

**Расчетно-графическая работа**

**ПО ДИСЦИПЛИНЕ «Параллельное программирование”»**

Факультет: АВТФ Преподаватель: Малявко А.А.

Группа: АВТ-918

Студент: Ванин К.Е.

# Лабораторная работа №1

**«Параллельное программирование для систем с общей памятью с использованием технологии OpenMP.»**

**Цель работы:**

- изучить технологию OpenMP и основные разработки параллельных программ для многоядерных процессоров;

- написать и отладить на языке C параллельную программу для решения поставленной задачи на системе с общей памятью;

**Задание**

Задача 7. Решение системы линейных алгебраических уравнений методом Якоби.

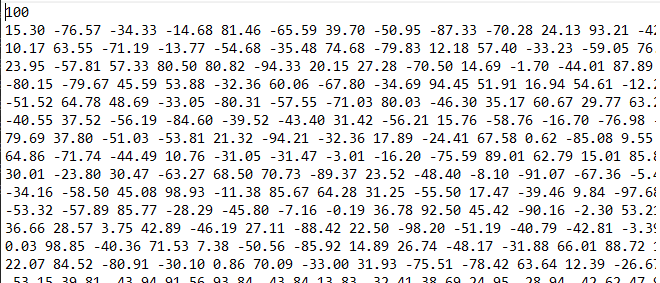
**Результат работы**

В ходе выполнения лабораторной работы были разработаны и отлажены последовательное и параллельное решение системы линейных уравнений методом Якоби.

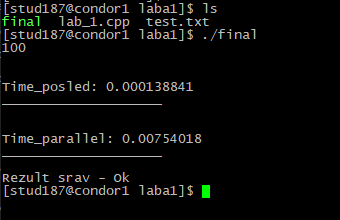
Входные значения была взята с генерации исходных данных.

Размерность СЛАУ: 100

Диапозон значений коэффициентов: [-100:100]



*Рис. 1 – Файл с исходными данными.*



*Рис. 2 – Результат выполнения программы.*

**Листинг программы лабораторной работы №1:**

#include <fstream>

#include <iostream>

#include <vector>

#include <locale.h>

#include <stdlib.h>

#include <time.h>

#ifdef \_OPENMP

#include <omp.h>

#endif

using namespace std;

vector <double> v1;

vector <double> v2;

void posled(int size, vector<vector<double> >& m\_vector, vector<double>& zna4, double eps)

{

vector<double> pred(size, 0.0);

vector<double> now(size, 0.0);

bool z = true;

double start1 = omp\_get\_wtime();

while (z)

{

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

{

now[i] = zna4[i];

for (int j = 0; j < size; j++)

{

if (i != j) now[i] -= m\_vector[i][j] \* pred[j];

}

now[i] /= m\_vector[i][i];

}

double er = 0.0;

for (int i = 0; i < size; i++) er += abs(now[i] - pred[i]);

if (er < eps) z = false;

pred = now;

v1 = pred;

}

double end1 = omp\_get\_wtime();

cout << endl;

cout << endl << "Time\_posled: " << end1 - start1 << endl;

cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" << endl;

}

void parallel(int size, vector<vector<double> >& m\_vector, vector<double>& zna4, double eps)

{

vector<double> pred(size, 0.0);

vector<double> now(size, 0.0);

bool z = true;

double start2 = omp\_get\_wtime();

while (z)

{

#pragma omp parallel

{

#pragma omp for

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

{

now[i] = zna4[i];

for (int j = 0; j < size; j++)

{

if (i != j) now[i] -= m\_vector[i][j] \* pred[j];

}

now[i] /= m\_vector[i][i];

}

double er = 0.0;

#pragma omp for

for (int i = 0; i < size; i++) er += abs(now[i] - pred[i]);

if (er < eps) z = false;

pred = now;

v2 = pred;

}

}

double end2 = omp\_get\_wtime();

cout << endl;

cout << endl << "Time\_parallel: " << end2 - start2 << endl;

cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" << endl;

}

bool srav()

{

for (unsigned int a = 0; a < v1.size(); a++)

