Донецкий Национальный Технический Университет

Лабораторная работа № 1

«Знакомство с GPCPU на базе технологии Nvidia Cuda»

Выполнил:

ст. группы ИПОИм -17

Лысенко А. С.

Проверила:

доцент каф. ПМИ

Назарова И. А.

Покровск 2017

**Задание к лабораторной работе**

1. Установить и настроить среды Cuda на рабочей машине.

2. Модифицировать код программы "Cuda: Hello, world!" для вывода ядром в файл фразы "I am from N block, M thread (global index: K)", где параметры N, M, K имеют соответствующие значения.

3. Создать и скомпилировать проект, который выводит параметры среды Cuda, объяснить значения соответствующих параметров.

4. Ознакомиться и описать примеры приведены в GPU Computing SDK.

2. Модифицированный код программы "Cuda: Hello, world!" для вывода ядром в файл фразы "I am from N block, M thread (global index: K)", где параметры N, M, K имеют соответствующие значения, где в украинском алфавите N – первая буква фамилии, М – первая буква имени и K – первая буква отчества. Но у некоторых вариантов возможна такая ситуация когда например перемножение количество блоков и количество потоков будет меньше чем глобальный индекс. Количество блоков задается в гриде с помощью функции:

dim3 dimGrid(4, 4, 1); //Размерность грида

, где параметры обозначают

uint3 blockIdx.x = 4

uint3 blockIdx.y = 4

uint3 blockIdx.z = 1

Следовательно количество блоков равно 4×4×1=16

Количество потоков задается в блоке с помощью функции:

dim3 dimBlock(1,1,1); //Размерность блока

, где параметры обозначают

uint3 threadIdx.x = 1

uint3 threadIdx.y = 1

uint3 threadIdx.z = 1

Следовательно количество потоков равно 1×1×1=1

Функция, для вывода ядром:

kernel<<<dimGrid, dimBlock>>>();

Индекс блока:

int BlockIndex = blockIdx.x + blockIdx.y\*gridDim.x + blockIdx.z\*gridDim.y\*gridDim.x;

Индекс потока:

Int ThreadIndex= threadIdx.x+ threadIdx.y\*blockDim.x + threadIdx.z\*blockDim.y\*blockDim.x;

Глобальный индекс:

int GlobalThreadIndex = BlockIndex\*blockDim.x\*blockDim.y\*blockDim.z + ThreadIndex;

Функция:

cudaDeviceSynchronize();

Блокирует, пока устройство не выполнит все предыдущее запрошенные задачи, возвращает ошибку, если одна из предыдущих задач не выполнена.

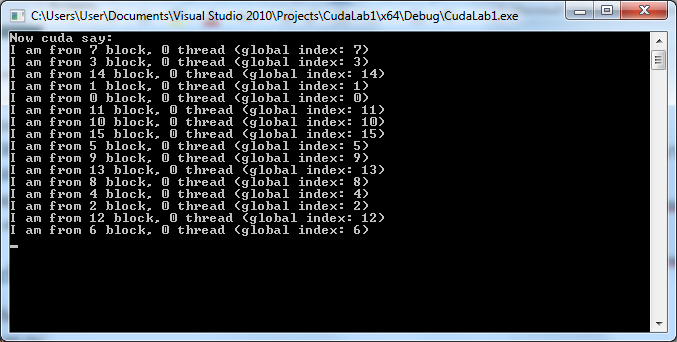


Рисунок 1 – Результат работы программы

Посмотреть характеристики карты можно вызвав функцию cudaGetDeviceProperties(), которая заполнит структуру типа cudaDeviceProp.

