###### 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 ЛАБОРАТОРНОЙ РАБОТЫ**

«Параллельная реализация решения системы линейных алгебраических уравнений с помощью MPI»

студента 2 курса, 21203 группы

**Неретина Степана Ивановича**

Направление 09.03.01 – «Информатика и вычислительная техника»

Преподаватель:

А.Ю. Власенко

Новосибирск 2023

**СОДЕРЖАНИЕ**

Цель [3](#_heading=h.gjdgxs)

Задание [4](#_heading=h.30j0zll)

Ход [работы 5](#_heading=h.30j0zll)

Заключение6

[Приложение 1. Листинг программы вариант1](#_heading=h.1fob9te)  12

Приложение 2. Листинг программы вариант 2 14

Приложение 3. Листинг программы вариант 3 18

Приложение 4. Один из выводов сбора времени 21

# Цель

Разобраться с базовыми возможностями работы с библиотекой MPI на примере распараллеливания одной задачи.

# ЗАДАНИЕ

I Написать 3 программы, каждая из которых рассчитывает число s по двум данным векторам a и b равной длины N в соответствии со следующим двойным циклом:

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

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

s += a[i] \* b[j];

a) последовательная программа

b) параллельная, использующая коммуникации типа точка-точка (MPI\_Send, MPI\_Recv)

c) параллельная, использующая коллективные коммуникации (MPI\_Scatter, MPI\_Reduce, MPI\_Bcast)

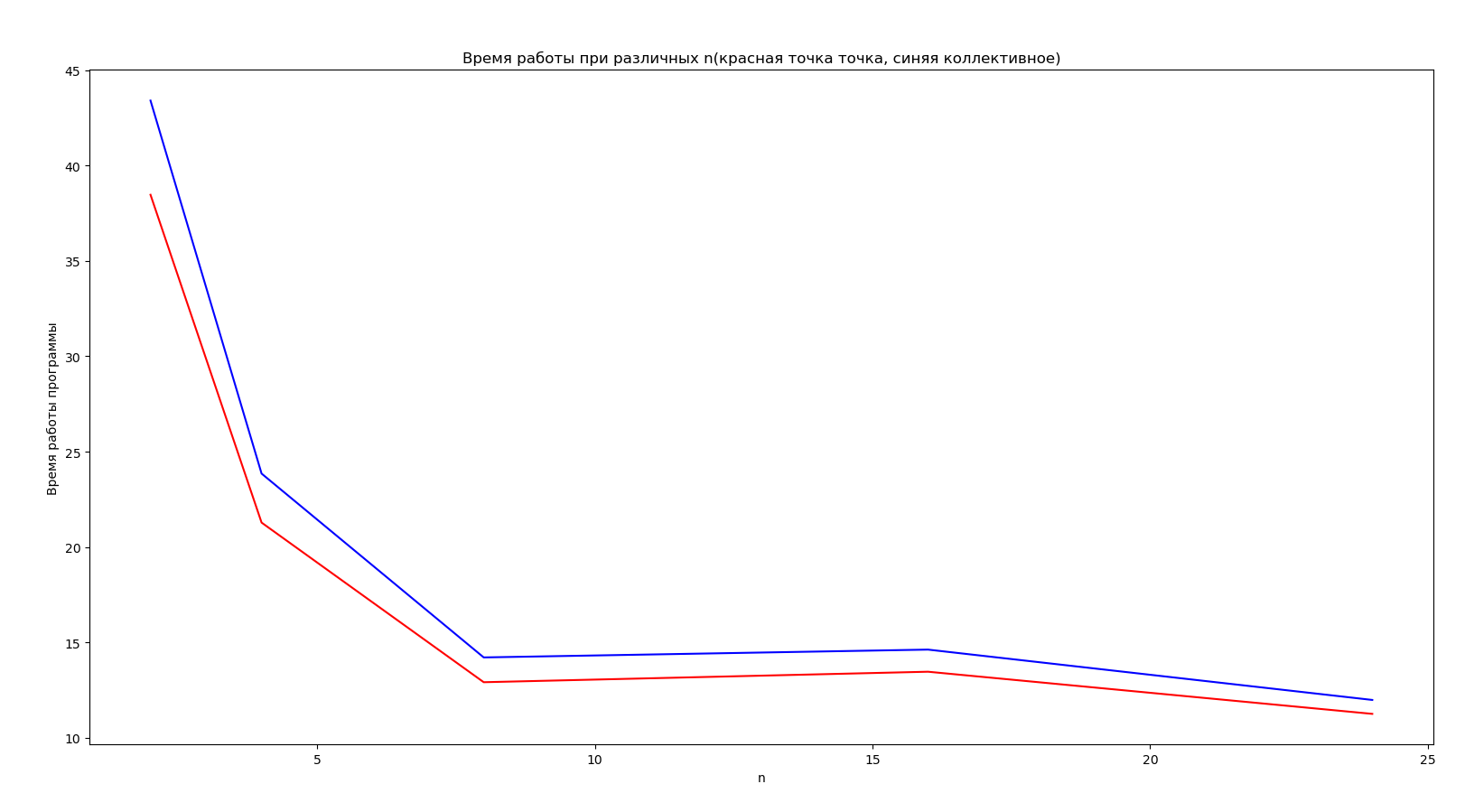
II Замерить время работы последовательной программы и параллельных на 2, 4, 8, 16, 24 процессах. Рекомендуется провести несколько замеров для каждого варианта запуска и выбрать минимальное время.

