МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

НОВОСИБИРСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Факультет информационных технологий

Кафедра параллельных вычислений

ОТЧЕТ

О ВЫПОЛНЕНИИ ЛАБОРАТОРНОЙ РАБОТЫ

«Умножение матрицы на матрицу в MPI 2D решетке»

студента 2 курса, 21204 группы

Осипова Александра Александровича

Направление 09.03.01 – «Информатика и вычислительная техника»

Преподаватель:
Кандидат технических наук
А. Ю. Власенко

Новосибирск 2023

СОДЕРЖАНИЕ

ЦЕЛЬ

Освоение концепции MPI-коммуникаторов и декартовых топологий, а также концепции произвольных типов данных.

ЗАДАНИЕ

1. Реализовать параллельный алгоритм умножения матрицы на матрицу

при 2D решетке процессов с соблюдением требований.

2. Исследовать производительность параллельной программы при

фиксированном размере матрицы в зависимости от и размера решетки:

2x12, 3x8, 4x6, 6x4, 8x3, 12x2.

3. Выполнить профилирование программы при использовании 8-и ядер

с решетками 2x4, 4x2.

Общий алгоритм

1. Матрица A распределяется по горизонтальным полосам вдоль координаты (x,0).

2. Матрица B распределяется по вертикальным полосам вдоль координаты (0,y).

3. Полосы A распространяются в измерении y.

4. Полосы B распространяются в измерении x.

5. Каждый процесс вычисляет одну подматрицу произведения.

6. Матрица C собирается из (x,y) плоскости.

ОПИСАНИЕ РАБОТЫ

1. Были сгенерированы значения матриц A[2400x1000] и B[1000x4800] в текстовые файлы A.txt и B.txt соответственно с помощью функции GenerateMatrix (см. Приложение 1)

2. Были написаны последовательная программа и параллельная, в которой использовался MPI (см. Приложение 2 и Приложение 3)

2.1 Были созданы три коммуникатора:

```
- MPI_Comm GridComm       //2D решетка p1xp2
- MPI_Comm RowsComm       //коммуникатор для каждой строчки
- MPI_Comm ColumnComm     //коммуникатор для каждого столбца
```

2.2 Функция, распределяющая матрицу A по горизонтальным полосам на вертикальную линейку процессов (0;0), (1;0), (2;0), …, (p1 - 1; 0).

```
void SliceMatrixA(double* A, double* Ap, TGrid* grid, MPI_Comm ColumnComm, MPI_Comm
RowsComm, TCoords coords) {
     int sendcount = N1 * N2 / grid->numRows;
     if (coords.y == 0) {
           MPI_Scatter(A, sendcount, MPI_DOUBLE, Ap, sendcount, MPI_DOUBLE, 0,
ColumnComm);
     }
     MPI_Bcast(Ap, sendcount, MPI_DOUBLE, 0, RowsComm);
}
```

2.3 Функция, распределяющая матрицу B по вертикальным полосам на горизонтальную линейку процессов (0;0), (0;1), (0;2), …, (0; p2 – 1).

```
void SliceMatrixB(double* B, double* Bp, int blockSize, MPI_Comm RowsComm, MPI_Comm
ColumnComm, TCoords coords) {
     if (coords.x == 0) {
           MPI_Datatype column, columntype;
           MPI_Type_vector(N2, blockSize, N3, MPI_DOUBLE, &column);
           MPI_Type_commit(&column);
           MPI_Type_create_resized(column, 0, blockSize * sizeof(double), &columntype);
           MPI_Type_commit(&columntype);

           MPI_Scatter(B, 1, columntype, Bp, blockSize * N2, MPI_DOUBLE, 0, RowsComm);

           MPI_Type_free(&column);
           MPI_Type_free(&columntype);
     }
     MPI_Bcast(Bp, blockSize * N2, MPI_DOUBLE, 0, ColumnComm);
}
```

2.4 Функция, собирающая результирующую матрицу C на (0;0) процессе.

```
void AssembleMatrixC(double* Cp, double* C, TGrid* grid, MPI_Comm GridComm, int rank) {
     int numRowsInBlock = N1 / grid->numRows;
     int numColumnInBlock = N3 / grid->numColumns;
```

```
    MPI_Datatype minor, minortype;

    MPI_Type_vector(numRowsInBlock, numColumnInBlock, N3, MPI_DOUBLE, &minor);
    MPI_Type_commit(&minor);

    MPI_Type_create_resized(minor, 0, numColumnInBlock * sizeof(double), &minortype);
    MPI_Type_commit(&minortype);

    int* rcount = (int*)malloc(grid->numProc * sizeof(int));
    int* displacement = (int*)malloc(grid->numProc * sizeof(int));

    for (int i = 0; i < grid->numRows; ++i) {
          for (int j = 0; j < grid->numColumns; ++j){
                rcount[i * grid->numColumns + j] = 1;
                displacement[i * grid->numColumns + j] = i * grid->numColumns *
    numRowsInBlock + j;
          }
    }

    MPI_Gatherv(Cp, numRowsInBlock * numColumnInBlock, MPI_DOUBLE, C, rcount,
    displacement, minortype, 0, GridComm);

    free(rcount);
    free(displacement);

    MPI_Type_free(&minor);
    MPI_Type_free(&minortype);
}
```

3. Было измерено время умножения матриц A и B в последовательной программе.

4. Было измерено время умножения матриц A и B в параллельной программе на следующих решетках: 2x12, 3x8, 4x6, 6x4, 8x3, 12x2.

