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 решетка»

студента Бородина Артёма Максимовича 2 курса, 19205 группы

Направление 09.03.01 – «Информатика и вычислительная техника»

Преподаватель:
к.т.н, доцент
А.Ю. Власенко

Новосибирск 2021

СОДЕРЖАНИЕ

ЦЕЛЬ

1. Умножение матриц, используя MPI.

ЗАДАНИЕ

1. Реализовать параллельный алгоритм умножения матрицы на матрицу при 2D решетке

2. Исследовать производительность параллельной программы в зависимости от размера матрицы и размера решетки.

3. Выполнить профилирование программы с помощью MPE при использовании 16-и ядер.

ОПИСАНИЕ РАБОТЫ

1. Были написаны два варианта программы (Приложение 1, Приложение 2), с использованием векторных расширений и без с MPI, реализующие алгоритм умножения матриц.

2. Выбраны размеры матриц, при которых время работы программы без векторизации и MPI занимало примерно 30 секунд.

3. Исследовано время работы программы при разных размерах решетки на 16-ти ядрах (Приложение 3). По полученным данным можно сделать вывод что программа работает наилучшим образом при размере решетки близкой к квадрату.

4. Сделано профилирование программы на 16-ти ядрах (Приложение 4). На скриншотах можно увидеть, как работает коммуникация между процессами (при решетке 4x4): 0-3 разделяют между собой матрицу A, а затем распространяют свои части другим процессам, процессы 0,4,8,12 делают тоже самое только с матрицей B. В конце можно увидеть, как собирается матрица C в 0-ом процессе.

5. Были проведены замеры времени работы вариантов на разном количестве ядрах (Приложение 5). Можно заметить заметное ускорение по сравнение с программой без векторизации и MPI. По графикам можно увидеть, что вариант с векторизацией работает примерно на 70% быстрее чем без.

ЗАКЛЮЧЕНИЕ

В ходе лабораторной работы были получены знания о разделения нагрузки на множество процессов для продуктивного умножения матриц. Так же в очередной раз доказали свою эффективность векторные расширения, позволяющие без особых усилий получить значительный прирост в производительности.

Приложение 1. *Код программы с векторизацией*

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

void printMat(double *mat, int rows, int columns, std::ofstream &stream) {
  for (int i = 0; i < rows; ++i) {
    for (int j = 0; j < columns; ++j) {
      stream << mat[i * columns + j] << " ";
    }
    stream << std::endl;
  }
}

int main(int argc, char *argv[]) {
  MPI_Init(&argc, &argv);
  const int ndims = 2;
  const int X = 0;
  const int Y = 1;

  int dims[ndims] = {0};
  int periods[ndims] = {0};
  int coords[ndims];
  int procsCount;
  int rank;
  MPI_Comm gridComm;
  MPI_Comm rowComm;
  MPI_Comm colComm;

  if (argc == 7) {
    dims[X] = atoi(argv[5]);
    dims[Y] = atoi(argv[6]);
  }

  MPI_Comm_size(MPI_COMM_WORLD, &procsCount);
  MPI_Comm_rank(MPI_COMM_WORLD, &rank);
  MPI_Dims_create(procsCount, ndims, dims);

  MPI_Cart_create(MPI_COMM_WORLD, ndims, dims, periods, 0, &gridComm);
  MPI_Cart_coords(gridComm, rank, ndims, coords);
  MPI_Comm_split(gridComm, coords[Y], coords[X], &rowComm);
  MPI_Comm_split(gridComm, coords[X], coords[Y], &colComm);

  if (argc < 5 && rank == 0) {
    std::cout << "Program needs 4 arguments: N1, N2, N3, fileName" << std::endl;
    MPI_Finalize();
    return 0;
  }
  int N1 = atoi(argv[1]);
  int N2 = atoi(argv[2]);
  int N3 = atoi(argv[3]);
  std::ofstream fileStream(std::to_string(rank) + argv[4]);
  if (!fileStream && rank == 0) {
    std::cout << "error with output file" << std::endl;
    MPI_Finalize();
    return 0;
  }
  fileStream << "N1: " << N1 << " N2: " << N2 << " N3: " << N3 << std::endl;
  fileStream << "procCount: " << procsCount << std::endl;
```

```
double *A;
double *B;
double *C;
if (rank == 0) {
    std::cout << "N1: " << N1 << " N2: " << N2 << " N3: " << N3 << std::endl;
    A = new double[N1 * N2];
    B = new double[N2 * N3];
    C = new double[N1 * N3];
    std::fill(A, A + N1 * N2, 1);
    std::fill(B, B + N2 * N3, 2);

    for (int i = 0; i < N1; ++i) {
        A[i * N2 + i] = 10;
    }
    for (int i = 0; i < N2; ++i) {
        B[i * N3 + i] = 20;
    }
}
fileStream << "dims: (" + std::to_string(dims[X]) + ", " + std::to_string(dims[Y]) + ")" << std::endl
           << "cords: (" + std::to_string(coords[X]) + ", " + std::to_string(coords[Y]) + ")" << std::endl;

int segmentRows = N1 / dims[Y];
int segmentCols = N3 / dims[X];
auto *segmentA = new double[segmentRows * N2];
auto *segmentB = new double[N2 * segmentCols];
auto *segmentC = new double[segmentRows * segmentCols];
std::fill(segmentC, segmentC + segmentRows * segmentCols, 0);

double matMulTime = -MPI_Wtime();

/// Distribute matrices
if (coords[X] == 0) {
    MPI_Scatter(A, segmentRows * N2, MPI_DOUBLE, segmentA, segmentRows * N2, MPI_DOUBLE, 0, colComm);
}
if (coords[Y] == 0) {
    MPI_Datatype sendSegment;
    MPI_Datatype sendSegmentDouble;

    MPI_Type_vector(N2, segmentCols, N3, MPI_DOUBLE, &sendSegment);
    MPI_Type_commit(&sendSegment);

