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»

студентки 2 курса, группы 21207

Черновской Яны Тихоновны

Направление 09.03.01 – «Информатика и вычислительная техника»

Преподаватель:

А.Ю. Власенко

Новосибирск 2023

СОДЕРЖАНИЕ

ЦЕЛЬ

Освоение концепции MPI коммуникаторов и декартовых топологий, а также концепции производных типов данных.

ЗАДАНИЕ

1. Реализовать параллельный алгоритм умножения матрицы на матрицу при 2D решетке процессов с соблюдением требований (см.выше).

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.

Алгоритм работы:

1.Создание решетки процессов p1 x p2.

2. Генерация матриц A[n1 x n2] и B[n2 x n3] на процессе с координатами (0;0) как одномерных массивов.

3. Раздача матрицы A по горизонтальным полосам на вертикальную линейку процессов
(0;0), (1;0), (2;0), …, (p1 - 1; 0) при помощи MPI_Scatter.

4. Определение нового производного типа данных для выбора из матрицы B вертикальных полос.

5. Раздача матрицы B по вертикальным полосам на горизонтальную линейку процессов (0;0), (0;1), (0;2), …, (0; p2 – 1) таким образом, что каждому процессу высылается только 1
элемент производного типа.

6. Каждый из процессов в левой вертикальной колонке ((1;0), (2;0), …, (p1 - 1; 0)) при помощи MPI_Bcast раздает свою полосу матрицы A всем процессам своей горизонтали. Т.е. процесс (1;0) раздает свою полосу процессам (1;1), (1;2),…

7. То же с полосами матрицы B, которые процессы первой горизонтали раздают по своим вертикальным столбцам решетки процессов (MPI_Bcast).

8. Теперь на каждом процессе есть по полосе A и по столбцу B, перемножаем, получаем миноры C.

9. Собираем всю C на процессе (0;0)

ОПИСАНИЕ РАБОТЫ

1. Была написана последовательная и параллельная программа с использованием MPI

2. Было измерено время последовательной программы:

3. Было измерено время работы параллельной программы на 24 процессах на сетках размером 2x12, 3x8, 4x6, 6x4, 8x3, 12x2

```
Number of MPI process: 24
2 12
Total time: 36.704099
3 8
Total time: 36.207232
4 6
Total time: 35.054315
6 4
Total time: 35.057117
8 3
Total time: 35.356191
12 2
Total time: 36.021557
```

4. Были построены графики времени, ускорения и эффективности, где

а) Ускорение: Sp = T1 / Tp, где T1 - время работы последовательной программы, Tp - время работы параллельной программы на p процессах

б) Эффективность Ep = Sp / p * 100%.

