МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ

по лабораторной работе №5

по дисциплине «Системы параллельной обработки данных»

Тема: УМНОЖЕНИЕ МАТРИЦ

Студентка гр. 5303 _____ Допира В.Е.

Преподаватель _____ Татаринов Ю.С.

Санкт-Петербург

2019

Формулировка задания

При реализации использование виртуальных архитектур ОБЯЗАТЕЛЬНО!

Задание:

1. Реализовать алгоритм умножения двух квадратных матриц.
   a. последовательный алгоритм
   b. параллельный алгоритм: 2-ва варианта умножения матриц при ленточной схеме разделения данных:

вариант А) - Матрица А разбивается на горизонтальные полосы, матрица В – делится на вертикальные полосы

вариант В) – матрица А и В – разбивается на горизонтальные полосы

2. Определить подзадачи (для решения задачи масштабируемости- размерность матрицы превышает число процессоров).
3. Определить информационные связи (простое решение этой проблемы – дублирование матрицы В во всех подзадачах – является, как правило, неприемлемым в силу больших затрат памяти для хранения данных).
4. Реализовать масштабирование и распределение подзадач по процессам.
5. Провести анализ эффективности (анализ эффективности алгоритмов – до проведения эксперимента).
6. Выполнить программную реализацию.
7. Привести результаты вычислительных экспериментов в виде таблицы, графика ускорения в зависимости от количества процессоров.

Сравнить экспериментальные и теоретические оценки времени выполнения. Оценить наиболее эффективную топологию сети для выбранных алгоритмов

Описание алгоритма с использованием аппарата Сетей Петри

P_0 - начальное состояние. Реализован параллельный ленточный алгоритм с использованием сдвига в кольце по модулю (топология кольцо). Нулевой процесс считывает нулевую строку. Осуществляется переход t_1 из P_0. Далее каждый процесс считывает последующие строки. Происходит переход к состояниям P_1, P_2, P_3. Процессы считывают столбцы матрицы В. Далее они

вычисляют часть матрицы произведения. Столбцы матрицы B сдвигаются вдоль кольца процессов. Нулевой процесс собирает результат в матрице C. Программа завершает работу.

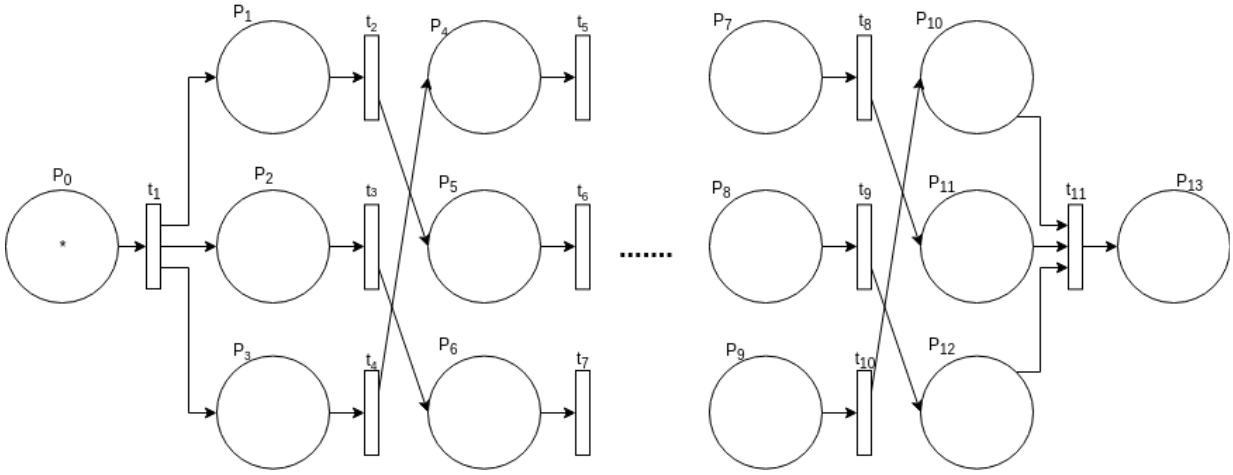

Рисунок 1 — Сеть Петри

Результаты работы программы на 1,2 …. N процессорах

```
$ mpirun 5.x -np 5
Parallel algorithm: 0.000081
Sequential algorithm: 0.000001

Process 0: MATRIX A
 677741.240000  611911.301000  516687.479000
 1039653.884000  807009.856000  115325.623000
 1224653.905000  2083069.270000  1106860.981000

Process 0: MATRIX B
 677741.240000  611911.301000  516687.479000
 1039653.884000  807009.856000  115325.623000
 1224653.905000  2083069.270000  1106860.981000

Process 0: MATRIX C
 1728272487966.835938  1984831784513.206055  992650674594.856323
 1684861218283.164551  1527672129744.188965  757894490975.074951
 4351191136015.867676  4736105091587.159180  2098135831397.142822
```