    MPI_Type_create_resized(sendSegment, 0, segmentCols * sizeof(double), &sendSegmentDouble);
    MPI_Type_commit(&sendSegmentDouble);

    MPI_Scatter(B, 1, sendSegmentDouble, segmentB, N2 * segmentCols, MPI_DOUBLE, 0, rowComm);

    MPI_Type_free(&sendSegment);
    MPI_Type_free(&sendSegmentDouble);
}
MPI_Bcast(segmentA, segmentRows * N2, MPI_DOUBLE, 0, rowComm);
MPI_Bcast(segmentB, N2 * segmentCols, MPI_DOUBLE, 0, colComm);

double segmentMulTime = -MPI_Wtime();
/// Matrix multiplication
for (int i = 0; i < segmentRows; ++i) {
    for (int j = 0; j < segmentCols; j += 2) {
        __m128d vR = _mm_setzero_pd();
        for (int k = 0; k < N2; k++) {
            __m128d vA = _mm_set1_pd(segmentA[i * N2 + k]);
            __m128d vB = _mm_loadu_pd(&segmentB[k * segmentCols + j]);
            vR += vA * vB;
        }
        _mm_storeu_pd(&segmentC[i * segmentCols + j], vR);
```

```
    }
  }
  segmentMulTime += MPI_Wtime();

  /// collect C
  MPI_Datatype recvSegment;
  MPI_Datatype recvSegmentDouble;

  MPI_Type_vector(segmentRows, segmentCols, N3, MPI_DOUBLE, &recvSegment);
  MPI_Type_commit(&recvSegment);

  MPI_Type_create_resized(recvSegment, 0, segmentCols * sizeof(double), &recvSegmentDouble);
  MPI_Type_commit(&recvSegmentDouble);

  int recvCounts[procsCount];
  std::fill(recvCounts, recvCounts + procsCount, 1);
  int displs[procsCount];
  for (int procRank = 0; procRank < procsCount; ++procRank) {
    MPI_Cart_coords(gridComm, procRank, ndims, coords);
    displs[procRank] = dims[X] * segmentRows * coords[Y] + coords[X];
  }

  MPI_Gatherv(segmentC, segmentRows * segmentCols, MPI_DOUBLE, C, recvCounts, displs, recvSegmentDouble,
          0, gridComm);

  MPI_Type_free(&recvSegment);
  MPI_Type_free(&recvSegmentDouble);

  MPI_Comm_free(&gridComm);
  MPI_Comm_free(&colComm);
  MPI_Comm_free(&rowComm);

  matMulTime += MPI_Wtime();

  /// Print info
  fileStream << std::endl << "segmentMulTime: " << segmentMulTime << "sec" << std::endl;

  /// Print matrices
  if (rank == 0) {
    std::cout << "segmentMulTime(0): " << segmentMulTime << "sec" << std::endl;
    std::cout << "matMulTime: " << matMulTime << "sec" << std::endl;
    fileStream << "matMulTime: " << matMulTime << "sec" << std::endl;
    fileStream << std::endl << "A: " << std::endl << "rows " << N1 << " cols: " << N2 << std::endl;
    printMat(A, N1, N2, fileStream);
    fileStream << "B: " << std::endl << "rows: " << N2 << " cols: " << N3 << std::endl;
    printMat(B, N2, N3, fileStream);
    fileStream << "C: " << std::endl << "rows: " << N1 << " cols: " << N3 << std::endl;
    printMat(C, N1, N3, fileStream);
  }
  fileStream.close();

  if (rank == 0) {
    free(A);
    free(B);
    free(C);
  }
  free(segmentA);
  free(segmentB);
  free(segmentC);

  MPI_Finalize();
  return 0;
}
```

Приложение 2. *Код программы без векторизации*

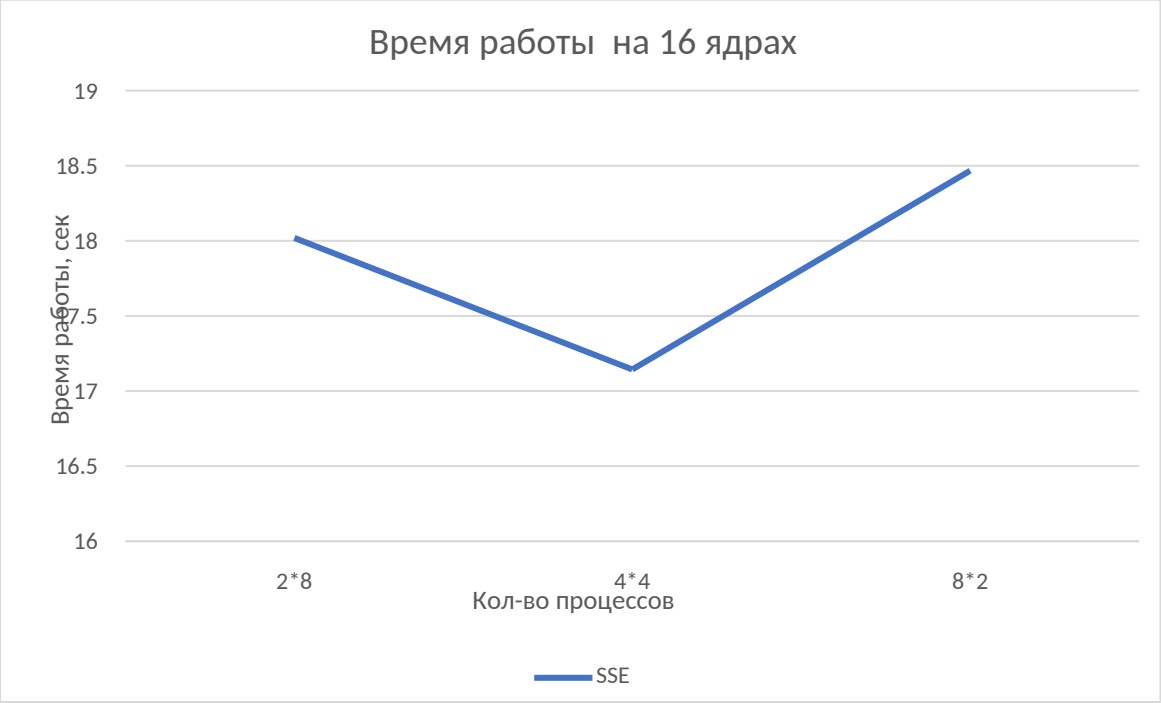
```
#include <iostream>
#include <fstream>
#include <xmmintrin.h>
#include "mpi.h"

