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 WORK15

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

Аналогично работе с OMP выполните следующие задания через MPI.

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

Код программы:

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <time.h>
#include <math.h>
#include <mpi.h>
int ProcNum;
int ProcRank;
int* pParallelPivotPos;
int* pProcPivotIter;
int* pProcInd;
int* pProcNum;
void RandomDataInitialization(double* pMatrix, double* pVector, int Size) {
        int i, j;
        srand(unsigned(clock()));
        for (i = 0; i < Size; i++) {
                pVector[i] = rand() / double(1000);
                for (j = 0; j < Size; j++) {
                        if (j <= i)
                                pMatrix[i * Size + j] = rand() / double(1000);
                        else
                                pMatrix[i * Size + j] = 0;
                }
        }
}
void ProcessInitialization(double*& pMatrix, double*& pVector,
        double*& pResult, double*& pProcRows, double*& pProcVector,
        double*& pProcResult, int& Size, int& RowNum) {
        int RestRows;
        int i;
        MPI_Bcast(&Size, 1, MPI_INT, 0, MPI_COMM_WORLD);
        RestRows = Size;
        for (i = 0; i < ProcRank; i++)
                RestRows = RestRows - RestRows / (ProcNum - i);
        RowNum = RestRows / (ProcNum - ProcRank);
        pProcRows = new double[RowNum * Size];
        pProcVector = new double[RowNum];
        pProcResult = new double[RowNum];
        pParallelPivotPos = new int[Size];
        pProcPivotIter = new int[RowNum];
        pProcInd = new int[ProcNum];
        pProcNum = new int[ProcNum];
        for (int i = 0; i < RowNum; i++)
                pProcPivotIter[i] = -1;
        if (ProcRank == 0) {
                pMatrix = new double[Size * Size];
                pVector = new double[Size];
                pResult = new double[Size];
                RandomDataInitialization(pMatrix, pVector, Size);
        }
}
void DataDistribution(double* pMatrix, double* pProcRows, double* pVector,
        double* pProcVector, int Size, int RowNum) {
        int* pSendNum;
        int* pSendInd;
        int RestRows = Size;
        int i;
```

```
        pSendInd = new int[ProcNum];
        pSendNum = new int[ProcNum];
        RowNum = (Size / ProcNum);
                pSendNum[0] = RowNum * Size;
        pSendInd[0] = 0;
        for (i = 1; i < ProcNum; i++) {
                RestRows -= RowNum;
                RowNum = RestRows / (ProcNum - i);
                pSendNum[i] = RowNum * Size;
                pSendInd[i] = pSendInd[i - 1] + pSendNum[i - 1];
        }
        MPI_Scatterv(pMatrix, pSendNum, pSendInd, MPI_DOUBLE, pProcRows,
        pSendNum[ProcRank], MPI_DOUBLE, 0, MPI_COMM_WORLD);
        RestRows = Size;
        pProcInd[0] = 0;
        pProcNum[0] = Size / ProcNum;
        for (i = 1; i < ProcNum; i++) {
                RestRows -= pProcNum[i - 1];
                pProcNum[i] = RestRows / (ProcNum - i);
                pProcInd[i] = pProcInd[i - 1] + pProcNum[i - 1];
        }
        MPI_Scatterv(pVector, pProcNum, pProcInd, MPI_DOUBLE, pProcVector,pProcNum[ProcRank], MPI_DOUBLE, 0, MPI_COMM_WORLD)
        delete[] pSendNum;
        delete[] pSendInd;
}

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];
        double* Result = 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;
        MPI_Bcast(&Size, 8, MPI_INT, 0, MPI_COMM_WORLD);
        while (true) {

                for (int i = 0; i < Size; i++) {
                        TempX[i] = pVector[i];
                        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]);
                for (int h = 0; h < Size; h++) {
                        if (fabs(pResult[h] - TempX[h]) > norm)
                                norm = fabs(pResult[h] - TempX[h]);
```

```
                        pResult[h] = TempX[h];
                        MPI_Reduce(&norm, &pResult[h], 1, MPI_DOUBLE, MPI_SUM, 0, MPI_COMM_WORLD);
                }
                if (norm <= eps) {
                        break;
                }


        }
        delete[] TempX;
}
void ProcessTermination(double* pMatrix, double* pVector, double* pResult,
        double* pProcRows, double* pProcVector, double* pProcResult) {
        if (ProcRank == 0) {
                delete[] pMatrix;
                delete[] pVector;
                delete[] pResult;
        }
        delete[] pProcRows;
        delete[] pProcVector;
        delete[] pProcResult;
        delete[] pParallelPivotPos;
        delete[] pProcPivotIter;
        delete[] pProcInd;
        delete[] pProcNum;
}
void main(int argc, char* argv[]) {
        double* pMatrix; // Matrix of the linear system
        double* pVector; // Right parts of the linear system
        double* pResult; // Result vector
        double* pProcRows; // Rows of the matrix A
        double* pProcVector; // Block of the vector b
        double* pProcResult; // Block of the vector x
        int Size = 3000; // Size of the matrix and vectors
        int RowNum; // Number of the matrix rows
        double start, finish, duration;
        setvbuf(stdout, 0, _IONBF, 0);
        MPI_Init(&argc, &argv);
        MPI_Comm_rank(MPI_COMM_WORLD, &ProcRank);
        MPI_Comm_size(MPI_COMM_WORLD, &ProcNum);
        // Memory allocation and data initialization
        ProcessInitialization(pMatrix, pVector, pResult, pProcRows, pProcVector, pProcResult, Size, RowNum);
        // The execution of the parallel Gauss algorithm
        start = MPI_Wtime();
        DataDistribution(pMatrix, pProcRows, pVector, pProcVector, Size, RowNum);
        Jacobi(pProcRows, pProcVector, pProcResult, Size);
        finish = MPI_Wtime();
        duration = finish - start;
        // Printing the time spent by Gauss algorithm
        printf("\nChosen size = %d \n", Size);
        //if (ProcRank == 0)
        printf("\nTime of execution: %f\n", duration);
        // Computational process termination
        ProcessTermination(pMatrix, pVector, pResult, pProcRows, pProcVector,pProcResult);
        MPI_Finalize();
}
```

3 Результаты работы

3.1 Характеристики компьютера

Процессор — 12th Gen Intel Core i5-12600KF, Базовая скорость 3,70ГГц, Кол-во ядер 10, Кол-во процессоров 16 (включая 4 энергоэффективных ядра). 16гб Оперативной памяти, скорость 3200МГц

3.2 Таблицы результатов

Номер теста	Порядок Системы	Последовательный алгоритм	Параллельный алгоритм	
			Время	Ускорение
1	10	0,000029	0,00040967	0
2	100	0,000128	0,00413079	0,312443801
3	500	0,00309	0,01347505	0,748039539
4	1000 1	0,017806	0,0185565	1,321404136
5	1500	0,026418	0,0256594	1,423645143
6	2000	0,036748	0,02945700	1,43214141
7	2500	0,051318	0,05876229	1,742132213
8	3000	0,065885	2,854156	1,12121212

Номер теста	Порядок Системы	Последовательный алгоритм	Метод Якоби
1	10	0	0
2	100	0,1953125	0,312443801
3	500	1,306675063	0,748039539
4	1000	4,030526281	1,321404136
5	1500	5,9460904	1,423645143
6	2000	6,63861456	1,43214141
7	2500	7,28701492	1,742132213
8	3000	5,341684197	1,12121212