###### 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

РЕШЕНИЕ СИСТЕМ ЛИНЕЙНЫХ АЛГЕБРАИЧЕСКИХ УРАВНЕНИЙ ИТЕРАЦИОННЫМИ МЕТОДАМИ

студента 2 курса, группы 22204

Соломенникова Николая Александровича

Направление 09.03.01 – «Информатика и вычислительная техника»

Преподаватель:

доцент

А.Ю.Власенко

Новосибирск 2023

**СОДЕРЖАНИЕ**

[ЦЕЛЬ 2](#__RefHeading___1)

[ЗАДАНИЕ 2](#__RefHeading___2)

[ОПИСАНИЕ РАБОТЫ 3](#__RefHeading___3)

[ЗАКЛЮЧЕНИЕ 6](#__RefHeading___4)

[ИСПОЛЬЗУЕМЫЕ КОМАНДЫ 6](#__RefHeading___5)

[ИСХОДНИКИ 7](#__RefHeading___6)

[config.h (для констант) 7](#__RefHeading___13)

[prog.cpp (последовательная программа) 7](#__RefHeading___10)

[prog2.cpp (параллельная программа) 11](#__RefHeading___8)

[launch.sbatch 19](#__RefHeading___16)

# ЦЕЛЬ

Решить задачу нахождения решений системы линейных алгебраических уравнений итерационным методом и ускорить программу с помощью распараллеливания.

# ЗАДАНИЕ

1. Написать 2 программы (последовательную и параллельную с использованием MPI) на языке C/C++, которые реализуют итерационный алгоритм решения системы линейных алгебраических уравнений вида *Ax=b* в соответствии с выбранным вариантом. Здесь *A*– матрица размером N×N, x и *b* – векторы длины N. Тип элементов – double.

2. Параллельную программу реализовать с тем условием, что матрица *A* и вектор *b* инициализируются на каком-либо одном процессе, а затем матрица *A* «разрезается» по строкам на близкие по размеру, возможно не одинаковые, части, а вектор *b* раздается каждому процессу.  
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 последовательного варианта программы, а также время работы параллельного при использовании различного числа процессорных ядер. Минимально на 2, 4, 8, 16, 24 (на каждом из наших узлов кластера по 12 ядер), но чем на большем количестве различного числа процессов будут выполнены замеры, тем лучше. Также чем больше замеров будет выполнено на одном и том же числе процессов, тем лучше. В этом случае для построения графиков следует брать минимальное время.  
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 jumpshot или ITAC (Intel Trace Analyzer and Collecter) при использовании 16-и или 24-х ядер.

5. На основании полученных результатов сделать вывод.

Вариант задачи - метод простой итерации

# ОПИСАНИЕ РАБОТЫ

В параллельной версии программы успользовались функции коллективной коммуникации с переменным количеством элементов, то есть векторы разбивались на не равные части. При этом размеры частей выбирались так, чтобы они отличаются максимум на единицу.

Результаты замеров времени для последовательной программы: T0 = 36,1

А для параллельной программы:

|  |  |  |  |
| --- | --- | --- | --- |
| Кол-во процессов | Кол-во процессов на узел | Кол-во  запрашиваемых узлов | Время |
| 1 | 1 | 1 | 39 |
| 2 | 2 | 2 | 23,6 |
| 4 | 4 | 4 | 11 |
| 8 | 4 | 4 | 5,6 |
| 12 | 4 | 4 | 3,5 |
| 16 | 8 | 4 | 5,4 |
| 24 | 8 | 4 | 3,7 |

Количество процессов на узел подбиралось так, чтобы было минимальное время, а количество узлов бралось с запасом для более стабильных результатов.