deviceProp.name - это строка ASCII, идентифицирующая устройство;

deviceProp.totalGlobalMem /(1024\*1024) - общий объем глобальной памяти, доступной на устройстве в Мегабайтах;

deviceProp.sharedMemPerBlock/1024 - это максимальный объем разделяемой памяти, доступный блоку потока в Килобайтах;

deviceProp.regsPerBlock - максимальное количество 32-разрядных регистров, доступных блоку потоков;

deviceProp.warpSize - размер основы в потоках;

deviceProp.memPitch - максимальный шаг в байтах, разрешенный функциями копирования памяти, которые связаны с областями памяти, выделенными cudaMallocPitch ();

deviceProp.maxThreadsPerBlock - максимальное количество потоков на блок;

deviceProp.maxThreadsDim[3] - содержит максимальный размер каждого измерения блока;

deviceProp.maxGridSize[3] - содержит максимальный размер каждого измерения сетки;

deviceProp.clockRate -  это тактовая частота в килогерце;

deviceProp.totalConstMem/1024 - общий объем постоянной памяти, доступной на устройстве в Килобайтах;

deviceProp.major, deviceProp.minor - это основные и младшие номера версий, определяющие вычислительную способность устройства;

deviceProp.textureAlignment - это требование выравнивания; текстурные базовые адреса, которые выровнены по тексту. Байт определения не требует смещения, применяемого к выборкам текстур;

deviceProp.deviceOverlap - 1, если устройство может одновременно копировать память между хостом и устройством во время выполнения ядра, или 0, если нет. Устаревший, используйте вместо deviceProp.asyncEngineCount.  
deviceProp.multiProcessorCount - это количество многопроцессоров на устройстве;

deviceProp.kernelExecTimeoutEnabled - 1, если для ядра, выполняемого на устройстве, существует ограничение на время выполнения, или 0, если нет.

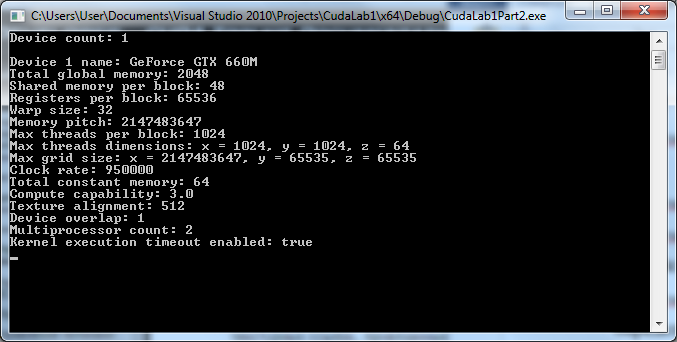


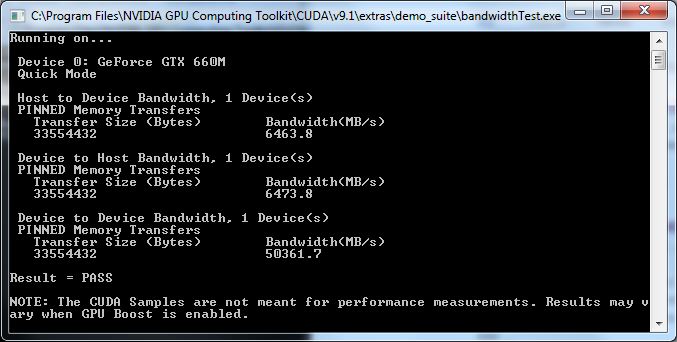
Рисунок 2 – Результат работы программы часть 2.  
  


Рисунок 3 – Результат работы программы bandwidthTest.exe.

Здесь важно содержимое пары строк: второй, которая отображает имя модели устройства, и предпоследней, которая сообщает, что все требуемые тесты пройдены.

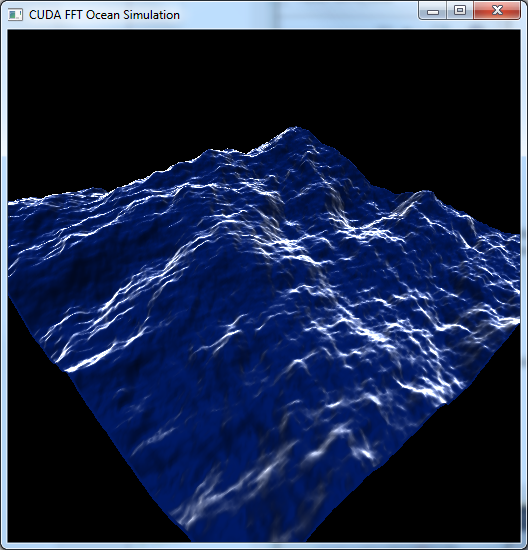


Рисунок 4 – Результат работы программы oceanFFT.exe.

