Фрагментированный алгоритм LU-разложения и его параллельная реализация

К. Г. Чаюк
Новосибирский государственный технический университет,
Новосибирск

Фрагментированный (сборочный) подход к параллельному программированию был предложен в [1], рассматривался в [2] и применялся для распараллеливания метода частиц-в-ячейках (PIC) в [3]. В [4] описана runtime-система для исполнения фрагментированной программы.

В данной работе необходимо было разработать и исследовать фрагментированный алгоритм LU-разложения и программно реализовать его, изучить качество исполнения фрагментированной программы, сравнить его с качеством исполнения последовательной и параллельной программ LU-разложения. Результаты работы предполагается использовать для создания стандартной параллельной подпрограммы LU-разложения.

Рассмотрим LU-разложение. Пусть дана система линейных алгебраических уравнений:

$$Ax = F,$$
$$A = LU,$$
$$LUx = F,$$
$$n^2.$$

Рассмотрим решение этой системы методом, основанном на LU-разложении.

Предположим, что нам удалось разложить матрицу

$$A = LU.$$

Подставляя в СЛАУ, получаем:

$$LUx = F$$

Обозначим $Ux = y$.

Тогда получим $Ly = F$.

Рассмотрим алгоритм получения LU-разложения. Матрицы L и U будем искать в следующем виде:

$$L = \begin{pmatrix} l_{11} & 0 & 0 & \dots \\ l_{21} & l_{22} & 0 & \\ l_{31} & l_{32} & l_{33} & \\ \dots & & & \end{pmatrix} \qquad U = \begin{pmatrix} 1 & u_{12} & u_{14} & \dots \\ 0 & 1 & u_{23} & \\ 0 & & 1 & \\ \dots & & & \end{pmatrix}$$

Учитывая равенство $A = LU$. и умножая последовательно строки матрицы L на столбцы матрицы U, получаем систему, состоящую из $n2$ уравнений с $n2$ неизвестными (в результате чего LU-разложение единственно) $l_{ij}, i = \overline{1,n}, j = \overline{1,i}$ и $u_{ij}, i = \overline{1,n}, j = \overline{i+1,n}$ (n – размерность СЛАУ):

$$a_{11} = l_{11}; a_{12} = l_{11}u_{12}; a_{13} = l_{11}u_{13}; \dots$$
$$a_{21} = l_{21}; a_{22} = l_{21}u_{12}; a_{12} = l_{11}u_{12}; a_{23} = l_{21}u_{13} + l_{22}u_{23}; \dots$$
$$a_{31} = l_{31}; a_{23} = l_{31}u_{12} + l_{32}; a_{33} = l_{31}u_{13} + l_{32}u_{23} + l_{33}; \dots$$
$$\dots$$

Решая систему можно получить общие формулы для нахождения элементов матриц L и U:

$$l_{ij} = a_{ij} - \sum_{k=1}^{j-1} l_{ik}u_{kj}, \; u_{ij} = \frac{1}{l_{ii}}\left[a_{ij} - \sum_{k=1}^{j-1} l_{ik}u_{kj} \right] \qquad (1)$$

По этим формулам производились вычисления элементов матриц L и U в фрагментах.

Идея фрагментированного алгоритма заключается в обработке в нужном порядке и параллельно (где это возможно) одним и тем же последовательным программным кодом различных частей матрицы (всех подматриц). Рассмотрим фрагментированный алгоритм.

Квадратами на рис. 1 обозначены фрагменты исходной матрицы, обрабатываемые как отдельный процесс. Каждый процесс производит LU-разложение в своем фрагменте. Цифрами и стрелками показан порядок обработки фрагментов.

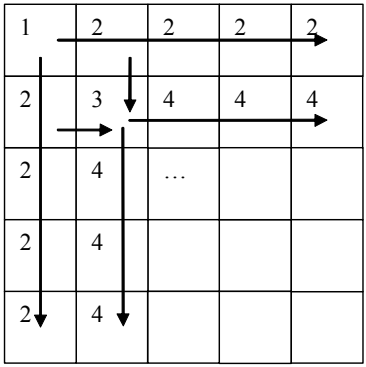


Рис. 1

Управление вычислениями основано на зависимости фрагментов по данным. Если посмотреть на формулы (1), по которым рассчитывались элементы матрицы, то видно, что для вычисления одного элемента необходима сумма $\sum_{k=1}^{j-1} l_{ik} u_{kj}$, т. е. нужно знать результаты разложения предыдущих фрагментов до диагонали. Поэтому сразу может вычисляться только фрагмент с номером один. После того, как он сделал LU-разложение, могут начать вычисления все фрагменты с номером два, и т. д. И, исходя из этой зависимости, выбирается последовательность запуска фрагментов на исполнение. Для исполнения каждого фрагмента необходимо чтобы все соседи слева и сверху (до диагональных включительно) уже завершили вычисления.

