ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ

ВЫСШЕГО ОБРАЗОВАНИЯ

**БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ НАЦИОНАЛЬНЫЙ**

**ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ**

(НИУ «БелГУ»)

ИНСТИТУТ ИНЖЕНЕРНЫХ И ЦИФРОВЫХ ТЕХНОЛОГИЙ

КАФЕДРА МАТЕМАТИЧЕСКОГО И ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ ИНФОРМАЦИОННЫХ СИСТЕМ

**Отчет по лабораторной работе №4**

**по дисциплине: «Программирование графических процессоров (NVIDIA CUDA)»**

студента очного отделения

3 курса 12001801 группы

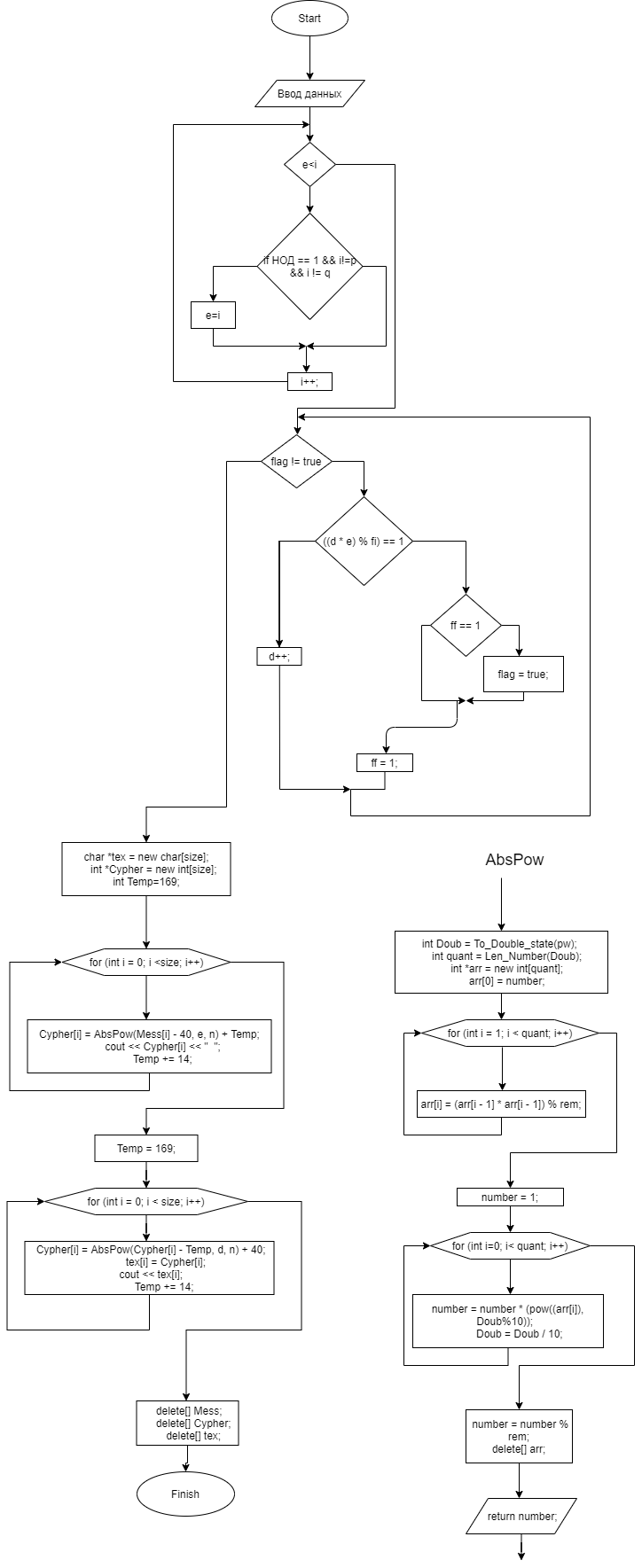
Капустина Виктора Сергеевича

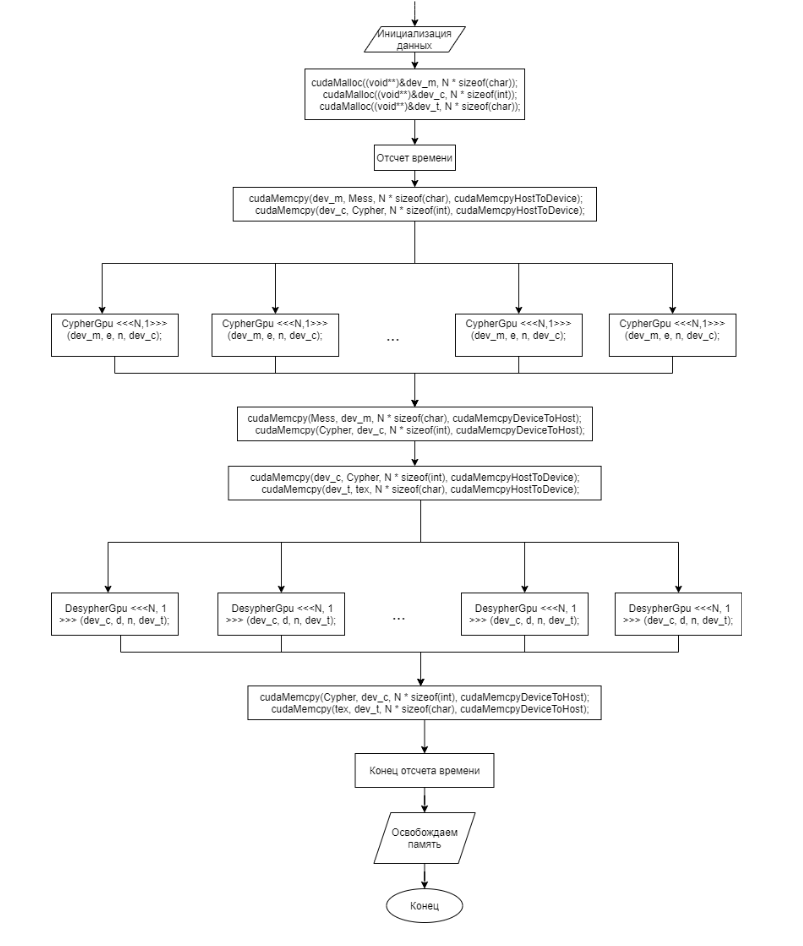
Проверил(а):