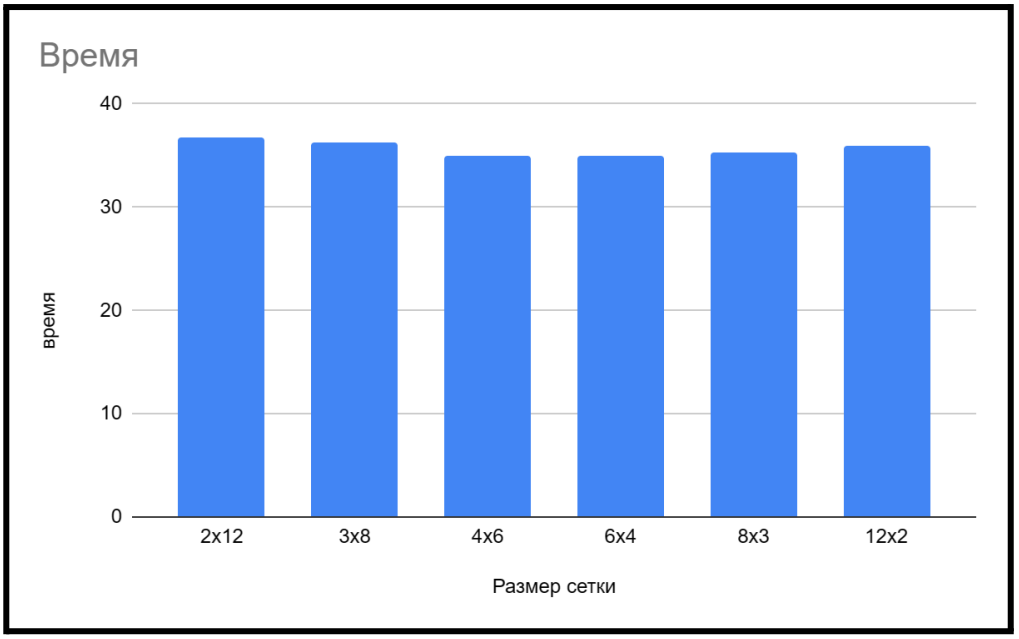


Рис.1 График времени

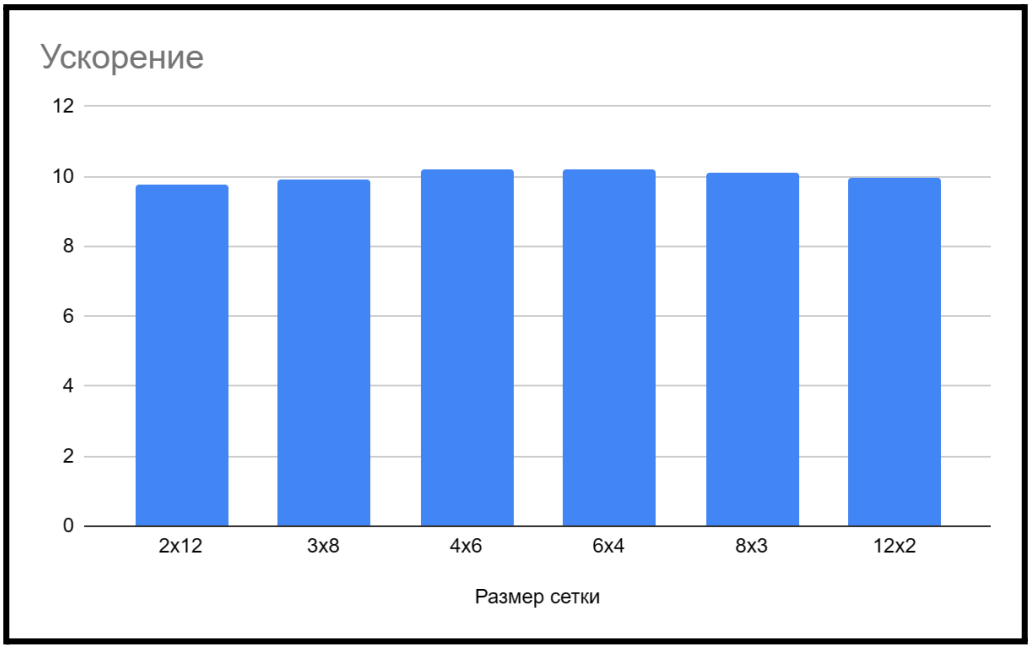


