МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
**«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИМЕНИ Н. Г. ЧЕРНЫШЕВСКОГО»**

ПАРАЛЛЕЛЬНОЕ И РАСПРЕДЕЛЕННОЕ ПРОГРАММИРОВАНИЕ. WORK06

ОТЧЕТ О ПРАКТИКЕ

студента 3 курса 311 группы
направления 02.03.02 — Фундаментальная информатика и информационные технологии
факультета КНиИТ
Вильцева Данила Денисовича

Проверил
Старший преподаватель _____ М. С. Портенко

Саратов 2023

СОДЕРЖАНИЕ

1 Work 6

1.1 Условие задачи

Выполните разработку параллельного варианта для одного из итерационных методов:

3. верхней релаксации.

Для тестовой матрицы из нулей и единиц проведите вычислительные эксперименты, результаты занесите в таблицу 1.

Таблица 1. Время выполнения последовательного и параллельного итерационного алгоритмов решения систем линейных уравнений и ускорение

Номер теста	Порядок системы	Последовательный алгоритм	Параллельный алгоритм	
			Время	Ускорение
1	10			
2	100			
3	500			
4	1000			
5	1500			
6	2000			
7	2500			
8	3000			

Какой из алгоритмов Гаусса или итерационный обладает лучшими показателями ускорения? Заполните таблицу 2.

Таблица 2. Ускорение параллельных алгоритмов Гаусса и итерационного (вариант) решения систем линейных уравнений

Номер теста	Порядок системы	Ускорение алгоритма Гаусса	Ускорение итерационного алгоритма (вариант)
1	10		
2	100		
3	500		
4	1000		
5	1500		
6	2000		
7	2500		
8	3000		

1.2 Решение

Последовательная реализация метода верхней релаксации

Рассмотрим подход к решению систем линейных уравнений $Ax = b$ с невырожденной квадратной матрицей, при котором используя заданное начальное приближение x^0 строится последовательность приближенных решений $x^0, x^1, \dots x^k, \dots$ до тех пор пока приближенное решение не будет найдено с требуемой точностью.

Итерации заканчиваются, когда:

- норма невязки $\|b - Ax^k\| = \max_{1 \le i \le n} \left| b_i - \sum_{j=1}^{n} a_{ij} x_j^k \right| < \varepsilon$ не станет малой;
- погрешность определения компонент решения $\|x^{k+1} - x^k\| < \varepsilon$, где через $\| \ \|$ обозначена любая векторная норма, не станет малой;
- достигнуто максимальное число итераций N, на которое готов пойти исследователь;
- перечисленные критерии могут совмещаться.

Метод Зейделя

Покомпонентная форма записи метода Зейделя имеет вид:

$$x_i^{k+1} = \frac{b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{k+1} - \sum_{j=i+1}^{n} a_{ij} x_j^k}{a_{ii}}, i = 1,2, \dots n; k = 0,1, \dots..$$

Метод верхней релаксации

Модификация метода Зейделя:

$$x_i^{k+1} = (1 - \omega) x_i^k + \omega \frac{b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{k+1} - \sum_{j=i+1}^{n} a_{ij} x_j^k}{a_{ii}}, i = 1,2, \dots n; k = 0,1, \dots..$$

где ω – параметр релаксации: $0 < \omega < 2$, при $\omega = 1$ сводится к итерациям метода Зейделя, при $1 < \omega < 2$ – ***метод верхней релаксации***.

