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

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 A на строки и матрицы B на столбцы с последующим получением миноров матрицы C 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 C в том же (0; 0) процессе.

4. 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.

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

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

Рисунок 1. Профилирование программы, 2x4.

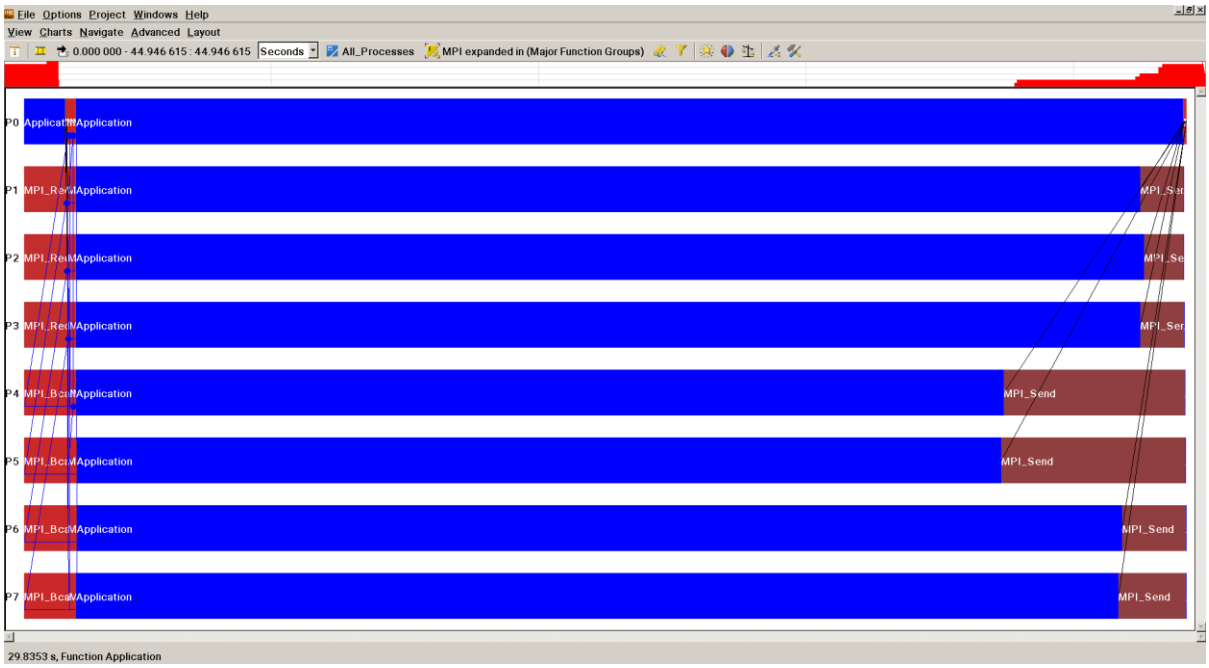

Рисунок 2. Профилирование программы, 2x4.

