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 решения системы линейных алгебраических уравнений с помощью MPI

студента 2 курса, группы 23201

Смирнова Гордея Андреевича

Направление 09.03.01 – «Информатика и вычислительная техника»

Преподаватель:
А.С. Матвеев

Новосибирск 2025

СОДЕРЖАНИЕ

ЦЕЛЬ

Научиться писать базовые параллельные программы с помощью MPI.

ЗАДАНИЕ

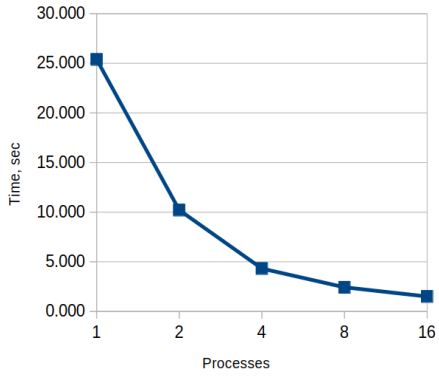
1. Написать программу на языке C или C++, которая реализует итерационный алгоритм решения системы линейных алгебраических уравнений вида Ax=b с помощью метода простой итерации. Здесь A – матрица размером N×N, x и b – векторы длины N. Тип элементов – float.
2. Программу распараллелить с помощью MPI с разрезанием матрицы A по строкам на близкие по размеру, возможно не одинаковые, части. Соседние строки матрицы должны располагаться в одном или в соседних MPI-процессах. Реализовать один вариант программы: векторы x и b дублируются в каждом MPI-процессе.
   Уделить внимание тому, чтобы при запуске программы на различном числе MPI-процессов решалась одна и та же задача (исходные данные заполнялись одинаковым образом).
3. Замерить время работы двух вариантов программы при использовании различного числа процессорных ядер: 1,2, 4, 8, 16. Построить графики зависимости времени работы программы, ускорения и эффективности распараллеливания от числа используемых ядер. Исходные данные, параметры N и ε подобрать таким образом, чтобы решение задачи на одном ядре занимало не менее 30 секунд.
4. Выполнить профилирование двух вариантов программы с помощью MPE при использовании 16-и ядер.
5. На основании полученных результатов сделать вывод о целесообразности использования одного или второго варианта программы.

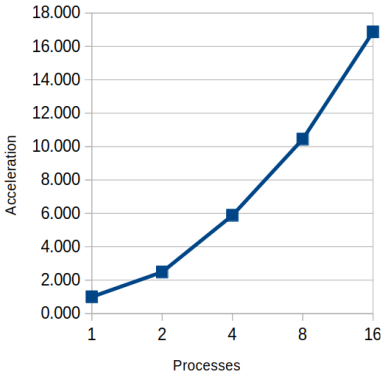
ОПИСАНИЕ РАБОТЫ

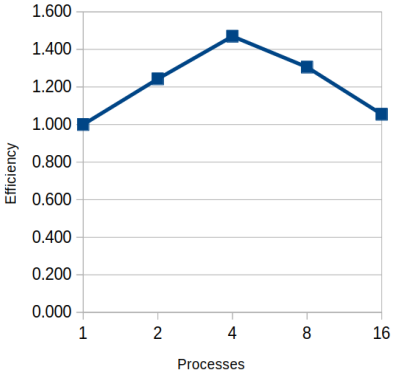
На языке C++ была написана MPI-программа для решения СЛАУ итерационным методом минимальных невязок. Были произведены замеры производительности программы при запуске на 1, 2, 4, 8, 16 MPI-процессах, а также MPE-профилирование при запуске на 16 процессах. Результаты замеров и профилирования приведены ниже. Исходный код программы приведён в приложениях.

