МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение высшего образования
«Национальный исследовательский Нижегородский государственный университет им. Н.И. Лобачевского» (ННГУ)

Институт информационных технологий, математики и механики

Направление подготовки: «Фундаментальная информатика и информационные технологии»
Магистерская программа: «Инженерия программного обеспечения»

ОТЧЕТ

по лабораторной работе
«Блочное LU – разложение для квадратной матрицы»

Выполнил: студент группы 382006-1м
__________________А.А. Солуянов

Проверил:
к.ф.-м. н., доц., доцент каф. МОСТ

__________________К.А. Баркалов

Нижний Новгород
2021

Оглавление

Введение ... 3
Постановка задачи ... 4
Описание метода ... 4
Трудоемкость ... 5
Реализация ... 6
Описание алгоритма ... 6
Схема распараллеливания ... 8
Подтверждение корректности алгоритма ... 8
Результаты ... 10
Тестовая инфраструктура ... 10
Эксперименты ... 10
Заключение ... 13
Литература ... 14

Введение

Решение систем линейных алгебраических уравнений (СЛАУ) является достаточно важной вычислительной задачей (примерно 75% всех расчетных математических задач приходится на их решение). С решением СЛАУ связаны такие задачи, как вычисление определителей, обращение матриц, вычисление собственных значений и собственных векторов матриц, интерполирование, аппроксимация по методу наименьших квадратов, решение систем дифференциальных уравнений и многие другие. Современная вычислительная математика располагает большим арсеналом методов решения СЛАУ, а математическое обеспечение ЭВМ – многими пакетами программ и программными системами, позволяющими решать СЛАУ.

Методы решения СЛАУ можно разделить на две группы:

1. Прямые методы позволяют найти точное решение системы (метод Крамера, разложение Холецкого, метод Гаусса, LU – разложение и т.д.)
2. Итерационные – позволяют получить решение в результате последовательных приближений (методы Зейделя и Якоби,

Очень важно знать методы решения СЛАУ и уметь их применять.

В данной работе рассматривается прямой метод решения СЛАУ – метод LU – разложения.

Постановка задачи

Описание метода

LU-разложение — это представление матрицы A в виде $A = LU$, где L — нижнетреугольная матрица с единичной диагональю, а U — верхнетреугольная матрица. LU – разложение является модификацией метода Гаусса.

LU-разложение существует только в том случае, когда матрица A обратима, а все ведущие (угловые) главные миноры матрицы A невырождены.

Недостаток стандартного алгоритма LU-разложения обусловлен тем, что его вычисления плохо соответствует правилам использования кэш-памяти – быстродействующей дополнительной памяти компьютера, используемой для хранения копии наиболее часто используемых областей оперативной памяти.

Это связано с тем, что при больших размерах матрицы во время арифметических операций над матрицами зачастую приходится обращаться к элементам, не лежащим вблизи в памяти, что приводит к неэффективному использованию кэша. Возможный способ улучшения ситуации – укрупнение вычислительных операций, приводящее к последовательной обработке некоторых прямоугольных подматриц матрицы A.

LU-разложение можно организовать так, что матричные операции (реализация которых допускает эффективное использование кэш-памяти) станут основными. Для этого представим матрицу $A \in R^{n\times n}$ в блочном виде

$$A = \begin{array}{cc} \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} & \begin{array}{l} r \\ n-r \end{array} \\ \begin{array}{cc} r & n-r \end{array} & \end{array},$$

где r – блочный параметр, A_{11} – подматрица матрицы A размера $r \times r$, A_{12} – размера $r \times (n - r)$, A_{21} – размера $(n - r) \times r$, A_{22} – размера $(n - r) \times (n - r)$. Компоненты L и U искомого разложения также запишем в блочном виде

$$L = \begin{array}{cc} \begin{bmatrix} L_{11} & 0 \\ L_{21} & L_{22} \end{bmatrix} & \begin{array}{l} r \\ n-r \end{array} \\ \begin{array}{cc} r & n-r \end{array} & \end{array}, \quad U = \begin{array}{cc} \begin{bmatrix} U_{11} & U_{12} \\ 0 & U_{22} \end{bmatrix} & \begin{array}{l} r \\ n-r \end{array} \\ \begin{array}{cc} r & n-r \end{array} & \end{array},$$

