###### 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 РЕШЁТКЕ

Студентки 2 курса, группы 21205

**Евдокимовой Дари Евгеньевны**

Направление 09.03.01 – «Информатика и вычислительная техника»

Преподаватель:

Кандидат технических наук, доцент

А.Ю. Власенко

Новосибирск 2023

**СОДЕРЖАНИЕ**

[​ ЦЕЛЬ 3](#__RefHeading___Toc1471_2204003058)

[​ ЗАДАНИЕ 3](#__RefHeading___Toc1473_2204003058)

[​ ОПИСАНИЕ РАБОТЫ 4](#__RefHeading___Toc1475_2204003058)

[​ ЗАКЛЮЧЕНИЕ 6](#__RefHeading___Toc1477_2204003058)

[Приложение 1. Листинг последовательной программы 7](#__RefHeading___Toc1479_2204003058)

[Приложение 2. Листинг параллельной программы 10](#__RefHeading___Toc1481_2204003058)

[Приложение 3. Проверка корректности вычислений параллельной программы 15](#__RefHeading___Toc1483_2204003058)

[​ Приложение 4. Скрипт для запуска параллельной программы 16](#__RefHeading___Toc1485_2204003058)

[​ Приложение 5. Результаты замеров выполнения работы на кластере 17](#__RefHeading___Toc1487_2204003058)

[​ Приложение 6. График зависимости времени от размера решетки 18](#__RefHeading___Toc1489_2204003058)

[​ Приложение 7. Скрипты для решеток размером 2x4 и 4x2 19](#__RefHeading___Toc1491_2204003058)

[​ Приложение 8. Скрины из traceanalyzer решетки размером 2x4 20](#__RefHeading___Toc1493_2204003058)

[​ Приложение 9. Скрины из traceanalyzer решетки размером 4x2 22](#__RefHeading___Toc1495_2204003058)

# ЦЕЛЬ

1. Написание программы, которая реализует умножение матрицы на матрицу с помощью средств MPI.
2. Ознакомление с производными типами данных в MPI и их применение на практике.
3. Освоение концепции MPI-коммуникаторов и декартовых топологий.

# ЗАДАНИЕ

1. Написать параллельную программу, которая реализует умножение матрицы на матрицу с помощью средств MPI.

1. Исследовать производительность параллельной программы при фиксированном размере матрицы в зависимости от размера решетки: 2x12, 3x8, 4x6, 6x4, 8x3, 12x2. Размер матриц подобрать таким образом, чтобы худшее из времен данного набора было не менее 30 сек.
2. Выполнить профилирование программы с помощью jumpshot или ITAC (Intel Trace Analyzer and Collecter) при использовании 8-и ядер с решетками 2x4 и 4x2.
3. Составить отчет, содержащий исходные коды разработанных программ и построенные графики.

# ОПИСАНИЕ РАБОТЫ

1. Был создан файл *sequential.cpp,* в котором написана последовательная программа умножения матрицы на матрицу. Полный компилируемый листинг программы см. Приложение 1.
2. Был создан файл *parallel.cpp*, в котором была реализована параллельная программа умножения матрицы на матрицу с помощью MPI. Полный компилируемый листинг программы см. Приложение 2.
3. Корректность работы программы была проверена c помощью сайта <https://matrixcalc.org/>. Результаты проверки корректности см. Приложение 3.
4. Результаты проверки корректности см. Приложение 2. Команда для компиляции (не на кластере!):

*mpicxx -o parallel parallel.cpp*

*mpirun –n [p1\*p2] ./parallel [n1] [n2] [n3] [p1] [p2]*

На вход программы подается 5 чисел:

[n1] – число строк 1й матрицы

[n2] – число столбцов 1й матрицы = число строк 2й матрицы

[n3] – число столбцов 3й матрицы

[p1] – число строк в решетке

[p2] – число столбцов в решетке

Число процессов равно p1\*p2.

1. Перейдём к работе на кластере. Количество процессов во всех измерениях равно 24.   
   Размер матриц, при котором время программы составляет не менее 30сек:   
   6168 3000 5184.
2. Скрипт (run\_parallel.sh) для запуска программы parallel.cpp смотреть в Приложении 4.
3. Полученные результаты измерения времени добавлены в Приложение 5 и в Таблицу 1.

Таблица1. Результаты измерений времени

|  |  |
| --- | --- |
| Размер решетки [строка x столбец] | Время выполнения, сек |
| 2 x 12 | 61,3384 |
| 3 x 8 | 57,5897 |
| 4 x 6 | 62,3831 |
| 6 x 4 | 65,0422 |
| 8 x 3 | 64,3465 |
| 12 x 2 | 67,2499 |

1. График зависимости времени от размера решетки смотреть в Приложении 6.
2. Было проведено профилирование программы на 8 процессах с размерами решёток  
    2x4 (скрипт go\_trace2\_4.sh, см. Приложение 7) и 4x2 (скрипт go\_trace4\_2.sh, см. Приложение 7).

Команда для запуска itac-а: (заходить на кластер с флагом ‘-X’)

*traceanalyzer parall.stf*

Скрины из traceanalyzer для решетки размером

- 2x4 смотреть в Приложении 8;

- 4x2 смотреть в Приложении 9.

1. Сравним результаты профилирования решеток размером 2x4 и 4x2:

В

|  |  |  |
| --- | --- | --- |
|  | Решетка 2х4 | Решетка 4х2 |
| Коллективная операция Scatter | Вызывается в 0 и 4 процессах, т. к. матрица А «разрезается» по строкам, а потом передается на столбцы решетки (их 2). | Вызывается в 0, 2, 4, 6 процессах, т. к. матрица А «разрезается» по строкам, а потом передается на столбцы решетки (их 4). |
| Операция типа точка-точка MPI\_Send() | Нулевой процесс отправляет данные всем остальным процессам (кроме самого себя). | |
| Операция типа точка-точка MPI\_Recv() | Для матрицы В и С: | |
| Все процессы, кроме нулевого, отправляют данные в нулевой процесс.  Функция вызывается 3 раза, т.к столбцов 4, а нулевой процесс сам себя не может принимать. | Все процессы, кроме нулевого, отправляют данные в нулевой процесс.  Функция вызывается 1 раза, т.к столбцов 2, , а нулевой процесс сам себя не может принимать. |
| Коллективная операция MPI\_Bcast | Нулевой процесс отправляет данные всем остальным процессам. | |

# ЗАКЛЮЧЕНИЕ

В ходе работы мы смогли «распараллелить» программу для умножения матрицы на матрицу с помощью декартовых топологий и производных типов данных.

