МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ 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

Кафедра: математического обеспечения и суперкомпьютерных технологий

Направление подготовки: «Программная инженерия»
Профиль подготовки: «Технологии цифровой трансформации»

ОТЧЕТ

по предмету

“Параллельные численные методы”

Выполнил: студент группы
3824М1ПР1 Д.Э. Булгаков
Подпись

Проверил: д.т.н., доц.,
К.А.Баркалов
Подпись

Нижний Новгород
2025

Содержание

1. Введение

Разложе́ние Холе́цкого (метод квадратного корня) — представление симметричной положительно определённой матрицы A в виде

$$A = LL^T,$$

где L — нижняя треугольная матрица со строго положительными элементами на диагонали. Иногда разложение записывается в эквивалентной форме:

$$A = U^T U,$$

где $U = L^T$ — верхняя треугольная матрица.

Разложение Холецкого всегда существует и единственно для любой симметричной положительно определённой матрицы.

2. Постановка задачи

Необходимо реализовать блочное разложение Холецкого для симметричной положительно определённой матрицы A, то есть представить её в виде произведения:

$$A = LL^T,$$

где L — нижняя треугольная матрица со строго положительными элементами на диагонали.

Требуется использовать технологии параллельного программирования с помощью OpenMP для повышения производительности.

3. Последовательный алгоритм

Шаги алгоритма

Пусть матрица A имеет размерность $n \times n$, и необходимо найти нижнюю треугольную матрицу L, для чего выполняются следующие шаги:

1. **Инициализация:** Создаём матрицу L размерности $n \times n$, инициализируя все её элементы нулями. Элементы на диагонали будут вычисляться как положительные числа, а элементы под диагональю — как значения, соответствующие разложению.

2. **Цикл по строкам и столбцам:** Основной цикл состоит из двух вложенных:

   - Внешний цикл перебирает строки матрицы A и вычисляет элементы матрицы L.
   - Внутренний цикл вычисляет элементы L в каждой строке, используя формулы разложения Холецкого.

   Для каждого элемента $L_{i,j}$, где $i \geq j$, вычисление происходит по следующей формуле:

   $$L_{i,j} = \sqrt{A_{i,j} - \sum_{k=1}^{j-1} L_{i,k} L_{j,k}},$$

   если $i = j$ (элементы на диагонали), или

   $$L_{i,j} = \frac{1}{L_{j,j}} \left(A_{i,j} - \sum_{k=1}^{j-1} L_{i,k} L_{j,k} \right),$$

   если $i > j$ (элементы ниже диагонали).

3. **Построение матрицы L:** После вычисления всех элементов матрицы L, мы получаем её как результат разложения Холецкого. После этого матрица L может быть использована для проверки, путём вычисления LL^T, и сравнения с исходной матрицей A.

4. **Завершение:** Алгоритм завершается после вычисления всех элементов матрицы L.

Время выполнения

Время выполнения этого алгоритма в последовательной версии имеет сложность $O(n^3)$, что обусловлено необходимостью вычисления каждого элемента матрицы L через сумму по всем предыдущим элементам строки и столбца. Это приводит к тройному вложенному циклу, где каждый цикл выполняется n раз.

$$T_{\text{seq}} = O(n^3).$$

4. Параллельный алгоритм

Параллельный алгоритм разложения Холецкого

Для распараллеливания алгоритма разложения Холецкого применяется блочное разложение.

Алгоритм

1. **Инициализация матрицы** L. На первом шаге происходит инициализация всех элементов матрицы L значениями нулей. Это выполняется параллельно для всех элементов матрицы:

```
#pragma omp parallel for collapse(2)
for (int i = 0; i < n; ++i)
    for (int j = 0; j < n; ++j)
        L[i * n + j] = 0.0;
```

2. **Обработка блоков матрицы**. Основной цикл алгоритма выполняет обработку матрицы по блокам. Каждый блок размером $B \times B$ обрабатывается отдельно, начиная с диагональных элементов:

```
for (int k = 0; k < n; k += blockSize) {
    int end = std::min(k + blockSize, n);
```

   (a) **Вычисление диагональных элементов** для каждого блока:

```
    for (int i = k; i < end; ++i) {
        double sum = 0.0;
        for (int p = 0; p < i; ++p)
            sum += L[i * n + p] * L[i * n + p];
        L[i * n + i] = std::sqrt(A[i * n + i] - sum);
    }
```