де L_{11}, L_{21}, L_{22}, U_{11}, U_{12}, U_{22} – соответствующего размера подматрицы матриц L и U. Рассмотрим теперь связь между исходной матрицей и ее разложением в блочном виде.

$$\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} = \begin{bmatrix} L_{11} & 0 \\ L_{21} & L_{22} \end{bmatrix} \cdot \begin{bmatrix} U_{11} & U_{12} \\ 0 & U_{22} \end{bmatrix} = \begin{bmatrix} L_{11}U_{11} & L_{11}U_{12} \\ L_{21}U_{11} & L_{21}U_{12} + L_{22}U_{22} \end{bmatrix}$$

Блоки L_{11} и U_{11} можно найти, применив стандартный метод Гаусса .Затем, решая треугольные системы с несколькими правыми частями будет получено решение для блоков L_{21} и U_{12}.

Следующий шаг алгоритма состоит в вычислении редуцированной матрицы $\widetilde{A}_{22}$, в процессе которого используются ставшие известными блоки L_{21} и U_{12} и соотношение $A_{22} = L_{21}U_{12} + L_{22}U_{22}$.

$$\widetilde{A}_{22} = A_{22} - L_{21}U_{12} = L_{22}U_{22}$$

Как следует из данной формулы, LU-разложение редуцированной матрицы $\widetilde{A}_{22}$ совпадает с искомыми блоками L_{22}, U_{22} матрицы A, и для его нахождения можно применить описанный алгоритм рекурсивно.

Трудоемкость

Приведенная блочная схема требует порядка $2/3n^3$ операций. Оценим долю матричных операций.

Пусть размер матрицы кратен размеру блоку, т.е. $n = rN$.

Операции, не являющиеся матричными, используются при выполнении разложения матрицы A на L и U и требуют $2/3r^3$ операций.

В процессе блочного разложения потребуется решать N подобных систем, поэтому доля матричных операций можно оценить как

$$1 - \frac{\frac{N2r^3}{3}}{\frac{2n^3}{3}} = 1 - \frac{1}{N^2}$$

Реализация

Описание алгоритма

Входными параметрами алгоритма является указатель на массив, в котором по строкам хранится матрица A и размерность матрицы n.

Формат выхода должен выглядеть как два указателя на массивы, в которых по строкам записаны матрицы L и U.

Весь итерационный процесс можно условно разделить на 4 секции:

1. Подсчет матриц L_{ii} и U_{ii} для блоков, располагающихся на диагонали.
2. Вычисление подматрицы U_{12}, решая верхнюю систему уравнений.
3. Вычисление подматрицы L_{21}, решая нижнюю систему уравнений.
4. Вычисление редуцированной матрицы $\widetilde{A}_{22}$.

Поэтому удобно выделить эти пункты в отдельные функции (Листинг 1-4).

```
void DiagonalMatrixDecomposition(int offset, int size, int &N, double* A, double* L, double*
U) {
    for (int i = 0; i < size; i++) {
        for (int j = 0; j < size; j++) {
            U[N * (offset + i) + offset + j] = A[N * (offset + i) + offset + j];
        }
    }

    for (int i = 0; i < size; i++) {
        L[N * (offset + i) + offset + i] = 1;

        for (int k = i + 1; k < size; k++) {
            double mu = U[N * (offset + k) + offset + i] / U[N * (offset + i) + offset + i];

            for (int j = i; j < size; j++) {
                U[N * (offset + k) + offset + j] -= mu * U[N * (offset + i) + offset + j];
            }


            L[N * (offset + k) + offset + i] = mu;
            L[N * (offset + i) + offset + k] = 0;
        }
    }

    for (int i = 1; i < size; i++) {
        for (int j = 0; j < i; j++) {
            U[N * (offset + i) + offset + j] = 0;
        }
    }
}
```

