МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего профессионального образования

«Вятский государственный университет»

(ФГБОУ ВО «ВятГУ»)

Факультет автоматики и вычислительной техники

Кафедра электронных вычислительных машин

Параллельное программирование

Многопоточная реализация вычислительно сложного алгоритма с применением библиотеки MPICH

Вариант 7

Выполнил студент группы ИВТ-31 _____/Кудяшев Я.Ю./

Проверил преподаватель_____/Долженкова М.Л./

Киров 2022

1. Задание

Познакомиться со стандартом MPI, получить навыки реализации многопоточных SPMD-приложений с применением MPI.

Этапы работы:

1) Изучить основные принципы создания приложений с использованием библиотеки MPI, рассмотреть базовый набор директив компилятора
2) Выделить в полученной в ходе выполнения первой лабораторной работы реализации алгоритма фрагменты кода, выполнение которых может быть разнесено на несколько процессорных ядер
3) Реализовать многопоточную версию алгоритма с помощью языка C++ и библиотеки MPI, используя при этом необходимые примитивы синхронизации
4) Показать корректность полученной реализации путём осуществления на построенном в ходе первой лабораторной работы наборе тестов
5) Провести доказательную оценку эффективности MPI-реализации алгоритма

2. Метод распараллеливания алгоритма

В качестве областей участков для распараллеливания при помощи MPI были выбраны те же участки, что и при простом распараллеливании. Это было сделано для наиболее точного сравнения результатов тестов.

Из исследований алгоритма для перемножения полиномов с помощью быстрого преобразованья Фурье удалось выяснить, что время в большей степени зависит от количества входных векторов, нежели от размерности. Было принято решение переложить работу по умножению каждой пары векторов на потоки. Помимо этого, для более эффективного использования потоков, было сделано одно нововведение в алгоритм распараллеливания: пока идёт цикл перемножения векторов, главный поток подготавливает новый массив данных для следующего теста. Данный подход позволяет уменьшить общее время выполнения программы.

Таким образом, после перемножения пары векторов, каждое последующее умножение будет происходить уже с полученным в результате предыдущего умножения вектором. В случае, когда количество входных векторов равно 2, параллельно будет выполняться ДПФ для каждого входного вектора.

3. Программная реализация

Листинг программной MPI-реализации алгоритма приведен в приложении А.

4. Тестирование

Тестирование проводилось на ЭВМ под управлением 64-разрядной ОС Windows 10, с 8 ГБ оперативной памяти, с процессором Intel Core i5-8250U с частотой 1.80 ГГц (8 логических и 4 физических ядра).

Результаты тестирования и сравнения с параллельной реализацией и OpenMP приведены в таблице 1.

Таблица 1 – Результаты тестирования

Исходные данные (кол-во полиномов, размерность)	Последовательная реализация, мс	Параллельная реализация, мс	OpenMP-реализация, мс	MPI-реализация, мс	Ускорение
2, 100000	210 мс	184 мс	208 мс	172 мс	1,20
4, 100000	1183 мс	581 мс	590 мс	490 мс	1,20
6, 100000	5485 мс	1046 мс	1217 мс	1023 мс	1,19
8, 100000	24990 мс	4133 мс	4308 мс	3929 мс	1,09
2, 1000000	1493 мс	1033 мс	1175 мс	1008 мс	1,16
4, 1000000	10734 мс	4008 мс	4101 мс	3511 мс	1,17
6, 1000000	50241 мс	9558 мс	10171 мс	8422 мс	1,21
8, 1000000	259036 мс	40712 мс	40150 мс	36258 мс	1,11
2, 10000000	26759 мс	19438 мс	21103 мс	19005 мс	1,11
8, 10000	2640 мс	419 мс	502	404 мс	1,24
				Среднее	1,17
				Максимальное	1,24
				Минимальное	1,09

Исходя из результатов тестирования можно сказать, что в реализации данного алгоритма MPI дает небольшой прирост в скорости при сравнении как с параллельной реализацией, так и с OpenMP. Все тесты выполняются немного быстрее.

5. Вывод

В ходе выполнения лабораторной работы была реализована MPI-версия алгоритма перемножения полиномов с помощью быстрого преобразованья Фурье на языке C++. MPI-версия алгоритма оказалась наиболее эффективной среди простого распараллеливания и OpenMP-версии. Благодаря задействованию главного потока в подготовке данных для следующих тестов, удалось сэкономить около 30 сек общего времени выполнения программы.

