**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 ПРАКТИЧЕСКОЙ РАБОТЫ**

«Умножение матрицы на матрицу в MPI 2D решетке»

студента 2 курса, группы 21206

**Балашова Вячеслава Вадимовича**

Направление 09.03.01 – «Информатика и вычислительная техника»

Преподаватель:

Кандидат технических наук

А. Ю. Власенко

Новосибирск 2023

**Содержание**

**Цель3**

**Задание3**

**Описание работы4**

**Заключение11**

**Приложение 1. Исходный код программы12**

**Цель**

1. Научиться создавать топологии и пользовательские типы данных в MPI программах

**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. Размер матриц подобрать таким образом, чтобы худшее из времен данного набора было не менее 30 сек.

3. Выполнить профилирование программы при использовании 8-и ядер с решетками 2x4, 4x2.

**Описание работы**

Ход выполнения работы:

1. Была создана программа на языке C с использованием интерфейса MPI, вычисляющая умножение матрицы на матрицу. Используется алгоритм разбиения матрицы А на строки и матрицы В на столбцы с последующим получением миноров матрицы С в соответствии топологией процессов.
2. Процессы образуют двумерную сетку, размеры которой задаются как аргументы командной строки.
3. Эмулирована ситуация получения матриц A и B на процессе с координатами (0; 0) и последующей раздачей определенным методом частей этих матриц и последующим сбором миноров матрицы С в том же (0; 0) процессе.

|  |  |
| --- | --- |
| Сетка | Время, с |
| 2x12 | 52.2389 |
| 3x8 | 50.4139 |
| 4x6 | 51.0657 |
| 6x4 | 49.8065 |
| 8x3 | 50.7267 |
| 12x2 | 52.7863 |

1. Был проведен замер времени работы программы от разных размеров сетки. Результаты замеров представлены в таблице.

Таблица 1. Соответствие размеров сетки процессов с временем работы программы.

1. Было выполнено профилирование программы при размере сетки процессов, равной 2x4 и 4x2.

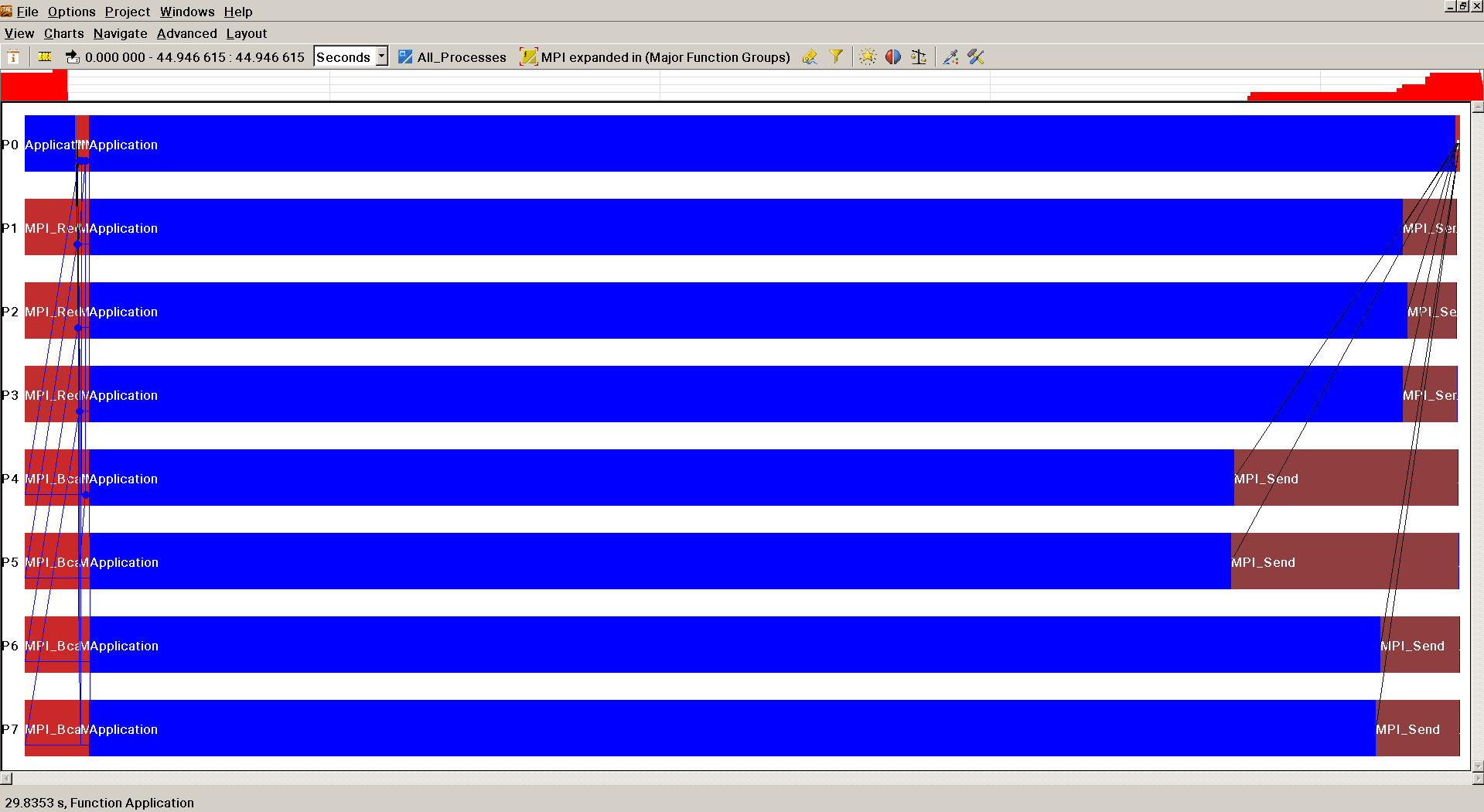
Рисунок 1. Профилирование программы, 2х4.

