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 2 курса, 19202 группы

Ивакина Александра Олеговича

Направление 09.03.01 – «Информатика и вычислительная техника»

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

Новосибирск 2021

СОДЕРЖАНИЕ

ЦЕЛЬ

Изучить создание производных типов и коммуникаторов с использованием технологии MPI.

ЗАДАНИЕ

Составить программу, разбивающую расчёт умножения матрицы на решетку, и исследовать зависимость скорости работы программы от формы решётки.

ОПИСАНИЕ РАБОТЫ

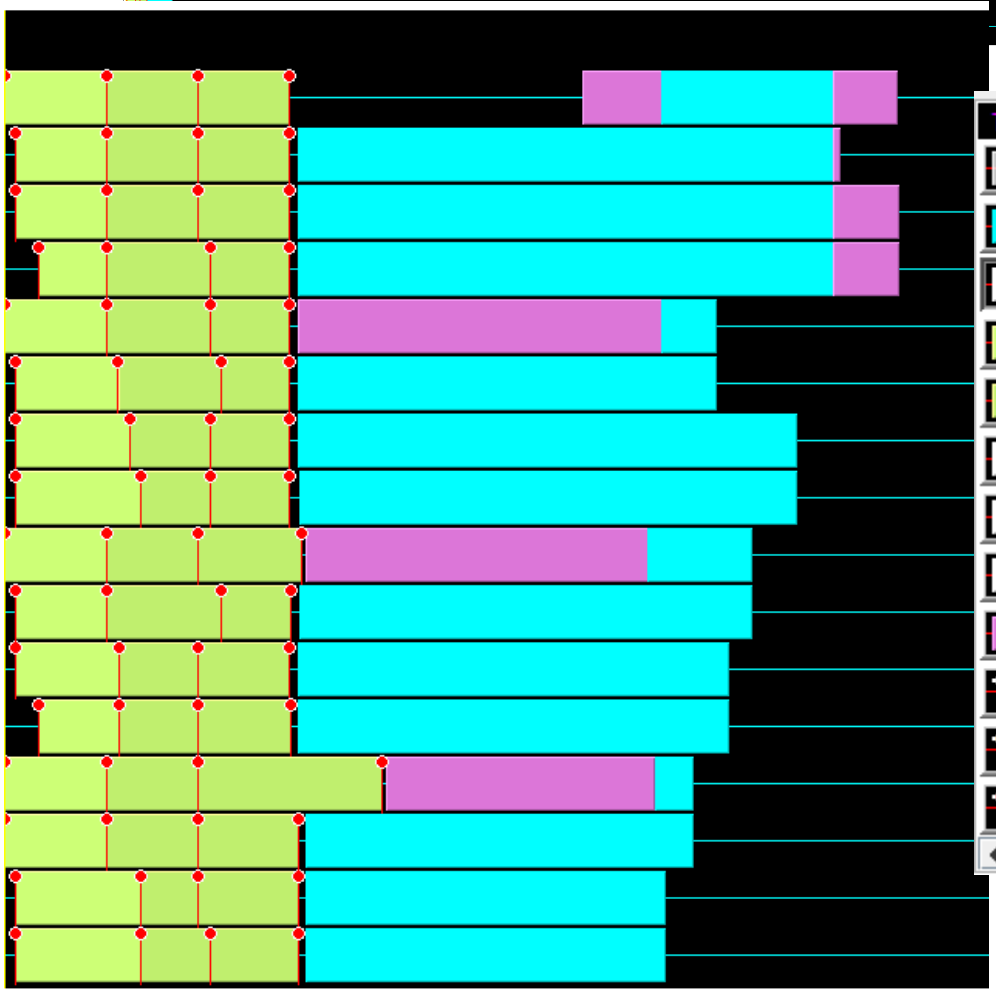
Для перемножения матрицы были разбиты на столбики и ряды, которые перемножались по отдельности и собирались в решётке результирующей матрицы. Для правильного разбиения были введены производные типы и коммуникаторы.

