Math-Net.Ru

Общероссийский математический портал

Д. Л. Головашкин, О. Е. Горбунов, Параллельное решение СЛАУ методом Зейделя, *Вестн. Сам. гос. техн. ун-та. Сер. Физ.-мат. науки*, 2004, выпуск 27, 10–13

DOI: <https://doi.org/10.14498/vsgtu259>

Использование Общероссийского математического портала Math-Net.Ru подразумевает, что вы прочитали и согласны с пользовательским соглашением

<http://www.mathnet.ru/rus/agreement>

Параметры загрузки:

IP: 54.210.77.194

26 апреля 2023 г., 09:47:20

Численные методы

УДК 519.612.2

Д.Л. Головашкин, О.Е. Горбунов

ПАРАЛЛЕЛЬНОЕ РЕШЕНИЕ СЛАУ МЕТОДОМ ЗЕЙДЕЛЯ

Рассмотрены основные идеи построения параллельного алгоритма, позволяющего находить решение системы линейных алгебраических уравнений (СЛАУ) путем проведения итераций по рекуррентным формулам Зейделя. Предложен алгоритм параллельного вычисления, а также получена теоретическая оценка его ускорения. Приведенные результаты, полученные в рамках теории параллельных вычислений линейной алгебры, могут послужить основой при разработке соответствующих параллельных программ.

Постановка задачи. Задача нахождения решения СЛАУ

$$\mathbf{Ax} = \mathbf{b},$$

где $\mathbf{A} \in \mathbf{F}^{N \times N}$ – матрица СЛАУ; $\mathbf{b} \in \mathbf{F}^N$ – вектор правой части; $\text{rank}(\mathbf{A}) = \text{rank}(\mathbf{A}:\mathbf{b})$ ($\text{rank}(\mathbf{A})$ – ранг матрицы $\mathbf{A}$); $\mathbf{A}:\mathbf{b}$ – матрица, полученная из $\mathbf{A}$ путем приписывания к ней справа вектора $\mathbf{b}$; $\mathbf{F}$ – множество, содержащее элементы, над которыми определены операции сложения и умножения, относится к классу задач линейной алгебры и для ее решения разработан целый ряд численных методов. Среди них немаловажное место занимают итерационные методы. Наиболее известным представителем данной группы методов является метод простой однопараметрической итерации, анализ которого приведен в работе [1].

Другим, не менее важным методом, является метод Зейделя, для которого итерационный процесс имеет следующий вид

$$(x_i)_n = \sum_{j=1}^{i-1} b_{ij}(x_j)_n + \sum_{j=i}^N b_{ij}(x_j)_{n-1} + g_i, \quad (1)$$

где $\mathbf{x}_n = ((x_1)_n, \dots, (x_N)_n)^T \in \mathbf{F}^N$ – вектор приближения, вычисляемый на n -ом шаге итерации (вектор n -го приближения); $\mathbf{B} = \|b_{ij}\| \in \mathbf{F}^{N \times N}$ – матрица итерации; $\mathbf{g} = (g_1, \dots, g_N)^T \in \mathbf{F}^N$ – вектор итерации; $\mathbf{x}_0 = \tilde{\mathbf{x}} \in \mathbf{F}^N$ – вектор начального приближения; $i = \overline{1, N}$, $n = \overline{0, \infty}$.

Матрица и вектор итерации определяются соотношениями $\mathbf{B} = \mathbf{E} - \alpha \mathbf{A}$, $\mathbf{g} = \alpha \mathbf{b}$, где α – параметр итерации.

В качестве оценки близости вектора n -го приближения к вектору-решению может использоваться оценка относительной погрешности вычисления: $\varepsilon = \frac{\|\mathbf{x}_n - \mathbf{x}_{n-1}\|}{\|\mathbf{x}_n\|}$, $n = \overline{1, \infty}$.

Более полную информацию о рассматриваемом методе можно найти в работе [2]. Нашей же целью будет построение параллельного алгоритма данного метода, что продиктовано наличием параллельных вычислительных систем, способных давать выигрыш при вычислениях по сравнению с их последовательными аналогами.