При анализе профилирования можно заключить, что время, затрачиваемое на вычисления значительно больше времени, затрачиваемого на распределение и сбор данных.

# Приложение 1. Листинг последовательной программы

#include <cstdio>

#include <iostream>

struct MyMatrix {

  double \*data = nullptr;

  size\_t colmns;

  size\_t rows;

  MyMatrix(size\_t rows, size\_t colmns) {

    data = new double[colmns \* rows];

    this->rows = rows;

    this->colmns = colmns;

  }

};

void initMatrix(MyMatrix matrix) {

  for (size\_t i = 0; i < matrix.rows; ++i) {

    for (size\_t j = 0; j < matrix.colmns; ++j) {

      matrix.data[i \* matrix.colmns + j] = rand() % 100 + 15;

    }

  }

}

void freeMatrix(MyMatrix matrix) { delete[] matrix.data; }

void printMatrix(MyMatrix matrix) {

  for (size\_t i = 0; i < matrix.rows; ++i) {

    for (size\_t j = 0; j < matrix.colmns; ++j) {

      std::cout << matrix.data[i \* matrix.colmns + j] << "\t";

    }

    std::cout << std::endl;

  }

}

void multimplyMtrices(MyMatrix m1, MyMatrix m2, MyMatrix mRes) {

  for (size\_t i = 0; i < m1.rows; i++) {

    for (size\_t j = 0; j < m2.colmns; j++) {

      for (size\_t k = 0; k < m1.colmns; k++) {

        mRes.data[i \* m2.colmns + j] +=

            m1.data[i \* m1.colmns + k] \* m2.data[k \* m2.colmns + j];

      }

    }

  }

}

void testWithRandomValues(MyMatrix m1, MyMatrix m2) {

  initMatrix(m1);

  initMatrix(m2);

}

void testWithConstantValues(MyMatrix m1, MyMatrix m2) {

  for (size\_t i = 0; i < m1.colmns \* m1.rows; ++i) {

    m1.data[i] = i + 10;

  }

  for (size\_t j = 0; j < m2.colmns \* m2.rows; ++j) {

    m2.data[j] = j + 20;

  }

}

int main(int argc, char \*\*argv) {

  if (argc != 4) {

    std::cout

        << "Error! Enter rows and columns amount for 2 matrixes!"

        << std::endl;

    return 0;

  }

  const size\_t dim1 = atoi(argv[1]); // m1.rows

  const size\_t dim2 = atoi(argv[2]); // m1.columns = m2.rows

  const size\_t dim3 = atoi(argv[3]); // m2.columns

  MyMatrix m1 = MyMatrix(dim1, dim2);

  MyMatrix m2 = MyMatrix(dim2, dim3);

  // testWithRandomValues(m1, m2);

  testWithConstantValues(m1, m2);

  std::cout << "Matrix1 : " << std::endl;

  printMatrix(m1);

  std::cout << "Matrix2 : " << std::endl;

  printMatrix(m2);

  MyMatrix mRes = MyMatrix(dim1, dim3);

  if (m1.colmns != m2.rows || m1.rows != mRes.rows ||

      m2.colmns != mRes.colmns) {

    std::cerr

        << "Error! You've entered a wrong dimention to   matrices"

        << std::endl;

    return 0;

  }

  multimplyMtrices(m1, m2, mRes);

  std::cout << "Matrix mRes : " << std::endl;

  printMatrix(mRes);

  freeMatrix(m1);

  freeMatrix(m2);

  freeMatrix(mRes);

}

# Приложение 2. Листинг параллельной программы

#include <iostream>

#include <mpi.h>

static int dimOfAnyGrid = 2;

static int X\_AXIS = 0;

static int Y\_AXIS = 1;

struct MyMatrix {

  size\_t row;

  size\_t column;

  double \*data = nullptr;

  MyMatrix(size\_t row, size\_t column) {

    this->row = row;

    this->column = column;

    data = new double[row \* column]();

  }

};

void freeMatrix(MyMatrix matrix) { delete[] matrix.data; }

void initMatrix(MyMatrix matrix) {

  for (size\_t i = 0; i < matrix.row; ++i) {

    for (size\_t j = 0; j < matrix.column; ++j) {

      matrix.data[i \* matrix.column + j] = rand() % 100 + 15;

    }

  }

}

void multimplyMtrices(MyMatrix m1, MyMatrix m2, MyMatrix mRes) {

  for (size\_t i = 0; i < m1.row; i++) {

    for (size\_t j = 0; j < m2.column; j++) {

      for (size\_t k = 0; k < m1.column; k++) {

        mRes.data[i \* m2.column + j] +=

            m1.data[i \* m1.column + k] \* m2.data[k \* m2.column + j];

      }

    }

  }

}

void printMat(MyMatrix matrix) {

  for (size\_t i = 0; i < matrix.row; i++) {

    for (size\_t j = 0; j < matrix.column; j++) {

      std::cout << matrix.data[i \* matrix.column + j] << " ";

    }

    std::cout << std::endl;

  }

}

int main(int argc, char \*\*argv) {

  if (argc != 6) {

    std::cout

        << "Error! Enter rowsInC and columns amount for 2 matrixes!"

        << std::endl;

    return 0;

  }

  // srand(time(nullptr)); // just for random vales each time

  srand(0); // for testint the same values in one task

  const size\_t dim1 = atoi(argv[1]);

  const size\_t dim2 = atoi(argv[2]);

  const size\_t dim3 = atoi(argv[3]);

  const int rowsGrid = atoi(argv[4]);    // height of 2d grid

  const int columnsGrid = atoi(argv[5]); // weight of 2d grid

  MPI\_Init(&argc, &argv);

  int amountOfProcs, rankOfCurrProc;

  MPI\_Comm\_size(MPI\_COMM\_WORLD, &amountOfProcs);

  MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rankOfCurrProc);

  if (amountOfProcs != rowsGrid \* columnsGrid) {

    if (rankOfCurrProc == 0) {

      std::cout << "Bad input! Amount of processes should be equal "

                   "to rowsGrid \* columnsGrid"

                << std::endl;

    }

    MPI\_Finalize();

    return 0;

