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 задачи Sor2D с помощью технологии MPI и сравнение результатов ее работы с результатами работы OMP

Отчет

студентки 320 группы

факультета ВМК МГУ

Кирсановой Софьи Игоревны

2021 год

## Постановка задачи

Разработать параллельную версию программы для задачи Sor2D с использованием технологии MPI, исследовать масштабируемость полученной программы в зависимости от числа нитей и размерности задачи, построить графики зависимости времени её выполнения от числа используемых ядер, сравнить результаты работы программы с реализацией задачи Sor2D с использованием технологии OMP.

Необходимо оценить время выполнения программы в зависимости от количества нитей и размерности задачи.

Метод релаксации (Successive over-relaxation, SOR) — итерационный метод решения систем линейных алгебраических уравнений. Под размерностью задачи понимается количество линейных уравнений в системе.

## Код параллельной программы

```
#include <math.h>
#include <stdlib.h>
#include <stdio.h>
#include <mpi.h>

#define  Max(a, b) ((a) > (b) ? (a) : (b))
#define  N 1000 // переменный параметр

double (*A)[N];
double maxeps = 0.1e-7, eps;
int itmax = 100, i, j, k, ll, shift;

MPI_Request req[4];
int myrank, ranksize;
int startrow, lastrow, nrow;
MPI_Status status[4];

void relax();
void init();
void verify();

int main(int an, char **as)
{
    int it;
    MPI_Init(&an, &as);
    MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
```

```
    MPI_Comm_size(MPI_COMM_WORLD, &ranksize);
    MPI_Barrier(MPI_COMM_WORLD);
    startrow = (myrank * N) / ranksize;
    lastrow = ((myrank + 1) * N) / ranksize - 1;
    nrow = lastrow - startrow + 1;
    double time = MPI_Wtime();
    init();
    for (it = 1; it <= itmax; it++)
    {
        eps = 0.;
        relax();
        if (myrank == 0)
        {
            printf("it = %4i   eps = %f\n", it, eps);
        }
        if (eps < maxeps) break;
    }
    verify();
    if (myrank == 0)
    {
        printf("time = %f\n", MPI_Wtime() - time);
    }
    MPI_Finalize();

    return 0;
}


void init()
{
    A = malloc(nrow * N * sizeof(double));
    for(i = 0; i <= nrow - 1; i++)
    for(j = 0; j <= N - 1; j++)
    {
        if (i == 0 || i == N - 1 || j == 0 || j == N - 1)
        A[i][j] = 0.;
        else A[i][j] = (1. + startrow + i + j ) ;
    }
}


void relax()
{
    if (myrank != 0)
        MPI_Irecv(&A[0][0], N, MPI_DOUBLE, myrank - 1, 1, MPI_COMM_WORLD, &req[0]);
    if (myrank != ranksize - 1)
        MPI_Isend(&A[nrow - 2][0], N, MPI_DOUBLE, myrank + 1, 1, MPI_COMM_WORLD, &req[2]);
    if (myrank != ranksize - 1)
```

```
        MPI_Irecv(&A[nrow - 1][0], N, MPI_DOUBLE, myrank + 1, 2, MPI_COMM_WORLD, &req[3]);
    if (myrank != 0)
        MPI_Isend(&A[1][0], N, MPI_DOUBLE, myrank - 1, 2, MPI_COMM_WORLD, &req[1]);
    ll = 4;
    shift = 0;
    if (myrank == 0) {ll = 2; shift = 2;}
    if (myrank == ranksize - 1) {ll = 2;}
    if (ranksize > 1)
        MPI_Waitall(ll, &req[shift], &status[0]);

    for(i = 1; i <= nrow - 2; i++)
    {
        if (((i == 1) && (myrank == 0)) || ((i == nrow - 2) && (myrank == ranksize - 1)))
            continue;
        for (j = 1; j <= N - 2; j++)
        {
            double e;
            e = A[i][j];
            A[i][j] = (A[i - 1][j] + A[i + 1][j] + A[i][j - 1] + A[i][j + 1]) / 4.;
            eps = Max(eps, fabs(e - A[i][j]));
        }
    }
    if (myrank == 0)
        for (i = 1; i < ranksize; i++)
        {
            double tmp;
            MPI_Recv(&tmp, 1, MPI_DOUBLE, MPI_ANY_SOURCE, 3, MPI_COMM_WORLD, &status[1]);
            eps = Max(eps, tmp);
        }
    if (myrank != 0)
        MPI_Ssend(&eps, 1, MPI_DOUBLE, 0, 3, MPI_COMM_WORLD);
    if (myrank != 0)
        MPI_Recv(&eps, 1, MPI_DOUBLE, 0, 4, MPI_COMM_WORLD, &status[1]);
    if (myrank == 0)
        for (i = 1; i < ranksize; i++)
            MPI_Ssend(&eps, 1, MPI_DOUBLE, i, 4, MPI_COMM_WORLD);
}


void verify()
{
    double s;
    s = 0.;
    for (i = 0; i <= nrow - 1; i++)
    for (j = 0; j <= N - 1; j++)
        s = s + A[i][j] * (i + 1 + startrow) * (j + 1) / (N * N);
    if (myrank == 0 && ranksize > 1)
        for (i = 1; i < ranksize; i++)
```

```
        {
            double tmp;
            MPI_Recv(&tmp, 1, MPI_DOUBLE, i, 5, MPI_COMM_WORLD, &status[1]);
            s += tmp;
        }
    if (myrank != 0)
        MPI_Ssend(&s, 1, MPI_DOUBLE, 0, 5, MPI_COMM_WORLD);
    if (myrank == 0)
    {
        printf("  S = %f\n",s);
    }
}
        MPI_Ssend(&s, 1, MPI_DOUBLE, 0, 5, MPI_COMM_WORLD);
    m_printf("  S = %f\n",s);
}
```