Вычислительные эксперименты

Размер матрицы	Количество процессов	Последовательный алгоритм	Параллельный алгоритм	Ускорение
10	1	0,000008	0,000008	1
	2		0,000031	0,2580645161
	4		0,000148	0,05405405405
	7		0,001607	0,004978220286
	10		0,013616	0,0005875440658
	12		0,024428	0,0003274930408
	16		0,12063	0,00006631849457
	20		0,146827	0,00005448589156
50	1	0,000634	0,000634	1
	2		0,000259	2,28057554
	4		0,000278	2,28057554
	7		0,000516	1,228682171
	10		0,03838	0,01651902032
	12		0,00064	0,990625
	16		0,11227	0,005647100739
	20		0,170091	0,003727416501
100	1	0,004808	0,004808	1
	2		0,002591	1,855654188
	4		0,001983	2,424609178
	7		0,00407	1,181326781
	10		0,399106	0,01204692488
	12		0,891653	0,00539223218
	16		1,059052	0,004539909277
	20		0,166178	0,02893283106
200	1	0,032926	0,032926	1
	2		0,015688	2,098801632
	4		0,012971	2,538431887
	7		0,012873	2,557756545
	10		0,210753	0,156230279
	12		0,430395	0,0765018181
	16		1,278618	0,02575124079
	20		1,485694	0,02216203337
500	1	0,549973	0,549973	1
	2		0,194072	2,833860629
	4		0,137159	4,009747811
	7		0,138165	3,980552238
	10		0,505623	1,087713573
	12		0,820163	0,6705654852
	16		1,523177	0,361069659
	20		1,496607	0,3674799062

1000	1	6,307629	6,307629	1
	2		2,091483	3,015864341
	4		1,337301	4,716686071
	7		1,148605	5,491556279
	10		1,670551	3,775777573
	12		2,533318	2,489868623
	16		2,926586	2,155285715
	20		5,627436	1,120870855

Построен график ускорения в зависимости от количества процессоров (см рис 2), где: x – количество процессов, y – ускорение. Легенда:

- зеленый - размерность матриц 10,
- желтый - размерность матриц 50,
- фиолетовый - размерность матриц 100,
- синий - размерность матриц 200,
- красный - размерность матриц 500,
- бордовый - размерность матриц 1000.

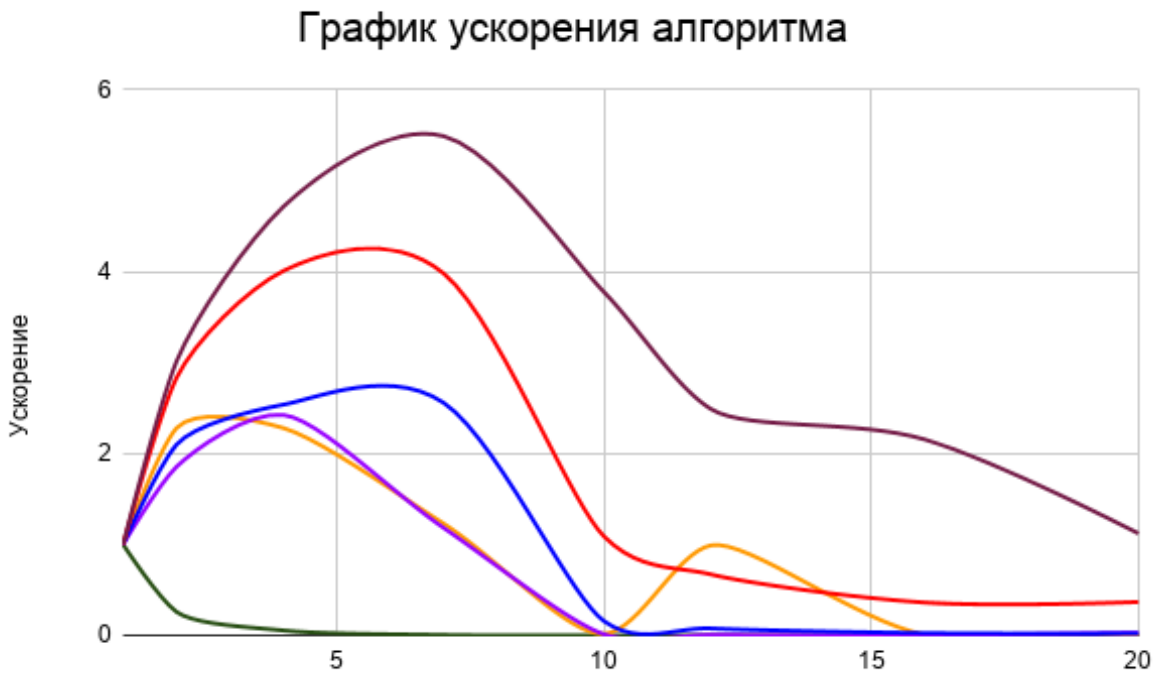

Рисунок 2 – График ускорения алгоритма

Вывод

В ходе лабораторной работы был реализован алгоритм параллельного перемножения матриц.