Рисунок 2. Профилирование программы, 2х4.

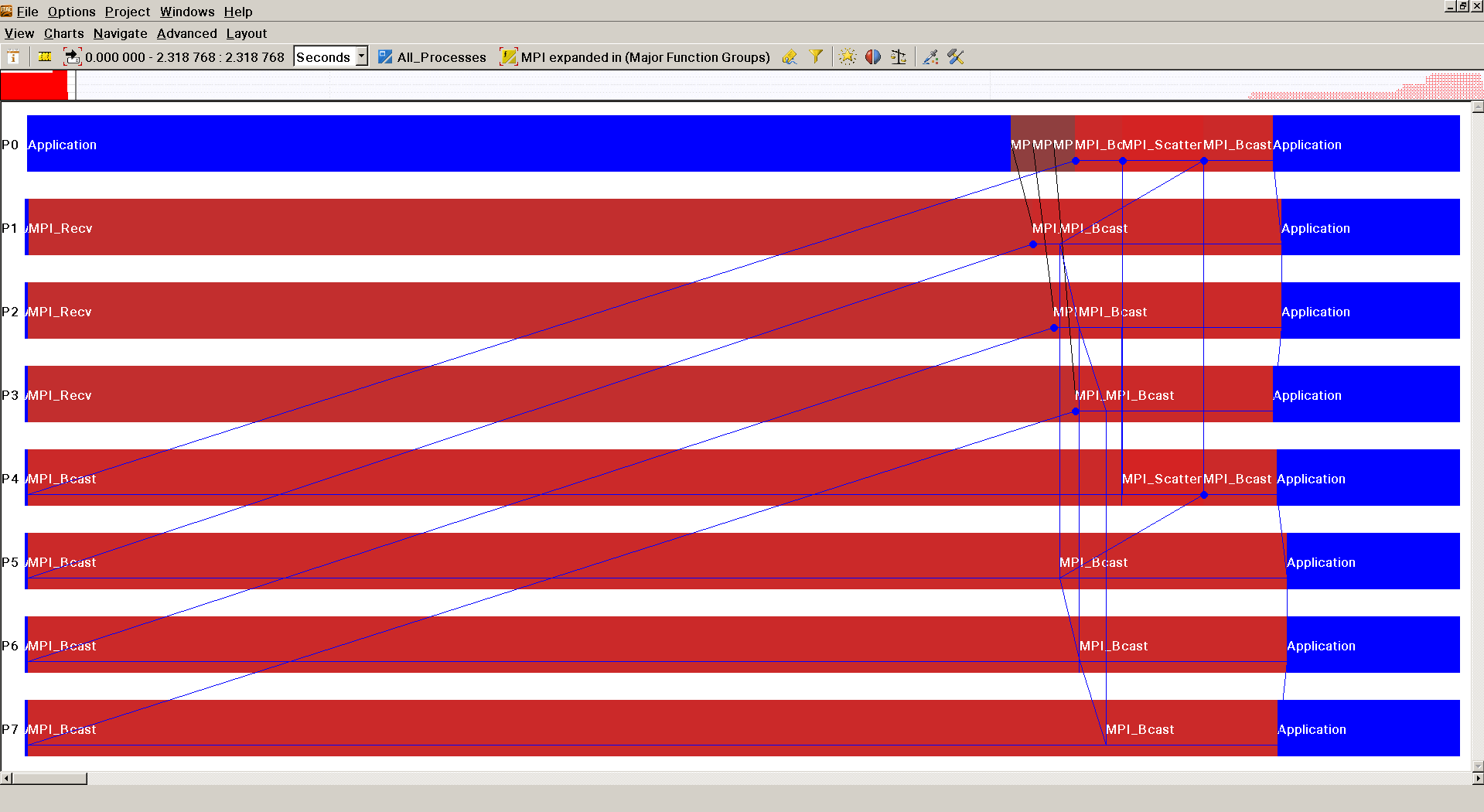


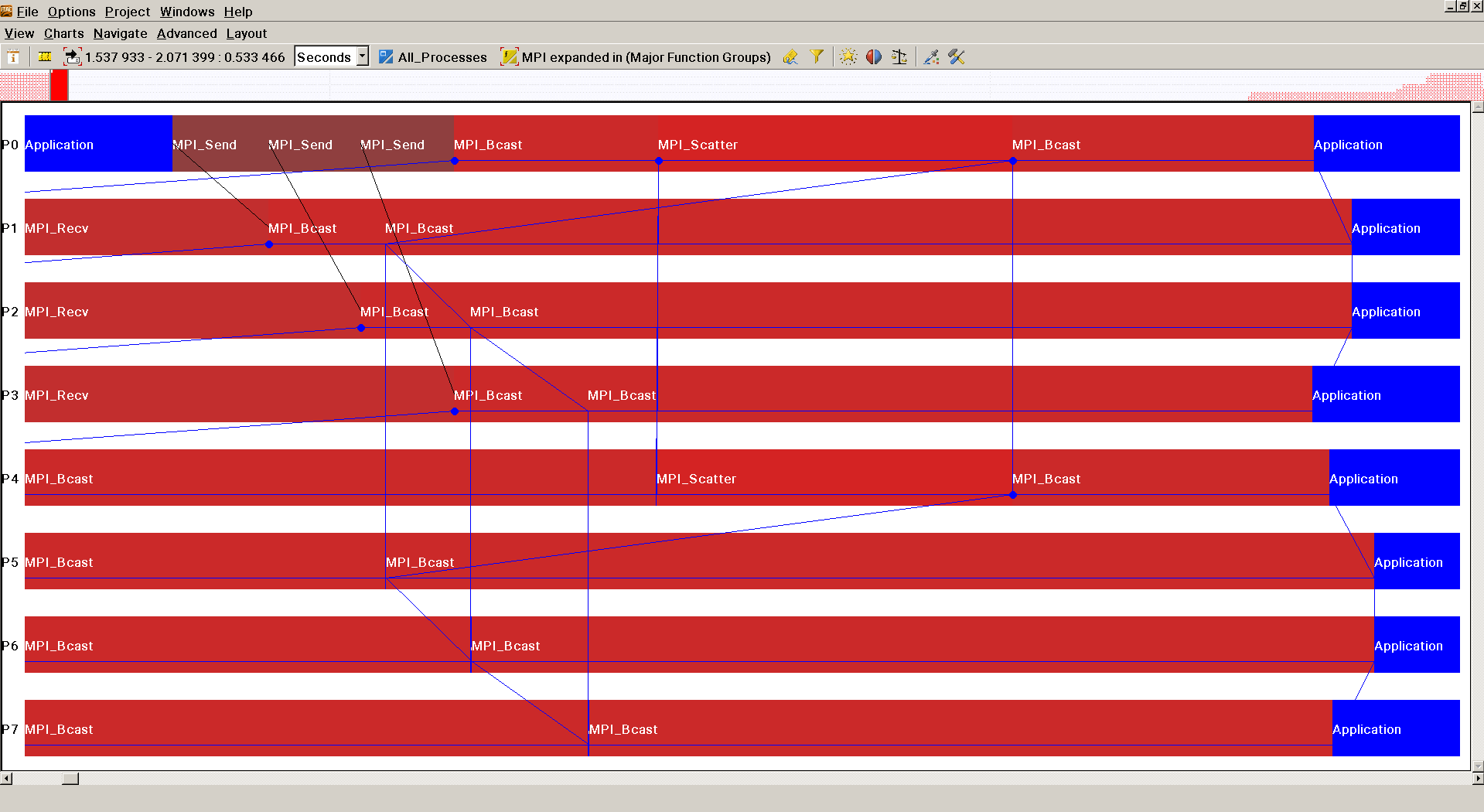
Рисунок 3. Профилирование программы, 2х4.

Рисунок 4. Профилирование программы, 2х4.