III Построить графики времени, ускорения и эффективности.

IV Составить отчет, содержащий исходные коды разработанных программ и построенные графики.

# Ход работы

В ходе работы были реализованы 3 версии программы и замерено время работы.



Ускорение программы и эффективность при различных n:

n = 2

2,25 – Точка точка

2,001 - Коллективное

Точка точка, коллективное

Эффективность (112,5 100,05)

n = 4

4,083 – Точка точка

3,643 - Коллективное

Точка точка, коллективное

Эффективность (102,07 91,07)

n = 8

6,731 – Точка точка

6,11 - Коллективное

Точка точка, коллективное

Эффективность (84,13 76,37)

n = 16

6,45 – Точка точка

5,94 - Коллективное

Точка точка, коллективное

Эффективность (40,31 37,12)

n = 24

7,72 – Точка точка

7,25 - Коллективное

Точка точка, коллективное

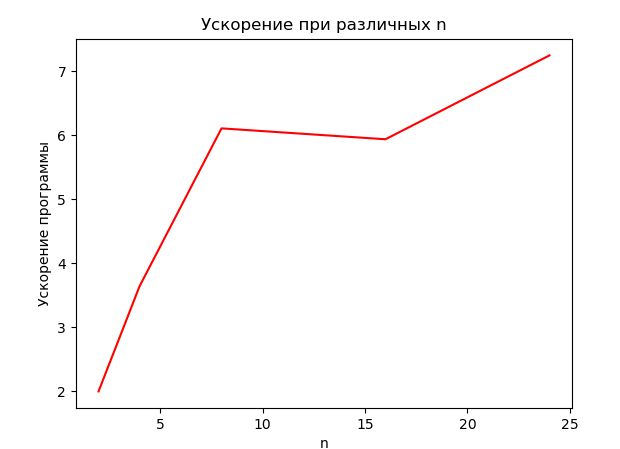
Эффективность (32,16 30,20)