Рис 2. График ускорения

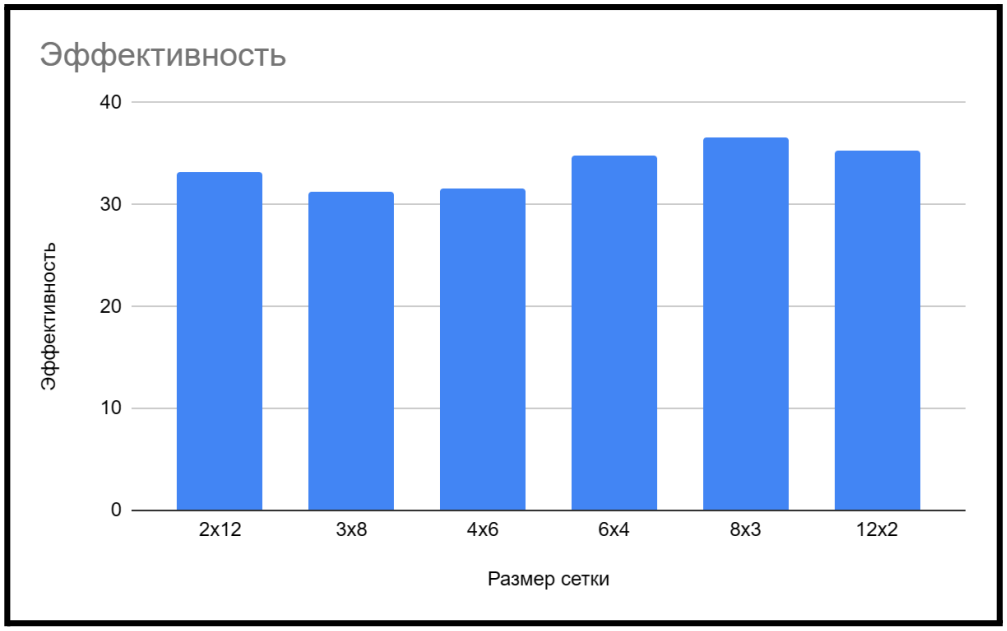