5. Было сделано профилирование на следующих решетках процессов: 2x4 и 4x2.

Оценка производительности (см. Приложение 8,9)

	Решетка						
	1x1	2x12	3x8	4x6	6x4	8x3	12x2
Время, сек	703	30.52	30.59	30.83	30.76	30.71	30.99
Ускорение	1	23.03	22.97	22.79	22.85	22.89	22.68
Эффективность, %	100	95.97	95.74	94.98	95.23	95.38	94.51

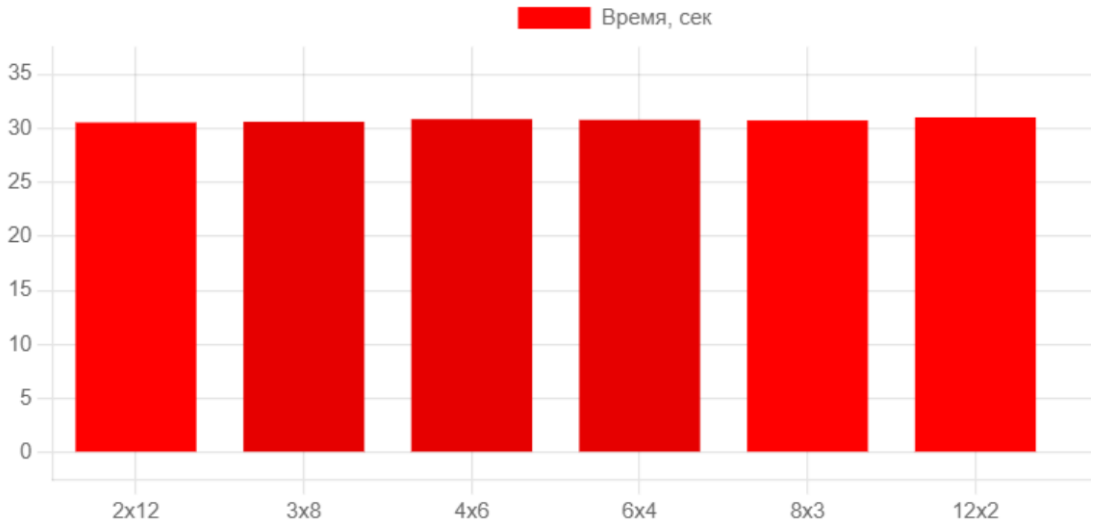


Рис.1 График времени

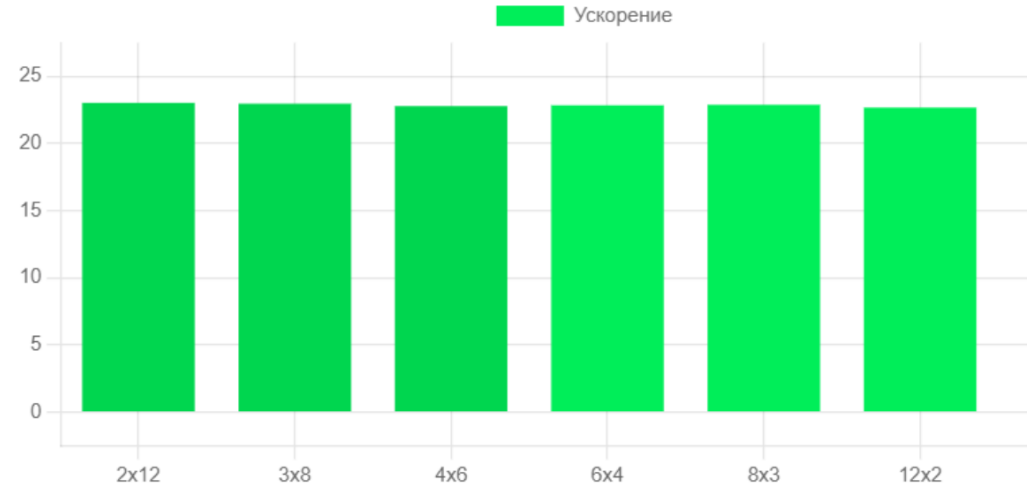


Рис.2 График ускорения

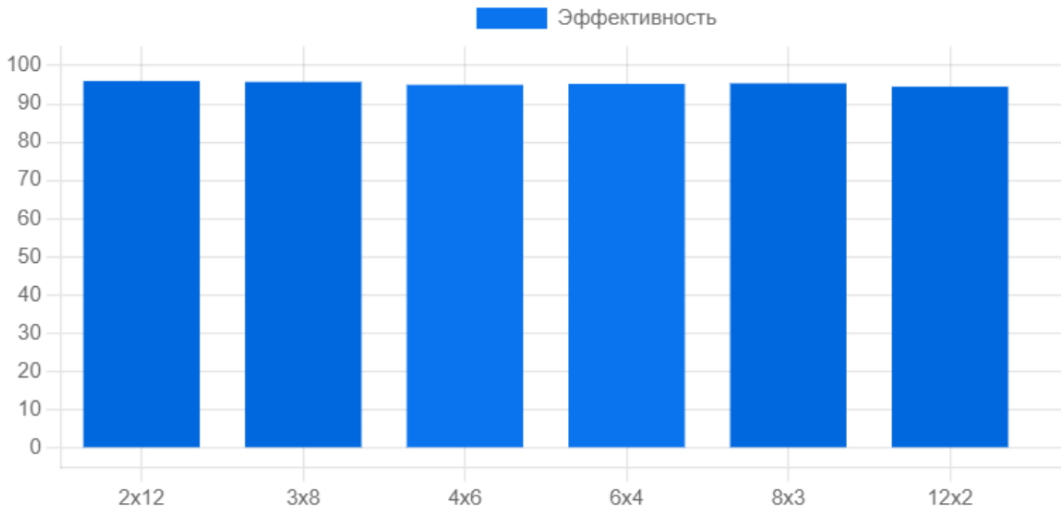
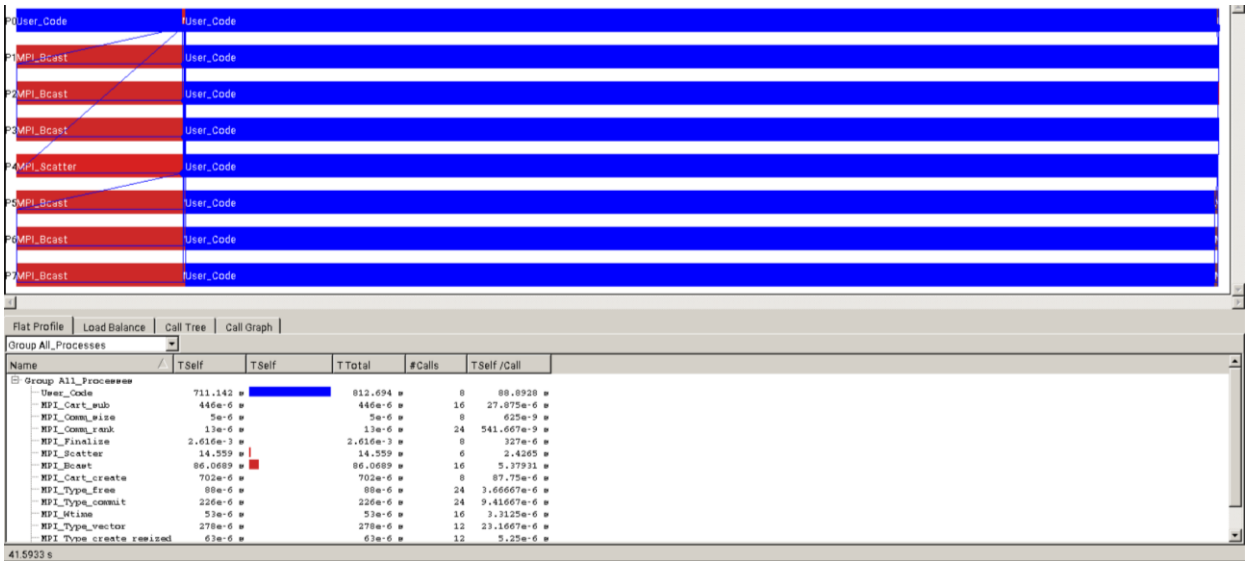
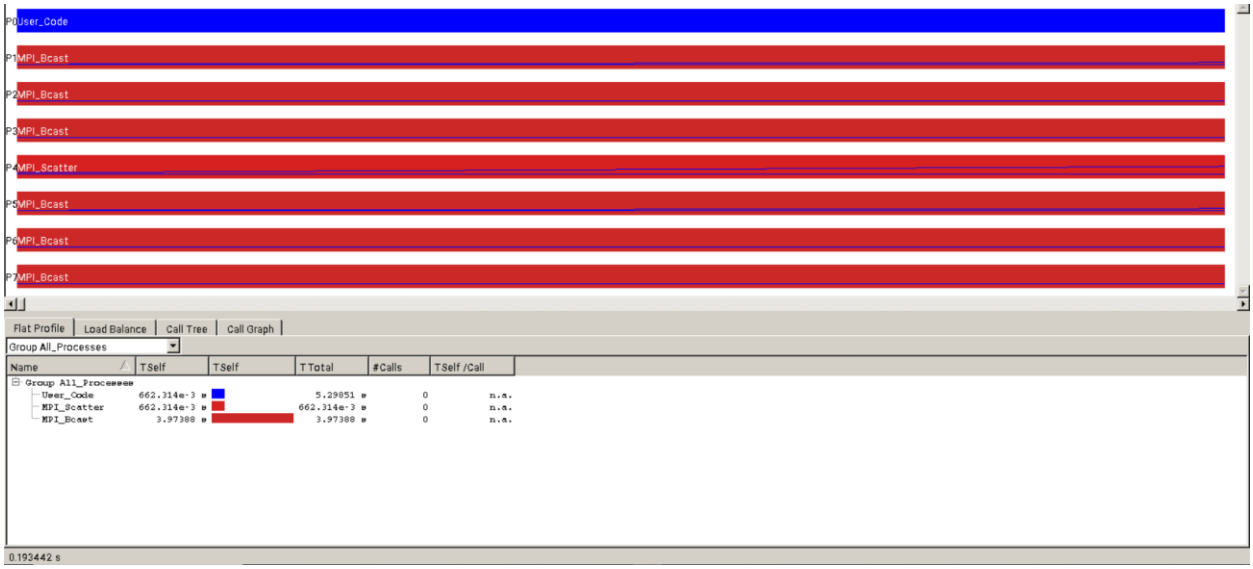