Точка точка



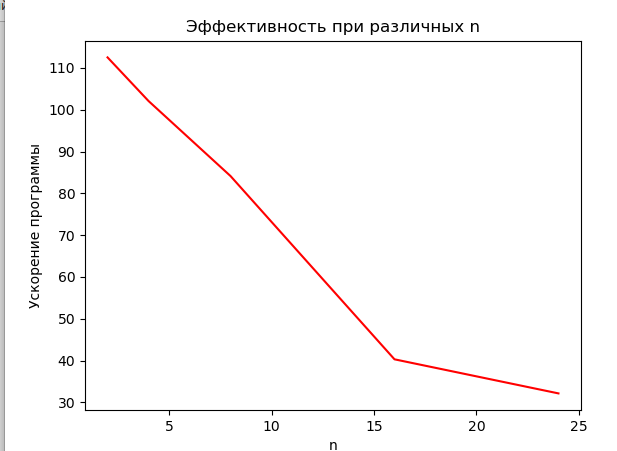
\

Коллективное

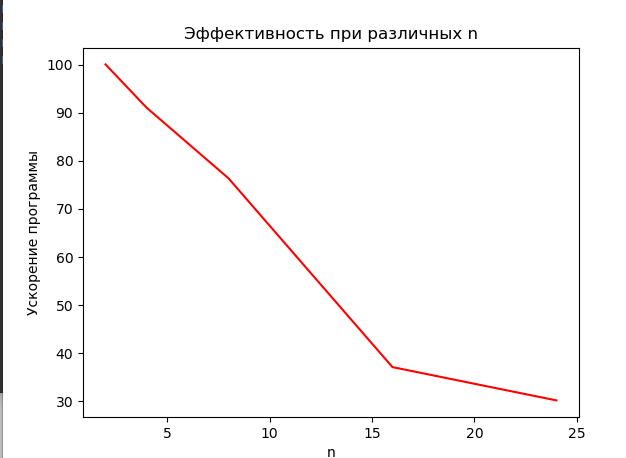


Эффективность

точка точка



коллективное



# Заключение

Познакомились с базовыми принципами распараллеливания программ при помощи MPI. Сделан вывод о том, что распараллеливание программы может существенно ускорить скорость выполнения

# **Приложение 1. Листинг программы вариант 1**

#include <cstdio>

#include <random>

#include <iostream>

using namespace std;

inline int VECTOR\_SIZE = 150000;

inline int generateRandomInRange(int a, int b) {

random\_device rd;

uniform\_int\_distribution gen(a, b);

return gen(rd);

}

void fill\_vector\_values(int \*vec){

for(size\_t i = 0; i < VECTOR\_SIZE; i++){

vec[i] = generateRandomInRange(1, 100);

}

}

void print\_vec(int \*vec){

for(size\_t i = 0; i < VECTOR\_SIZE; i++){

std::cout << vec[i] << endl;

}

std::cout << "---------" << std::endl;

}

int calculate(const int \*vec1, const int \*vec2){

int res = 0;

for (int i = 0; i < VECTOR\_SIZE; i++){

for (int j = 0; j < VECTOR\_SIZE; j++){

// std::cout << i << " " << j << endl;

res += vec1[i] \* vec2[j];

}

//std::cout << std::endl;

}

return res;

}

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

std::cout << VECTOR\_SIZE << endl;

auto \*vec1 = new int[VECTOR\_SIZE];

auto \*vec2 = new int[VECTOR\_SIZE];

fill\_vector\_values(vec1);

fill\_vector\_values(vec2);

auto res = calculate(vec1, vec2);

std::cout << res;

delete []vec1;

delete []vec2;

}

# **Приложение 2. Листинг программы вариант 2**

#include <iostream>

#include "mpi.h"

#include <random>

#include <cassert>

using namespace std;

//#define VECTOR\_SIZE 200000

#define VECTOR\_SIZE 150000

inline int generateRandomInRange(int a, int b) {

random\_device rd;

uniform\_int\_distribution gen(a, b);

return gen(rd);

}

void fill\_vector\_values(int \*vec){

for(size\_t i = 0; i < VECTOR\_SIZE; i++){

vec[i] = generateRandomInRange(1, 100);

}

}

void print\_vec(int \*vec, int n){

for(size\_t i = 0; i < n; i++){

std::cout << vec[i] << endl;

}

}

int calculate(const int \*vec1, const int \*vec2){

int res = 0;

for (int i = 0; i < VECTOR\_SIZE; i++){

for (int j = 0; j < VECTOR\_SIZE; j++){

res += vec1[i] \* vec2[j];