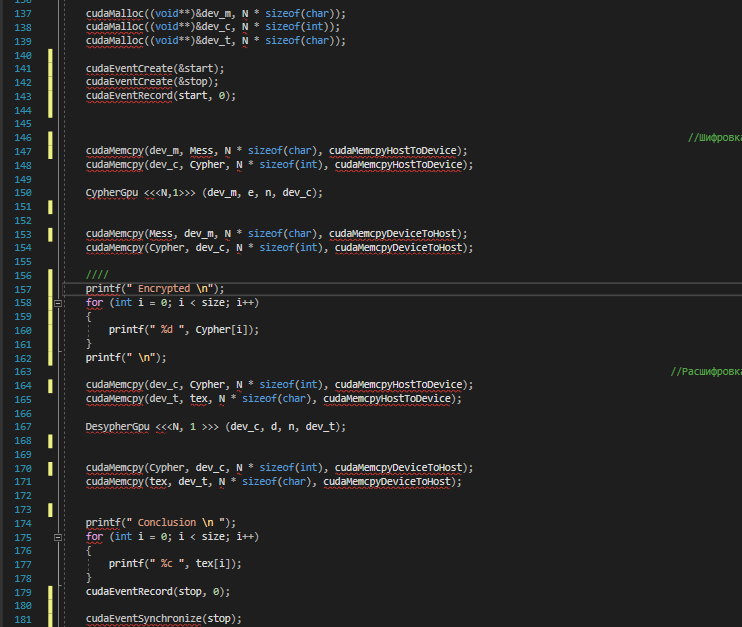
Петров Денис Васильевич

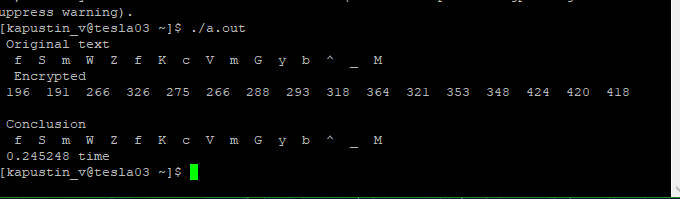
Белгород 2020

**Блок-схема**





  
Рис.1 Процесс разработки ПО

  
Рис.2 Работа программы в тестовом режиме

## 

## Таблица 1. Зависимость времени вычислений от объема исходных данных и количества задействованных узлов

| **Объем задачи** | Время расчета на CPU | Время расчета на GPU | Ускорение |
| --- | --- | --- | --- |
| **16** | 0.000000 | 0.00137760 | 0,000007 |
| **1600** | 0.000000 | 0.00245952 | 0,000008 |
| **16000** | 0.000000 | 0.01189152 | 0,000002 |
| **160000** | 0.000000 | 0.01883200 | 0,000002 |
| **1600000** | 0.240000 | 0.20255936 | 1,2 |
| **16000000** | 2.200000 | 1.40146240 | 1,57 |
| **160000000** | 20.040000 | 13.65425842 | 1,46 |
| **1600000000** | 200.380000 | 23.27707642 | 8,611 |

  
График 1. График зависимости времени от кол-ва расчетов

**Листинг программы**.

#include <iostream>

#include <cstdlib>

#include <cctype>

#include <string>

#include <bitset>

#include <cmath>

#include <time.h>

#include <stdio.h>

#include <ctime>

#include <iomanip>

#include <locale.h>

#include <stdlib.h>

#include <cuda\_runtime.h>

#include <cuda.h>

#define size 16 //Размер сообщения

int gcd(int x, int y)

{

return y ? gcd(y, x % y) : x;

}

/\*

void CPUC(char \*Mess, int e, int n, int \*Cypher)

{

int tid = 0;

while (tid < size)

{

int TValue = 169 + (tid \* 14);

int g = Mess[tid] - 40;

int result = 1;

int exponent = e;

while (exponent > 0)

{

if (exponent % 2 == 1)

{

result = (result \* g) % n;

}

exponent = exponent >> 1;

g = (g \* g) % n;

}

Cypher[tid] = result + TValue;

tid += 1;

}

}

void CPUD(int\* Cypher, int d, int n, char\* tex)

{

int tid = 0;

while (tid < size)

{

int TValue = 169 + (tid \* 14);

int g = Cypher[tid] - TValue;

int result = 1;

int exponent = d;

while (exponent > 0)

{

if (exponent % 2 == 1)

{

result = (result \* g) % n;

}

exponent = exponent >> 1;

g = (g \* g) % n;

}

Cypher[tid] = result + 40;

tex[tid] = Cypher[tid];

tid += 1;

}

}

\*/

\_\_global\_\_ void DesypherGpu(int \*Cypher, int d, int n, char \*tex)

{

int TValue = 169 + (blockIdx.x \* 14);

int g = Cypher[blockIdx.x] - TValue;

int result = 1;

int exponent = d;

while (exponent > 0)

{

if (exponent % 2 == 1)

{

result = (result \* g) % n;

}

exponent = exponent >> 1;

g = (g \* g) % n;

}

Cypher[blockIdx.x] = result + 40;

tex[blockIdx.x] = Cypher[blockIdx.x];

}

\_\_global\_\_ void CypherGpu(char \*Mess, int e, int n, int \*Cypher)