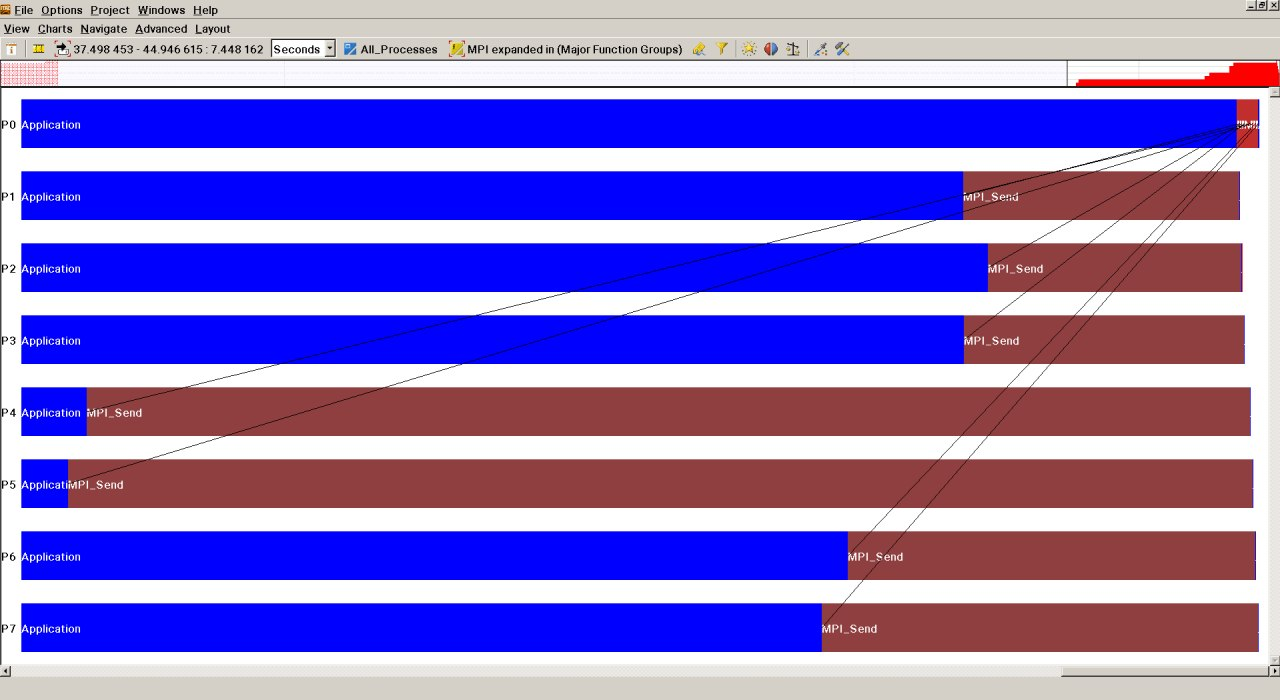


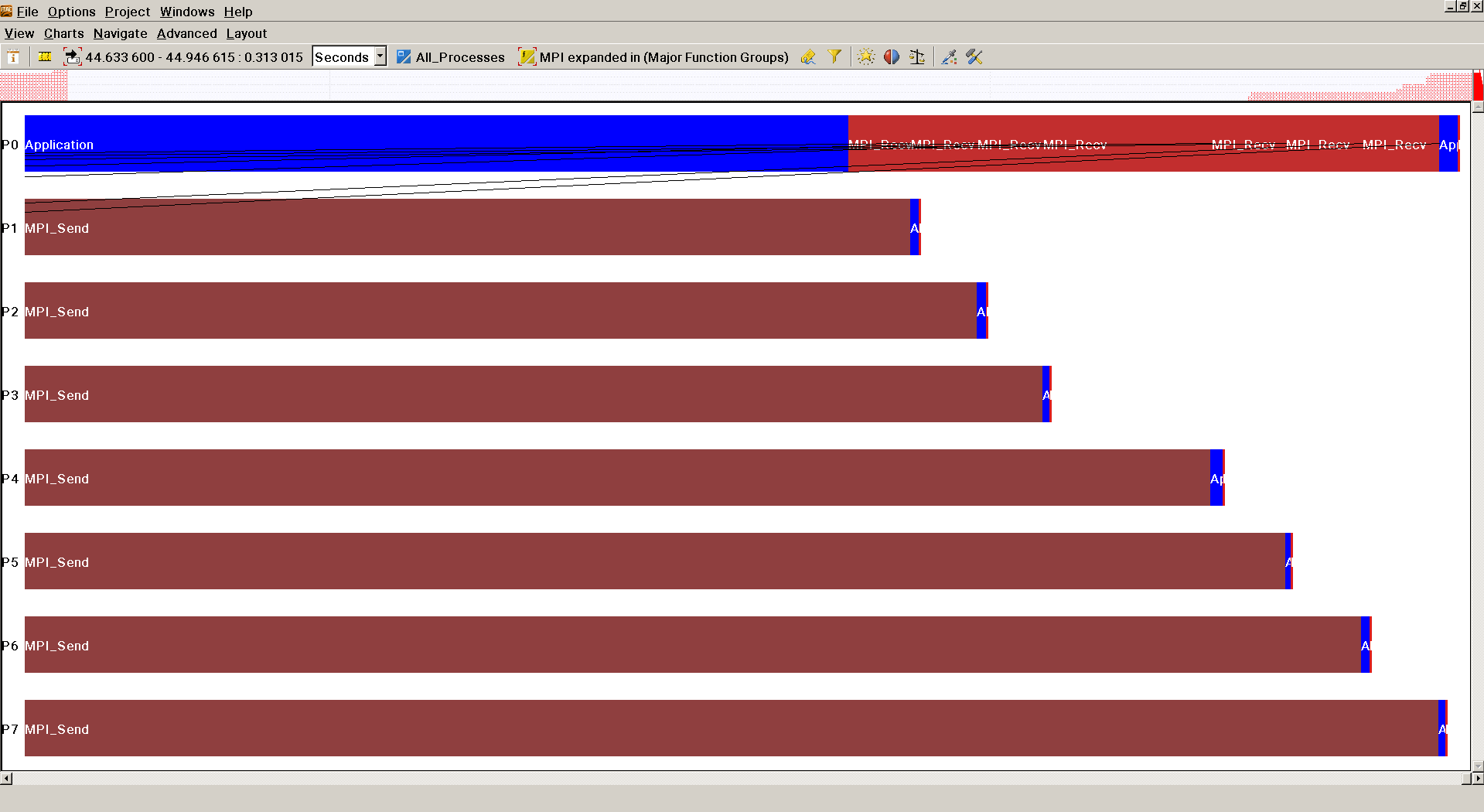
Рисунок 5. Профилирование программы, 2х4.

Рисунок 6. Профилирование программы, 2х4.

