###### 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, 18204 группы

Хорошавина Алексея Константиновича

Направление 09.03.01 – «Информатика и вычислительная техника»

Преподаватель:

Кандидат технических наук,

Доцент.

А.Ю. Власенко

Новосибирск 2019

Оглавление

[Задание 3](#_Toc37285410)

[Результаты измерений 4](#_Toc37285411)

[Графики зависимости времени выполнения 5](#_Toc37285412)

[Графики зависимости ускорения 5](#_Toc37285413)

[Графики зависимости эффективности 6](#_Toc37285414)

[Профилирование 7](#_Toc37285415)

[Заключение 8](#_Toc37285416)

[Приложение 9](#_Toc37285417)

**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.

# Результаты измерений

Матрица: N = 640, M = 640, K = 640;

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Решетка | Ядра | Время | Ускорение | Эффективность |
| 1x1 | 1 | 2,39 | 1 | 1 |
| 2x1 | 2 | 1,40 | 1,68 | 0,84 |
| 2x2 | 4 | 0,76 | 3,14 | 0,78 |
| 4x2 | 8 | 0,47 | 5,08 | 0,63 |
| 4x4 | 16 | 0,31 | 7,70 | 0,48 |

Матрица: N = 800, M = 800, K = 800;

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Решетка | Ядра | Время | Ускорение | Эффективность |
| 1x1 | 1 | 3,80 | 1 | 1 |
| 1x2 | 2 | 2,22 | 1,71 | 0,85 |
| 2x2 | 4 | 1,18 | 3,22 | 0,80 |
| 4x2 | 8 | 0,68 | 5,58 | 0,69 |
| 4x4 | 16 | 0,49 | 7,75 | 0,48 |

Матрица: N = 960, M = 960, K = 960;

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Решетка | Ядра | Время | Ускорение | Эффективность |
| 1x1 | 1 | 9,63 | 1 | 1 |
| 2x1 | 2 | 4,87 | 1,97 | 0,98 |
| 2x2 | 4 | 3,15 | 3,05 | 0,76 |
| 4x2 | 8 | 1,53 | 6,29 | 0,78 |
| 4x4 | 16 | 0,79 | 12,18 | 0,76 |

Матрица: N = 1600, M = 1600, K = 1600;

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Решетка | Ядра | Время | Ускорение | Эффективность |
| 1x1 | 1 | 51,84 | 1 | 1 |
| 1x2 | 2 | 25,95 | 1,99 | 0,99 |
| 2x2 | 4 | 13,26 | 3,90 | 0,97 |
| 4x2 | 8 | 7,80 | 6,64 | 0,83 |
| 4x4 | 16 | 3,46 | 14,98 | 0,93 |