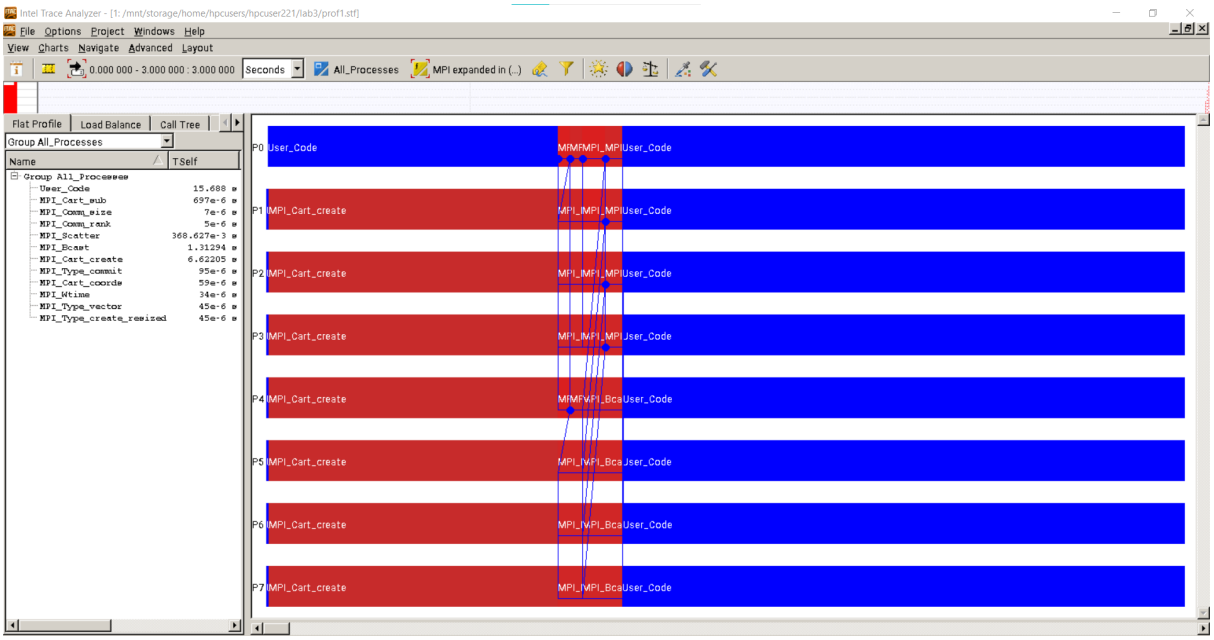
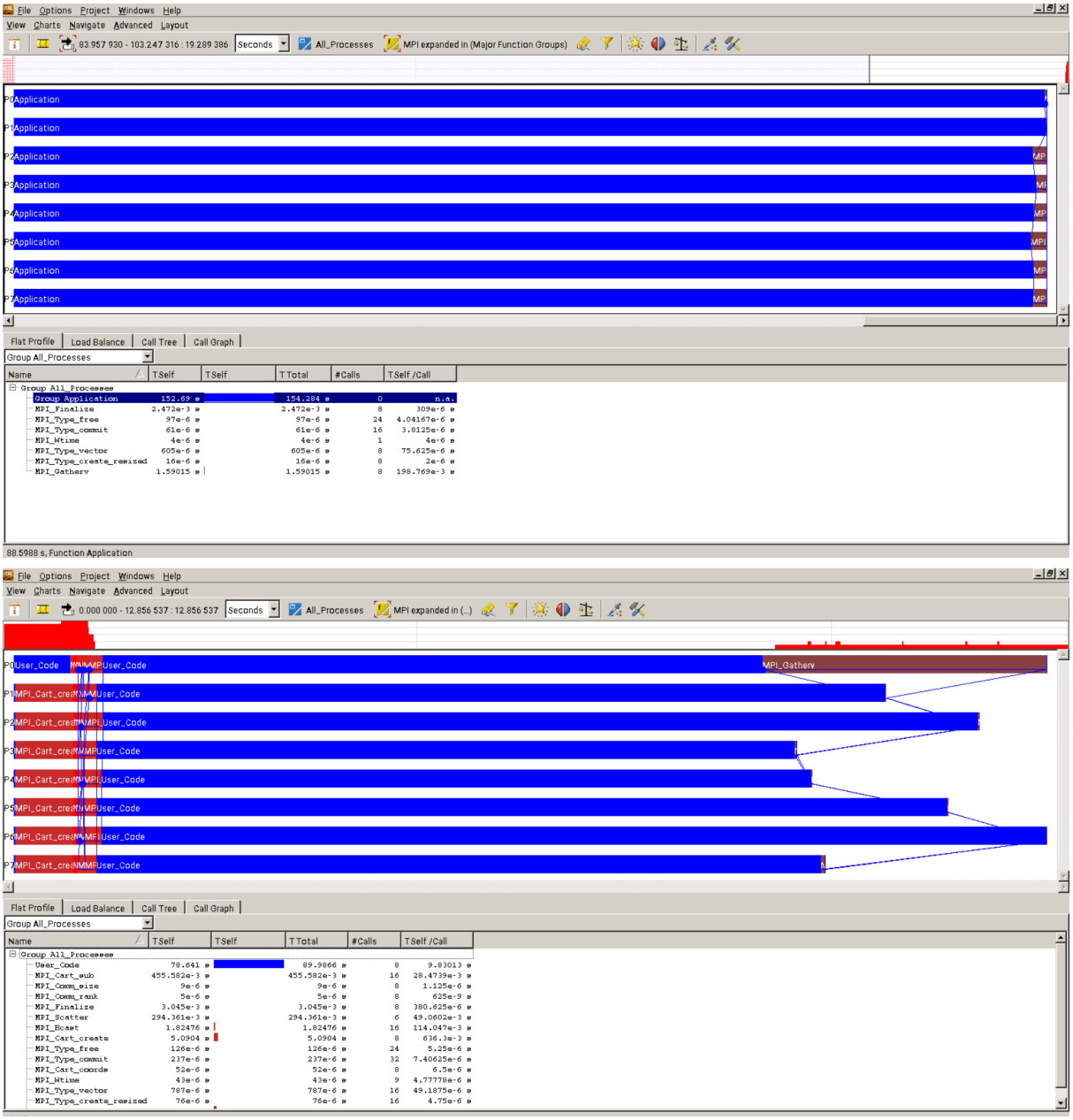


