

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 WORK13

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
3	Результаты работы	8
3.1	Характеристики компьютера	8
3.2	Фото результатов	8

1 Условие задачи

Аналогично работе с OMP выполните следующее задание через MPI.

Решите систему линейных уравнений согласно варианту параллельным методом Гаусса.

Вариант 2.

$$\begin{aligned} & \begin{matrix} 1 & 2 & 3 & 4 & 5 \end{matrix} \\ & \begin{matrix} 1 & 2 & 3 & 4 & 5 \end{matrix} \end{aligned} \quad \begin{matrix} 2. \\ 3. \end{matrix}$$
$$\begin{aligned} x_1 + 2x_2 + 3x_3 + 4x_4 + 5x_5 &= 2, \\ 2x_1 + 3x_2 + 7x_3 + 10x_4 + 13x_5 &= 12, \\ 3x_1 + 5x_2 + 11x_3 + 16x_4 + 21x_5 &= 17, \\ 2x_1 - 7x_2 + 7x_3 + 7x_4 + 2x_5 &= 57, \\ x_1 + 4x_2 + 5x_3 + 3x_4 + 10x_5 &= 7. \end{aligned}$$

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 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;
}

void ParallelEliminateColumns(double* pProcRows, double* pProcVector,
    double* pPivotRow, int Size, int RowNum, int Iter) {
    double multiplier;
    for (int i = 0; i < RowNum; i++) {
        if (pProcPivotIter[i] == -1) {
            multiplier = pProcRows[i * Size + Iter] / pPivotRow[Iter];
            for (int j = Iter; j < Size; j++) {
                pProcRows[i * Size + j] -= pPivotRow[j] * multiplier;
            }
            pProcVector[i] -= pPivotRow[Size] * multiplier;
        }
    }
}

// Function for the Gaussian elimination
void ParallelGaussianElimination(double* pProcRows, double* pProcVector,
    int Size, int RowNum) {
    double MaxValue;
    int PivotPos;
    struct { double MaxValue; int ProcRank; } ProcPivot, Pivot;
    double* pPivotRow = new double[Size + 1];
    for (int i = 0; i < Size; i++) {
        double MaxValue = 0;
        for (int j = 0; j < RowNum; j++) {
            if ((pProcPivotIter[j] == -1) &&
                (MaxValue < fabs(pProcRows[j * Size + i]))) {
                MaxValue = fabs(pProcRows[j * Size + i]);
                PivotPos = j;
            }
        }
        ProcPivot.MaxValue = MaxValue;
        ProcPivot.ProcRank = ProcRank;
        MPI_Allreduce(&ProcPivot, &Pivot, 1, MPI_DOUBLE_INT, MPI_MAXLOC, MPI_COMM_WORLD);
        if (ProcRank == Pivot.ProcRank) {
            pProcPivotIter[PivotPos] = i;
            pParallelPivotPos[i] = pProcInd[ProcRank] + PivotPos;
        }
        MPI_Bcast(&pParallelPivotPos[i], 1, MPI_INT, Pivot.ProcRank, MPI_COMM_WORLD);
        if (ProcRank == Pivot.ProcRank) {
            for (int j = 0; j < Size; j++) {
                pPivotRow[j] = pProcRows[PivotPos * Size + j];
            }
            pPivotRow[Size] = pProcVector[PivotPos];
        }
        MPI_Bcast(pPivotRow, Size + 1, MPI_DOUBLE, Pivot.ProcRank, MPI_COMM_WORLD);
        ParallelEliminateColumns(pProcRows, pProcVector, pPivotRow, Size, RowNum, i);
    }
}

void FindBackPivotRow(intRowIndex, int Size, int& IterProcRank,

```

```

    int& IterPivotPos) {
    for (int i = 0; i < ProcNum - 1; i++) {
        if ((pProcInd[i] <=RowIndex) && (RowIndex < pProcInd[i + 1]))
            IterProcRank = i;
    }
    if (RowIndex >= pProcInd[ProcNum - 1])
        IterProcRank = ProcNum - 1;
    IterPivotPos = RowIndex - pProcInd[IterProcRank];
}

void ParallelBackSubstitution(double* pProcRows, double* pProcVector,
    double* pProcResult, int Size, int RowNum) {
    int IterProcRank;
    int IterPivotPos;
    double IterResult;
    double val;
    for (int i = Size - 1; i >= 0; i--) {
        FindBackPivotRow(pParallelPivotPos[i], Size, IterProcRank,
            IterPivotPos);
        if (ProcRank == IterProcRank) {
            IterResult = pProcVector[IterPivotPos] / pProcRows[IterPivotPos*Size + i];
            pProcResult[IterPivotPos] = IterResult;
        }
        MPI_Bcast(&IterResult, 1, MPI_DOUBLE, IterProcRank, MPI_COMM_WORLD);
        for (int j = 0; j < RowNum; j++)
            if (pProcPivotIter[j] < i) {
                val = pProcRows[j * Size + i] * IterResult;
                pProcVector[j] = pProcVector[j] - val;
            }
    }
}

void RandomDataInitialization(double* pMatrix, double* pVector, int Size) {
    //vec
    pVector[0] = 2;
    pVector[1] = 12;
    pVector[2] = 17;
    pVector[3] = 57;
    pVector[4] = 7;

    pMatrix[0] = 1;
    pMatrix[1] = 2;
    pMatrix[2] = 3;
    pMatrix[3] = 4;
    pMatrix[4] = 5;
    pMatrix[5] = 2;
    pMatrix[6] = 3;
    pMatrix[7] = 7;
    pMatrix[8] = 10;
    pMatrix[9] = 13;
    pMatrix[10] = 3;
    pMatrix[11] = 5;
    pMatrix[12] = 11;
    pMatrix[13] = 16;
    pMatrix[14] = 21;
    pMatrix[15] = 2;
    pMatrix[16] = -7;
    pMatrix[17] = 7;
    pMatrix[18] = 7;
    pMatrix[19] = 2;
    pMatrix[20] = 1;
    pMatrix[21] = 4;

```

```

        pMatrix[22] = 5;
        pMatrix[23] = 3;
        pMatrix[24] = 10;
    }

    void ParallelResultCalculation(double* pProcRows, double* pProcVector,
        double* pProcResult, int Size, int RowNum) {
        ParallelGaussianElimination(pProcRows, pProcVector, Size, RowNum);
        ParallelBackSubstitution(pProcRows, pProcVector, pProcResult, Size,
            RowNum);
    }

    void PrintVector(double* pResult, int Size) {
        int i;
        for (i = 0; i < Size; i++)
            printf("%7.4f ", pResult[pParallelPivotPos[i]]);
    }

    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 = 25; // 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);
        ParallelResultCalculation(pProcRows, pProcVector, pProcResult, Size, RowNum);
        finish = MPI_Wtime();
        duration = finish - start;
        PrintVector(pResult, Size);
        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 Фото результатов

```
Result  
3.0000 -5.0000 4.0000 -2.0000 1.0000  
Time of execution: 0.000041
```