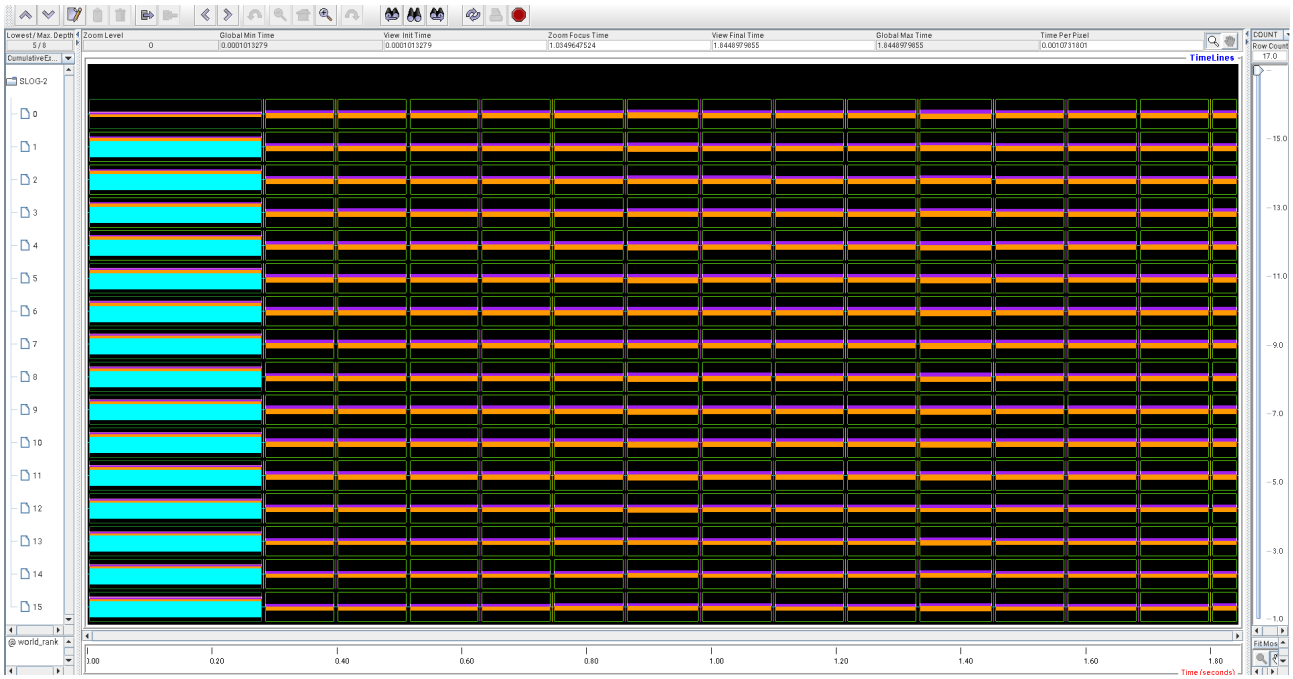
Processes	Time, sec	Acceleration	Efficiency
1	25.380	1.000	1.000
2	10.210	2.486	1.243
4	4.317	5.879	1.470
8	2.430	10.443	1.305
16	1.504	16.874	1.055

select	ncpus	mpiprocs	loops
1	8	1	16
2	8	1	16
2	8	2	16
2	8	4	64
2	8	8	64









По результатам профилирования видно, что коммуникации занимают заметно меньшую часть времени в сравнении с локальными вычислениями, что говорит об эффективном распараллеливании. Основные вызовы (оранжевые и фиолетовые) – MPI_Allreduce и MPI_Allgather. Голубая часть (MPI_Bcast) занимает значительную часть из-за чтения и обработки больших бинарных файлов со входными данными 0-ым процессом.

ЗАКЛЮЧЕНИЕ

В ходе выполнения практической работы были изучены основные функции библиотеки MPI и получены базовые знания о параллельных вычислениях, а также о профилировании параллельных программ.

ПРИЛОЖЕНИЯ

Приложение 1: *BinIO.h*

```
#pragma once

#include <string>
#include <vector>

class BinIO {
 public:
    typedef std::vector<float> fvector;

    static fvector readVecFromBin(const std::string& filename);
    static void writeVecToBin(const fvector& vec,
                              const std::string& filename);
};
```

Приложение 2: *BinIO.cpp*

```
#include "BinIO.h"

#include <fstream>
#include <string>

BinIO::fvector BinIO::readVecFromBin(const std::string& filename) {
    std::ifstream file(filename.c_str(), std::ios::binary | std::ios::ate);
    if (!file.is_open()) {
        throw std::runtime_error("Could not open file: " + filename);
    }

    size_t fileSize = file.tellg();
    if (fileSize % sizeof(float) =/= 0) {
        throw std::runtime_error(
            "File size is not a multiple of sizeof(float): " + filename);
    }

    fvector floats(fileSize / sizeof(float));
    file.seekg(0, std::ios::beg);
    file.read(reinterpret_cast<char*>(floats.data()),
        static_cast<std::streamsize>(fileSize));

    file.close();
    return floats;
}

void BinIO::writeVecToBin(const fvector& vec, const std::string& filename) {
    std::ofstream file(filename.c_str(), std::ios::binary);
```

```
    if (!file.is_open()) {
        throw std::runtime_error("Could not open file: " + filename);
    }
    file.write(reinterpret_cast<const char*>(vec.data()),
        static_cast<std::streamsize>(vec.size() * sizeof(float)));
    file.close();
}
```

