МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

имени М.В. Ломоносова

Факультет вычислительной математики и кибернетики

Практикум по учебному курсу

"Суперкомпьютеры и параллельная обработка данных"

**Задание №2:**

### Разработка параллельной версии программы для перемножения матриц с использованием алгоритма Кэннона с использованием технологии MPI и сравнение результатов ее работы с результатами решения задачи на OMP.

Отчет

студента 320 группы

факультета ВМК МГУ

Михельсон Герман Владимирович

2021 год

**Постановка задачи**

Разработать параллельную версию программы для перемножения матриц с использованием алгоритма Кэннона с использованием технологии MPI, а затем исследовать масштабируемость полученной программы, построить графики зависимости времени её выполнения от числа используемых узлов и объёма входных данных, сравнить полученные результаты с результатами, полученными на OMP и сделать выводы.

Код программы

#include <stdio.h>

#include <stdlib.h>

#include <math.h>

#include <mpi.h>

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

{

    if (argc != 2)

    {

        printf("ERROR: \nPlease enter NODE\_NUMBERA: \n./run NODE\_NUMBERS\n");

        return -1;

    }

    long long int N = atoi(argv[1]);

    MPI\_Comm cannon\_comm;

    MPI\_Status status;

    int rank, size;

    int shift;

    int i, j, k;

    int dims[2];

    int periods[2];

    int left, right, up, down;

    double \*A, \*B, \*C;

    double \*buf, \*tmp;

    double start, end;

    unsigned int iseed = 0;

    int Nl;

    MPI\_Init(&argc, &argv);

    MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank);

    MPI\_Comm\_size(MPI\_COMM\_WORLD, &size);

    srand(iseed);

    dims[0] = 0;

    dims[1] = 0;

    periods[0] = 1;

    periods[1] = 1;

    MPI\_Dims\_create(size, 2, dims);

    if(dims[0] != dims[1])

    {

        if(rank == 0)

        {

            printf("The number of processors must be a square.\n");

        }

        MPI\_Finalize();

        return 0;

    }

    Nl = N / dims[0];

    A = (double\*)malloc(Nl \* Nl \* sizeof(double));

    B = (double\*)malloc(Nl \* Nl \* sizeof(double));

    buf = (double\*)malloc(Nl \*Nl \* sizeof(double));

    C = (double\*)calloc(Nl \* Nl, sizeof(double));

    for(i=0; i<Nl; i++)

    for(j=0; j<Nl; j++)

    {

        A[i\*Nl+j] = 5 - (int)( 10.0 \* rand() / ( RAND\_MAX + 1.0 ) );

        B[i\*Nl+j] = 5 - (int)( 10.0 \* rand() / ( RAND\_MAX + 1.0 ) );

        C[i\*Nl+j] = 0.0;

    }

    MPI\_Cart\_create(MPI\_COMM\_WORLD, 2, dims,periods, 1, &cannon\_comm);

    MPI\_Cart\_shift(cannon\_comm, 0, 1, &left, &right);

    MPI\_Cart\_shift(cannon\_comm, 1, 1, &up, &down);

    start=MPI\_Wtime();

    for(shift=0; shift < dims[0]; shift++)

    {

        for(i=0; i<Nl; i++)

        for(k=0; k<Nl; k++)

        for(j=0; j<Nl; j++)

        C[i\*Nl+j] += A[i\*Nl+k]\*B[k\*Nl+j];

        if(shift == dims[0]-1)

        {

            break;

        }

        MPI\_Sendrecv(A, Nl \* Nl, MPI\_DOUBLE, left, 1, buf, Nl \* Nl, MPI\_DOUBLE, right, 1, cannon\_comm, &status);

        tmp = buf;

        buf = A;

        A = tmp;

        MPI\_Sendrecv(B, Nl \* Nl, MPI\_DOUBLE, up, 2, buf, Nl \* Nl, MPI\_DOUBLE, down, 2, cannon\_comm, &status);

        tmp = buf;

        buf = B;

        B = tmp;

    }

    MPI\_Barrier(cannon\_comm);

    end = MPI\_Wtime();

    if(rank == 0)

    {

        printf("Time: %.4fs\n",end-start);

    }

    free(A);

    free(B);

    free(buf);

    free(C);