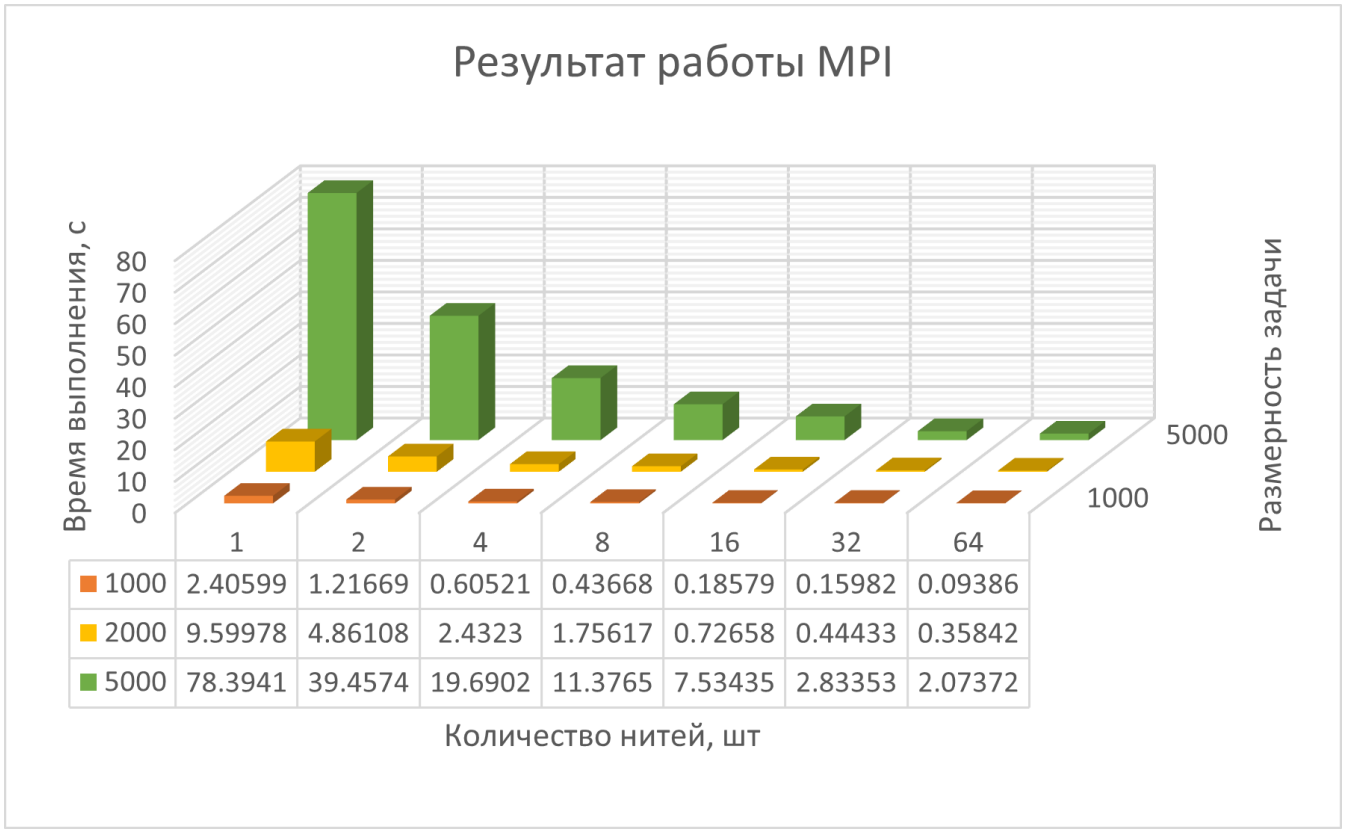
## Результаты замеров времени выполнения