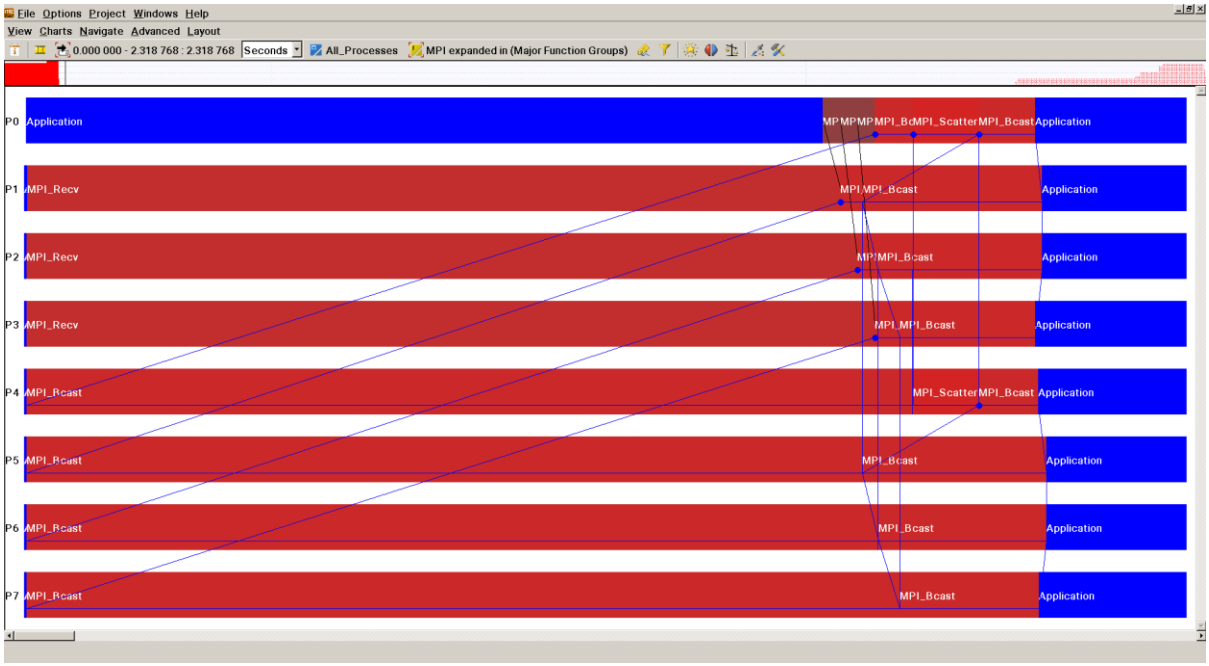

Рисунок 3. Профилирование программы, 2x4.

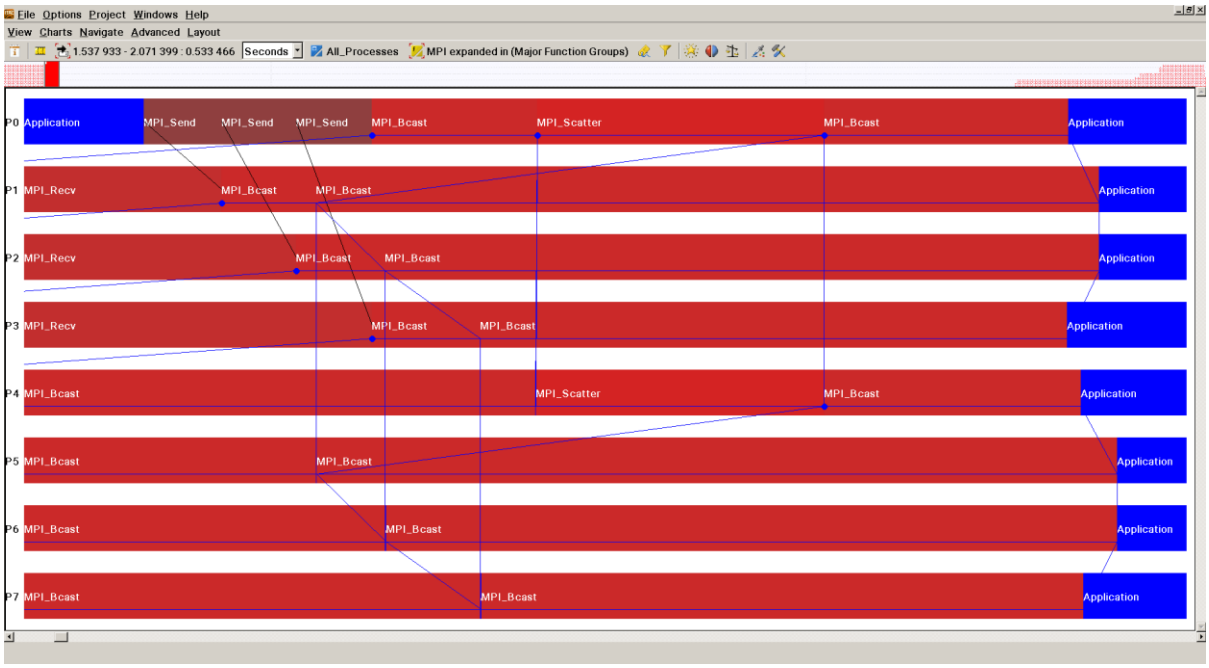

Рисунок 4. Профилирование программы, 2x4.