Приложение А

(обязательное)

Листинг программы

```
#include <mpi.h>
#include <stdio.h>
#include <iostream>
#include <vector>
#include <complex>
#include <chrono>
#include <fstream>


using namespace std;

typedef complex<double> base;

	//result vector
int counter = 1;
int thread_counter = 8;

int rev(int num, int lg_n) {			//begining of good realisation
	int res = 0;
	for (int i = 0; i < lg_n; ++i)
		if (num & (1 << i))
			res |= 1 << (lg_n - 1 - i);
	return res;
}

void good_realisation(vector<base>& a, bool invert) { //БПФ и обратное БПФ
	int n = (int)a.size();
	int lg_n = 0;
	while ((1 << lg_n) < n)  ++lg_n;

	for (int i = 0; i < n; ++i)
		if (i < rev(i, lg_n))
			swap(a[i], a[rev(i, lg_n)]);

	for (int len = 2; len <= n; len <<= 1) {
		double ang = 2 * 3.14 / len * (invert ? -1 : 1);
		base wlen(cos(ang), sin(ang));
		for (int i = 0; i < n; i += len) {
			base w(1);
			for (int j = 0; j < len / 2; ++j) {
				base u = a[i + j], v = a[i + j + len / 2] * w;
				a[i + j] = u + v;
				a[i + j + len / 2] = u - v;
				w *= wlen;
			}
		}
	}
	if (invert)
		for (int i = 0; i < n; ++i)
			a[i] /= n;
}

void good_multiplication(const vector<int>& a, const vector<int>& b, vector<int>& res,
int number) {		//multiplication of two vectors
	vector<base> fa(a.begin(), a.end()), fb(b.begin(), b.end());
	int n = 1;
	while (n < max(a.size(), b.size()))  n <<= 1;
	n <<= 1;
```

```
	fa.resize(n), fb.resize(n);
	if (number == 2) {

		{
			good_realisation(fa, false);

			{

				good_realisation(fb, false);
			}
		}
	}
	else {
		good_realisation(fa, false);

		good_realisation(fb, false);
	}

	for (int i = 0; i < n; ++i)
		fa[i] *= fb[i];
	good_realisation(fa, true);

	res.resize(n);
	for (int i = 0; i < n; ++i)
		res[i] = int(fa[i].real() + 0.5);
}

void fill_from_file(string path, int number_of_vectors, int size_of_vectors, vector<int>
information[]) {	//reading data from the file
	ifstream vectorr("C:\\Programming\\Parallel programming\\Lab 1\\" + path);

	for (int i = 0; i < number_of_vectors; i++) {
		information[i].resize(size_of_vectors);
	}

	for (int i = 0; i < number_of_vectors; i++) {
		for (int j = 0; j < size_of_vectors; j++) {
			vectorr >> information[i].at(j);
		}
	}
	vectorr.close();
}

void enter(string first, int number, int size, string path, vector<int> information[]) {
	//input

	std::cout << "\n" + first + " test is running\n";
	fill_from_file(path, number, size, information);	//number, size


	//counter = 1;
}


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

	vector<int> information[30];	//data vector
	//vector<int> result(10000000);
	MPI_Init(NULL, NULL);
	int world_rank;
	int world_size;
	// Get the rank of the process

	MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);
```

```
	MPI_Status status;
	MPI_Request request;

	if (world_rank == 1) {
		/*Data for first test*/
		MPI_Recv(&information[0], 1000000, MPI_INT, 0, 0, MPI_COMM_WORLD,
MPI_STATUS_IGNORE);

		good_multiplication(information[0], information[1], information[0], 2);
		MPI_Send(&information[0], 1000000, MPI_INT, 3, 2, MPI_COMM_WORLD);

		/*Data for second test*/
		MPI_Recv(&information[0], 1000000, MPI_INT, 0, 4, MPI_COMM_WORLD,
MPI_STATUS_IGNORE);

		good_multiplication(information[0], information[1], information[0], 2);
		MPI_Send(&information[0], 1000000, MPI_INT, 3, 6, MPI_COMM_WORLD);

		/*Data for third test*/
		MPI_Recv(&information[0], 1000000, MPI_INT, 0, 8, MPI_COMM_WORLD,
MPI_STATUS_IGNORE);

		good_multiplication(information[0], information[1], information[0], 2);