Алгоритм предназначен для исполнения на многопроцессорных машинах с общей памятью. В этом случае отсутствует необходимость пересылки и фрагментации данных.

Все расчеты проводились на smp4x64.sscc.ru. Это четырехпроцессорный сервер с общей памятью HP Integrity rx4640. Он включает 4 процессора Intel Itanium2 (1,5 ГГц) 4 Мбайт кэш, 64 Гбайт оперативной памяти при пропускной способности шины 12,8 Гб/с, 2 жестких диска 146 Гбайт.

Результаты численных экспериментов

Размер матрицы	2500x2500		10000x10000			22500x22500		
Время счета последовательной программы (с)	0.210367		3.535771			17.840454		
Размер фрагмента	1250×1250	625×625	5000×5000	2500×2500	2000×2000	11250×11250	7500×7500	5625×5625
Количество фрагментов	4	16	4	16	25	4	9	16
Время счета фрагментированной программы (с)	0.002578	0.013047	0.037955	0.355869	0.591370	0.171227	0.503275	1.070491

По таблице видно, что на четырех процессорах мы получаем ускорение в 100 раз. При увеличении количества фрагментов ускорение падает, это связано с увеличением затрат на управление,

но и в этом случае, получено ускорение в 10 раз, и с дальнейшим ростом количества фрагментов существенно ниже оно не опускается.

В случае распределенной памяти (мультикомпьютеров), появляется необходимость в пересылке вектора накопленных сумм, что замедляет работу программы. Но затраты на пересылку сопоставимы с тем, что в случае общей памяти, каждый фрагмент сам считает необходимые суммы. При этом появляется более жесткое управление запуском фрагментов. Время счета текущей реализации программы в случае распределенной памяти оказалась в среднем в полтора раза хуже, чем в случае общей.

Подобный выигрыш в системах с общей памятью получается в задачах, использующих при решении явные разностные схемы. Проводя анализ возможных задач, можно предположить, что задачи, в которых допустимо разбиение расчетной области на подобласти или задачи, в которых параллелизм есть на уровне алгоритма (слабые зависимости между действиями или ветвями алгоритма), должны хорошо фрагментироваться. Например, явные разностные алгоритмы, нейросетевые модели и т. д.

В фрагментированной версии размер фрагмента задается пользователем до начала вычислений и, исходя из этого, получается общее количество фрагментов, которых может быть много на узле. Наличие многих фрагментов на узле позволяет скрыть обмены на фоне вычислений.

Динамические свойства фрагментированной программы основаны на том, что на каждом узле исполняется много фрагментов, и они могут единообразным управлением перемещаться на другие узлы, тем самым, обеспечивая равномерную загрузку вычислительных ресурсов.

Таким образом, фрагментированный подход к параллельному программированию численных алгоритмов позволяет, используя последовательные программы технологично конструировать параллельные.

Фрагментированное программирование позволяет автоматически обеспечивать различные динамические качества параллельной прикладной программы, что очень важно при решении задач большого размера на мультикомпьютерах, в том числе и на кластерах и т. д. При этом оно позволяет получить выигрыш во времени исполнения. Главным преимуществом фрагменти-

рованного подхода является высокая способность программы настраиваться на все доступные ресурсы вычислителя (конфигурации сети и связей между узлами мультикомпьютера, количество каждого ресурса и т. д.).

В дальнейшем планируется развивать реализацию с распределенной памятью, так как она лучше использует явные преимущества фрагментированного подхода, а также реализовать другие численные алгоритмы и на их базе накопить библиотеку фрагментированных программ.

Литература

1. **Малышкин В. Э.** Синтез параллельных программ и систем на основе вычислительных моделей. – Новосибирск : Наука, 1988. — 128 с.

2. **Малышкин В. Э.** Введение в параллельное программирование мультикомпьютеров. — Новосибирск, 2003.

3. **Kraeva M. A., Malyshkin V. E.** Assembly Technology for Parallel Realization of Numerical Models on MIMD-Multicomputers // International Journal on Future Generation Computer Systems. — 2001. — Vol. 17. — № 6. — P. 755—765.

4. **Kalgin K. V., Malyshkin V. E., Nechaev S. P., Tschukin G.** Runtime System for Parallel Execution of Fragmented Subroutines // Springer Verlag: LNCS, — Vol. 4671. — P. 546-554.