1.3 Фрагмент кода (последовательная реализация)

```
#include <omp.h>
#include <iostream>
#include <time.h>
#include <cmath>
#include <windows.h>
#include<cstdlib>
using namespace std;
// Function that converts numbers form LongInt type to
// double type
double LiToDouble(LARGE_INTEGER x) {
        double result = ((double)x.HighPart) * 4.294967296E9 +
                (double)((x).LowPart);
        return result;
}
```

```
// Function that gets the timestamp in seconds
double GetTime() {
        LARGE_INTEGER lpFrequency, lpPerfomanceCount;
        QueryPerformanceFrequency(&lpFrequency);
        QueryPerformanceCounter(&lpPerfomanceCount);
        return LiToDouble(lpPerfomanceCount) / LiToDouble(lpFrequency);
}

double* upper_relaxation_method(double** a, double* b, int n, double eps, double w, double*
↪  x, double* xn) {
        int i, j, k = 0;
        double norma;

        for (i = 0; i < n; i++)
        {
                xn[i] = 0;
                x[i] = xn[i];
        }
        do
        {
                k++;
                norma = 0;

                for (i = 0; i < n; i++)
                {
                        x[i] = b[i];
                        for (j = 0; j < n; j++)
                        {
                                if (i != j)
                                        x[i] = x[i] - a[i][j] * x[j];
                        }
                        x[i] /= a[i][i];

                        x[i] = w * x[i] + (1 - w) * xn[i];

                        if (fabs(x[i] - xn[i]) > norma)
                                norma = fabs(x[i] - xn[i]);
                        xn[i] = x[i];
                }
        } while (norma > eps);

        return x;

}

double experiment(double* res, double** a, double* b, int n, double eps, double w, double*
↪  x, double* xn)
{
```

```
        double stime, ftime; // время начала и конца расчета
        stime = GetTime();
        upper_relaxation_method(a, b, n, eps, w, x, xn); // вызов функции интегрирования
        ftime = GetTime();
        return (ftime - stime) / CLOCKS_PER_SEC;
}


int main()
{
        setlocale(LC_CTYPE, "RUSSIAN");
        int n;
        double eps;
        double w;

        cout << "Введите размерность матрицы N*N:";
        cin >> n;
        double** a = new double* [n];
        for (int i = 0; i < n; i++)
                a[i] = new double[n];


        double* b = new double[n];
        double* x = new double[n];
        double* xn = new double[n];

        for (int i = 0; i < n; i++)
        {
                for (int j = 0; j < n; j++)
                {

                        a[i][j] = rand() / double(1000);
                }
        }

        for (int i = 0; i < n; i++)
        {

                b[i] = rand() / double(1000);
        }

        eps = 0.001;
        w = 1.12;

        double time; // время проведенного эксперимента
        double res; // значение вычисленного интеграла
        double min_time; // минимальное время работы
                                        // реализации алгоритма
```

```
        double max_time; // максимальное время работы
                                        // реализации алгоритма
        double avg_time; // среднее время работы
                                        // реализации алгоритма
        int numbExp = 10; // количество запусков программы

        // первый запуск
        min_time = max_time = avg_time = experiment(&res, a, b, n, eps, w, x, xn);
        // оставшиеся запуски
        for (int i = 0; i < numbExp - 1; i++)
        {
                time = experiment(&res, a, b, n, eps, w, x, xn);
                avg_time += time;
                if (max_time < time) max_time = time;
                if (min_time > time) min_time = time;
        }
        // вывод результатов эксперимента
        cout << "execution time : " << avg_time / numbExp << "; " <<
                min_time << "; " << max_time << endl;
        cout.precision(8);
}
```

Параллельная реализация метода верхней релаксации

Используем директиву pragma omp parallel for, которая создаст несколько вычислительных потоков и разделит итерации между ними.

1.4 Фрагмент кода (параллельная реализация)

```
#include <omp.h>
#include <iostream>
#include <time.h>
#include <cmath>
#include <windows.h>
#include<cstdlib>
using namespace std;
// Function that converts numbers form LongInt type to
// double type
double LiToDouble(LARGE_INTEGER x) {
        double result = ((double)x.HighPart) * 4.294967296E9 +
                (double)((x).LowPart);
        return result;
}

// Function that gets the timestamp in seconds
double GetTime() {
        LARGE_INTEGER lpFrequency, lpPerfomanceCount;
        QueryPerformanceFrequency(&lpFrequency);
        QueryPerformanceCounter(&lpPerfomanceCount);
```

```
        return LiToDouble(lpPerfomanceCount) / LiToDouble(lpFrequency);
}

double* upper_relaxation_method(double** a, double* b, int n, double eps, double w, double*
↪  x, double* xn) {
        int i, j, k = 0;
        double norma;
#pragma omp parallel for
        for (i = 0; i < n; i++)
        {
                xn[i] = 0;
                x[i] = xn[i];
        }
        do
        {
                k++;
                norma = 0;

                for (i = 0; i < n; i++)
                {
                        x[i] = b[i];
#pragma omp parallel for
                        for (j = 0; j < n; j++)
                        {
                                if (i != j)
                                        x[i] = x[i] - a[i][j] * x[j];
                        }
                        x[i] /= a[i][i];

                        x[i] = w * x[i] + (1 - w) * xn[i];

                        if (fabs(x[i] - xn[i]) > norma)
                                norma = fabs(x[i] - xn[i]);
                        xn[i] = x[i];
                }
        } while (norma > eps);

        return x;

}

double experiment(double* res, double** a, double* b, int n, double eps, double w, double*
↪  x, double* xn)
{
        double stime, ftime; // время начала и конца расчета
        stime = GetTime();
        upper_relaxation_method(a, b, n, eps, w, x, xn); // вызов функции интегрирования
        ftime = GetTime();
```

```
        return (ftime - stime) / CLOCKS_PER_SEC;
}


int main()
{
        setlocale(LC_CTYPE, "RUSSIAN");
        int n;
        double eps;
        double w;

        cout << "Введите размерность матрицы N*N:";
        cin >> n;
        double** a = new double* [n];
        for (int i = 0; i < n; i++)
                a[i] = new double[n];


        double* b = new double[n];
        double* x = new double[n];
        double* xn = new double[n];

        for (int i = 0; i < n; i++)
        {
                for (int j = 0; j < n; j++)
                {

                        a[i][j] = rand() / double(1000);
                }
        }

        for (int i = 0; i < n; i++)
        {

                b[i] = rand() / double(1000);
        }

        eps = 0.001;
        w = 1.12;

        double time; // время проведенного эксперимента
        double res; // значение вычисленного интеграла
        double min_time; // минимальное время работы
                                        // реализации алгоритма
        double max_time; // максимальное время работы
                                        // реализации алгоритма
        double avg_time; // среднее время работы
                                        // реализации алгоритма
```

```
        int numbExp = 10; // количество запусков программы

        // первый запуск
        min_time = max_time = avg_time = experiment(&res, a, b, n, eps, w, x, xn);
        // оставшиеся запуски
        for (int i = 0; i < numbExp - 1; i++)
        {
                time = experiment(&res, a, b, n, eps, w, x, xn);
                avg_time += time;
                if (max_time < time) max_time = time;
                if (min_time > time) min_time = time;
        }
        // вывод результатов эксперимента
        cout << "execution time : " << avg_time / numbExp << "; " <<
                min_time << "; " << max_time << endl;
        cout.precision(8);
}
```