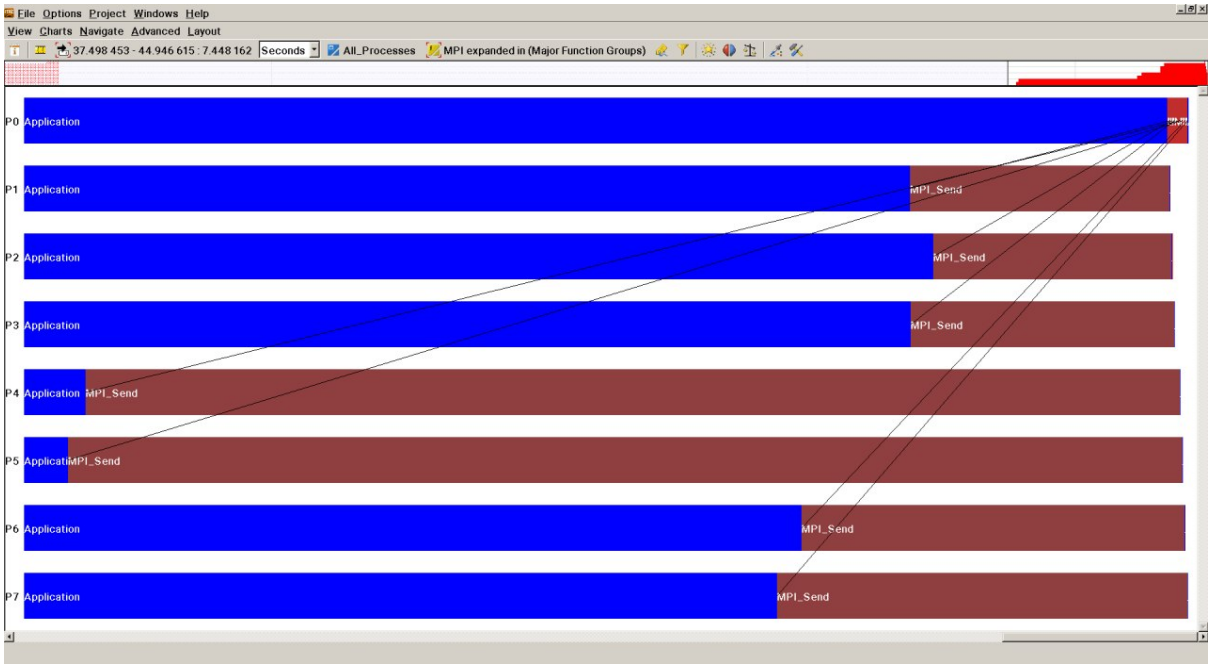

Рисунок 5. Профилирование программы, 2x4.