  }

  int dimSize[] = {rowsGrid, columnsGrid};

  int periods[] = {0};

  MPI\_Comm commGrid;

  double startt = MPI\_Wtime();

  MPI\_Cart\_create(MPI\_COMM\_WORLD, dimOfAnyGrid, dimSize, periods, 0,

                  &commGrid);

  int coordsOfCurrProc[dimOfAnyGrid];

  MPI\_Cart\_coords(commGrid, rankOfCurrProc, dimOfAnyGrid,

                  coordsOfCurrProc);

  MPI\_Comm commRow;

  int remainX[] = {X\_AXIS, Y\_AXIS};

  MPI\_Cart\_sub(commGrid, remainX, &commRow);

  MPI\_Comm commColumn;

  int remainY[] = {Y\_AXIS, X\_AXIS};

  MPI\_Cart\_sub(commGrid, remainY, &commColumn);

  MyMatrix A(dim1, dim2);

  MyMatrix B(dim2, dim3);

  MyMatrix C(dim1, dim3);

  if (rankOfCurrProc == 0) {

    initMatrix(A);

    initMatrix(B);

    // std::cout << "A[" << A.row << " x " << A.column << "]"

    //           << std::endl;

    // printMat(A);

    // std::cout << "B[" << B.row << " x " << B.column << "]"

    //           << std::endl;

    // printMat(B);

  }

  MyMatrix partA(dim1 / dimSize[X\_AXIS], dim2);

  if (coordsOfCurrProc[Y\_AXIS] == 0) {

    MPI\_Scatter(A.data, partA.row \* partA.column, MPI\_DOUBLE,

                partA.data, partA.row \* partA.column, MPI\_DOUBLE, 0,

                commColumn);

  }

  MPI\_Bcast(partA.data, partA.row \* partA.column, MPI\_DOUBLE, 0,

            commRow);

  MyMatrix partB(dim2, dim3 / dimSize[Y\_AXIS]);

  MPI\_Datatype bSendType; // type of columns

  MPI\_Type\_vector(dim2, partB.column, dim3, MPI\_DOUBLE, &bSendType);

  MPI\_Type\_commit(&bSendType);

  MPI\_Datatype bRecvType; // type of only ONE (!!!) column

  MPI\_Type\_vector(1, partB.row \* partB.column, 0, MPI\_DOUBLE,

                  &bRecvType);

  MPI\_Type\_commit(&bRecvType);

  if (coordsOfCurrProc[X\_AXIS] == 0 &&

      coordsOfCurrProc[Y\_AXIS] == 0) {

    for (int i = 1; i < columnsGrid; i++) {

      MPI\_Send(B.data + partB.column \* i, 1, bSendType, i, 11,

               commRow);

    }

    for (int i = 0; i < B.row; i++) {

      for (int j = 0; j < partB.column; j++) {

        partB.data[i \* partB.column + j] = B.data[i \* B.column + j];

      }

    }

  }

  else if (coordsOfCurrProc[X\_AXIS] == 0) {

    MPI\_Recv(partB.data, 1, bRecvType, 0, 11, commRow,

             MPI\_STATUS\_IGNORE);

  }

  MPI\_Bcast(partB.data, dim2 \* partB.column, MPI\_DOUBLE, 0,

            commColumn);

  MyMatrix partC(dim1 / dimSize[X\_AXIS], dim3 / dimSize[Y\_AXIS]);

  multimplyMtrices(partA, partB, partC);

  MPI\_Datatype cRecvType;

  MPI\_Type\_vector(partC.row, partC.column, dim3, MPI\_DOUBLE,

                  &cRecvType);

  MPI\_Type\_commit(&cRecvType);

  int offset[amountOfProcs];

  for (int procRank = 0; procRank < amountOfProcs; procRank++) {

    MPI\_Cart\_coords(commGrid, procRank, dimOfAnyGrid,

                    coordsOfCurrProc);

    offset[procRank] =

        coordsOfCurrProc[Y\_AXIS] \* partC.column +

        coordsOfCurrProc[X\_AXIS] \* partC.row \* C.column;

  }

  if (rankOfCurrProc == 0) {

    for (int i = 0; i < partC.row; i++) {

      for (int j = 0; j < partC.column; j++) {

        C.data[i \* C.column + j] = partC.data[i \* partC.column + j];

      }

    }

    for (int i = 1; i < amountOfProcs; i++) {

      MPI\_Recv(C.data + offset[i], 1, cRecvType, i, 0, MPI\_COMM\_WORLD,

               MPI\_STATUS\_IGNORE);

    }

  } else {

    MPI\_Send(partC.data, partC.row \* partC.column, MPI\_DOUBLE, 0, 0,

             MPI\_COMM\_WORLD);

  }

  MPI\_Type\_free(&bSendType);

  MPI\_Type\_free(&cRecvType);

  MPI\_Comm\_free(&commGrid);

  MPI\_Comm\_free(&commColumn);

  MPI\_Comm\_free(&commRow);

  double endt = MPI\_Wtime();

  if (rankOfCurrProc == 0) {

    // std::cout << "C[" << C.row << " x " << C.column << "]"

    //           << std::endl;

    // printMat(C);

    std::cout << "Time taken: " << endt - startt << " sec"

              << std::endl;

  }

  freeMatrix(A);

  freeMatrix(B);

  freeMatrix(C);

  freeMatrix(partA);

  freeMatrix(partB);

  freeMatrix(partC);

  MPI\_Finalize();

  return 0;