void printMat(double *mat, int rows, int columns, std::ofstream &stream) {
  for (int i = 0; i < rows; ++i) {
    for (int j = 0; j < columns; ++j) {
      stream << mat[i * columns + j] << " ";
    }
    stream << std::endl;
  }
}

int main(int argc, char *argv[]) {
  MPI_Init(&argc, &argv);
  const int ndims = 2;
  const int X = 0;
  const int Y = 1;

  int dims[ndims] = {0};
  int periods[ndims] = {0};
  int coords[ndims];
  int procsCount;
  int rank;
  MPI_Comm gridComm;
  MPI_Comm rowComm;
  MPI_Comm colComm;

  if (argc == 7) {
    dims[X] = atoi(argv[5]);
    dims[Y] = atoi(argv[6]);
  }

  MPI_Comm_size(MPI_COMM_WORLD, &procsCount);
  MPI_Comm_rank(MPI_COMM_WORLD, &rank);
  MPI_Dims_create(procsCount, ndims, dims);

  MPI_Cart_create(MPI_COMM_WORLD, ndims, dims, periods, 0, &gridComm);
  MPI_Cart_coords(gridComm, rank, ndims, coords);
  MPI_Comm_split(gridComm, coords[Y], coords[X], &rowComm);
  MPI_Comm_split(gridComm, coords[X], coords[Y], &colComm);

  if (argc < 5 && rank == 0) {
    std::cout << "Program needs 4 arguments: N1, N2, N3, fileName" << std::endl;
    MPI_Finalize();
    return 0;
  }
  int N1 = atoi(argv[1]);
  int N2 = atoi(argv[2]);
  int N3 = atoi(argv[3]);
  std::ofstream fileStream(std::to_string(rank) + argv[4]);
  if (!fileStream && rank == 0) {
    std::cout << "error with output file" << std::endl;
    MPI_Finalize();
    return 0;
  }
  fileStream << "N1: " << N1 << " N2: " << N2 << " N3: " << N3 << std::endl;
  fileStream << "procCount: " << procsCount << std::endl;

  double *A;
  double *B;
```

```
double *C;
if (rank == 0) {
    std::cout << "N1: " << N1 << " N2: " << N2 << " N3: " << N3 << std::endl;
    A = new double[N1 * N2];
    B = new double[N2 * N3];
    C = new double[N1 * N3];
    std::fill(A, A + N1 * N2, 1);
    std::fill(B, B + N2 * N3, 2);

    for (int i = 0; i < N1; ++i) {
        A[i * N2 + i] = 10;
    }
    for (int i = 0; i < N2; ++i) {
        B[i * N3 + i] = 20;
    }
}
fileStream << "dims: (" + std::to_string(dims[X]) + ", " + std::to_string(dims[Y]) + ")" << std::endl
           << "cords: (" + std::to_string(coords[X]) + ", " + std::to_string(coords[Y]) + ")" << std::endl;

int segmentRows = N1 / dims[Y];
int segmentCols = N3 / dims[X];
auto *segmentA = new double[segmentRows * N2];
auto *segmentB = new double[N2 * segmentCols];
auto *segmentC = new double[segmentRows * segmentCols];
std::fill(segmentC, segmentC + segmentRows * segmentCols, 0);

double matMulTime = -MPI_Wtime();

/// Distribute matrices
if (coords[X] == 0) {
    MPI_Scatter(A, segmentRows * N2, MPI_DOUBLE, segmentA, segmentRows * N2, MPI_DOUBLE, 0, colComm);
}
if (coords[Y] == 0) {
    MPI_Datatype sendSegment;
    MPI_Datatype sendSegmentDouble;

    MPI_Type_vector(N2, segmentCols, N3, MPI_DOUBLE, &sendSegment);
    MPI_Type_commit(&sendSegment);

    MPI_Type_create_resized(sendSegment, 0, segmentCols * sizeof(double), &sendSegmentDouble);
    MPI_Type_commit(&sendSegmentDouble);

    MPI_Scatter(B, 1, sendSegmentDouble, segmentB, N2 * segmentCols, MPI_DOUBLE, 0, rowComm);

    MPI_Type_free(&sendSegment);
    MPI_Type_free(&sendSegmentDouble);
}
MPI_Bcast(segmentA, segmentRows * N2, MPI_DOUBLE, 0, rowComm);
MPI_Bcast(segmentB, N2 * segmentCols, MPI_DOUBLE, 0, colComm);

double segmentMulTime = -MPI_Wtime();
/// Matrix multiplication
for (int i = 0; i < segmentRows; ++i) {
for (int k = 0; k < N2; ++k) {
    for (int j = 0; j < segmentCols; ++j) {
        segmentC[i * segmentCols + j] += segmentA[i * N2 + k] * segmentB[k * segmentCols + j];
       }
     }
}
segmentMulTime += MPI_Wtime();

/// collect C
MPI_Datatype recvSegment;
```

```
  MPI_Datatype recvSegmentDouble;

  MPI_Type_vector(segmentRows, segmentCols, N3, MPI_DOUBLE, &recvSegment);
  MPI_Type_commit(&recvSegment);

  MPI_Type_create_resized(recvSegment, 0, segmentCols * sizeof(double), &recvSegmentDouble);
  MPI_Type_commit(&recvSegmentDouble);

  int recvCounts[procsCount];
  std::fill(recvCounts, recvCounts + procsCount, 1);
  int displs[procsCount];
  for (int procRank = 0; procRank < procsCount; ++procRank) {
    MPI_Cart_coords(gridComm, procRank, ndims, coords);
    displs[procRank] = dims[X] * segmentRows * coords[Y] + coords[X];
  }

