, где $\lambda = \min\{\lambda_k,\ \lambda_{\max}\}$, а $\lambda_{\max} = \arg\max\limits_{\lambda}\{\lambda : \alpha^i \le x^k + \lambda p^k \le \beta^i\}$.

Если $\lambda_k \ge \lambda_{\max}$, то по методу приведенного градиента меняется состав супербазисных и других переменных, процесс по методу сопряженных градиентов начинается в новом подпространстве. В противном случае процесс по методу сопряженных градиентов продолжается до достижения с заданной точностью равенства нулю приведенного градиента.

Для уменьшения влияния ошибок вычислений на процесс по методу сопряженных градиентов осуществляется контроль точности решения систем уравнений $B^T y^k = \nabla f_B(x_B^k)$, $B p_B^k = S p_S^k$ и, если необходимо, их решения уточняются.

Против зацикливания метода сопряженных градиентов в плохо обусловленных задачах используется прием из [3], который состоит в проверке через определенное число итераций выполнения неравенства $||h^k||_2^2 \le c(k)||\bar{h}^k - h^k||_2^2$, где величина $c(k)$ возрастает с увеличением k, h^k — приведенный градиент с непосредственным вычислением $\nabla f(x_S^k)$, а в $\bar{h}^k$ используется рекуррентно вычисленный $\nabla f(x_S^k)$. При выполнении неравенства процесс по методу сопряженных градиентов в данном супербазисном подпространстве завершается.

После завершения оптимизации методом сопряженных градиентов в текущем супербазисном подпространстве на основе двойственных оценок небазисных столбцов проверяется

возможность сформировать новый набор супербазисных переменных. Если кандидатов на перевод из небазисных переменных в супербазисные нет, то оценочная задача решена. В противном случае используется метод сопряженных градиентов в новом супербазисном подпространстве.

2. Алгоритм целочисленного квадратичного программирования

Оценки границ целевой функции (1) с учетом (4) получаются из решений оценочных задач квадратичного программирования. Выбор переменной ветвления в ЦЛП часто осуществляют с использованием штрафов [1, 2]. В рассматриваемом алгоритме ЦКП также делается попытка, когда это возможно, для выбора переменной ветвления использовать штрафные оценки.

Вначале рассмотрим линейный случай. Значение базисной переменной x_j для $j \in J$ в оптимальном базисе очередной оценочной задачи представим в виде $x_j = [x_j] + v_j$, где $[x_j]$ — целая часть x_j. Штраф за увеличение x_j на величину $1 - v_j$ обозначим через P_j^+, а за уменьшение на величину v_j — через P_j^-.

На примере P_j^+ покажем алгоритм определения штрафных оценок. Номера небазисных переменных зададим в двух списках: в I_x^- поместим номера переменных, фиксированных на нижней границе, а в I_x^+ — на верхней границе. Если переменная x_j занимает в базисе p-ю позицию, то вычислим $z = e_p^T B^{-1}$, e_p — p-й орт в R^m. Тогда

$$P_j^+ = \min\left\{ \min_{q:q\in I_x^-,\ a_{pq}<0} \left\{(1-v_j)\Big(\frac{d_q}{-a_{pq}}\Big)\right\},\ \min_{q:q\in I_x^+,\ a_{pq}>0} \left\{(1-v_j)\Big(\frac{-d_q}{a_{pq}}\Big)\right\}\right\},$$

где $a_{pq} = (z, A_q)$, A_q — q-й небазисный столбец матрицы A; d_q — его приведенная оценка. Если q-я небазисная переменная целочисленная, то величины $(1 - v_j)(d_q/ - a_{pq})$ или $(1 - v_j)(-d_q/a_{pq})$, которые меньше $|d_q|$, заменяются на $|d_q|$. Начальные значения P_j^- и P_j^+ полагаются равными $+\infty$.

Штрафы для квадратичной функции f заменим штрафами для линейной функции $g(x) = (\nabla f(x), x - x^*)$, где x^* — решение очередной оценочной задачи квадратичного программирования. Поскольку исходная функция выпуклая, штрафные оценки для $g(x)$ будут не больше штрафов для $f(x)$. В нелинейном случае при вычислении штрафов необходимо дополнительно учесть супербазисные переменные (если они присутствуют в оптимальном решении оценочной задачи).