# 

# Графики зависимости времени выполнения

# Графики зависимости ускорения

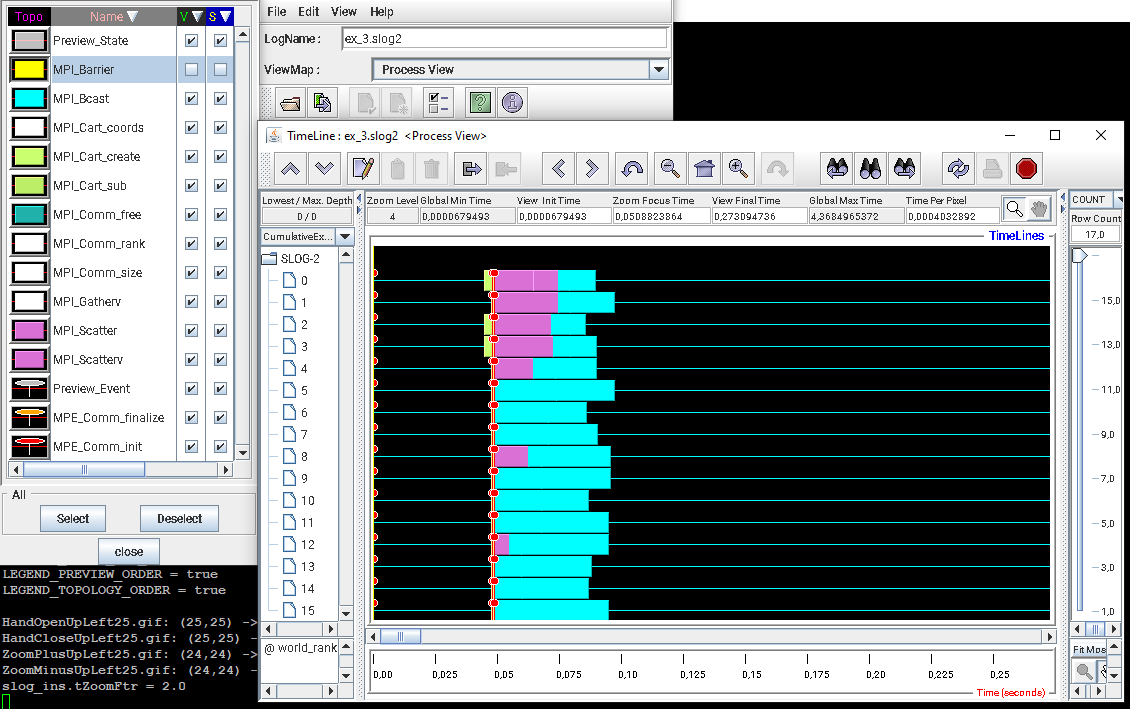
# 

# Графики зависимости эффективности

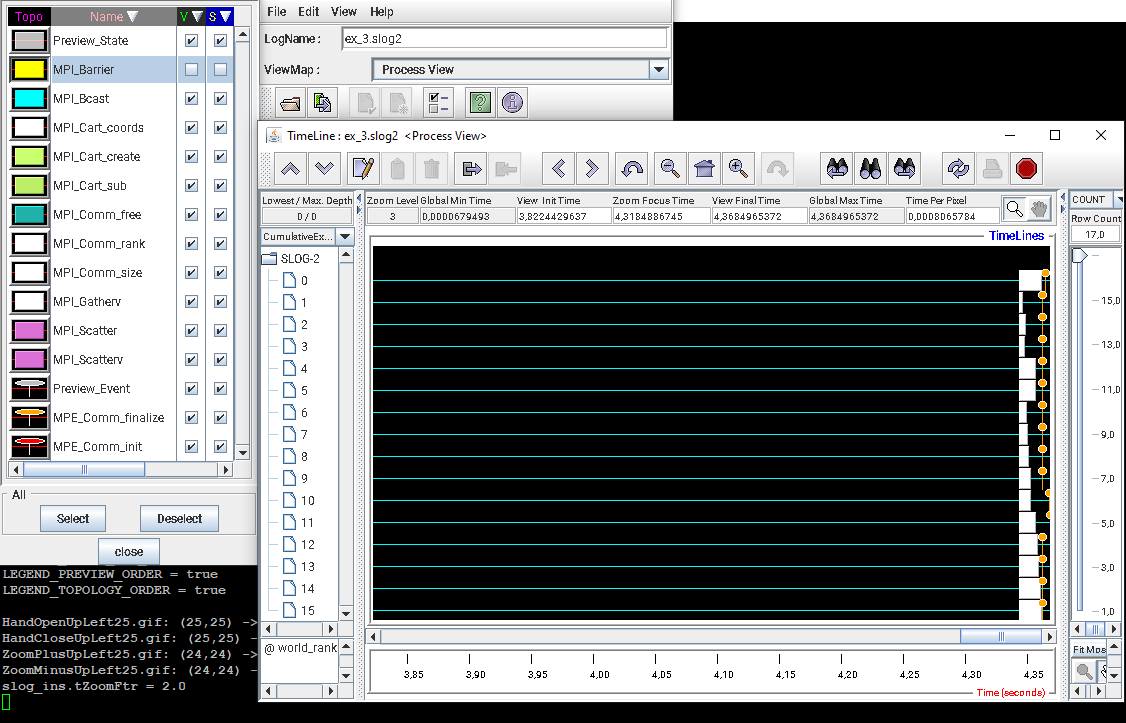
# Профилирование

Размер матрицы:(1600), размер решетки:(4х4)

Начало работы программы. Создание топологии и распределение данных.



Конец работы программы. Сборка матрицы.



# Заключение

Был реализован алгоритм умножения матрицы на матрицу с использованием топологии “2D решетка”. Были выполнены замеры времени при различных размерах матрицы и различных размерах решетки.