Приложение 3: *sleSolver.h*

```
#pragma once

#include <string>
#include <vector>

class sleSolver {
 public:
    typedef std::vector<float> fvector;

    explicit sleSolver(const fvector& locMatA, const fvector& globVecB,
        int rank, const std::vector<int>& vsc, const std::vector<int>& vdp);

    void solve();
    float getDiff(const std::string& vecXbin) const;
    fvector getActualX() const;

 private:
    void computeYAndNorm();
    void computeTau();
    void computeX();

    const fvector& locMatA, globVecB;
    const int globN, locN;
    const int rank;
    const std::vector<int>& vecSendcounts, vecDispls;

    fvector globX, globY, locX, locY;
    float globTau, globNormY;

    const float EPSILON;
};
```

Приложение 4: *sleSolver.cpp*

```
#include "sleSolver.h"

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

```
#include <stdexcept>
#include <vector>
#include <string>
#include "BinIO.h"

sleSolver::sleSolver(const fvector& locMatA, const fvector& globVecB,
int rank, const std::vector<int>& vsc, const std::vector<int>& vdp) :
locMatA(locMatA), globVecB(globVecB),
globN(static_cast<int>(globVecB.size())),
locN(static_cast<int>(locMatA.size()) / globN), rank(rank),
vecSendcounts(vsc), vecDispls(vdp), EPSILON(4e-7) {
    globX = fvector(globN);
    globY = fvector(globN);
    locX = fvector(locN);
    locY = fvector(locN);
    globTau = globNormY = 0;
}

float sleSolver::getDiff(const std::string& vecXbin) const {
    fvector expectedX;
    try {
        expectedX = BinIO::readVecFromBin(vecXbin);
        if (static_cast<int>(expectedX.size()) != globN) return -1;
    } catch (const std::runtime_error&) {
        return -1;
    }

    float diff = 0;
    for (int i = 0; i < globN; i++) {
        diff += std::abs(expectedX[i] - globX[i]);
    }
    return diff;
}

sleSolver::fvector sleSolver::getActualX() const {
    return globX;
}

void sleSolver::computeYAndNorm() {
    float locNormY = 0;
    for (int i = 0; i < locN; i++) {
        float Ax_i = 0;
        for (int j = 0; j < globN; j++) {
            Ax_i += locMatA[i * globN + j] * globX[j];
        }
        locY[i] = Ax_i - globVecB[vecDispls[rank] + i];
        locNormY += locY[i] * locY[i];
    }

    MPI_Allgatherv(locY.data(), vecSendcounts[rank], MPI_FLOAT,
        globY.data(), vecSendcounts.data(),
        vecDispls.data(), MPI_FLOAT, MPI_COMM_WORLD);

    MPI_Allreduce(&locNormY, &globNormY, 1, MPI_FLOAT, MPI_SUM,
```

```
MPI_COMM_WORLD);
    globNormY = std::sqrt(globNormY);
}

void sleSolver::computeTau() {
    float locNumerator = 0, locDenominator = 0;
    for (int i = 0; i < locN; i++) {
        float Ay_i = 0;
        for (int j = 0; j < globN; j++) {
            Ay_i += locMatA[i * globN + j] * globY[j];
        }
        locNumerator += locY[i] * Ay_i;
        locDenominator += Ay_i * Ay_i;
    }

    float globNumerator, globDenominator;
    MPI_Allreduce(&locNumerator, &globNumerator,
        1, MPI_FLOAT, MPI_SUM, MPI_COMM_WORLD);
    MPI_Allreduce(&locDenominator, &globDenominator,
        1, MPI_FLOAT, MPI_SUM, MPI_COMM_WORLD);

    globTau = globNumerator / globDenominator;
}

void sleSolver::computeX() {
    for (int i = 0; i < locN; i++) {
        locX[i] -= globTau * locY[i];
    }
    MPI_Allgatherv(locX.data(), vecSendcounts[rank], MPI_FLOAT,
        globX.data(), vecSendcounts.data(),
        vecDispls.data(), MPI_FLOAT, MPI_COMM_WORLD);
}

void sleSolver::solve() {
    float normB = 0;
    for (int i = 0; i < globN; ++i) {
        normB += globVecB[i] * globVecB[i];
    }
    normB = std::sqrt(normB);

    while (true) {
        computeYAndNorm();
        if (globNormY / normB < EPSILON) break;
        computeTau();
        computeX();
    }
}
```

Приложение 5: *main.cpp*

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

