Московский авиационный институт
(национальный исследовательский университет)

Институт №8 «Информационные технологии и прикладная математика»

Кафедра 806 «Вычислительная математика и программирование»

Лабораторная работа №1 по курсу «Программирование графических процессоров»

Изучение технологии CUDA

Студент: Л. Я. Вельтман
Преподаватель: К. Г. Крашенинников
А. Ю. Морозов
Группа: М8О-407Б
Дата:
Оценка:
Подпись:

Москва, 2020

Условие

Задача: Цель работы: Ознакомление и установка программного обеспечения для работы с программно-аппаратной архитектурой параллельных вычислений(CUDA). Реализация одной из примитивных операций над векторами.

Вариант 8: Реверс вектора.
Входные данные: данные. На первой строке задано число n – размер векторов. На следующей строке записано n вещественных чисел – элементы вектора.
Выходные данные: Необходимо вывести n чисел – результат реверса исходного вектора.
Программное и аппаратное обеспечение:
Device Number: 0
Device name: GeForce GT 545
TotalGlobalMem: 3150381056
Const Mem : 65536
Max shared mem for blocks 49152
Max regs per block 32768
Max thread per block 1024
multiProcessorCount : 3
maxThreadsDim 1024 1024 64
maxGridSize 65535 65535 65535
OS: macOS Catalina version 10.15.5
Text Editor: Sublime Text 3

1 Описание

Метод решения
Каждая последующая нить - это индекс для нового массива (idx), в который мы будем записывать элементы исходного массива, но в обратном порядке (size - idx - 1).

Описание программы
Введем данные для массива в оперативную память. Затем перенесем их на видеопамять. Также, выделим память на графическом устройстве под результат реверса исходного массива. Для реверса вектора запускаем ядро с заранее заданным количеством блоков и количеством нитей в каждом из них, далее в ядре нужно записать в результирующий массив нужный элемент исходного массива по идентификатору. В ядре(Reverse) вычисляется индекс исполняемой нити. Он будет индексом для обратного массива. Также вычисляется индекс для прохода по исходному массиву (size - ixd - 1), он позволит брать элементы массива с конца. После вычисления, для обработки результатов, перенесим данные с видеопамяти в оперативную память оперативную. Выводим полученный массив.