}

# Приложение 3. Проверка корректности вычислений параллельной программы

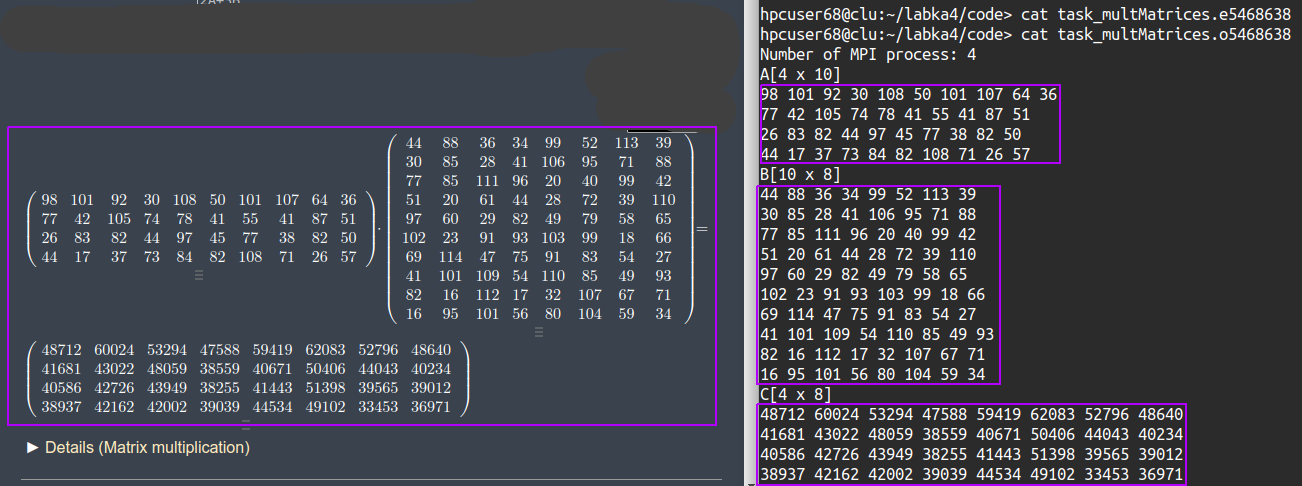


Рис. 1. Умножение матриц размером 4x10, 10x8

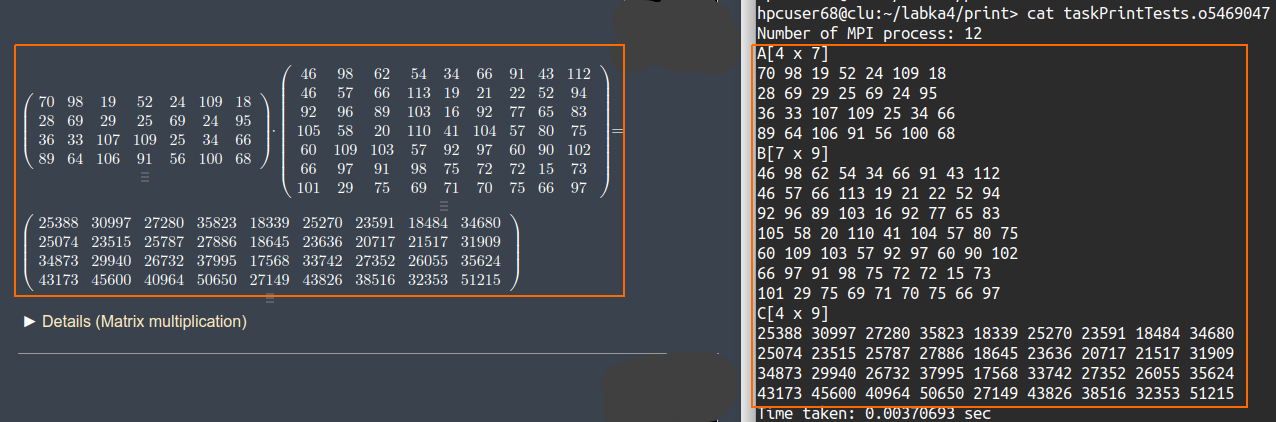


Рис. 2. Умножение матриц размером 4x7, 7x9

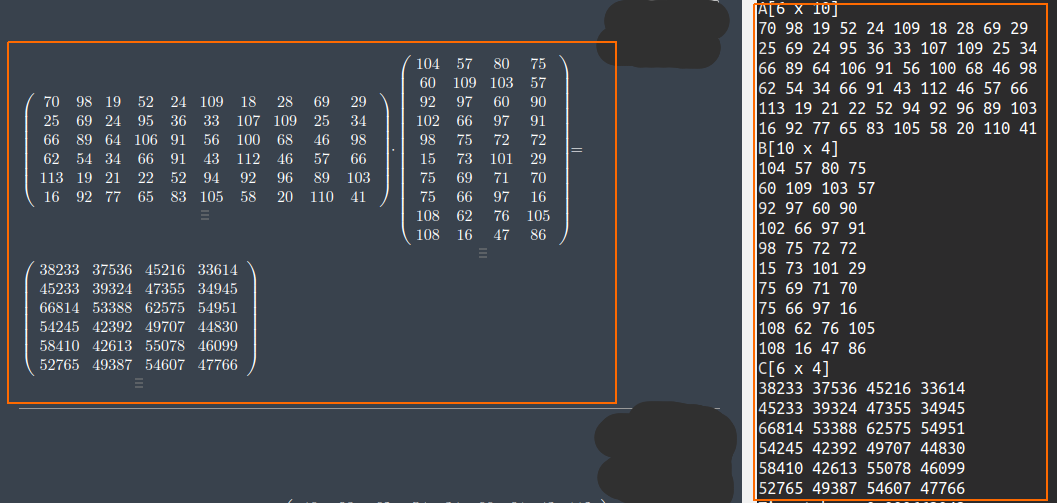


Рис. 3. Умножение матриц размером 6x10, 10x4

# Приложение 4. Скрипт для запуска параллельной программы

1. Файл run\_parallel.sh:

#!/bin/bash

#PBS -l walltime=00:10:00

#PBS -l select=2:ncpus=12:mpiprocs=12:mem=10000m

#PBS -m n

#PBS -N task\_multMatrices

cd $PBS\_O\_WORKDIR

MPI\_NP=$(wc -l $PBS\_NODEFILE | awk '{ print $1 }')

echo "Number of MPI process: $MPI\_NP"

mpicxx parallel.cpp -o parallel -std=c++11

echo "2 x 12 :"

mpirun -hostfile $PBS\_NODEFILE -np $MPI\_NP ./parallel 6168 3000 5184 2 12

echo "3 x 8 :"

mpirun -hostfile $PBS\_NODEFILE -np $MPI\_NP ./parallel 6168 3000 5184 3 8

echo "4 x 6 :"

mpirun -hostfile $PBS\_NODEFILE -np $MPI\_NP ./parallel 6168 3000 5184 4 6

echo "6 x 4 :"

mpirun -hostfile $PBS\_NODEFILE -np $MPI\_NP ./parallel 6168 3000 5184 6 4

echo "8 x 3 :"

