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
«Метод бисопряженных градиентов решения СЛАУ»

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

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 систем линейных алгебраических уравнений классифицируют на прямые (точные) и итерационные. Прямые методы основаны на выполнении конечного числа арифметических операций (метод обратной матрицы, метод Гаусса, метод разложения Холецкого и т.д.). В основе итерационных методов лежит последовательность приближений, позволяющая получить решение системы, определяемое необходимой точностью. Общими словами, такие методы устанавливают процедуру уточнения определённого начального приближения к решению. При выполнении условий сходимости они позволяют достичь любой точности путем повторения итераций. Преимущество этих методов в том, что часто они позволяют достичь решения с заранее заданной точностью быстрее прямых методов, а также позволяют решать большие системы уравнений. К итерационным методам относятся: метод сопряженных градиентов, метод бисопряженных градиентов, методы Зейделя и Якоби и т.д.

В данной работе рассматривается метод бисопряженных градиентов решения СЛАУ (итерационный метод) с разреженной матрицей A и плотными векторами x и b.

Постановка задачи

Описание метода

Метод бисопряженных градиентов является обобщением метода сопряженных градиентов на случае линейной системы $Ax = b$ с произвольной квадратной невырожденной матрицей A. Хорошо известно, что матрица $A^T A$ – симметрична и положительно определена, а система $A^T Ax = A^T b$ эквивалентная исходной. Для решения данной системы мы не можем применить метод сопряженных градиентов, так как обусловленность матрицы $A^T A$ много хуже обусловленности матрицы A, а именно $cond(A^T A) \approx cond(A)^2$.

С другой стороны, доказано, что с помощью рекуррентного соотношения небольшого порядка добиться попарной ортогонализации невязок $\{r_k\}$ в случае произвольной матрицы A невозможно. В методе бисопряженных градиентов реализована другая идея. Последовательность ортогональных невязок $\{r_k\}$ заменяется на две биортогональные последовательности $\{r_k\}$ и $\{\tilde{r_k}\}$, а последовательность сопряженных направлений $\{p_k\}$ заменена на две бисопряженные $-\{p_k\}$ и $\{\tilde{p_k}\}$.

Системы векторов $\{v_1, \dots, v_m\}$ и $\{w_1, \dots, w_m\}$ называются биортогональными, если $(v_i, w_j) = 0$ при $i \neq j$.

Задача лабораторной работы

В данной лабораторной работе требуется реализовать метод бисопряженных градиентов решения СЛАУ с разреженной матрицей и плотными векторами x и b. Для реализации параллельных вычислений необходимо использовать технологию OpenMP.

Необходимо произвести замеры времени выполнения и сравнить эффективность параллельных вычислений с последовательной реализацией.

Алгоритм решения

1. Подготовка перед итерационным процессом:
    a. Задается начальное приближение x_0
    b. $r_0 = b - Ax_0$ – вектор невязки
    c. $p_0 = r_0$
2. На каждой k-ой итерации метода вычисляются:

a. $a_k = \frac{(r_k, \tilde{r}_k)}{(Ap_k, \tilde{p}_k)}$ (1)

b. $x_{k+1} = x_k + a_k p_k$ (2)

c. $r_{k+1} = r_k - a_k A p_k$ (3)

d. $\tilde{r}_{k+1} = \tilde{r}_k - a_k A^T \tilde{p}_k$ (4)

e. $\beta_k = \frac{(r_{k+1}, \tilde{r}_{k+1})}{(r_k, \tilde{r}_k)}$ (5)

3. Критерий остановки

   $\beta_k < \varepsilon'$ или $\|r_{k+1}\| < \varepsilon$ (6)

4. Если критерий не выполняется, то вычисляются p_{k+1}, $\tilde{p}_{k+1}$ и начинается следующая итерация:

   f. $p_{k+1} = r_{k+1} + \beta_k p_k$ (7)

   g. $\tilde{p}_{k+1} = \tilde{r}_{k+1} + \beta_k \tilde{p}_k$ (8)

Трудоемкость

Подсчитаем количество операций, которое требуется для данного метода.