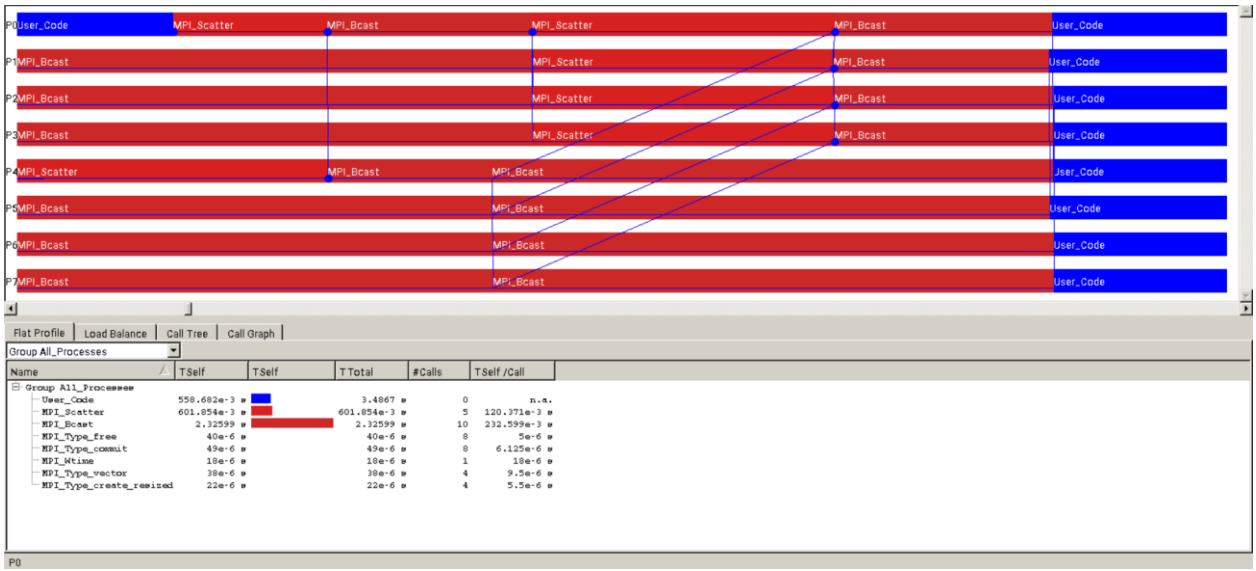
Рис.3 График эффективности

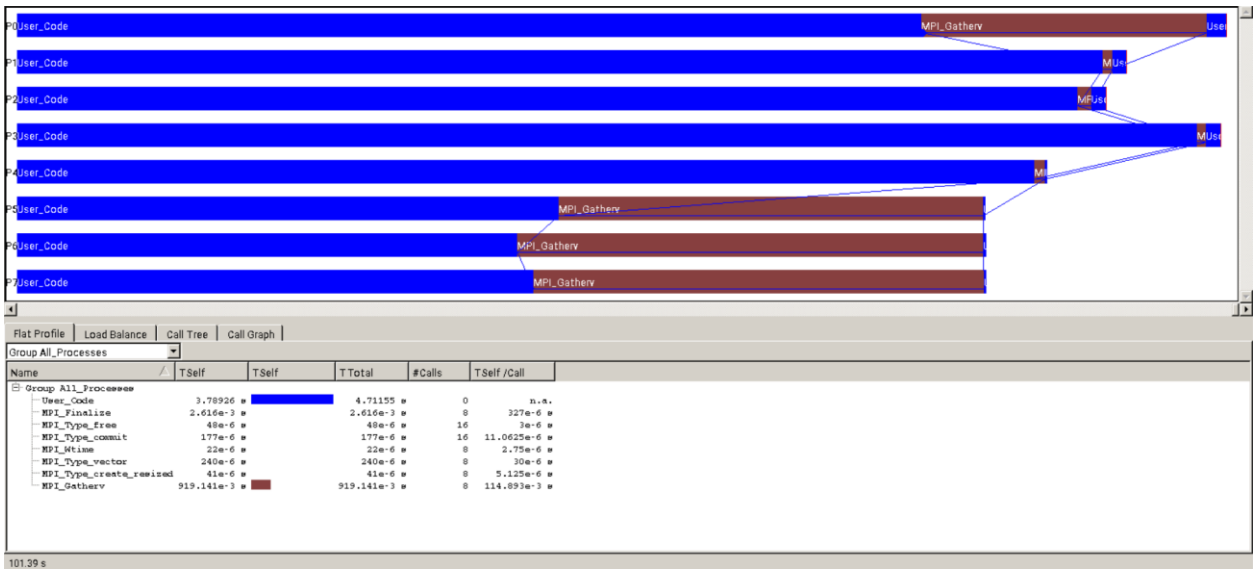
Профилирование

1. Решетка 2x4

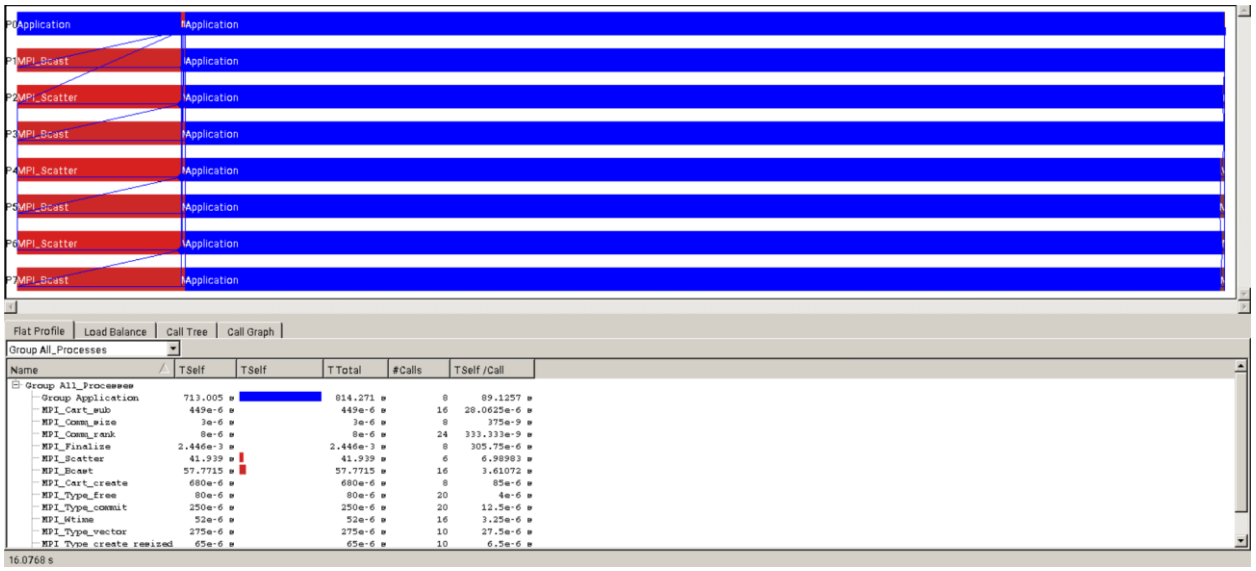


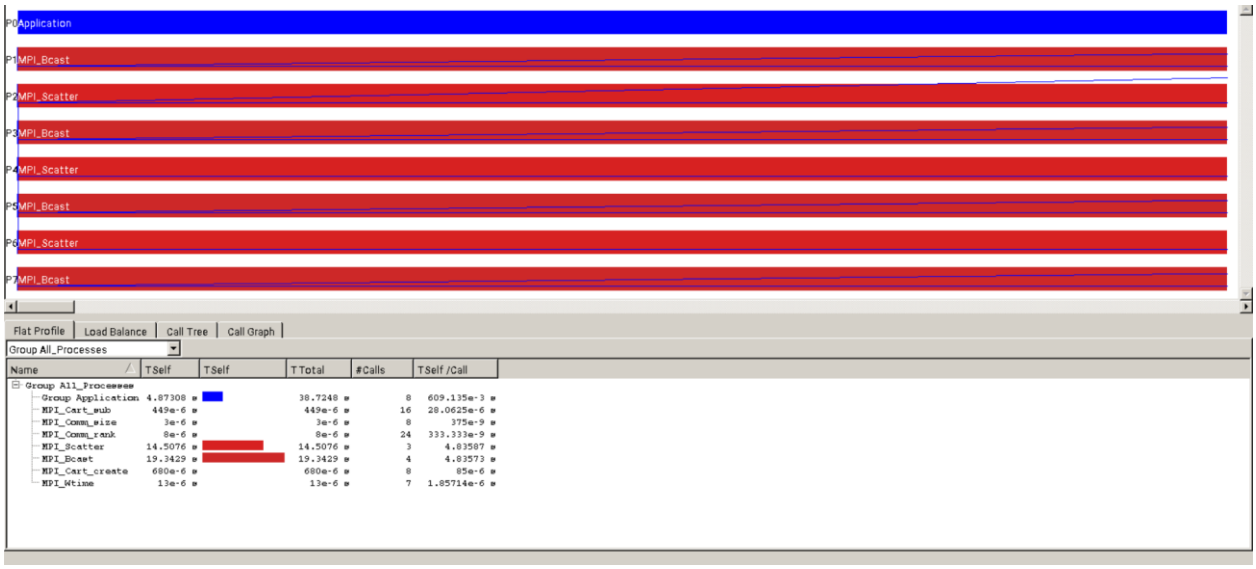


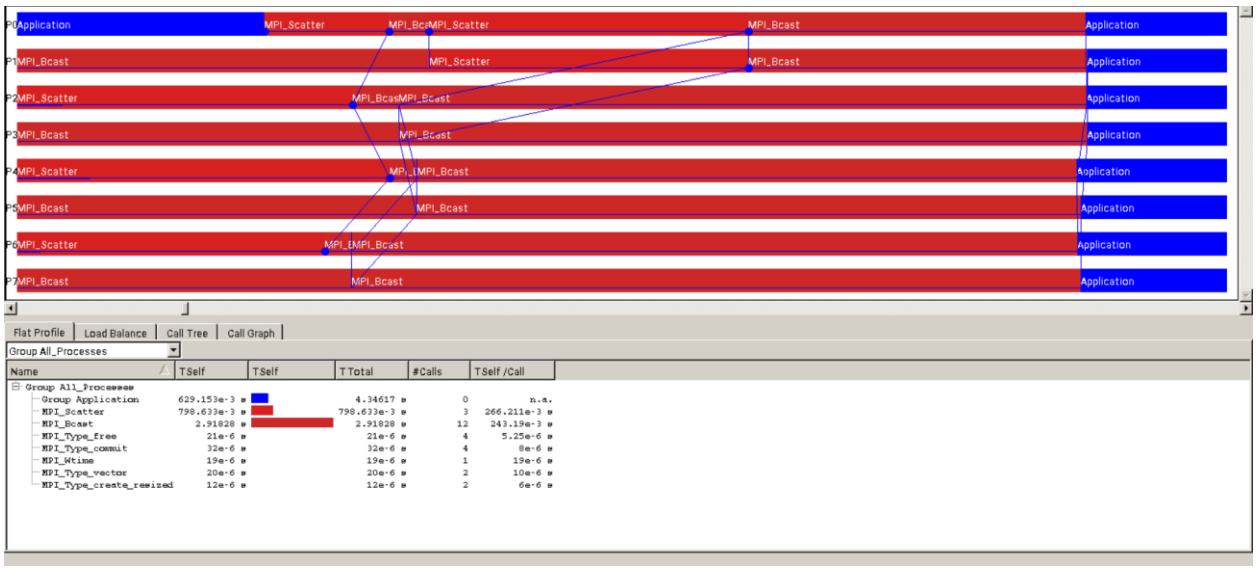




2. Решетка 4x2







Flat Profile | Load Balance | Call Tree | Call Graph

Group All_Processes

Name	TSelf	TSelf	TTotal	#Calls	TSelf /Call
Group All_Processes					
Group Application	629.153e-3 s		4.34617 s	0	n.a.
MPI_Scatter	798.633e-3 s		798.633e-3 s	3	266.211e-3 s
MPI_Bcast	2.91828 s		2.91828 s	12	243.19e-3 s
MPI_Type_free	21e-6 s		21e-6 s	4	5.25e-6 s
MPI_Type_commit	32e-6 s		32e-6 s	4	8e-6 s
MPI_Wtime	19e-6 s		19e-6 s	1	19e-6 s
MPI_Type_vector	20e-6 s		20e-6 s	2	10e-6 s
MPI_Type_create_resized	12e-6 s		12e-6 s	2	6e-6 s

Flat Profile | Load Balance | Call Tree | Call Graph

Group All_Processes

Name	TSelf	TSelf	TTotal	#Calls	TSelf /Call
Group All_Processes					
Group Application	2.46356 s		4.01747 s	0	n.a.
MPI_Finalize	2.446e-3 s		2.446e-3 s	8	305.75e-6 s
MPI_Type_free	59e-6 s		59e-6 s	16	3.6875e-6 s
MPI_Type_commit	218e-6 s		218e-6 s	16	13.625e-6 s
MPI_Wtime	20e-6 s		20e-6 s	8	2.5e-6 s