		MPI_Send(&information[0], 1000000, MPI_INT, 6, 10, MPI_COMM_WORLD);

		/*Data for fourth test*/
		MPI_Recv(&information[0], 1000000, MPI_INT, 0, 12, MPI_COMM_WORLD,
MPI_STATUS_IGNORE);

		good_multiplication(information[0], information[1], information[0], 2);
		MPI_Send(&information[0], 1000000, MPI_INT, 6, 14, MPI_COMM_WORLD);

		/*Data for fith test*/
		MPI_Recv(&information[0], 1000000, MPI_INT, 0, 16, MPI_COMM_WORLD,
MPI_STATUS_IGNORE);

		good_multiplication(information[0], information[1], information[0], 2);
		MPI_Send(&information[0], 1000000, MPI_INT, 6, 18, MPI_COMM_WORLD);

		/*Data for sixth test*/
		MPI_Recv(&information[0], 1000000, MPI_INT, 0, 20, MPI_COMM_WORLD,
MPI_STATUS_IGNORE);

		good_multiplication(information[0], information[1], information[0], 2);
		MPI_Send(&information[0], 1000000, MPI_INT, 6, 22, MPI_COMM_WORLD);

		/*Data for seventh test*/
		MPI_Recv(&information[0], 1000000, MPI_INT, 0, 24, MPI_COMM_WORLD,
MPI_STATUS_IGNORE);

		good_multiplication(information[0], information[1], information[0], 2);
		MPI_Send(&information[0], 1000000, MPI_INT, 6, 26, MPI_COMM_WORLD);

		/*Data for eighth test*/
		MPI_Recv(&information[0], 1000000, MPI_INT, 0, 28, MPI_COMM_WORLD,
MPI_STATUS_IGNORE);

		good_multiplication(information[0], information[1], information[0], 2);
		MPI_Send(&information[0], 1000000, MPI_INT, 6, 30, MPI_COMM_WORLD);

	}

	if (world_rank == 2) {

		/*Data for first test*/
```

```
            MPI_Recv(&information[2], 1000000, MPI_INT, 0, 0, MPI_COMM_WORLD,
MPI_STATUS_IGNORE);

            good_multiplication(information[0], information[1], information[0], 2);
            MPI_Send(&information[2], 1000000, MPI_INT, 3, 1, MPI_COMM_WORLD);

            /*Data for second test*/
            MPI_Recv(&information[2], 1000000, MPI_INT, 0, 5, MPI_COMM_WORLD,
MPI_STATUS_IGNORE);

            good_multiplication(information[0], information[1], information[0], 2);
            MPI_Send(&information[2], 1000000, MPI_INT, 3, 6, MPI_COMM_WORLD);

            /*Data for third test*/
            MPI_Recv(&information[2], 1000000, MPI_INT, 0, 9, MPI_COMM_WORLD,
MPI_STATUS_IGNORE);

            good_multiplication(information[0], information[1], information[0], 2);
            MPI_Send(&information[2], 1000000, MPI_INT, 6, 10, MPI_COMM_WORLD);

            /*Data for fourth test*/
            MPI_Recv(&information[2], 1000000, MPI_INT, 0, 13, MPI_COMM_WORLD,
MPI_STATUS_IGNORE);

            good_multiplication(information[0], information[1], information[0], 2);
            MPI_Send(&information[2], 1000000, MPI_INT, 6, 14, MPI_COMM_WORLD);

            /*Data for fith test*/
            MPI_Recv(&information[2], 1000000, MPI_INT, 0, 17, MPI_COMM_WORLD,
MPI_STATUS_IGNORE);

            good_multiplication(information[0], information[1], information[0], 2);
            MPI_Send(&information[2], 1000000, MPI_INT, 6, 18, MPI_COMM_WORLD);

            /*Data for sixth test*/
            MPI_Recv(&information[2], 1000000, MPI_INT, 0, 21, MPI_COMM_WORLD,
MPI_STATUS_IGNORE);

            good_multiplication(information[0], information[1], information[0], 2);
            MPI_Send(&information[2], 1000000, MPI_INT, 6, 22, MPI_COMM_WORLD);

            /*Data for seventh test*/
            MPI_Recv(&information[2], 1000000, MPI_INT, 0, 25, MPI_COMM_WORLD,
MPI_STATUS_IGNORE);

            good_multiplication(information[0], information[1], information[0], 2);
            MPI_Send(&information[2], 1000000, MPI_INT, 6, 26, MPI_COMM_WORLD);

