Министерство образования Республики Беларусь

Учреждение образования «Полоцкий государственный университет имени Евфросинии Полоцкой»

Факультет информационных технологий

Кафедра технологий программирования

**Отчёт по лабораторной работе № 2 по курсу «Распределенные вычисления»**

«ВВЕДЕНИЕ В ПРОГРАММИРОВАНИЕ GPU»

ВЫПОЛНИЛ студент группы 21-ИТ-1

Чиникайло А.П.

ПРОВЕРИЛ преподаватель

Адамовский Е.Р.

Полоцк, 2024 г.

Цель: ознакомиться с основами использования технологии NVIDIA CUDA

для программирования GPU.

**ОПИСАНИЕ ПРОДЕЛАННОЙ РАБОТЫ**

Задание:

1. Реализовать консольное приложение (с использованием NVIDIA CUDA) для выполнения алгоритма K-средних согласно представленным вариантам в таблице.

2. Запустить реализованное консольное приложение, измерить встроенными средствами время выполнения алгоритма и его отдельных итераций на CPU и GPU.

3. Представить результаты измерений в графическом виде и сравнить их.

|  |  |  |
| --- | --- | --- |
| **Вариант** | **Количество элементов и атрибутов элемента** | **Количество кластеров** |
| 12 | 120 000 × 5 | 6 |

Листинг 1 – lab\_2.cpp

#include <iostream>

#include <vector>

#include <ctime>

#include <chrono>

#include <random>

#include <fstream>

#include <thread>

#include <omp.h>

using namespace std;

//класс для статических элементов

class Const {

public:

static const int amountOfElements = 30;//количество элементов

static const int numberOfElementAttributes = 2;// количество атрибутов измерений

static const int numberOfClusters = 3;// количество кластеров

};

//функция подсчета расстояния между точками

double calculateDistance(vector<double>& point1, vector<double>& point2) {

double distance = 0.0;// объявление переменной расстояние

for (int i = 0; i < Const::numberOfElementAttributes; i++) {

distance += pow((point2[i] - point1[i]), 2);// вычисляем по формуле

}

return sqrt(distance);// возвращаем квадрат

}

// функция для параллельного назначения кластера

void assignToClusterParaller(int i, vector<vector<double>>& points, vector<vector<double>>& centroids, vector<int>& clusterAssignments) {

double minDistance = numeric\_limits<double>::max(); // берем значения что было с чем сравнивать

int closestCentroid = -1;// минимальное значение для сравнения ближайшего центроида

for (int j = 0; j < Const::numberOfClusters; j++) {

double distance = calculateDistance(points[i], centroids[j]);// вычисляем дистанцию

if (distance < minDistance) { // если ближе запомнаем

minDistance = distance;//запоминаем наименьшее расстояние

closestCentroid = j;//запоминаем ближайший центроид

}

}

clusterAssignments[i] = closestCentroid; // назначаем кластер

}

// присваиваем точки к кластеру

void assignToClusters(vector<vector<double>>& points, vector<vector<double>>& centroids, vector<int>& clusterAssignments) {

vector<thread> threads;//объявляем вектор потоков

for (int i = 0; i < Const::amountOfElements; i++) {

threads.emplace\_back(assignToClusterParaller, i, ref(points),

ref(centroids), ref(clusterAssignments));// запускаем поток

}

for (auto& thread : threads) {

thread.join();// ожидаем выполнение всех потоков

}

}

// функция для параллельного обновление центроидов

void updateCentroidParaller(int i, vector<vector<double>>\* points, vector<vector<double>>\* centroids, vector<int>\* clusterAssignments) {

clock\_t s = clock();//получаем время в секундах

vector<double> newCentroid(Const::numberOfElementAttributes);//объявляем новый центроид

int count = 0;// количество точек принадлежащих кластеру

for (int j = 0; j < Const::amountOfElements; j++) {

if ((\*clusterAssignments)[j] == i) {

for (int l = 0; l < Const::numberOfElementAttributes; l++) {

newCentroid[l] += (\*points)[j][l];// вычисляем сумму атрибутов точек принадлежащих кластеру

}

count++;// считает количество точек

}

}

if (count > 0) {

for (int l = 0; l < Const::numberOfElementAttributes; l++) {

newCentroid[l] /= count; // делим на количество точек в кластере

}

(\*centroids)[i] = newCentroid;// запоминаем новый центроид

}

clock\_t e = clock();//получаем время в секундах

cout << i << endl;

cout << "time: " << e - s << endl;// вычисляем сколько времени потребовалось на обновление центроида