Величины d_q для супербазисных переменных равны нулю (d_q — q-я компонента приведенного градиента), и штрафы при рассмотрении очередной j-й переменной не обнуляются, если $\alpha_{pq} = 0$. Для любой j-й супербазисной переменной $P_j^+ = P_j^- = 0$.

Пусть $P_{\max} = \max\limits_j \max\{P_j^-, P_j^+\}$. В задачах ЦЛП $P_{\max} = 0$ получается в оценочных задачах, которые имеют не единственный оптимальный базис. В нелинейном случае возможность получить $P_{\max} = 0$ увеличивается за счет супербазисных переменных. Однако численные эксперименты показали, что использование штрафов в отдельных задачах ЦКП позволяет сократить время решения в несколько раз.

В случае $P_{\max} = 0$ для ветвления выбирается базисная или супербазисная переменная x_j, которая имеет значение, наиболее уклоняющееся от целочисленного. При этом только для определения переменной ветвления временно полагаются $P_j^+ = 1 - v_j$ и $P_j^- = v_j$.

Схемы метода ветвей и границ с односторонним ветвлением позволяют использовать компактную форму списков оценочных задач. Пусть на некотором уровне k для ветвления выбрана переменная x_j со штрафами P_j^- и P_j^+. Если $P_j^- \le P_j^+$, то в качестве очередной оценочной задачи выбирается задача, соответствующая P_j^-, а в списке оценочных задач (во вспомогательном массиве h) необходимо запомнить информацию об альтернативной задаче. Для этого производятся присвоения: $h(1,k) = j$, $h(2,k) = \beta_j^i$, где β_j^i — верхняя граница переменной x_j в i-й оценочной задаче на предшествующем уровне $(k-1)$; $h(3,k) = 1$, если $f^i + P_j^+ \ge r^i$ и $h(3,k) = 0$ в противном случае, здесь f^i — оптимальное значение f в i-й оценочной задаче, а r^i — текущее значение рекорда; $h(4,k) = f^i$; $h(5,k) = P_j^+$. При $P_j^+ < P_j^-$ выбирается оценочная задача, отвечающая штрафу P_j^+, а в массиве h запоминаются величины: $h(1,k) = -j$, $h(2,k) = \alpha_j^i$, $h(3,k)$, равная 0 либо 1 в зависимости от выполнения неравенства $f^i + P_j^- \ge r^i$, $h(4,k) = f^i$, $h(5,k) = P_j^-$.

В дальнейшем при рассмотрении параллельного алгоритма ветвь, которой соответствует $h(3,k) = 1$, будем называть помеченной и не помеченной при $h(3,k) = 0$.

Алгоритм.

Шаг 0. Положить $i = 0$, $k = 0$, значение рекорда $r^0 = +\infty$, $\alpha^0 = \alpha$ и $\beta^0 = \beta$.

Шаг 1. Решить текущую задачу квадратичного программирования. Пусть x^i — ее оптимальное решение и $f^i = f(x^i)$.

А. Если $f^i \ge r^i$ или система ограничений несовместна, то присвоить $r^{i+1} = r^i$, $i = i+1$ и перейти на шаг 3.

Б. Пусть $f^i < r^i$. Если вектор x^i нецелочисленный, то перейти на шаг 2. Если решение x^i целочисленное, то присвоить $r^{i+1} = f^i$, $i = i+1$. При $f^i = f^0$ перейти на шаг 5, иначе на шаг 3.

Шаг 2. Для тех базисных и супербазисных переменных x_j, $j \in J$, у которых x_j^i — нецелое, вычислить штрафы P_j^+, P_j^-. Среди них найти минимальный штраф $P_{\min}$:

А. Если $f^i + P_{\min} \ge r^i$, то перейти к шагу 3.