            /*Data for eighth test*/
            MPI_Recv(&information[2], 1000000, MPI_INT, 0, 29, MPI_COMM_WORLD,
MPI_STATUS_IGNORE);

            good_multiplication(information[0], information[1], information[0], 2);
            MPI_Send(&information[2], 1000000, MPI_INT, 6, 30, MPI_COMM_WORLD);

      }

      if (world_rank == 3) {

            /*Data for first test*/
            MPI_Recv(&information[0], 1000000, MPI_INT, 1, 2, MPI_COMM_WORLD,
MPI_STATUS_IGNORE);
            MPI_Recv(&information[2], 1000000, MPI_INT, 2, 3, MPI_COMM_WORLD,
MPI_STATUS_IGNORE);
```

```
            good_multiplication(information[0], information[2], information[0], 2);
            MPI_Send(&information[2], 1000000, MPI_INT, 0, 4, MPI_COMM_WORLD);

            /*Data for third test*/
            MPI_Recv(&information[0], 1000000, MPI_INT, 1, 6, MPI_COMM_WORLD,
MPI_STATUS_IGNORE);
            MPI_Recv(&information[2], 1000000, MPI_INT, 2, 7, MPI_COMM_WORLD,
MPI_STATUS_IGNORE);

            good_multiplication(information[0], information[2], information[0], 2);
            MPI_Send(&information[2], 1000000, MPI_INT, 0, 8, MPI_COMM_WORLD);

      }

      if (world_rank == 4) {

            /*Data for third test*/
            MPI_Recv(&information[0], 1000000, MPI_INT, 1, 5, MPI_COMM_WORLD,
MPI_STATUS_IGNORE);

            good_multiplication(information[2], information[3], information[2], 2);
            MPI_Send(&information[2], 1000000, MPI_INT, 3, 7, MPI_COMM_WORLD);

            /*Data for fith test*/
            MPI_Recv(&information[0], 1000000, MPI_INT, 1, 4, MPI_COMM_WORLD,
MPI_STATUS_IGNORE);

            good_multiplication(information[0], information[1], information[0], 2);
            MPI_Send(&information[0], 1000000, MPI_INT, 3, 6, MPI_COMM_WORLD);

            /*Data for sixth test*/
            MPI_Recv(&information[0], 1000000, MPI_INT, 1, 4, MPI_COMM_WORLD,
MPI_STATUS_IGNORE);

            good_multiplication(information[0], information[1], information[0], 2);
            MPI_Send(&information[0], 1000000, MPI_INT, 3, 6, MPI_COMM_WORLD);

            /*Data for third test*/
            MPI_Recv(&information[0], 1000000, MPI_INT, 1, 4, MPI_COMM_WORLD,
MPI_STATUS_IGNORE);

            good_multiplication(information[0], information[1], information[0], 2);
            MPI_Send(&information[0], 1000000, MPI_INT, 3, 6, MPI_COMM_WORLD);

            /*Data for fourth test*/
            MPI_Recv(&information[0], 1000000, MPI_INT, 1, 4, MPI_COMM_WORLD,
MPI_STATUS_IGNORE);

            good_multiplication(information[0], information[1], information[0], 2);
            MPI_Send(&information[0], 1000000, MPI_INT, 3, 6, MPI_COMM_WORLD);

      }

      if (world_rank == 5) {

            /*Data for third test*/
            MPI_Recv(&information[0], 1000000, MPI_INT, 1, 5, MPI_COMM_WORLD,
MPI_STATUS_IGNORE);

            good_multiplication(information[2], information[3], information[2], 2);
            MPI_Send(&information[2], 1000000, MPI_INT, 3, 7, MPI_COMM_WORLD);

            /*Data for fourth test*/
            MPI_Recv(&information[0], 1000000, MPI_INT, 1, 5, MPI_COMM_WORLD,
MPI_STATUS_IGNORE);
```

```
            good_multiplication(information[2], information[3], information[2], 2);
            MPI_Send(&information[2], 1000000, MPI_INT, 3, 7, MPI_COMM_WORLD);

            /*Data for fith test*/
            MPI_Recv(&information[0], 1000000, MPI_INT, 1, 5, MPI_COMM_WORLD,
MPI_STATUS_IGNORE);

            good_multiplication(information[2], information[3], information[2], 2);