Рис 3. График эффективности

4. Было выполнено профилирование

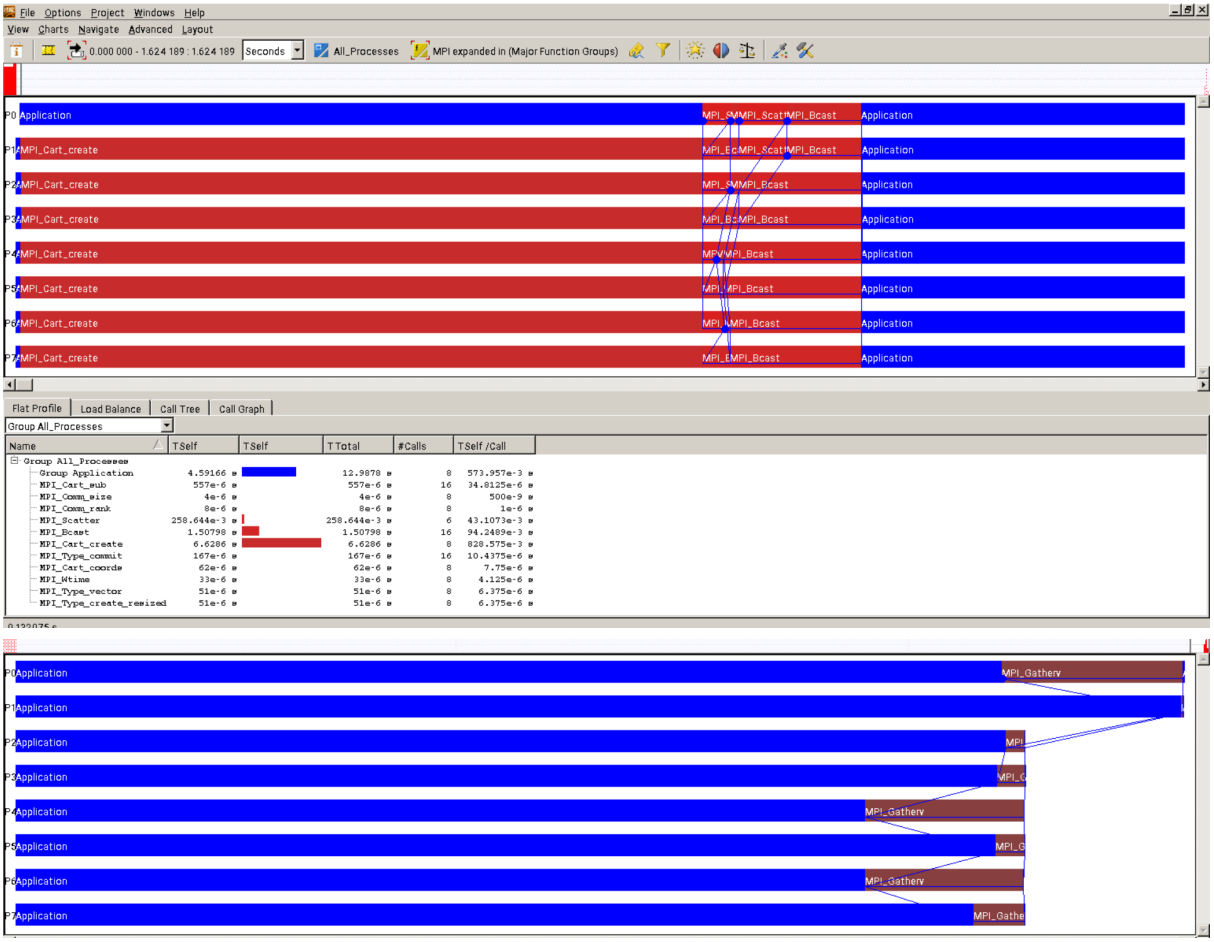
На сетке 2x4




File Options Project Windows Help
View Charts Navigate Advanced Layout
83.957 930 - 103.247 316 : 19.289 386
Seconds
All_Processes
MPI expanded in (Major Function Groups)
P0 Application
P1 Application
P2 Application
P3 Application
P4 Application
P5 Application
P6 Application
P7 Application
Flat Profile
Load Balance
Call Tree
Call Graph
Group All_Processes
Name
TSelf
TSelf
T Total
#Calls
TSelf /Call
Group All_Processes
Group Application 152.69 s n.a.
MPI_Finalize 2.472e-3 s 2.472e-3 s 8 309e-6 s
MPI_Type_free 97e-6 s 97e-6 s 24 4.04167e-6 s
MPI_Type_commit 61e-6 s 61e-6 s 16 3.8125e-6 s
MPI_Wtime 4e-6 s 4e-6 s 1 4e-6 s
MPI_Type_vector 605e-6 s 605e-6 s 8 75.625e-6 s
MPI_Type_create_resized 16e-6 s 16e-6 s 8 2e-6 s
MPI_Gatherv 1.59015 s 1.59015 s 8 198.769e-3 s
88.5988 s, Function Application

File Options Project Windows Help
View Charts Navigate Advanced Layout
0.000 000 - 12.856 537 : 12.856 537
Seconds
All_Processes
MPI expanded in (...)
P0 User_Code MPI_Gatherv
P1 MPI_Cart_cre User_Code
P2 MPI_Cart_cre User_Code
P3 MPI_Cart_cre User_Code
P4 MPI_Cart_cre User_Code
P5 MPI_Cart_cre User_Code
P6 MPI_Cart_cre User_Code
P7 MPI_Cart_cre User_Code
Flat Profile
Load Balance
Call Tree
Call Graph
Group All_Processes
Name
TSelf
TSelf
T Total
#Calls
TSelf /Call
Group All_Processes
User_Code 78.641 s 89.9866 s 8 9.83013 s
MPI_Cart_sub 455.582e-3 s 455.582e-3 s 16 28.4739e-3 s
MPI_Comm_size 9e-6 s 9e-6 s 8 1.125e-6 s
MPI_Comm_rank 5e-6 s 5e-6 s 8 625e-9 s
MPI_Finalize 3.045e-3 s 3.045e-3 s 8 380.625e-6 s
MPI_Scatter 294.361e-3 s 294.361e-3 s 6 49.0602e-3 s
MPI_Bcast 1.82476 s 1.82476 s 16 114.047e-3 s
MPI_Cart_create 5.0904 s 5.0904 s 8 636.3e-3 s
MPI_Type_free 126e-6 s 126e-6 s 24 5.25e-6 s
MPI_Type_commit 237e-6 s 237e-6 s 32 7.40625e-6 s
MPI_Cart_coords 52e-6 s 52e-6 s 8 6.5e-6 s
MPI_Wtime 43e-6 s 43e-6 s 9 4.77778e-6 s
MPI_Type_vector 787e-6 s 787e-6 s 16 49.1875e-6 s
MPI_Type_create_resized 76e-6 s 76e-6 s 16 4.75e-6 s

