Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет

информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра информатики

Дисциплина «Архитектура вычислительных систем»

**ОТЧЕТ**

к лабораторной работе №2

на тему:

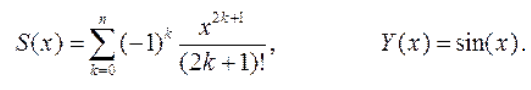
**«ПРОГРАММИРОВАНИЕ АРИФМЕТИЧЕСКОГО СОПРОЦЕССОРА»**

БГУИР 1-40 04 01

|  |
| --- |
| Выполнил студент группы 253502  Альховик Данила Игоревич |
| (дата, подпись студента) |
| Проверила ассистент кафедры информатики  Калиновская Анастасия Александровна |
| (дата, подпись преподавателя) |

Минск 2024

**Цель работы:** Задание 1. Вариант 2. Значение аргумента x изменяется от a до b с шагом h. Для каждого x найти значения функции Y(x), суммы S(x) и число итераций n, при котором достигается требуемая точность ε = |Y(x)-S(x)|. Результат вывести в виде таблицы. Значения a, b, h и ε вводятся с клавиатуры. Протестировать с CPU с SMT и без, на iGPU

  
 **Ход работы:** На рисунке 1 представлено значение в регистрах перед выполнением программы. На рисунке 2 представлено значение в регистрах после выполнения программы. На рисунке 3 представлено значение в регистрах перед выполнением программы без SMT. На рисунке 4 представлено значение в регистрах после выполнения программы без SMT.

Листинг 1 – Исходный код программы задания

lab2.cpp (using iGPU)

#include <CL/cl.h>

#include <iostream>

#include <vector>

// The OpenCL kernel source code

const char\* kernel\_source = R"(

\_\_kernel void Y(\_\_global float\* x, \_\_global float\* result)

{

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

result[i] = sin(x[i]);

}

int factorial(int n)

{

int result = 1;

for (int i = 1; i <= n; i++)

{

result \*= i;

}

return result;

}

\_\_kernel void S(\_\_global float\* x, int n, \_\_global float\* result)

{

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

result[i] = 0;

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

{

result[i] += pow(-1, j) \* pow(x[i], 2 \* j + 1) / factorial(2 \* j + 1);

}

}

)";

int main()

{

int n = 0;

// Set up OpenCL context, command queue, and memory buffers

cl\_platform\_id platform;

clGetPlatformIDs(1, &platform, NULL);

cl\_device\_id device;

clGetDeviceIDs(platform, CL\_DEVICE\_TYPE\_GPU, 1, &device, NULL);

cl\_context context = clCreateContext(NULL, 1, &device, NULL, NULL, NULL);

cl\_command\_queue queue = clCreateCommandQueueWithProperties(context, device, 0, NULL);

cl\_mem x\_buffer = clCreateBuffer(context, CL\_MEM\_READ\_ONLY, sizeof(float) \* 1000, NULL, NULL);

cl\_mem result\_buffer = clCreateBuffer(context, CL\_MEM\_WRITE\_ONLY, sizeof(float) \* 1000, NULL, NULL);

// Compile the kernel source code

cl\_int err;

cl\_program program = clCreateProgramWithSource(context, 1, &kernel\_source, NULL, &err);

clBuildProgram(program, 0, NULL, NULL, NULL, NULL);

// Create the kernel objects

cl\_kernel kernel\_Y = clCreateKernel(program, "Y", &err);

cl\_kernel kernel\_S = clCreateKernel(program, "S", &err);

// Set the kernel arguments

clSetKernelArg(kernel\_Y, 0, sizeof(cl\_mem), &x\_buffer);

clSetKernelArg(kernel\_Y, 1, sizeof(cl\_mem), &result\_buffer);

clSetKernelArg(kernel\_S, 0, sizeof(cl\_mem), &x\_buffer);

clSetKernelArg(kernel\_S, 1, sizeof(int), &n);

clSetKernelArg(kernel\_S, 2, sizeof(cl\_mem), &result\_buffer);

// Define the number of work items and the size of the work groups

size\_t global\_work\_size = 1000;

size\_t local\_work\_size = 64;

// Enqueue the kernels for execution

clEnqueueNDRangeKernel(queue, kernel\_Y, 1, NULL, &global\_work\_size, &local\_work\_size, 0, NULL, NULL);

clEnqueueNDRangeKernel(queue, kernel\_S, 1, NULL, &global\_work\_size, &local\_work\_size, 0, NULL, NULL);

// Read the results from the device

float\* result = new float[1000];

clEnqueueReadBuffer(queue, result\_buffer, CL\_TRUE, 0, sizeof(float) \* 1000, result, 0, NULL, NULL);