            MPI_Send(&information[2], 1000000, MPI_INT, 3, 7, MPI_COMM_WORLD);

            /*Data for sixth test*/
            MPI_Recv(&information[0], 1000000, MPI_INT, 1, 5, MPI_COMM_WORLD,
MPI_STATUS_IGNORE);

            good_multiplication(information[2], information[3], information[2], 2);
            MPI_Send(&information[2], 1000000, MPI_INT, 3, 7, MPI_COMM_WORLD);

      }


      if (world_rank == 6) {

            /*Data for third test*/
            MPI_Recv(&information[0], 1000000, MPI_INT, 1, 5, MPI_COMM_WORLD,
MPI_STATUS_IGNORE);

            good_multiplication(information[2], information[3], information[2], 2);
            MPI_Send(&information[2], 1000000, MPI_INT, 3, 7, MPI_COMM_WORLD);

            /*Data for third test*/
            MPI_Recv(&information[0], 1000000, MPI_INT, 1, 5, MPI_COMM_WORLD,
MPI_STATUS_IGNORE);

            good_multiplication(information[2], information[3], information[2], 2);
            MPI_Send(&information[2], 1000000, MPI_INT, 3, 7, MPI_COMM_WORLD);

            /*Data for fourth test*/
            MPI_Recv(&information[0], 1000000, MPI_INT, 1, 5, MPI_COMM_WORLD,
MPI_STATUS_IGNORE);

            good_multiplication(information[2], information[3], information[2], 2);
            MPI_Send(&information[2], 1000000, MPI_INT, 3, 7, MPI_COMM_WORLD);

            /*Data for fith test*/
            MPI_Recv(&information[0], 1000000, MPI_INT, 1, 5, MPI_COMM_WORLD,
MPI_STATUS_IGNORE);

            good_multiplication(information[2], information[3], information[2], 2);
            MPI_Send(&information[2], 1000000, MPI_INT, 3, 7, MPI_COMM_WORLD);

            /*Data for sixth test*/
            MPI_Recv(&information[0], 1000000, MPI_INT, 1, 5, MPI_COMM_WORLD,
MPI_STATUS_IGNORE);

            good_multiplication(information[2], information[3], information[2], 2);
            MPI_Send(&information[2], 1000000, MPI_INT, 3, 7, MPI_COMM_WORLD);

      }

      if (world_rank == 7) {

            /*Data for third test*/
            MPI_Recv(&information[0], 1000000, MPI_INT, 1, 5, MPI_COMM_WORLD,
MPI_STATUS_IGNORE);
```

```
            good_multiplication(information[2], information[3], information[2], 2);
            MPI_Send(&information[2], 1000000, MPI_INT, 3, 7, MPI_COMM_WORLD);

            /*Data for fourth test*/
            MPI_Recv(&information[0], 1000000, MPI_INT, 1, 5, MPI_COMM_WORLD,
MPI_STATUS_IGNORE);

            good_multiplication(information[2], information[3], information[2], 2);
            MPI_Send(&information[2], 1000000, MPI_INT, 3, 7, MPI_COMM_WORLD);

            /*Data for fith test*/
            MPI_Recv(&information[0], 1000000, MPI_INT, 1, 5, MPI_COMM_WORLD,
MPI_STATUS_IGNORE);

            good_multiplication(information[2], information[3], information[2], 2);
            MPI_Send(&information[2], 1000000, MPI_INT, 3, 7, MPI_COMM_WORLD);

            /*Data for sixth test*/
            MPI_Recv(&information[0], 1000000, MPI_INT, 1, 5, MPI_COMM_WORLD,
MPI_STATUS_IGNORE);

            good_multiplication(information[2], information[3], information[2], 2);
            MPI_Send(&information[2], 1000000, MPI_INT, 3, 7, MPI_COMM_WORLD);
      }


      /*0 - Master thread, 1-2 and 4-5 - threads for first multiplication, 3 - last
thread, 6-7 - threds for second multiplication(6 thread for 6-size vectors)*/

      if(world_rank==0){ /*Перемножение полиномов с помощью быстрого преобразования
Фурье.*/

            /*First test*/
            cout << "4 vectors of size 100000";