Б. При $f^i + P_{\min} < r^i$ осуществить ветвление по базисной или супербазисной переменной x_j, которой соответствует максимальный штраф P_j^+ или P_j^-. Из двух величин P_j^+, P_j^- выбрать меньшую. Пусть $P_j^- \le P_j^+$, тогда выбрать очередную оценочную задачу, соответствующую P_j^-. Присвоить $k = k+1$, записать в список задачу, отвечающую штрафу P_j^+. Изменить верхнюю границу переменной x_j, положив ее равной $[x_j^i]$. Перейти на шаг 1.

(Если $P_j^+ < P_j^-$, то в списке h запоминается задача, отвечающая P_j^-, а в формируемой задаче нижняя граница переменной x_j полагается равной $[x_j^i] + 1$.)

Шаг 3. Если $k = 0$, то перейти к шагу 5.

Если $k > 0$ и $h(3,k) = 1$ или $h(4,k) + h(5,k) \ge r^i$, то перейти к шагу 4.

Иначе, используя массив h, сформировать задачу, альтернативную образованной в шаге 2 Б. Присвоить $h(3,k) = 1$ и перейти на шаг 1.

Шаг 4. Установить двусторонние ограничения на переменную x_j, используя значения $h(1,k)$, $h(2,k)$. Присвоить $k = k-1$ и перейти на шаг 3.

Шаг 5. Остановиться.

Схемы ветвей и границ с односторонним ветвлением позволяют с небольшими затратами осуществить переход к очередной оценочной задаче. Рассмотренная реализация схемы во многом сходна с реализацией для ЦЛП в [4].

3. Параллельный алгоритм

Параллельный алгоритм устроен так, что на каждом из процессорных элементов исполняется рассмотренный ранее алгоритм с небольшими изменениями. Среди процессорных элементов выделяется элемент с нулевым номером, который кроме исполнения алгоритма ветвей и границ выполняет некоторые диспетчерские функции. По ходу решения на нулевом процессоре накапливается справочная информация, необходимая для организации параллельных вычислений.

На начальном этапе данные о задаче считываются с дисковой памяти нулевым процессором и пересылаются всем остальным. Далее на нулевом процессоре решается нулевая оценочная задача (исходная задача без учета условий целочисленности), а остальные процессоры находятся в ожидании.

Для загрузки любого процессора в его оперативную память пересылаются следующие данные: целочисленные массивы I_B, I_S, I_N — соответственно списки базисных, супербазисных и небазисных переменных; часть массива h (часть списка оценочных задач); x_S — массив значений супербазисных переменных. Значения элементов массивов I_B, I_S, I_N, x_S и какая часть массива h пересылается, определяется уровнем k, с которого начинается решение первой после загрузки оценочной задачи. На основе списка I_B строится матрица, обратная к базисной, и начинается исполнение алгоритма ветвей и границ с уровня k.

Для упорядочения загрузки процессорных элементов на нулевом процессоре хранится массив пар (i, k_i), где i — номер процессора, а k_i — уровень непомеченной ветви на i-м процессоре. В списке пар для каждого процессора i сохраняется k_i с наименьшим значением (что соответствует наиболее высокой непомеченной ветви на данном процессоре). Если некоторый i-й процессор простаивает, то это помечается присвоением $k_i = -1$.

Рассмотрим случай, когда i-й процессор завершил выполнение алгоритма (достигнут нулевой уровень) или процессор изначально еще не загружен. Если номер процессорного элемента $i > 0$, то нулевому процессору i-й процессор посылает запрос на загрузку. После получения запроса на нулевом процессоре выбирается пара (j, k_j) с минимальным значением $k_j > 0$ и на j-й процессор отправляется сообщение о необходимости загрузки i-го процессора. В результате на j-м процессоре ветвь уровня k_j помечается и отправляются данные для загрузки на i-й процессор. Если не существует $k_j > 0$, то i-й процесс находится в ожидании. При завершении исполнения алгоритма на нулевом процессоре аналогично определяется номер j-го процессора для загрузки.

Если на некотором процессоре получено новое значение рекорда r, то это значение пересылается всем остальным процессорам. При получении значения рекорда на каждом из процессоров проверяется выполнение неравенства $h(4, k) - h(5, k) \geq r$, где k — уровень, с которого начиналось исполнение алгоритма на данном процессоре. Если неравенство выполняется на каком-либо из процессоров, то на этом процессоре завершается исполнение алгоритма ветвей и границ и формируется запрос на загрузку.

