УДК 519.612 **DOI: 10.14529/mmp210309**

РЕШАТЕЛИ СЛАУ С БЛОЧНО-ЛЕНТОЧНЫМИ МАТРИЦАМИ

***Б.Я. Штейнберг*[1], *А.А. Василенко*[1], *В.В. Веселовский*[1], *Н.А. Живых*[1]**
[1]Южный федеральный университет, г. Ростов-на-Дону, Российская Федерация

В статье предлагаются методы построения быстрых решателей для систем линейных алгебраических уравнений с блочно-ленточными матрицами. Предлагается структура данных для эффективного хранения таких матриц в оперативной памяти и быстрый алгоритм решения систем линейных уравнений с такой матрицей, основанный на этой структуре. Статья ориентирована на создание решателей, основанных на итерационных алгоритмах решения систем линейных уравнений как с симметрическими матрицами, так и с матрицами, имеющими седловую особенность. Предлагается разрабатывать и использовать специальный предкомпилятор для ускорения решателя. В данной работе экспериментальный решатель реализован на языке Си, предварительная компиляция выполнена на основе имеющейся у авторов Оптимизирующей распараллеливающей системы. Приводятся результаты численных экспериментов, демонстрирующие высокую эффективность разработанных методов, в том числе, и эффективность предкомпилятора.

Ключевые слова: параллельные вычисления; кэш-промахи; системы линейных алгебраических уравнений.

Введение

Во многих задачах математического моделирования возникает необходимость решения систем линейных алгебраических уравнений (СЛАУ) большой размерности с разреженными матрицами. В данной работе описано создание быстрого решателя для СЛАУ с блочно-ленточными матрицами. Разработаны структуры для эффективного хранения данных и быстрые алгоритмы. Проведены численные эксперименты, демонстрирующие высокую производительность представленных методов.

СЛАУ с блочно-ленточной матрицей при численном решении задач математического моделирования возникает непосредственно при использовании сеточных методов или метода конечных элементов в случае прямоугольных областей и ортогональной сетки. Но во многих случаях сложных областей можно построить криволинейную, но ортогональную сетку [1] и прийти от СЛАУ с разреженной матрицей общего вида к СЛАУ с блочно-ленточной матрицей.

В статье рассматриваются итерационные решатели для СЛАУ с симметричными матрицами, обладающими седловой особенностью [2]. Поскольку решатель пакета прикладных программ ориентирован на некоторое семейство близких задач, этот решатель предполагается параметризованным. Параметры решателя – это и характеристики начальных данных задачи (например, размерность блоков в получаемой СЛАУ).

Для получения быстрого кода решателя предлагается разрабатывать и использовать специальный предкомпилятор. Предкомпилятор – это конвертор, преобразующий исходный высокоуровневый код в другой высокоуровневый код, который более эффективно отобразится на архитектуру вычислительной системы. Разработанный на основе ОРС (Оптимизирующей распараллеливающей системы [3]) предкомпилятор ускоряет код решателя более чем на 30% для тестового набора данных.

1. Итерационные алгоритмы для решения целевых СЛАУ

Для полноты изложения приведем необходимую информацию об итерационных алгоритмах [4]. Поскольку предполагается рассматривать решатели для СЛАУ большой размерности, чтобы обеспечить приемлемую точность и не выходить за пределы имеющейся памяти, необходимо использовать итерационные алгоритмы. Как обычно, при решении задач большой размерности необходимо сбалансировать соблюдение точности (не должна накапливаться погрешность), ограничения по памяти и время работы вычислительной системы. Время работы итерационного процесса зависит от количества итераций и времени выполнения одной итерации.

Систему уравнений $Ax = b_0$ будем решать итерационным алгоритмом

$$x^{(k+1)} = Bx^{(k)} + b, \tag{1}$$

$$B = I - t\,C^{-1}A,\ b = t\,C^{-1}b_0. \tag{2}$$

Здесь $0, 1, 2, ..., k$ – номер итерации, A – квадратная матрица размера $n \times n$, b – вектор размера n, $x^{(0)}$ – начальное приближение, которое задает пользователь, $x^{(k)}$ – результат выполнения k-ой итерации, C – произвольная невырожденная матрица, t – положительный числовой параметр. Для сходимости итерационного процесса спектральный радиус матрицы B должен быть меньше единицы. Основная отличительная особенность процессоров последних лет состоит в том, что время выполнения арифметических операций более чем на порядок дольше времени считывания аргументов таких операций из оперативной памяти [5].

