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

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.

2 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.

3 Выполнить профилирование программы с помощью jumpshot или ITAC (Intel Trace Analyzer and Collecter) при использовании 8-и ядер с решетками 2x4 и 4x2.

4 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 с 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.

5 Перейдём к работе на кластере. Количество процессов во всех измерениях равно 24.

Размер матриц, при котором время программы составляет не менее 30сек:

A = [6168 x 3000], B = [3000 x 5184].

6 Скрипт (run_parallel.sh) для запуска программы parallel.cpp смотреть в Приложении 4.

7 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	61.7622
3 x 8	60.2443
4 x 6	62.471
6 x 4	64.5881
8 x 3	64.2037
12 x 2	66.3319

8 График зависимости времени от размера решетки смотреть в Приложении 6.

9 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.

10 Сравним результаты профилирования решеток размером 2x4 и 4x2:

<table>
<tr><td></td><td>Решетка 2x4</td><td>Решетка 4x2</td></tr>
<tr><td>Коллективная операция Scatter</td><td>Вызывается в 0 и 4 процессах, т. к. матрица A «разрезается» по строкам, а потом передается на столбцы решетки (их 2).</td><td>Вызывается в 0, 2, 4, 6 процессах, т. к. матрица A «разрезается» по строкам, а потом передается на столбцы решетки (их 4).</td></tr>
<tr><td>Операция типа точка-точка MPI_Send()</td><td colspan="2">Нулевой процесс отправляет данные всем остальным процессам (кроме самого себя).</td></tr>
<tr><td rowspan="2">Операция типа точка-точка MPI_Recv()</td><td colspan="2">Для матрицы B и C:</td></tr>
<tr><td>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.</td><td>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.</td></tr>
<tr><td>Коллективная операция MPI_Bcast</td><td colspan="2">Нулевой процесс отправляет данные всем остальным процессам.</td></tr>
</table>

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>

/* Axises are
|-----Y
|
|
X
*/

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 multiplyMtrices(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); // reorder = 0
```

```
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

// number of blocks, numbers of elems in each block, number of
// elems between the start of each block, old type, new type
MPI_Type_vector(dim2, partB.column, dim3, MPI_DOUBLE, &bSendType);
MPI_Type_commit(&bSendType);

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, partB.row * partB.column, MPI_DOUBLE, 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]);
multiplyMtrices(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);

  // define the location of sqaure (partC)
  // relatively from the start of full matrix C
  offset[procRank] =
      coordsOfCurrProc[X_AXIS] * partC.row * C.column +
      coordsOfCurrProc[Y_AXIS] * partC.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