Листинг 1. Выполнение разложения для диагональных блоков

```
void SolveUpper(int offset, int size, int& N, double* A, double* L, double* U) {
    int row_num;
    int col_num;

#pragma omp parallel for private(row_num, col_num)
    for (int k = 0; k < N - offset - BLOCK_SIZE; k++) {
        row_num = offset * N;
        col_num = offset + BLOCK_SIZE;

        U[row_num + col_num + k] = A[row_num + col_num + k];

        for (int i = 1; i < size; i++) {
            U[row_num + i * N + col_num + k] = A[row_num + i * N + col_num + k];

            for (int j = 0; j < i; j++) {
                U[row_num + i * N + col_num + k] -= L[row_num + i * N + j + offset] *
U[row_num + j * N + col_num + k];
            }
        }
    }
}
```

Листинг 2. Решение верхней системы уравнений

```
void SolveLower(int offset, int size, int& N, double* A, double* L, double* U) {
    int row_num;
    int col_num;

#pragma omp parallel for private(row_num, col_num)
    for (int k = 0; k < N - offset - BLOCK_SIZE; k++) {
        row_num = (offset + BLOCK_SIZE + k) * N;
        col_num = offset;

        L[row_num + col_num] = A[row_num + col_num] / U[offset * N + col_num];

        for (int i = 1; i < size; i++) {
            L[row_num + col_num + i] = A[row_num + col_num + i];

            for (int j = 0; j < i; j++) {
                L[row_num + col_num + i] -= L[row_num + j + offset] * U[(offset + j) * N +
col_num + i];
            }

            L[row_num + col_num + i] /= U[(offset + i) * N  + col_num + i];
        }
    }
}
```

Листинг 3. Решение нижней системы уравнений

```
void UpdateDiagonalSubmatrix(int offset, int size, int& N, double* A, double* L, double* U)
{

    int row_offset;

#pragma omp parallel for if (N - offset > 400) private(row_offset)
    for (int i = 0; i < N - offset - BLOCK_SIZE; i++) {

        row_offset = (offset + BLOCK_SIZE + i) * N;

        for (int j = 0; j < N - offset - BLOCK_SIZE; j++) {
            double sum = 0;

            for (int k = 0; k < BLOCK_SIZE; k++) {
                sum += L[row_offset + k + offset] * U[(offset + k) * N + offset + BLOCK_SIZE
+ j];
            }

            A[row_offset + offset + BLOCK_SIZE + j] -= sum;
        }
    }
}
```

Листинг 4. Вычисление редуцированной матрицы

В конечном счете целевую функцию можно определить последовательным вызовом всех 4 функций N раз.

Схема распараллеливания

Так как большая часть алгоритма состоит циклов по обходу матриц, целесообразно применить механизма их распараллеливания.

Для этого была использована библиотека OpenMP, а именно директива «#pragma omp parallel for», с помощью которой циклы были поделены между несколькими потоками для независимого исполнения.

Подтверждение корректности алгоритма

Для проверки корректности рабы блочного LU – разложения был разработан метод, сравнивающий модуль разности соответствующих значений матрицы A и результата перемножения матриц L и U с некоторой малой константой (Листинг 5).

```
#define  num(row,col)  ((col) + (row) * size)
void IsCorrect(double* A, double* L, double* U, int size, double eps) {
    for (int i = 0; i < size; ++i)
        for (int j = 0; j < size; ++j)
        {
            double sum = 0.;
            for (int k = 0; k < size; ++k)
                sum += L[num(i, k)] * U[num(k, j)];

            if (abs(A[num(i, j)] - sum) <= eps) {
                continue;
            }
            else {
                std::cout << "LU decomposition isn't correct (Error > eps)" <<
std::endl;
                return;
            }
        }
    std::cout << "Correct!" << std::endl;
}
```

Листинг 5. Проверка корректности алгоритма

Результаты

Тестовая инфраструктура

Вычислительные эксперименты проводились с использованием следующей инфраструктуры (Таблица 1).

Таблица 1. Тестовая инфраструктура

Процессор	4 ядра, Intel(R) Core(TM) i7-8550U CPU @ 1.80GHz 1.99 GHz
Память (RAM, Cache)	8,00 ГБ, L1 – 256 Kb, L2 – 1 Mb, L3 – 8 Mb
Операционная система	Windows 10
Среда разработки	Visual Studio 2019
Компилятор	Intel(R) oneAPI DPC++/C++ Compiler 2021.1
Библиотеки	OpenMP

Эксперименты

Опытным путем было установлено для данной тестовой инфраструктуры оптимальным значением размером блока является значение 128 (Таблица 2).
Далее с этим значением были проведены испытания (Таблица 3) зависимости времени выполнения алгоритма с разным числом потоков (от 1 до 4 по числу имеющихся физических ядер).

