МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

имени М.В. Ломоносова

Факультет вычислительной математики и кибернетики

Практикум по учебному курсу

"Суперкомпьютеры и параллельная обработка данных"

**Задание №1:**

### Разработка параллельной версии программы для перемножения матриц с использованием алгоритма Кэннона с использованием технологии OpenMP.

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 OpenMP, а затем исследовать масштабируемость полученной программы, построить графики зависимости времени её выполнения от числа используемых нитей и объёма входных данных.

Код программы

#include <stdio.h>

#include <stdlib.h>

#include <omp.h>

#include <time.h>

#include <sys/time.h>

void get\_walltime(double \*wcTime);

int data\_set(double \*a, double \*b, double \*c, int NUM\_THREADS, long long int N);

int serial\_mult(double \*a, double \*b, double \*c, int NUM\_THREADS, long long int N);

int mp\_mult(double \*a, double \*b, double \*c, int NUM\_THREADS, long long int N);

int mp\_mult(double \*a, double \*b, double \*c, int NUM\_THREADS, long long int N)

{

    int i, j, k;

    omp\_set\_num\_threads(NUM\_THREADS);

    #pragma omp parallel shared(a, b, c) private(i, j, k)

    {

    #pragma omp for schedule(dynamic)

        for (i = 0; i < N; i++)

        {

            for (j = 0; j < N; j++)

            {

                for (k = 0; k < N; k++)

                {

                    c[i \* N + j] += a[i \* N + k] \* b[k \* N + j];

                }

            }

        }

    }

    return 0;

}

int serial\_mult(double \*a, double \*b, double \*c, int NUM\_THREADS, long long int N)

{

    int i, j, k;

    for (i = 0; i < N; i++)

    {

        for (j = 0; j < N; j++)

        {

            for (k = 0; k < N; k++)

            {

                c[i \* N + j] += a[i \* N + k] \* b[k \* N + j];

            }

        }

    }

    return 0;

}

int data\_set(double \*a, double \*b, double \*c, int NUM\_THREADS, long long int N)

{

    int i, j;

    for (i = 0; i < N; i++)

    {

        for (j = 0; j < N; j++)

        {

            a[i \* N + j] = 2.0;

            b[i \* N + j] = 3.0;

            c[i \* N + j] = 0.0;

        }

    }

    return 0;

}

void get\_walltime(double \*wcTime)

{

    struct timeval tp;

    gettimeofday(&tp, NULL);

    \*wcTime = (double)(tp.tv\_sec + tp.tv\_usec / 1000000.0);

}

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

{

    if (argv != 3)

    {

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

        return -1;

    }

    int NUM\_THREADS = atoi(argc[1]);

    long long int N = atoi(argc[2]);

    double \*a = (double \*)malloc(N \* N \* sizeof(double));

    double \*b = (double \*)malloc(N \* N \* sizeof(double));

    double \*c = (double \*)malloc(N \* N \* sizeof(double));

    double d\_s, d\_e;

    double d\_s\_mp, d\_e\_mp;

    double d\_s\_mpi, d\_e\_mpi;

    int i, j;

    data\_set(a, b, c, NUM\_THREADS, N);

    get\_walltime(&d\_s);

    serial\_mult(a, b, c, NUM\_THREADS, N);

    get\_walltime(&d\_e);

    data\_set(a, b, c, NUM\_THREADS, N);

    printf("openMP:\n");

    get\_walltime(&d\_s\_mp);

    mp\_mult(a, b, c, NUM\_THREADS, N);

    get\_walltime(&d\_e\_mp);

    printf("-----------------\nN=%lld, NUM\_THREADS=%d\n", N, NUM\_THREADS);

    printf("serial time: %lf\n", d\_e - d\_s);

    printf("openMP time: %lf\n-----------------\n\n\n", d\_e\_mp - d\_s\_mp);

    free(a);

    free(b);

    free(c);

    return 0;

}

**Результаты замеров времени выполнения**

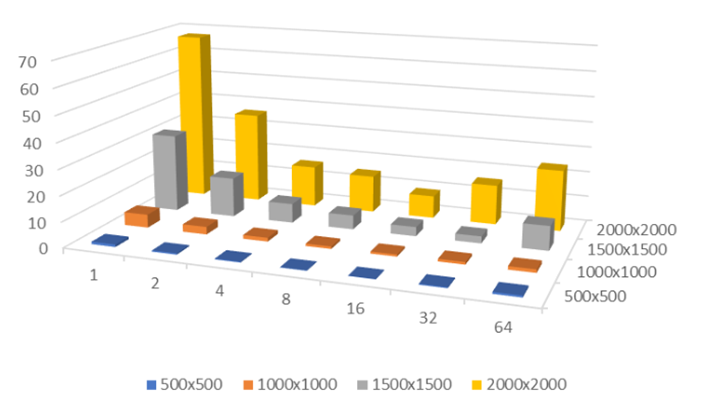
Работа задачи рассмотрена на суперкомпьютере Polus с различным числом нитей (от 1 до 64) и различными размерами матрицы (от 500 до 2000).

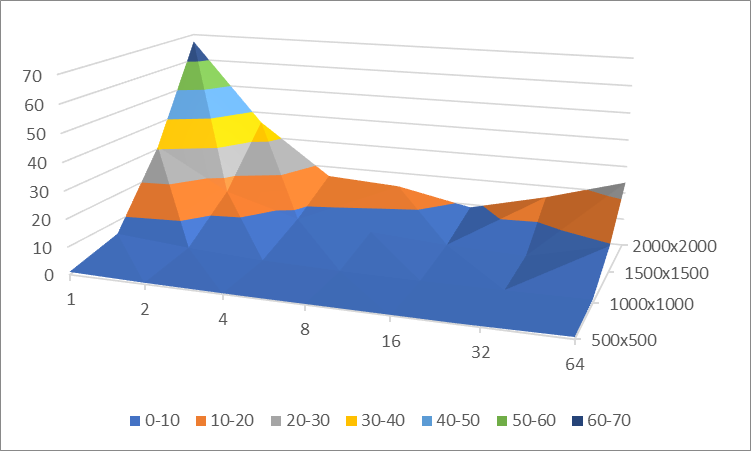
Каждое измерение проводилось 3 раза. В таблице и на графиках записаны усредненные результаты времени выполнения.

**Таблица с результатами**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Нити/размер матрицы | 500 | 1000 | 1500 | 2000 |
| 1 | 1.309660 | 11.000788 | 37.817510 | 94.073729 |
| 2 | 0.656973 | 5.248969 | 18.980589 | 47.191494 |
| 4 | 0.330377 | 2.775974 | 9.604778 | 23.966895 |
| 8 | 0.173858 | 1.455101 | 5.159510 | 13.163472 |
| 16 | 0.125082 | 1.044231 | 3.708013 | 9.118330 |
| 32 | 0.104678 | 0.838682 | 3.297730 | 7.728176 |
| 64 | 0.397161 | 0.814691 | 2.810341 | 6.870052 |

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





**Вывод:**

Согласно результатам и диаграмме графика OpenMP, мы можем знать, что время выполнения увеличивается, когда размерность матрицы увеличивается. Затем время выполнения уменьшается, когда нити увеличиваются.