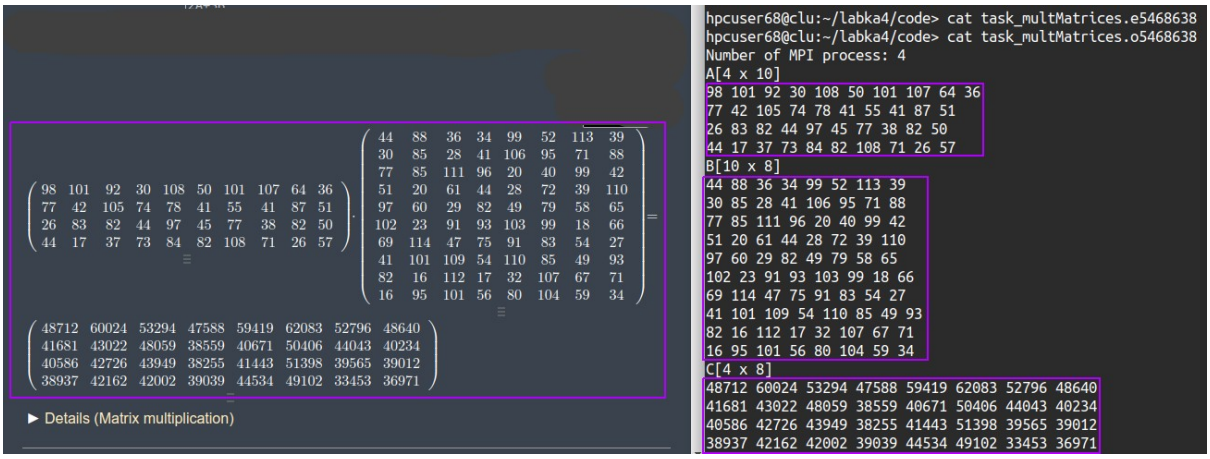


Рис. 1. Умножение матриц размером 4x10, 10x8

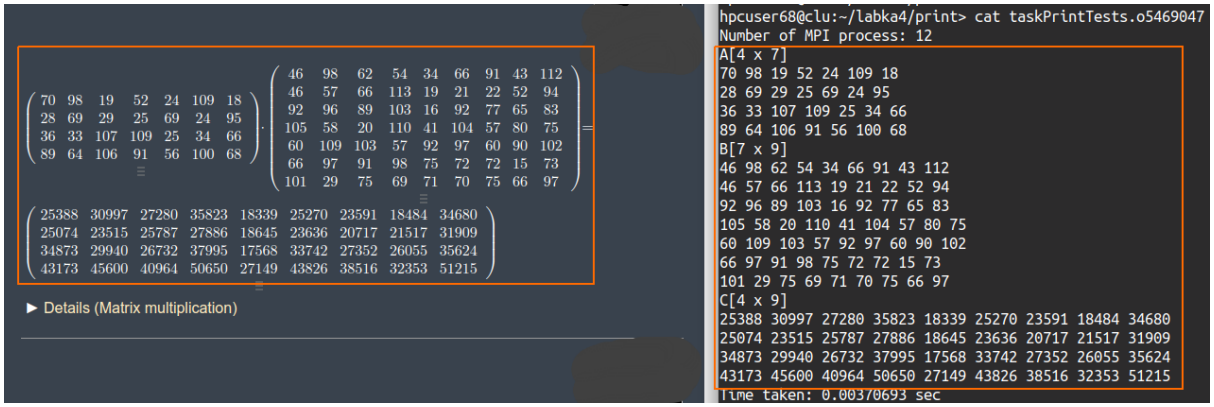


Рис. 2. Умножение матриц размером 4x7, 7x9

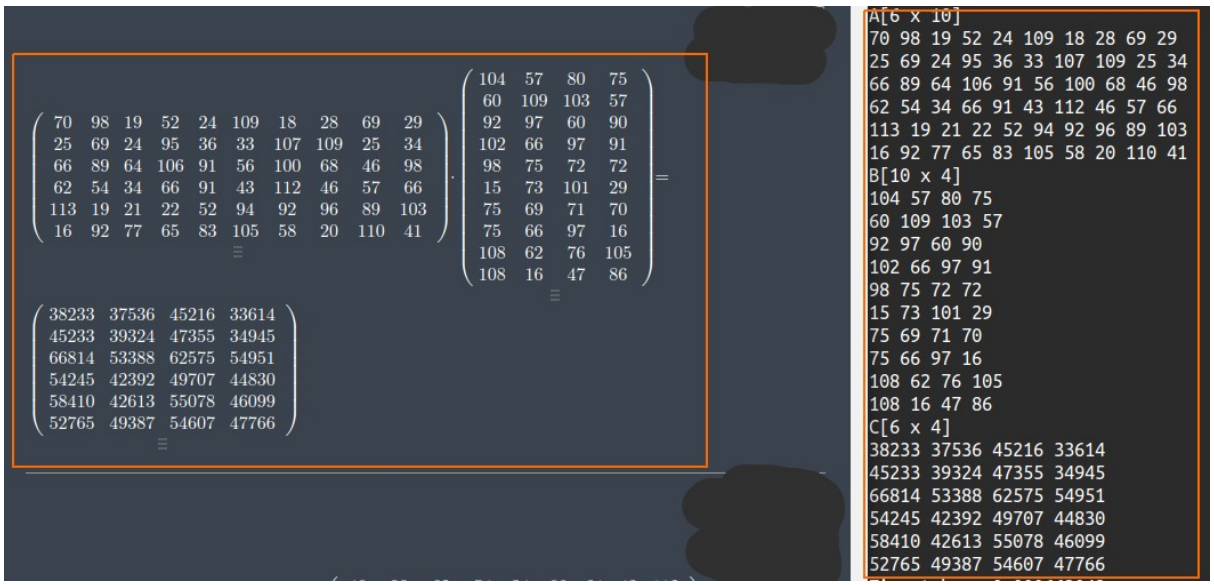


Рис. 3. Умножение матриц размером 6x10, 10x4

Приложение 4. Скрипт для запуска параллельной программы

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

```
hpcuser68@clu:~/labka4/remade> cat task_multMatrices.e5487311
hpcuser68@clu:~/labka4/remade> cat task_multMatrices.o5487311
Number of MPI process: 24
2 x 12 :
Time taken: 61.7622 sec
3 x 8 :
Time taken: 60.2443 sec
4 x 6 :
Time taken: 62.471 sec
6 x 4 :
Time taken: 64.5881 sec
8 x 3 :
Time taken: 64.2037 sec
12 x 2 :
Time taken: 66.3319 sec
```