mpirun -hostfile $PBS\_NODEFILE -np $MPI\_NP ./parallel 6168 3000 5184 8 3

echo "12 x 2 :"

mpirun -hostfile $PBS\_NODEFILE -np $MPI\_NP ./parallel 6168 3000 5184 12 2

# Приложение 5. Результаты замеров выполнения работы на кластере

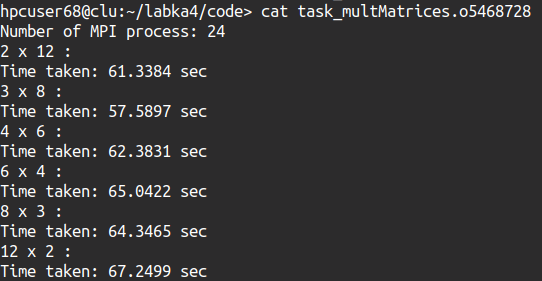


Рис.1. Результаты замеров выполнения работы на кластере

# Приложение 6. График зависимости времени от размера решетки



Рис. 1. График зависимости времени от размера решетки

# Приложение 7. Скрипты для решеток размером 2x4 и 4x2

##Cкрипт go\_trace2\_4.sh

#!/bin/bash

#PBS -l walltime=00:05:00

#PBS -l select=1:ncpus=8:mpiprocs=8:mem=10000m,place=scatter:exclhost

#PBS -m n

#PBS -N trace\_2x4\_\_analyzeMatrix

cd $PBS\_O\_WORKDIR

MPI\_NP=$(wc -l $PBS\_NODEFILE | awk '{ print $1 }')

echo "Number of MPI process: $MPI\_NP"

mpirun -trace -machinefile $PBS\_NODEFILE -np $MPI\_NP -perhost 2 ./parallel 5328 1300 4968 2 4

--------------------------------------

##Скрипт go\_trace4\_2.sh

#!/bin/bash

#PBS -l walltime=00:05:00

#PBS -l select=1:ncpus=8:mpiprocs=8:mem=10000m,place=scatter:exclhost

#PBS -m n

#PBS -N trace\_4x2\_analyzeMatrix

cd $PBS\_O\_WORKDIR

MPI\_NP=$(wc -l $PBS\_NODEFILE | awk '{ print $1 }')

echo "Number of MPI process: $MPI\_NP"

mpirun -trace -machinefile $PBS\_NODEFILE -np $MPI\_NP -perhost 2 ./parallel 5328 1300 4968 4 2

