

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

Кафедра математической кибернетики и компьютерных наук

ОТЧЕТ ПО ПРАКТИКЕ WORK6

ОТЧЕТ

Студента 3 курса 311 группы
направления 02.03.02 — Фундаментальная информатика и информационные
технологии
факультета КНИИТ
Забоева Максима Владиславовича

Проверил

Старший преподаватель

М. С. Портенко

Саратов 2023

СОДЕРЖАНИЕ

1	Условие задачи	3
2	Практическая часть	4
2.1	Код программы	4
2.2	Таблицы результатов	7
2.2.1	Таблица запуска алгоритма Якоби	7
2.2.2	Таблица сравнения алгоритмов	7
3	Тестовые запуски	9
3.1	Последовательный алгоритм	9
3.2	Параллельный алгоритм	9

1 Условие задачи

Выполните разработку параллельного варианта для одного из итерационных методов: **Якоби**.

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		

2 Практическая часть

2.1 Код программы

Код программы:

```
#include <iostream>
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <time.h>
#include <math.h>
#include <iostream>
#include <omp.h>

using namespace std;
int* pPivotPos; // The number of pivot rows selected at the iterations
int* pPivotIter; // The iterations, at which the rows were pivots
typedef struct {
    int PivotRow;
    double MaxValue;
} TThreadPivotRow;
// Finding the pivot row
int ParallelFindPivotRow(double* pMatrix, int Size, int Iter) {
    int PivotRow = -1; // The index of the pivot row
    double MaxValue = 0; // The value of the pivot element
    int i; // Loop variable
    // Choose the row, that stores the maximum element
#pragma omp parallel
    {
        TThreadPivotRow ThreadPivotRow;
        ThreadPivotRow.MaxValue = 0;
        ThreadPivotRow.PivotRow = -1;
#pragma omp for
        for (i = 0; i < Size; i++) {
            if ((pPivotIter[i] == -1) &&
                (fabs(pMatrix[i * Size + Iter]) > ThreadPivotRow.MaxValue))
            {
                ThreadPivotRow.PivotRow = i;
                ThreadPivotRow.MaxValue = fabs(pMatrix[i * Size + Iter]);
            }
        }
#pragma omp critical
        {
            if (ThreadPivotRow.MaxValue > MaxValue) {
                MaxValue = ThreadPivotRow.MaxValue;
                PivotRow = ThreadPivotRow.PivotRow;
            }
        }
    }
    return PivotRow;
}
// Column elimination
void ParallelColumnElimination(double* pMatrix, double* pVector,
    int Pivot, int Iter, int Size) {
    double PivotValue, PivotFactor;
    PivotValue = pMatrix[Pivot * Size + Iter];
#pragma omp parallel for private(PivotFactor) schedule(dynamic,1)
    for (int i = 0; i < Size; i++) {
        if (pPivotIter[i] == -1) {
            PivotFactor = pMatrix[i * Size + Iter] / PivotValue;
            for (int j = Iter; j < Size; j++) {
```

```

        pMatrix[i * Size + j] -= PivotFactor * pMatrix[Pivot * Size + j];
    }
    pVector[i] -= PivotFactor * pVector[Pivot];
}

}

// Gaussian elimination
void ParallelGaussianElimination(double* pMatrix, double* pVector,
    int Size) {
    int Iter; // The number of the iteration of the Gaussian
    // elimination
    int PivotRow; // The number of the current pivot row
    for (Iter = 0; Iter < Size; Iter++) {
        // Finding the pivot row
        PivotRow = ParallelFindPivotRow(pMatrix, Size, Iter);
        pPivotPos[Iter] = PivotRow;
        pPivotIter[PivotRow] = Iter;
        ParallelColumnElimination(pMatrix, pVector, PivotRow, Iter, Size);
    }
}

void DummyDataInitialization(double* pMatrix, double* pVector, int Size) {
    int i, j;
    for (i = 0; i < Size; i++) {
        pVector[i] = i + 1;
        for (j = 0; j < Size; j++) {
            if (j == i)
                pMatrix[i * Size + j] = 1;
            else
                pMatrix[i * Size + j] = 0;
        }
    }
}

// Function for memory allocation and definition of the objects elements
void ProcessInitialization(double*& pMatrix, double*& pVector, double*& pResult, int& Size)
{
    // Setting the size of the matrix and the vector
    do {
        printf("\nEnter size of the matrix and the vector: ");
        scanf_s("%d", &Size);
        printf("\nChosen size = %d \n", Size);
        if (Size <= 0)
            printf("\nSize of objects must be greater than 0!\n");
    } while (Size <= 0);
    // Memory allocation
    pMatrix = new double[Size * Size];
    pVector = new double[Size];
    pResult = new double[Size];
    // Initialization of the matrix and the vector elements
    //RandomDataInitialization(pMatrix, pVector, Size);
    DummyDataInitialization(pMatrix, pVector, Size);
    //RandomDataInitialization(pMatrix, pVector, Size);
}

// Function for computational process termination
void ProcessTermination(double* pMatrix, double* pVector, double*
    pResult) {
    delete[] pMatrix;
    delete[] pVector;
    delete[] pResult;
}

void PrintMatrix(double* pMatrix, int RowCount, int ColCount) {