Значения констант брались следующие:

* количество строк в системе N = 2500
* максимально количество итераций = 10000 (при тестах было около 6000 итераций)
* константа epsilon = 0,00001
* константа tau = 0,0005 (используется в алгоритме)

График времени работы:



График ускорения (Sp = T0 / Tp, где Tp – время работы программы на p процессах):

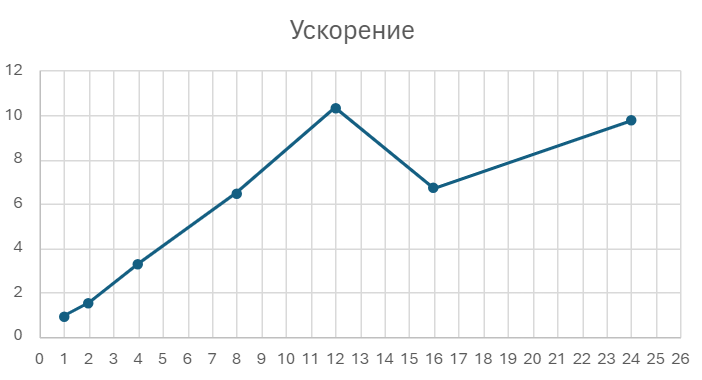
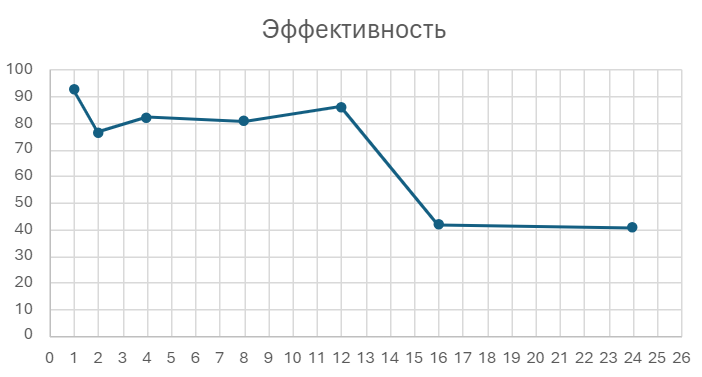
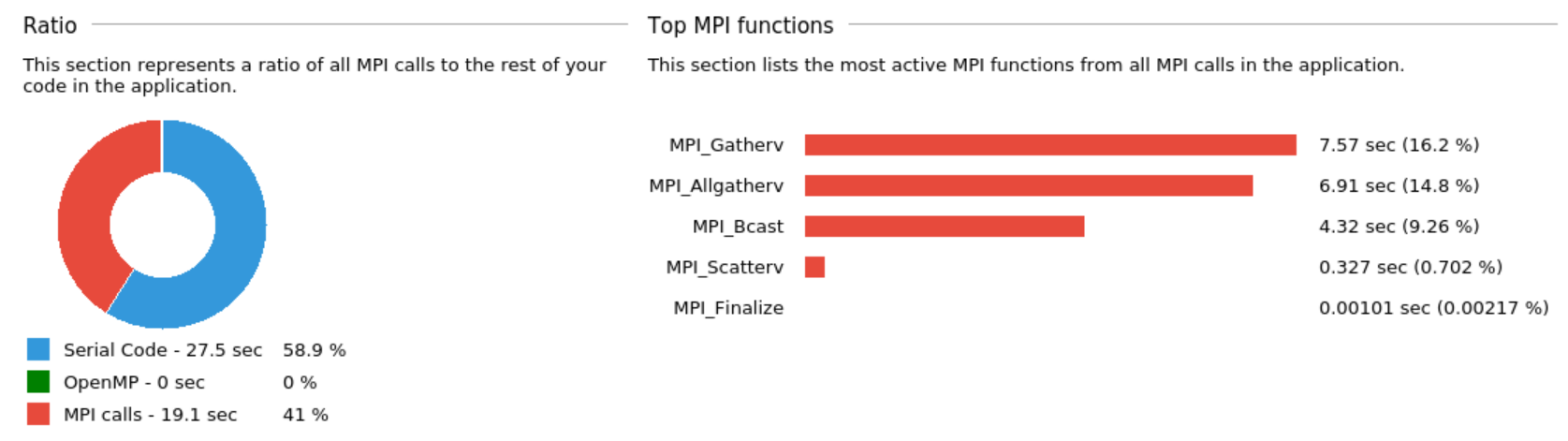


График эффективности (Ep = Sp / p \* 100%):