Рис.1. Результаты замеров выполнения работы на кластере

Приложение 6. График зависимости времени от размера решетки

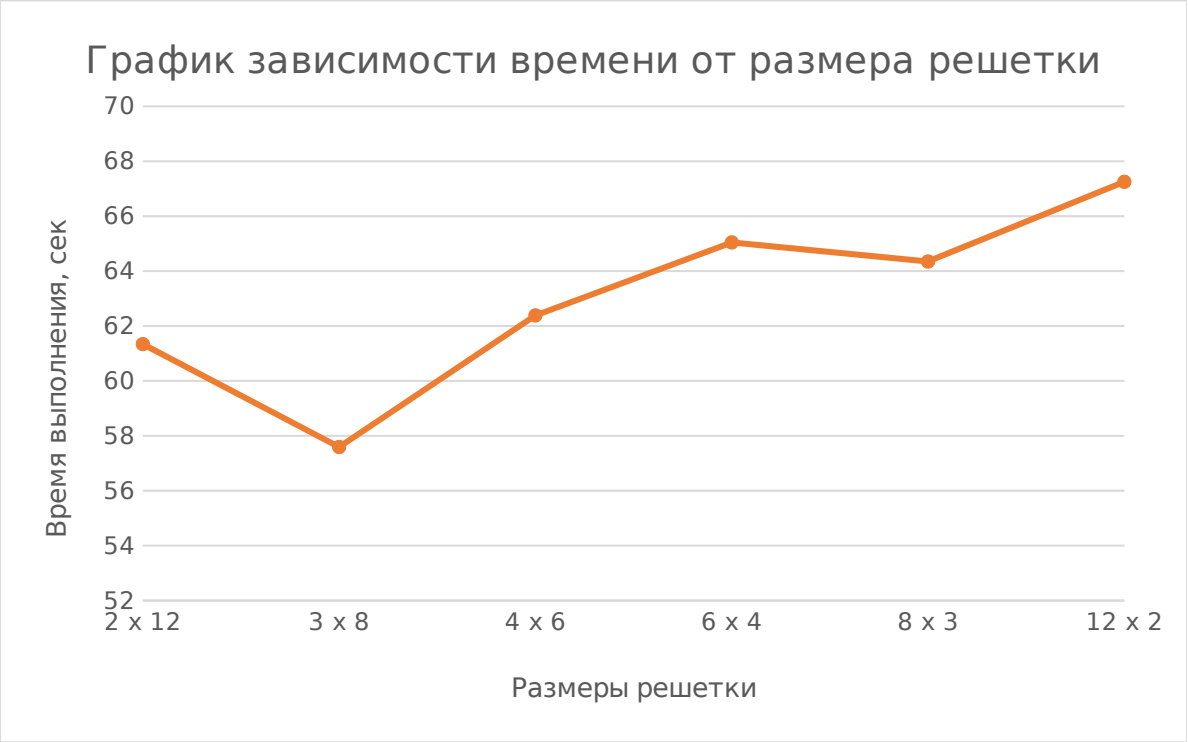


Рис. 1. График зависимости времени от размера решетки

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

##Скрипт 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

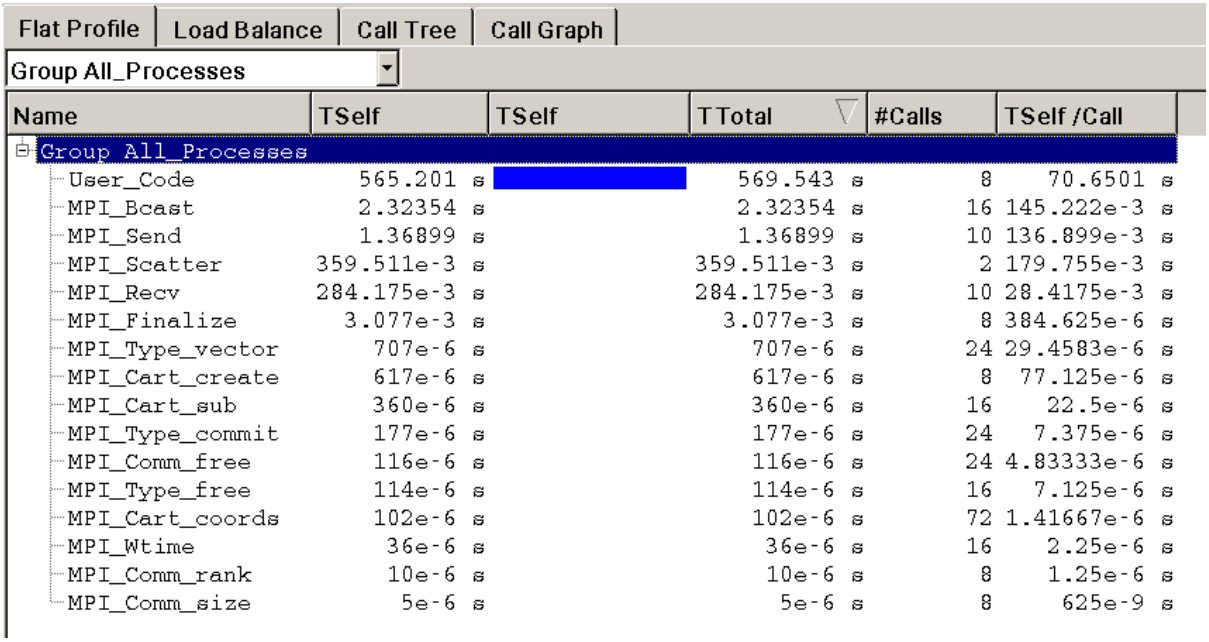
```
#!/bin/bash
#PBS -l walltime=00:05:00
#PBS -l select=1:ncpus=8:mpiprocs=8:mem=10000m,place=scatter:exclhost
#PBS -m n
#PBS -N trace_4x2_analyzeMatrix

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



Flat Profile | Load Balance | Call Tree | Call Graph