{

int TValue = 169 + (blockIdx.x \* 14);

int g = Mess[blockIdx.x] - 40;

int result = 1;

int exponent = e;

while (exponent > 0)

{

if (exponent % 2 == 1)

{

result = (result \* g) % n;

}

exponent = exponent >> 1;

g = (g \* g) % n;

}

Cypher[blockIdx.x] = result + TValue;

//printf("Hello from %d value= %d Tvalue= %d", blockIdx.x, Cypher[blockIdx.x], TValue);

}

int main()

{

// int size = 16;

//int N = size;

srand(time(0));

int p;

int q;

int n;

int fi;

int e = 0;

int rnd = rand() % 2;

//printf(" Original text \n ");

char\* Mess = new char[size];

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

{

Mess[i] = 65 + rand() % 57;

//printf(" %c ", Mess[i]);

}

printf("\n ");

switch (rnd)

{

case 0: p = 7; break;

case 1: p = 13; break;

}

q = 17;

n = abs(p \* q);

fi = (p - 1) \* (q - 1);

int i = 2;

while (e < 1) {

if ((gcd(i, fi) == 1) && (i != p) && (i != q))

{

e = i;

}

i++;

}

bool flag = false;

int ff = 0;

int d = 1;

while (flag != true)

{

if (((d \* e) % fi) == 1)

{

if (ff == 1)

{

flag = true;

}

ff = 1;

}

else d++;

}

char\* tex = new char[size];

int\* Cypher = new int[size];

/\*

double startTime = clock();

srand(time(NULL));

CPUC(Mess, e, n, Cypher); /

CPUD(Cypher, d, n, tex); //

uble endTime = clock();

double cpuTime = (endTime - startTime) / CLOCKS\_PER\_SEC \* 1000;

printf("\n %f time \n", cpuTime);

\*/

char \*dev\_m = 0;

char \*dev\_t = 0;

int \*dev\_c = 0;

cudaEvent\_t start, stop;

float gpuTime = 0.0f;

cudaMalloc((void\*\*)&dev\_m, size \* sizeof(char));

cudaMalloc((void\*\*)&dev\_c, size \* sizeof(int));

cudaMalloc((void\*\*)&dev\_t, size \* sizeof(char));

cudaEventCreate(&start);

cudaEventCreate(&stop);

cudaEventRecord(start, 0);

//Шифровка

cudaMemcpy(dev\_m, Mess, size \* sizeof(char), cudaMemcpyHostToDevice);

cudaMemcpy(dev\_c, Cypher, size \* sizeof(int), cudaMemcpyHostToDevice);

CypherGpu <<<size,1>>> (dev\_m, e, n, dev\_c);

cudaMemcpy(Mess, dev\_m, size \* sizeof(char), cudaMemcpyDeviceToHost);

cudaMemcpy(Cypher, dev\_c, size \* sizeof(int), cudaMemcpyDeviceToHost);

////

/\* printf(" Encrypted \n");

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

{

printf(" %d ", Cypher[i]);

}

printf(" \n");\*/

//Расшифровка

cudaMemcpy(dev\_c, Cypher, size \* sizeof(int), cudaMemcpyHostToDevice);

cudaMemcpy(dev\_t, tex, size \* sizeof(char), cudaMemcpyHostToDevice);

DesypherGpu <<<size, 1 >>> (dev\_c, d, n, dev\_t);

cudaMemcpy(Cypher, dev\_c, size \* sizeof(int), cudaMemcpyDeviceToHost);

cudaMemcpy(tex, dev\_t, size \* sizeof(char), cudaMemcpyDeviceToHost);

/\*/printf(" Conclusion \n ");

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

{

printf(" %c ", tex[i]);

}\*/

cudaEventRecord(stop, 0);

cudaEventSynchronize(stop);

cudaEventElapsedTime(&gpuTime, start, stop);

printf("\n %f time \n", gpuTime);

cudaFree(dev\_m);

cudaFree(dev\_t);

cudaFree(dev\_c);

return 1;

}