1.5 Результат работы программы

Последовательная реализация метода верхней релаксации

```
Введите размерность матрицы N*N:10
execution time : 4.70599e-08; 4.63985e-08; 5.23999e-08
```

```
Введите размерность матрицы N*N:100
execution time : 6.3934e-07; 5.47901e-07; 8.27601e-07
```

```
Введите размерность матрицы N*N:500
execution time : 3.39393e-06; 3.0939e-06; 4.1417e-06
```

```
Введите размерность матрицы N*N:1000
execution time : 1.05139e-05; 9.6889e-06; 1.21694e-05
```

```
Введите размерность матрицы N*N:1500
execution time : 2.32889e-05; 2.19749e-05; 2.50751e-05
```

```
Введите размерность матрицы N*N:2000
execution time : 2.66843e-05; 2.58876e-05; 2.81921e-05
```

```
Введите размерность матрицы N*N:2500
execution time : 4.23384e-05; 4.08226e-05; 4.38005e-05
```

```
Введите размерность матрицы N*N:3000
execution time : 5.99791e-05; 5.8297e-05; 6.16869e-05
```

Параллельная реализация метода верхней релаксации

```
Введите размерность матрицы N*N:10
execution time : 1.46298e-08; 1.45994e-08; 1.48993e-08
```

```
Введите размерность матрицы N*N:100
execution time : 1.8293e-07; 1.511e-07; 3.15702e-07
```

```
Введите размерность матрицы N*N:500
execution time : 1.2432e-06; 1.1267e-06; 1.6371e-06
```

```
Введите размерность матрицы N*N:1000
execution time : 3.96223e-06; 3.6279e-06; 5.3243e-06
```

```
Введите размерность матрицы N*N:1500
execution time : 8.88255e-06; 8.252e-06; 9.6961e-06
```

```
Введите размерность матрицы N*N:2000
execution time : 1.0432e-05; 9.9998e-06; 1.12144e-05
```

```
Введите размерность матрицы N*N:2500
execution time : 1.61646e-05; 1.53606e-05; 1.69935e-05
```

```
Введите размерность матрицы N*N:3000
execution time : 2.39651e-05; 2.24418e-05; 2.68569e-05
```

2 Вывод

Номер теста	Порядок системы	Последовательный алгоритм	Параллельный алгоритм	
			Время	Ускорение
1	10	4.70599e-08	1.46298e-08	3.219
2	100	6.3934e-07	1.8293e-07	3.510
3	500	3.39393e-06	1.2432e-06	2.733
4	1000	3.96223e-05	1.05139e-05	3.768
5	1500	3.32889e-05	1.24255e-05	2.677
6	2000	2.66843e-05	1.0432e-05	2.549
7	2500	4.23384e-05	1.61646e-05	2.619
8	3000	5.99791e-05	2.39651e-05	2.506

Номер теста	Порядок системы	Ускорение Гаусса	Ускорение верхней релаксации
1	10	0	3.219
2	100	0.2642	3.510
3	500	1.30684012	2.733
4	1000	1.79439	3.768
5	1500	2.0259	2.677
6	2000	2.08326	2.549
7	2500	2.09565	2.619
8	3000	2.1707	2.506

Метод верхней релаксации имеет преимущество перед методом Гаусса, что видно по значениям таблицы.

3 Характеристики компьютера