При размерности матрицы 10 параллельное выполнение не является рациональным. Для матриц большего размера, оптимальным количеством является 2-7 процессов. Если количество процессов больше 10, то скорость вычисления снижается из-за накладных расходов на коммуникацию и передачу данных между процессами.

Листинг программы

```
#include <mpi.h>
#include <iostream>

#include "matrix_functions.h"

int ProcNum;
int ProcRank;
int matrix_size = 3;
double *A;
double *B;
double *C;

double multiply_matrix(double *A, double *B,  double *C, int size)
//Parallel Ribbonn algo
{
    double *bufA, *bufB, *bufC;
    int new_size = size;

    MPI_Status Status;

    int proc_size = new_size/ProcNum; //process part size
    int proc_elem = proc_size*new_size; //process part element

    bufA = new double[proc_elem];
    bufB = new double[proc_elem];
    bufC = new double[proc_elem];

    for (int i = 0; i < proc_elem; i++)
    {
        bufC[i] = 0;
    }

    double start_time = MPI_Wtime(); //возвращает количество
секунд, представляя полное время по отношению к некоторому моменту
времени в прошлом.
```

```
    MPI_Scatter(A, proc_elem, MPI_DOUBLE, bufA, proc_elem,
MPI_DOUBLE, 0, MPI_COMM_WORLD); //рассылка
    MPI_Scatter(B, proc_elem, MPI_DOUBLE, bufB, proc_elem,
MPI_DOUBLE, 0, MPI_COMM_WORLD);

    int NextProc = ProcRank + 1;
    if ( ProcRank == ProcNum - 1 ) NextProc = 0;

    int PrevProc = ProcRank - 1;
    if ( ProcRank == 0 ) PrevProc = ProcNum - 1;

    int PrevNum = ProcRank;
    for (int p = 0; p < ProcNum; p++)
    {
        for (int i = 0; i < proc_size; i++)
        {
            for (int j = 0; j < size; j++)
            {
                double tmp = 0;
                for (int k = 0; k < proc_size; k++)
                    tmp += bufA[PrevNum * proc_size + i * size +
k] * bufB[k * size + j];
                bufC[i * size + j] += tmp;
            }
        }

        PrevNum -= 1;

        if (PrevNum < 0)
            PrevNum = ProcNum - 1;

        //Совмещенные прием и передача
        MPI_Sendrecv_replace(bufB, proc_elem, MPI_DOUBLE,
NextProc, 0, PrevProc, 0, MPI_COMM_WORLD, &Status);
    }

    MPI_Gather(bufC, proc_elem, MPI_DOUBLE, C, proc_elem,
MPI_DOUBLE, 0, MPI_COMM_WORLD); //сборка данных

    double end_time = MPI_Wtime();
    return end_time - start_time;
}

void InitProcess (double* &A,double* &B,double* &C ,int &Size) {
    MPI_Comm_size(MPI_COMM_WORLD, &ProcNum);
    MPI_Comm_rank(MPI_COMM_WORLD, &ProcRank);

    //Broadcasts a message from the process with rank "root" to
all other processes of the communicator
    MPI_Bcast(&Size, 1, MPI_INT, 0, MPI_COMM_WORLD);
```

```
    if (ProcRank == 0) {
        A = new double [Size*Size];
        B = new double [Size*Size];
        C = new double [Size*Size];
        RandInit (A, Size); RandInit (B, Size);
    }
}


int main(int argc, char **argv) {
    double beg, end, serial;

    MPI_Init (&argc, &argv);
    InitProcess (A, B, C, matrix_size);

    double parallel = multiply_matrix(A, B, C, matrix_size);

    if (ProcRank == 0)
    {
        printf("Parallel algorithm: %f\n", parallel);

        double sequential = matrix_s_multiply(A, B, C,
matrix_size); //Sequential matrix multiplication
        printf("Sequential algorithm: %f\n", sequential);

        printMatrixes(A, B, C, matrix_size);
    }
    MPI_Finalize();
    delete [] A;
    delete [] B;
    delete [] C;
}
```