Сложение векторов на CUDA

GPU Teaching Kit – Accelerated Computing

# Цели

Цель данной лабораторной – познакомить слушателей с CUDA API, реализовав сложение векторов. Студентам будет предложено реализовать сложение векторов, написав как код ядра для GPU, так и соответствующий код на хосте.

# Предварительные требования

Прежде чем начать, убедитесь, что:

* Вы закончили Модуль 2
* Вы выполнили лабораторную "Обращение к устройству"

# Ход выполнения

Отредактируйте код, чтобы выполнить следующие действия:

* выделение памяти на устройстве
* копирование памяти хоста на устройство
* инициализация размерностей сетки ядер и блоков нитей
* запуск ядра CUDA
* копирование результатов с устройства на девайс
* освобождение памяти устройства
* написание ядра CUDA

Места, куда следует вставить код, помечены комментариями //@@.

# Инструкция по установке

Последнюю версию исходного кода лабораторной, наряду со скриптами сборки, можно найти в репозитории [Bitbucket](https://bitbucket.org/hwuligans/gputeachingkit-labs/src/master/Module3). Инструкции к Cmake и сборки лабораторной можно найти в файле [README](https://bitbucket.org/hwuligans/gputeachingkit-labs/src/master/README.md) в корневой директории репозитория.

Исполняемый файл, являющийся результатом компиляции лабораторной, можно запустить следующей командой:

./VectorAdd\_Template -e <expected.raw> -i <intput1.raw>,<input2.raw> \  
 -o <output.raw> -t vector

где <expected.ppm> – имя ожидаемого результата, <input.ppm> – входной набор данных и <output.ppm> – необязательный путь для результатов. Набор данных можно сгенерировать, используя генератор, собранный во время компиляции.

# Вопросы

1. Сколько операций с плавающей точкой выполняется в вашем ядре сложения векторов? ОБЪЯСНИТЬ.
2. Сколько чтений из глобальной памяти производится вашим ядром? ОБЪЯСНИТЬ.
3. Сколько записей в глобальную память производится вашим ядром? ОБЪЯСНИТЬ.
4. Опишите возможный вариант оптимизации ядра для увеличения производительности.
5. Назовите три приложения сложения векторов.

# Шаблон кода

Представленный код предлагается как отправная точка. Импорты, экспорты и проверка решения уже представлены в коде. Требуется вставить свой код в области, обозначенные //@@. Остальной код трогать не нужно. Руководство описывает функционал методов с префиксом wb\*.

#include <wb.h>  
  
\_\_global\_\_ void vecAdd(float \*in1, float \*in2, float \*out, int len) {  
 //@@ Вставьте код, выполняющий сложение векторов  
}  
  
int main(int argc, char \*\*argv) {  
 wbArg\_t args;  
 int inputLength;  
 float \*hostInput1;  
 float \*hostInput2;  
 float \*hostOutput;  
 float \*deviceInput1;  
 float \*deviceInput2;  
 float \*deviceOutput;  
  
 args = wbArg\_read(argc, argv);  
  
 wbTime\_start(Generic, "Importing data and creating memory on host");  
 hostInput1 =  
 (float \*)wbImport(wbArg\_getInputFile(args, 0), &inputLength);  
 hostInput2 =  
 (float \*)wbImport(wbArg\_getInputFile(args, 1), &inputLength);  
 hostOutput = (float \*)malloc(inputLength \* sizeof(float));  
 wbTime\_stop(Generic, "Importing data and creating memory on host");  
  
 wbLog(TRACE, "The input length is ", inputLength);  
  
 wbTime\_start(GPU, "Allocating GPU memory.");  
 //@@ Выделите память GPU  
  
 wbTime\_stop(GPU, "Allocating GPU memory.");  
  
 wbTime\_start(GPU, "Copying input memory to the GPU.");  
 //@@ Скопируйте память на GPU  
  
 wbTime\_stop(GPU, "Copying input memory to the GPU.");  
  
 //@@ Инициализируйте размерности сетки и блоков  
  
 wbTime\_start(Compute, "Performing CUDA computation");  
 //@@ Запустите ядро GPU  
  
 cudaDeviceSynchronize();  
 wbTime\_stop(Compute, "Performing CUDA computation");  
  
 wbTime\_start(Copy, "Copying output memory to the CPU");  
 //@@ Скопируйте память GPU обратно на хост  
  
 wbTime\_stop(Copy, "Copying output memory to the CPU");  
  
 wbTime\_start(GPU, "Freeing GPU Memory");  
 //@@ Освободите память GPU  
  
 wbTime\_stop(GPU, "Freeing GPU Memory");  
  
 wbSolution(args, hostOutput, inputLength);  
  
 free(hostInput1);  
 free(hostInput2);  
 free(hostOutput);  
  
 return 0;  
}