}

}

return res;

}

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

int rank, size;

int VEC\_SIZE = VECTOR\_SIZE;

MPI\_Init(&argc, &argv);

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

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

size\_t sub\_size = VEC\_SIZE / size;

size\_t rest = VEC\_SIZE % size;

int result = 0;

auto\* received\_vec1 = new int[sub\_size];

auto\* received\_vec2 = new int[VECTOR\_SIZE];

if(rank == 0){

std::cout << sub\_size << " " << rest << endl;

auto \*vec1 = new int[VECTOR\_SIZE];

auto \*vec2 = new int[VECTOR\_SIZE];

fill\_vector\_values(vec1);

fill\_vector\_values(vec2);

for(size\_t i = 1; i < size; i++) {

//std::cout << sub\_size\*i << endl;

MPI\_Send(&vec1[sub\_size\*i], int(sub\_size), MPI\_INT, (int)i, (int)i, MPI\_COMM\_WORLD);

MPI\_Send(vec2, int(VEC\_SIZE), MPI\_INT, (int)i, (int)i, MPI\_COMM\_WORLD);

}

for(size\_t i = 0; i < rest; i++){

//std::cout << sub\_size\*size + i << endl;

if(i == 0){

for(size\_t j = 0; j < VECTOR\_SIZE; j++){

result += vec1[sub\_size\*size + i]\*vec2[j];

}

continue;

}

MPI\_Send(&vec1[sub\_size\*size + i], 1, MPI\_INT, (int)i, (int)i, MPI\_COMM\_WORLD);

MPI\_Send(vec2, VEC\_SIZE, MPI\_INT, (int)i, (int)i, MPI\_COMM\_WORLD);

}

for(size\_t i = 0; i < sub\_size; i++){

int fixed\_el = vec1[i];

//std::cout << i << endl;

for(size\_t j = 0; j < VECTOR\_SIZE; j++){

result += fixed\_el\*vec2[j];

}

}

for(size\_t i = 1; i < size; i++){

int local\_res;

MPI\_Recv(&local\_res, 1, MPI\_INT, MPI\_ANY\_SOURCE, (int)i, MPI\_COMM\_WORLD, MPI\_STATUS\_IGNORE);

result += local\_res;

}

std::cout << "Result " << result << endl;

//std::cout << "calculate " << calculate(vec1, vec2) << endl;

delete []vec1;

delete []vec2;

}

if(rank != 0){

MPI\_Recv(received\_vec1, (int)sub\_size, MPI\_INT, 0, rank, MPI\_COMM\_WORLD, MPI\_STATUS\_IGNORE);

MPI\_Recv(received\_vec2, VEC\_SIZE, MPI\_INT, 0, rank, MPI\_COMM\_WORLD, MPI\_STATUS\_IGNORE);

int curr\_res = 0;

for(size\_t i = 0; i < sub\_size; i++){

int fixed\_el = received\_vec1[i];

for(size\_t j = 0; j < VECTOR\_SIZE; j++){

curr\_res += fixed\_el\*received\_vec2[j];

}

}

if(rank + 1 <= rest){

MPI\_Recv(received\_vec1, (int)sub\_size, MPI\_INT, 0, rank, MPI\_COMM\_WORLD, MPI\_STATUS\_IGNORE);

MPI\_Recv(received\_vec2, VEC\_SIZE, MPI\_INT, 0, rank, MPI\_COMM\_WORLD, MPI\_STATUS\_IGNORE);

for(size\_t i = 0; i < sub\_size; i++){

int fixed\_el = received\_vec1[i];

for(size\_t j = 0; j < VECTOR\_SIZE; j++){

curr\_res += fixed\_el\*received\_vec2[j];

}

}

}

MPI\_Send(&curr\_res, 1, MPI\_INT, 0, rank, MPI\_COMM\_WORLD);

}

delete []received\_vec1;

delete []received\_vec2;