Можно сформулировать условия для выбора параметров итерационного процесса C и $t > 0$: 1) вычисление $C^{-1}x$ должно требовать мало времени; 2) объем памяти для матриц B и C должен быть равен или ненамного больше объема памяти для хранения матрицы A; 3) спектральная норма матрицы B должна быть меньше 1.

Матрица B обратима для всех малых чисел t таких, что число $t - 1$ больше наибольшего собственного значения матрицы $C^{-1}A$. В качестве t можно взять число $t = \dfrac{1}{||C^{-1}|| \cdot ||A||} - eps$ (здесь eps – некоторое заданное положительное число), поскольку спектральная норма матрицы $C^{-1}A$ не больше любой нормы этой матрицы, которая не больше $||C^{-1}|| \cdot ||A||$. Используется тот факт, что спектр полинома от линейного оператора равен полиному от его спектра (см. [3]) применительно к оператору $I - t\,C^{-1}A$, т.е. $Sp\{I - t\,C^{-1}A\} = (1 - t\,Sp\{C^{-1}A\})$. Выбор параметра t позволит соблюсти требование 3). От этого параметра зависит количество шагов итерационного процесса.

В качестве матрицы C будем рассматривать блочную ленточную матрицу с малым количеством ненулевых диагоналей $2 \cdot k + 1$ близких к главной. Эта матрица получается из матрицы A обнулением всех элементов A_{ij}, для которых $|i - j| > k$. Матрицу A представим в виде $A = C + O$, где O – матрица, состоящая из оставшихся элементов матрицы A после обнуления (вычитания) элементов по главной диагонали и k соседних (с двух сторон) диагоналей. Сделав LU-разложение матрицы C (или $C = LDL^*$ для симметричной матрицы), можно быстро вычислять $C^{-1}x$ на каждой итерации алгоритма. Во многих приложениях матрица A является симметрической или положительно определенной. В этих случаях матрица C тоже может быть симметрической или положительно определенной и, как следствие, матрица O – симметрическая или положительно определенная соответственно. Может быть, что A – не

симметрическая или не положительно определенная (например, с седловой особенностью), но C – симметрическая или положительно определенная. Если у A было преобладание по главной диагонали, то и C обладает этим же свойством.

Рассмотрим сложность вычисления одной итерации алгоритма:

$$Bx + b = (I - tC^{-1}A)x + b = (1 - t)x - tC^{-1}Ox + b. \tag{3}$$

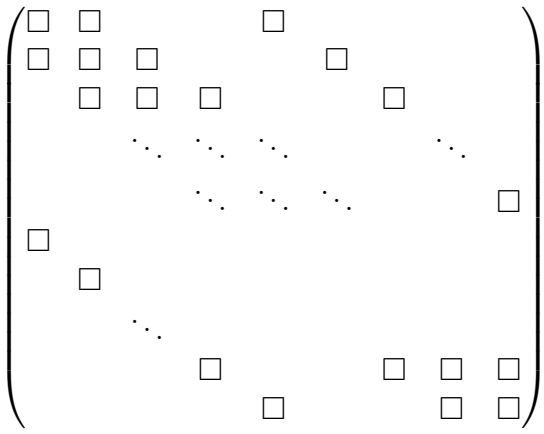
Основное время вычисления одной итерации занимает вычисление вектора $C^{-1}Ox$.

2. Хранение блочно-ленточных матриц

Разреженные матрицы хранятся в памяти [6] только ненулевыми элементами и номерами их строк и столбцов (итого, 3 числа). В некоторых случаях заранее известно, что у разреженной матрицы все ненулевые элементы сосредоточены на нескольких (м.б. блочных) диагоналях. Если хранить только эти диагонали, то расход памяти для хранения такой матрицы будет меньше, чем в предыдущем случае, поскольку для таких элементов не следует запоминать номера строк и столбцов.

Ленточная матрица возникает при численных методах конечно-разностных или конечно-элементных при покрытии прямоугольника прямоугольной сеткой (и при соответствующей нумерации узлов). Если область односвязная и не прямоугольная, то на эту область можно построить отображение прямоугольника с его прямоугольной сеткой, получив ту же ленточную матрицу.

Если построить сетку на (трехмерном) прямоугольном параллелепипеде, узлы которой занумеровать соответствующим образом, матрица конечно-разностной схемы также получится ленточной. Можно рассматривать отображения прямоугольного параллелепипеда с координатной сеткой на другие области, перенося на них сетку.