	16	32	64	128	256	512
1000	0,2375	0,120495	0,107233	0,110736	0,100439	0,156114
2000	1,1369	1,12101	1,09944	1,02716	1,14272	1,45883
3000	3,28771	3,33355	3,2762	3,09814	3,90548	5,10591
4000	8,04812	8,02063	8,0707	7,4997	10,7184	16,7397
5000	19,5974	15,3946	15,9139	14,8915	22,6361	28,9361

Таблица 2. Зависимость времени алгоритма от размера блока

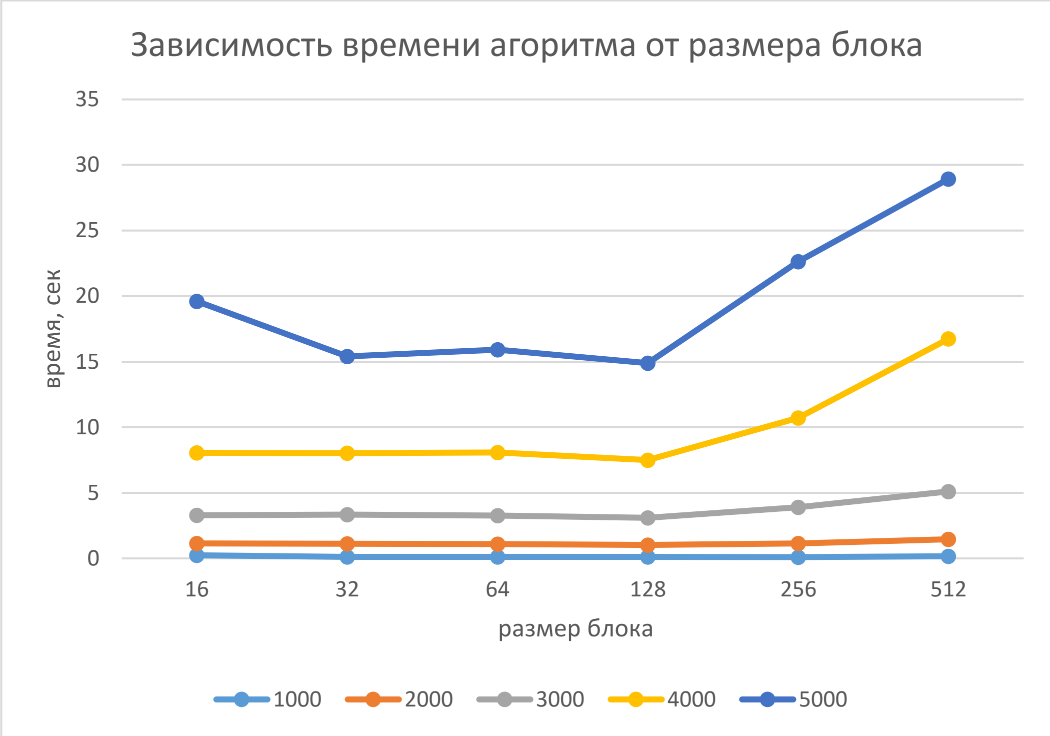

Рисунок 1. Зависимость времени алгоритма от размера блока

	1	2	3	4
1000	0,43813	0,290818	0,214973	0,168785
2000	3,22723	1,80779	1,54497	1,46131
3000	10,3489	5,89579	4,83923	3,95649
4000	25,7261	13,6968	11,9213	11,46
5000	51,8605	33,5792	25,7835	21,5718