{

if (v1[a] != v2[a])

return false;

cout << v1[a] << "|" << v2[a] << endl;

}

return true;

}

int main()

{

setlocale(LC\_ALL, "Russian");

int size = 0;

ifstream file;

file.open("test.txt");

file >> size;

cout << size << endl;

vector <vector<double> > m\_vector;

m\_vector.resize(size);

vector <double> zna4;

zna4.resize(size);

for (int ab = 0; ab < size; ab++)

{

m\_vector[ab].resize(size);

for (int ac = 0; ac < size; ac++)

{

file >> m\_vector[ab][ac];

if (ab == ac) m\_vector[ab][ac] = 1000;

}

file >> zna4[ab];

}

for (int ab = 0; ab < size; ab++)

{

for (int ac = 0; ac < size; ac++)

{

cout << m\_vector[ab][ac] << " ";

}

cout << endl;

}

double eps = 0.00001;

posled(size, m\_vector, zna4, eps);

parallel(size, m\_vector, zna4, eps);

cout << endl << "Rezult srav - " << (srav() ? "Ok" : "No") << endl;

file.close();

}

# Лабораторная работа №2

**«Параллельное программирование для графического процессора в среде NVidia CUDA.»**

**Цель работы:**

- изучить технологию CUDA и основные разработки параллельных программ для совместного использования CPU и GPU;

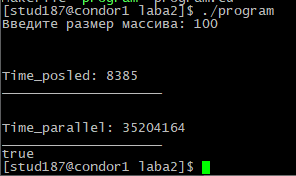
- написать и отладить на языке C параллельную программу для решения поставленной задачи на графическом процессоре;

**Задание**

Задача 7. Пирамидальная сортировка.

**Результат работы**

В ходе выполнения лабораторной работы были разработаны и отлажены последовательное и параллельное решение. С помощью утилиты make выполнили сборку программы program.cu. Команда make run.



*Рис. 3 Результат выполнения программы.*

Как видим, программа скомпилировалась. Ввели число 100. Получили результат с двумя способами. А время работы получились разные. В данном случае время последовательного метода намного меньше, чем у параллельного метода, так как ввели малое число. А если считать огромные числа, то параллельный метод окажется быстрее, чем последовательный.

**Листинг программы лабораторной работы №2:**

#include <stdio.h>

#include <iostream>

#include <fstream>

#include <locale.h>

#include <time.h>

#include "cuda\_runtime.h"

#include "device\_launch\_parameters.h"

const int cuSize = 100;

const int bSize = 256;

#define INT\_64 long long

using namespace std;

INT\_64 LongRand()

{

return rand() << 16 | rand();

}

int\* mas2;

int minInt(int x, int y)

{

if (x <= y) return x;

else return y;

}

//построение дерева из массива

void downHeap(int\* a, long k, long n) {

int new\_elem;

long child;

new\_elem = a[k];

while (k <= n / 2) { // пока у a[k] есть дети

child = 2 \* k;

// выбираем большего сына

if (child < n && a[child] < a[child + 1])

child++;

if (new\_elem >= a[child]) break;

// иначе

a[k] = a[child]; // переносим сына наверх

k = child;

}

a[k] = new\_elem;

}

//сортировка пирамидой

void sort(int\* a, int size)

{

for (int i = size / 2; i >= 0; i--) downHeap(a, i, size - 1);

for (int i = size - 1; i > 0; i--)

{

int temp = a[i];

a[i] = a[0];

a[0] = temp;

downHeap(a, 0, i - 1);

}

mas2 = a;

}

\_\_global\_\_ void cudaSort(int\* a, int size, int part\_size)

{

int id = blockDim.x \* blockIdx.x + threadIdx.x;

if (id \* size >= part\_size) return;

a += id \* size;

if (part\_size - id \* size < size) size = part\_size - id \* size;

for (int i = size / 2; i >= 0; i--)

{

long k = i; long n = size - 1;

int new\_elem;

long child;

new\_elem = a[k];

while (k <= n / 2) { // пока у a[k] есть дети

child = 2 \* k;