1. На каждой итерации:

   - Две операции умножения матрицы на вектор (Ap_k и $A^T \tilde{p}_k$) - $2n^2$ операций.
   - Четыре операции скалярного произведения - $4n$ операций.
   - Десять операций над векторами (сложение, разность векторов, умножение константы на вектор, взятие нормы) - $11n$ операций

2. Общая трудоемкость одной итерации:

$$t = 2n^2 + 15n \quad (9)$$

3. Для выполнения K итераций метода необходимо:

$$T = K(2n^2 + 15n) \quad (10)$$

Параллельный алгоритм

Выполнение итераций метода осуществляется последовательно, так как каждая следующая итерация уточняет решение, полученное на предыдущей итерации.

Анализ последовательного алгоритма позволяет сделать следующие выводы:

1. Основные вычисления метода приходятся на умножение Ap_k и $A^T\tilde{p}_k$. При использовании алгоритмов параллельного умножения матрицы на вектор, можно повысить эффективность выполнения итерации исходного алгоритма.
2. Дополнительные вычисления по обработке векторов (скалярное произведение, сложение и вычитание, умножение на скаляр, взятие нормы) так же могут быть выполнены параллельно в силу наличия независимых шагов.

Поэтому эффективность последовательного алгоритма может быть повышена за счет применения параллельных вычислений в ходе выполнения отдельных итераций.

Реализация

В данной задаче используются разреженные матрицы. Для описания матрицы необходимо знать:

- n – число строк в матрице
- m – число столбцов в матрице
- nz – число ненулевых элементов
- Позиции элементов.

Существует несколько способов представления таких матриц в памяти компьютера. В данной лабораторной работе будет использован разреженный строчный формат (CRS – Compressed Row Storage). Данный способ хранения (Листинг 1) требует разделения матрицы на 3 массива:

1. Массив значений value, который содержит ненулевые значения матрицы, записанные построчно сверху вниз.
2. Массив номеров столбцов colIndex.
3. Массив индексов начала строк rowPtr.

Способ требует объем памяти равный

$$\mathrm{M} = 8NZ + 4\mathrm{NZ} + 4(\mathrm{N}+1) = 12NZ + 4N + 4 \qquad (11)$$

```
struct CRSMatrix
{
 int n; // Число строк в матрице
 int m; // Число столбцов в матрице
 int nz; // Число ненулевых элементов в разреженной матрице
 vector val; // Массив значений матрицы по строкам
 vector colIndex; // Массив номеров столбцов
 vector rowPtr; // Массив индексов начала строк
};
```

Листинг 1. Структура хранения разреженной матрицы

```
//Умножение матрицы на вектора
void Multiplication(CRSMatrix& A, double* v, double* res) {
#pragma omp parallel for
    for (int i = 0; i < A.n; i++) {
        res[i] = 0.0;
        for (int j = A.rowPtr[i]; j < A.rowPtr[i + 1]; j++) {
            res[i] += A.val[j] * v[A.colIndex[j]];
        }
    }
}

//Вычисление скалярного произведения векторов
double Scalar(double* a, double* b, int & size) {
    double res = 0;
#pragma omp parallel for reduction(+: res)
    for (int i = 0; i < size; i++) {
        res += a[i] * b[i];
    }
    return res;
}

//Вычисление коэффициента альфа
void GetAlpha(double* r, double* r_, CRSMatrix& A, double* p,
double* p_, double* tmp, double &alpha) {
    Multiplication(A, p, tmp);

    alpha = Scalar(r, r_, A.n) / Scalar(tmp, p_, A.n);
}

//Вычисление коэффициента бетта
void GetBeta(double* r, double* r_, double* r_prev, double*
r_prev_, double& beta, int size) {
    beta = Scalar(r, r_, size) / Scalar(r_prev, r_prev_,
size);
}

//Сложение векторов с некоторым коэффициентом
void Addition(double* l, double* r, double* res, double coef,
int size) {
#pragma omp parallel for
    for (int i = 0; i < size; i++) {
        res[i] = l[i] + coef * r[i];
    }
}

//Вычисление нормы вектора
double Norma(double* a, int size) {
    double sum = 0.0;
#pragma omp parallel for reduction(+: sum)
    for (int i = 0; i < size; i++) {
        sum += a[i] * a[i];
    }
    return sqrt(sum);
}
```

Листинг 2. Дополнительные операции

Внутри одной итерации необходимо несколько раз просчитывать операции с матрицами и векторами, поэтому имеет смысл вынести их в отдельные функции (Листинг 2).