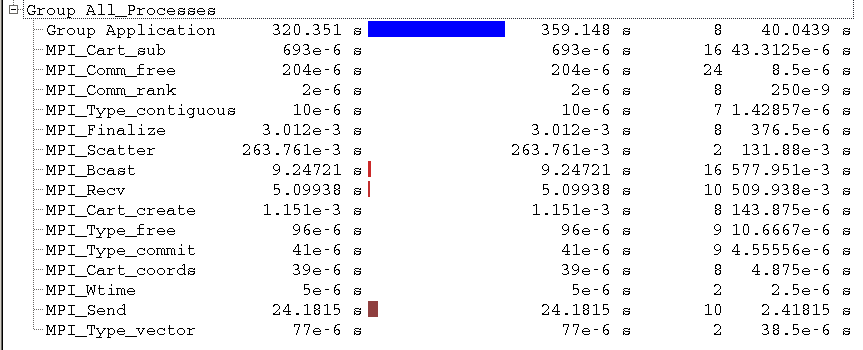


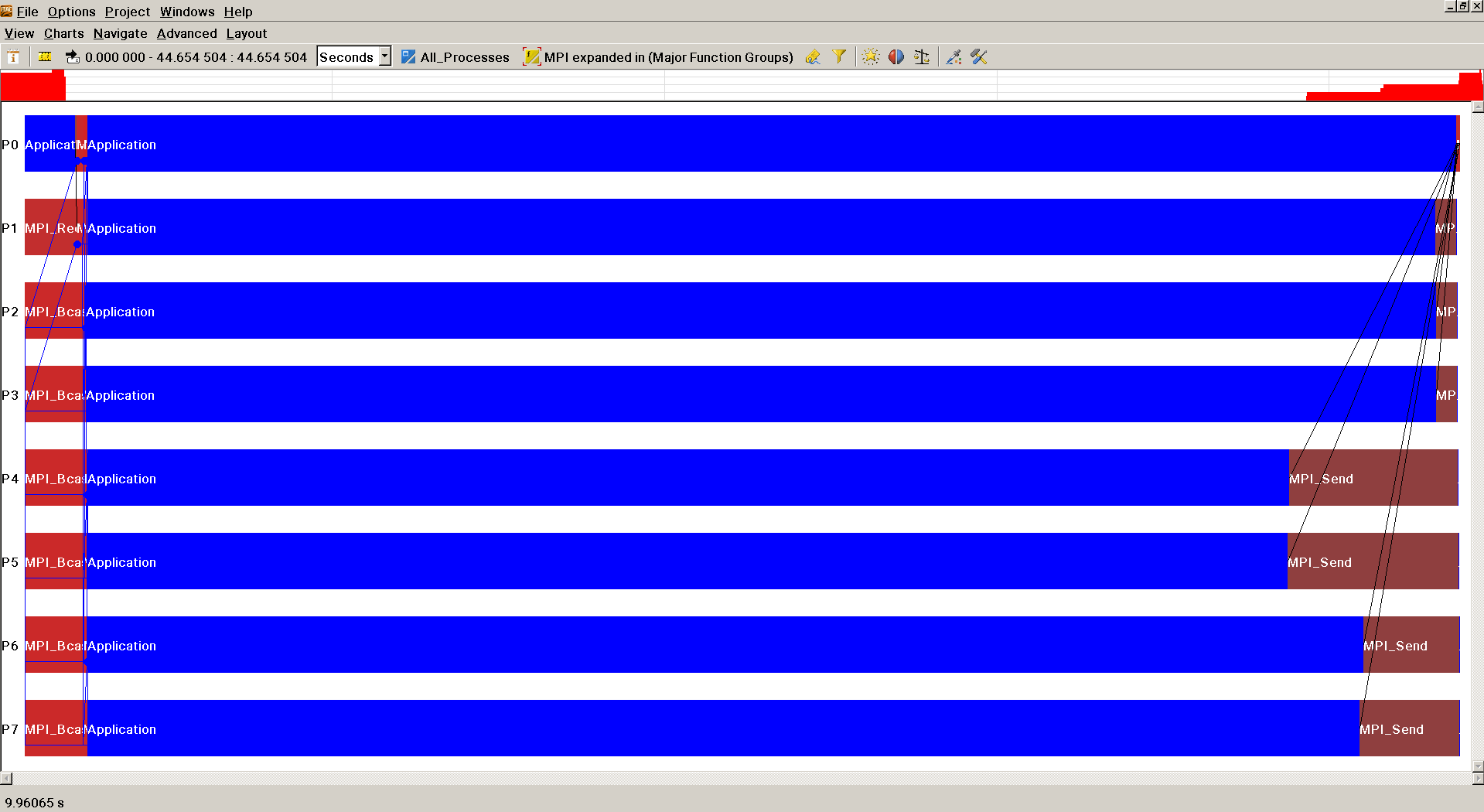
Рисунок 7. Профилирование программы, 4х2.

Рисунок 8. Профилирование программы, 4х2.