Рекуррентные формулы Зейделя обладают той особенностью, что компоненты вектора приближений не могут вычисляться параллельно. Они определяются последовательно, друг за другом, и каждое новое значение выражается через найденные предыдущие. Несмотря на это, имеется возможность распараллеливания рассматриваемого итерационного процесса. Мы не будем модифицировать метод с целью повышения его степени параллелизма (данные подходы можно найти в работах [3, 4]), а рассмотрим непосредственно метод Зейделя, представленный формулами (1). При этом будем предполагать, что каждый шаг итерации алгоритма может быть выполнен параллельно. В процессе анализа рассматриваемой задачи будем использовать понятия, введенные в работах [5] и [6], где также рассмотрены основные принципы построения параллельных алгоритмов.

Параллелизм метода Зейделя. Характер задачи полностью определяется формулами (1), поэтому процедура построения метода параллельного вычисления должна быть основана на них.

Будем предполагать, что в нашем распоряжении имеются p ветвей алгоритма, способных проводить вычисления независимо друг от друга. Присвоим каждой ветви ее порядковый номер, причем нумерацию начнем с нуля. Будем также считать, что все ветви равноценны в вычислительном отношении. Это означает, что вычисления, проводимые над заданным набором данных выполняются всеми ветвями за одно и то же время (время вычислений не зависит от порядкового номера ветви). Порядок проведения вычислений каждой ветвью, а также процедуры взаимодействия, осуществляемые при этом, будем описывать с помощью параллельного алгоритма (p -алгоритма).

Заметим, что в (1) для фиксированного i ($i = \overline{1, N}$) все произведения могут выполняться параллельно. После суммирования результатов умножений получается i -ая компонента вектора нового приближения. Указанное параллельное вычисление будет иметь смысл только в случае, когда количество ветвей алгоритма меньше либо равно размерности задачи. При равенстве указанных параметров мы получаем случай, когда каждая ветвь производит одно умножение при вычислении очередной компоненты вектора. При этом было бы правильным каждый столбец матрицы $\mathbf{B}$, а также соответствующие компоненты векторов приближений, обрабатывать в отдельной ветви алгоритма. Если столбцы матрицы пронумеровать слева направо, начиная с нуля, то k -ый ($k = \overline{1, N}$) столбец используется в ветви с номером k . Компоненты вектора итерации также распределим по ветвям так, что компонента с номером k ($k = \overline{1, N}$) достается $(k-1)$ -ой ветви. На рис. 1 в прямоугольники заключены данные, получаемые k -ой ветвью в результате указанной декомпозиции.

Параллельное вычисление вектора n -го приближения состоит в последовательном нахождении его компонент по формулам (1), причем операции умножения, необходимые для этого, выполняются ветвями параллельно. Результаты произведений должны быть просуммированы, что осуществляется путем взаимодействия ветвей алгоритма. Максимальная эффективность при суммировании достигается в случае, когда ветви организуют двоичное дерево коммуникаций. На рис. 2, а приведено дерево для семи ветвей алгоритма. Каждому узлу дерева соответствует своя ветвь алгоритма. Обозначим за $\sigma(i)$ слагаемое, содержащееся в i -ом узле дерева. Стрелками показано направление передачи результатов суммирования (частичных сумм). Процедура получения суммы $\sum_{i=0}^{p-1} \sigma(i)$ называется сдваиванием относительно операции

суммы. Она производится снизу-вверх путем последовательного выполнения элементарных сдваиваний, осуществляемых тройкой узлов. Указанная тройка представляет собой поддерево исходного дерева (см. рис. 2, б) и является 3-х узловым шаблоном взаимодействия $\Psi = (b_i, b_j, b_k)$. Получение суммы осуществляется в следующей последовательности.

$$\begin{bmatrix} x_1 & x_2 & \cdots & \boxed{x_k} & \cdots & x_N \end{bmatrix}_{n-1}^T \begin{bmatrix} b_{1,1} & b_{1,2} & \vdots & \boxed{b_{1,k}} & \vdots & b_{1,N} \\ b_{2,1} & b_{2,2} & \vdots & \boxed{b_{2,k}} & \vdots & b_{2,N} \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ b_{k,1} & b_{k,2} & \vdots & \boxed{b_{k,k}} & \vdots & b_{k,N} \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ b_{N,1} & b_{N,2} & \vdots & \boxed{b_{N,k}} & \vdots & b_{N,N} \end{bmatrix} \begin{bmatrix} g_1 \\ g_2 \\ \vdots \\ \boxed{g_k} \\ \vdots \\ g_N \end{bmatrix} = \begin{bmatrix} x_1 & x_2 & \cdots & \boxed{x_k} & \cdots & x_N \end{bmatrix}_n^T$$