Простые методы, демонстрирующие

* Основные подходы к вычислению GPU
* Рекомендации по наиболее важным функциям
* Эффективная работа с пользовательскими типами данных
* Быстрое интегрирование ускорения графического процессора в приложения C и C ++

Примеры использования, охватывающие такие темы, как:

* Добавление поддержки библиотек с ускорением GPU в приложение
* Использование таких функций, как память нулевого копирования, асинхронные передачи данных, унифицированная виртуальная адресация, одноранговая связь, параллельные ядра и многое другое
* Совместное использование данных между графическими API-интерфейсами CUDA и Direct3D / OpenGL (совместимость)

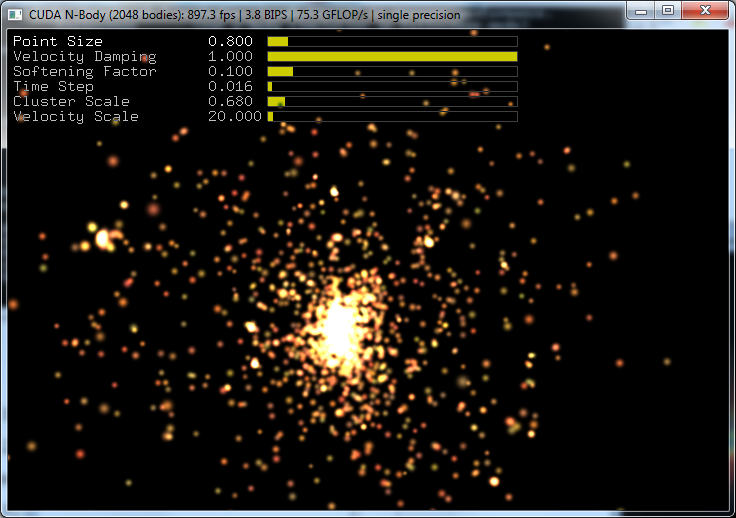


Рисунок 5 – Результат работы программы nbody.exe.

Данные-параллельные алгоритмы и примитивы для операций линейной алгебры:

* Матрица транспонирована
* Матричное умножение
* Матричное умножение с несколькими правыми частями
* Параллельная префиксная сумма больших массивов

Измерение и оптимизация производительности

* Тесты пропускной способности
* Профилирование приложений с использованием таймеров

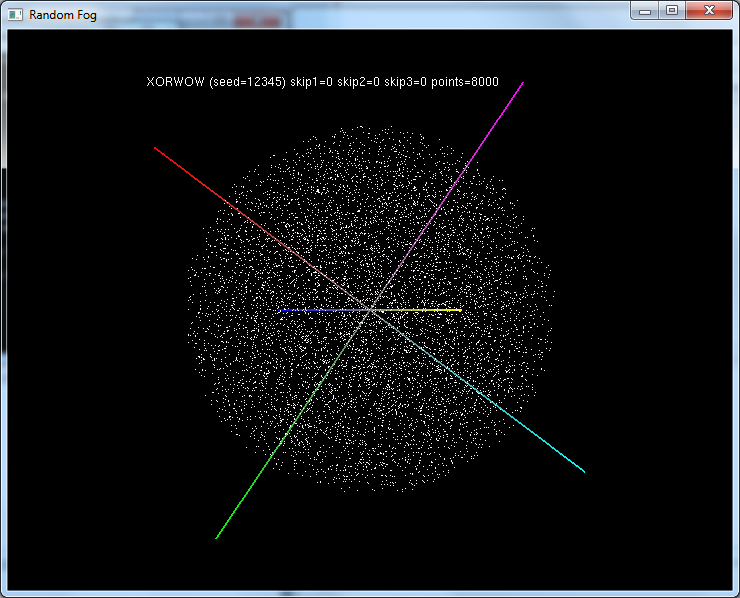


Рисунок 6 – Результат работы программы randomFog.exe.

