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 ПРАКТИЧЕСКОЙ РАБОТЫ

Параллельная реализация метода Якоби в трехмерной области

студента 2 курса, группы 23201

Смирнова Гордея Андреевича

Направление 09.03.01 – «Информатика и вычислительная техника»

Преподаватель:
А.С. Матвеев

Новосибирск 2025

СОДЕРЖАНИЕ

ЦЕЛЬ

Практическое освоение методов распараллеливания численных алгоритмов на регулярных сетках на примере реализации метода Якоби в трехмерной области.

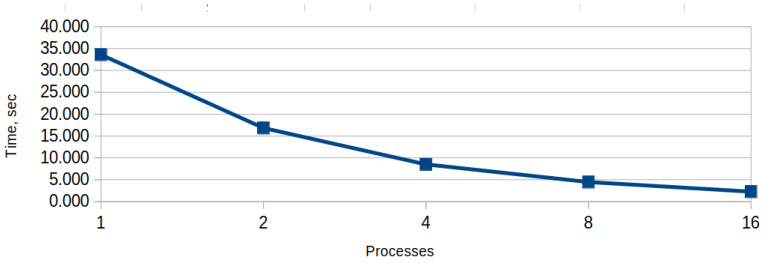
ЗАДАНИЕ

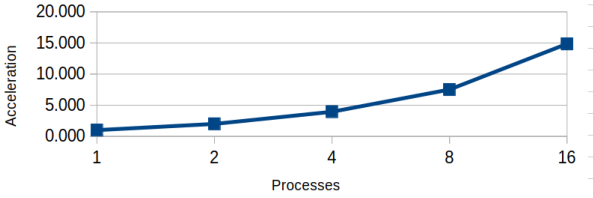
1. Написать параллельную программу на языке C/C++ с использованием MPI, реализующую решение уравнения (1) методом Якоби в трехмерной области в случае одномерной декомпозиции области. Уделить внимание тому, чтобы обмены граничными значениями подобластей выполнялись на фоне счета.
2. Измерить время работы программы при использовании различного числа процессорных ядер: 1, 2, 4, 8, 16. Размеры сетки и порог сходимости подобрать таким образом, чтобы решение задачи на одном ядре занимало не менее 30 секунд. Построить графики зависимости времени работы программы, ускорения и эффективности распараллеливания от числа используемых ядер.
3. Выполнить профилирование программы с помощью MPE при использовании 16-и ядер. По профилю убедиться, что коммуникации происходят на фоне счета