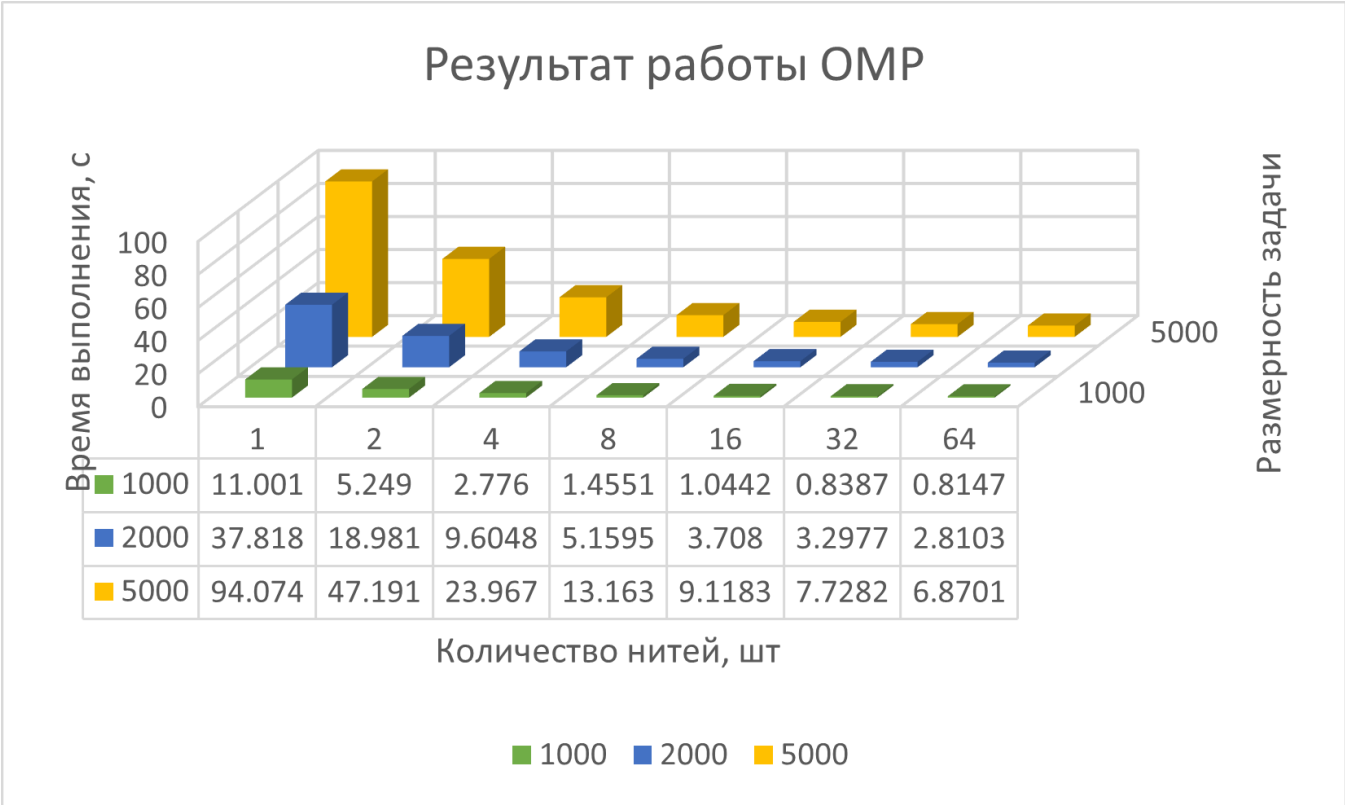
Работа программы была рассмотрена на суперкомпьютере Polus с различным числом нитей (1, 2, 4, 8, 16, 32 и 64) и с различной размерностью задачи (1000, 2000 и 5000).