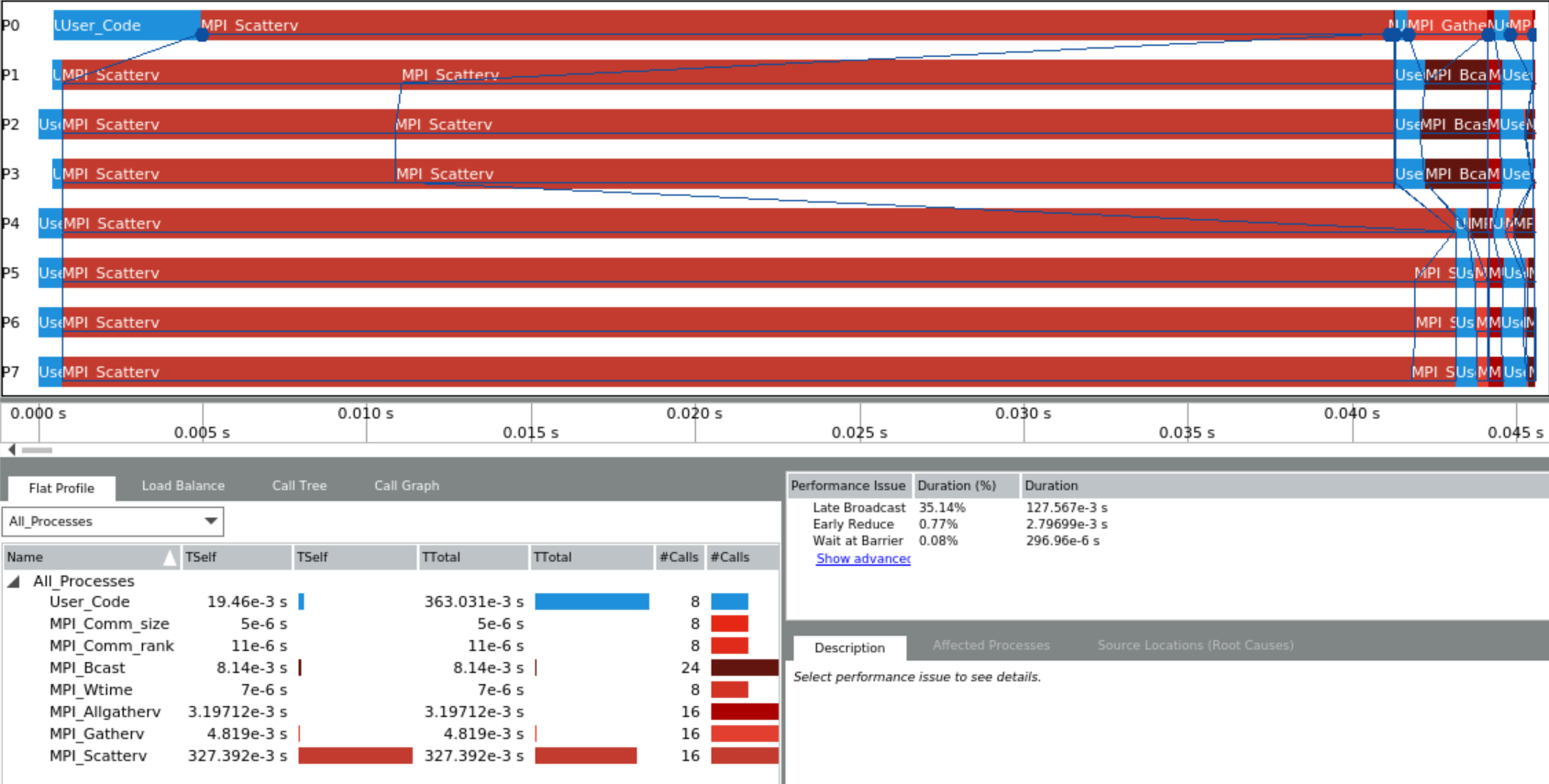


Профилирование делалось на основе параллельной программы, запущенной на восьми процессах.