Из результатов видно, что при увеличении размера матрицы и размера решетки производительность программы увеличивается.

# Приложение

Листинг программы:

#include **<stdio.h>**#include **<stdlib.h>**#include **<mpi.h>**#define DIMENSION 2  
  
**void** setMatrix(**double** \*matrix, **int** size1, **int** size2, **int** value){  
 **for**(**int** i = 0; i < size1; ++i){  
 **for**(**int** j = 0; j < size2; ++j){  
 matrix[i\*size2 + j] = value;  
 }  
 }  
}  
  
**void** setType(**int** count, **int** blockLen, **int** offset, MPI\_Datatype \*type){  
 **long** lbTypeDouble, extentDouble;  
  
 MPI\_Type\_vector(count, blockLen, offset, MPI\_DOUBLE, type);  
 MPI\_Type\_get\_extent(MPI\_DOUBLE, &lbTypeDouble, &extentDouble);  
 MPI\_Type\_create\_resized(\*type, 0, extentDouble\* blockLen, type);  
 MPI\_Type\_commit(type);  
}  
  
**void** mul\_matrices(**double** \*result, **double** \*matrix\_1, **double** \*matrix\_2, **int** sizeN, **int** sizeM, **int** sizeK){  
 **for**(**int** i = 0; i < sizeN; i++){  
 **for**(**int** j = 0; j < sizeK; j++){  
 **for**(**int** k = 0; k < sizeM; k++){  
 result[i\*sizeK + j] += matrix\_1[i \* sizeM + k] \* matrix\_2[sizeK \* k + j];  
 }  
 }  
 }  
}  
  
**int** main(**int** argc, **char** \*argv[]) {  
 **int** size;  
 **int** rank;  
 **double** start, end;  
  
 MPI\_Datatype typeB, typeC;  
 MPI\_Comm comm2D = {0};  
 MPI\_Comm comm\_1D[DIMENSION];  
  
 **const int** N = 10;  
 **const int** M = 10;  
 **const int** K = 10;  
  
 **int** localN;  
 **int** localK;  
  
 **int** err = 0;  
  
 **int** p[DIMENSION];  
 **int** periods[DIMENSION] = {0};  
 **int** coords[DIMENSION];  
 **int** remains[DIMENSION] = {0};  
  
 **int** \*offsetB = NULL;  
 **int** \*countB = NULL;  
  
 **int** \*offsetC = NULL;  
 **int** \*countC = NULL;  
  
 **double** \*A = NULL;  
 **double** \*B = NULL;  
 **double** \*C = NULL;  
  
 **double** \*partA;  
 **double** \*partB;  
 **double** \*partC;  
  
 MPI\_Init(&argc, &argv);  
 MPI\_Comm\_size(MPI\_COMM\_WORLD, &size);  
 MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank);  
  
 **if**(argc == 3){  
 p[0] = atoi(argv[1]);  
 p[1] = atoi(argv[2]);  
  
 **if**(p[0]\*p[1] != size){  
 printf(**"Error. Wrong size of lattice\n"**);  
 MPI\_Abort(MPI\_COMM\_WORLD, err);  
 **return** 1;  
 }  
 }  
 **else**{  
 p[0] = 1;  
 p[1] = size;  
 }  
  
 **if**(0 == rank){  
 A = (**double** \*)malloc(N \* M \* **sizeof**(**double**));  
 B = (**double** \*)malloc(M \* K \* **sizeof**(**double**));  
 C = (**double** \*)malloc(N \* K \* **sizeof**(**double**));  
  
 **if**(A == NULL || B == NULL || C == NULL){  
 printf(**"Allocation fail!\n"**);  
 MPI\_Abort(MPI\_COMM\_WORLD, err);  
 **return** 1;  
 }  
  
 setMatrix(A, N, M, 1);  
 setMatrix(B, M, K, 2);  
 setMatrix(C, N, K, 0);  
 }  
  
 MPI\_Barrier(MPI\_COMM\_WORLD);  
  
 err = MPI\_Cart\_create(MPI\_COMM\_WORLD, DIMENSION, p, periods, 0, &comm2D);  
  
 **if**(0 != err){  
 printf(**"MPI\_Cart\_create error! %d\n"**, err);  
 MPI\_Abort(MPI\_COMM\_WORLD, err);  
 **return** 1;  
 }  
  