Пятидиагональная блочно-ленточная матрица

Если распределенную матрицу хранить тройками чисел (i, j, $x[i, j]$), где целые (4 байта) i, j – номера строки и столбца, в которых хранится ненулевой элемент матрицы $x[i, j]$ (тип double), то объем памяти для одного ненулевого элемента равен 16 байт $(4 + 4 + 8)$. Ленточную матрицу можно хранить диагоналями. Каждая диагональ – это одномерный массив double. Объем памяти получается в 2 раза меньше.

Если рассматривается не одно дифференциальное уравнение, а система дифференциальных уравнений на регулярной области, то матрица СЛАУ получается

блочно-ленточной. Элементы матрицы СЛАУ – блоки (матрицы), размерность которых равна количеству уравнений в исходной системе дифференциальных уравнений. Такие же матрицы возникают и при использовании метода конечных элементов.

Каждая блочная диагональ блочно-ленточной матрицы хранится в памяти одномерным массивом. Элементы каждого блока размещаются в оперативной памяти рядом построчно. Такое размещение приводит к минимизации кэш-промахов. Диагональ матрицы A состоит из всех элементов A_{ij}, для которых $(i-j) = k$. Это позволяет легко по номеру строки элемента находить номер столбца.

3. Компилятор для ускорения решателей и результаты численных экспериментов

Программы решения СЛАУ с блочно-ленточной матрицей имеют особенности, учитывая которые можно повысить производительность алгоритма. После LU-разложения по главной диагонали у матриц L и U часть программных циклов имеет треугольный вид. Блоки по главной диагонали имеют треугольный вид. Разложение симметричной матрицы можно получить методом Холецкого [7]. Блоки по главной диагонали нижне-треугольной матрицы L обозначим $D[i], i = 1, 2, ..., n$. Эти блоки имеют нижне-треугольный вид. Блоки, расположенные на поддиагонали, обозначим $B[i], i = 2, 3, ..., n$. Обозначая $x[i]$ и $h[i], i = 1, 2, ..., n$ блочные векторы неизвестных и правой части соответственно, получим блочную систему уравнений:

$$D[1] \cdot x[1] = h[1]; D[i] \cdot x[i] + B[i] \cdot x[i-1] = h[i],\ i = 2, 3, ..., n. \tag{4}$$

Аналогичные уравнения возникают при решении СЛАУ с верхнетреугольной матрицей. Программа решения блочно-ленточных уравнений (4) содержит двойные циклы с квадратным и треугольным пространством итераций (эти двойные циклы сканируют квадратный и треугольный блоки соответственно). Количество итераций в этих циклах невелико, и раскрутка таких циклов может дать заметное ускорение (что подтверждается численными экспериментами).

Блочные ленты (диагонали) хранятся в памяти в виде одномерных массивов. Это улучшает локализацию данных, но усложняет индексы элементов такого массива. Индексные выражения элементов массива аффинно зависят от счетчиков циклов, в которых находятся вхождения этих элементов. Оптимизация вычисления таких индексных выражений может достигаться преобразованиями «вычисление общих подвыражений», «вынос инвариантов из цикла» [8–10].

4. Численные эксперименты

Авторами статьи разработан экспериментальный решатель СЛАУ с блочно-ленточными матрицами. Решатель ориентирован на внедрение в конечно-элементный пакет ACELAN-COMPOS. Задача программы – нахождение решений СЛАУ для сетки $100 \times 100 \times 100$. Для текущей задачи предполагаются блоки размера 4. Задача трехмерная, поэтому в простейшем случае количество блочных диагоналей равно 7. Вычисление нормы $|x^{(k+1)} - x^k|$ занимает время, сопоставимое со временем одной итерации, поэтому проверка проводится через группу итераций (выбран размер группы 50). Итого, количество ненулевых элементов матрицы примерно $7 \cdot 4^2 \cdot 10^6 \approx 10^8$. Матрица системы уравнений имеет седловую особенность, положительных собствен-

ных чисел – $4 \cdot 666666$, отрицательных собственных чисел – $4 \cdot 333333$. Подматрица с положительными собственными числами и подматрица с отрицательными собственными числами имеют преобладание по главной диагонали. Учитывая симметрию, в памяти можно хранить половину элементов $5 \cdot 10^7$ восьмибайтных чисел (double). На каждой итерации итерационного алгоритма каждый элемент матрицы используется 1 раз. Итого, при выполнении алгоритма производится $50 \cdot 10^8 = 5 \cdot 10^9$ операций над числами типа double.

Программа решения СЛАУ написана на языке C (C89). С помощью ОРС (оптимизирующей распараллеливающей системы) разработанная Программа (исходная) преобразована к Программе-2 (оптимизированная). Оптимизация ОРС выполняет последовательность преобразований из набора «линеаризация», «вынос инвариантных выражений из циклов», «раскрутка циклов», «канонизация».