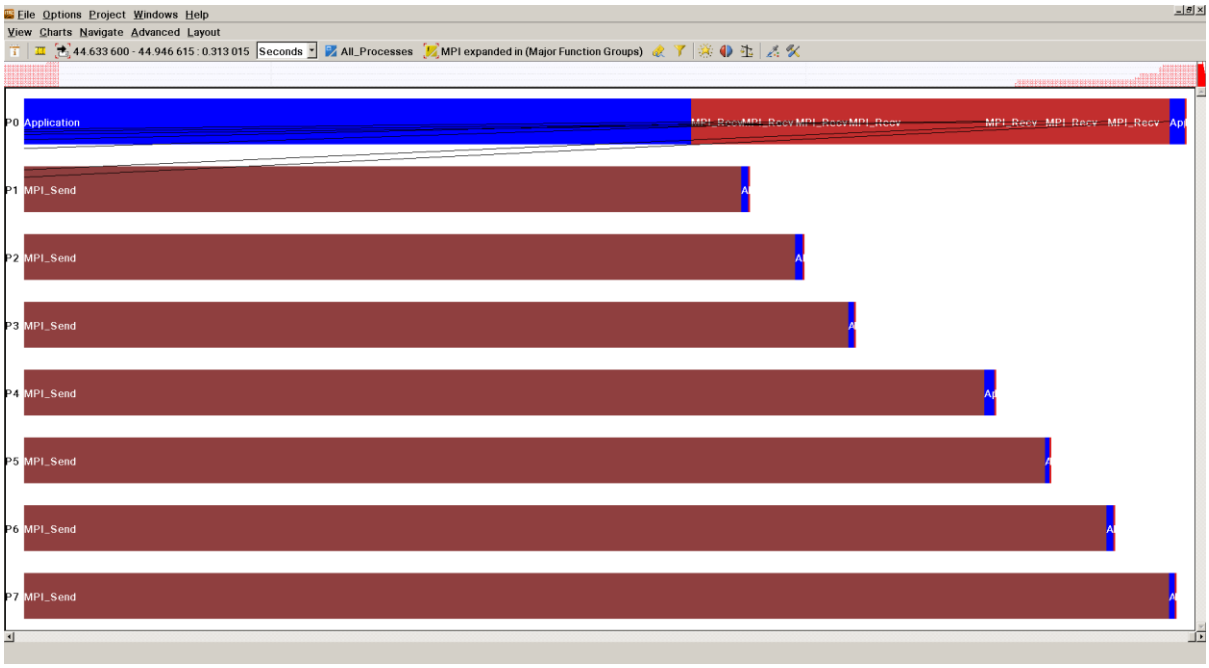

Рисунок 6. Профилирование программы, 2x4.

Group All_Processes					
Group Application	320.351 s		359.148 s	8	40.0439 s
MPI_Cart_sub	693e-6 s		693e-6 s	16	43.3125e-6 s
MPI_Comm_free	204e-6 s		204e-6 s	24	8.5e-6 s
MPI_Comm_rank	2e-6 s		2e-6 s	8	250e-9 s
MPI_Type_contiguous	10e-6 s		10e-6 s	7	1.42857e-6 s
MPI_Finalize	3.012e-3 s		3.012e-3 s	8	376.5e-6 s
MPI_Scatter	263.761e-3 s		263.761e-3 s	2	131.88e-3 s
MPI_Bcast	9.24721 s		9.24721 s	16	577.951e-3 s
MPI_Recv	5.09938 s		5.09938 s	10	509.938e-3 s
MPI_Cart_create	1.151e-3 s		1.151e-3 s	8	143.875e-6 s
MPI_Type_free	96e-6 s		96e-6 s	9	10.6667e-6 s
MPI_Type_commit	41e-6 s		41e-6 s	9	4.55556e-6 s
MPI_Cart_coords	39e-6 s		39e-6 s	8	4.875e-6 s
MPI_Wtime	5e-6 s		5e-6 s	2	2.5e-6 s
MPI_Send	24.1815 s		24.1815 s	10	2.41815 s
MPI_Type_vector	77e-6 s		77e-6 s	2	38.5e-6 s

Рисунок 7. Профилирование программы, 4x2.