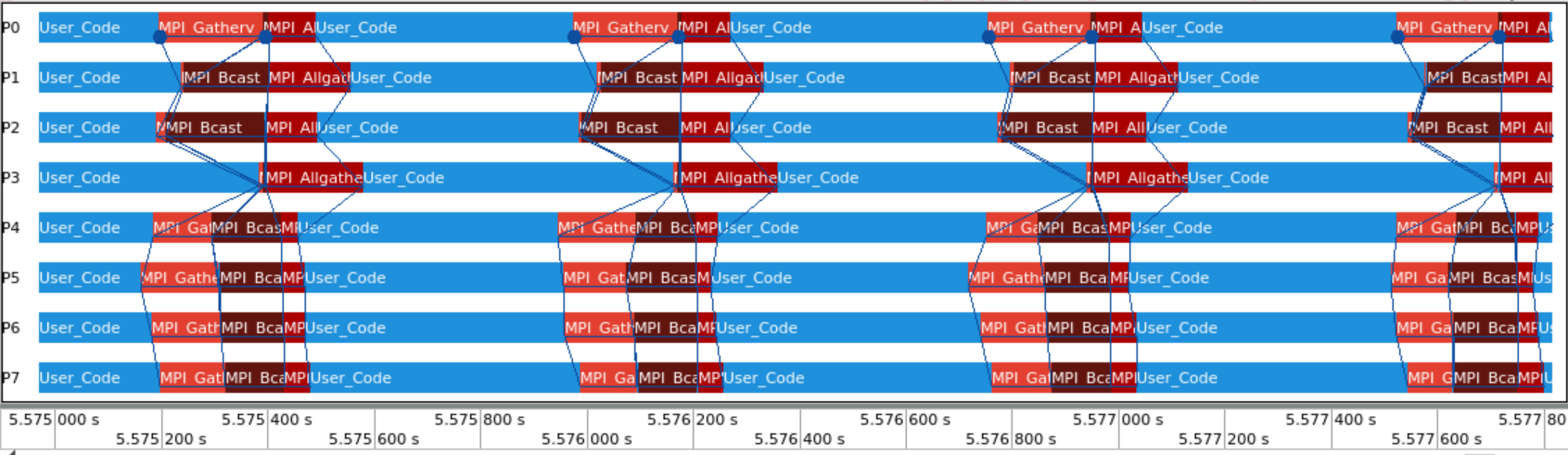
Общие сведения о работе программы:



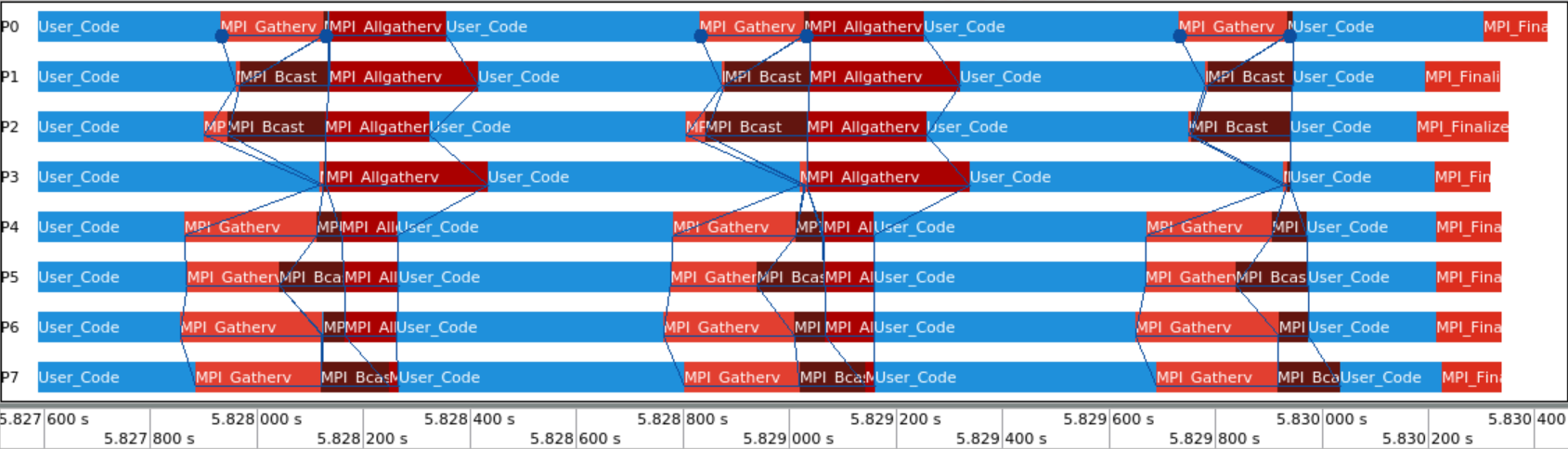
Начало работы программы:



Середина работы программы:



Конец работы программы:



# ЗАКЛЮЧЕНИЕ

В ходе эксперементов было замечено, что чем больше параметр TAU, тем быстрее сходится ответ, но если TAU слишком большой, то ответ начинает расходиться.