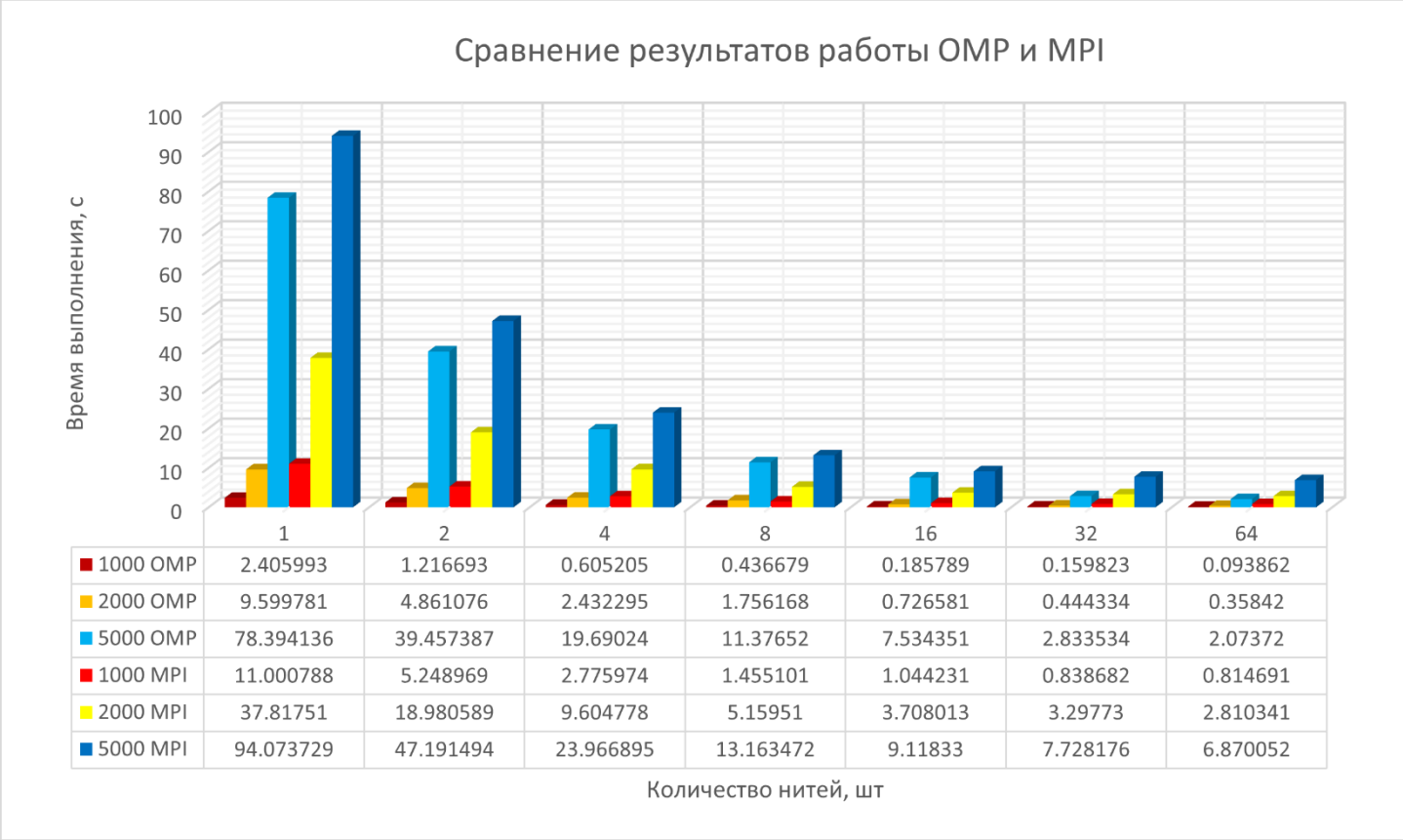
## Таблица с результатами MPI

| *Нити / размерность* | 1000 | 2000 | 5000 |
|---|---|---|---|
| *1* | 2.405993 | 9.599781 | 78.394136 |
| *2* | 1.216693 | 4.861076 | 39.457387 |
| *4* | 0.605205 | 2.432295 | 19.690240 |
| *8* | 0.436679 | 1.756168 | 11.376520 |
| *16* | 0.185789 | 0.726581 | 7.534351 |
| *32* | 0.159823 | 0.444334 | 2.833534 |
| *64* | 0.093862 | 0.358420 | 2.073720 |

# Графики зависимости времени выполнения программы от количества потоков и размерности задачи






| | 1 | 2 | 4 | 8 | 16 | 32 | 64 |
|---|---|---|---|---|---|---|---|
| 1000 OMP | 2.405993 | 1.216693 | 0.605205 | 0.436679 | 0.185789 | 0.159823 | 0.093862 |
| 2000 OMP | 9.599781 | 4.861076 | 2.432295 | 1.756168 | 0.726581 | 0.444334 | 0.35842 |
| 5000 OMP | 78.394136 | 39.457387 | 19.69024 | 11.37652 | 7.534351 | 2.833534 | 2.07372 |
| 1000 MPI | 11.000788 | 5.248969 | 2.775974 | 1.455101 | 1.044231 | 0.838682 | 0.814691 |
| 2000 MPI | 37.81751 | 18.980589 | 9.604778 | 5.15951 | 3.708013 | 3.29773 | 2.810341 |
| 5000 MPI | 94.073729 | 47.191494 | 23.966895 | 13.163472 | 9.11833 | 7.728176 | 6.870052 |

# Вывод

Согласно результатам и диаграмме графика MPI, можно наблюдать, что время выполнения прямо пропорционально размерности задачи и обратно пропорционально количеству нитей. Таким образом, с увеличением количества нитей время выполнения резко снижается при любой рассмотренной размерности.

Согласно диаграмме сравнения работы OMP и MPI, можно с уверенностью сказать, что при одинаковом количестве нитей алгоритм, реализованный с помощью технологии MPI работает быстрее. У обоих технологий в применении к данной задаче одинаковая зависимость от количества нитей.