Р и с. 1. Декомпозиция данных для случая, когда $N = p$

Р и с. 2. Деревья коммуникаций. Дерево коммуникаций ветвей алгоритма (а), 3-х узловое поддерево сдваивания (б)

Узел b_p получает от узла b_{c1} величину $\sigma(b_{c1})$ и увеличивает $\sigma(b_p)$ на принятую величину $\sigma(b_p) = \sigma(b_p) + \sigma(b_{c1})$. Затем выполняется аналогичное взаимодействие с узлом b_{c2} и

выполняется пересчет значения $\sigma(b_p)$: $\sigma(b_p) = \sigma(b_p) + \sigma(b_{c_2})$. В итоге узел b_p содержит сумму величин, хранящихся в рассмотренном поддереве. При этом будем говорить, что к узлу b_p применена операция сдваивания на шаблоне $\Psi = (b_p, b_{c_1}, b_{c_2})$. Полученная в результате сумма затем выступает в качестве элементарного слагаемого в поддереве, для которого b_p является дочерним узлом, и т. д. Процедура получения суммы всех элементов дерева завершается после применения операции сдваивания к корневому узлу. Несложно заметить, что при суммировании существуют сдваивания, которые можно выполнять параллельно. Соответствующие тройки узлов лежат на одном уровне дерева, а количество этих уровней равно: $n_p = \lceil \log_2(p) \rceil$, где $\lceil a \rceil$ – целая часть числа a .

Возвращаясь к формулам (1), заметим, что i -ая компонента вектора нового приближения, получаемая при сдваивании, предназначена для дальнейшего использования в ветви с номером i . Поэтому целесообразно именно эту ветвь соотнести с корневым узлом дерева сдваивания. Указанное предложение может быть реализовано с помощью динамического дерева, в котором корню по очереди соответствуют все ветви алгоритма.

Опишем порядок выполнения параллельных вычислений для случая, когда $N = p$. Пусть $(x_i)_{пред}$ – предыдущее значение компоненты вектора приближения, используемое i -ой ветвью алгоритма, $(x_i)_{тек}$ – значение, используемое ветвью при вычислениях по формулам (1).

Параллельный алгоритм метода Зейделя (p -алгоритм Зейделя; случай, когда $N = p$).

Последовательность вычислений, проводимая i -той ($i = \overline{0, p-1}$) ветвью алгоритма, представляет собой периодическое выполнение следующих действий.

1. Участие в параллельном нахождении j -ой ($j = \overline{1, N}$) компоненты нового вектора. При этом возможны два случая: а) $j \neq i$; здесь ветвь вычисляет $\sigma_j(i) = b_{ji}(x_i)_{тек}$, а затем использует найденную величину в процедуре сдваивания на дереве коммуникаций; б) $j = i$, т.е. ветвь участвует в вычислении соответствующей ей компоненты вектора нового приближения; для этого находится величина $\sigma_i(i) = b_{ii}(x_i)_{тек} + g_i$ и далее выполняется сдваивание величин $\sigma_i(k)$ ($k = \overline{0, p-1}$), и пересчет $(x_i)_{пред}$ и $(x_i)_{тек}$: $(x_i)_{пред} = (x_i)_{тек}$; $(x_i)_{тек} = \sum_{k=0}^{p-1} \sigma_i(k)$ (результат сдваивания).

2. Проверка условия завершения процесса вычисления после вычисления всех компонент нового вектора. Взаимодействие при проверке также осуществляется на дереве коммуникаций. Норма вектора, необходимая для оценки близости найденного вектора к точному решению может быть вычислена различными способами, каждый из которых определяет свой порядок взаимодействия ветвей. В любом случае, в качестве исходных данных в узлах дерева выступают величины $\mu(i) = (x_i)_{тек} - (x_i)_{пред}$ и $\eta(i) = (x_i)_{тек}$. В случае невыполнения условия окончания работы $\varepsilon \geq \varepsilon_0$, где ε_0 – заданная относительная погрешность вычисления принимается решение о продолжении вычислений, начиная с п. 1.