Ускорение программы увеличивается при увеличении количества процессов. Пик достигается при 12 процессах (ускорение примерно в 10 раз). При большем количестве процессов время программы не уменьшается.

В начале работы параллельной программы, как и ожидалось, много времени ухоит на то, чтобы распределить матрицу A и вектор b по процессам. Интересно заметить, что 1, 2 и 3 процессы выполняют scatterv быстрее чем остальные. Это связано с тем, что они рапускались но том же узле, что и корневой процесс (нулевой). В середине работы программы всё тоже работает как надо (выполняется пользовательский код, потом проходят коллективные операции, причём нет процессов, которые подолгу ждут остальные процессы).

# ИСПОЛЬЗУЕМЫЕ КОМАНДЫ

Загрузка модулей:

$module load mpi/intelmpi

$module load itac

Компиляция mpi программы (реализация от компании intel) с использованием компилятора icpc:

$mpiicpc src/prog1.cpp -o src/prog1.out

Поставить задачу на выполнение:

$sbatch ./launch.sbatch

Посмтреть очередь:

$squeue

Для профилирования:

$traceanalyzer prog2.out.stf

Компиляция и запуск для тех, кто не ищет лёгких путей:

$export PATH=/opt/intel/basekit/mpi/2021.4.0/bin:$PATH

$export CPLUS\_INCLUDE\_PATH=/opt/intel/basekit/mpi/2021.4.0/include/:$CPLUS\_INCLUDE\_PATH

$mpicxx src/prog1.cpp -o src/prog1.out -L /opt/intel/basekit/mpi/2021.4.0/lib -L /opt/intel/basekit/mpi/2021.4.0/lib/release

$export LD\_LIBRARY\_PATH=/opt/intel/basekit/mpi/2021.4.0/lib/release:$LD\_LIBRARY\_PATH

$mpiexec -n 8 src/prog2.out

# ИСХОДНИКИ

### config.h (для констант)

const int N = 2500;                   // Размер векторов

constexpr int MAX\_ITERS = 1e4;        // Максимальное количество итераций

constexpr double EPSILON = 1e-5;      // Точность вычислений

constexpr double TAU = 5e-4;          // Константа, используемая в алгоритме

const double CALCULATION\_ERROR = 0.1; // Погрешность вычислений (нужно для проверки ответа)

### prog.cpp (последовательная программа)

#include "config.h"

#include <cmath>

#include <exception>

#include <iostream>

#include <mpi.h>

// Класс исключений для отслеживания ошибок

class myException : public std::exception

{

private:

    const char \*msg\_;

public:

    explicit myException(const char \*msg) : msg\_(msg){};

    const char \*what() const noexcept override { return msg\_; }

};

// Инициализация входных данных и вектора для записи результата

void initData(double \*&A, double \*&b, double \*&x, double \*&res)

{

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

    if (A == nullptr)

        throw myException("Malloc fail");

    for (int i = 0; i < N; ++i)

    {

        for (int j = 0; j < N; ++j)

        {

            if (i == j)

                A[i \* N + j] = 2.0;

            else

                A[i \* N + j] = 1.0;

        }

    }

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

    if (b == nullptr)

        throw myException("Malloc fail");

    for (int i = 0; i < N; ++i)

        b[i] = N + 1.0;

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

    if (x == nullptr)

        throw myException("Malloc fail");

    for (int i = 0; i < N; ++i)

        x[i] = sin(i);

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

    if (res == nullptr)

        throw myException("Malloc fail");

}

// Освобождение ресурсов

void deleteData(double \*&A, double \*&b, double \*&x, double \*&res)

{

    free(A);

    free(b);

    free(x);

    free(res);

}

// res = A \* x

void mul(const double \*const &A, const double \*const &x, double \*const &res)

{

    for (int i = 0; i < N; ++i)

    {

        res[i] = 0;

        const int iN = i \* N;

        for (int j = 0; j < N; ++j)

            res[i] += A[iN + j] \* x[j];