            enter("The second", 4, 100000, "int_0-100 4_100000.txt",information);
            cout << "Good algorithm: ";
            unsigned int start_time = clock();
            MPI_Send(&information[0],100000, MPI_INT, 1, 0, MPI_COMM_WORLD);
            MPI_Send(&information[2], 100000, MPI_INT, 2, 1, MPI_COMM_WORLD);

            cout << "4 vectors of size 1000000";
            enter("The fifth", 4, 100000, "int_0-100 4_1000000.txt",information);
            MPI_Recv(&information[2], 100000, MPI_INT, 3, 4, MPI_COMM_WORLD,
MPI_STATUS_IGNORE);
            unsigned int end_time = clock();
            unsigned int search_time = end_time - start_time;
            std::cout << search_time << " mc\n";
            cout << '\n';

            /*Second test*/
       start_time = clock();
            MPI_Send(&information[0], 1000000, MPI_INT, 1, 5, MPI_COMM_WORLD);
            MPI_Send(&information[2], 1000000, MPI_INT, 2, 6, MPI_COMM_WORLD);

            cout << "6 vectors of size 100000";
            enter("The seventh", 6, 100000, "int_0-100 6_100000.txt", information);

            MPI_Recv(&information[2], 1000000, MPI_INT, 3, 8, MPI_COMM_WORLD,
MPI_STATUS_IGNORE);
       end_time = clock();
       search_time = end_time - start_time;
            std::cout << search_time << " mc\n";
            cout << '\n';
```

```
            /*Third test*/
            start_time = clock();
            MPI_Send(&information[0], 100000, MPI_INT, 1, 9, MPI_COMM_WORLD);
            MPI_Send(&information[2], 100000, MPI_INT, 2, 10, MPI_COMM_WORLD);
            MPI_Send(&information[4], 100000, MPI_INT, 3, 11, MPI_COMM_WORLD);
            MPI_Send(&information[6], 100000, MPI_INT, 4, 12, MPI_COMM_WORLD);

            cout << "6 vectors of size 1000000";
            enter("The eighth", 6, 1000000, "int_0-100 6_1000000.txt", information);


            MPI_Recv(&information[2], 100000, MPI_INT, 3, 13, MPI_COMM_WORLD,
MPI_STATUS_IGNORE);
            end_time = clock();
            search_time = end_time - start_time;
            cout << search_time << " mc\n";
            cout << '\n';

            /*Fourth test*/
            start_time = clock();
            MPI_Send(&information[0], 1000000, MPI_INT, 1, 13, MPI_COMM_WORLD);
            MPI_Send(&information[2], 1000000, MPI_INT, 2, 14, MPI_COMM_WORLD);
            MPI_Send(&information[4], 1000000, MPI_INT, 1, 13, MPI_COMM_WORLD);

            cout << "8 vectors of size 100000";
            enter("The third", 8, 100000, "int_0-100 8_100000.txt", information);
            MPI_Recv(&information[2], 1000000, MPI_INT, 3, 16, MPI_COMM_WORLD,
MPI_STATUS_IGNORE);
            end_time = clock();
            search_time = end_time - start_time;
            std::cout << search_time << " mc\n";
            cout << '\n';

            /*Fifth test*/
            start_time = clock();
            MPI_Send(&information[0], 1000000, MPI_INT, 1, 17, MPI_COMM_WORLD);
            MPI_Send(&information[2], 1000000, MPI_INT, 2, 18, MPI_COMM_WORLD);
            MPI_Send(&information[0], 1000000, MPI_INT, 1, 17, MPI_COMM_WORLD);
            MPI_Send(&information[2], 1000000, MPI_INT, 2, 18, MPI_COMM_WORLD);

            cout << "8 vectors of size 100000";
            enter("The third", 8, 100000, "int_0-100 8_100000.txt", information);
            MPI_Recv(&information[2], 1000000, MPI_INT, 3, 20, MPI_COMM_WORLD,
MPI_STATUS_IGNORE);
            end_time = clock();
            search_time = end_time - start_time;