}

// считаем новые координаты центроидов

void updateCentroids(vector<vector<double>>& points, vector<vector<double>>& centroids, vector<int>& clusterAssignments) {

vector<thread> threads; ;//объявляем вектор потоков

for (int i = 0; i < Const::numberOfClusters; i++) {

threads.emplace\_back(updateCentroidParaller, i, &points,

&centroids, &clusterAssignments); ));// запускаем поток

}

for (auto& thread : threads) {

thread.join();// ожидаем выполнение всех потоков

}

}

//функция вывода для каждой итерации промежуточных значений

void PrintIteration(int& count, vector<vector<double>>& points, vector<vector<double>>& centroids, vector<int>& clusterAssignments) {

cout << "iteration " << count << endl;

int countPoints = 0;// объявляем количество точек принадлежащих кластеру

for (int i = 0; i < Const::numberOfClusters; i++) {

cout << "cluster " << i << ":" << endl;

countPoints = 0;

for (int j = 0; j < Const::amountOfElements; j++) {

if (clusterAssignments[j] == i) {

countPoints++;

cout << "point " << j << ": ";

cout << calculateDistance(points[j], centroids[i]) << "\t";// вычисляем расстояние между точкой и центроидом

for (int k = 0; k < Const::numberOfClusters; k++) {

if (k != i) {

cout << k << ": " << calculateDistance(points[j], centroids[k]) << "\t";// вычисляем расстояние между точками, что бы сравнить с

другими цектроидами

}

}

cout << endl;

}

}

cout << "count points in cluster: " << countPoints << endl;

}

cout << "\n\n\n";

}

//функция выполнение алгоритма kMeans

void kMeans(vector<vector<double>>& points, vector<vector<double>>& centroids, vector<double>& iterationTimes) {

vector<int> OldclusterAssignments(Const::amountOfElements);// объявляем для сравнение достигла ли программа конца, то есть алгоритм сработал до конца

int count = 0;// количество итераций

while (true) {

double s = clock();//получаем время в секундах

vector<int> clusterAssignments(Const::amountOfElements);

assignToClusters(points, centroids, clusterAssignments);// вызываем функцию назначения кластера

updateCentroids(points, centroids, clusterAssignments);// вызываем функцию обновления центоидов

double e = clock(); //получаем время в секундах

iterationTimes.push\_back((e - s) / CLOCKS\_PER\_SEC); // вычисляем сколько времени было потраченно на выполнении итерации

PrintIteration(count, points, centroids, clusterAssignments);// вызываем функцию для вывода на экран промежуточных данныз

if (clusterAssignments == OldclusterAssignments) { // если ничего не поменялось, значит алгоритм завершил свою работу

cout << "True" << count << endl;

ofstream outFile("../../CountIteration.txt");//открываем файл

if (outFile.is\_open()) {

outFile << count;// записываем количество итераций

outFile.close();//закрываем файл

}

else {

cout << "Error, could not open file.";

}

break;

}

OldclusterAssignments = clusterAssignments;// запоминаем для дальнейшего сравнения

count++;

}

}

int main() {

vector<vector<double>> points(Const::amountOfElements, vector<double>(Const::numberOfElementAttributes));// объявляем точки

vector<vector<double>> centroids(Const::numberOfClusters, vector<double>(Const::numberOfElementAttributes));// объявляем центроиды

vector<double> iterationTimes; // объявляем вектор для хранения времени каждой итерации

for (int i = 0; i < Const::amountOfElements; i++) { // цикл для того что бы расставить рандомно точки

for (int j = 0; j < Const::numberOfElementAttributes; j++) {

points[i][j] = 1 + rand() % 100;// присваиваем рандомное значение

}

}

for (int i = 0; i < Const::numberOfClusters; i++) { // так же рандомно кластеры

for (int j = 0; j < Const::numberOfElementAttributes; j++) {

centroids[i][j] = 1 + rand() % 100;// присваиваем рандомное значение

}

}

double s = clock();//начало отсчета

kMeans(points, centroids, iterationTimes);

double e = clock();//конец отсчета

cout << "K-Means Algorithm Execution Time: " << (e - s) / CLOCKS\_PER\_SEC << " s" << endl;//вычисляем время выполнения алгоритма

ofstream outFile("../../IterationTimes.txt");//открываем файл

if (outFile.is\_open()) {

for (int i = 0; i < iterationTimes.size(); i++) {//записываем в него время каждой итерации

outFile << iterationTimes[i] << " ";

}

outFile.close();//закрываем файл

}

else { // если не получилось открыть файл

cout << "Error, could not open file.";

}

return 0;

}

**Результаты выполнения лабораторной работы**

Для демонстрации работы программы были изменены входные данные, а именно количество элементов равное 30, количество атрибутов элементов равное 2, количество кластеров равное 3.