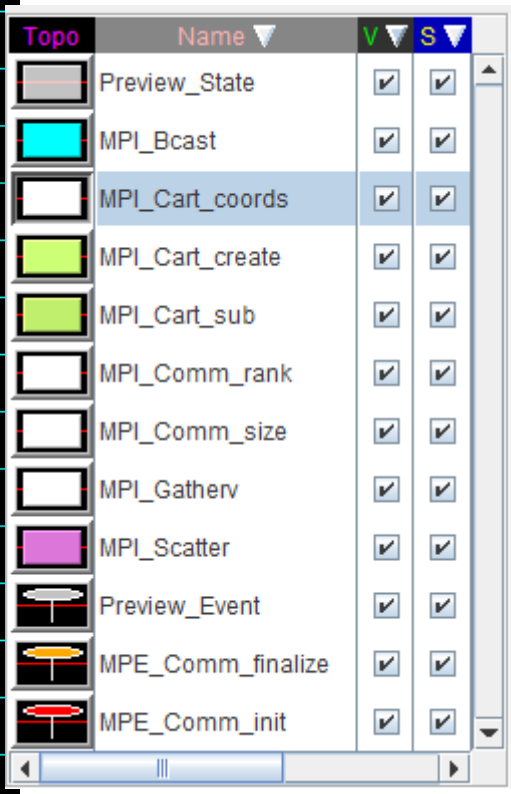
Матрицы, разбитые на решетку формы 4x4 подсчитывалась 5,5 секунд.

2x8 – 4,93 секунд.

16x1 – 5,12 секунд.

Профилирование для топологии 4x4:





ЗАКЛЮЧЕНИЕ

В результате проделанной работы были изучены и разобраны способы задания произвольных типов данных для любого нужного нам разбиения.

