1. Министерство образования и науки Российской Федерации
2. Санкт-Петербургский Политехнический Университет Петра Великого
3. —
4. Институт прикладной математики и механики
5. **Кафедра «Информационная безопасность компьютерных систем»**

**ЛАБОРАТОРНАЯ РАБОТА №2**

1. по дисциплине «Языки программирования»
2. Выполнил
3. студентка гр. 23636/2 Ивченкова Н. С

1. Проверил Алексеев И.В
2. преподаватель

1. Санкт-Петербург
2. 2018

**Ход работы:**

**Описание реализованного алгоритма:** В двумерный массив строк заносим сначала строки из файла, затем столбцы.

В ядре на каждый поток приходится по одной такой строке массива. id потока определяет индекс строки, которую он будет обрабатывать.

Что происходит в ядре:

Считать непрерывные наборы единиц или нулей будем отдельно для вертикали и горизонтали. Так как массив со всеми исходными строками один, а выходных два - вертикаль, горизонталь - еще одним передаваемым в ядро параметром будет количество строк по горизонтали. Зачем – дальше.

В каждой строке поток ищет непрерывные последовательности «1» или «0». Считает их длину и заносит в один из массивов таким образом: длина последовательности = индекс массива, при получении определенной длины инкрементируем данный элемент массива. То есть мы считаем, сколько у нас существует последовательностей определенной длины. А выбор массива определяется номером потока: если он превышает меньше количества строк по горизонтали – записываем в первый (pB), иначе – в массив для обработки столбцов (pBVer).   
Затем запрашиваем у пользователя количество возможных переходов - tired. И далее, получив выходные массивы с результатами, идем с конца, складывая tired максимальных длин.   
И для получения ответа вычитаем сумму максимальных длин из общего количества элементов в файле (его мы считаем в самом начале, когда заносим элементы в массив).

Выводы: В ходе проделанной работы мы научились выполнять параллельные вычисления с помощью OpenCL. Разбиение общей задачи на потоки по своей сути является более эффективным решением. И чем больше количество информации, которую необходимо обработать, тем разница будет все заметнее.

**Код:**

**Хост:**

#include <stdio.h>

#include <stdlib.h>

#include <tchar.h>

#include "CL\cl.h"

#include "utils.h"

#include <Windows.h>

#define OPENCL\_VERSION\_1\_2 1.2f

#define SIZE 90

#define COUNT\_STRING 180

struct ocl\_args\_d\_t

{

ocl\_args\_d\_t();

~ocl\_args\_d\_t();

cl\_context context; // hold the context handler

cl\_device\_id device; // hold the selected device handler

cl\_command\_queue commandQueue; // hold the commands-queue handler

cl\_program program; // hold the program handler

cl\_kernel kernel; // hold the kernel handler

float platformVersion; // hold the OpenCL platform version (default 1.2)

float deviceVersion; // hold the OpenCL device version (default. 1.2)

float compilerVersion; // hold the device OpenCL C =

cl\_mem srcA; // hold first source buffer

cl\_mem srcB; // hold second source buffer

cl\_mem dstMem; // hold destination buffer

cl\_mem srcBVer;

cl\_mem srcCount;

};

ocl\_args\_d\_t::ocl\_args\_d\_t() :

context(NULL),

device(NULL),

commandQueue(NULL),

program(NULL),

kernel(NULL),

srcA(NULL),

srcB(NULL),

srcBVer(NULL),

srcCount(NULL),

dstMem(NULL)

{

}

ocl\_args\_d\_t::~ocl\_args\_d\_t(){

cl\_int err = CL\_SUCCESS;}

int writeInArrayVer(char inputArray[][SIZE], int countOfString)//создаем большой массив строк и суем туда все строки из файла

{

int i2 = countOfString; int j2 = 0; int p;

for (int i = 0; i < SIZE; i++) {

int c = 0;

for (int j = 0; j < countOfString; j++) {

if (inputArray[j][i] == '1' || inputArray[j][i] == '0'){

inputArray[i2][j2] = inputArray[j][i]; j2++; c++; p = 0;}

else {if (p == 0)

{i2++;

j2 = 0;

}p++;}}

if (p == 0) {

i2++; p++;} j2 = 0; if (c == 0)return i2;

}

return i2;

int writeInArray(char inputArray[][SIZE], int countString[])//создаем большой массив строк и суем туда все строки из файла

{

FILE \*f;

f = fopen("test.txt", "r");

int count\_of\_all\_file = 0; //char \*s = 1;

int num\_string = -1;

while (num\_string == -1 || \*inputArray[num\_string] != 'М')

{//идем по всему файлу и записываем в массив строк все строки)

num\_string++;

fgets(inputArray[num\_string], SIZE, f);

count\_of\_all\_file += strlen(inputArray[num\_string]) - 1;

}

count\_of\_all\_file -= (strlen(inputArray[num\_string]) - 1);

fclose(f);

countString[0] = num\_string;

return count\_of\_all\_file;

}

void SetupOpenCL(ocl\_args\_d\_t \*ocl, cl\_device\_type deviceType)

{

cl\_platform\_id platformId = FindOpenCLPlatform("Intel", deviceType);

ocl->context = clCreateContextFromType(0, deviceType, NULL, NULL, NULL);

clGetContextInfo(ocl->context, CL\_CONTEXT\_DEVICES, sizeof(cl\_device\_id), &ocl->device, NULL);

ocl->commandQueue = clCreateCommandQueue(ocl->context, ocl->device, 0, NULL);

}

void CreateAndBuildProgram(ocl\_args\_d\_t \*ocl)

{

char\* source = NULL;

size\_t src\_size = 0;