MPI_Type_vector	255e-6 s		255e-6 s	8	31.875e-6 s
MPI_Type_create_resized	53e-6 s		53e-6 s	8	6.625e-6 s
MPI_Gatherv	1.55086 s		1.55086 s	8	193.858e-3 s

19.8289 s

ЗАКЛЮЧЕНИЕ

В рамках данной лабораторной работы были освоены концепции MPI-коммуникаторов и декартовых топологий, а также концепции произвольных типов данных. С помощью использованного параллельного алгоритма умножения матриц удалось добиться ускорения в 23 раза, причем эффективность осталась такой же высокой (95%). Ощутимой разности по производительности между решетками с одним количеством процессов, но с разной размерностью, обнаружено не было.

Приложение 1. Функция, генерирующая матрицу в файле

```
void GenerateMatrix(string filename, int N, int M) {
	std::ofstream fileIn;
    fileIn.open(filename);
    if (fileIn.is_open()) {
	for (int i = 0; i < N * M; ++i) {
		double value = (double)(int)(((double)rand() / RAND_MAX) * 100) / 100;
		fileIn << value << std::endl;
			}
				fileIn.close();
				}
}
```

Приложение 2. Листинг последовательной программы

```
#include <cstdlib>
#include <iostream>
#include <fstream>

using namespace std;

enum MatrixDimension {
    N1 = 2400, // кратно 24
    N2 = 5500,
    N3 = 4800  // кратно 24