MPI\_Finalize();

return 0;

}

# **Приложение 3. Листинг программы вариант 3**

#include <iostream>

#include "mpi.h"

#include <random>

#include <cassert>

using namespace std;

//#define VECTOR\_SIZE 200000

#define VECTOR\_SIZE 150000

inline int generateRandomInRange(int a, int b) {

random\_device rd;

uniform\_int\_distribution gen(a, b);

return gen(rd);

}

void fill\_vector\_values(int \*vec) {

for (size\_t i = 0; i < VECTOR\_SIZE; i++) {

vec[i] = generateRandomInRange(1, 100);

}

}

std::string print\_vec(int \*vec, int n) {

std::string buffer;

for (size\_t i = 0; i < n; i++) {

buffer.append(to\_string(vec[i]));

buffer.append(", ");

}

return buffer;

}

int calculate(const int \*vec1, const int \*vec2) {

int res = 0;

for (int i = 0; i < VECTOR\_SIZE; i++) {

for (int j = 0; j < VECTOR\_SIZE; j++) {

res += vec1[i] \* vec2[j];

}

}

return res;

}

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

int rank, size;

MPI\_Init(&argc, &argv);

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

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

size\_t sub\_size = VECTOR\_SIZE / size;

size\_t rest = VECTOR\_SIZE % size;

int result = 0;

int \*vec1 = NULL;

auto \*received\_vec1 = new int[sub\_size];

auto \*vec2 = new int[VECTOR\_SIZE];

if (rank == 0) {

vec1 = new int[VECTOR\_SIZE];

fill\_vector\_values(vec1);

fill\_vector\_values(vec2);

// std::cout << "generated data : vec1 : {" << print\_vec(vec1, VECTOR\_SIZE) << "}, vec2 : {"

// << print\_vec(vec2, VECTOR\_SIZE) << "}" << std::endl;

}

// send parts vec1 to processes(includes 0 root process)

MPI\_Scatter(vec1, sub\_size, MPI\_INT, received\_vec1, sub\_size, MPI\_INT, 0, MPI\_COMM\_WORLD);

//send vec2 to all processes

MPI\_Bcast(vec2, (int) VECTOR\_SIZE, MPI\_INT, 0, MPI\_COMM\_WORLD);

// std::cout << "rank is : " << rank << ", left : {" << print\_vec(received\_vec1, sub\_size) << "}, right : {"

// << print\_vec(vec2, VECTOR\_SIZE) << "}" << std::endl;

// calculate part of vec1 in some process

int curr\_res = 0;

for (size\_t i = 0; i < sub\_size; i++) {

for (size\_t j = 0; j < VECTOR\_SIZE; j++) {

curr\_res += received\_vec1[i] \* vec2[j];

}

}

//summ all calculations

MPI\_Reduce(&curr\_res, &result, 1, MPI\_INT, MPI\_SUM, 0, MPI\_COMM\_WORLD);

if (rank == 0) {

std::cout << "Result " << result << endl;

//std::cout << "calculate " << calculate(vec1, vec2) << endl;

delete[]vec1;

}

delete[] received\_vec1;

delete[] vec2;

MPI\_Finalize();

return 0;

}

# **Приложение 4. Один из выводов сбора времени**

--- 86.90033507347107 seconds ---

Starting with n: 2

--- 38.4712233543396 seconds ---

--- 43.41948199272156 seconds ---

===========================

===========================

===========================

Starting with n: 4

--- 21.284065008163452 seconds ---

--- 23.857536792755127 seconds ---

===========================

===========================

===========================

Starting with n: 8

--- 12.917084455490112 seconds ---

--- 14.211663722991943 seconds ---

===========================

===========================

===========================

Starting with n: 16

--- 13.469727516174316 seconds ---

--- 14.623409271240234 seconds ---

===========================

===========================

===========================

Starting with n: 24

--- 11.256603002548218 seconds ---

--- 11.98368215560913 seconds ---

===========================

===========================

===========================