В функции SLE_Solver_CRS_BICG реализовано основное тело алгоритма с тремя критериями остановки: по максимальному числу итераций и двум критериям точности (Листинг 3).

```
void SLE_Solver_CRS_BICG(CRSMatrix& A, double* b, double eps,
int max_iter, double* x, int& count) {
    int n = A.n;

    double* r = new double[n];
    double* r_prev = new double[n];
    double* r_ = new double[n];
    double* r_prev_ = new double[n];
    double* p = new double[n];
    double* p_ = new double[n];
    double* tmp = new double[n];
    double alpha, beta;

    GenerateStartSolution(n, x);
    GetR0(A, x, b, r);
    CopyArray(r, r_, n);
    CopyArray(r, p, n);
    CopyArray(r, p_, n);

    CRSMatrix AT = Transponse(A);

    while (true) {
        count++;

        GetAlpha(r, r_, A, p, p_, tmp, alpha);
        Addition(x, p, x, alpha, n);

        Multiplication(A, p, tmp);
        CopyArray(r, r_prev, n);
        Addition(r, tmp, r, -alpha, A.n);

        Multiplication(AT, p_, tmp);
        CopyArray(r_, r_prev_, n);
        Addition(r_, tmp, r_, -alpha, n);

        GetBeta(r, r_, r_prev, r_prev_, beta, n);

        if (abs(beta) < 1e-10) {
            break;
        }
        if (Norma(r, n) < eps) {
            break;
        }
        if (count >= max_iter) {
            break;
        }
        Addition(r, p, p, beta, n);
        Addition(r_, p_, p_, beta, n);
    }

    delete[] r;
    delete[] r_prev;
    delete[] r_;
    delete[] r_prev_;
    delete[] p;
    delete[] p_;
```

Листинг 3. Функция решения СЛАУ методом бисопряженных градиентов

Результаты

Тестовая инфраструктура

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

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

Таблица 1. Тестовая инфраструктура

Результаты

Для проведения эксперимента была сгенерирована матрица размера $n = 20\,000$ и общим количеством ненулевых элементов $nz = 2 * 10^6$. Точность алгоритма $eps = 10^{-6}$ и максимальное число итераций 1000.

Наилучшим результатом является запуск на 4 потоках, что объясняется наличием 4 физических ядер (Рисунок 1).