};

double* InitMatrixA() {
    double* A = (double*)malloc(N1 * N2 * sizeof(double));
    std::ifstream file;
    file.open("A.txt");
    for (int i = 0; i < N1 * N2; ++i) {
	file >> A[i];
    }
    file.close();
    return A;
}

double* InitMatrixB() {
    double* B = (double*)malloc(N2 * N3 * sizeof(double));
    std::ifstream file;
    file.open("B.txt");
    for (int i = 0; i < N2 * N3; ++i) {
	file >> B[i];
    }
    file.close();
    return B;
}

void MatrixMULT(double* A, double* B, double* C, int L, int M, int N) {
    for (int i = 0; i < L; ++i) {
	for (int k = 0; k < M; ++k) {
		for (int j = 0; j < N; ++j) {
			C[i * N + j] += A[i * M + k] * B[k * N + j];
```

```
                }
            }
        }
}

void FreeMemory(double* A, double* B, double* C) {
    free(A);
    free(B);
    free(C);
}

int main(int argc, char** argv) {
    double* A = InitMatrixA();
    double* B = InitMatrixB();
    double* C = (double*)calloc(N1 * N3, sizeof(double));

    time_t startTime, endTime;

    time(&startTime);

    MatrixMULT(A, B, C, N1, N2, N3);

    time(&endTime);

    printf("Time: %f sec\n", difftime(endTime, startTime));

    FreeMemory(A, B, C);
}
```