// выбираем большего сына

if (child < n && a[child] < a[child + 1])

child++;

if (new\_elem >= a[child]) break;

// иначе

a[k] = a[child]; // переносим сына наверх

k = child;

}

a[k] = new\_elem;

}

for (int i = size - 1; i > 0; i--)

{

int temp = a[i];

a[i] = a[0];

a[0] = temp;

long k = 0, n = i - 1;

int new\_elem;

long child;

new\_elem = a[k];

while (k <= n / 2) { // пока у a[k] есть дети

child = 2 \* k;

// выбираем большего сына

if (child < n && a[child] < a[child + 1])

child++;

if (new\_elem >= a[child]) break;

// иначе

a[k] = a[child]; // переносим сына наверх

k = child;

}

a[k] = new\_elem;

}

}

//Слияние двух расположенных рядом друг с другом частей массива

void merge(int\* A, int\* B, int l, int m, int r)

//A - сортируемый массив, B - буфер для слияния, l - первый элемент первой части, r - последний элемент второй части, m - последний элемент первой части

{

int i = l;

int j = m + 1;

int k = l;

//Вставлять минимальные элементы в B пока не кончится одна из последовательностей

while ((i <= m) && (j <= r))

{

if (A[i] < A[j])

{

B[k] = A[i];

i++;

}

else {

B[k] = A[j];

j++;

}

k++;

}

//Скопировать остаток, если таковой имеется

while (i <= m)

{

B[k] = A[i];

k++;

i++;

}

while (j <= r)

{

B[k] = A[j];

k++;

j++;

}

//Отсортированная часть остаётся в B

}

//Сортировка слиянием, распараллеленная

int\* mp\_sort(int\* A, int N, int P)

//A - сортируемый массив, B - буфер для слияния, N - размер массива, P - число параллельно выполняемых потоков

{

int\* tA = A;

int\* tB = new int[N];

//1)Разбиение на блоки и сортировка их по отдельности

int i;

int part\_size = (int)ceil((float)N / (float)P);

int\* mas = new int[part\_size];

for (i = 0; i < P; i++)

{

part\_size = minInt(part\_size, N - i \* part\_size);

for (int j = 0; j < part\_size; j++)

mas[j] = tA[j + i \* ((int)ceil((float)N / (float)P))];

int thCount = N / (bSize \* cuSize);

if (N % (bSize \* cuSize) > 0) thCount++;

int\* cuMas;

cudaMalloc((void\*\*)&cuMas, sizeof(int) \* N);

cudaMemcpy(cuMas, mas, sizeof(int) \* N, cudaMemcpyHostToDevice);

cudaSort << < thCount, bSize >> > (cuMas, cuSize, N);

cudaMemcpy(mas, cuMas, sizeof(int) \* N, cudaMemcpyDeviceToHost);

for (int j = 0; j < part\_size; j++)

tA[i \* ((int)ceil((float)N / (float)P)) + j] = mas[j];

}

part\_size = (int)ceil((float)N / (float)P);

int r2, m;

while (part\_size < (2 \* N))

{

for (i = 0; i < N; i += part\_size)

{

r2 = minInt(i + part\_size - 1, N - 1);

m = minInt(((i + i + part\_size - 1) >> 1), N - 1);

merge(tA, tB, i, m, r2);

}

part\_size \*= 2;

int\* t = tA;

tA = tB;

tB = t;

}

return mas2;

}

bool test(int\* A, int\* B, int size)

{

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

if (A[i] != B[i])

{

return false;

}

//for (int i = 0; i < size; i++) cout << A[i] << "|" << B[i] << endl;

return true;

}

int main()

{

setlocale(LC\_ALL, ".ACP");

struct timespec mk1, mk2;

struct timespec mt1, mt2;

cout << "Введите размер массива: ";

int n;

cin >> n;

int\* mas = new int[n];

int\* mas1 = new int[n];

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

{

int r = rand();

mas[i] = r;

mas1[i] = r;