На сетке 4x2


File Options Project Windows Help
View Charts Navigate Advanced Layout
0.000 000 - 1.624 189 : 1.624 189
Seconds
All_Processes
MPI expanded in (Major Function Groups)
P0 Application
P1 MPI_Cart_create
P2 MPI_Cart_create
P3 MPI_Cart_create
P4 MPI_Cart_create
P5 MPI_Cart_create
P6 MPI_Cart_create
P7 MPI_Cart_create
MPI_Scatt MPI_Bcast
Application
Flat Profile
Load Balance
Call Tree
Call Graph
Group All_Processes
Name
TSelf
TSelf
T Total
#Calls
TSelf /Call
Group All_Processes
Group Application 4.59166 s 12.9878 s 8 573.957e-3 s
MPI_Cart_sub 557e-6 s 557e-6 s 16 34.8125e-6 s
MPI_Comm_size 4e-6 s 4e-6 s 8 500e-9 s
MPI_Comm_rank 8e-6 s 8e-6 s 8 1e-6 s
MPI_Scatter 258.644e-3 s 258.644e-3 s 6 43.1073e-3 s
MPI_Bcast 1.50798 s 1.50798 s 16 94.2489e-3 s
MPI_Cart_create 6.6286 s 6.6286 s 8 828.575e-3 s
MPI_Type_commit 167e-6 s 167e-6 s 16 10.4375e-6 s
MPI_Cart_coords 62e-6 s 62e-6 s 8 7.75e-6 s
MPI_Wtime 33e-6 s 33e-6 s 8 4.125e-6 s
MPI_Type_vector 51e-6 s 51e-6 s 8 6.375e-6 s
MPI_Type_create_resized 51e-6 s 51e-6 s 8 6.375e-6 s

P0 Application
P1 Application
P2 Application
P3 Application
P4 Application
P5 Application
P6 Application
P7 Application
MPI_Gatherv

ЗАКЛЮЧЕНИЕ

Были освоены концепции MPI коммуникаторов и декартовых топологий, а также концепции производных типов данных.

ПРИЛОЖЕНИЕ

Приложение 1. Полный листинг последовательной программы на C

```c
#include <stdio.h>
#include <time.h>
#include <stdlib.h>

const int n1 = 2400;
const int n2 = 5500;
const int n3 = 4800;

void generate_matrix(double* matrix, const int size1, int size2){
    for(int i = 0; i < size1; i++){
        for(int j = 0; j < size2; j++){
            matrix[i * size2 + j] = (double)rand() / RAND_MAX * 20.0 - 10.0;
        }
    }
}

void mul_matrix(const double* matrix1, const double* matrix2, double* result, int size1, int size2, int size3){
    for (int i = 0; i < size1; i++)
        for (int j = 0; j < size3; j++)
            for (int k = 0; k < size2; k++)
                result[i*size3 + j] += matrix1[i*size2 + k] * matrix2[k*size3 + j];
}

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

    double* a_matrix;
    double* b_matrix;
    double* c_matrix;

    a_matrix = malloc(sizeof(double) * n1 * n2);
    b_matrix = malloc(sizeof(double) * n2 * n3);
    c_matrix = calloc(n1 * n3, sizeof(double));
    generate_matrix(a_matrix, n1, n2);
    generate_matrix(b_matrix, n2, n3);

    time_t begin = time(NULL);
    mul_matrix(a_matrix, b_matrix, c_matrix, n1, n2, n3);
    time_t end = time(NULL);
    printf("Total time is %ld seconds\n", (end - begin));
    free(a_matrix);
    free(b_matrix);
    free(c_matrix);