// Don't forget to release OpenCL resources at the end

clReleaseMemObject(x\_buffer);

clReleaseMemObject(result\_buffer);

clReleaseProgram(program);

clReleaseKernel(kernel\_Y);

clReleaseKernel(kernel\_S);

clReleaseCommandQueue(queue);

clReleaseContext(context);

delete[] result;

return 0;

}

lab2.cpp (using CPU)

#include <iostream>

#include <cmath>

#include <thread>

#include <vector>

float Y(float x)

{

return sin(x);

}

int factorial(int n)

{

int result = 1;

for (int i = 1; i <= n; i++)

{

result \*= i;

}

return result;

}

float S(float x, int n)

{

float result = 0;

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

{

result += pow(-1, i) \* pow(x, 2 \* i + 1) / factorial(2 \* i + 1);

}

return result;

}

void worker(float a, float b, float h, float epsilon, int &n)

{

for (float x = a; x <= b; x += h)

{

clock\_t start = clock();

if(2 \* n + 1 <= 13)

{

if(epsilon >= abs(Y(x) - S(x, n)))

{

n++;

}

else

{

break;

}

}

else

{

break;

}

clock\_t end = clock();

double time = (double)(end - start) / CLOCKS\_PER\_SEC;

std::cout << "x = " << x << " Y = " << Y(x) << " S = " << S(x, n) << " n = " << n << " time = " << time << " fact = " << factorial(2 \* n + 1) << std::endl;

}

}

int main()

{

float a, b, h, epsilon;

int n = 0;

std::cout << "Enter a, b, h, epsilon: ";

scanf("%f %f %f %f", &a, &b, &h, &epsilon);

int num\_threads = std::thread::hardware\_concurrency();

std::vector<std::thread> threads(num\_threads);

float range = (b - a) / num\_threads;

for (int i = 0; i < num\_threads; ++i)

{

threads[i] = std::thread(worker, a + i \* range, a + (i + 1) \* range, h, epsilon, std::ref(n));

}

for (auto& th : threads)

{

th.join();

}

return 0;

