Федеральное государственное бюджетное образовательное учреждение высшего образования

«Уфимский Государственный Авиационный Технический Университет»

Кафедра ВВТиС

Отчет по дисциплине «Суперкомпьютерные технологии» по лабораторной работе №5

Выполнила студентка группы ПРО

Проверил старший преподаватель

Юлдашев А.В.

Уфа 2017

**Задание**

На примере операций линейной алгебры научиться использовать:

- CUDA Streams,

- функции асинхронного копирования и pinned-память.

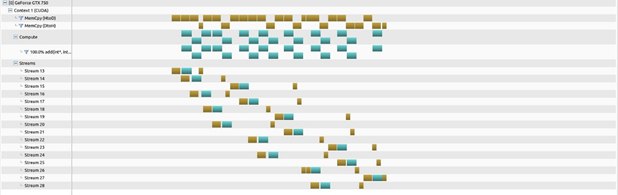
**Практическая часть**

Замер времен для различных размерностей N:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Число CUDA потоков | 1 | 2  Оптим. | 4 | 8 | 16 | 32 |
| N=10^5 | 0.2161 | 0.20649 | 0.22723 | 0.2566 | 0.30976 | 0.4155 |
| N=10^6 | 2.02627 | 1.8349 | 1.85286 | 1.8885 | 1.95968 | 2.0879 |
| N=10^7 | 20.1544 | 18.1056 | 18.17408 | 18.2881 | 18.6367 | 18.6104 |
| N=10^8 | 198.5711 | 180.8028 | 181.3937 | 182.27956 | 184.3655 | 183.7630 |
| N=MAX  (145000000) | 287.7947 | 262.4351 | 263.2625 | 264.9980 | 265.7744 | 267.1520 |

График зависимости времени показывает, что при увеличении размерности N время расчета увеличивается.

Информация, собранная NVIDIA Visual Profiler:



Можно сделать вывод, что выполнение функции ядра и копирование с хоста на девайс и с девайса на хост перекрываются между собой.

**Заключение**

Оптимальным является 2 CUDA потока, поскольку оно имеет минимальное значение среди всех других вариаций количества потоков независимо от различных размерностей N.

Листинг разработанной программы:

#include "cuda\_runtime.h"

#include "device\_launch\_parameters.h"

#include<iostream>

#include<ctime>

#include <stdio.h>

using namespace std;

#define N 100000

#define M 1024 // THREADS\_PER\_BLOCK

#define streamCount 32

\_\_global\_\_ void add(int \*a, int \*b, int \*c) //выполняется на GPU, вызывается с CPU

{

int index = threadIdx.x + blockIdx.x \* blockDim.x; // 1 - индекс текущей нити в вычислении на GPU, 2 - индекс текущего блока в вычислении на GPU, 3 - размерность блока

if (index < N) c[index] = a[index] + b[index];

}

//функция заполнения массива случайными числами

void random\_ints(int\* arr, int n) {

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

arr[i] = rand() % 100;

}

}

//проверка результатов

void checkResults(int\* a, int\* b, int\* c) {

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

if (a[i] + b[i] != c[i]) {

cout << "Wrong!" << endl;//

//return;

}

}

//cout << "Right" << endl;

}

int main(void) {

srand(time(NULL));

int \*a, \*b, \*c; //выделяем память под вектора

int \*d\_a, \*d\_b, \*d\_c; //указатели на память видеокарте

int size = N\*sizeof(int);

cudaMalloc((void\*\*)&d\_a,size); //выделяем память для векторов на видеокарте

cudaMalloc((void\*\*)&d\_b,size);

cudaMalloc((void\*\*)&d\_c,size);

cudaMallocHost((void\*\*)&a, size); //страницы вирт. памяти 100% окажутся в опер. памяти

cudaMallocHost((void\*\*)&b, size);

cudaMallocHost((void\*\*)&c, size);

random\_ints(a, N);

random\_ints(b, N);

cudaStream\_t\* streams = new cudaStream\_t[streamCount]; // поток, в котором будет произведен вызов

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

cudaStreamCreate(&streams[i]);

cudaEvent\_t start, stop;

float time = 0;

cudaEventCreate(&start);

cudaEventCreate(&stop);

cudaEventRecord(start, 0);

int width = N / streamCount;

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

{

int offset = i\*width;

cudaMemcpyAsync(&d\_a[offset], &a[offset], size/streamCount, cudaMemcpyDefault, streams[i]);

cudaMemcpyAsync(&d\_b[offset], &b[offset], size/streamCount, cudaMemcpyDefault, streams[i]);

add <<<(N / streamCount + M - 1) / M, M,0,streams[i]>>>(&d\_a[offset], &d\_b[offset], &d\_c[offset]);

cudaMemcpyAsync(&c[offset], &d\_c[offset], size/streamCount, cudaMemcpyDefault, streams[i]);

}

cudaEventRecord(stop, 0);

cudaEventSynchronize(stop);

cudaEventElapsedTime(&time, start, stop);

printf("Elapsed time : %f\n", time);

cout << endl;

checkResults(a, b, c);

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

cudaStreamDestroy(streams[i]);

cudaFreeHost(a);

cudaFree(d\_a);

cudaFreeHost(b);

cudaFree(d\_b);

cudaFreeHost(c);

cudaFree(d\_c);

//delete a; delete b; delete c;

return 0;

}