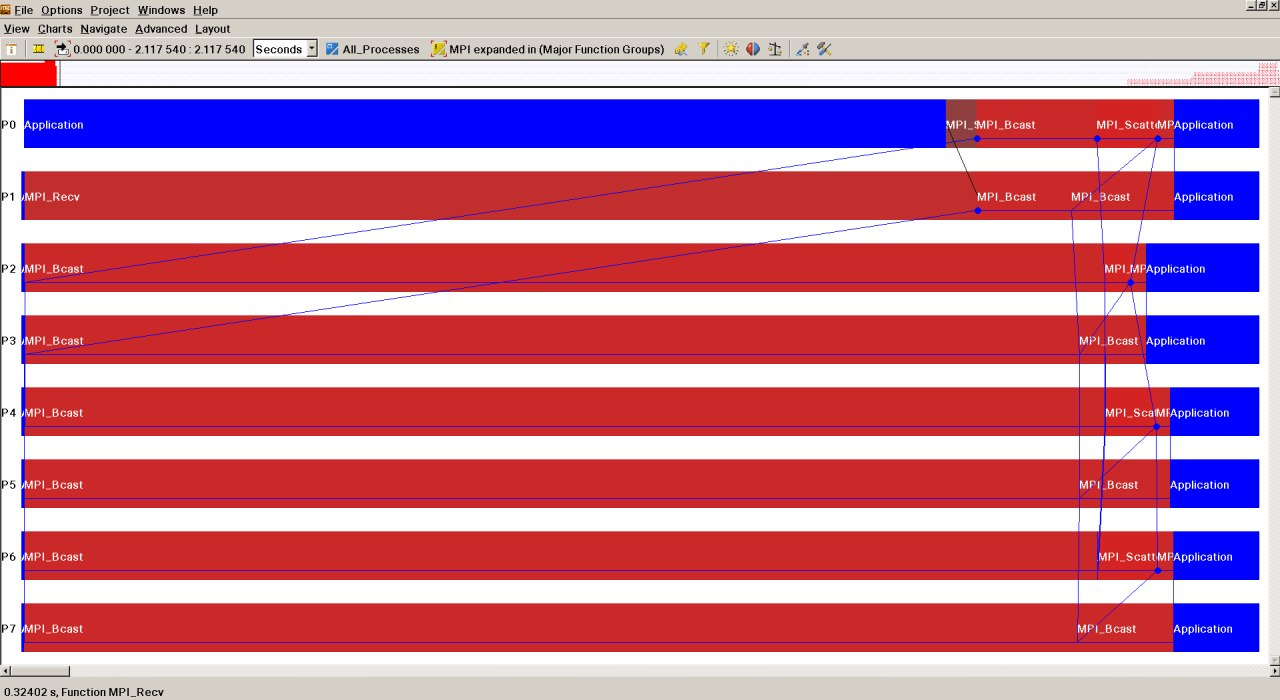


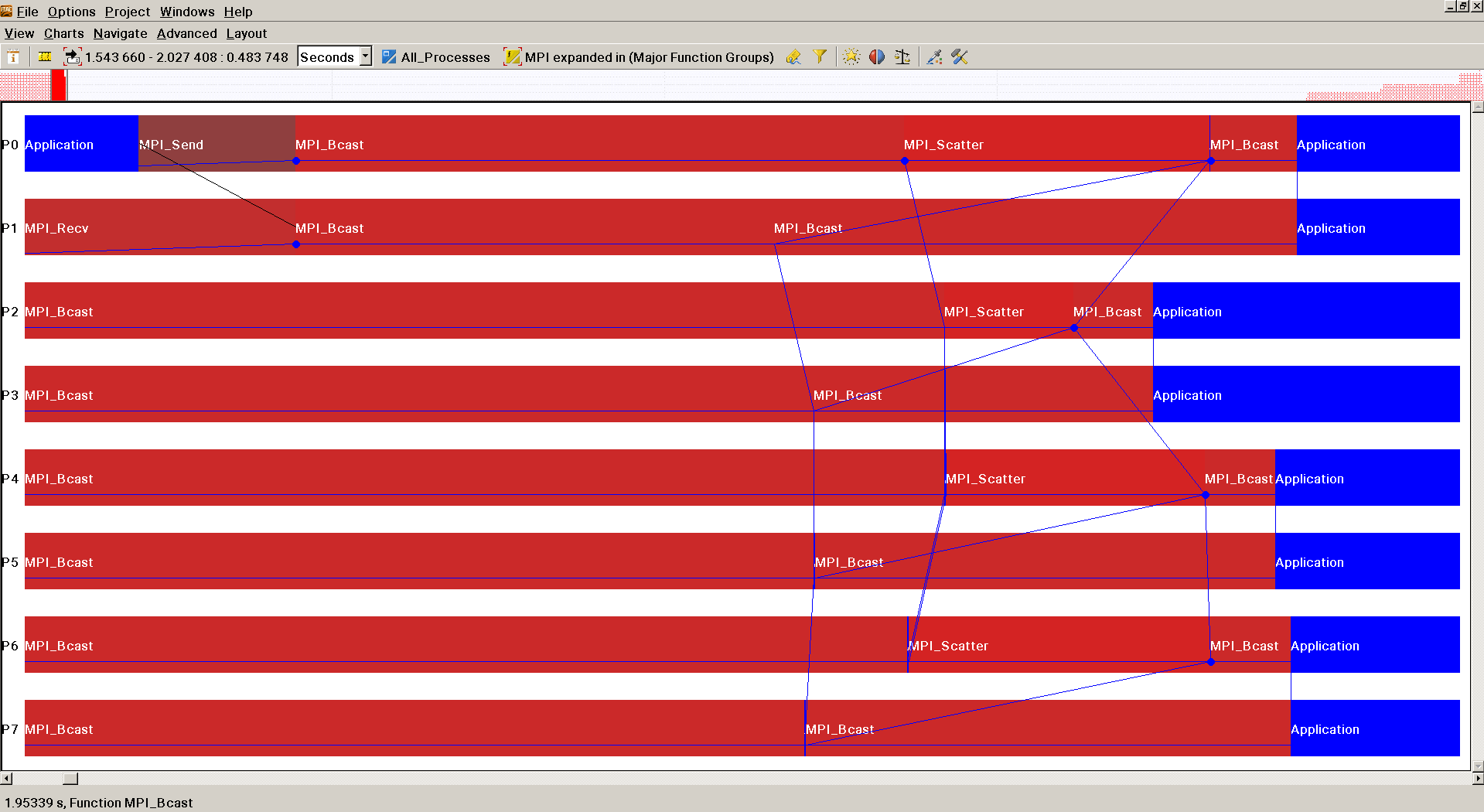
Рисунок 9. Профилирование программы, 4х2.

Рисунок 10. Профилирование программы, 4х2.