    }

}

// res = res \* TAU

void mulTAU(double \*const &res)

{

    for (int i = 0; i < N; ++i)

        res[i] \*= TAU;

}

// res = res - b

void sub(double \*const &res, const double \*const &b)

{

    for (int i = 0; i < N; ++i)

        res[i] -= b[i];

}

// Квадрат модуля вектора

double mod2(const double \*const &x)

{

    double ans = 0;

    for (int i = 0; i < N; ++i)

        ans += x[i] \* x[i];

    return ans;

}

// res = x

void copy(const double \*const &x, double \*const &res)

{

    for (int i = 0; i < N; ++i)

        res[i] = x[i];

}

// Модуль для double

double abs(const double a)

{

    return a < 0 ? -a : a;

}

// Проверка ответа на корректность

void testAns(const double \*const &res)

{

    for (int i = 0; i < N; ++i)

    {

        // std::cout << abs(res[i] - 1.0) << " ";

        if (abs(res[i] - 1.0) > CALCULATION\_ERROR)

        {

            fprintf(stderr, "res[%d] = %lf, but expected 1.0\n", i, res[i]);

            throw myException("Wrong result");

        }

    }

}

int main()

{

    double \*A, \*b, \*x, \*res;

    const double startTime = MPI\_Wtime();

    try

    {

        // Инициализация начальных данных

        initData(A, b, x, res);

        int iters = 0; // Количество итераций алгоритма

        const double constForCheck = mod2(b) \* EPSILON \* EPSILON; // Для проверки критерия завершения счёта

        while (true)

        {

            mul(A, x, res);

            sub(res, b);

            if (mod2(res) < constForCheck)

            {

                copy(x, res);

                break;

            }

            mulTAU(res);

            sub(x, res);

            if ((++iters) > MAX\_ITERS)

                throw myException("Too many iterations");

        }

        const double endTime = MPI\_Wtime();

        std::cout << "Total time: " <<endTime - startTime << "\n";

        testAns(res);

    }

    catch (const myException &e)

    {

        std::cerr << e.what() << '\n';

    }

    deleteData(A, b, x, res);

    return 0;

}

### prog2.cpp (параллельная программа)

#include "config.h"

#include <cmath>

#include <exception>

#include <iostream>

#include <mpi.h>

// Класс исключений для отслеживания ошибок

class myException : public std::exception

{

private:

    const char \*msg\_;

public:

    explicit myException(const char \*msg) : msg\_(msg){};

    const char \*what() const noexcept override { return msg\_; }

};

// Инициализация и заполнение массивов для размеров частей и их начальных позиций

void initSendcountsAndDispls(int \*&sendcountsA, int \*&sendcountsB, int \*&displsA, int \*&displsB, const int size)

{

    const int bigParts = N % size;

    const int smallPartSize = N / size;

    int pos = 0;

    sendcountsA = (int \*)malloc(sizeof(int) \* size);

    sendcountsB = (int \*)malloc(sizeof(int) \* size);

    displsA = (int \*)malloc(sizeof(int) \* size);

    displsB = (int \*)malloc(sizeof(int) \* size);

    if (sendcountsA == nullptr || sendcountsB == nullptr || displsA == nullptr || displsB == nullptr)

        throw myException("Malloc fail");

    for (int p = 0; p < bigParts; ++p)

    {

        sendcountsA[p] = (smallPartSize + 1) \* N;

        displsA[p] = pos;

        pos += (smallPartSize + 1) \* N;

    }

    for (int p = bigParts; p < size; ++p)

    {

        sendcountsA[p] = smallPartSize \* N;

        displsA[p] = pos;

        pos += smallPartSize \* N;

    }

    pos = 0;

    for (int p = 0; p < bigParts; ++p)

    {

        sendcountsB[p] = smallPartSize + 1;

        displsB[p] = pos;

        pos += smallPartSize + 1;

    }

    for (int p = bigParts; p < size; ++p)

    {

        sendcountsB[p] = smallPartSize;

        displsB[p] = pos;

        pos += smallPartSize;

    }

}

// Инициализация буферов для корневого процесса

void initRootData(double \*&A, double \*&b, double \*&partA, double \*&partB, double \*&x, double \*&res, double \*&partRes, const int \*const &sendcountsA, const int \*const &sendcountsB)