Приложение 1. Код программы

```
#include <stdio.h>
1.#include <stdlib.h>
2.#include <math.h>
3.#include <time.h>
4.#include <string.h>
5.#include <mpi.h>
6.
7.double rand_double(double min, double max) {
8.return min + (rand() / (RAND_MAX / (max - min)));
9.}
10.
11.void make_random_matrix(double* matrix, int n1, int n2) {
12.for (int i = 0; i < n1; i++) {
13.for (int j = i; j < n2; j++) {
14.matrix[i*n1 + j] = rand_double(0, 1);
15.}
16.}
17.}
18.
19.void fill_with_zero(double* matrix, int n1, int n2) {
20.for (int i = 0; i < n1; i++) {
21.for (int j = 0; j < n2; j++) {
22.matrix[i * n2 + j] = 0;
23.}
24.}
25.}
26.
27.void make_id_matrix(double* matrix, int n1, int n2) {
28.for (int i = 0; i < n1; i++) {
29.for (int j = 0; j < n2; j++) {
30.if (i == j) {
31.matrix[i * n2 + j] = 1;
32.}
33.else {
34.matrix[i * n2 + j] = 0;
35.}
36.}
37.}
38.}
39.
40.void multiply_matrix(double* A_matrix, double* B_matrix, double* C_matrix, int n1,
int n2, int n3) {
41.for (int i = 0; i < n1; i++) {
42.for (int j = 0; j < n3; j++) {
43.for (int k = 0; k < n2; k++) {
44.C_matrix[i*n3 + j] += A_matrix[i*n2 + k] * B_matrix[k*n3 + j];
45.}
46.}
47.}
48.}
49.
50.int grid[2];
51.
52.int n1 = 24;
53.int n2 = 8;
54.int n3 = 16;
55.int p1 = 2;
56.int p2 = 2;
57.
58.int main(int argc, char** argv) {
59.// SETTING UP
```

```
MPI_Init(&argc, &argv);
int proc_num;
MPI_Comm_size(MPI_COMM_WORLD, &proc_num);
int rank;
MPI_Comm_rank(MPI_COMM_WORLD, &rank);

// Creating grid
int dim[2];
dim[0] = p1;
dim[1] = p2;

int periodic[2];
periodic[0] = 1;
periodic[1] = 1;

MPI_Comm grid_comm;
MPI_Comm col_comm;
MPI_Comm row_comm;

MPI_Cart_create(MPI_COMM_WORLD, 2, dim, periodic, 0, &grid_comm);
MPI_Cart_coords(grid_comm, rank, 2, grid);

int dimen[2];
dimen[0] = 0;
dimen[1] = 1;
MPI_Cart_sub(grid_comm, dimen, &row_comm);

dimen[0] = 1;
dimen[1] = 0;
MPI_Cart_sub(grid_comm, dimen, &col_comm);

double* A_matrix;
double* B_matrix;
double* C_matrix;

// Creating matrixes
if (rank == 0) {
A_matrix = (double*)malloc(n1 * n2 * sizeof(double));
B_matrix = (double*)malloc(n3 * n2 * sizeof(double));
C_matrix = (double*)malloc(n1 * n3 * sizeof(double));

// Generating matrixies
make_random_matrix(A_matrix, n1, n2);
make_random_matrix(B_matrix, n2, n3);

fill_with_zero(C_matrix, n1, n3);
}

int height = n1 / p1;
int width = n3 / p2;

double* A_hold_matrix = (double*)malloc(height * n2 * sizeof(double));
double* B_hold_matrix = (double*)malloc(width * n2 * sizeof(double));
double* C_hold_matrix = (double*)malloc(width * height * sizeof(double));

fill_with_zero(C_hold_matrix, height, width);

// Datatype for columns
MPI_Datatype col, col_resized;

MPI_Type_vector(n2, width, n3, MPI_DOUBLE, &col);
MPI_Type_commit(&col);
```

```
MPI_Type_create_resized(col, 0, width * sizeof(double), &col_resized);
MPI_Type_commit(&col_resized);

double start;
if (rank == 0) {
start = MPI_Wtime();
}

// Scattering and broadcasting
// Rows
if (grid[1] == 0) {
MPI_Scatter(A_matrix, height * n2, MPI_DOUBLE, A_hold_matrix, height * n2,
MPI_DOUBLE, 0, col_comm);
}

MPI_Bcast(A_hold_matrix, height * n2, MPI_DOUBLE, 0, row_comm);

// Columns
if (grid[0] == 0) {
MPI_Scatter(B_matrix, 1, col_resized, B_hold_matrix, width * n2, MPI_DOUBLE, 0,
row_comm);
}

// ACTION
multiply_matrix(A_hold_matrix, B_hold_matrix, C_hold_matrix, height, n2, width);

// Type for gathering from cells in C matrix
MPI_Datatype cell, cell_resized;

MPI_Type_vector(height, width, n3, MPI_DOUBLE, &cell);
MPI_Type_commit(&cell);

MPI_Type_create_resized(cell, 0, width * sizeof(double), &cell_resized);
MPI_Type_commit(&cell_resized);

// Gathering
int recv_counts[p1 * p2];
int displs[p1 * p2];
for (int i = 0; i < p1 * p2; i++) {
recv_counts[i] = 1;
displs[i] = (i / p2) * height + (i % p2) * width;
}

MPI_Gatherv(C_hold_matrix, height * width, MPI_DOUBLE, C_matrix, recv_counts,
displs, cell_resized, 0, MPI_COMM_WORLD);

// FINISHING
if (rank == 0) {
for (int i = 0; i < n1; i++) {
for (int j = 0; j < n3; j++) {
printf("%lf ", C_matrix[i * n3 + j]);
}
printf("\n");
}
double end = MPI_Wtime();
printf("Time taken: %lf seconds\n", end - start);
}

if (rank == 0) {
free(A_matrix);
free(B_matrix);
free(C_matrix);
}
free(A_hold_matrix);
```

```
184.free(B_hold_matrix);
185.free(C_hold_matrix);
186.MPI_Finalize();
187.return 0;
188.}
189.
```