Group All_Processes

Name	TSelf	TSelf	TTotal	#Calls	TSelf /Call
Group All_Processes					
User_Code	565.201 s		569.543 s	8	70.6501 s
MPI_Bcast	2.32354 s		2.32354 s	16	145.222e-3 s
MPI_Send	1.36899 s		1.36899 s	10	136.899e-3 s
MPI_Scatter	359.511e-3 s		359.511e-3 s	2	179.755e-3 s
MPI_Recv	284.175e-3 s		284.175e-3 s	10	28.4175e-3 s
MPI_Finalize	3.077e-3 s		3.077e-3 s	8	384.625e-6 s
MPI_Type_vector	707e-6 s		707e-6 s	24	29.4583e-6 s
MPI_Cart_create	617e-6 s		617e-6 s	8	77.125e-6 s
MPI_Cart_sub	360e-6 s		360e-6 s	16	22.5e-6 s
MPI_Type_commit	177e-6 s		177e-6 s	24	7.375e-6 s
MPI_Comm_free	116e-6 s		116e-6 s	24	4.83333e-6 s
MPI_Type_free	114e-6 s		114e-6 s	16	7.125e-6 s
MPI_Cart_coords	102e-6 s		102e-6 s	72	1.41667e-6 s
MPI_Wtime	36e-6 s		36e-6 s	16	2.25e-6 s
MPI_Comm_rank	10e-6 s		10e-6 s	8	1.25e-6 s
MPI_Comm_size	5e-6 s		5e-6 s	8	625e-9 s

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

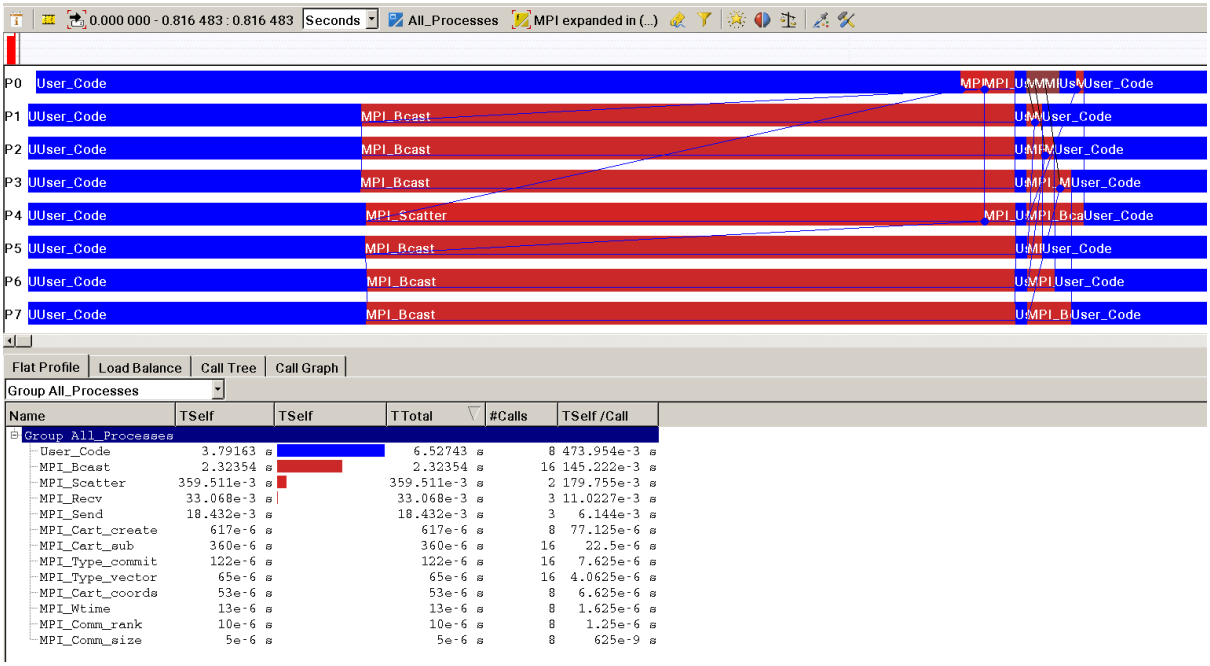


Рис. 2. Начало работы программы

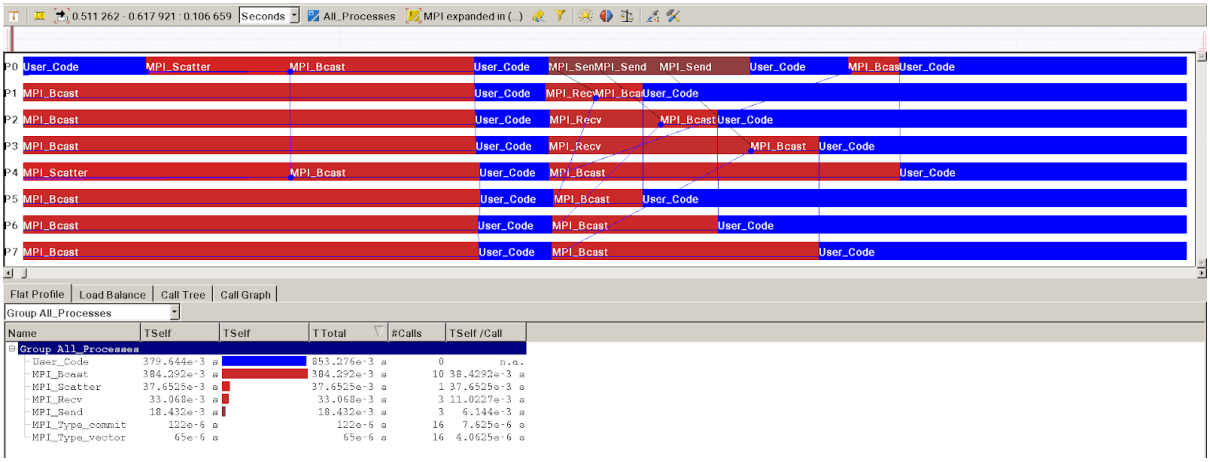


Рис. 3. Момент раздачи матрицы A с помощью Scatter и передача матрицы B

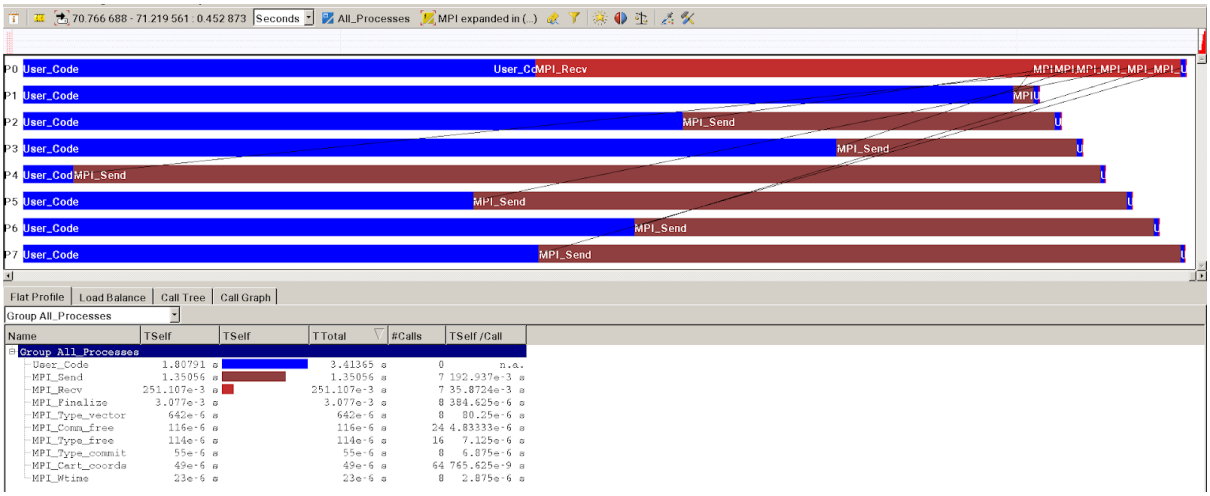


Рис. 4. Окончание работы программы, сбор всех данных в нулевом процессе

Приложение 9. Скрины из traceanalyzer решетки размером 4x2

