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

ПАРАЛЛЕЛЬНОЕ И РАСПРЕДЕЛЕННОЕ ПРОГРАММИРОВАНИЕ. WORK14

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 14

1.1 Условие задачи

Аналогично работе с OMP выполните следующее задание через MPI.

Решите систему линейных уравнений согласно варианту параллельным методом Гаусса.

2.

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

1.2 Решение. Параллельная версия

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <time.h>
#include <math.h>
#include "mpi.h"
int ProcNum;
int ProcRank;
int* pParallelPivotPos; // The Number of pivot rows selected at the iterations
int* pProcPivotIter; // The Iterations, at which the rows were pivots
int* pProcInd;
int* pProcNum;
// Function for my initialization of the matrix and the vector elements
void MyDataInitialization(double* pMatrix, double* pVector, int Size) {
        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;
}
// Function for memory allocation and definition of the objects elements
void ProcessInitialization(double*& pMatrix, double*& pVector, double*& pResult, double*&
↪  pProcRows, double*& pProcVector, double*& pProcResult, int& Size, int& RowNum) {
        int RestRows;
        int i;
        Size = 5;
        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];
                MyDataInitialization(pMatrix, pVector, Size);
        }
}
// Function for data distribution
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;
}
// Function for collecting results
void ResultCollection(double* pProcResult, double* pResult) {
        MPI_Gatherv(pProcResult, pProcNum[ProcRank], MPI_DOUBLE, pResult, pProcNum,
↪  pProcInd, MPI_DOUBLE, 0, MPI_COMM_WORLD);
}
// Function for formatted vector output
void PrintVector(double* pResult, int Size) {
        int i;
        for (i = 0; i < Size; i++)
                printf("%7.4f ", pResult[pParallelPivotPos[i]]);
}
```

```
// Column elimination
void ParallelColumnElimination(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;
                }
        }
}
// 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++) {
                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);
```

```
                ParallelColumnElimination(pProcRows, pProcVector, pPivotRow, Size, RowNum,
↪  i);
        }
}
// Finding back pivot row
void FindBackPivotRow(int RowIndex, 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];
}
// Back substation
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;
                        }
                }
        }
}
// Function for the execution of Gauss algorithm
void ParallelResultCalculation(double* pProcRows, double* pProcVector, double* pProcResult,
↪  int Size, int RowNum) {
        ParallelGaussianElimination(pProcRows, pProcVector, Size, RowNum);
        ParallelBackSubstitution(pProcRows, pProcVector, pProcResult, Size, RowNum);
}
// Function for computational process termination
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;
}
int main() {
        double* pMatrix; // The matrix of the linear system
        double* pVector; // The right parts of the linear system
        double* pResult; // The result vector
        double* pProcRows;
        double* pProcVector;
        double* pProcResult;
        int Size; // The size of the matrix and the vectors
        int RowNum;
        double start, finish, duration;
        setvbuf(stdout, 0, _IONBF, 0);
        MPI_Init(NULL, NULL);
        MPI_Comm_rank(MPI_COMM_WORLD, &ProcRank);
        MPI_Comm_size(MPI_COMM_WORLD, &ProcNum);
        // Data initialization
        ProcessInitialization(pMatrix, pVector, pResult, pProcRows, pProcVector,
↪  pProcResult, Size, RowNum);
        start = MPI_Wtime();
        DataDistribution(pMatrix, pProcRows, pVector, pProcVector, Size, RowNum);
        ParallelResultCalculation(pProcRows, pProcVector, pProcResult, Size, RowNum);
        ResultCollection(pProcResult, pResult);
        finish = MPI_Wtime();
        duration = finish - start;
        if (ProcRank == 0) {
                printf("Result Vector: \n");
                PrintVector(pResult, Size);
        }
        if (ProcRank == 0)
                printf("\n Time of execution: %f\n", duration); // Printing the time spent
↪  by parallel Gauss algorithm
        ProcessTermination(pMatrix, pVector, pResult, pProcRows, pProcVector, pProcResult);
↪  // Program termination
        MPI_Finalize();
}
```

2 Результат работы

```
Result Vector:
 3.0000 -5.0000  4.0000 -2.0000  1.0000
 Time of execution: 0.000057
```

Ответ:

$$\begin{cases} x_1 = 3 \\ x_2 = -5 \\ x_3 = 4 \\ x_4 = -2 \\ x_5 = 1 \end{cases}$$

3 Характеристики компьютера

Характеристики устройства

Имя устройства	DESKTOP-MSS8D39
Процессор	Intel(R) Core(TM) i5-6500 CPU @ 3.20GHz 3.20 GHz
Оперативная память	8,00 ГБ
Код устройства	E3BB953D-13B0-42A7-944B-1ED9FD0E C328
Код продукта	00330-80000-00000-AA153
Тип системы	64-разрядная операционная система, процессор x64