```

```

    int i, j; // Loop variables
    for (i = 0; i < RowCount; i++) {
        for (j = 0; j < ColCount; j++)
            printf("%7.4f ", pMatrix[i * RowCount + j]);
        printf("\n");
    }
}

// Function for formatted vector output
void PrintVector(double* pVector, int Size) {
    int i;
    for (i = 0; i < Size; i++)
        printf("%7.4f ", pVector[i]);
}

const double eps = 0.001;
void Jacobi(double* pMatrix, double* pVector, double* pResult, int Size)
{
    double* TempX = new double[Size];
    double** A = new double* [Size];

    for (int i = 0; i < Size; i++) {
        A[i] = new double[Size];
    }
    int k = 0;

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

        for (int j = 0; j < Size; j++) {
            A[i][j] = pMatrix[k];
            k++;
        }
    }
    double norm;

    while (true) {
#pragma omp parallel for
        for (int i = 0; i < Size; i++) {
            TempX[i] = pVector[i];
#pragma omp parallel for
            for (int g = 0; g < Size; g++) {
                if (i != g)
                    TempX[i] -= A[i][g] * pResult[g];
            }
            TempX[i] /= A[i][i];
        }
        norm = fabs(pResult[0] - TempX[0]);
#pragma omp parallel for
        for (int h = 0; h < Size; h++) {
            if (fabs(pResult[h] - TempX[h]) > norm)
                norm = fabs(pResult[h] - TempX[h]);
            pResult[h] = TempX[h];
        }
        if (norm <= eps) {
            break;
        }
    }
    delete[] TempX;
}

int main() {
    double* pMatrix; // The matrix of the linear system
    double* pVector; // The right parts of the linear system
    double* pResult; // The result vector

```

```

int Size; // The size of the matrix and the vectors
double start, finish, duration;
// Data initialization

ProcessInitialization(pMatrix, pVector, pResult, Size);
//PrintMatrix(pMatrix, Size, Size);
//PrintMatrix(pMatrix, Size, Size);
//PrintVector(pVector, Size);
start = omp_get_wtime();
//ParallelResultCalculation(pMatrix, pVector, pResult, Size);
Jacobi(pMatrix, pVector, pResult, Size);
finish = omp_get_wtime();
duration = finish - start;
// Testing the result
//PrintVector(pVector, Size);
//TestResult(pMatrix, pVector, pResult, Size);
// Printing the time spent by parallel Gauss algorithm
printf("\n Time of execution: %f\n", duration);
// Program termination
ProcessTermination(pMatrix, pVector, pResult);
return 0;
}

```

2.2 Таблицы результатов

2.2.1 Таблица запуска алгоритма Якоби

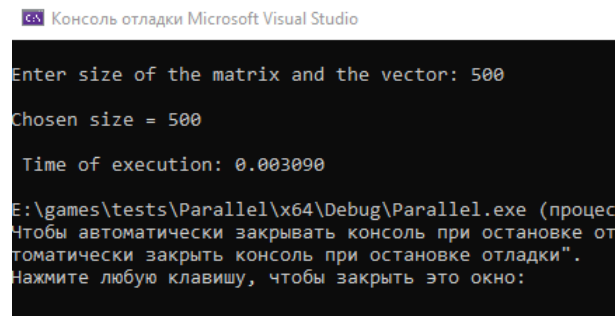
Номер теста	Порядок системы	Последовательный алгоритм	Параллельный алгоритм	
			Время	Ускорение
1	10	0.000029	0.003112	0
2	100	0.000128	0.006956	0,0184013801
3	500	0.003090	0.006951	0,4445403539
4	1000	0.017806	0.013517	1,317304136
5	1500	0.026418	0.015364	1,719474095
6	2000	0.036748	0.030761	1,194629563
7	2500	0.051318	0.052195	0,9831976243
8	3000	0.065885	0.056067	1,17511192

2.2.2 Таблица сравнения алгоритмов

Номер теста	Порядок системы	Последовательный алгоритм	Ускорение итерационного алгоритма Якоби
1	10	0	0
2	100	0,08954956568	0,0184013801
3	500	1,195362569	0,4445403539
4	1000	3,284408214	1,317304136
5	1500	5,3015206	1,719474095
6	2000	5,661290283	1,194629563
7	2500	5,290039695	0,9831976243
8	3000	4,779076634	1,17511192