}

cout << endl;

clock\_gettime(CLOCK\_REALTIME, &mk1);

sort(mas, n);

clock\_gettime(CLOCK\_REALTIME, &mk2);

cout << endl;

cout << endl << "Time\_posled: " << mk2.tv\_nsec - mk1.tv\_nsec << endl;

cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" << endl;

clock\_gettime(CLOCK\_REALTIME, &mt1);

mas1 = mp\_sort(mas1, n, 4);

clock\_gettime(CLOCK\_REALTIME, &mt2);

cout << endl;

cout << endl << "Time\_parallel: " << mt2.tv\_nsec - mt1.tv\_nsec << endl;

cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" << endl;

if (test(mas, mas1, n)) cout << "true" << endl;

else cout << "false" << endl;

delete[] mas;

// delete[] mas1;

}

# Лабораторная работа №3

**«Параллельное программирование для гетерогенных вычислительных систем с использованием технологии OpenCL.»**

**Цель работы:**

- изучить библиотечное расширение OpenCL языка C и основные разработки параллельных программ для совместного использования CPU и GPU;

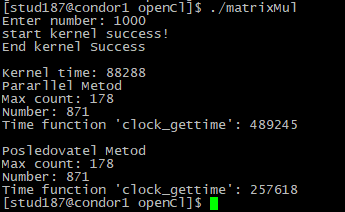
- написать и отладить с использованием технологии OpenCL параллельную программу для решения поставленной задачи на графическом процессоре;

**Задание**

Задача 34**.** Любое число n, большее единицы, порождает последовательность вида: ni+1 = (ni % 2 == 0? ni / 2 : 3 \* ni + 1), последним элементом которой является 1 (например: 13->40->20->10->5->16->8->4->2->1). Найти наибольшее, меньшее заданного N, число, порождающее последовательность заданной длины.

**Результат работы**

В ходе выполнения лабораторной работы были разработаны и отлажены последовательное и параллельное решение. С помощью скрипта createMMul.sh выполнили сборку программу matrixMul.c.



*Рис 4. Результат выполнения программы.*

Таким образом, программа скомпилировалась. Ввели число 1000 и получили одинаковый результат для двух методов решения. Время выполнения параллельного метода быстрее, чем у последовательного метода.

**Листинг программы лабораторной работы №3:**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <malloc.h>

#include <CL/cl\_platform.h>

#include <CL/cl.h>

#include <time.h>

char\* errCodeToString(int err) {

switch (err) {

case CL\_SUCCESS: return "Success!";

case CL\_DEVICE\_NOT\_FOUND: return "Device not found.";

case CL\_DEVICE\_NOT\_AVAILABLE: return "Device not available";

case CL\_COMPILER\_NOT\_AVAILABLE: return "Compiler not available";

case CL\_MEM\_OBJECT\_ALLOCATION\_FAILURE: return "Memory object allocation failure";

case CL\_OUT\_OF\_RESOURCES: return "Out of resources";

case CL\_OUT\_OF\_HOST\_MEMORY: return "Out of host memory";

case CL\_PROFILING\_INFO\_NOT\_AVAILABLE: return "Profiling information not available";

case CL\_MEM\_COPY\_OVERLAP: return "Memory copy overlap";

case CL\_IMAGE\_FORMAT\_MISMATCH: return "Image format mismatch";

case CL\_IMAGE\_FORMAT\_NOT\_SUPPORTED: return "Image format not supported";

case CL\_BUILD\_PROGRAM\_FAILURE: return "Program build failure";

case CL\_MAP\_FAILURE: return "Map failure";

case CL\_INVALID\_VALUE: return "Invalid value";

case CL\_INVALID\_DEVICE\_TYPE: return "Invalid device type";

case CL\_INVALID\_PLATFORM: return "Invalid platform";

case CL\_INVALID\_DEVICE: return "Invalid device";

case CL\_INVALID\_CONTEXT: return "Invalid context";

case CL\_INVALID\_QUEUE\_PROPERTIES: return "Invalid queue properties";

case CL\_INVALID\_COMMAND\_QUEUE: return "Invalid command queue";

case CL\_INVALID\_HOST\_PTR: return "Invalid host pointer";

case CL\_INVALID\_MEM\_OBJECT: return "Invalid memory object";