  MPI_Gatherv(segmentC, segmentRows * segmentCols, MPI_DOUBLE, C, recvCounts, displs, recvSegmentDouble,
          0, gridComm);

  MPI_Type_free(&recvSegment);
  MPI_Type_free(&recvSegmentDouble);

  MPI_Comm_free(&gridComm);
  MPI_Comm_free(&colComm);
  MPI_Comm_free(&rowComm);

  matMulTime += MPI_Wtime();

  /// Print info
  fileStream << std::endl << "segmentMulTime: " << segmentMulTime << "sec" << std::endl;

  /// Print matrices
  if (rank == 0) {
    std::cout << "segmentMulTime(0): " << segmentMulTime << "sec" << std::endl;
    std::cout << "matMulTime: " << matMulTime << "sec" << std::endl;
    fileStream << "matMulTime: " << matMulTime << "sec" << std::endl;
    fileStream << std::endl << "A: " << std::endl << "rows " << N1 << " cols: " << N2 << std::endl;
    printMat(A, N1, N2, fileStream);
    fileStream << "B: " << std::endl << "rows: " << N2 << " cols: " << N3 << std::endl;
    printMat(B, N2, N3, fileStream);
    fileStream << "C: " << std::endl << "rows: " << N1 << " cols: " << N3 << std::endl;
    printMat(C, N1, N3, fileStream);
  }
  fileStream.close();

  if (rank == 0) {
    free(A);
    free(B);
    free(C);
  }
  free(segmentA);
  free(segmentB);
  free(segmentC);