Компьютер, на котором проводились численные эксперименты, имеет следующие характеристики: CPU – i7-9700; 3,00 GHz; L1 – 256 Kb; L2 – 2 Mb; L3 – 12 Mb; частота системной шины – 8 GT/s; максимальная пропускная способность – 41,6 GB/s; RAM – DDR4, 16Gb; мин. частота – 1600 МГц; макс. частота – 2666 МГц. Из таблицы численных экспериментов видно, что оптимизирующий предкомпилятор ускоряет решатель на 30% (при использовании компилятора ICC).

Таблица

Результаты численных экспериментов

Компилятор, ключ оптимизации	**GCC, O2**	**GCC, O3**	**ICC, O2**	**ICC, O3**	**MSVS, O2**
Программа, с	5,0357	5,3133	7,0142	6,7631	10,1573
Программа-2, с	4,9197	4,6092	4,2593	4,2085	9,2703

Заключение

Распараллеливание пока не проводилось. При решении СЛАУ с треугольными матрицами возникают циклы с линейной рекуррентной зависимостью [11], распараллеливание которых предполагается реализовать в следующей работе. Высокое быстродействие представленного решателя достигнуто несколькими действиями. Некоторые оптимизации выполнены предкомпилятором. Существенное ускорение достигнуто за счет представления матрицы системы блочными диагоналями и за счет хранения блоков в одномерном массиве.

За многие годы накоплены сотни пакетов прикладных программ, как с открытым кодом, так и коммерческих. Большинство этих пакетов нуждаются в быстроте их решателей. Разумеется, решатели, написанные более 10 лет назад, не ориентированы на современные процессоры. Более того, решатель, оптимизированный под один процессор, может неэффективно отображаться на другой процессор, даже выпущенный тем же производителем. Такая ситуация приводит к проблеме эффективной переносимости кода. Создание новых процессоров с более сложными архитектурами обостряет проблему переносимости кода с сохранением высокой эффективности. Эту проблему отчасти могли бы решать оптимизирующие компиляторы нового поколения, а в первое время могут использоваться специальные предкомпиляторы для решателей пакетов.

Работа проводилась при финансовой поддержке гранта Правительства РФ № 075-15-2019-1928.

Литература

1. Гун, В.С. Система построения двумерных ортогональных сеток общего назначения / В.С. Гун, В.С. Морозова, В.Л. Поляцко // Математическое моделирование. – 2017. – Т. 29, № 11. – С. 71–88.
2. Fang Chen. Updated Preconditioned Hermitian and Skew-Hermitian Splitting-Type Iteration Methods for Solving Saddle-Point Problems / Fang Chen, Tian-Yi Li, Kang-Ya Lu // Computational and Applied Mathematics. – 2020. – V. 39. – Article ID: 162. – 10 p.
3. Оптимизирующая распараллеливающая система. – URL: www.ops.rsu.ru (дата обращения 05.08.2021).
4. Козин, Р.Г. Алгоритмы численных методов линейной алгебры и их программная реализация / Р.Г. Козин. – М.: НИЯУ МИФИ, 2012.
5. Graham, S.L. Getting up to Speed: The Future of Supercomputing / S.L. Graham, M. Snir, C.A. Patterson. – Washington: National Academies Press, 2005.
6. Писсанецки, С. Технология разреженных матриц / С. Писсанецки. – М.: Мир, 1988.
7. Gill, P.E. On the Stability of Cholesky Factorization for Symmetric Quasidefinite Systems / P.E. Gill, A.S. Sauders, J.R. Shinnerl // SIAM Journal on Matrix Analysis and Applications. – 1996. – V. 17, № 1. – P. 35–46.
8. Allen, R. Optimizing Compilers for Modern Architectures: A Dependence-Based Approach / R. Allen, K. Kennedy. – San Francisco: Morgan Kaufmann Publisher, 2002.
9. Евстигнеев, В.А. Оптимизирующие преобразования в распараллеливающих компиляторах / В.А. Евстигнеев, В.Н. Касьянов // Программирование. – 1996. – № 6. – С. 12–26.
10. Muchnick, S. Advanced Compiler Design Implementation / S. Muchnick. – San Francisco: Morgan Kaufmann Publisher, 1997.
11. Steinberg, O.B. Parallelization of Recurrent Loops Due to the Preliminary Computation of Superpositions / O.B. Steinberg // Вестник ЮУрГУ. Серия: Математическое моделирование и программирование. – 2020. – Т. 13, № 3. – С. 59–67.