# Приложение 8. Скрины из traceanalyzer решетки размером 2x4

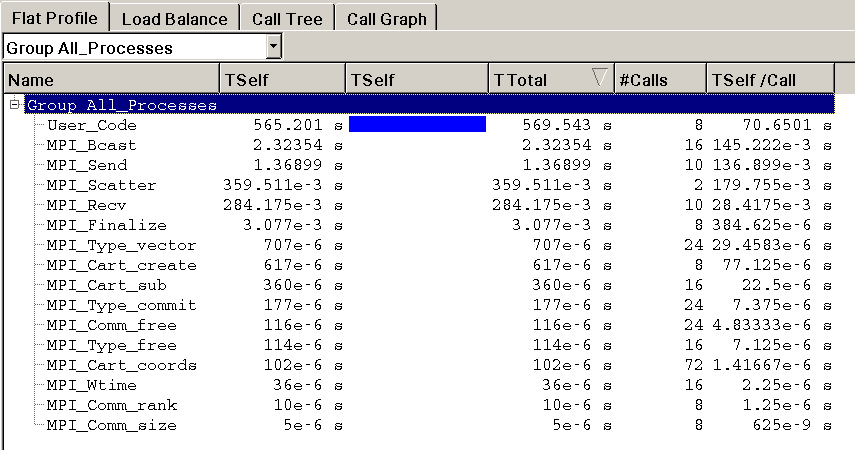


Рис. 1. Общее время работы всех функций в порядке убывания по времени

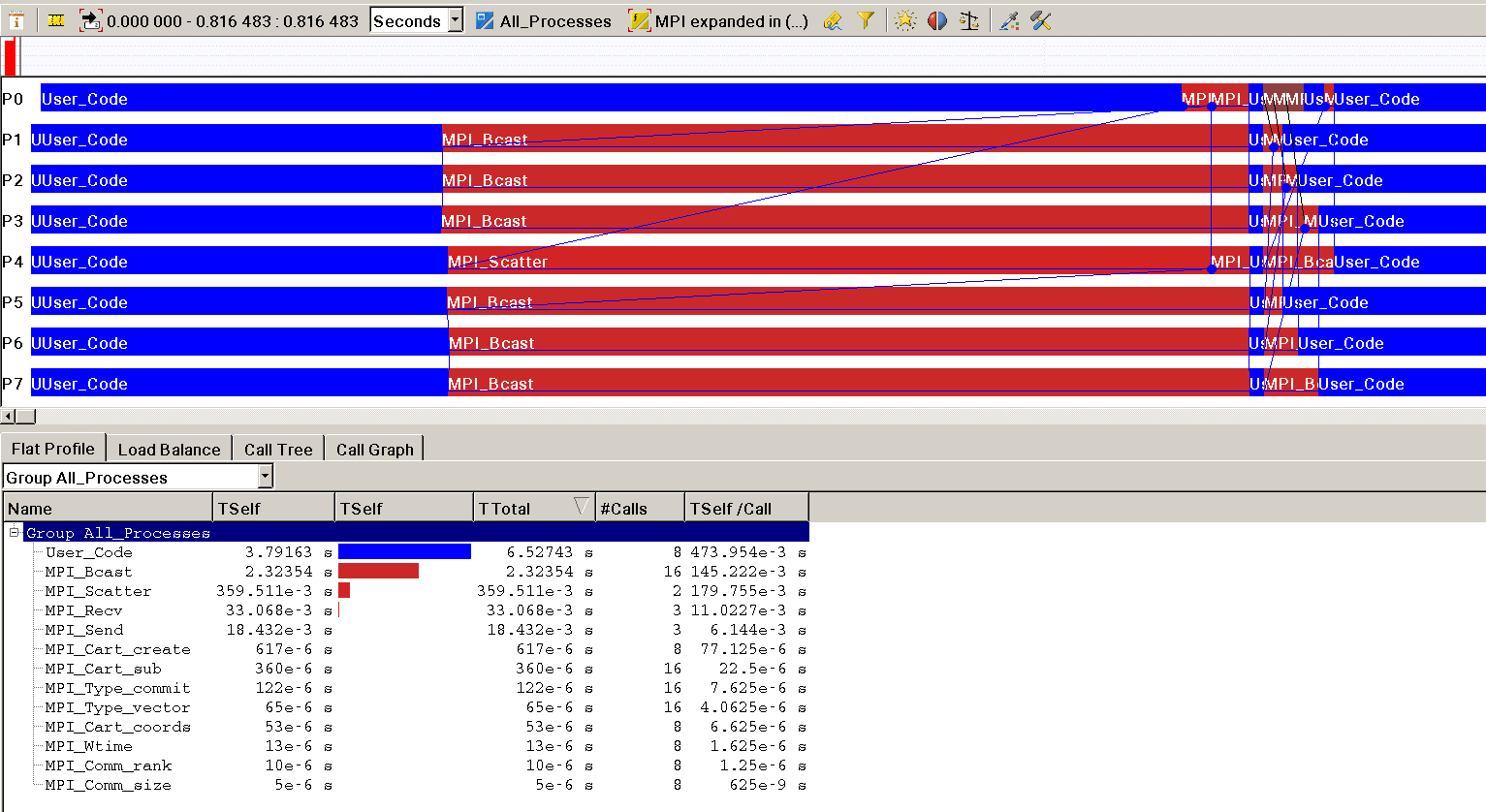


Рис. 2. Начало работы программы

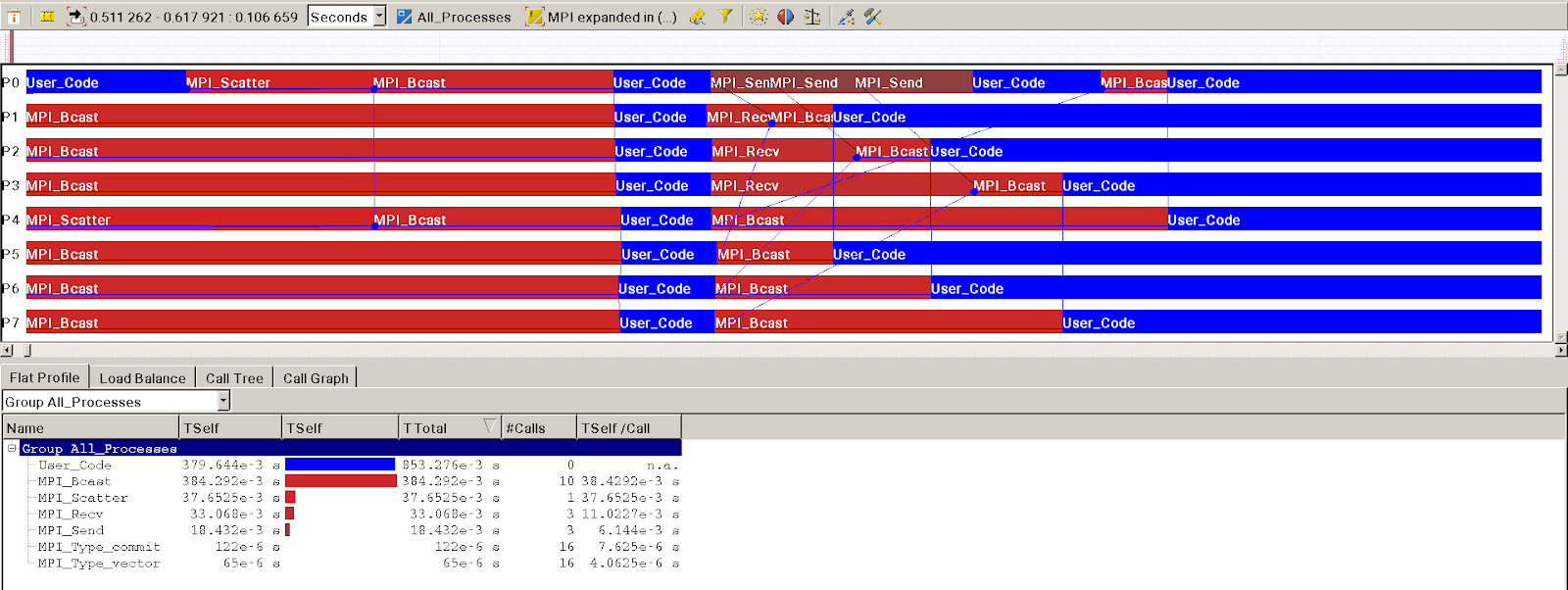