  MPI_Finalize();
  return 0;
}
```

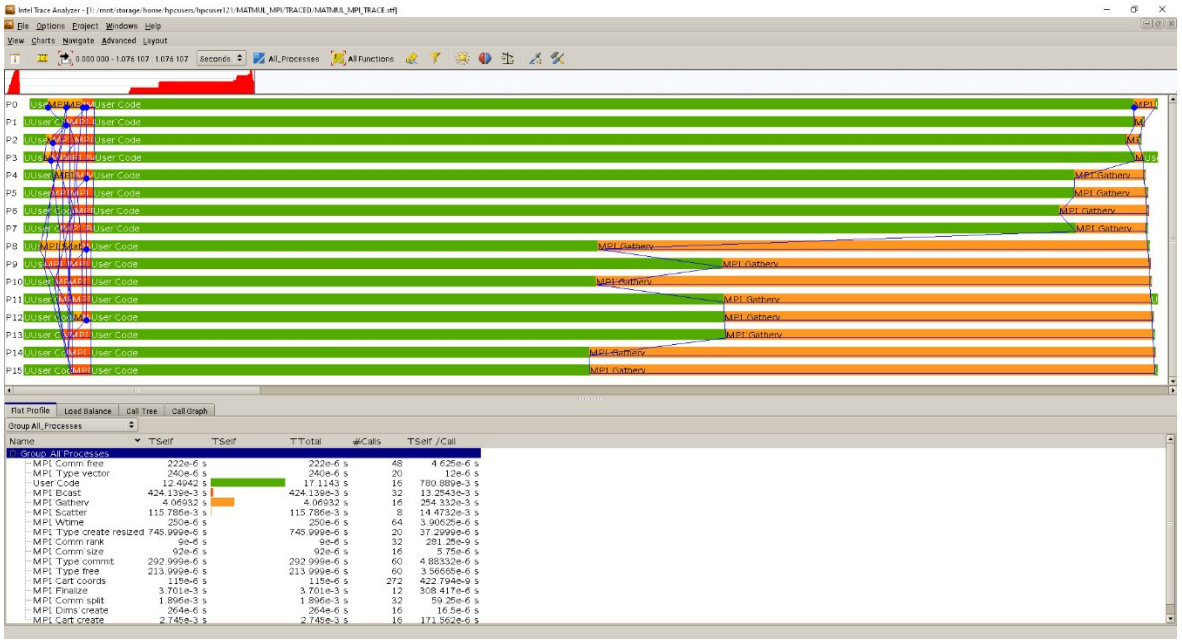
Приложение 3. *Исследование работы программы при разных размерах решетки*

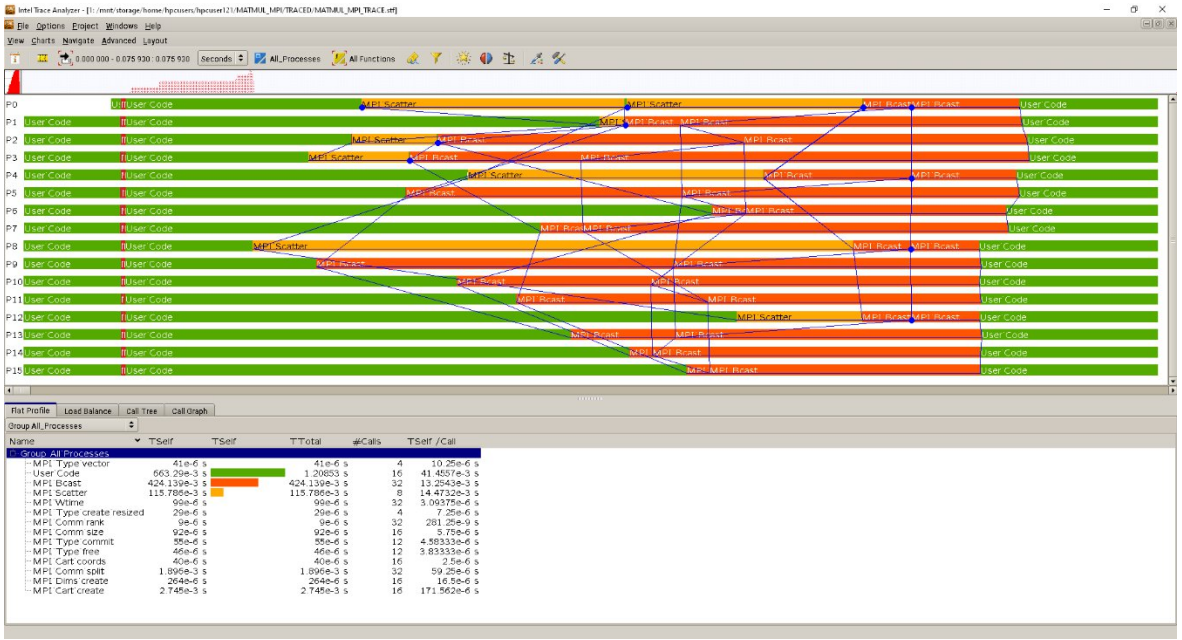
	Время
*2*8*	18,0195
*4*4*	17,1437
*8*2*	18,4672



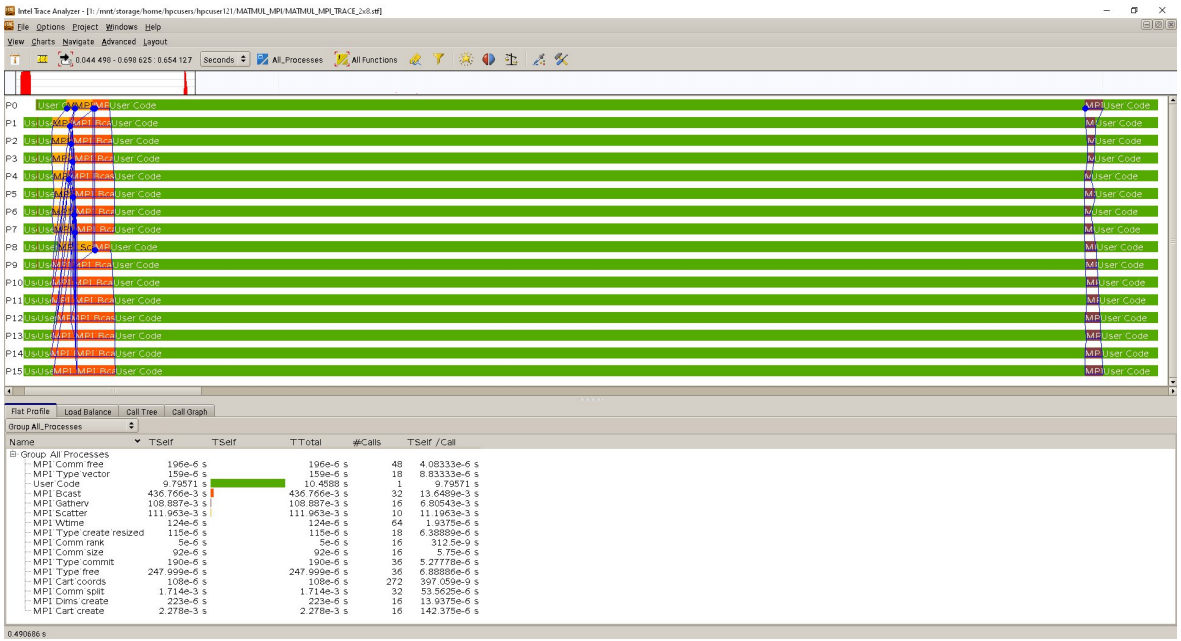
Приложение 4. *Профилирование на 16 ядрах*

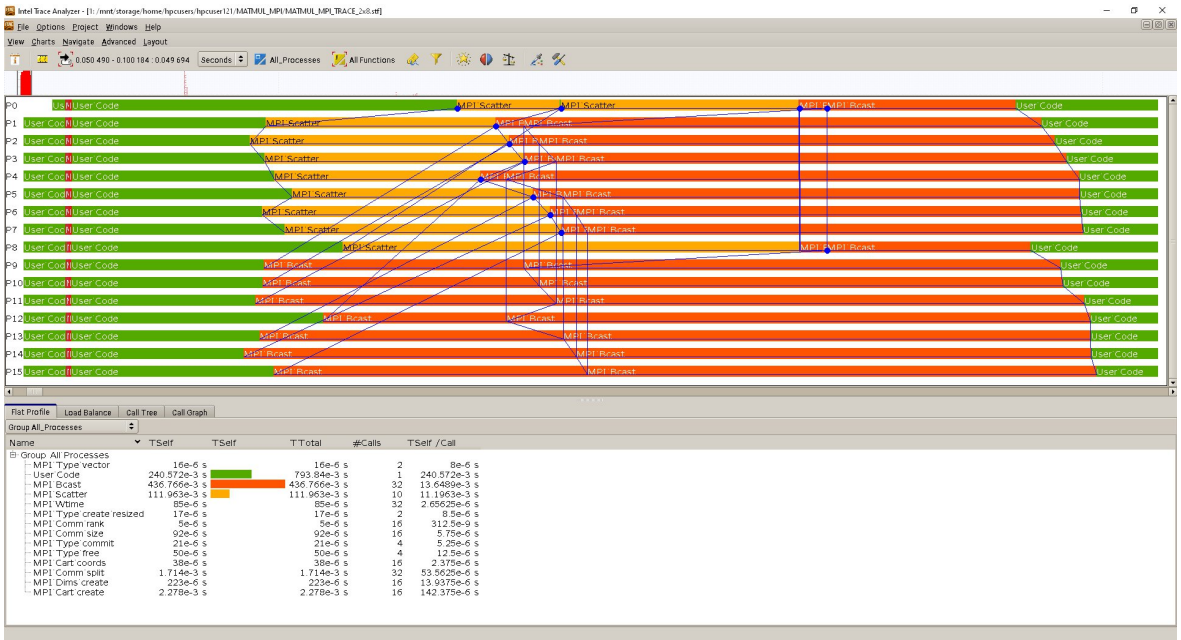
4x4:



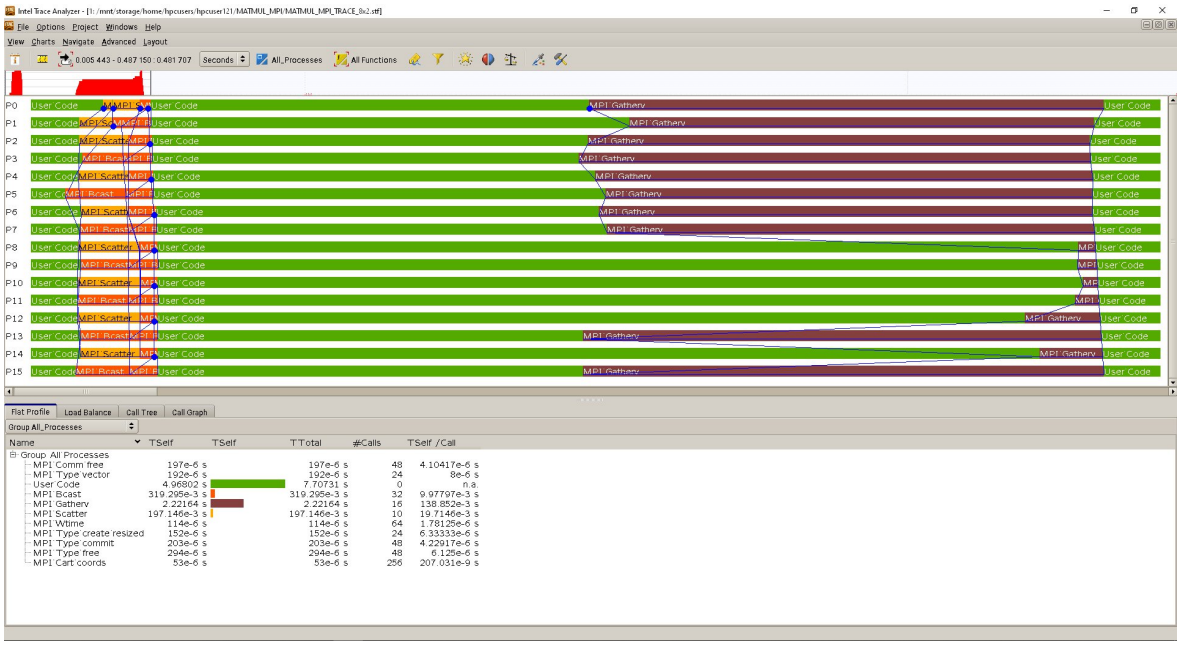


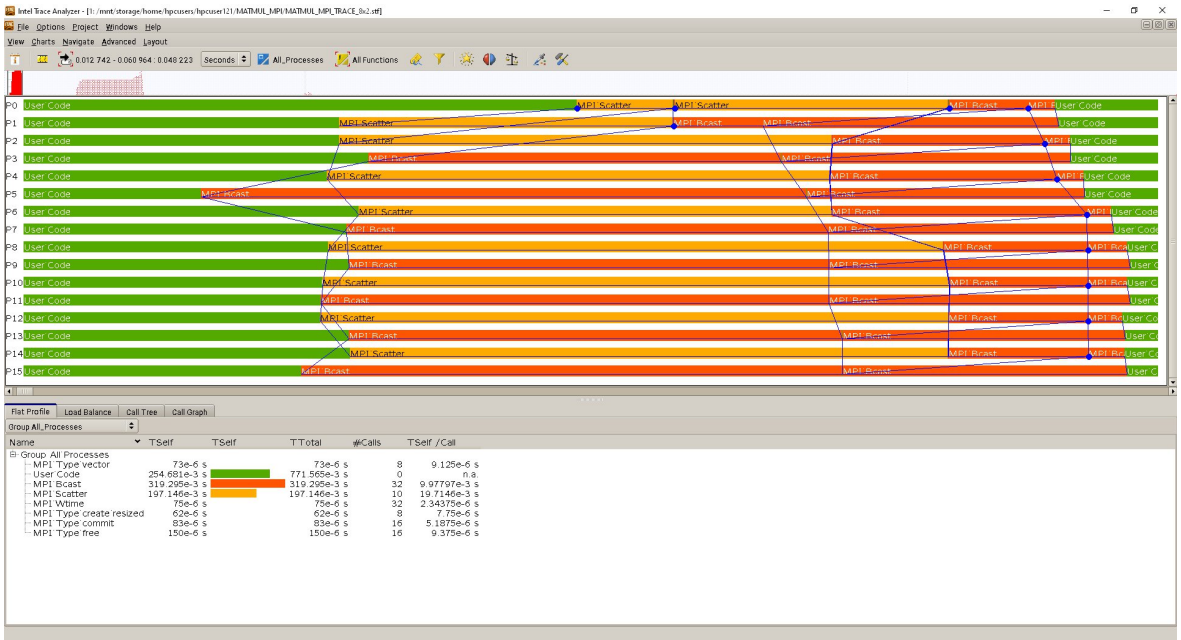
2x8:


Intel Trace Analyzer - [I:/mnt/storage/home/hpcusers/hpcuser121/MATMUL_MPI/MATMUL_MPI_TRACE_2x8.stf]
File Options Project Windows Help
View Charts Navigate Advanced Layout
0.044 498 - 0.698 625 : 0.654 127
Seconds
All_Processes
All Functions
P0
P1
P2
P3
P4
P5
P6
P7
P8
P9
P10
P11
P12
P13
P14
P15
User Code
MPI_Bcast
MPI_Scatter
Flat Profile
Load Balance
Call Tree
Call Graph
Group All_Processes
Name
TSelf
TSelf
TTotal
#Calls
TSelf /Call
Group All Processes
MPI Comm free 196e-6 s 196e-6 s 48 4.08333e-6 s
MPI Type vector 159e-6 s 159e-6 s 18 8.83333e-6 s
User Code 9.79571 s 10.4588 s 1 9.79571 s
MPI Bcast 436.766e-3 s 436.766e-3 s 32 13.6489e-3 s
MPI Gatherv 108.887e-3 s 108.887e-3 s 16 6.80543e-3 s
MPI Scatter 111.963e-3 s 111.963e-3 s 10 11.1963e-3 s
MPI Wtime 124e-6 s 124e-6 s 64 1.9375e-6 s
MPI Type create resized 115e-6 s 115e-6 s 18 6.38889e-6 s