Отметим, что результат проверки должен быть получен во всех ветвях алгоритма. Это означает, что после сдваивания необходимо распределить флаг окончания работы по всем ветвям алгоритма. Указанное распределение может быть выполнено в направлении, обратном направлению сдваивания, т. е. сверху-вниз. При этом осуществляется взаимодействие узлов в противоположном направлении (передача информации от узла-предка к узлу-потомку). Так же как и для сдваивания, для распределения можно выделить тройки узлов, взаимодействующих параллельно. Они лежат на одном уровне дерева коммуникаций, а количество этих уровней равно n_p .

Имея в распоряжении параллельный алгоритм, хотелось бы выяснить целесообразность его использования. Для этого нам необходимо оценить время, которое требуется для проведения вычислений по последовательному и параллельному алгоритмам.

Пусть T_1 – время выполнения последовательного алгоритма, T_p – время выполнения p -алгоритма. Базовые операции, используемые в (1), а также проверка условия окончания вычислений требуют определенных временных затрат. Будем считать, что 1) время, необходимое для выполнения операции сложения равно $t_{с.лж}$; 2) время, затрачиваемое на

выполнение операции умножения равно $t_{умн}$; 3) время сравнения двух операндов – $t_{срв}$; 4) количество итераций последовательного и параллельного алгоритмов равно k ; 5) время передачи d элементов от одной ветви другой – $t_{ком}(d)$; 6) при проверке условия окончания работы вычисляется кубическая норма $\|x\|_3 = \max_{1 \leq i \leq N} |x_i|$.

Можно показать, что тогда время вычислений по последовательному и параллельному алгоритмам будет оцениваться соответственно следующими выражениями:

$$T_1 = k \{ (N^2 + 1)t_{умн} + N(N + 1)t_{слож} + (2N + 1)t_{срв} \},$$

$$T_p = k \{ (N + 1)t_{умн} + (N + 2n_p + 1)t_{слож} + 2n_p((N + 1)t_{ком}(1) + t_{ком}(2)) + 2(2n_p + 1)t_{срв} \}.$$

Оценка ускорения представленного алгоритма имеет вид $S_p = T_1 / T_p$. На рис. 3 приведен график зависимости ускорения алгоритма от количества параллельных ветвей. Он построен в предположении, что $t_{ком}(1) \sim 5 \cdot 10^2 t_{умн}$, $t_{ком}(d) \sim d \cdot t_{ком}(1)$ и $t_{умн} \approx t_{слож} \approx t_{срв}$. Как видно из рисунка ускорение растет с ростом размерности задачи. Однако оно растет очень медленно, что объясняется наличием частых и продолжительных взаимодействий между ветвями алгоритма. Несмотря на это, существует возможность ускорить процесс вычисления и получить выигрыш во времени.

В настоящей работе рассмотрен общий подход к распараллеливанию итерационного метода Зейделя, предназначенного для решения СЛАУ. Построен алгоритм для случая, когда количество ветвей алгоритма равно размерности задачи. Полученные результаты следует рассматривать как отправную точку при построении алгоритмов для более общего случая. При веденная оценка ускорения не исключает возможности получения выигрыша

Р и с. 3. График зависимости ускорения p -алгоритма Зейделя от количества параллельных ветвей

при параллельных вычислениях, что говорит о целесообразности использования параллельных вычислительных систем для решения СЛАУ большой размерности.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Головашкин Д.Л. Методы параллельных вычислений (Часть 2)/ Учеб. Пособие. Самара: СГАУ. 2002. 103 с.
2. Бахвалов Н.С. Численные методы. М.: Наука, 1975. 632 с.
3. Корнеев В.Д. Параллельное программирование в MPI. Новосибирск: Изд-во СО РАН, 2000. 213 с.
4. Шпаковский Г.И., Серикова Н.В. Программирование для многопроцессорных систем в стандарте MPI / Пособие. Минск: БГУ, 2002. 323 с.
5. Миренков Н.Н. Структурное параллельное программирование // Программирование. 1975. № 3. С. 3–15.
6. Ортега Дж. Введение в параллельные и векторные методы решения линейных систем: Пер. с англ. М.: Мир, 1991. 367 с.

Работа выполнена при поддержке Министерства образования РФ, Администрации Самарской области и Американского фонда гражданских исследований и развития (CRDF) в рамках Российско-американской программы “Фундаментальные исследования и высшее образование” (BRHE) и при частичной поддержке Фонда содействия отечественной науке.

Поступила 3.03.2004 г.