Полученное значение r используется для обновления информации о помеченных и непомеченных ветвях в массиве h. Может оказаться, что на некотором из процессоров j помечается не помеченная ветвь с минимальным значением k_j. В этом случае на нулевой процессор посылается информация об изменении k_j.

После решения очередной оценочной задачи на каждом из процессоров производится обращение к функции MPI_IPROBE [5] для проверки, имеются ли сообщения для данного процесса. Кроме того, обращение к этой функции производится после выполнения определенного числа итераций решения оценочной задачи. Частота обращений к функции за-

дается в таблице управляющих параметров, где заданы и другие значения или начальные значения управляющих параметров, такие как частота перепостроения матриц, обратных к базисным, допустимый уровень ведущих элементов и др.

Задача решена, если на всех процессорных элементах достигнут нулевой уровень.

Передача исходных данных о задаче производится с помощью блокированной функции MPI "один — всем". В дальнейшем все процессы выполняются в асинхронном режиме и обмены осуществляются с помощью неблокированных функций.

4. Решение задач

Тестовые задачи взяты из [6]. В табл. 1 указаны входные параметры задач.

В табл. 2 приведены результаты расчетов в однопроцессорном режиме на системе МВС 1000М с процессорами альфа-21264, с тактовой частотой 830 МГц.

Таблица 1

Входные параметры задач

ь задачи	Название задачи	m	m_e	n	n_i	n_b	nz
1	*ibell*3*a*	106	–	122	60	31	304
2	*ibienst*1	576	128	505	28	28	2184
3	*ield*76	75	75	1898	1898	1898	19111
4	*imas*284	68	–	151	150	150	9631
5	*imisc*07	212	35	260	259	259	8619
6	*imod*011	4480	4367	10957	97	96	22253
7	*iqiu*	1192	132	840	48	48	3432
8	*iran*13	195	26	338	169	169	676
9	*iran*8	296	40	512	256	256	1024

Примечание. m — общее число ограничений, среди которых m_e ограничений являются равенствами; n — число переменных, среди которых n_i являются целочисленными и n_b переменных — булевыми; nz — число ненулей в матрице A.

Таблица 2

Результаты расчетов в однопроцессорном режиме на системе МВС 1000М

ь задачи	it	cp	t	P_1	P_2	P_0
1	354065	35765	66.76	2680	9648	18
2	32317905	71883	45315.	26	2500	30294
3	42003240	91423	51143.	150	4385	10577
4	1054772	53052	380.8	409	16672	127
5	3848924	54826	3057.	202	6797	2086
6	5500000	5411	83258.	15	193	–
7	32000000	150246	85163.	368	16556	4166
8	18496522	1324478	4480.	8847	537387	6604
9	16239520	985215	5751.	4385	400440	15115

Примечание. it — суммарное число итераций в оценочных задачах; cp — количество оценочных задач; t — время решения, P_1 указывает, сколько раз в процессе решения задачи выполнилось условие $f^i + P_{\min} \geq r^i$; P_2 — число случаев выполнения условия $f^i + P_j \geq r^i$, где $P_j = P_j^+$ или $P_j = P_j^-$; P_0 — указывает, сколько раз в процессе решения задачи было получено $P_{\max} = 0$.

В задачах 6 и 7 в однопроцессорном режиме не удалось получить точное решение из-за слишком больших затрат машинного времени. Задачи снимались со счета после выполнения заданного количества итераций. Обозначим через $\hat{f}$ полученное приближение по функции, через f^* — оптимальное значение и определим относительную ошибку $\delta f = \dfrac{\hat{f} - f^*}{|f^*|}$. Значения величины δf для задач 6 и 7 соответственно равны 0.119 и 0.002.

Характеристики процесса решения задач параллельным алгоритмом содержит табл. 3. В параллельном варианте расчетов использовались восемь процессоров. Во всех вариантах расчетов значения управляющих парметров выбирались одни и те же, кроме параметра, определяющего частоту обращения к функции MPI_IPROBE. Этот параметр для разных задач принимался в пределах от 10 до 30 в зависимости от числа строк в матрице ограничений.

