МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ

(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)

Кафедра вычислительной математики и программирования

**спецкурс «Параллельные и распределенные вычисления»**

**ОТЧЕТ**

**Лабораторная работа № 2**

**«Использование различных типов памяти»**

Выполнил: Каратаева Е.С.

Группа: М8О-114М-22, **вар. 9**

Преподаватель: Семенов С. А.

Москва, 2022

Содержание

[1. Постановка задачи 2](file:///C:\Users\simpl\Downloads\КаратаеваЕ.С._ЛР1_М8О-114М-22.docx#_Toc52570380)

[2. Описание решения 2](file:///C:\Users\simpl\Downloads\КаратаеваЕ.С._ЛР1_М8О-114М-22.docx#_Toc52570381)

[3. Аппаратное обеспечение и ПО 2](file:///C:\Users\simpl\Downloads\КаратаеваЕ.С._ЛР1_М8О-114М-22.docx#_Toc52570382)

[4. Основные моменты кода 2](file:///C:\Users\simpl\Downloads\КаратаеваЕ.С._ЛР1_М8О-114М-22.docx#_Toc52570383)

[5. Результат работы программы 3](file:///C:\Users\simpl\Downloads\КаратаеваЕ.С._ЛР1_М8О-114М-22.docx#_Toc52570384)

[6. Сравнение скорости выполнения на CPU и GPU 7](file:///C:\Users\simpl\Downloads\КаратаеваЕ.С._ЛР1_М8О-114М-22.docx#_Toc52570385)

[7. Выводы 8](file:///C:\Users\simpl\Downloads\КаратаеваЕ.С._ЛР1_М8О-114М-22.docx#_Toc52570386)

[8. Приложения 8](file:///C:\Users\simpl\Downloads\КаратаеваЕ.С._ЛР1_М8О-114М-22.docx#_Toc52570387)

# 1. Постановка задачи

Вариант 9. Найти простые множители натурального числа n.

# 2. Описание решения

При нахождении простых множителей натурального числа использовалась написанная функция find\_multipliers () - ей на вход подаётся число N.

# 3. Аппаратное обеспечение и ПО

|  |  |
| --- | --- |
| Видеокарта | Nvidia Geforce RTX 3060 6Gb |
| Процессор | Intel Core i7-11800H |
| IDE | Visual Studio 2022 |
| OC | Windows 10 Pro 21H2 |

# 4. Основные моменты кода

Исходный код представлен в приложении А. В функции main идёт вызов и функции расчёта на GPU и на CPU. Время, занимаемое расчётами, выводится в конце после всех рассчитанных элементов.

А именно, функция my\_find предназначена для расчетов на CPU. Функции kernel – на GPU. Код представлен при n = 10000.

# 5. Результат работы программы



**Рис.2.** Вывод программы при N=4.



**Рис.3.** Вывод программы при N=8.



**Рис.4.** Вывод программы при N=16.



**Рис.5.** Вывод программы при N=32.



**Рис.6.** Вывод программы при N=64.



**Рис.7.** Вывод программы при N=128.



**Рис.8.** Вывод программы при N=256.



**Рис.9.** Вывод программы при N=512.



**Рис.10.** Вывод программы при N=1024.



**Рис.11.** Вывод программы при N=10000.

# 6. Сравнение скорости выполнения на CPU и GPU

При запуске программы с различными значениями *N* видно, что вычисления на видеокарте произвелись быстрее, чем на процессоре компьютера, при *N*>16.

Время выполнения программы при различных значениях *N*:

|  |  |  |  |
| --- | --- | --- | --- |
| **N** | **GPU**  **время выполнения, мс** | **CPU**  **время выполнения, мс** | **tCPU/tGPU** |
| 4 | 0,102400 | 0 | 0 |
| 8 | 0,179200 | 0 | 0 |
| 16 | 0,297984 | 3 | 10,068 |
| 32 | 0,423936 | 7 | 16,512 |
| 64 | 0,592896 | 42 | 70,839 |
| 128 | 0,817152 | 57 | 69,754 |
| 256 | 1,478528 | 164 | 110,92 |
| 512 | 1,857408 | 499 | 268,65 |
| 1024 | 4,357120 | 566 | 129,9 |
| 10000 | 60,176319 | 2710 | 45,034 |

**Рис.12.** График зависимости времени выполнения программы от числа *N*.

# 7. Выводы

В Лабораторной работе №2 проведен анализ работы различных программ по решению задачи нахождения простых множителей натурального числа n.