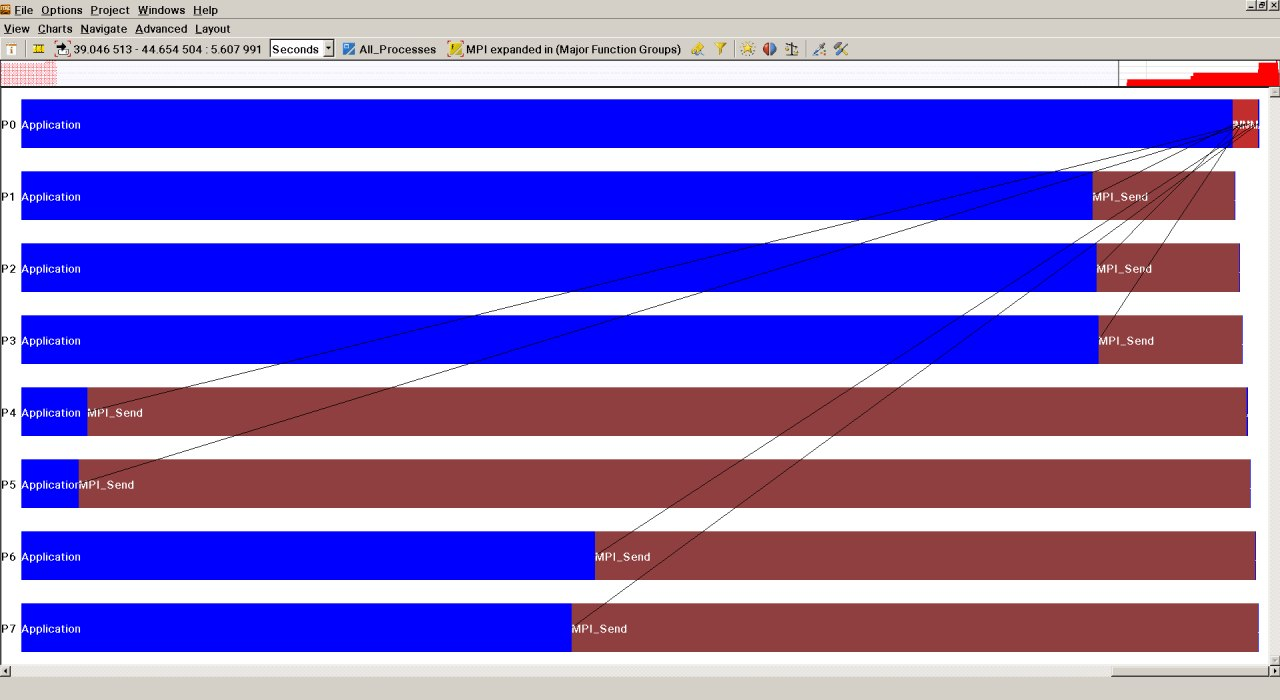


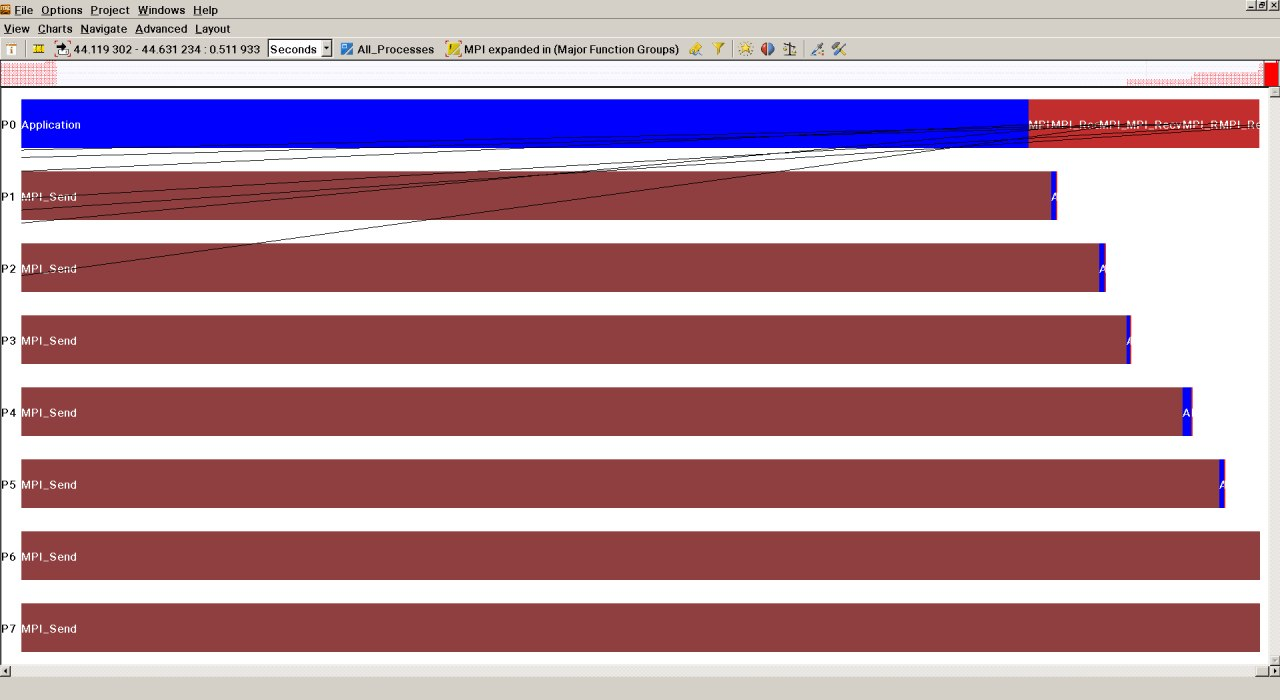
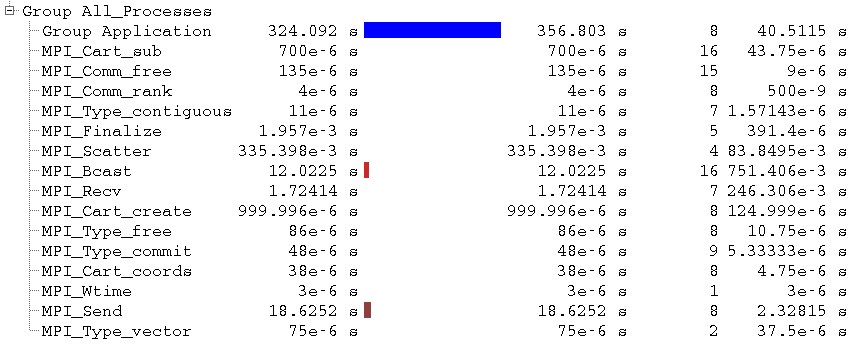
Рисунок 11. Профилирование программы, 4х2.

Рисунок 12. Профилирование программы, 4х2.



**Заключение**

В ходе выполнения лабораторной работы была изучена возможность создания новых топологий процессов в интерфейсе MPI и возможность создания пользовательских типов данных, а также влияние топологий на производительность.

**Приложение 1. Исходный код программы.**

#include <stdio.h>

#include <stdlib.h>

#include <mpi.h>

#define B\_COLUMN\_SEND 100

#define C\_MINOR\_SEND 200

void makeMatrixRandomInt(double \* matrix, int height, int width, int limit)

{

for (int i = 0; i < height; ++i)

{

for (int j = 0; j < width; ++j)

{

matrix[i \* width + j] = rand() % limit;

}

}

}

void printMatrix(double \* matrix, int height, int width)

{

for (int i = 0; i < height; ++i)

{

for (int j = 0; j < width; ++j)

{

printf("%6.2lf ", matrix[i \* width + j]);

}

printf("\n");

}

}

int main(int argc, char \* argv[])

{

srand(12345);

double \* A = NULL;

double \* B = NULL;

double \* C = NULL;

double \* partOfA = NULL;

int gridHeight = 1;

int gridWidth = 1;

int N = 1;

int M = 2;

int K = 3;