```

```
44.     return EXIT_SUCCESS;
45. }
46.
```

Приложение 1. Полный листинг параллельной программы на C

```c
#include "stdio.h"
#include "stdlib.h"
#include "mpi.h"

#define X 0
#define Y 1

const int n1 = 2400;
const int n2 = 5500;
const int n3 = 4800;

int p1 = 2;
int p2 = 12;

void generate_matrix(double* matrix, const int size1, int size2){
    for(int i = 0; i < size1; i++){
        for(int j = 0; j < size2; j++){
            matrix[i * size2 + j] = (double)rand() / RAND_MAX * 20.0 - 10.0;
        }
    }
}

void mul_matrix(const double* matrix1, const double* matrix2, double* result, int
size1, int size2, int size3){
    for (int i = 0; i < size1; i++)
        for (int j = 0; j < size3; j++)
            for (int k = 0; k < size2; k++)
                result[i*size3 + j] += matrix1[i*size2 + k] * matrix2[k*size3 + j];
}

void print_matrix(const double* matrix, int row_count, int col_count) {
    for (int i = 0; i < row_count; i++) {
        for (int j = 0; j < col_count; j++)
            printf("%2.4f ", matrix[i * col_count + j]);
        printf("\n");
    }
}


void create_cartesian_lattice(MPI_Comm* cartesian_lattice){
    int dimensions_size[] = {p1, p2};
    int periodic_status[] = {1, 1};

    MPI_Cart_create(MPI_COMM_WORLD, 2, dimensions_size, periodic_status,
                    1, cartesian_lattice);

```

```
46.  }
47.
48.  void create_sub_comm(MPI_Comm* cartesian_lattice, MPI_Comm* row,
     MPI_Comm* column){
49.    int varying_coords[2];
50.    varying_coords[X] = 0;
51.    varying_coords[Y] = 1;
52.    MPI_Cart_sub(*cartesian_lattice, varying_coords, row);
53.
54.    varying_coords[X] = 1;
55.    varying_coords[Y] = 0;
56.    MPI_Cart_sub(*cartesian_lattice, varying_coords, column);
57.  }
58.
59.  void gather_matrix_c(double* c_matrix, double* c_part, int a_part_size, int
     b_part_size, int proc_number){
60.    MPI_Datatype block_not_resized, block_resized;
61.
62.    MPI_Type_vector(a_part_size, b_part_size, n3, MPI_DOUBLE,
     &block_not_resized);
63.    MPI_Type_commit(&block_not_resized);
64.
65.    MPI_Type_create_resized(block_not_resized, 0, b_part_size * sizeof(double),
     &block_resized);
66.    MPI_Type_commit(&block_resized);
67.
68.    int *recv_counts = malloc(sizeof(int) * proc_number);
69.    int *displs = malloc(sizeof(int) * proc_number);
70.
71.    int block_size = a_part_size * b_part_size;
72.
73.    for (int i = 0; i < p1; ++i)
74.      for (int j = 0; j < p2; ++j)
75.      {
76.        recv_counts[i * p2 + j] = 1;
77.        displs[i * p2 + j] = j + i * p2 * a_part_size;
78.      }
79.
80.    MPI_Gatherv(c_part, block_size, MPI_DOUBLE, c_matrix,
81.          recv_counts, displs, block_resized, 0, MPI_COMM_WORLD);
82.
83.    MPI_Type_free(&block_resized);
84.    MPI_Type_free(&block_not_resized);
85.  }
86.
87.  int main(int argc, char** argv){
88.
89.    int proc_number, proc_rank;
```

```
90.
91.     double* a_matrix;
92.     double* b_matrix;
93.     double* c_matrix;
94.
95.     double* a_part;
96.     double* b_part;
97.     double* c_part;
98.
99.     MPI_Comm cartesian_lattice;
100.    MPI_Comm row;
101.    MPI_Comm column;
102.
103.    int a_part_size;
104.    int b_part_size;
105.
106.    MPI_Init(&argc, &argv);
107.    MPI_Comm_size(MPI_COMM_WORLD, &proc_number);
108.    MPI_Comm_rank(MPI_COMM_WORLD, &proc_rank);
109.
110.    a_part_size = n1 / p1;
111.    b_part_size = n3 / p2;
112.
113.    p1 = atoi(argv[1]);
114.    p2 = atoi(argv[2]);
115.
116.    a_part = malloc(sizeof(double) * n2 * a_part_size);
117.    b_part = malloc(sizeof(double) * n2 * b_part_size);
118.    c_part = calloc(a_part_size * b_part_size, sizeof(double));
119.
120.    double start_time, end_time;
121.
122.    if (proc_rank == 0){
123.        a_matrix = malloc(sizeof(double) * n1 * n2);
124.        b_matrix = malloc(sizeof(double) * n2 * n3);
125.        c_matrix = calloc(n1 * n3, sizeof(double));
126.        generate_matrix(a_matrix, n1, n2);
127.        generate_matrix(b_matrix, n2, n3);
128.    }
129.
130.    start_time = MPI_Wtime();
131.    //create communicators
132.    create_cartesian_lattice(&cartesian_lattice);
133.
134.    int coordinates[2];
135.    MPI_Cart_coords(cartesian_lattice, proc_rank, 2,coordinates);
136.
137.    create_sub_comm(&cartesian_lattice, &row, &column);
```

```
138.
139.
140.     // distribute data
141.
142.     if (coordinates[Y] == 0){
143.         MPI_Scatter(a_matrix, a_part_size * n2, MPI_DOUBLE, a_part,
144.                 a_part_size * n2, MPI_DOUBLE, 0, column);
145.     }
146.
147.     MPI_Bcast(a_part, a_part_size * n2, MPI_DOUBLE, 0, row);
148.
149.     MPI_Datatype column_not_resized, column_resized;
150.     MPI_Type_vector(n2, b_part_size, n3, MPI_DOUBLE, &column_not_resized);
151.     MPI_Type_commit(&column_not_resized);
152.
153.     MPI_Type_create_resized(column_not_resized, 0, b_part_size * sizeof(double),
    &column_resized);
154.     MPI_Type_commit(&column_resized);
155.
156.     if (coordinates[X] == 0) {
157.         MPI_Scatter(b_matrix, 1, column_resized, b_part, n2 * b_part_size,
158.                 MPI_DOUBLE, 0, row);
159.     }
160.
161.     MPI_Bcast(b_part, b_part_size * n2, MPI_DOUBLE, 0, column);
162.
163.
164.     mul_matrix(a_part, b_part, c_part, a_part_size, n2, b_part_size);
165.
166.
167.     //gather matrix
168.     gather_matrix_c(c_matrix, c_part, a_part_size, b_part_size, proc_number);
169.
170.     if (proc_rank == 0){
171.         end_time = MPI_Wtime();
172.         /*print_matrix(c_matrix, n1, n3);
173.         printf("_------------------------------------------------------------\n");
174.         double* new_matrix = calloc(n1 * n3, sizeof(double));
175.         mul_matrix(a_matrix, b_matrix, new_matrix, n1, n2,n3);
176.         print_matrix(new_matrix, n1, n3);*/
177.         printf("%d %d\n", p1, p2);
178.         printf("Total time: %f\n", end_time - start_time);
179.         free(a_matrix);
180.         free(b_matrix);
181.         free(c_matrix);
182.     }
183.
184.     free(a_part);
```

```
185.     free(b_part);
186.     free(c_part);
187.
188.     MPI_Type_free(&column_not_resized);
189.     MPI_Type_free(&column_resized);
190.     MPI_Finalize();
191.     return EXIT_SUCCESS;
192. }
193.
```