Расширенные примеры приложений

* Использование CUDA с MPI и OpenMP
* Динамика вычислительной текучей среды (CFD)
* Гравитационное моделирование n-тела
* Блэк-Скоулз и биномиальная цена опциона
* 3D-конечная разность во времени (FDTD)
* Видеокодирование / декодирование
* Свертка изображений

Листинг программы

Код на С

Параметры ядра

#include "cuda\_runtime.h"

#include "device\_launch\_parameters.h"

#include <stdio.h>

#include <conio.h>

#include <stdlib.h>

#include <cuda.h>

#include <iostream>

#include <fstream>

#include <string>

using namespace std;

\_\_global\_\_ void kernel()

{

int BlockIndex = blockIdx.x + blockIdx.y\*gridDim.x + blockIdx.z\*gridDim.y\*gridDim.x;

int ThreadIndex = threadIdx.x + threadIdx.y\*blockDim.x + threadIdx.z\*blockDim.y\*blockDim.x;

int GlobalThreadIndex = BlockIndex\*blockDim.x\*blockDim.y\*blockDim.z + ThreadIndex;

printf("I am from %d block, %d thread (global index: %d)\n", BlockIndex, ThreadIndex, GlobalThreadIndex);

}

int main(int agrc, char\* argv[])

{

printf("Now cuda say:\n");

dim3 dimGrid(4,4,1); //16 - Л

dim3 dimBlock(1,1,1); //1 - А

kernel<<<dimGrid, dimBlock>>>();

cudaDeviceSynchronize();

\_getch();

return 0;

}

Характеристики видеокарты

#include "cuda\_runtime.h"

#include "device\_launch\_parameters.h"

#include <stdio.h>

#include <conio.h>

#include <iostream>

using namespace std;

int main()

{

int deviceCount;

cudaDeviceProp deviceProp;

cudaGetDeviceCount(&deviceCount);

printf("Device count: %d\n\n", deviceCount);

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

cudaGetDeviceProperties(&deviceProp, i);

printf("Device %d name: %s\n", i + 1, deviceProp.name);

printf("Total global memory: %lu\n", deviceProp.totalGlobalMem/(1024\*1024));

printf("Shared memory per block: %lu\n", deviceProp.sharedMemPerBlock/1024);

printf("Registers per block: %d\n", deviceProp.regsPerBlock);

printf("Warp size: %d\n", deviceProp.warpSize);

printf("Memory pitch: %lu\n", deviceProp.memPitch);

printf("Max threads per block: %d\n", deviceProp.maxThreadsPerBlock);

printf("Max threads dimensions: x = %d, y = %d, z = %d\n", deviceProp.maxThreadsDim[0], deviceProp.maxThreadsDim[1], deviceProp.maxThreadsDim[2]);

printf("Max grid size: x = %d, y = %d, z = %d\n", deviceProp.maxGridSize[0], deviceProp.maxGridSize[1], deviceProp.maxGridSize[2]);

printf("Clock rate: %d\n", deviceProp.clockRate);

printf("Total constant memory: %lu\n", deviceProp.totalConstMem/1024);

printf("Compute capability: %d.%d\n", deviceProp.major, deviceProp.minor);

printf("Texture alignment: %lu\n", deviceProp.textureAlignment);

printf("Device overlap: %d\n", deviceProp.deviceOverlap);

printf("Multiprocessor count: %d\n", deviceProp.multiProcessorCount);

printf("Kernel execution timeout enabled: %s\n", deviceProp.kernelExecTimeoutEnabled ? "true" : "false");

}

\_getch();

return 0;

}