MPI Comm rank 5e-6 s 5e-6 s 16 312.5e-9 s
MPI Comm size 92e-6 s 92e-6 s 16 5.75e-6 s
MPI Type commit 190e-6 s 190e-6 s 36 5.27778e-6 s
MPI Type free 247.999e-6 s 247.999e-6 s 36 6.88886e-6 s
MPI Cart coords 108e-6 s 108e-6 s 272 397.059e-9 s
MPI Comm split 1.714e-3 s 1.714e-3 s 32 53.5625e-6 s
MPI Dims create 223e-6 s 223e-6 s 16 13.9375e-6 s
MPI Cart create 2.278e-3 s 2.278e-3 s 16 142.375e-6 s
0.490686 s


Intel Trace Analyzer - [I:/mnt/storage/home/hpcusers/hpcuser121/MATMUL_MPI/MATMUL_MPI_TRACE_2x8.stf]
File Options Project Windows Help
View Charts Navigate Advanced Layout
0.050 490 - 0.100 184 : 0.049 694
Seconds
All_Processes
All Functions
P0
P1
P2
P3
P4
P5
P6
P7
P8
P9
P10
P11
P12
P13
P14
P15
User Code
MPI_Bcast
MPI_Scatter
Flat Profile
Load Balance
Call Tree
Call Graph
Group All_Processes
Name
TSelf
TSelf
TTotal
#Calls
TSelf /Call
Group All Processes
MPI Type vector 16e-6 s 16e-6 s 2 8e-6 s
User Code 240.572e-3 s 793.84e-3 s 1 240.572e-3 s
MPI Bcast 436.766e-3 s 436.766e-3 s 32 13.6489e-3 s
MPI Scatter 111.963e-3 s 111.963e-3 s 10 11.1963e-3 s
MPI Wtime 85e-6 s 85e-6 s 32 2.65625e-6 s
MPI Type create resized 17e-6 s 17e-6 s 2 8.5e-6 s
MPI Comm rank 5e-6 s 5e-6 s 16 312.5e-9 s
MPI Comm size 92e-6 s 92e-6 s 16 5.75e-6 s
MPI Type commit 21e-6 s 21e-6 s 4 5.25e-6 s
MPI Type free 50e-6 s 50e-6 s 4 12.5e-6 s