            std::cout << search_time << " mc\n";
            cout << '\n';

            /*Sixth test*/
            start_time = clock();
            MPI_Send(&information[0], 1000000, MPI_INT, 1, 21, MPI_COMM_WORLD);
            MPI_Send(&information[2], 1000000, MPI_INT, 2, 22, MPI_COMM_WORLD);
            MPI_Send(&information[0], 1000000, MPI_INT, 1, 17, MPI_COMM_WORLD);
            MPI_Send(&information[2], 1000000, MPI_INT, 2, 18, MPI_COMM_WORLD);

            cout << "8 vectors of size 1000000";
            enter("The sixth", 8, 1000000, "int_0-100 8_1000000.txt", information);
            MPI_Recv(&information[2], 1000000, MPI_INT, 3, 24, MPI_COMM_WORLD,
MPI_STATUS_IGNORE);
            end_time = clock();
            search_time = end_time - start_time;
            std::cout << search_time << " mc\n";
            cout << '\n';
```

```
            /*seventh test*/
            start_time = clock();
            MPI_Send(&information[0], 1000000, MPI_INT, 1, 25, MPI_COMM_WORLD);
            MPI_Send(&information[2], 1000000, MPI_INT, 2, 16, MPI_COMM_WORLD);
            MPI_Send(&information[0], 1000000, MPI_INT, 1, 17, MPI_COMM_WORLD);
            MPI_Send(&information[2], 1000000, MPI_INT, 2, 18, MPI_COMM_WORLD);

            cout << "8 vectors of size 10000";
            enter("The nineth", 8, 10000, "int_0-100 8_10000.txt",information);
            MPI_Recv(&information[2], 1000000, MPI_INT, 3, 18, MPI_COMM_WORLD,
MPI_STATUS_IGNORE);
            end_time = clock();
            search_time = end_time - start_time;
            std::cout << search_time << " mc\n";
            cout << '\n';

            /*Eighth test*/
            start_time = clock();
            MPI_Send(&information[0], 1000000, MPI_INT, 1, 17, MPI_COMM_WORLD);
            MPI_Send(&information[0], 1000000, MPI_INT, 1, 17, MPI_COMM_WORLD);
            MPI_Send(&information[2], 1000000, MPI_INT, 2, 18, MPI_COMM_WORLD);
            MPI_Send(&information[2], 1000000, MPI_INT, 2, 18, MPI_COMM_WORLD);

            cout << "2 vectors of size 10000000";
            enter("The tenth", 2, 10000000, "int_0-100 2_10000000.txt",information);
            MPI_Recv(&information[2], 1000000, MPI_INT, 3, 20, MPI_COMM_WORLD,
MPI_STATUS_IGNORE);
            end_time = clock();
            search_time = end_time - start_time;
            std::cout << search_time << " mc\n";
            cout << '\n';

            /*Nineth test*/
            start_time = clock();
            MPI_Send(&information[0], 1000000, MPI_INT, 1, 17, MPI_COMM_WORLD);
            MPI_Send(&information[2], 1000000, MPI_INT, 2, 18, MPI_COMM_WORLD);

            cout << "2 vectors of size 1000000";
            enter("The fourth", 2, 1000000, "int_0-100 2_1000000.txt",information);
            MPI_Recv(&information[2], 1000000, MPI_INT, 3, 20, MPI_COMM_WORLD,
MPI_STATUS_IGNORE);
            end_time = clock();
            search_time = end_time - start_time;
            std::cout << search_time << " mc\n";
            cout << '\n';

            /*Tenth test*/
            start_time = clock();
            MPI_Send(&information[0], 1000000, MPI_INT, 1, 17, MPI_COMM_WORLD);
            MPI_Send(&information[2], 1000000, MPI_INT, 2, 18, MPI_COMM_WORLD);

            MPI_Recv(&information[2], 1000000, MPI_INT, 3, 20, MPI_COMM_WORLD,
MPI_STATUS_IGNORE);
            end_time = clock();
            search_time = end_time - start_time;
            std::cout << search_time << " mc\n";
            cout << '\n';
      }

      MPI_Finalize();

}
```