Приложение 3. Скрипт для запуска параллельной программы

```
1.  #!/bin/bash
2.
3.  #PBS -l walltime=00:01:00
4.  #PBS -l select=2:ncpus=12:mpiprocs=12:mem=2000m,place=scatter
5.  #PBS -m n
6.
7.  cd $PBS_O_WORKDIR
8.
9.  MPI_NP=$(wc -l $PBS_NODEFILE | awk '{ print $1 }')
10. echo "Number of MPI process: $MPI_NP"
11.
12. mpirun -hostfile $PBS_NODEFILE -perhost 1 -np $MPI_NP ./matrix_mult 2 12
13. mpirun -hostfile $PBS_NODEFILE -perhost 1 -np $MPI_NP ./matrix_mult 3 8
14. mpirun -hostfile $PBS_NODEFILE -perhost 1 -np $MPI_NP ./matrix_mult 4 6
15. mpirun -hostfile $PBS_NODEFILE -perhost 1 -np $MPI_NP ./matrix_mult 6 4
16. mpirun -hostfile $PBS_NODEFILE -perhost 1 -np $MPI_NP ./matrix_mult 8 3
17. mpirun -hostfile $PBS_NODEFILE -perhost 1 -np $MPI_NP ./matrix_mult 12 2
```

Приложение 4. Скрипт для запуска параллельной программы с профилированием

```
#!/bin/bash

#PBS -l walltime=00:10:00
#PBS -l select=1:ncpus=8:mpiprocs=8:mem=2000m,place=scatter
#PBS -m n

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 ./prof
```