Приложение 3. Листинг параллельной программы

```
#include "mpi.h"
#include <iostream>
#include <fstream>
#include <cstdlib>

using namespace std;

#define T1 703

enum MatrixDimension {
    N1 = 2400,
    N2 = 8000,
    N3 = 4800
};

typedef struct TCoords {
    int x;
    int y;
} TCoords;

typedef struct TGridProcesses {
    int numProc;
    int numRows;
    int numColumns;
}TGrid;

typedef struct TSubmatrix {
    double* data;
    int numColumns;
    int numRows;
```

```
	int size;
} TSubmatrix;

void CreateGridComm(MPI_Comm* GridComm, TGrid* grid) {
	int dimension[2] = { grid->numRows, grid->numColumns };
	int periods[2] = { 0, 0 };
	MPI_Cart_create(MPI_COMM_WORLD, 2, dimension, periods, true, GridComm);
}

void CreateRowsComm(MPI_Comm GridComm, MPI_Comm* RowsComm) {
	int subDimension[2] = { false, true };
	MPI_Cart_sub(GridComm, subDimension, RowsComm);
}

void CreateColumnComm(MPI_Comm GridComm, MPI_Comm* ColumnComm) {
	int subDimension[2] = { true, false };
	MPI_Cart_sub(GridComm, subDimension, ColumnComm);
}

double* InitMatrixA() {
	double* A = (double*)malloc(N1 * N2 * sizeof(double));
	std::ifstream file;
	file.open("A.txt");
	for (int i = 0; i < N1 * N2; ++i) {
		file >> A[i];
	}
	file.close();
	return A;
}

double* InitMatrixB() {
	double* B = (double*)malloc(N2 * N3 * sizeof(double));
	std::ifstream file;
	file.open("B.txt");
	for (int i = 0; i < N2 * N3; ++i) {
		file >> B[i];
	}
	file.close();
	return B;
}

void SliceMatrixA(double* A, double* Ap, TGrid* grid, MPI_Comm ColumnComm, MPI_Comm
RowsComm, TCoords coords) {
	int sendcount = N1 * N2 / grid->numRows;
	if (coords.y == 0) {
		MPI_Scatter(A, sendcount, MPI_DOUBLE, Ap, sendcount, MPI_DOUBLE, 0,
ColumnComm);
	}
	MPI_Bcast(Ap, sendcount, MPI_DOUBLE, 0, RowsComm);
}

void SliceMatrixB(double* B, double* Bp, int blockSize, MPI_Comm RowsComm, MPI_Comm
ColumnComm, TCoords coords) {
	if (coords.x == 0) {
		MPI_Datatype column, columntype;
		MPI_Type_vector(N2, blockSize, N3, MPI_DOUBLE, &column);
		MPI_Type_commit(&column);
		MPI_Type_create_resized(column, 0, blockSize * sizeof(double), &columntype);
		MPI_Type_commit(&columntype);

		MPI_Scatter(B, 1, columntype, Bp, blockSize * N2, MPI_DOUBLE, 0, RowsComm);
```

```
            MPI_Type_free(&column);
            MPI_Type_free(&columntype);
      }
      MPI_Bcast(Bp, blockSize * N2, MPI_DOUBLE, 0, ColumnComm);
}

void AssembleMatrixC(double* Cp, double* C, TGrid* grid, MPI_Comm GridComm, int rank) {
      int numRowsInBlock = N1 / grid->numRows;
      int numColumnInBlock = N3 / grid->numColumns;

      MPI_Datatype minor, minortype;

      MPI_Type_vector(numRowsInBlock, numColumnInBlock, N3, MPI_DOUBLE, &minor);
      MPI_Type_commit(&minor);

      MPI_Type_create_resized(minor, 0, numColumnInBlock * sizeof(double), &minortype);
      MPI_Type_commit(&minortype);

      int* rcount = (int*)malloc(grid->numProc * sizeof(int));
      int* displacement = (int*)malloc(grid->numProc * sizeof(int));

      for (int i = 0; i < grid->numRows; ++i) {
            for (int j = 0; j < grid->numColumns; ++j) {
                  rcount[i * grid->numColumns + j] = 1;
                  displacement[i * grid->numColumns + j] = i * grid->numColumns *
numRowsInBlock + j;
            }
      }

      MPI_Gatherv(Cp, numRowsInBlock * numColumnInBlock, MPI_DOUBLE, C, rcount,
displacement, minortype, 0, GridComm);

      free(rcount);
      free(displacement);

      MPI_Type_free(&minor);
      MPI_Type_free(&minortype);
}

void MatrixMULT(double* A, double* B, double* C, int L, int M, int N) {
      for (int i = 0; i < L; ++i) {
            for (int k = 0; k < M; ++k) {
                  for (int j = 0; j < N; ++j) {
                        C[i * N + j] += A[i * M + k] * B[k * N + j];
                  }
            }
      }
}

void FreeMemory(double* A, double* B, double* C, double* Ap, double* Bp, double* Cp, int
rank) {
      if (rank == 0) {
            free(A);
            free(B);
            free(C);
      }
      free(Ap);
      free(Bp);
      free(Cp);
}
```

```
void PrintResult(float totalTime, int numProc, TGrid* grid) {
	float boost = T1 / totalTime;
	float efficiency = (boost / (float)numProc) * 100;
	printf("Number of processes: %d\n", numProc);
	printf("Grid: %d x %d\n", grid->numRows, grid->numColumns);
	printf("Total time: %f sec\n", totalTime);
	printf("Sp = %f\n", boost);
	printf("Ep = %f\n\n", efficiency);
}

int main(int argc, char** argv) {
	int numProc, rank;
	MPI_Status status;

	MPI_Init(&argc, &argv);
	MPI_Comm_size(MPI_COMM_WORLD, &numProc);
	MPI_Comm_rank(MPI_COMM_WORLD, &rank);

	TGrid grid;
	grid.numProc = numProc;
	grid.numRows = atoi(argv[1]);
	grid.numColumns = atoi(argv[2]);

	TCoords coords;

	MPI_Comm GridComm;
	CreateGridComm(&GridComm, &grid);

	MPI_Comm RowsComm;
	CreateRowsComm(GridComm, &RowsComm);
	MPI_Comm_rank(RowsComm, &coords.y);

	MPI_Comm ColumnComm;
	CreateColumnComm(GridComm, &ColumnComm);
	MPI_Comm_rank(ColumnComm, &coords.x);

	double* A = NULL;
	double* B = NULL;
	double* C = NULL;

	if ((coords.x == 0) && (coords.y == 0)) {
		A = InitMatrixA();
		B = InitMatrixB();
		C = (double*)malloc(N1 * N3 * sizeof(double));
	}

	double startTime = MPI_Wtime();

	int sizeAp = (N1 / grid.numRows) * N2;
	double* Ap = (double*)malloc(sizeAp * sizeof(double));
	SliceMatrixA(A, Ap, &grid, ColumnComm, RowsComm, coords);

	int sizeBp = N2 * (N3 / grid.numColumns);
	double* Bp = (double*)malloc(sizeBp * sizeof(double));
	SliceMatrixB(B, Bp, N3 / grid.numColumns, RowsComm, ColumnComm, coords);

	int sizeCp = (N1 / grid.numRows) * (N3 / grid.numColumns);
	double* Cp = (double*)calloc(sizeCp, sizeof(double));
	MatrixMULT(Ap, Bp, Cp, N1 / grid.numRows, N2, N3 / grid.numColumns);

	AssembleMatrixC(Cp, C, &grid, GridComm, rank);
```

```
    double endTime = MPI_Wtime();

    if (rank == 0) {
        PrintResult(endTime - startTime, numProc, &grid);
    }

    FreeMemory(A, B, C, Ap, Bp, Cp, rank);

    MPI_Finalize();
}
```