Рис. 3. Момент раздачи матрицы A с помощью Scatter и передача матрицы B

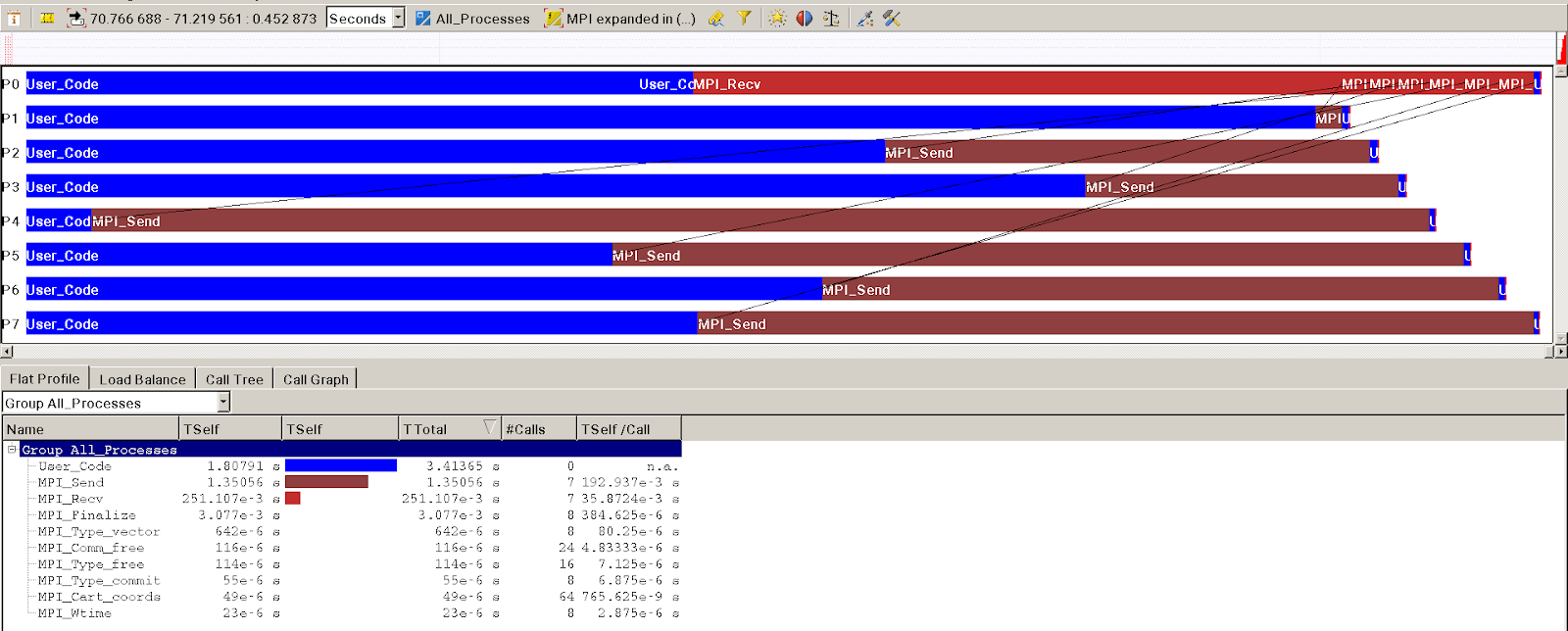


Рис. 4. Окончание работы программы, сбор всех данных в нулевом процессе

# Приложение 9. Скрины из traceanalyzer решетки размером 4x2

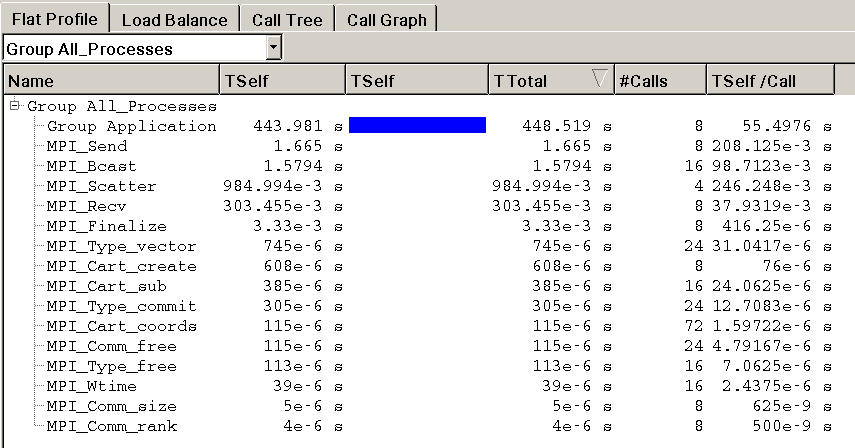


Рис. 1. Общее время работы всех функций в порядке убывания по времени