MPI Cart coords 38e-6 s 38e-6 s 16 2.375e-6 s
MPI Comm split 1.714e-3 s 1.714e-3 s 32 53.5625e-6 s
MPI Dims create 223e-6 s 223e-6 s 16 13.9375e-6 s
MPI Cart create 2.278e-3 s 2.278e-3 s 16 142.375e-6 s

8x2:

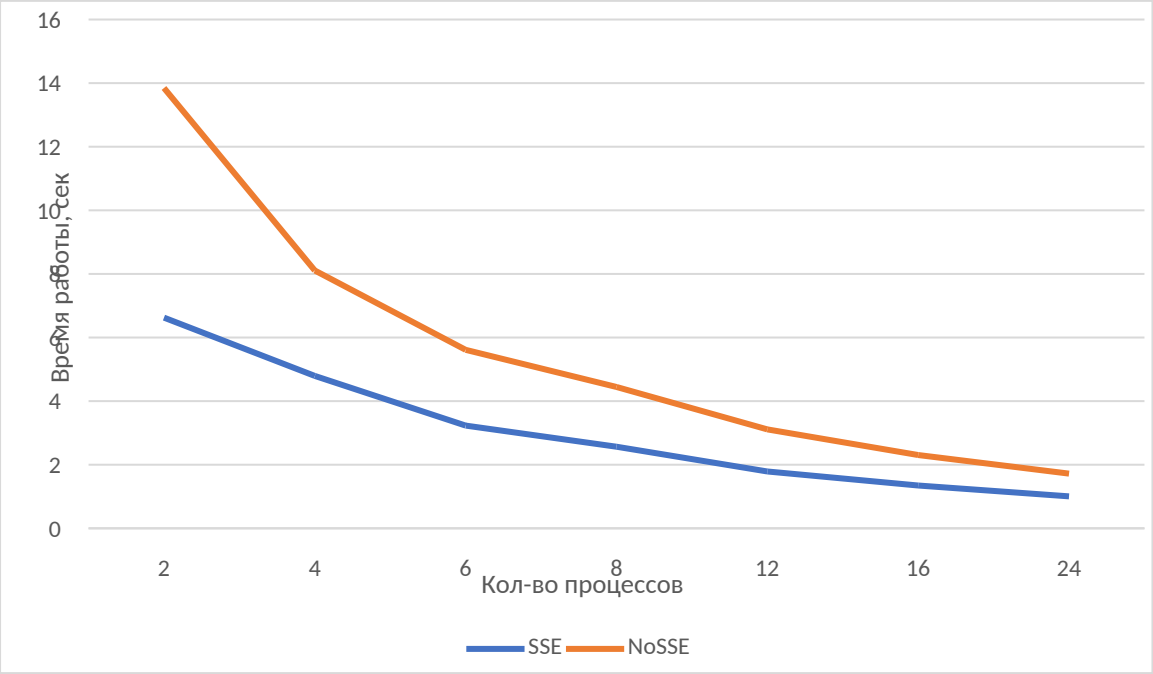

Intel Trace Analyzer - [I:/mnt/storage/home/hpcusers/hpcuser121/MATMUL_MPI/MATMUL_MPI_TRACE_8x2.stf]
File Options Project Windows Help
View Charts Navigate Advanced Layout
Seconds
All_Processes
All Functions
P0 P1 P2 P3 P4 P5 P6 P7 P8 P9 P10 P11 P12 P13 P14 P15
User Code
MPI Gatherv
MPI Scatter
MPI Bcast
Flat Profile
Load Balance
Call Tree
Call Graph
Group All_Processes
Name TSelf TSelf TTotal #Calls TSelf /Call
Group All Processes
MPI Comm free 197e-6 s 197e-6 s 48 4.10417e-6 s
MPI Type vector 192e-6 s 192e-6 s 24 8e-6 s
User Code 4.96802 s 7.70731 s 0 n.a.
MPI Bcast 319.295e-3 s 319.295e-3 s 32 9.97797e-3 s
MPI Gatherv 2.22164 s 2.22164 s 16 138.852e-3 s
MPI Scatter 197.146e-3 s 197.146e-3 s 10 19.7146e-3 s
MPI Wtime 114e-6 s 114e-6 s 64 1.78125e-6 s
MPI Type create resized 152e-6 s 152e-6 s 24 6.33333e-6 s
MPI Type commit 203e-6 s 203e-6 s 48 4.22917e-6 s
MPI Type free 294e-6 s 294e-6 s 48 6.125e-6 s
MPI Cart coords 53e-6 s 53e-6 s 256 207.031e-9 s


Intel Trace Analyzer - [I:/mnt/storage/home/hpcusers/hpcuser121/MATMUL_MPI/MATMUL_MPI_TRACE_8x2.stf]