    MPI\_Finalize();

    return 0;

}

**Результаты замеров времени выполнения**

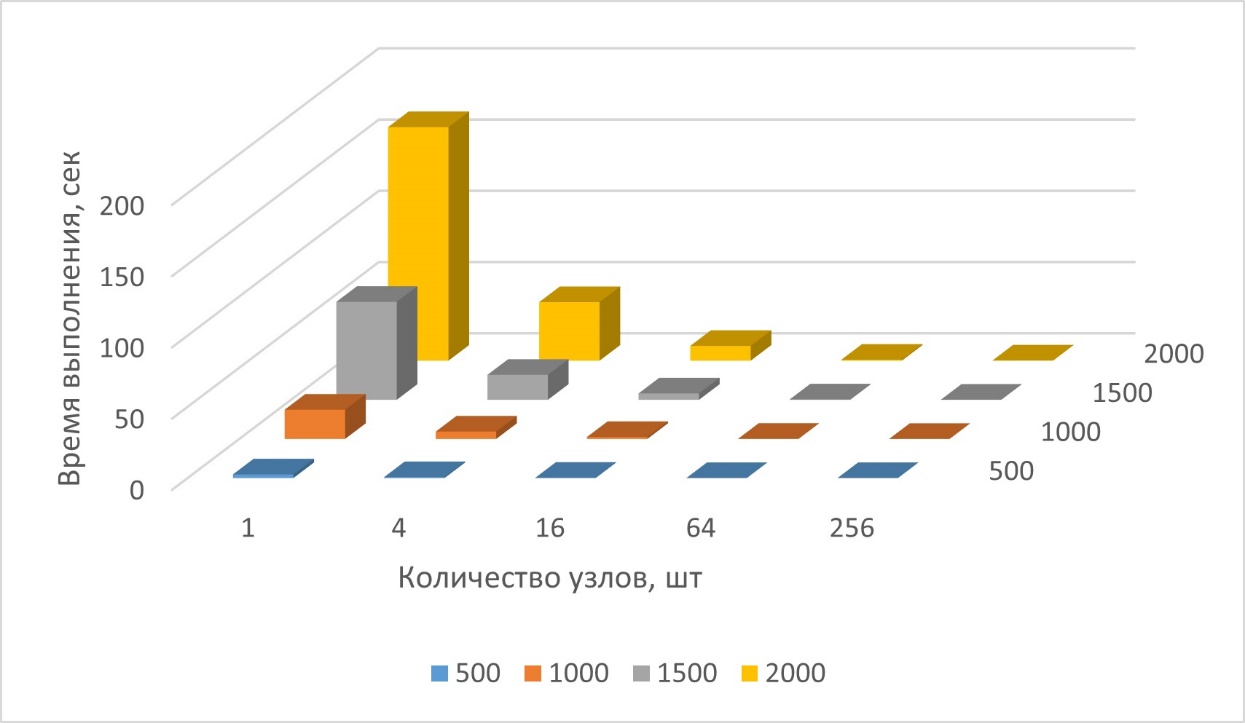
Работа задачи рассмотрена на суперкомпьютере Blue Gene с различным числом нитей (от 1 до 256) и различными размерами матрицы (от 500 до 2000).

Каждое измерение проводилось 3 раза. В таблице и на графиках записаны усредненные результаты времени выполнения.