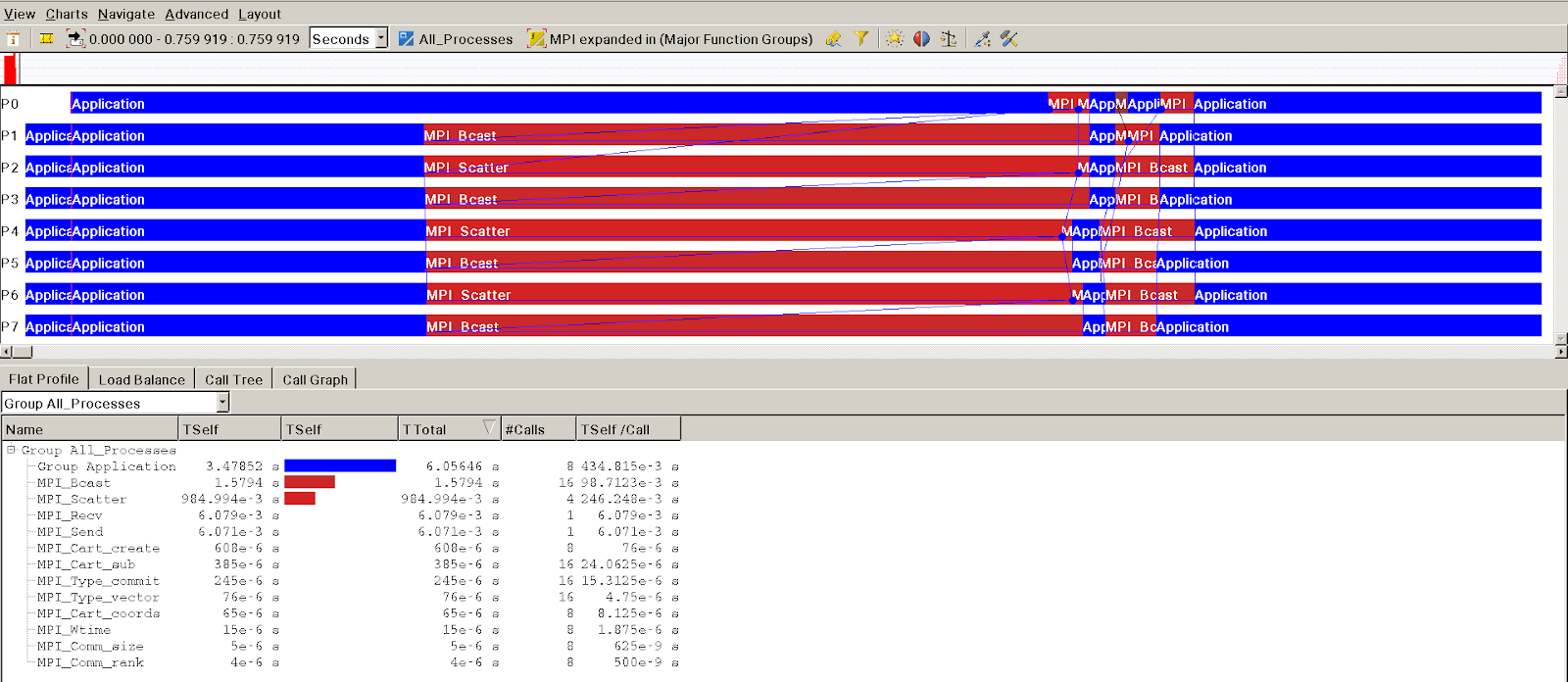


Рис. 2. Начало работы программы

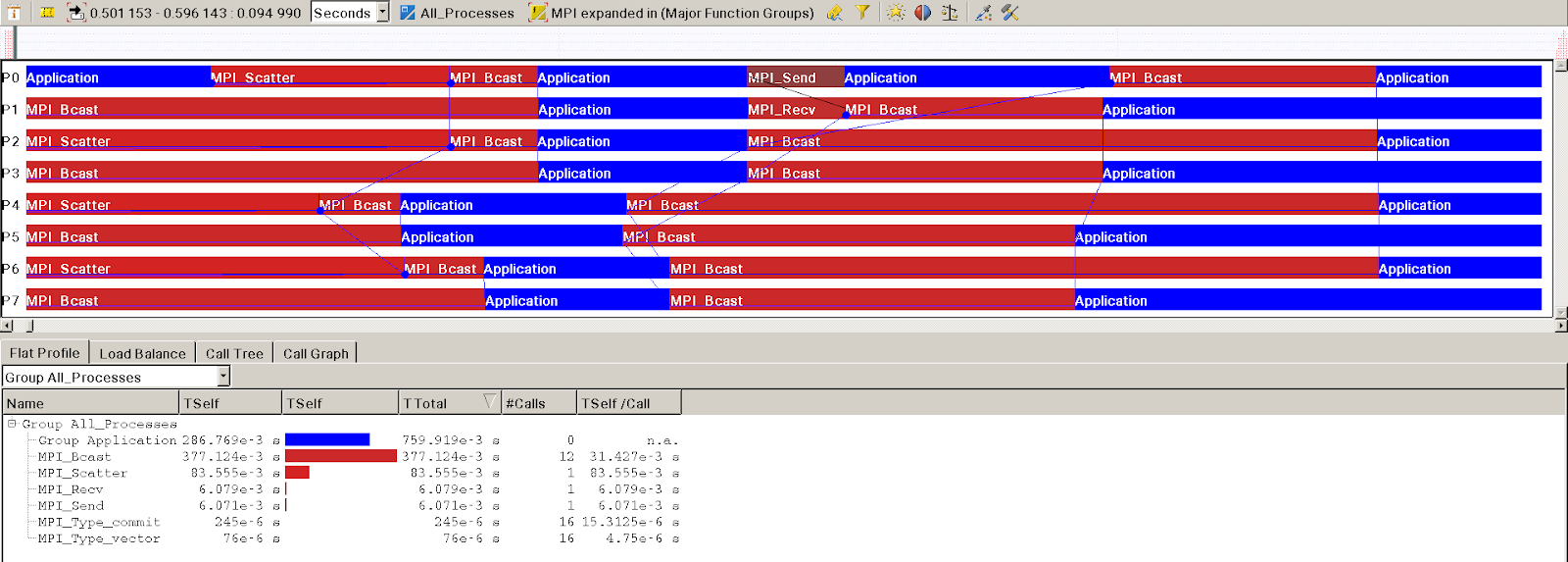


Рис. 3. Момент раздачи матрицы A с помощью Scatter и передача матрицы B

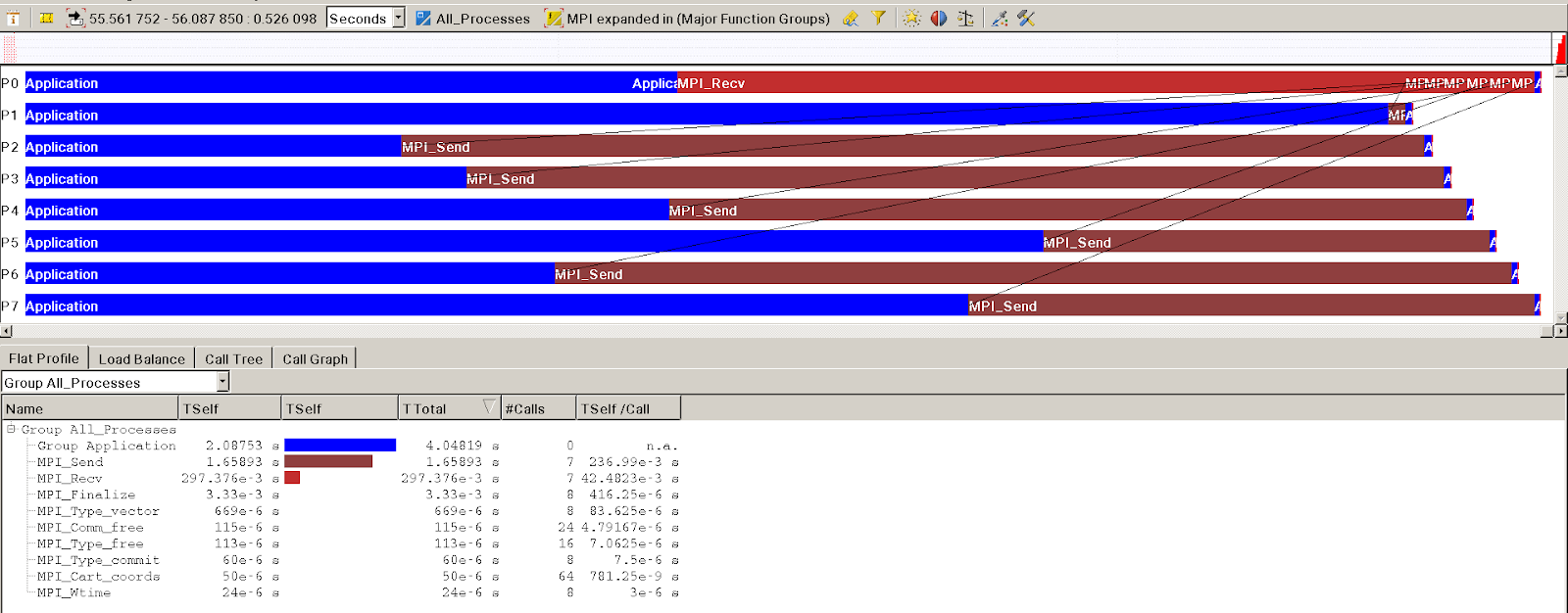


Рис. 4. Окончание работы программы, сбор всех данных в нулевом процессе