{

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

    if (A == nullptr)

        throw myException("Malloc fail");

    for (int i = 0; i < N; ++i)

    {

        for (int j = 0; j < N; ++j)

        {

            if (i >= N)

                A[i \* N + j] = 0.0;

            else if (i == j)

                A[i \* N + j] = 2.0;

            else

                A[i \* N + j] = 1.0;

        }

    }

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

    if (b == nullptr)

        throw myException("Malloc fail");

    for (int i = 0; i < N; ++i)

        b[i] = N + 1.0;

    partA = (double \*)malloc(sizeof(double) \* sendcountsA[0]);

    partB = (double \*)malloc(sizeof(double) \* sendcountsB[0]);

    if (partA == nullptr || partB == nullptr)

        throw myException("Malloc fail");

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

    if (x == nullptr)

        throw myException("Malloc fail");

    for (int i = 0; i < N; ++i)

        x[i] = sin(i);

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

    partRes = (double \*)malloc(sizeof(double) \* sendcountsB[0]);

    if (res == nullptr || partRes == nullptr)

        throw myException("Malloc fail");

}

// Инициализация буферов для не корневых процессов

void initSideData(double \*&partA, double \*&partB, double \*&x, double \*&partRes, const int \*const &sendcountsA, const int \*const &sendcountsB, const int rank)

{

    partA = (double \*)malloc(sizeof(double) \* sendcountsA[rank]);

    partB = (double \*)malloc(sizeof(double) \* sendcountsB[rank]);

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

    partRes = (double \*)malloc(sizeof(double) \* sendcountsB[rank]);

    if (partA == nullptr || partB == nullptr || x == nullptr || partRes == nullptr)

        throw myException("Malloc fail");

}

// Освобождение ресурсов

void deleteData(double \*&A, double \*&b, double \*&partA, double \*&partB, double \*&x, double \*&res, double \*&partRes, int \*&sendcountsA, int \*&sendcountsB, int \*&displsA, int \*&displsB)

{

    free(A);

    free(b);

    free(partA);

    free(partB);

    free(x);

    free(res);

    free(partRes);

    free(sendcountsA);

    free(sendcountsB);

    free(displsA);

    free(displsB);

}

// partRes = partA \* x

void mul(const double \*const &partA, const double \*const &x, double \*const &partRes, const int partSize)

{

    for (int i = 0; i < partSize; ++i)

    {

        partRes[i] = 0;

        const int iN = i \* N;

        for (int j = 0; j < N; ++j)

            partRes[i] += partA[iN + j] \* x[j];

    }

}

// partRes = partRes \* TAU

void mulTAU(double \*const &partRes, const int partSize)

{

    for (int i = 0; i < partSize; ++i)

        partRes[i] \*= TAU;

}

// partRes = partRes - partB

void sub(double \*const &partRes, const double \*const &partB, const int partSize)

{

    for (int i = 0; i < partSize; ++i)

        partRes[i] -= partB[i];

}

// partRes = partX - partRes

void subPartX(double \*const &partRes, const double \*const &x, const int partSize, const int startPos)

{

    for (int i = 0; i < partSize; ++i)

        partRes[i] = x[startPos + i] - partRes[i];

}

// Квадрат модуля вектора длины N

double mod2(const double \*const &x)

{

    double ans = 0;

    for (int i = 0; i < N; ++i)

        ans += x[i] \* x[i];

    return ans;

}

// Клпиркет первые N элементов из x в res

void copy(const double \*const &x, double \*const &res)

{

    for (int i = 0; i < N; ++i)

        res[i] = x[i];

}

// Модуль для double

double abs(const double a)

{

    return a < 0 ? -a : a;

}

// Проверка ответа на корректность

void testAns(const double \*const &res)

{

    for (int i = 0; i < N; ++i)

    {

        // std::cout << res[i]  << " ";

        // std::cout << abs(res[i] - 1.0) << " ";

        if (abs(res[i] - 1.0) > CALCULATION\_ERROR)

        {

            fprintf(stderr, "res[%d] = %lf, but expected 1.0\n", i, res[i]);

            throw myException("Wrong result");

        }

    }

}

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

{

    int rank, size;

    MPI\_Init(&argc, &argv);