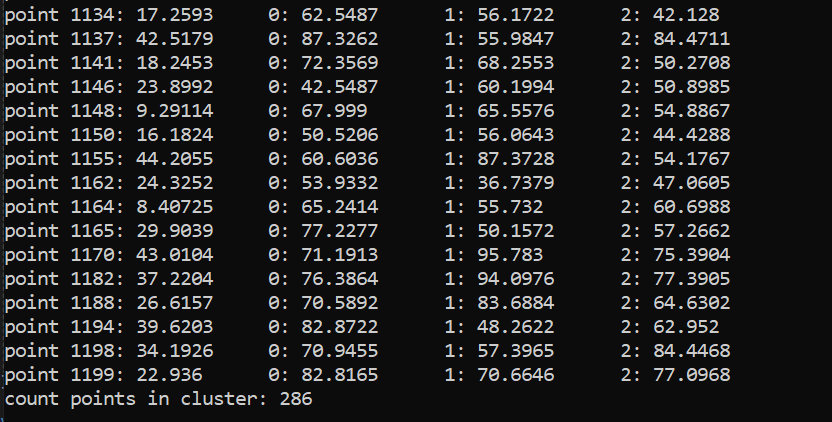


Рисунок 1 – расстояние до принадлежащего кластера и для всех остальных кластеров

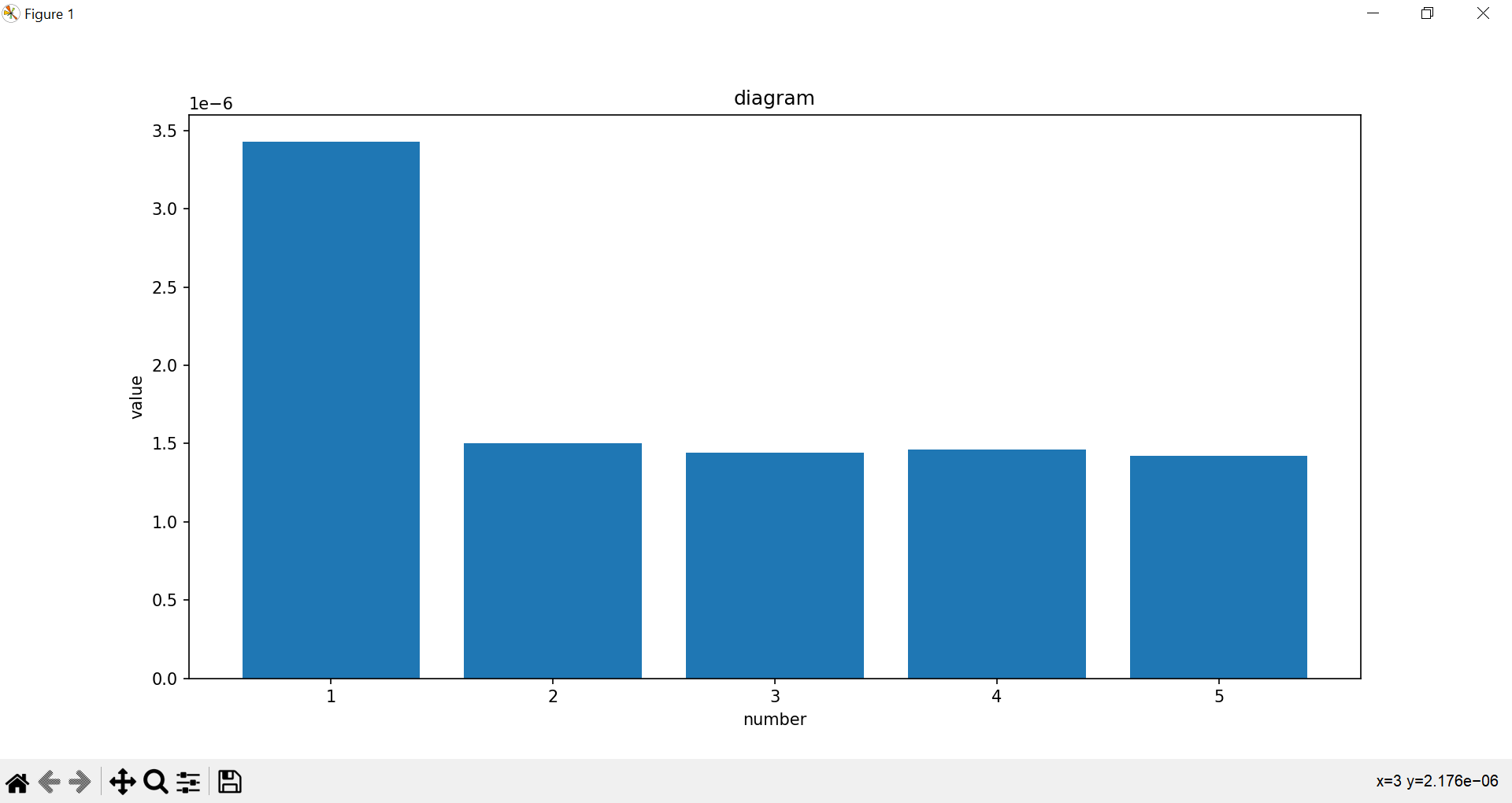


Рисунок 2 – Время выполнения итераций алгоритма без использования потоков

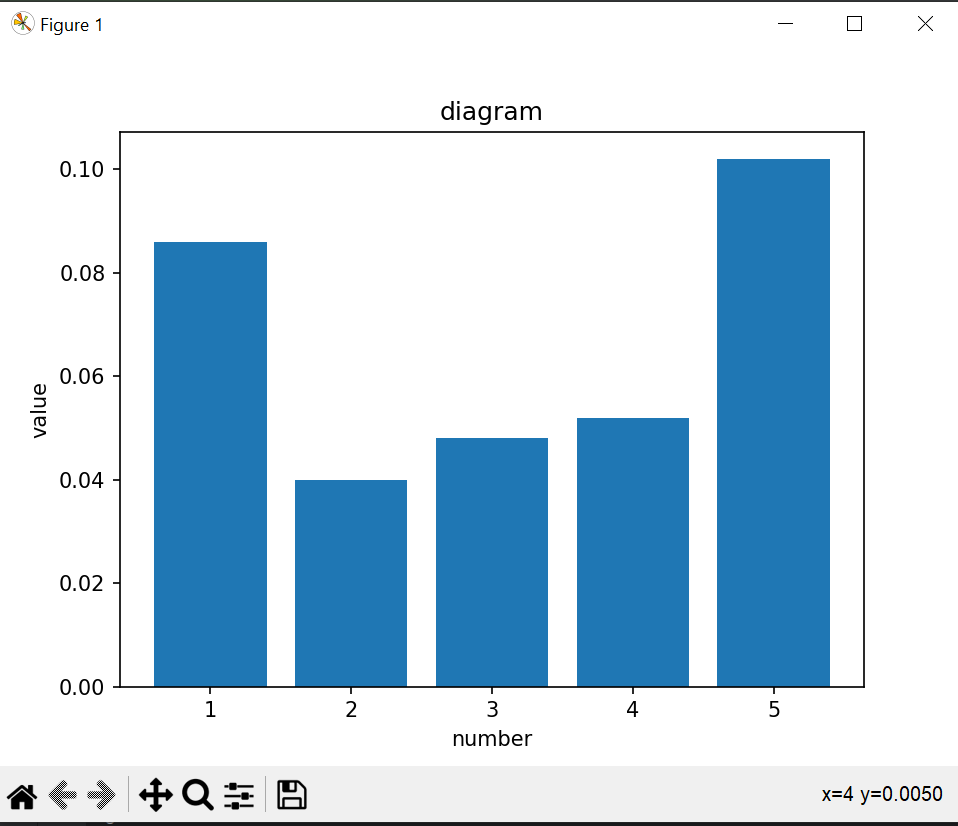


Рисунок 2 – Время выполнения итераций алгоритма с использованием потоков

**ВЫВОД**

В результате выполнения лабораторной работы был написан алгоритм кластеризации методом K-средних на C++ с распараллеливанием на CPU. При сравнении алгоритма без распараллеливания и с распараллеливанием, можно увидеть что с программа работает быстрее с распараллеливанием. Стоит отметить, что с одними и теми же входными данными, выходные данные остаются такими же.