MPI\_Comm gridComm;

int ndims = 2;

int dims[2];

int periods[2] = {0, 0};

int reorder = 1;

MPI\_Comm rowsComm;

MPI\_Comm columnsComm;

int remainDimsRows [2] = {0, 1};

int remainDimsColumns[2] = {1, 0};

int rootRank = 0;

int rowRootRank = 0;

int columnRootRank = 0;

int gridRank;

int gridCoords[2];

double startTime;

int segmentAHeight;

int segmentBWidth;

if (argc > 1)

{

gridHeight = atoi(argv[1]);

}

if (argc > 2)

{

gridWidth = atoi(argv[2]);

}

if (argc > 3)

{

N = atoi(argv[3]);

}

if (argc > 4)

{

M = atoi(argv[4]);

}

if (argc > 5)

{

K = atoi(argv[5]);

}

dims[0] = gridHeight;

dims[1] = gridWidth;

segmentAHeight = N / gridHeight;

segmentBWidth = M / gridWidth;

MPI\_Init(&argc, &argv);

MPI\_Cart\_create(MPI\_COMM\_WORLD, ndims, dims, periods, reorder, &gridComm);

MPI\_Cart\_sub(gridComm, remainDimsRows, &rowsComm);

MPI\_Cart\_sub(gridComm, remainDimsColumns, &columnsComm);

MPI\_Comm\_rank(gridComm, &gridRank);

MPI\_Cart\_coords(gridComm, gridRank, ndims, gridCoords);

if (gridRank == rootRank)

{

printf(

"Started program with:\n"

"grid height = %d\n"

"grid width = %d\n"

"first matrix size = %d\n"

"first matrix width/second matrix height = %d\n"

"second matrix width = %d\n",

gridHeight, gridWidth, N, M, K

);

startTime = MPI\_Wtime();

A = (double \*) malloc(sizeof(double) \* N \* M);

B = (double \*) malloc(sizeof(double) \* M \* K);

makeMatrixRandomInt(A, N, M, 10);

makeMatrixRandomInt(B, M, K, 20);

MPI\_Datatype sendColumn;

MPI\_Type\_vector(M, segmentBWidth, K, MPI\_DOUBLE, &sendColumn);

MPI\_Type\_commit(&sendColumn);

for (int i = 1; i < gridWidth; ++i)

{

MPI\_Send(B + i \* segmentBWidth, 1, sendColumn, i, B\_COLUMN\_SEND + i, gridComm);

}

MPI\_Bcast(B, 1, sendColumn, columnRootRank, columnsComm);

MPI\_Type\_free(&sendColumn);

}

else

{

MPI\_Datatype recvContiguos;

MPI\_Type\_contiguous(M \* segmentBWidth, MPI\_DOUBLE, &recvContiguos);

MPI\_Type\_commit(&recvContiguos);

B = (double \*) malloc(sizeof(double) \* M \* segmentBWidth);

MPI\_Status status;

if (gridCoords[0] == 0)

{

MPI\_Recv(B, 1, recvContiguos, rootRank, B\_COLUMN\_SEND + gridRank, gridComm, &status);

}

MPI\_Bcast(B, 1, recvContiguos, columnRootRank, columnsComm);

MPI\_Type\_free(&recvContiguos);

}

partOfA = (double \*) malloc(sizeof(double) \* segmentAHeight \* M);

if (gridCoords[1] == 0)

{

MPI\_Scatter(A, segmentAHeight \* M, MPI\_DOUBLE, partOfA, segmentAHeight \* M, MPI\_DOUBLE, columnRootRank, columnsComm);

}

MPI\_Bcast(partOfA, segmentAHeight \* M, MPI\_DOUBLE, rowRootRank, rowsComm);

if (gridRank == rootRank)

{

C = (double \*) calloc(sizeof(double), N \* K);

}

else

{

C = (double \*) calloc(sizeof(double), segmentAHeight \* segmentBWidth);

}

if (gridRank == rootRank)

{

for (int i = 0; i < segmentAHeight; ++i)

{

for (int j = 0; j < M; ++j)

{

for (int k = 0; k < segmentBWidth; ++k)

{

C[i \* K + k] += partOfA[i \* M + j] \* B[j \* K + k];

}

}

}

}

else

{

for (int i = 0; i < segmentAHeight; ++i)

{

for (int j = 0; j < M; ++j)

{

for (int k = 0; k < segmentBWidth; ++k)

{

C[i \* segmentBWidth + k] += partOfA[i \* M + j] \* B[j \* segmentBWidth + k];

}

}

}

}

if (gridRank == rootRank)

{

MPI\_Datatype minor;

MPI\_Type\_vector(segmentAHeight, segmentBWidth, K, MPI\_DOUBLE, &minor);

MPI\_Type\_commit(&minor);

MPI\_Status status;

for (int i = 0; i < gridHeight; ++i)

{

for (int j = 0; j < gridWidth; ++j)

{

if (i != 0 || j != 0)

{

MPI\_Recv(C + i \* segmentAHeight \* K + j \* segmentBWidth, 1, minor, i \* gridWidth + j, C\_MINOR\_SEND + i \* gridWidth + j, gridComm, &status);

}

}

}

MPI\_Type\_free(&minor);

}

else

{

MPI\_Send(C, segmentAHeight \* segmentBWidth, MPI\_DOUBLE, rootRank, C\_MINOR\_SEND + gridRank, gridComm);

}

if (gridRank == rootRank) {

printf("Answer found for %lf secs\n", MPI\_Wtime() - startTime);

free(A);

}

free(partOfA);

free(B);

free(C);

MPI\_Comm\_free(&gridComm);

MPI\_Comm\_free(&columnsComm);

MPI\_Comm\_free(&rowsComm);

MPI\_Finalize();

return 0;

}