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

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

    double \*A = nullptr, \*b = nullptr, \*x = nullptr, \*res = nullptr;

    double \*partA = nullptr, \*partB = nullptr, \*partRes = nullptr;

    int \*sendcountsA = nullptr, \*sendcountsB = nullptr;

    int \*displsA = nullptr, \*displsB = nullptr;

    const double startTime = MPI\_Wtime();

    try

    {

        // Инициализация начальных данных

        initSendcountsAndDispls(sendcountsA, sendcountsB, displsA, displsB, size);

        if (rank == 0)

            initRootData(A, b, partA, partB, x, res, partRes, sendcountsA, sendcountsB);

        else

            initSideData(partA, partB, x, partRes, sendcountsA, sendcountsB, rank);

        // Раздача данных не корневым процессам

        MPI\_Scatterv(A, sendcountsA, displsA, MPI\_DOUBLE, partA, sendcountsA[rank], MPI\_DOUBLE, 0, MPI\_COMM\_WORLD);

        MPI\_Scatterv(b, sendcountsB, displsB, MPI\_DOUBLE, partB, sendcountsB[rank], MPI\_DOUBLE, 0, MPI\_COMM\_WORLD);

        MPI\_Bcast(x, N, MPI\_DOUBLE, 0, MPI\_COMM\_WORLD);

        int iters = 0;        // Количество итераций алгоритма

        double constForCheck; // Для проверки критерия завершения счёта

        if (rank == 0)

            constForCheck = mod2(b) \* EPSILON \* EPSILON;

        char flag = 0; // 0 - продолжать счёт, 1 - счёт удачно закончен, 2 - слишком много итераций

        while (true)

        {

            // Вычисления (partRes = partA \* x - partB)

            mul(partA, x, partRes, sendcountsA[rank] / N);

            sub(partRes, partB, sendcountsB[rank]);

            // Проверка на то, нужно ли завершать вычисления

            MPI\_Gatherv(partRes, sendcountsB[rank], MPI\_DOUBLE, res, sendcountsB, displsB, MPI\_DOUBLE, 0, MPI\_COMM\_WORLD);

            if (rank == 0 && mod2(res) < constForCheck)

            {

                copy(x, res);

                flag = 1;

            }

            MPI\_Bcast(&flag, 1, MPI\_CHAR, 0, MPI\_COMM\_WORLD);

            if (flag == 1)

                break;

            // Вычисления (partRes = partX - TAU \* (partA \* x - partB))

            mulTAU(partRes, sendcountsB[rank]);

            subPartX(partRes, x, sendcountsB[rank], displsB[rank]);

            // Переход на новую итерацию (x\_(n+1) = f(x\_n))

            MPI\_Allgatherv(partRes, sendcountsB[rank], MPI\_DOUBLE, x, sendcountsB, displsB, MPI\_DOUBLE, MPI\_COMM\_WORLD);

            // Проверка на количество итераций

            if ((++iters) > MAX\_ITERS)

            {

                flag = 2;

                break;

            }

        }

        if (rank == 0 && flag == 2)

            std::cerr << "Too many iterations\n";

        else if (rank == 0)

        {

            const double endTime = MPI\_Wtime();

            std::cout << "Total time: " << endTime - startTime << "\n";

            testAns(res);

        }

    }

    catch (const myException &e)

    {

        std::cerr << "Process " << rank << ": " << e.what() << '\n';

    }

    deleteData(A, b, partA, partB, x, res, partRes, sendcountsA, sendcountsB, displsA, displsB);

    MPI\_Finalize();

    return 0;

}

### launch.sbatch

#!/bin/bash

#SBATCH -J solomennikov

#SBATCH -p compclass

#SBATCH -o reports/job%j.out

#SBATCH -e reports/job%j.err

#SBATCH -N 4 # Total number of nodes requested

#SBATCH -n 8 # Total number of mpi tasks requested

#SBATCH -t 00:01:00

#Run executable files and commands

module load mpi/intelmpi

module load itac

#TCP-ports, opened for MPI-communications

export I\_MPI\_PORT\_RANGE=23000:23100

which mpiexec

#mpiexec -ppn 1 -n 1 ./src/prog1.out

mpiexec -trace -ppn 4 -n 8 ./src/prog2.out