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

Вариант 7

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

Проверил преподаватель_____/Долженкова М.Л./

Киров 2022

1. Задание

Изучить средства работы с потоками операционной системы, получить навыки реализации многопоточных приложений.

Этапы работы:

1) Выделить в полученной в ходе первой лабораторной работы реализации перемножения полиномов с помощью быстрого преобразования Фурье фрагменты кода, выполнение которых может быть распределено на несколько процессорных ядер
2) Реализовать многопоточную версию алгоритма с помощью языка C++ и потоков стандартной библиотеки C++, используя при этом необходимые примитивы синхронизации
3) Показать корректность полученной реализации, запустив её на наборе тестов, построенных в ходе первой лабораторной работы
4) Реализовать обе версии алгоритма с помощью языка C++
5) Провести доказательную оценку эффективности многопоточной реализации алгоритма

2. Метод распараллеливания алгоритма

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.

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 реализации алгоритма при помощи потоков 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 последовательной реализацией приведены в таблице 1.

Таблица 1 – Результаты тестирования

Исходные данные (кол-во полиномов, размерность)	Последовательная реализация, мс	Параллельная реализация, мс	Ускорение
2, 100000	210 мс	184 мс	1,14
4, 100000	1183 мс	581 мс	2,04
6, 100000	5485 мс	1046 мс	5,24
8, 100000	24990 мс	4133 мс	6,04
2, 1000000	1493 мс	1033 мс	1,45
4, 1000000	10734 мс	4008 мс	2,68
6, 1000000	50241 мс	9558 мс	5,26
8, 1000000	259036 мс	40712 мс	6,36
2, 10000000	26759 мс	19438 мс	1,38
8, 10000	2640 мс	419 мс	6,30
		Среднее	3,79
		Максимальное	6,36
		Минимальное	1,14

В некоторых случаях ускорение зависит от количества потоков, которое используется при вычислении: так в перемножении 2-х полиномов задействовано лишь 2 потока. Также распараллеливание более эффективно при распределении потоков под умножение пары полиномов нежели под нахождение ДПФ.

5. Вывод

В ходе выполнения лабораторной работы была реализована многопоточная версия алгоритма перемножения полиномов с помощью быстрого преобразованья Фурье на языке C++. Многопоточная версия оказалась намного эффективнее линейной в случае перемножения большого количества полиномов. Перемножение пары полиномов в потоке оказалось намного эффективнее чем распределение потоков на нахождение ДПФ.

Приложение А

(обязательное)

Листинг программы

```
#include <iostream>
#include <chrono>
#include <vector>
#include <complex>
#include <fstream>
#include <thread>

using namespace std;

typedef complex<double> base;

vector<int> information[30];      //data vector
vector<int> result(10000000);     //result vector
int counter = 1;
int thread_counter = 8;
std::thread threads[8];    //8 threads were created


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) {
	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) {
			threads[0] = thread(good_realisation, ref(fa), false);
			threads[1] = thread(good_realisation, ref(fb), false);

			threads[0].join();
			threads[1].join();
		}
		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) {
	//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) {		//input

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

	std::cout << "Good algorithm: ";
	unsigned int start_time = clock();


	switch(number) {
	case 2:
		good_multiplication(information[0], information[1],information[0], 2);

		break;

	case 4:
		for (int i = 0, j = 0; j < 2; i + 2, j++) {
			threads[j] = thread(good_multiplication, ref(information[i]),
ref(information[i + 1]), ref(information[i]), 1);
		}

		for (int i = 0; i < 2; i++) {	//Waiting for the end of the threads
			threads[i].join();
		}
```

```
            threads[0] = thread(good_multiplication, ref(information[0]),
ref(information[2]), ref(information[0]), 1);
            threads[0].join();

            break;

      case 6:
            for (int i = 0, j = 0; j < 2; i + 2, j++) {
                  threads[j] = thread(good_multiplication, ref(information[i]),
ref(information[i + 1]), ref(information[i]), 1);
            }

            for (int i = 0; i < 2; i++) {     //Waiting for the end of the threads
                  threads[i].join();
            }

                  threads[0] = thread(good_multiplication, ref(information[0]),
ref(information[2]), ref(information[0]), 1);
                  threads[1] = thread(good_multiplication, ref(information[4]),
ref(information[5]), ref(information[4]), 1);

            for (int i = 0; i < 2; i++) {     //Waiting for the end of the threads
                  threads[i].join();
            }

            threads[0] = thread(good_multiplication, ref(information[0]),
ref(information[4]), ref(information[0]), 1);

            threads[0].join();

            break;

      case 8:
            for (int i = 0, j = 0; j < 4; j++, i + 2) {
                  threads[j] = thread(good_multiplication, ref(information[i]),
ref(information[i + 1]), ref(information[i]), 1);
            }


            for (int i = 0; i < 4; i++) {     //Waiting for the end of the threads
                  threads[i].join();
            }

            for (int i = 0, j = 0; j < 2; j++, i + 4) {
                  threads[j] = thread(good_multiplication, ref(information[i]),
ref(information[i + 2]), ref(information[i]), 1);
            }


            for (int i = 0; i < 2; i++) {     //Waiting for the end of the threads
                  threads[i].join();
            }

            threads[0] = thread(good_multiplication, ref(information[0]),
ref(information[4]), ref(information[0]), 1);
            threads[0].join();

            break;

      default:
            std::cout << "Wrong number of vectors";
      }


      unsigned int end_time = clock();
```

```
        unsigned int search_time = end_time - start_time;
        std::cout << search_time << " mc\n";
        cout << '\n';
        counter = 1;
}



int main()
{

        cout << "2 vectors of size 100000";
        enter("The first", 2, 100000, "int_0-100 2_100000.txt");

        cout << "4 vectors of size 100000";
        enter("The second", 4, 100000, "int_0-100 4_100000.txt");

        cout << "8 vectors of size 100000";
        enter("The third", 8, 100000, "int_0-100 8_100000.txt");

        cout << "2 vectors of size 1000000";
        enter("The fourth", 2, 1000000, "int_0-100 2_1000000.txt");

        cout << "4 vectors of size 1000000";
        enter("The fifth", 4, 1000000, "int_0-100 4_1000000.txt");

        cout << "8 vectors of size 1000000";
        enter("The sixth", 8, 1000000, "int_0-100 8_1000000.txt");

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

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

        cout << "8 vectors of size 10000";
        enter("The nineth", 8, 10000, "int_0-100 8_10000.txt");

        cout << "2 vectors of size 10000000";
        enter("The tenth", 2, 10000000, "int_0-100 2_10000000.txt");

}
```