 **for**(**int** i = 0; i < DIMENSION; i++){  
 **for**(**int** j = 0; j < DIMENSION; j++){  
 remains[j] = (**int**)(i == j);  
 }  
  
 MPI\_Cart\_sub(comm2D, remains, &comm\_1D[i]);  
 }  
  
 localN = N / p[0];  
 localK = K / p[1];  
  
 partA = (**double** \*)malloc(localN \* M \* **sizeof**(**double**));  
 partB = (**double** \*)malloc(M \* localK \* **sizeof**(**double**));  
 partC = (**double** \*)malloc(localN \* localK \* **sizeof**(**double**));  
  
 **if**(partA == NULL || partB == NULL || partC == NULL){  
 printf(**"Allocation fail!\n"**);  
 **return** 1;  
 }  
  
 **if**(0 == rank){  
 *//Устанавливаем типы для полос матрицы B и подматриц C* setType(M, localK, K, &typeB);  
 setType(localN, localK, K, &typeC);  
 }  
  
 MPI\_Comm\_rank(comm2D, &rank);  
 MPI\_Cart\_coords(comm2D, rank, DIMENSION, coords);  
  
 start = MPI\_Wtime();  
  
 *//Разделение матрицы А на полосы вдоль координаты x* **if**(0 == coords[1]){  
 MPI\_Scatter(A, localN \* M, MPI\_DOUBLE, partA, localN \* M, MPI\_DOUBLE, 0, comm\_1D[0]);  
 }  
  
 *//Разделение матрицы B вдоль координаты у* **if**(0 == coords[0]){  
 offsetB = (**int** \*)malloc(p[1] \* **sizeof**(**double**));  
 countB = (**int** \*)malloc(p[1] \* **sizeof**(**double**));  
  
 **for**(**int** i = 0; i < p[1]; ++i){  
 offsetB[i] = i;  
 countB[i] = 1;  
 }  
  
 MPI\_Scatterv(B, countB, offsetB, typeB, partB, localK \* M, MPI\_DOUBLE, 0, comm\_1D[1]);  
 }  
  
 MPI\_Bcast(partA, localN \* M, MPI\_DOUBLE, 0, comm\_1D[1]);  
  
 MPI\_Bcast(partB, localK \* M, MPI\_DOUBLE, 0, comm\_1D[0]);  
  
 mul\_matrices(partC, partA, partB, localN, M, localK);  
  
 MPI\_Barrier(comm2D);  
  
 offsetC = (**int** \*)malloc(p[0] \* p[1] \* **sizeof**(**double**));  
 countC = (**int** \*)malloc(p[0] \* p[1] \* **sizeof**(**double**));  
  
 **for**(**int** i = 0; i < p[0]; i++){  
 **for**(**int** j = 0; j < p[1]; j++){  
 offsetC[i \* p[1] + j] = p[1] \* localN \* i + j;  
 countC[i \* p[1] + j] = 1;  
 }  
 }  
  
 MPI\_Barrier(comm2D);  
  
 MPI\_Gatherv(partC, localK \* localN, MPI\_DOUBLE, C, countC, offsetC, typeC, 0, comm2D);  
  
 end = MPI\_Wtime();  
  
 **if**(rank == 0){  
 printf(**"Time: %f \n"**, end - start);  
  
 **for**(**int** i = 0; i < N; i++){  
 **for**(**int** j = 0; j < K; j++){  
 printf(**"%.02f "**, C[i\*K + j]);  
 }  
 printf(**"\n"**);  
 }  
 }  
  
 MPI\_Barrier(comm2D);  
  
 free(A);  
 free(B);  
 free(C);  
  
 free(partA);  
 free(partB);  
 free(partC);  
  
 free(offsetB);  
 free(offsetC);  
 free(countB);  
 free(countC);  
  
 **if**(rank == 0){  
 MPI\_Comm\_free(&comm2D);  
 MPI\_Comm\_free(&comm\_1D[0]);  
 MPI\_Comm\_free(&comm\_1D[1]);  
  
 MPI\_Type\_free(&typeB);  
 MPI\_Type\_free(&typeC);  
 }  
  
 MPI\_Finalize();  
 **return** 0;  
}