Во второй части Лабораторной работы было проведено сравнение времени выполнения программ на CPU и GPU.

# 8. Приложения

Приложение А. Исходный код kernel.cu.

#include "cuda\_runtime.h"

#include "device\_launch\_parameters.h"

#include <stdio.h>

#include <stdlib.h>

#include <math.h>

#include <time.h>

static void CatchError(cudaError\_t err, const char\* file, int line) {

if (err != cudaSuccess) {

printf("%s in %s at line %d\n", cudaGetErrorString(err),

file, line);

exit(EXIT\_FAILURE);

}

}

#define CATCH\_ERROR( err ) (CatchError( err, \_\_FILE\_\_, \_\_LINE\_\_ ))

int\* find\_multipliers(long long int N)

{

int k;

auto temp = new int[1000];

for (int i = 2; i \* i <= N; i++)

{

// if composite number:

if (N % i == 0)

{

temp[0] = i;

printf("%d^", i);

// calc how many same numbers in composite num

for (k = 0; N % i == 0; k++)

{

temp[k + 1] = k;

N /= i;

}

printf("%d ", k);

}

}

if (N > 1) {

temp[0] = N;

printf("%d ", N);

};

printf("\n");

return temp;

}

// func for CPU calculations

void my\_find(int\*\* res, long long int n) {

long long int tid = 0;

while (tid<n)

{

res[tid] = find\_multipliers(tid);

tid++; }

}

\_\_device\_\_ int\* find\_multipliers\_gpu(long long int N)

{

int k;

int\* temp = (int\*)malloc(100);

for (int i = 2; i \* i <= N; i++)

{

// if composite number:

if (N % i == 0)

{

printf("%d^", i);

temp[0] = i;

// calc how many same numbers in composite num

for (k = 0; N % i == 0; k++)

{

N /= i;

}

temp[k + 1] = k;

printf("%d ", k);

}

}

if (N > 1)

{

temp[0] = N;

printf("%d ", N);

}

printf("\n");

return temp;

}

\_\_global\_\_ void kernel(int \*\*res, long long int n)

{

long long int tid = blockIdx.x \* blockDim.x + threadIdx.x;

long long int offset = blockDim.x \* gridDim.x;

while (tid < n) {

res[tid] = find\_multipliers\_gpu(tid);

tid += offset;

}

}

int main()

{

long long int n = 10000;

int count;

CATCH\_ERROR(cudaGetDeviceCount(&count));

if (count == 0) {

printf("there is no cuda device");

return -1;

}

// for timer

cudaEvent\_t time\_of\_start, time\_of\_end;

float res\_timer\_gpu;

CATCH\_ERROR(cudaEventCreate(&time\_of\_start));

CATCH\_ERROR(cudaEventCreate(&time\_of\_end));

int\*\* res = (int\*\*)malloc(n \* 1000 \* sizeof(int\*));

int\*\* res\_dev;

CATCH\_ERROR(cudaMalloc(&res\_dev, n \* 1000 \* sizeof(int\*)));

CATCH\_ERROR(cudaEventRecord(time\_of\_start)); // start of timer (GPU)

kernel << <256, 256 >> > (res\_dev, n);

CATCH\_ERROR(cudaEventRecord(time\_of\_end)); // end of timer (GPU)

CATCH\_ERROR(cudaEventSynchronize(time\_of\_end));

CATCH\_ERROR(cudaEventElapsedTime(&res\_timer\_gpu, time\_of\_start, time\_of\_end));

CATCH\_ERROR(cudaEventDestroy(time\_of\_start));

CATCH\_ERROR(cudaEventDestroy(time\_of\_end));

CATCH\_ERROR(cudaMemcpy(res, res\_dev, n \* sizeof(int), cudaMemcpyDeviceToHost));

CATCH\_ERROR(cudaFree(res\_dev));

free(res);

res = (int\*\*)malloc(n \* n \* sizeof(int));

double res\_timer\_cpu = 0.0;

clock\_t begin = clock();

my\_find(res, n);

clock\_t end = clock();

res\_timer\_cpu += (double)(end - begin) / CLOCKS\_PER\_SEC;

printf("Results by CPU: %f\n", res\_timer\_cpu \* 1000);

printf("Results by GPU: %f\n", res\_timer\_gpu);

free(res);

return 0;

}