case CL\_INVALID\_IMAGE\_FORMAT\_DESCRIPTOR: return "Invalid image format descriptor";

case CL\_INVALID\_IMAGE\_SIZE: return "Invalid image size";

case CL\_INVALID\_SAMPLER: return "Invalid sampler";

case CL\_INVALID\_BINARY: return "Invalid binary";

case CL\_INVALID\_BUILD\_OPTIONS: return "Invalid build options";

case CL\_INVALID\_PROGRAM: return "Invalid program";

case CL\_INVALID\_PROGRAM\_EXECUTABLE: return "Invalid program executable";

case CL\_INVALID\_KERNEL\_NAME: return "Invalid kernel name";

case CL\_INVALID\_KERNEL\_DEFINITION: return "Invalid kernel definition";

case CL\_INVALID\_KERNEL: return "Invalid kernel";

case CL\_INVALID\_ARG\_INDEX: return "Invalid argument index";

case CL\_INVALID\_ARG\_VALUE: return "Invalid argument value";

case CL\_INVALID\_ARG\_SIZE: return "Invalid argument size";

case CL\_INVALID\_KERNEL\_ARGS: return "Invalid kernel arguments";

case CL\_INVALID\_WORK\_DIMENSION: return "Invalid work dimension";

case CL\_INVALID\_WORK\_GROUP\_SIZE: return "Invalid work group size";

case CL\_INVALID\_WORK\_ITEM\_SIZE: return "Invalid work item size";

case CL\_INVALID\_GLOBAL\_OFFSET: return "Invalid global offset";

case CL\_INVALID\_EVENT\_WAIT\_LIST: return "Invalid event wait list";

case CL\_INVALID\_EVENT: return "Invalid event";

case CL\_INVALID\_OPERATION: return "Invalid operation";

case CL\_INVALID\_GL\_OBJECT: return "Invalid OpenGL object";

case CL\_INVALID\_BUFFER\_SIZE: return "Invalid buffer size";

case CL\_INVALID\_MIP\_LEVEL: return "Invalid mip-map level";

default: return "Unknown";

}

}

int Summ(int x)

{

int64\_t sum = 0, sn = 0;

int count = 0;

sum = x;

sn = x;

int j;

for (j = 0; sum != 1; j++) {

sn = (sn % 2 == 0 ? sn / 2 : 3 \* sn + 1);

sum = sn;

count++;

}

return count;

}

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

{

int arr\_size = 1024;

int i, j;

int N;

int maxcount = 0;

int arrpos = 0;

struct timespec start1, end1;

struct timespec start2, end2;

printf("Enter number: ");

scanf("%d", &N);

int\* ResultArray = (int\*)malloc(sizeof(int) \* arr\_size);

int\* Numbers = (int\*)malloc(sizeof(int) \* arr\_size);

for (i = 0; i < N; i++) Numbers[i] = i;

cl\_mem ResultArray\_mem\_obj;

cl\_mem Numbers\_mem\_obj;

//parallel part

cl\_int clerr;

cl\_uint qty\_platforms = 0;

cl\_platform\_id\* platforms;

cl\_uint ui;

cl\_uint\* qty\_devices;

cl\_device\_id\*\* devices;

cl\_context cntx;

cl\_event kEvent;

long long tStart = 0l;

long long tEnd = 0l;

cl\_command\_queue cq;

cl\_device\_id\* ds; // devices

cl\_program p;

cl\_kernel k;

char\* program\_source = "\_\_kernel void matrix\_multiply ("

"\_\_global int \* ResultArray, \_\_global int \* Numbers, int N)"

"{"

"int id = get\_global\_id(0);"

"int j = 0;"

"int sum = Numbers[id];"

"int count = 0;"

"/\*for (j = 0; sum != 1;) {\*/"

"if ((sum == 0) || (sum == 1)) ResultArray[id] = 0;"

"else {"

"for (j = 0; sum != 1; j++) {"

" sum = (sum % 2 == 0 ? sum / 2 : 3 \* sum + 1); count++; }"

" ResultArray[id] = count; }"