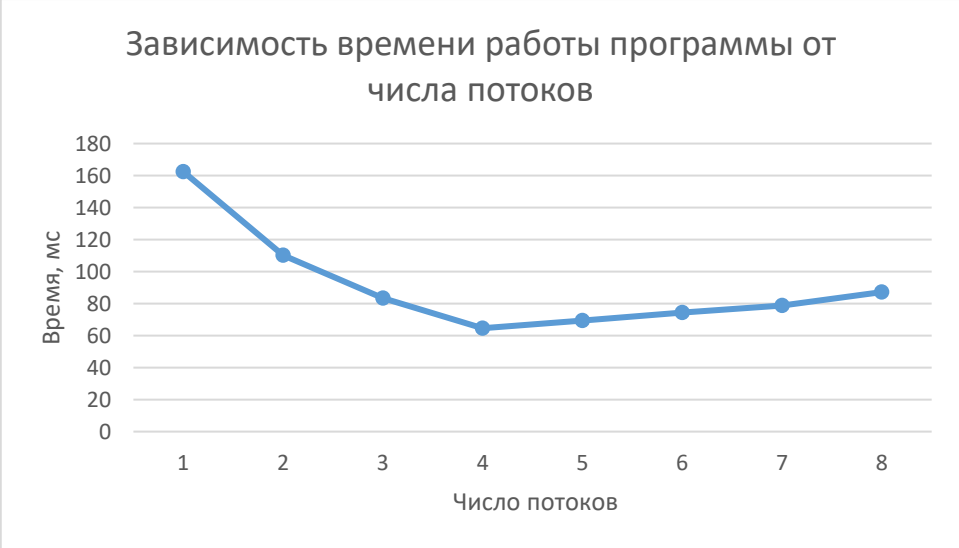

Рисунок 1. Зависимость времени работы программы от числа потоков

	1	2	3	4	5	6	7	8
Time, ms	162,447	110,221	83,2974	64,6079	69,3849	74,457	78,8757	87,2545
Speed up	1	1,47383	1,950205	2,514352	2,341244	2,181756	2,059532	1,861761

Таблица 2. Время работы программы и ускорение

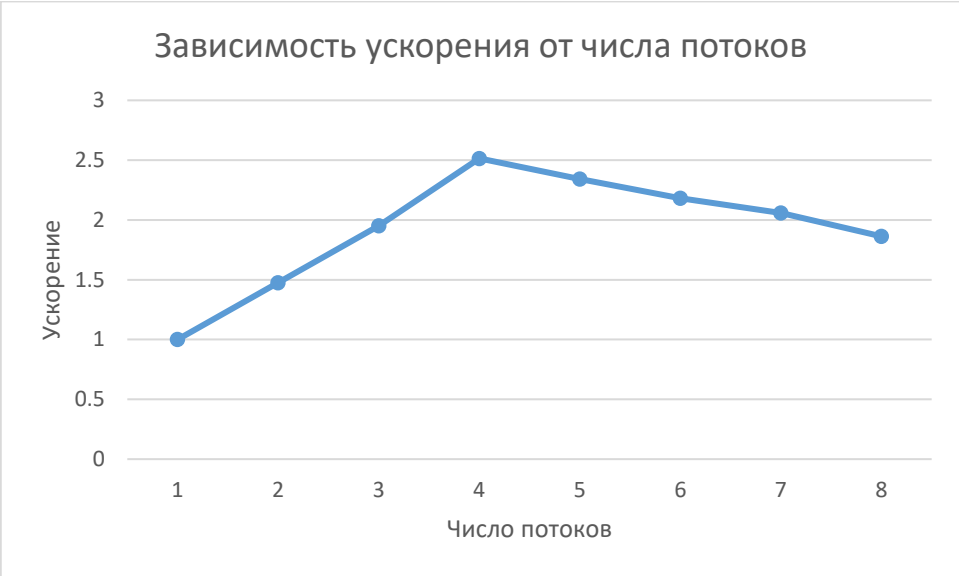

Рисунок 2. Зависимость ускорения от числа потоков

Заключение

В ходе данной лабораторной работы с помощью языка C++ и технологии OpenMP был реализован метод бисопряженных градиентов для решения СЛАУ с разреженной матрицей A и плотными векторами x и b.

Был изучен теоретический материал задачи, анализ трудоемкости вычислений и способ распараллеливания.

Р результате анализа производительности было отмечено ее повышение при использовании параллельных вычислений, что говорит об эффективности применяемой схемы распараллеливания

Литература

1. Баркалов К.А. Образовательный комплекс «Параллельные численные методы». - Н.Новгород, Изд-во ННГУ 2011
2. Кобельков Г.М. «Численные методы. Часть 2» - Москва. Мехмат МГУ
3. Белов С.А., Золотых Н.Ю. Численные методы линейной алгебры. – Н.Новгород, Изд-во ННГУ, 2005.
4. Писсанецки С. Технология разрежённых матриц = Sparse Matrix Technology. — М.: Мир, 1988. — 410 с
5. Джордж А., Лю Дж. Численное решение больших разрежённых систем уравнений = Computer Solution of Large Sparse Positive Definite Systems. — М.: Мир, 1984. — 333 с.