ReadSourceFromFile("Template.cl", &source, &src\_size);

ocl->program = clCreateProgramWithSource(ocl->context, 1, (const char\*\*)&source, &src\_size, NULL);

clBuildProgram(ocl->program, 1, &ocl->device, "", NULL, NULL);

}

void ExecuteAddKernel(ocl\_args\_d\_t \*ocl, int countOfString)

{

size\_t globalWorkSize[] = { countOfString };

clEnqueueNDRangeKernel(ocl->commandQueue, ocl->kernel, 1, NULL, globalWorkSize, NULL, 0, NULL, NULL);

clFinish(ocl->commandQueue);//AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA}

int ReadAndVerify(ocl\_args\_d\_t \*ocl, int \*Ver){

cl\_int err = CL\_SUCCESS;

cl\_int \*resultPtrVer = (cl\_int \*)clEnqueueMapBuffer(ocl->commandQueue, ocl->srcBVer, true, CL\_MAP\_READ, 0, sizeof(int) \* SIZE, 0, NULL, NULL, &err);

cl\_int \*resultPtr = (cl\_int \*)clEnqueueMapBuffer(ocl->commandQueue, ocl->srcB, true, CL\_MAP\_READ, 0, sizeof(int) \* SIZE, 0, NULL, NULL, &err);

clFinish(ocl->commandQueue);

int tired; int sum = 0;

scanf("%d", &tired); int tired2 = tired;

for (int i = SIZE - 1; i > 0, tired > 0; i--){

while (resultPtr[i] > 0 && tired != 0){

resultPtr[i]--;

tired--;

sum += i; }

}

for (int i = SIZE - 1; i > 0, tired2 > 0; i--)

{

while (resultPtrVer[i] > 0 && tired2 != 0)

{

resultPtrVer[i]--;

tired2--;

\*Ver += i;}}

return sum;

}

int \_tmain(int argc, TCHAR\* argv[])

{

cl\_int err;

ocl\_args\_d\_t ocl;

cl\_device\_type deviceType = CL\_DEVICE\_TYPE\_GPU;

LARGE\_INTEGER perfFrequency;//time

LARGE\_INTEGER performanceCountNDRangeStart;

LARGE\_INTEGER performanceCountNDRangeStop;

SetupOpenCL(&ocl, deviceType);//информация о платфоре тыры-пыры все дела

char inputAllFile[COUNT\_STRING][SIZE];

int Res[30]; memset(Res, 0, 30 \* sizeof(int));

int ResVer[SIZE]; memset(ResVer, 0, SIZE \* sizeof(int));

int countOfString = 0 ;

int countOfsymhor = writeInArray(inputAllFile, &countOfString);

int countOfStringVer = writeInArrayVer(inputAllFile, countOfString);

//Create Buffer Arguments

ocl.srcA = clCreateBuffer(ocl.context, CL\_MEM\_READ\_ONLY | CL\_MEM\_USE\_HOST\_PTR, sizeof(char) \* COUNT\_STRING \* SIZE, inputAllFile, NULL);

ocl.srcB = clCreateBuffer(ocl.context, CL\_MEM\_READ\_WRITE | CL\_MEM\_USE\_HOST\_PTR, sizeof(int) \* SIZE, Res, NULL);

ocl.srcBVer = clCreateBuffer(ocl.context, CL\_MEM\_READ\_WRITE | CL\_MEM\_USE\_HOST\_PTR, sizeof(int) \* SIZE, ResVer, NULL);

ocl.srcCount = clCreateBuffer(ocl.context, CL\_MEM\_READ\_WRITE | CL\_MEM\_USE\_HOST\_PTR, sizeof(int), &countOfString, NULL);

CreateAndBuildProgram(&ocl);

ocl.kernel = clCreateKernel(ocl.program, "Add", &err);

//Set Kernel Arguments

clSetKernelArg(ocl.kernel, 0, sizeof(cl\_mem), (void \*)&ocl.srcA);

clSetKernelArg(ocl.kernel, 1, sizeof(cl\_mem), (void \*)&ocl.srcB);

clSetKernelArg(ocl.kernel, 2, sizeof(cl\_mem), (void \*)&ocl.srcBVer);

clSetKernelArg(ocl.kernel, 3, sizeof(cl\_mem), (void \*)&ocl.srcCount);

QueryPerformanceCounter(&performanceCountNDRangeStart);

ExecuteAddKernel(&ocl, countOfStringVer);

QueryPerformanceCounter(&performanceCountNDRangeStop);

QueryPerformanceFrequency(&perfFrequency);

LogInfo("NDRange performance counter time %f ms.\n",

1000.0f\*(float)(performanceCountNDRangeStop.QuadPart - performanceCountNDRangeStart.QuadPart) / (float)perfFrequency.QuadPart);

int Ver = 0;

int Hor = countOfsymhor - ReadAndVerify(&ocl, &Ver);

Ver = countOfsymhor - Ver;

printf("Hor %d\n", Hor);

printf("Ver %d\n", Ver);

if (Hor<Ver)

printf("min %d\n", Hor);

else

printf("min %d\n", Ver);

system("pause");

return 0;}

**Ядро:**

\_\_kernel void Add(\_\_global char pA[][30], \_\_global int pB[],\_\_global int pBVer[],\_\_global int \*c2)

{int c=\*c2;

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

while (pA[id][i] != '\n'&&pA[id][i]!='М')

{

if (pA[id][i] == '2')

i++;

else if (pA[id][i] == '1'){

int onec = 0;

while (pA[id][i] == '1')

{

onec++; i++;}

if (id>=c)

pBVer[onec]++;

else

pB[onec]++;

}

else

{

int zeroc = 0;

while (pA[id][i] == '0')

{

zeroc++; i++;

}

if (id>=c)

pBVer[zeroc]++;

else

pB[zeroc]++;

}

}

}