Борис Яковлевич Штейнберг, доктор технических наук, старший научный сотрудник, заведующий кафедры алгебры и дискретной математики, Южный федеральный университет, (г. Ростов-на-Дону, Российская Федерация), borsteinb@mail.ru.

Александр Александрович Василенко, аспирант, ассистент кафедры информатики и вычислительного эксперимента, Южный федеральный университет, (г. Ростов-на-Дону, Российская Федерация), avas@sfedu.ru.

Вадим Владимирович Веселовский, студент, кафедра алгебры и дискретной математики, Южный федеральный университет, (г. Ростов-на-Дону, Российская Федерация), vesvadvlad1@gmail.com.

Никита Александрович Живых, студент, кафедра алгебры и дискретной математики, Южный федеральный университет, (г. Ростов-на-Дону, Российская Федерация), zivih.n@gmail.com.

Поступила в редакцию 25 января 2021 г.

MSC 15A06 DOI: 10.14529/mmp210309

SOLVERS FOR SYSTEMS OF LINEAR ALGEBRAIC EQUATIONS WITH BLOCK-BAND MATRICES

***B.Ya. Steinberg*[1], *A.A. Vasilenko*[1], *V.V. Veselovskiy*[1], *N.A. Zhivykh*[1]**
[1]Southern Federal University, Rostov-on-Don, Russian Federation
E-mail: borsteinb@mail.ru, avas@sfedu.ru, vesvadvlad1@gmail.com, zivih.n@gmail.com

The article proposes methods for constructing fast solvers for systems of linear algebraic equations with block-band matrices. A data structure for efficient storage of such matrices in RAM and a fast algorithm for solving systems of linear equations with such matrices based on this structure are proposed. The article is focused on the creation of solvers based on iterative algorithms for solving systems of linear equations with both symmetric matrices and matrices having a saddle-point singularity. It is proposed to develop and use a special precompiler to speed up the solver. The experimental solver is implemented in C, and the preliminary compilation is based on the Optimizing Parallelizing System in this paper. The results of numerical experiments that demonstrate high efficiency of the developed methods, including the efficiency of the precompiler, are presented.

Keywords: concurrent computing; cache misses; systems of linear algebraic equations.

References

1. Gun V.S., Morozova V.S., Polyaczko V.L. The General Purpose System for Construction of Two-Dimensional Orthogonal Grids. *Matematicheskoe modelirovanie*, 2017, vol. 29, no. 11, pp. 71–88. (in Russian)
2. Fang Chen, Tian-Yi Li, Kang-Ya Lu. Updated Preconditioned Hermitian and Skew-Hermitian Splitting-Type Iteration Methods for Solving Saddle-Point Problems. *Computational and Applied Mathematics*, 2020, vol. 39, article ID: 162, 10 p. DOI: 10.1007/s40314-020-01187-7
3. *Optimizing Parallelizing System* (2016). Available at: www.ops.rsu.ru (accessed 05.08.2021)
4. Kozin R.G. *Algoritmy chislennyx metodov lineynoy algebry i ix programmnaya realizaciya* [Algorithms of Numerical Methods of Linear Algebra and Their Software Implementation], Moscow, NIYaU MIFI, 2012. (in Russian)
5. Graham, S.L., Snir M., Patterson C.A. *Getting up to Speed: the Future of Supercomputing*, Washington, National Academies Press, 2005.
6. Pissaneczki S. *Texnologiya razrezhennyx matricz* [Sparse Matrix Technology], Moscow, Mir, 1988. (in Russian)
7. Gill, P.E., Sauders A.S., Shinnerl J.R. On the Stability of Cholesky Factorization for Symmetric Quasidefinite Systems. *SIAM Journal on Matrix Analysis and Applications*, 1996, vol. 17, no. 1, pp. 35–46.
8. Allen, R. *Optimizing Compilers for Modern Architectures: a Dependence-Based Approach*, San Francisco, Morgan Kaufmann Publisher, Academic Press, 2002.
9. Evstigneev V.A., Kas'yanov V.N. [Optimizing Conversions in Parallelizing Compilers]. *Programmirovanie*, 1996, no. 6, pp. 12–26. (in Russian)
10. Muchnick S. *Advanced Compiler Design Implementation*, San Francisco, Morgan Kaufmann Publisher, 1997.
11. Steinberg O.B. Parallelization of Recurrent Loops Due to the Preliminary Computation of Superpositions. *Bulletin of the South Ural State University. Series: Mathematical Modelling, Programming and Computer Software*, 2020, vol. 13, no. 3, pp. 59–67. DOI: 10.14529/mmp200305

Received January 25, 2021