```
#include <string>
#include <vector>
#include <cstdlib>
#include <cfloat>

#include "sleSolver.h"
#include "BinIO.h"

#define LOOPS 8

void solveSLE(const std::vector<float>& localMatA,
const std::vector<float>& vecB, int rank, int commSize) {
    if (rank == 0) {
        std::cout << "Running with " <<
            commSize << " processes..." << std::endl;
    }

    std::vector<int> vecSendcounts = std::vector<int>(commSize);
    std::vector<int> vecDispls = std::vector<int>(commSize);

    const int baseSize = static_cast<int>(vecB.size()) / commSize;
    const int remainder = static_cast<int>(vecB.size()) % commSize;
    int shift = 0;
    for (int i = 0; i < commSize; ++i) {
        vecSendcounts[i] = baseSize + (i < remainder);
        vecDispls[i] = shift;
        shift += vecSendcounts[i];
    }

    double duration = DBL_MAX;
    for (int i = 0; i < LOOPS; i++) {
        sleSolver solver(localMatA, vecB, rank, vecSendcounts, vecDispls);
        const double start = MPI_Wtime();
        solver.solve();
        const double end = MPI_Wtime();
        duration = std::min(duration, end - start);
    }

    if (rank == 0) {
        std::cout << "Done! Time taken: " << duration << " sec" <<
std::endl;

        // BinIO::writeVecToBin(solver.getActualX(), "actualX.bin");

        // const float diff = solver.getDiff("vecX.bin");
        // const float avg = diff / static_cast<float>(vecB.size());
        // std::cout << "Total diff: " << diff <<
        //     " (avg " << avg << ")" << std::endl;
        std::cout << "================================" <<
                     "================================" << std::endl;
    }
}

void prepareScatterv(const int vecSize, const int commSize,
```

```
std::vector<int>& matSendcounts, std::vector<int>& matDispls) {
    const int baseSize = vecSize / commSize;
    const int remainder = vecSize % commSize;
    int shift = 0;
    for (int i = 0; i < commSize; ++i) {
        matSendcounts[i] = (baseSize + (i < remainder)) * vecSize;
        matDispls[i] = shift;
        shift += matSendcounts[i];
    }
}

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);

    try {
        std::vector<float> matA, vecB;
        if (rank == 0) {
            matA = BinIO::readVecFromBin("matA.bin");
            vecB = BinIO::readVecFromBin("vecB.bin");
            if (matA.size() != vecB.size() * vecB.size()) {
                throw std::runtime_error(
                    "Invalid matrix size: matA must be of size N x N.");
            }
        }

        int N = static_cast<int>(vecB.size());
        MPI_Bcast(&N, 1, MPI_INT, 0, MPI_COMM_WORLD);

        if (rank != 0) vecB.resize(N);
        MPI_Bcast(vecB.data(), N, MPI_FLOAT, 0, MPI_COMM_WORLD);

        std::vector<int> matSendcounts = std::vector<int>(size);
        std::vector<int> matDispls = std::vector<int>(size);
        prepareScatterv(N, size, matSendcounts, matDispls);

        std::vector<float> localMatA(matSendcounts[rank]);
        MPI_Scatterv(matA.data(), matSendcounts.data(),
            matDispls.data(), MPI_FLOAT, localMatA.data(),
            matSendcounts[rank], MPI_FLOAT, 0, MPI_COMM_WORLD);

        solveSLE(localMatA, vecB, rank, size);
    } catch (const std::runtime_error& e) {
        std::cerr << "Process " << rank << ": " << e.what() << std::endl;
        MPI_Abort(MPI_COMM_WORLD, EXIT_FAILURE);
    }

    MPI_Finalize();
    return EXIT_SUCCESS;
}
```