"}"; // kernel source code

size\_t global\_work\_size = arr\_size;

size\_t local\_work\_size = 2;

size\_t cb;

// ...

clerr = clGetPlatformIDs(0, NULL, &qty\_platforms);

if (clerr != CL\_SUCCESS) {

fprintf(stderr, "Error, code = %d.\n", clerr);

return 1;

}

platforms = (cl\_platform\_id\*)malloc(sizeof(cl\_platform\_id) \* qty\_platforms);

devices = (cl\_device\_id\*\*)malloc(sizeof(cl\_device\_id\*) \* qty\_platforms);

qty\_devices = (cl\_uint\*)malloc(sizeof(cl\_uint) \* qty\_platforms);

clerr = clGetPlatformIDs(qty\_platforms, platforms, NULL);

for (ui = 0; ui < qty\_platforms; ui++) {

clerr = clGetDeviceIDs(platforms[ui], CL\_DEVICE\_TYPE\_ALL, 0, NULL, &qty\_devices[ui]);

if (qty\_devices[ui]) {

devices[ui] = (cl\_device\_id\*)malloc(qty\_devices[ui] \* sizeof(cl\_device\_id));

clerr = clGetDeviceIDs(platforms[ui], CL\_DEVICE\_TYPE\_ALL, qty\_devices[ui], devices[ui], NULL);

}

}

// clerr = clGetDeviceInfo(\*devices[0],CL\_DEVICE\_MAX\_WORK\_GROUP\_SIZE,sizeof(cl\_uint),&workGroupSize,NULL);

// clerr = clGetDeviceInfo(\*devices[0],CL\_DEVICE\_MAX\_COMPUTE\_UNITS,sizeof(cl\_uint),&computeUnits,NULL);

// clerr = clGetDeviceInfo(\*devices[0],CL\_DEVICE\_MAX\_WORK\_ITEM\_SIZES,3 \* sizeof(cl\_uint),&workItemSizes,NULL);

// if(clerr)

// errCodeToString(clerr);

clerr = CL\_SUCCESS;

cntx = clCreateContext(0, qty\_devices[0], devices[0], NULL, NULL, &clerr);

cq = clCreateCommandQueue(cntx, devices[0][0], CL\_QUEUE\_PROFILING\_ENABLE, &clerr);

p = clCreateProgramWithSource(cntx, 1, (const char\*\*)&program\_source, NULL, &clerr);

// sprintf(clcompileflags, "-cl -mad enable");

clerr = clBuildProgram(p, 0, NULL, NULL /\*clcompileflags\*/, NULL, NULL);

if (clerr == CL\_BUILD\_PROGRAM\_FAILURE) {

// Determine the size of the log

size\_t log\_size;

clGetProgramBuildInfo(p, devices[0][0], CL\_PROGRAM\_BUILD\_LOG, 0, NULL, &log\_size);

// Allocate memory for the log

char\* log = (char\*)malloc(log\_size);

// Get the log

clGetProgramBuildInfo(p, devices[0][0], CL\_PROGRAM\_BUILD\_LOG, log\_size, log, NULL);

// Print the log

FILE\* flog = fopen("errorLog.txt", "w");

fprintf(flog, "%s\n", log);

fclose(flog);

return 1;

}

if (clerr != CL\_SUCCESS) fprintf(stderr, "Error1, code = %d.\n%s\n", clerr, errCodeToString(clerr));

k = clCreateKernel(p, "matrix\_multiply", &clerr);

if (clerr != CL\_SUCCESS) fprintf(stderr, "Error2, code = %d.\n%s\n", clerr, errCodeToString(clerr));

//start of change program

clock\_gettime(CLOCK\_REALTIME, &start1);

ResultArray\_mem\_obj = clCreateBuffer(cntx, CL\_MEM\_READ\_ONLY, arr\_size \* sizeof(cl\_int), NULL, &clerr);