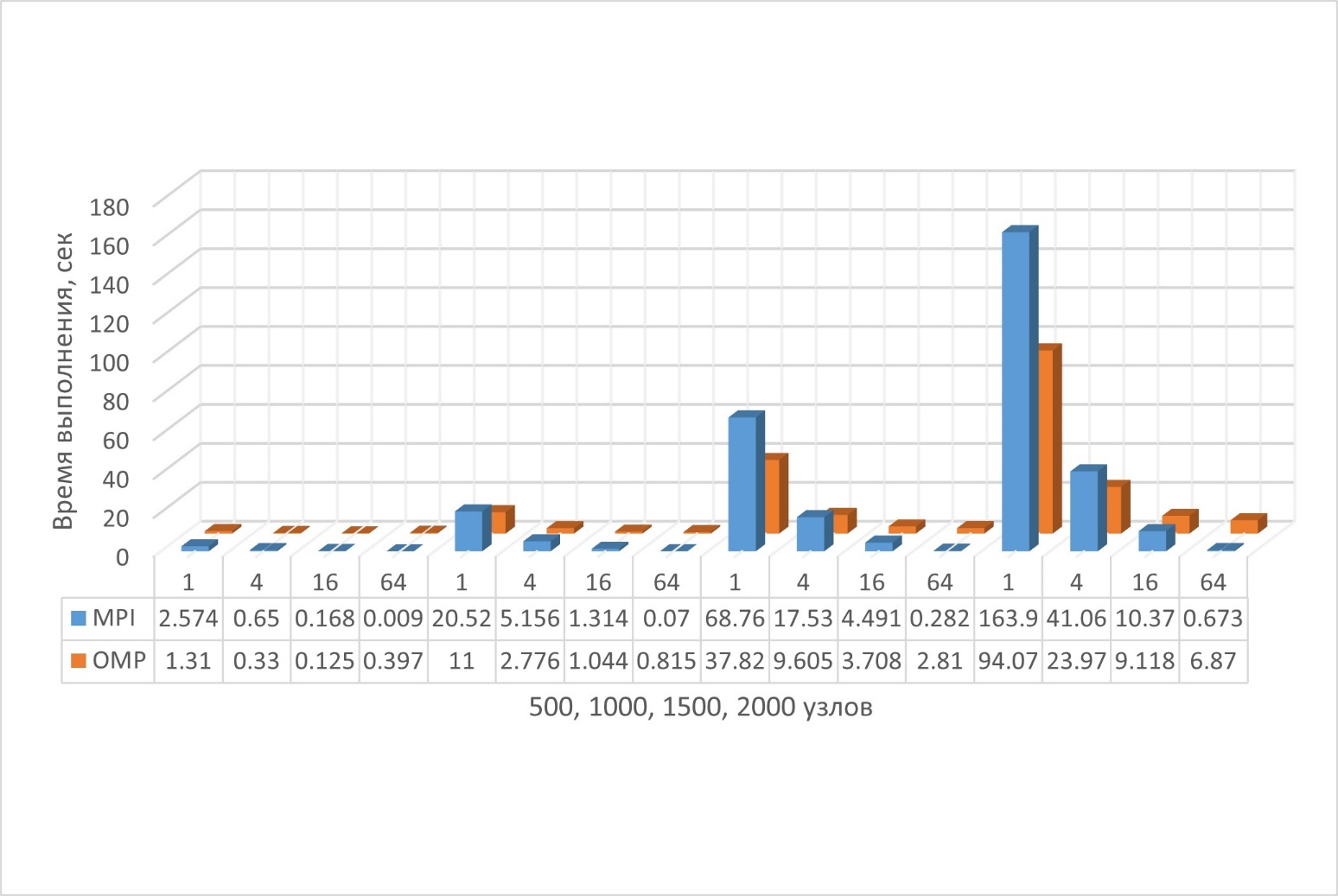
**Таблица с результатами**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Узлы/размер матрицы | 500 | 1000 | 1500 | 2000 |
| 1 | 2.5742s | 20.5184s | 68.7587s | 163.8796s |
| 4 | 0.6495s | 5.1558s | 17.5349s | 41.0632s |
| 16 | 0.1675s | 1.3138s | 4.4914s | 10.3717s |
| 64 | 0.0088s | 0.0699s | 0.2818s | 0.6727s |
| 256 | 0.0030s | 0.0178s | 0.0533s | 0.1411s |

**Графики: время выполнения программы в зависимости от размера матрицы и количества потоков**



**Графики: Сравнение результатов OMP и MPI**



**Вывод:**

Согласно результатам и диаграмме графика MPI, мы можем наблюдать, что время выполнения примерно одинаково для нитей ≥ 16. При количестве нитей = 64 время работы минимально. Следовательно, количество нитей = 64 является лучшим дизайном для дизайна версии MPI.

Согласно диаграмме сравнения работы OMP и MPI, можно cказать, что алгоритм MPI при количестве нитей ≤ 16 работает дольше OMP, но переходя данный порог MPI демонстрирует время гораздо лучшее, чем OMP. У обоих технологий в применении к данной задаче примерно одинаковая зависимость начиная с ≥ 16 нитей.