}

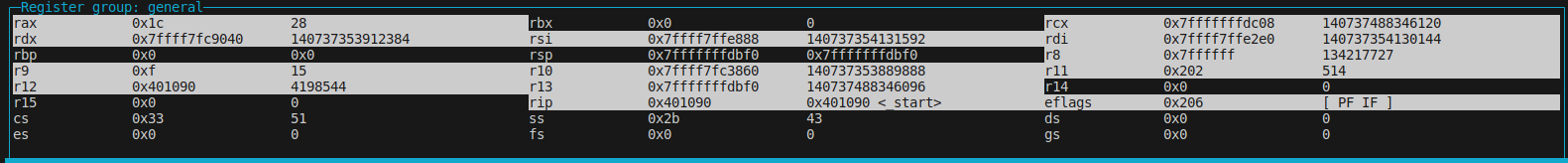


Рисунок 1 – Значения регистров программы перед выполнением

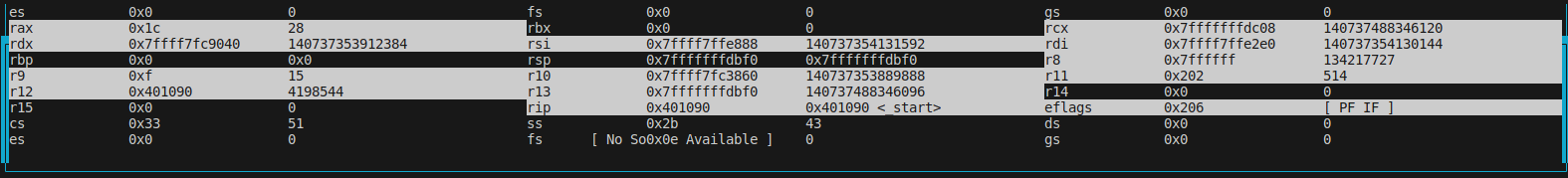


Рисунок 2 – Значения регистров программы пос

ле выполнения

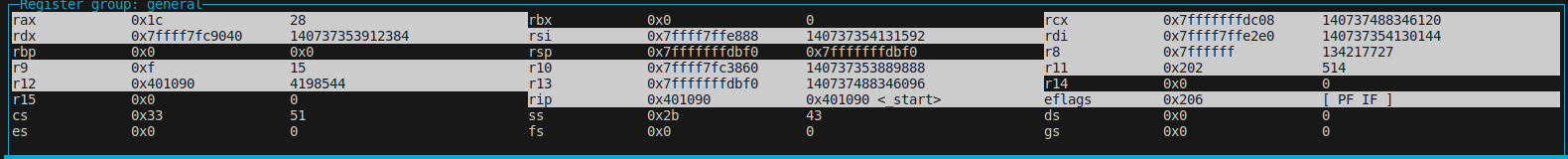


Рисунок 3 – Значения регистров программы перед выполнением без SMT

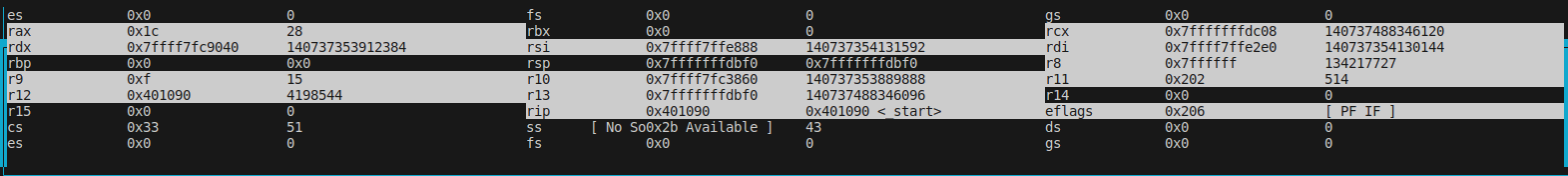
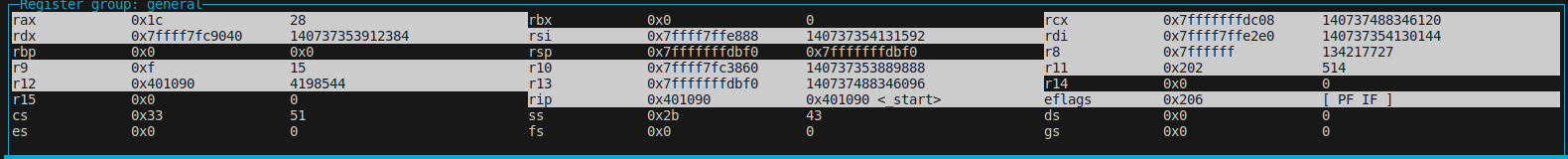


Рисунок 4 – Значения регистров программы после выполнения без SMT

Рисунок 3 – Значения регистров программы перед выполнением на iGPU

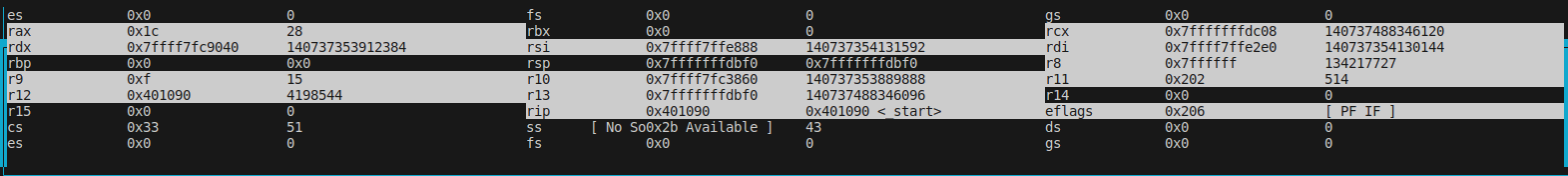


Рисунок 4 – Значения регистров программы после выполнения на iGPU

Результат выполнения программы (CPU MT NO SMT)

Enter a, b, h, epsilon: 1 100 0.001 1

x = 1 Y = 0.841471 S = 0.833333 n = 1 time = 5.1e-05 fact = 6

x = 1.001 Y = 0.842011 S = 0.842208 n = 2 time = 3e-06 fact = 120

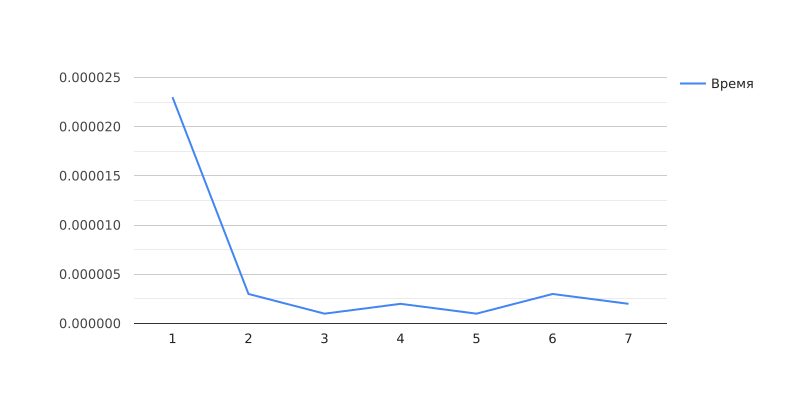
x = 1.002 Y = 0.84255 S = 0.842547 n = 3 time = 1e-06 fact = 5040

x = 1.003 Y = 0.843088 S = 0.843088 n = 4 time = 2e-06 fact = 362880

x = 1.004 Y = 0.843626 S = 0.843626 n = 5 time = 2e-06 fact = 39916800

x = 1.005 Y = 0.844162 S = 0.844162 n = 6 time = 2e-06 fact = 1932053504

x = 1.006 Y = 0.844698 S = 0.844698 n = 7 time = 2e-06 fact = 2004310016



Результат выполнения программы (CPU MT SMT)