Приложение 4. default.sh

```
#!/bin/bash

#PBS -l walltime=00:20:00
#PBS -l select=1:ncpus=1:mpiprocs=1:mem=4000m,place=free
#PBS -m n

cd $PBS_O_WORKDIR

MPI_NP=$(wc -l $PBS_NODEFILE | awk '{ print $1 }')

./default
```

Приложение 5. parallel.sh

```
#!/bin/bash

#PBS -l walltime=00:20:00
#PBS -l select=2:ncpus=12:mpiprocs=12:mem=4000m,place=free
#PBS -m n

cd $PBS_O_WORKDIR

MPI_NP=$(wc -l $PBS_NODEFILE | awk '{ print $1 }')

mpirun -machinefile $PBS_NODEFILE -np 24 ./parallel 2 12
mpirun -machinefile $PBS_NODEFILE -np 24 ./parallel 3 8
mpirun -machinefile $PBS_NODEFILE -np 24 ./parallel 4 6
mpirun -machinefile $PBS_NODEFILE -np 24 ./parallel 6 4
mpirun -machinefile $PBS_NODEFILE -np 24 ./parallel 8 3
mpirun -machinefile $PBS_NODEFILE -np 24 ./parallel 12 2
```

Приложение 6. trace2x4.sh

```
#!/bin/bash

#PBS -l walltime=00:10:00
#PBS -l select=1:ncpus=8:mpiprocs=8:mem=4000m,place=free
#PBS -m n

cd $PBS_O_WORKDIR
```

```
MPI_NP=$(wc -l $PBS_NODEFILE | awk '{ print $1 }')

mpirun -trace -machinefile $PBS_NODEFILE -np 8 ./parallel 2 4
```

Приложение 7. trace4x2.sh

```
#!/bin/bash

#PBS -l walltime=00:10:00
#PBS -l select=1:ncpus=8:mpiprocs=8:mem=4000m,place=free
#PBS -m n

cd $PBS_O_WORKDIR

MPI_NP=$(wc -l $PBS_NODEFILE | awk '{ print $1 }')

mpirun -trace -machinefile $PBS_NODEFILE -np 8 ./parallel 4 2
```

Приложение 8. Результат работы последовательной программы

```
hpcuser60@clu:~/lab3>
hpcuser60@clu:~/lab3>
hpcuser60@clu:~/lab3>
hpcuser60@clu:~/lab3> cat default.sh.o5504920
Time: 703.000000 sec
hpcuser60@clu:~/lab3>
hpcuser60@clu:~/lab3>
hpcuser60@clu:~/lab3>
```

Приложение 9. Результат работы параллельной программы

```
hpcuser60@clu:~/lab3> cat parallel.sh.o5504930
Number of processes: 24
Grid: 2 x 12
Total time: 30.520563 sec
Sp = 23.033651
Ep = 95.973549

Number of processes: 24
Grid: 3 x 8
Total time: 30.594727 sec
Sp = 22.977816
Ep = 95.740898

Number of processes: 24
Grid: 4 x 6
Total time: 30.838913 sec
Sp = 22.795876
Ep = 94.982819

Number of processes: 24
Grid: 6 x 4
Total time: 30.759123 sec
Sp = 22.855007
Ep = 95.229195

Number of processes: 24
Grid: 8 x 3
Total time: 30.711260 sec
Sp = 22.890627
Ep = 95.377609

Number of processes: 24
Grid: 12 x 2
Total time: 30.991093 sec
Sp = 22.683937
Ep = 94.516403

hpcuser60@clu:~/lab3>
```