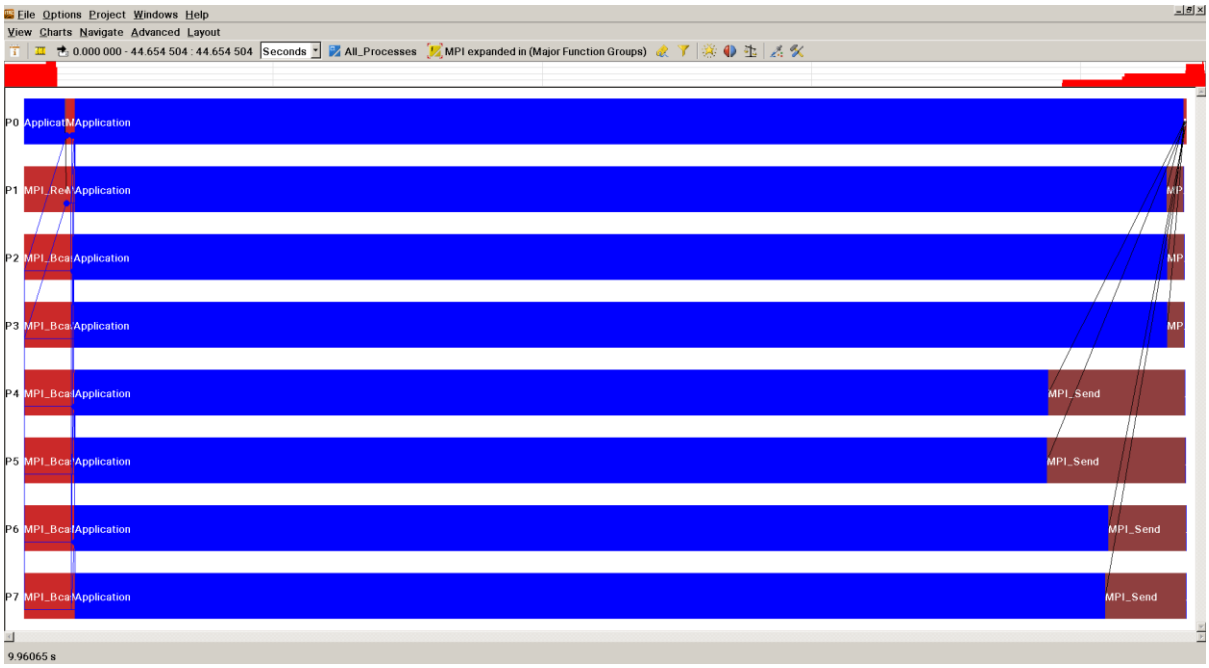

Рисунок 8. Профилирование программы, 4x2.

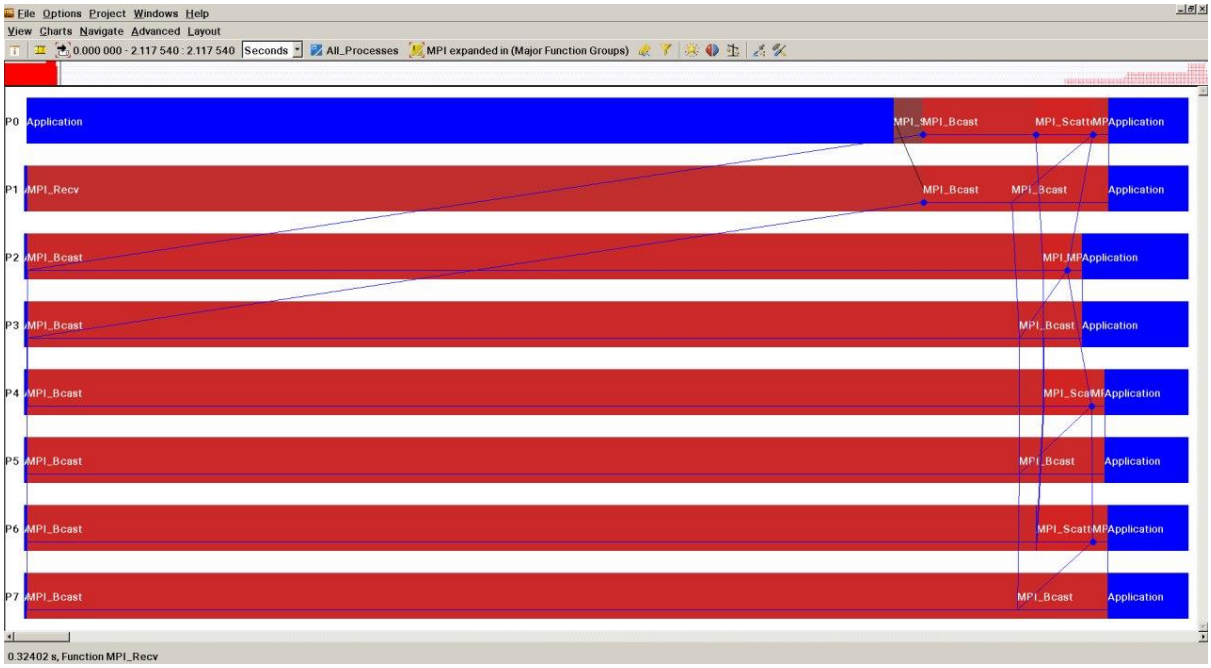

Рисунок 9. Профилирование программы, 4x2.