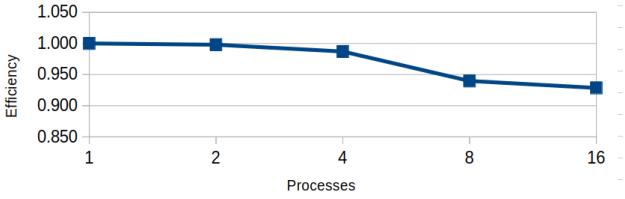
ОПИСАНИЕ РАБОТЫ

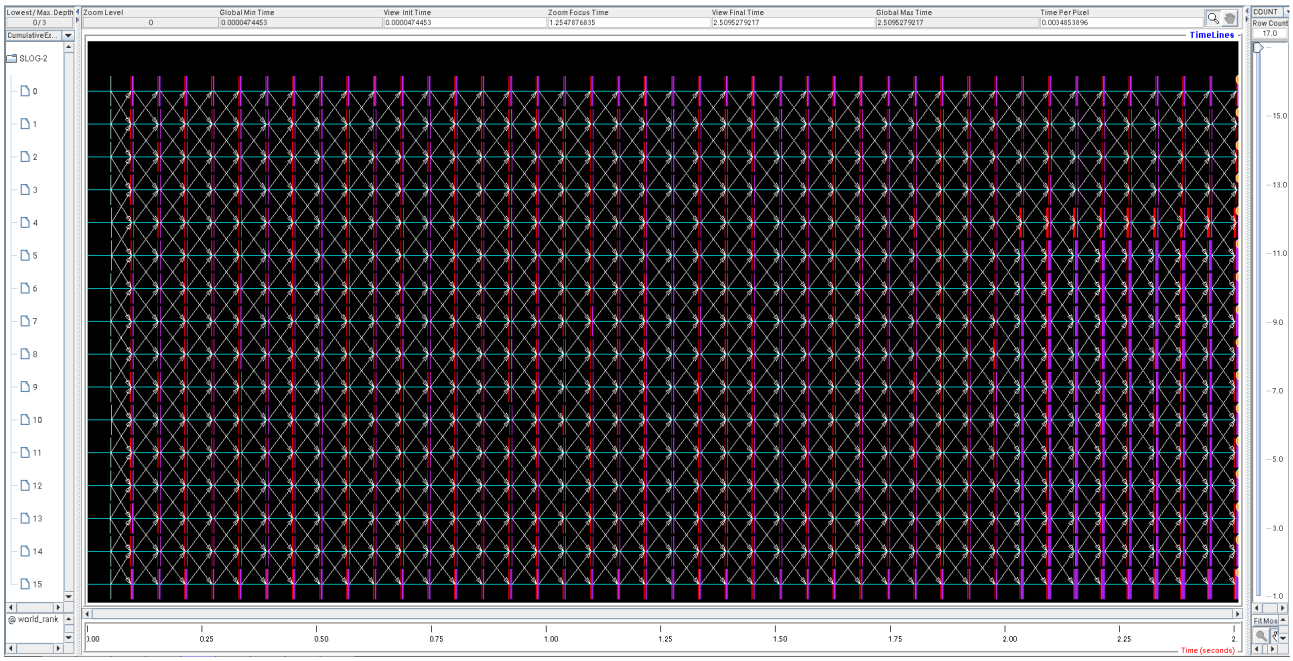
На языке C++ была написана MPI-программа для решения ДУ в трехмерной области методом Якоби. Было уделено внимание использование неблокирующих MPI-операций для обмена крайними слоями между соседними процессами. Ниже приведены результаты замеров производительности на 1, 2, 4, 8, 16 процессах, а также результаты профилирования на 16 процессах.

Processes	Time, sec	Acceleration	Efficiency
1	33.589	1.000	1.000
2	16.830	1.996	0.998
4	8.508	3.948	0.987
8	4.468	7.518	0.940
16	2.261	14.856	0.929









По результатам профилирования установлено, что весь обмен граничными слоями сетки между процессами выполняется на фоне вычислений основной области сетки.

ЗАКЛЮЧЕНИЕ

В ходе выполнения практической работы были изучены функции библиотеки MPI, позволяющие производить межпроцессный обмен данными на фоне вычислений без блокирования выполнения программы.

ПРИЛОЖЕНИЯ

Приложение 1: *JacobiSolver3D.h*

```
#pragma once

#include <mpi.h>
#include <vector>

typedef std::vector<float> fvector;

class JacobiSolver3D {
 public:
    JacobiSolver3D(int rank, int size);
    float solve();

 private:
    void distributeLayers();
    void initBuffers();

    float calcCore();
    float calcBorders();

    void isendrecvBorders();
    void waitBorders();

    float phi(int i, int j, int k) const;
    float rho(int i, int j, int k) const;

    int getBorderIdx(int j, int k) const;
    float& curr(int i, int j, int k);
    float& prev(int i, int j, int k);

    const int procRank, commSize;
    const float x_0, y_0, z_0;
    const float D_x, D_y, D_z;
    const float parA, eps;
    const int N_x, N_y, N_z;
    const float h_x, h_y, h_z;
    const float h_x2, h_y2, h_z2;
    const float coef;

    int locN_x, locOffset;
    fvector prevBuffer, currBuffer;
    fvector topBorder, bottomBorder;

    MPI_Request reqSendTop, reqSendBottom;
    MPI_Request reqRecvTop, reqRecvBottom;
};
```