2 Исходный код

```
#include <iostream>
#include <iomanip>
#include <stdio.h>
#include <stdlib.h>


void checkCudaError(const char* msg)
{
    cudaError_t err = cudaGetLastError();
    if (cudaSuccess != err)
    {
        fprintf(stderr, "ERROR: %s: %s.\n", msg, cudaGetErrorString(err));
        exit(0);
    }
}


__global__ void Reverse(float* res, float* vec, int size)
{
    int idx = blockDim.x * blockIdx.x + threadIdx.x;
    int offset = gridDim.x * blockDim.x;

    while (idx < size)
    {
        res[idx] = vec[size - idx - 1];
        idx += offset;
    }
}


int main(int argc, const char* argv[])
{
    int size;
    std::cin >> size;

    const int MAX = 33554432;
    const int MIN = 0;
    if (size < MIN && size > MAX)
    {
       std::cerr << "ERROR: Incorrect size!\n";
       exit(0);
    }

    float *hostVec = new float[size];

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

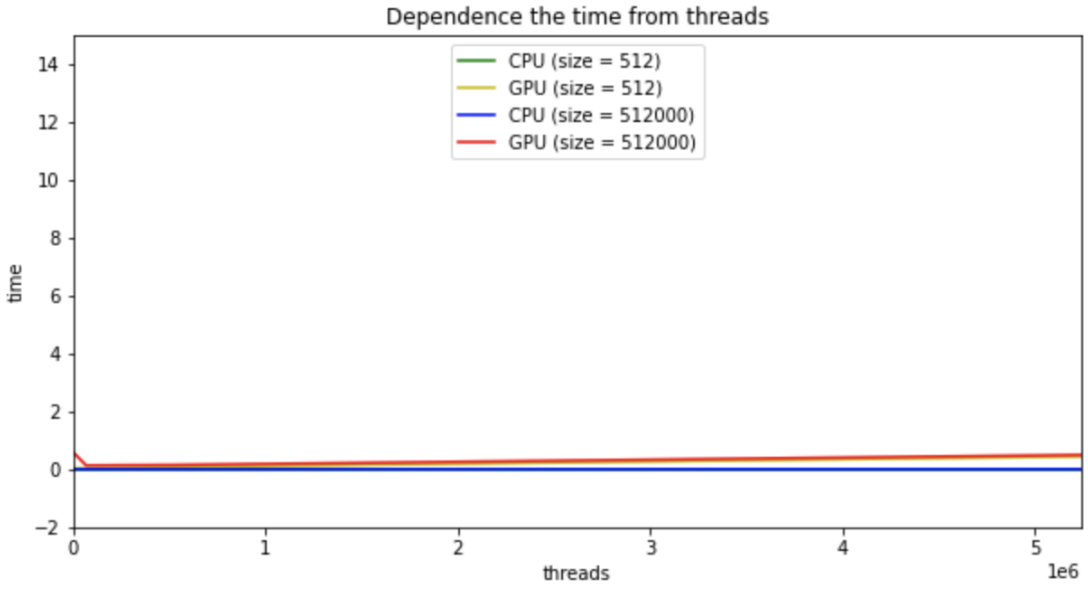
```
48          std::cin >> hostVec[i];
49      }
50
51      float *deviceVec, *deviceRes;
52
53      cudaMalloc((void**) &deviceVec, sizeof(float) * size);
54      checkCudaError("Malloc");
55
56      cudaMalloc((void**) &deviceRes, sizeof(float) * size);
57      checkCudaError("Malloc");
58
59      cudaMemcpy(deviceVec, hostVec, sizeof(float) * size, cudaMemcpyHostToDevice);
60      checkCudaError("Memcpy");
61
62      int blockCount = 256;
63      int threadsCount = 256;
64
65      Reverse<<<blockCount, threadsCount>>>(deviceRes, deviceVec, size);
66      checkCudaError("Kernel invocation");
67
68      cudaMemcpy(hostVec, deviceRes, sizeof(float) * size, cudaMemcpyDeviceToHost);
69      checkCudaError("Memcpy");
70
71      const int accuracy = 10;
72      for (int i = 0; i < size - 1; ++i)
73      {
74          std::cout << std::scientific << std::setprecision(accuracy) << hostVec[i] << "
                ";
75      }
76      std::cout << std::scientific << std::setprecision(accuracy) << hostVec[size - 1];
77
78      cudaFree(deviceVec);
79      checkCudaError("Free");
80
81      cudaFree(deviceRes);
82      checkCudaError("Free");
83
84      delete[] hostVec;
85
86      return 0;
87  }
```

3 Результаты

```
vector size = 512
CPU
time = 1.2e-05
GPU
time = 0.029376
blocks = 32
threads = 32

vector size = 512000
CPU
time = 0.004143
GPU
time = 0.572416
blocks = 32
threads = 32

vector size = 512
CPU
time = 1.3e-05
GPU
time = 0.034624
blocks = 256
threads = 256

vector size = 512000
CPU
time = 0.004002
GPU
time = 0.133408
blocks = 256
threads = 256

vector size = 512
CPU
time = 1.1e-05
GPU
time = 0.039680
blocks = 256
threads = 512
```

```
vector size = 512000
CPU
time = 0.004019
GPU
time = 0.134240
blocks = 256
threads = 512

vector size = 512
CPU
time = 1.3e-05
GPU
time = 0.047200
blocks = 512
threads = 512

vector size = 512000
CPU
time = 0.004025
GPU
time = 0.139936
blocks = 512
threads = 512

vector size = 512
CPU
time = 1.5e-05
GPU
time = 0.068576
blocks = 1024
threads = 512

vector size = 512000
CPU
time = 0.004007
GPU
time = 0.149120
blocks = 1024
threads = 512
```

```
vector size = 512
CPU
time = 1.2e-05
GPU
time = 0.433920
blocks = 10240
threads = 512

vector size = 512000
CPU
time = 0.004084
GPU
time = 0.495072
blocks = 10240
threads = 512
```



4 Выводы

Для выполнения данной лабораторной работы мне потребовалось познакомиться с новой для меня технологией CUDA, с помощью которой можно осуществлять параллельное программирование на видеокарте. Изучила базисные функции CUDA такие как выделение памяти на видеокарте для device копий, копирование данных на device и т.д. Я научилась реализовывать одну из примитивных операций над векторами: параллельный реверс вектора, используя технологию CUDA. Наибольшая производительность, судя по полученным данным времени работы обоих алгоритмов (на CPU и GPU), была достигнута обычным не параллельным алгоритмом на CPU. Выигрыш на графическом процессоре можно получить только при действительно больших данных. Так что для данной задачи лучше производить расчеты на процессоре. Отладка программы происходила в Google Colab.