   (b) **Вычисление элементов нижней треугольной части блока**:

```
for (int j = i + 1; j < end; ++j) {
    double sum = 0.0;
    for (int p = 0; p < i; ++p)
        sum += L[j * n + p] * L[i * n + p];
    L[j * n + i] = (A[j * n + i] - sum) / L[i * n + i];
}
```

(c) **Обновление оставшихся блоков**: Обновление правого блока (верхняя часть матрицы A, которая не была обработана в первом цикле):

```
#pragma omp parallel for
for (int i = end; i < n; ++i) {
    for (int j = k; j < end; ++j) {
        double sum = 0.0;
        for (int p = 0; p < j; ++p)
            sum += L[i * n + p] * L[j * n + p];
        L[i * n + j] = (A[i * n + j] - sum) / L[j * n + j];
    }
}
```

Оценка сложности

Параллельный алгоритм имеет теоретическую сложность $O\left(\frac{n^3}{p}\right)$, где p — количество потоков, используемых в OpenMP. Эта оценка предполагает идеальную масштабируемость, что, однако, редко наблюдается на практике из-за накладных расходов на создание и синхронизацию потоков.

5. Результаты

Алгоритм был реализован на языке C++ с использованием библиотеки OpenMP для параллельных вычислений. Для проверки корректности параллельного алгоритма была использована случайно сгенерированная положительно определенная матрица. Результаты работы программы для различных размеров матриц и количества потоков приведены ниже.

Далее для тестов использовались случайно сгенерированные матрицы. Размер блока выбран 32. Время выполнения в секундах.

Потоки/Размерность	**1000x1000**	**3000x3000**	**5000x5000**
1	0.4551	12.1241	56.6328
2	0.2386	6.2803	28.8508
4	0.1358	3.2617	14.8578
6	0.0994	2.3387	10.5903

Таблица 1: Время выполнения разложения Холецкого для разных размеров матриц и количества потоков

6. Заключение

В ходе выполнения работы был изучен алгоритм разложения Холецкого. На основе полученных знаний был реализован блочный алгоритм разложения Холецкого на языке C++ с использованием библиотеки OpenMP для параллельных вычислений.

7. Список литературы

1. Нестеренко, В. П., Рябов, А. В. *Методы численного решения линейных систем.* – М.: Наука, 2009. – 512 с.

2. Чесноков, В. И. *Параллельные вычисления. Теория и практика.* – М.: МЦНМО, 2017. – 304 с.

3. Папалексис, В. В., Ортега, Дж. *Введение в параллельные методы решения линейных систем.* — М.: Мир, 1991. — 376 с.

4. OpenMP Architecture Review Board. *OpenMP Application Programming Interface. Version 4.5.* – 2015. – Режим доступа:. – Загл. с экрана.

8. Приложение

```
void Cholesky_Decomposition(double* A, double* L, int n) {
    int blockSize = 32;

    #pragma omp parallel for collapse(2)
    for (int i = 0; i < n; ++i)
        for (int j = 0; j < n; ++j)
            L[i * n + j] = 0.0;

    for (int k = 0; k < n; k += blockSize) {
        int end = std::min(k + blockSize, n);

        for (int i = k; i < end; ++i) {
            double sum = 0.0;
            for (int p = 0; p < i; ++p)
                sum += L[i * n + p] * L[i * n + p];
            L[i * n + i] = std::sqrt(A[i * n + i] - sum);

            for (int j = i + 1; j < end; ++j) {
                sum = 0.0;
                for (int p = 0; p < i; ++p)
                    sum += L[j * n + p] * L[i * n + p];
                L[j * n + i] = (A[j * n + i] - sum) / L[i * n + i];
            }
        }

        #pragma omp parallel for collapse(2)
        for (int i = end; i < n; ++i) {
            for (int j = k; j < end; ++j) {
                double sum = 0.0;
                for (int p = 0; p < j; ++p)
                    sum += L[i * n + p] * L[j * n + p];
```

```
                L[i * n + j] = (A[i * n + j] - sum) / L[j * n + j];
            }
        }
    }
}
```