Таблица 3. Зависимость времени алгоритма от числа потоков при размере блока 128

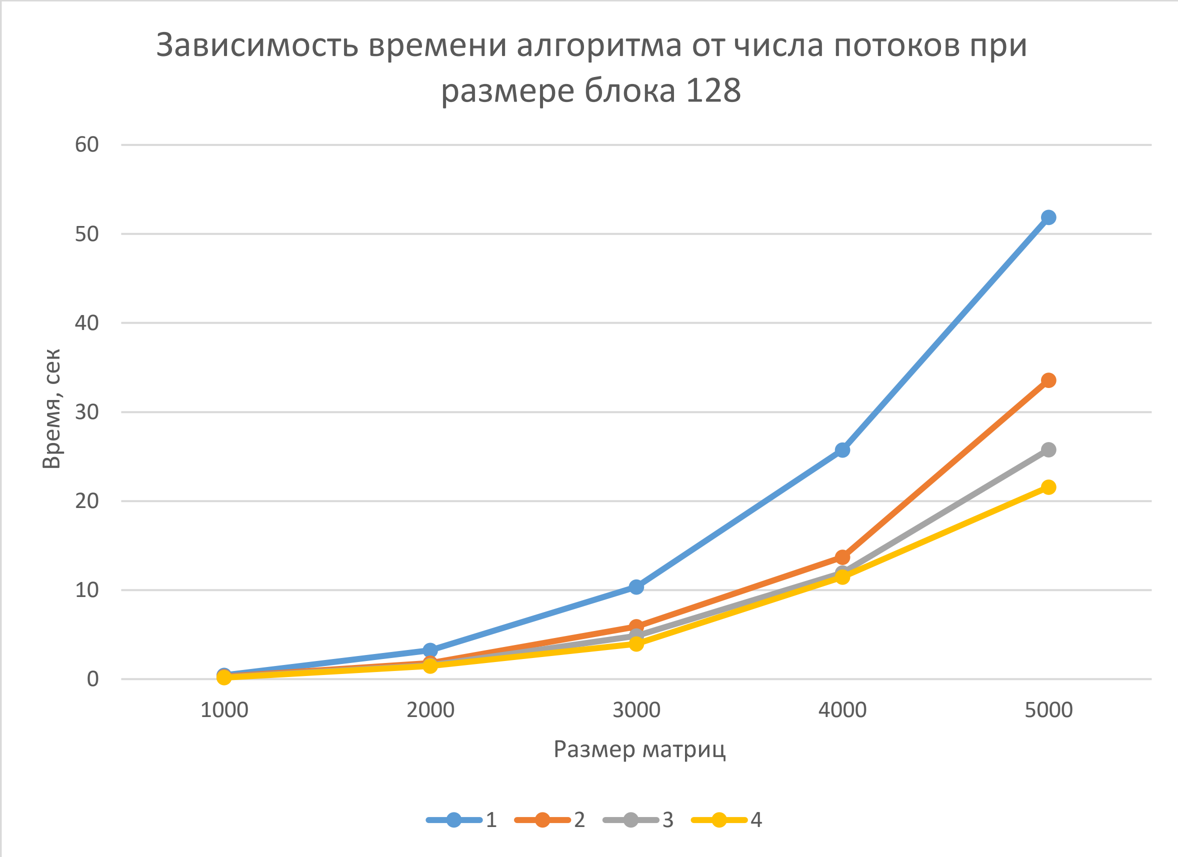

Рисунок 2. Зависимость времени алгоритма от числа потоков при размере блока 128

Заключение

В ходе данной лабораторной работы был изучен прямой метод решения СЛАУ - алгоритм блочного LU – разложения, применяемый к обратимой матрице А с невырожденными главными минорами.
В отличие от стандартного алгоритма LU – разложения, его модификация путем применения блочного разбиения позволила ускорить время работы программы за счет более эффективного использования кэш – памяти и удобного процесса распараллеливания вычислений.
В результате анализа производительности параллельной версии блочного разложения по сравнению с последовательной отмечается ее рост, что говорит об эффективности применяемой схемы.

Литература

1. Баркалов К.А. Образовательный комплекс «Параллельные численные методы». - Н.Новгород, Изд-во ННГУ 2011.
2. Самарский А.А., Гулин А.В. Численные методы. – М.: Наука, 1989.
3. Белов С.А., Золотых Н.Ю. Численные методы линейной алгебры. – Н.Новгород, Изд-во ННГУ, 2005
4. Вербицкий В.В., Реут В.В. Введение в численные методы алгебры: учебное пособие/ В.В. Вербицкий, В.В. Реут. - Одесса: Одесский национальный университет имени И.И. Мечникова, 2015. - 165 с