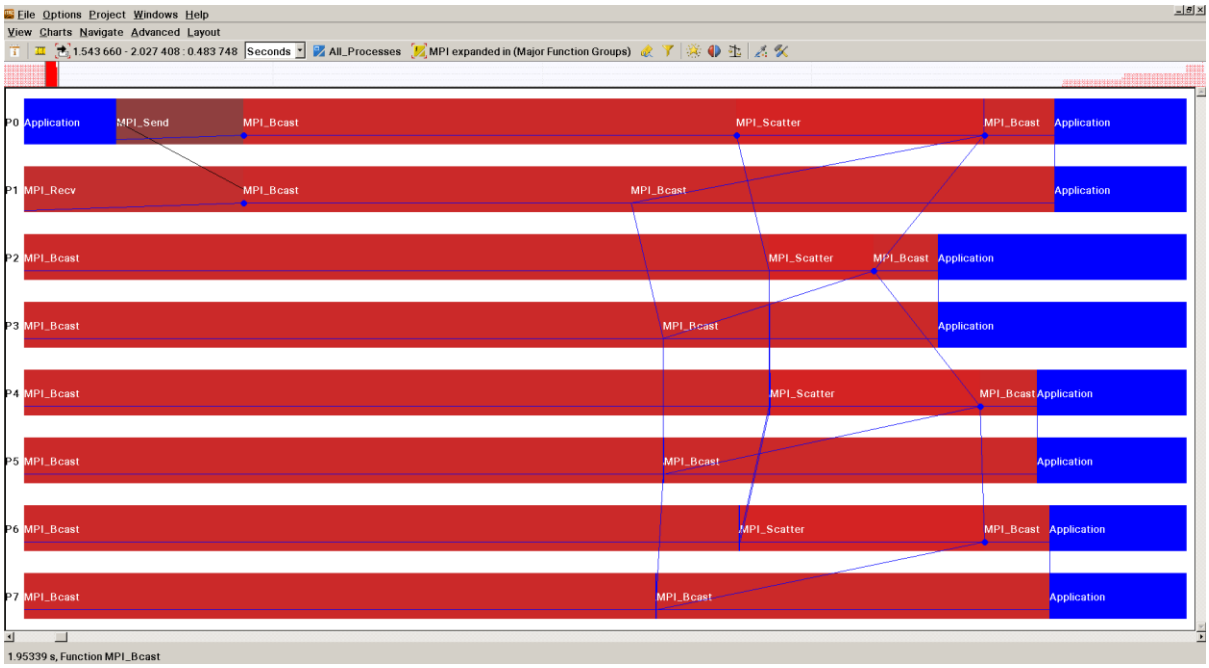

Рисунок 10. Профилирование программы, 4x2.

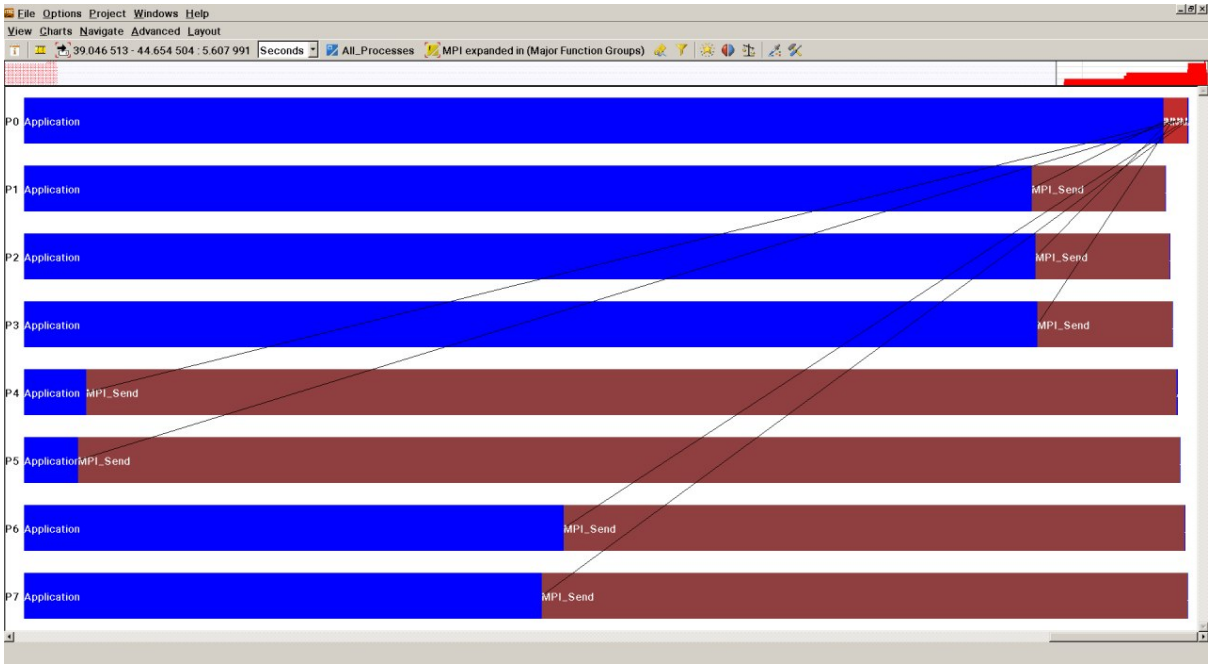

Рисунок 11. Профилирование программы, 4x2.