File Options Project Windows Help
View Charts Navigate Advanced Layout
Seconds
All_Processes
All Functions
P0 P1 P2 P3 P4 P5 P6 P7 P8 P9 P10 P11 P12 P13 P14 P15
User Code
MPI Scatter
MPI Bcast
Flat Profile
Load Balance
Call Tree
Call Graph
Group All_Processes
Name TSelf TSelf TTotal #Calls TSelf /Call
Group All Processes
MPI Type vector 73e-6 s 73e-6 s 8 9.125e-6 s
User Code 254.681e-3 s 771.565e-3 s 0 n.a.
MPI Bcast 319.295e-3 s 319.295e-3 s 32 9.97797e-3 s
MPI Scatter 197.146e-3 s 197.146e-3 s 10 19.7146e-3 s
MPI Wtime 75e-6 s 75e-6 s 32 2.34375e-6 s
MPI Type create resized 62e-6 s 62e-6 s 8 7.75e-6 s
MPI Type commit 83e-6 s 83e-6 s 16 5.1875e-6 s
MPI Type free 150e-6 s 150e-6 s 16 9.375e-6 s

Приложение 5. *Замеры времени*

Время работы, сек

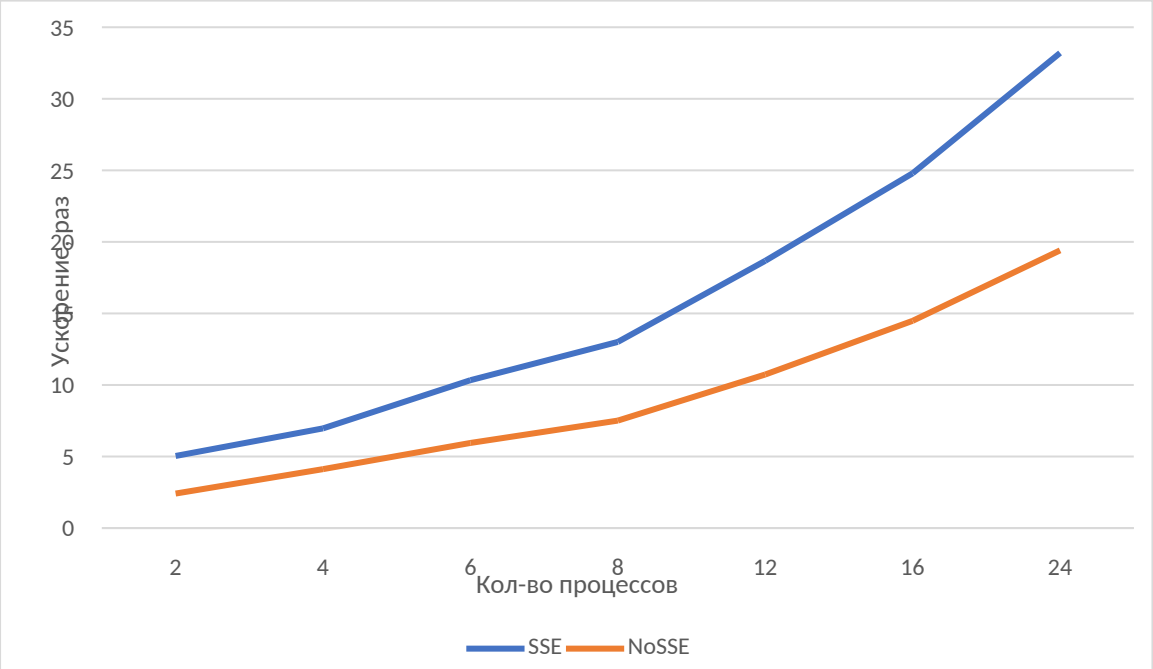
	2	4	6	8	12	16	24
SSE	6,62566	4,79269	3,23063	2,56637	1,78766	1,34649	1,00546
NoSSE	13,8463	8,10751	5,61282	4,44495	3,11108	2,30501	1,72028



Время работы последовательной программы: 33,3864с

Ускорение, раз

	2	4	6	8	12	16	24
SSE	5,038955	6,966109	10,33433	13,00919	18,67603	24,79513	33,2051
NoSSE	2,411215	4,11796	5,94824	7,511086	10,73145	14,48428	19,40754



Эффективность на процесс, %

	2	4	6	8	12	16	24
SSE	2,519477	1,741527	1,722389	1,626149	1,556336	1,549696	1,383546
NoSSE	1,205607	1,02949	0,991373	0,938886	0,894288	0,905267	0,808647