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**

Исследование и последовательная реализация вычислительно сложного алгоритма

Вариант 7

Выполнил студент группы ИВТ-31 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/Кудяшев Я.Ю./

Проверил преподаватель\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/Долженкова М.Л./

Киров 2022

1. Задание

Необходимо реализовать алгоритм быстрого преобразования Фурье, применяемого к умножению полиномов, двумя способами: неоптимизированный и оптимизированный.

Этапы работы:

1. Изучить алгоритм
2. Провести доказательную оценку алгоритма по временной сложности и затратами по памяти
3. Проанализировать предыдущий алгоритм и оптимизировать его работу
4. Реализовать обе версии алгоритма с помощью языка C++
5. Построить набор тестовых примеров и провести оценку эффективности реализованных алгоритмов
6. Изучение предметной области

Быстрое преобразованье Фурье – алгоритм ускоренного вычисления дискретного преобразования Фурье, позволяющий получить результат за время, меньшее чем *O(N2)* (требуемого для прямого, поформульного вычисления).

Метод БФП основывается на свойствах комплексных корней из единицы: на том, что степени одних корней дают другие корни. Благодаря данному свойству этот метод позволяет вычислить ДПФ и ОПФ за время *O(n log n)*.

1. Оптимизация исходного алгоритма

Одним из главных недостатков первой версии алгоритма является рекурсия в явном виде. Если в стандартной версии алгоритма приходится разделять входной вектор на 2 вектора, следуя определенному алгоритму, то в оптимизированной версии алгоритма вектора заранее упорядочены определенным образом так, что необходимость в создании временных векторов отпадает.

Помимо этого, на первом уровне рекурсии элементы, младшие биты, позиции которых равны нулю, относятся к 1-му входному вектору, а младшие биты, позиции которых равны единице – ко 2-му вектору. На следующем уровне рекурсии выполняются аналогичные действия, но уже для следующих битов и так далее. Поэтому, если в определенных позициях каждого элемента инвертировать порядок битов и переупорядочить элементы в соответствии с новыми значениями, то мы получим искомый порядок. Данный подход называется поразрядно обратной перестановкой.

Данные преобразования позволяют добиться ускорения выполнения алгоритма почти в 2 раза.

1. Программная реализация

Листинг программной реализации двух версий алгоритма приведен в приложении А.

1. Тестирование

В ходе тестирования выполнялось умножение различного количества полиномов разных размерностей. Количество перемножаемых векторов и их размерности вместе с результатами тестирований приведены в таблице 1.

Таблица 1 – Результаты тестирования

|  |  |  |
| --- | --- | --- |
| Исходные данные (кол-во полиномов, размерность) | Время при «плохом» алгоритме, мс | Время при «хорошем» алгоритме, мс |
| 2, 100000 | 428 мс | 210 мс |
| 4, 100000 | 3211 мс | 1183 мс |
| 6, 100000 | 11605 мс | 5485 мс |
| 8, 100000 | 51745 мс | 24990 мс |
| 2, 1000000 | 2897 мс | 1493 мс |
| 4, 1000000 | 21058 мс | 10734 мс |
| 6, 1000000 | 98548 мс | 50241 мс |
| 8, 1000000 | 565176 мс | 259036 мс |
| 2, 10000000 | 52532 мс | 26759 мс |
| 8, 10000 | 52532 мс | 2640 мс |

1. Вывод

В ходе выполнения лабораторной работы был реализован алгоритм перемножения полиномов с помощью быстрого преобразованья Фурье. Была проделана определённая оптимизация алгоритма для улучшения его работы, что помогло ускорить его почти в 2 раза. В ходе тестирования было замечено, что время работы алгоритма увеличивается как при увеличении размерности векторов, так и при увеличении их количества. Второй критерий в большей степени влияет на время выполнения алгоритма

Приложение А

(обязательное)

Листинг программы

#include <iostream>

#include <fstream>

#include <vector>

#include <complex>

#include <bitset>

#include <ctime>

using namespace std;

typedef complex<double> base;

vector<int> information[100]; //data vector

vector<int> result(1000000000); //result vector

int counter = 1;

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 bad\_realisation(vector<base>& a,bool invert) { //main part of the fast conversion

int n = (int)a.size();

if (n == 1) return;

vector<base> a0(n / 2), a1(n / 2);

for (int i = 0, j = 0; i < n; i += 2, ++j) {

a0[j] = a[i];

a1[j] = a[i + 1];

}

bad\_realisation(a0,invert);

bad\_realisation(a1,invert);

double ang = 2 \* 3.14 / n \* (invert ? -1 : 1);

base w(1), wn(cos(ang), sin(ang));

for (int i = 0; i < n / 2; ++i) {

a[i] = a0[i] + w \* a1[i];

a[i + n / 2] = a0[i] - w \* a1[i];

if (invert)

a[i] /= 2, a[i + n / 2] /= 2;

w \*= wn;

}

}

void bad\_multiplication(const vector<int>& a, const vector<int>& b, vector<int>& res,int number) { //multiplication of two vectors

if (counter != number) {

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);

bad\_realisation(fa, false), bad\_realisation(fb, false);

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

fa[i] \*= fb[i];

bad\_realisation(fa, true);

res.resize(n);

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

res[i] = int(fa[i].real() + 0.5);

counter++;

bad\_multiplication(information[counter], result, result,number);

}

}

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

if (counter != number) {

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);

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);

counter++;

good\_multiplication(information[counter], result, result,number);

}

}

void enter(string first,int number,int size, string path) { //input

cout <<"\n" + first + " test is running\n";

fill\_from\_file(path, number, size); //number, size

cout << "Bad algorithm: ";

unsigned int start\_time = clock();

bad\_multiplication(information[0], information[1], result, number);

unsigned int end\_time = clock();

unsigned int search\_time = end\_time - start\_time;

cout << search\_time << " mc\n";

counter = 1;

cout << "Good algorithm: ";

unsigned int start\_time = clock();

good\_multiplication(information[0], information[1], result, number);

unsigned int end\_time = clock();

unsigned int search\_time = end\_time - start\_time;

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");

}