Приложение 2: *JacobiSolver.cpp*

```
#include "JacobiSolver3D.h"

#include <mpi.h>
#include <iostream>
#include <vector>
#include <algorithm>

JacobiSolver3D::JacobiSolver3D(int rank, int size)
: procRank(rank), commSize(size),
x_0(-1), y_0(-1), z_0(-1),
D_x(2), D_y(2), D_z(2),
parA(1e5), eps(1e-6),
N_x(400), N_y(400), N_z(400),
h_x(D_x / (N_x - 1)), h_y(D_y / (N_y - 1)), h_z(D_z / (N_z - 1)),
h_x2(h_x * h_x), h_y2(h_y * h_y), h_z2(h_z * h_z),
coef(1 / (2 / h_x2 + 2 / h_y2 + 2 / h_z2 + parA)) {
    distributeLayers();
    initBuffers();
}

float JacobiSolver3D::solve() {
    float globDiff = eps;
    while (globDiff ⩾ eps) {
        std::swap(prevBuffer, currBuffer);

        isendrecvBorders();
        float locCoreDiff = calcCore();
        waitBorders();
        float locBorderDiff = calcBorders();
        float locDiff = std::max(locCoreDiff, locBorderDiff);

        MPI_Allreduce(&locDiff, &globDiff, 1, MPI_FLOAT,
            MPI_MAX, MPI_COMM_WORLD);
    }
    return globDiff;
}

void JacobiSolver3D::distributeLayers() {
    std::vector<int> allLocN_x(commSize), allOffsets(commSize);
    int shift = 0;
    for (int i = 0; i < commSize; i++) {
        allLocN_x[i] = N_x / commSize;
        allLocN_x[i] += i < N_x % commSize;
        allOffsets[i] = shift;
        shift += allLocN_x[i];
    }
    locN_x = allLocN_x[procRank];
    locOffset = allOffsets[procRank];
}

void JacobiSolver3D::initBuffers() {
    prevBuffer = fvector(locN_x * N_y * N_z);
```

```
    currBuffer = fvector(locN_x * N_y * N_z);
    for (int i = 0; i < locN_x; i++) {
        for (int j = 0; j < N_y; j++) {
            for (int k = 0; k < N_z; k++) {
                int iOff = i + locOffset;
                bool isBorder = iOff == 0 || iOff == N_x - 1 ||
                    j == 0 || j == N_y - 1 ||
                    k == 0 || k == N_z - 1;
                if (isBorder) {
                    curr(i, j, k) = phi(iOff, j, k);
                    prev(i, j, k) = phi(iOff, j, k);
                }
            }
        }
    }
    topBorder = fvector(N_y * N_z);
    bottomBorder = fvector(N_y * N_z);
}

float JacobiSolver3D::calcCore() {
    float maxDiff = 0;
    for (int i = 1; i < locN_x - 1; i++) {
        for (int j = 1; j < N_y - 1; j++) {
            for (int k = 1; k < N_z - 1; k++) {
                float f_i = (prev(i + 1, j, k) + prev(i - 1, j, k)) / h_x2;
                float f_j = (prev(i, j + 1, k) + prev(i, j - 1, k)) / h_y2;
                float f_k = (prev(i, j, k + 1) + prev(i, j, k - 1)) / h_z2;

                curr(i, j, k) = coef *
                    (f_i + f_j + f_k - rho(i + locOffset, j, k));
                maxDiff = std::max(maxDiff,
                    std::abs(curr(i, j, k) - prev(i, j, k)));
            }
        }
    }
    return maxDiff;
}

float JacobiSolver3D::calcBorders() {
    float maxDiff = 0;
    for (int j = 1; j < N_y - 1; j++) {
        for (int k = 1; k < N_z - 1; k++) {
            if (procRank != 0) {
                int i = 0;
                float f_i = (prev(i + 1, j, k) +
                    topBorder[getBorderIdx(j, k)]) / h_x2;
                float f_j = (prev(i, j + 1, k) + prev(i, j - 1, k)) / h_y2;
                float f_k = (prev(i, j, k + 1) + prev(i, j, k - 1)) / h_z2;

                curr(i, j, k) = coef *
                    (f_i + f_j + f_k - rho(i + locOffset, j, k));
