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 С++ 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
6. 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.

1. Программная реализация

Листинг программной MPI-реализации алгоритма приведен в приложении А.

1. Тестирование

Тестирование проводилось на ЭВМ под управлением 64-разрядной OC 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.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 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 |

Таблица 1 – Результаты тестирования

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.

1. Вывод

В ходе выполнения лабораторной работы была реализована MPI-версия алгоритма перемножения полиномов с помощью быстрого преобразованья Фурье на языке С++. 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();

}