Flat Profile | Load Balance | Call Tree | Call Graph

Group All_Processes

Name	TSelf	TSelf	T Total	#Calls	TSelf /Call
Group All_Processes					
Group Application	443.981 s		448.519 s	8	55.4976 s
MPI_Send	1.665 s		1.665 s	8	208.125e-3 s
MPI_Bcast	1.5794 s		1.5794 s	16	98.7123e-3 s
MPI_Scatter	984.994e-3 s		984.994e-3 s	4	246.248e-3 s
MPI_Recv	303.455e-3 s		303.455e-3 s	8	37.9319e-3 s
MPI_Finalize	3.33e-3 s		3.33e-3 s	8	416.25e-6 s
MPI_Type_vector	745e-6 s		745e-6 s	24	31.0417e-6 s
MPI_Cart_create	608e-6 s		608e-6 s	8	76e-6 s
MPI_Cart_sub	385e-6 s		385e-6 s	16	24.0625e-6 s
MPI_Type_commit	305e-6 s		305e-6 s	24	12.7083e-6 s
MPI_Cart_coords	115e-6 s		115e-6 s	72	1.59722e-6 s
MPI_Comm_free	115e-6 s		115e-6 s	24	4.79167e-6 s
MPI_Type_free	113e-6 s		113e-6 s	16	7.0625e-6 s
MPI_Wtime	39e-6 s		39e-6 s	16	2.4375e-6 s
MPI_Comm_size	5e-6 s		5e-6 s	8	625e-9 s
MPI_Comm_rank	4e-6 s		4e-6 s	8	500e-9 s

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

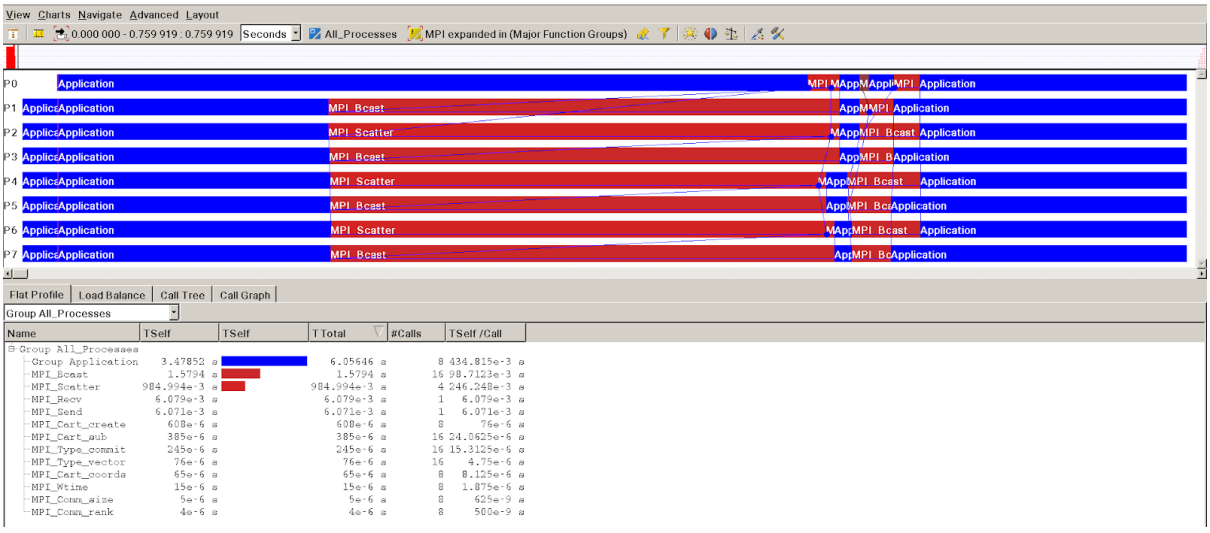


Рис. 2. Начало работы программы

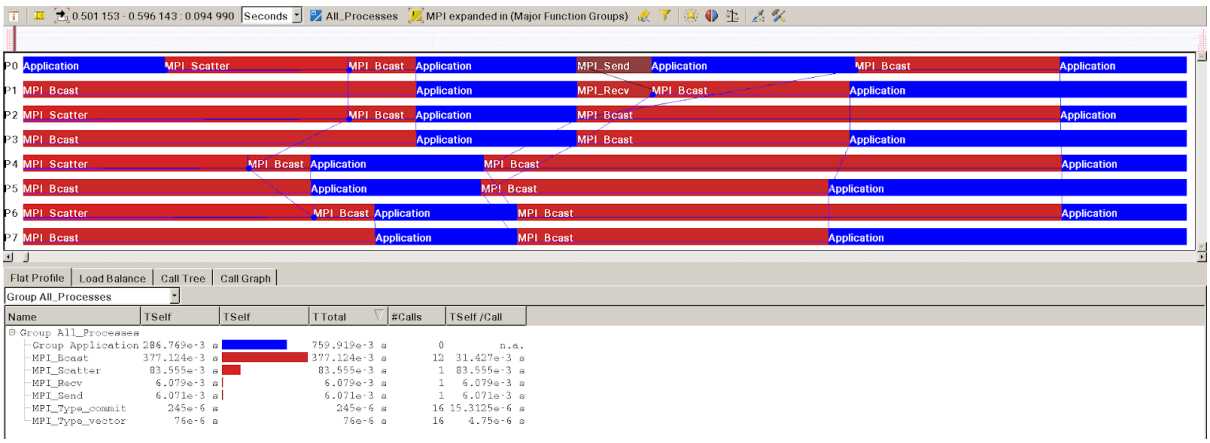


Рис. 3. Момент раздачи матрицы A с помощью Scatter и передача матрицы B

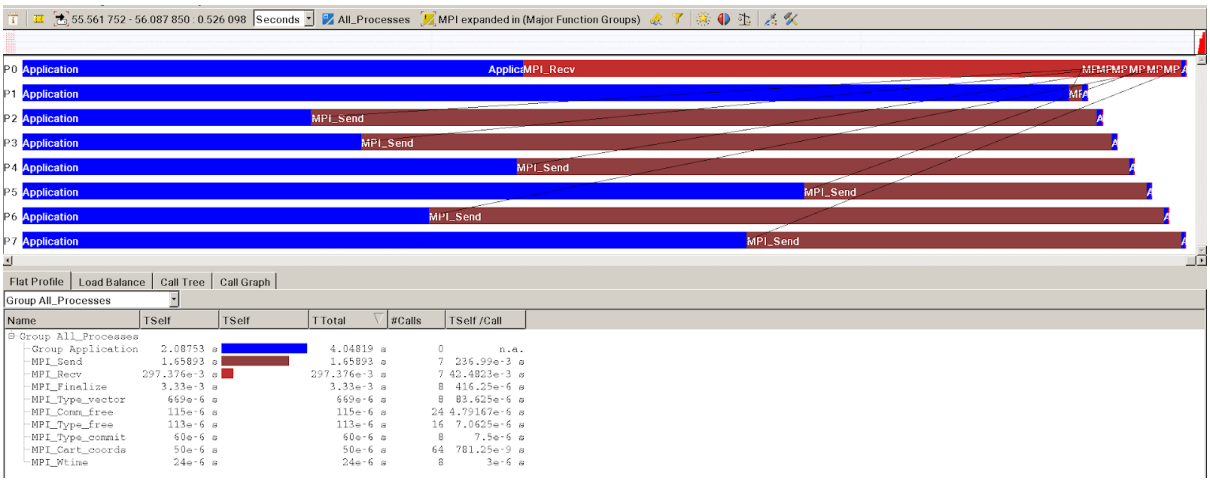


Рис. 4. Окончание работы программы, сбор всех данных в нулевом процессе