Numbers\_mem\_obj = clCreateBuffer(cntx, CL\_MEM\_READ\_ONLY, arr\_size \* sizeof(cl\_int), NULL, &clerr);

clerr = clEnqueueWriteBuffer(cq, ResultArray\_mem\_obj, CL\_TRUE, 0, arr\_size \* sizeof(cl\_int), ResultArray, 0, NULL, NULL);

clerr = clEnqueueWriteBuffer(cq, Numbers\_mem\_obj, CL\_TRUE, 0, arr\_size \* sizeof(cl\_int), Numbers, 0, NULL, NULL);

if (clerr != CL\_SUCCESS) fprintf(stderr, "Error3, code = %d.\n%s\n", clerr, errCodeToString(clerr));

i = -1;

clerr = clSetKernelArg(k, i += 1, sizeof(cl\_mem), (void\*)&ResultArray\_mem\_obj);

if (clerr != CL\_SUCCESS) fprintf(stderr, "Error4.1, code = %d.\n%s\n", clerr, errCodeToString(clerr));

clerr = clSetKernelArg(k, i += 1, sizeof(cl\_mem), (void\*)&Numbers\_mem\_obj);

clerr = clSetKernelArg(k, i += 1, sizeof(int), (void\*)&N);

if (clerr != CL\_SUCCESS) fprintf(stderr, "Error4.2, code = %d.\n%s\n", clerr, errCodeToString(clerr));

kEvent = clCreateUserEvent(cntx, &clerr);

//check clerr need

printf("start kernel success!\n");

clerr = clEnqueueNDRangeKernel(cq, k, 1, NULL, &global\_work\_size, &local\_work\_size, 0, NULL, &kEvent);

if (clerr != 0) printf("\nError5: %s\n", errCodeToString(clerr));

clerr = clEnqueueReadBuffer(cq, ResultArray\_mem\_obj, CL\_TRUE, 0, arr\_size \* sizeof(cl\_int), ResultArray, 0, NULL, NULL);

if (clerr != 0) printf("\nError6.1: %s\n", errCodeToString(clerr));

clerr = clGetEventProfilingInfo(kEvent, CL\_PROFILING\_COMMAND\_START, sizeof(tStart), &tStart, NULL);

clerr = clGetEventProfilingInfo(kEvent, CL\_PROFILING\_COMMAND\_END, sizeof(tEnd), &tEnd, NULL);

printf("End kernel Success\n");

clock\_gettime(CLOCK\_REALTIME, &end1);

//for (i = 2; i < N; i++) printf("%d) %d\n", i, ResultArray[i]);

for (i = 0; i < N; i++)

{

if (ResultArray[i] > ResultArray[arrpos]) arrpos = i;

maxcount = ResultArray[arrpos];

}

printf("\nKernel time: %lld\n", tEnd - tStart);

printf("Pararllel Metod\n");

printf("Max count: %d\n", maxcount);

printf("Number: %d\n", arrpos);

printf("Time function 'clock\_gettime': %ld\n", end1.tv\_nsec - start1.tv\_nsec);

clReleaseMemObject(ResultArray\_mem\_obj);

clReleaseKernel(k);

clReleaseProgram(p);

clReleaseCommandQueue(cq);

clReleaseContext(cntx);

free(ResultArray);

int maxcount1;

int arrpos1;

int\* ResultArray1 = (int\*)malloc(sizeof(int) \* N);

clock\_gettime(CLOCK\_REALTIME, &start2);

ResultArray1[0] = 0;

ResultArray1[1] = 0;

for (i = 2; i < N; i++)

{

int y = Summ(i);

ResultArray1[i] = y;

//printf("%d) %d\n", i, ResultArray1[i]);

}

maxcount1 = 0;

for (i = 0; i < N; i++)

{

if (ResultArray1[i] > ResultArray1[arrpos1]) arrpos1 = i;

maxcount1 = ResultArray1[arrpos1];

}

clock\_gettime(CLOCK\_REALTIME, &end2);

printf("\nPosledovatel Metod\n");

printf("Max count: %d\n", maxcount1);

printf("Number: %d\n", arrpos1);

printf("Time function 'clock\_gettime': %ld\n", end2.tv\_nsec - start2.tv\_nsec);

free(ResultArray1);

return 0;

}