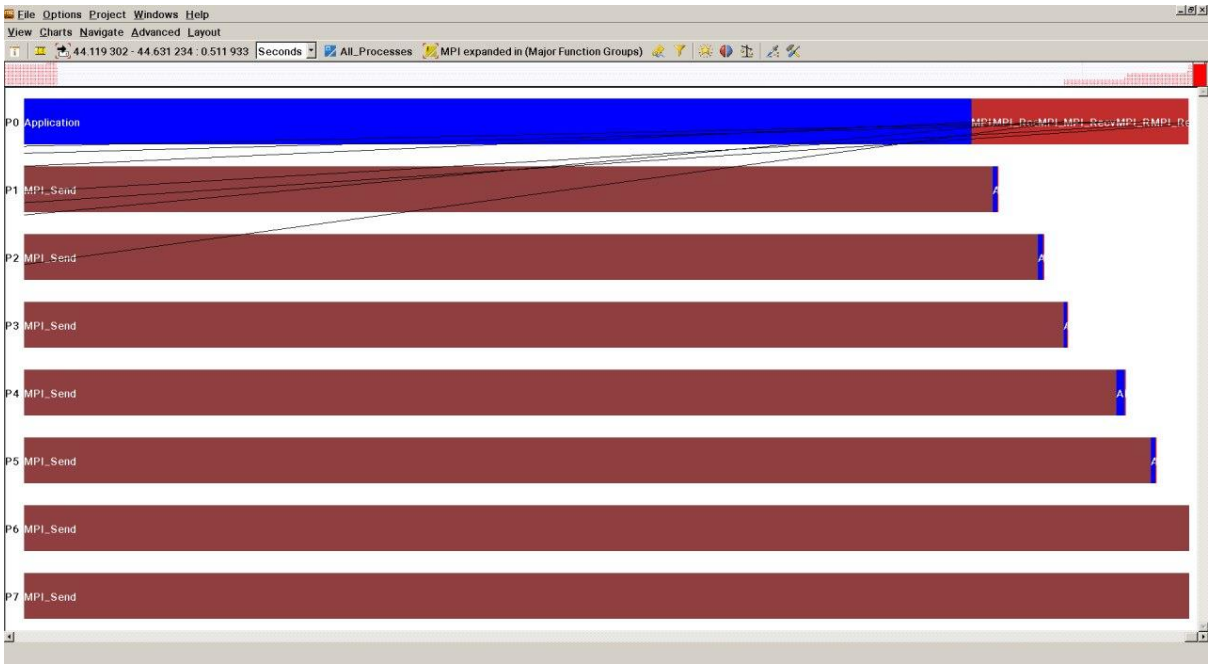

Рисунок 12. Профилирование программы, 4x2.

Group All_Processes				
Group Application	324.092 s	356.803 s	8	40.5115 s
MPI_Cart_sub	700e-6 s	700e-6 s	16	43.75e-6 s
MPI_Comm_free	135e-6 s	135e-6 s	15	9e-6 s
MPI_Comm_rank	4e-6 s	4e-6 s	8	500e-9 s
MPI_Type_contiguous	11e-6 s	11e-6 s	7	1.57143e-6 s
MPI_Finalize	1.957e-3 s	1.957e-3 s	5	391.4e-6 s
MPI_Scatter	335.398e-3 s	335.398e-3 s	4	83.8495e-3 s
MPI_Bcast	12.0225 s	12.0225 s	16	751.406e-3 s
MPI_Recv	1.72414 s	1.72414 s	7	246.306e-3 s
MPI_Cart_create	999.996e-6 s	999.996e-6 s	8	124.999e-6 s
MPI_Type_free	86e-6 s	86e-6 s	8	10.75e-6 s
MPI_Type_commit	48e-6 s	48e-6 s	9	5.33333e-6 s
MPI_Cart_coords	38e-6 s	38e-6 s	8	4.75e-6 s
MPI_Wtime	3e-6 s	3e-6 s	1	3e-6 s
MPI_Send	18.6252 s	18.6252 s	8	2.32815 s
MPI_Type_vector	75e-6 s	75e-6 s	2	37.5e-6 s

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;
}
```