                maxDiff = std::max(maxDiff,
                    std::abs(curr(i, j, k) - prev(i, j, k)));
            }
```

```
            if (procRank != commSize - 1) {
                int i = locN_x - 1;
                float f_i = (prev(i - 1, j, k) +
                    bottomBorder[getBorderIdx(j, k)]) / h_x2;
                float f_j = (prev(i, j + 1, k) + prev(i, j - 1, k)) / h_y2;
                float f_k = (prev(i, j, k + 1) + prev(i, j, k - 1)) / h_z2;

                curr(i, j, k) = coef *
                    (f_i + f_j + f_k - rho(i + locOffset, j, k));
                maxDiff = std::max(maxDiff,
                    std::abs(curr(i, j, k) - prev(i, j, k)));
            }
        }
    }
    return maxDiff;
}

void JacobiSolver3D::isendrecvBorders() {
    if (procRank != 0) {
        float* prevTopBorderPtr = prevBuffer.data();
        MPI_Isend(prevTopBorderPtr, N_y * N_z, MPI_FLOAT,
            procRank - 1, procRank, MPI_COMM_WORLD, &reqSendTop);
        MPI_Irecv(topBorder.data(), N_y * N_z, MPI_FLOAT,
            procRank - 1, procRank - 1, MPI_COMM_WORLD, &reqRecvTop);
    }
    if (procRank != commSize - 1) {
        float* prevBottomBorderPtr =
            prevBuffer.data() + (locN_x - 1) * N_y * N_z;
        MPI_Isend(prevBottomBorderPtr, N_y * N_z, MPI_FLOAT,
            procRank + 1, procRank, MPI_COMM_WORLD, &reqSendBottom);
        MPI_Irecv(bottomBorder.data(), N_y * N_z, MPI_FLOAT,
            procRank + 1, procRank + 1, MPI_COMM_WORLD, &reqRecvBottom);
    }
}

void JacobiSolver3D::waitBorders() {
    if (procRank != 0) {
        MPI_Wait(&reqSendTop, MPI_STATUS_IGNORE);
        MPI_Wait(&reqRecvTop, MPI_STATUS_IGNORE);
    }
    if (procRank != commSize - 1) {
        MPI_Wait(&reqSendBottom, MPI_STATUS_IGNORE);
        MPI_Wait(&reqRecvBottom, MPI_STATUS_IGNORE);
    }
}

float JacobiSolver3D::phi(int i, int j, int k) const {
    float x = x_0 + i * h_x;
    float y = y_0 + j * h_y;
    float z = z_0 + k * h_z;
    return x * x + y * y + z * z;
}
```

```
float JacobiSolver3D::rho(int i, int j, int k) const {
    return 6 - parA * phi(i, j, k);
}

int JacobiSolver3D::getBorderIdx(int j, int k) const { return j * N_z + k; }

float& JacobiSolver3D::curr(int i, int j, int k) {
    return currBuffer[i * N_y * N_z + j * N_z + k];
}

float& JacobiSolver3D::prev(int i, int j, int k) {
    return prevBuffer[i * N_y * N_z + j * N_z + k];
}
```

Приложение 3: *main.cpp*

```
#include <mpi.h>
#include <cstdlib>
#include <cfloat>
#include <iostream>
#include <algorithm>
#include "JacobiSolver3D.h"

#define LOOPS 8

int main(int argc, char **argv) {
    MPI_Init(&argc, &argv);
    int rank, size;
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);

    double duration = DBL_MAX;
    float diff = -1;
    for (int i = 0; i < LOOPS; i++) {
        JacobiSolver3D solver(rank, size);
        const double start = MPI_Wtime();
        diff = solver.solve();
        const double end = MPI_Wtime();
        duration = std::min(duration, end - start);
    }
    if (rank == 0) {
        std::cout << "Time taken: " << duration << " sec" << std::endl;
        std::cout << "Max diff: " << diff << std::endl;
        std::cout << "================================" <<
                     "================================" << std::endl;
    }
    MPI_Finalize();
    return EXIT_SUCCESS;
}
```