3 Тестовые запуски

3.1 Последовательный алгоритм



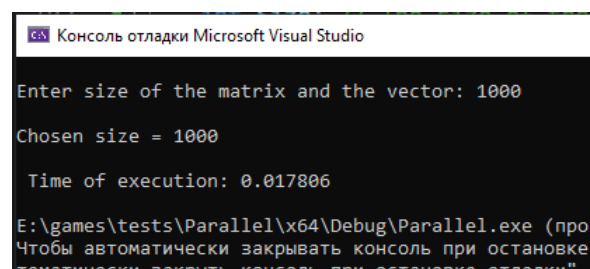
```
Консоль отладки Microsoft Visual Studio

Enter size of the matrix and the vector: 500

Chosen size = 500

Time of execution: 0.003090

E:\games\tests\Parallel\x64\Debug\Parallel.exe (процес
Чтобы автоматически закрывать консоль при остановке от
томатически закрыть консоль при остановке отладки".
Нажмите любую клавишу, чтобы закрыть это окно:
```



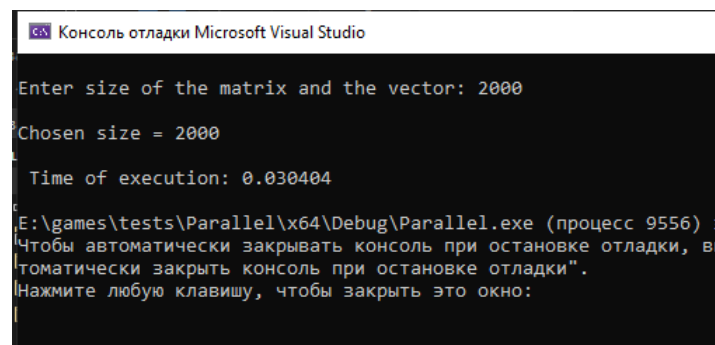
```
Консоль отладки Microsoft Visual Studio

Enter size of the matrix and the vector: 1000

Chosen size = 1000

Time of execution: 0.017806

E:\games\tests\Parallel\x64\Debug\Parallel.exe (про
Чтобы автоматически закрывать консоль при остановке
томатически закрыть консоль при остановке отладки"
```



```
Консоль отладки Microsoft Visual Studio

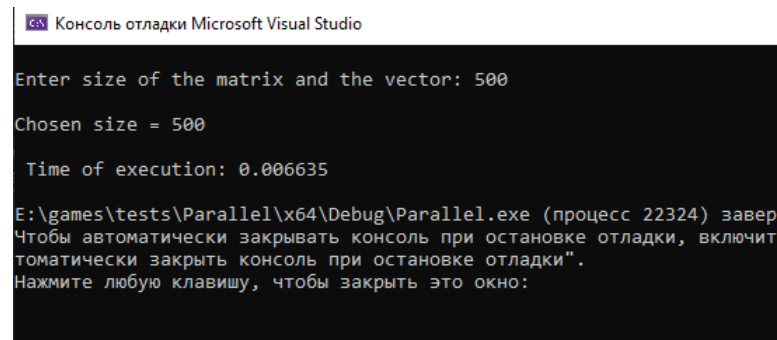
Enter size of the matrix and the vector: 2000

Chosen size = 2000

Time of execution: 0.030404

E:\games\tests\Parallel\x64\Debug\Parallel.exe (процесс 9556) в
Чтобы автоматически закрывать консоль при остановке отладки, вк
томатически закрыть консоль при остановке отладки".
Нажмите любую клавишу, чтобы закрыть это окно:
```

3.2 Параллельный алгоритм



```
Консоль отладки Microsoft Visual Studio

Enter size of the matrix and the vector: 500

Chosen size = 500

Time of execution: 0.006635

E:\games\tests\Parallel\x64\Debug\Parallel.exe (процесс 22324) завер
Чтобы автоматически закрывать консоль при остановке отладки, включит
томатически закрыть консоль при остановке отладки".
Нажмите любую клавишу, чтобы закрыть это окно:
```

```
Enter size of the matrix and the vector: 1000  
Chosen size = 1000  
  
Time of execution: 0.013517  
  
E:\games\tests\Parallel\x64\Debug\Parallel.exe (процесс 11804) заверш  
Чтобы автоматически закрывать консоль при остановке отладки, включите  
томатически закрыть консоль при остановке отладки".  
Нажмите любую клавишу, чтобы закрыть это окно:
```

```
Enter size of the matrix and the vector: 2000  
Chosen size = 2000  
  
Time of execution: 0.030761  
  
E:\games\tests\Parallel\x64\Debug\Parallel.exe (процесс 28080) заве  
Чтобы автоматически закрывать консоль при остановке отладки, включ  
томатически закрыть консоль при остановке отладки".  
Нажмите любую клавишу, чтобы закрыть это окно:
```