Для задач 6 и 7, в которых в однопроцессорном режиме не удалось получить точное решение, были проведены дополнительно расчеты на 12 процессорах. В результате суммарное число итераций сократилось по сравнению с расчетами на восьми процессорах, для задачи 6 — на 4.1 млн итераций и для задачи 7 — на 9.6 млн итераций. Отношение t_8/t_{12}, где t_8 — время исполнения на восьми процессорах и t_{12} — на двенадцати, для задач 6 и 7 соответственно составило 1.9 и 1.7.

Анализ результатов численного эксперимента показывает, что параллельный алгоритм обеспечивает достаточно равномерную загрузку процессоров. С увеличением размерности, например, при числе строк, равном нескольким десяткам тысяч, можно ожидать возрастания простоев на начальном этапе у процессорных элементов с номерами $i > 0$. Здесь был бы полезен вариант MPI с динамическим определением числа используемых процессоров [7].

Параллельный вариант программы включен в пакет программ ЦКП. В него кроме программ решения задач входят программы обработки данных, которые осуществляют логический контроль данных, их перевод из входного MPS-формата [2] во внутренний разреженный формат и другие преобразования. Пакет организован аналогично пакету ЦЛП [4].

Таблица 3

Характеристики процесса решения задач параллельным алгоритмом

	Номер задачи								
	1	2	3	4	5	6	7	8	9
it_1	48243	3650064	7246301	123501	339065	2319167	15065957	2018111	1466794
it_2	47246	3770439	4379823	116647	324842	2500000	14285510	2005200	1490414
it_3	48071	3504662	4669789	121271	333464	2348354	14172405	2040903	1486702
it_4	48566	3947817	4523608	115781	315766	2361807	13030163	2039631	1423969
it_5	47621	3745081	3431145	113520	348888	2320268	14535137	1993103	1460954
it_6	46654	3707780	3473066	115535	309939	2326606	14784738	2019802	1449547
it_7	47678	3413254	3294001	115900	307952	2412781	13322902	2010772	1467747
it_8	45102	3532041	3281568	119760	302719	2317997	14299376	1963402	1529396
S_{it}	379181	29271138	34299301	941915	2582635	18906980	113496188	16090924	11775523
t_p	9.18	5187.	8851	45.49	293.4	40439.	38292	502.7	528.6
t/t_p	7.27	8.74	5.78	8.37	10.42	—	—	8.91	10.88

Примечание: it_i — суммарное число итераций в оценочных задачах, выполненных на i-м процессоре; S_{it} — суммарное число итераций, выполненное на всех процессорах; t_p — время решения задач параллельным алгоритмом, t/t_p — отношение времени решения задачи последовательным алгоритмом ко времени решения параллельным алгоритмом.

Заключение

Реализован параллельный алгоритм целочисленного квадратичного программирования, который позволяет снизить суммарную трудоемкость решения задач по сравнению с последовательным алгоритмом, и за счет этого обеспечивается ускорение выше линейного.

Список литературы

[1] Ковалев М.М. Дискретная оптимизация (целочисленное программирование). Минск: Изд-во Белорус. ун-та, 1977.

[2] Муртаф Б. Современное линейное программирование. Теория и практика. М.: Мир, 1984.

[3] Уилкинсон Дж.Х., Райнш К. Справочник алгоритмов на языке Алгол. Линейная алгебра. М.: Машиностроение, 1976.

[4] Zabinyako G.I., Kotel'nikov E.A. Linear optimization programs // NCC Bulletin, Series Num. Anal. Novosibirsk: NCC Publisher, 2002. Issue 11. P. 103–112.

[5] Корнеев В.Д. Параллельное программирование в MPI. Новосибирск: Изд-во СО РАН, 2000.

[6] ftp://plato.la.asu.edu

[7] Воеводин В.В., Воеводин Вл.В. Параллельные вычисления. СПб.: БХВ Санкт-Петербург, 2002.

Поступила в редакцию 1 октября 2003 г.