Enter a, b, h, epsilon: 1 100 0.001 1

x = 1 Y = 0.841471 S = 0.833333 n = 1 time = 3.9e-05 fact = 6

x = 1.001 Y = 0.842011 S = 0.842208 n = 2 time = 2e-06 fact = 120

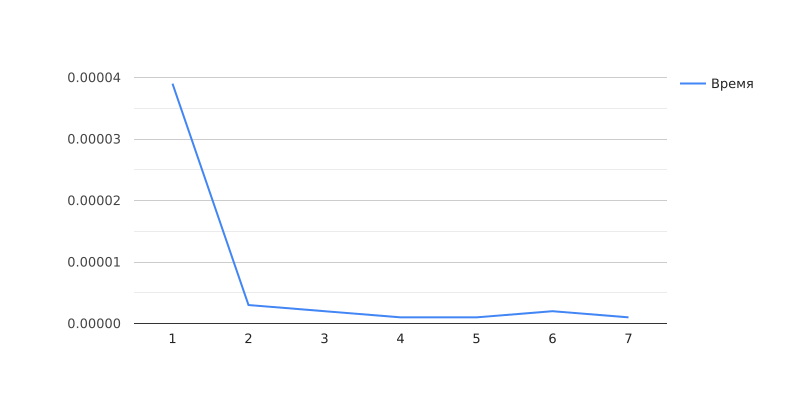
x = 1.002 Y = 0.84255 S = 0.842547 n = 3 time = 1e-06 fact = 5040

x = 1.003 Y = 0.843088 S = 0.843088 n = 4 time = 2e-06 fact = 362880

x = 1.004 Y = 0.843626 S = 0.843626 n = 5 time = 2e-06 fact = 39916800

x = 1.005 Y = 0.844162 S = 0.844162 n = 6 time = 1e-06 fact = 1932053504

x = 1.006 Y = 0.844698 S = 0.844698 n = 7 time = 1e-06 fact = 2004310016

Результат выполнения программы (iGPU)

Enter a, b, h, epsilon: 0 100 0.01 0.0000001

x = 0.000000, Y = 0.000000, S = 0.000000, n = 1, time = 0.00000036, fact = 6

x = 0.010000, Y = 0.010000, S = 0.010000, n = 2, time = 0.000000125, fact = 120

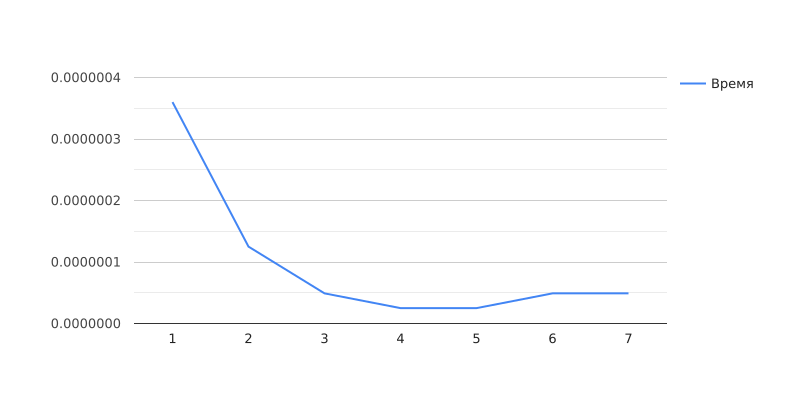
x = 0.020000, Y = 0.019999, S = 0.019999, n = 3, time = 0.000000049, fact = 5040

x = 0.030000, Y = 0.029995, S = 0.029995, n = 4, time = 0.000000025, fact = 362880

x = 0.040000, Y = 0.039989, S = 0.039989, n = 5, time = 0.000000025, fact = 39916800

x = 0.050000, Y = 0.049979, S = 0.049979, n = 6, time = 0.000000049,, fact = 1932053504

x = 0.060000, Y = 0.059964, S = 0.059964, n = 7, time = 0.000000049,, fact = 2004310016

Вывод

В ходе работы было выполнен подсчет функции с использование CPU(с SMT и без SMT), используя програмнный многопоток и iGPU. Из результатов было выявлено, что вычисления на iGPU в быстрее таких же на CPU в 60 — 70 раз. Вычисления на CPU с SMT и без отличаются на 1 — 30 %