CUDA Thrust Vector Add

GPU Teaching Kit – Accelerated Computing

# Цели

Цель данной лабораторной – познакомить слушателей с CUDA API, реализовав сложение векторов, используя Thrust.

# Предварительные требования

Прежде чем начать, убедитесь, что:

* вы закончили Модуль 2
* вы выполнили лабораторную "Обращение к устрйству"

# Ход выполнения

Отредактируйте код, чтобы выполнить следующие действия:

* сгенерируйте thrust::dev\_ptr<float> для входных массивов хоста
* скопируйте память хоста на устройство
* вызовите thrust::transform()
* скопируйте результат на хост

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:

./ThrustVectorAdd\_Template -e <expected.raw> \  
 -i <input0.raw>,<input1.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. Каким образом функциональный стиль кода сложения векторов в Thrust делает разработку проще или сложнее?

# 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 <thrust/device\_vector.h>  
#include <thrust/host\_vector.h>  
#include <wb.h>  
  
int main(int argc, char \*argv[]) {  
 wbArg\_t args;  
 float \*hostInput1 = nullptr;  
 float \*hostInput2 = nullptr;  
 float \*hostOutput = nullptr;  
 int inputLength;  
  
 args = wbArg\_read(argc, argv); /\* чтение входных аргументов \*/  
  
 // Импорт входных данных на хост  
 wbTime\_start(Generic, "Importing data to host");  
 hostInput1 =  
 (float \*)wbImport(wbArg\_getInputFile(args, 0), &inputLength);  
 hostInput2 =  
 (float \*)wbImport(wbArg\_getInputFile(args, 1), &inputLength);  
 wbTime\_stop(Generic, "Importing data to host");  
  
 // Объявление и выделение памяти под выходные данные  
 //@@ Место для вставки кода  
 wbTime\_start(GPU, "Doing GPU Computation (memory + compute)");  
  
 // Объявление и выделение памяти под входные и выходные данные на устройства через thrust  
 wbTime\_start(GPU, "Doing GPU memory allocation");  
 //@@ Место для вставки кода  
 wbTime\_stop(GPU, "Doing GPU memory allocation");  
  
 // Копирование на устройство  
 wbTime\_start(Copy, "Copying data to the GPU");  
 //@@ Место для вставки кода  
 wbTime\_stop(Copy, "Copying data to the GPU");  
  
 // Выполнение операции сложения векторов  
 wbTime\_start(Compute, "Doing the computation on the GPU");  
 //@@ Место для вставки кода  
 wbTime\_stop(Compute, "Doing the computation on the GPU");  
 /////////////////////////////////////////////////////////  
  
 // Копирование данных обратно на хост  
 wbTime\_start(Copy, "Copying data from the GPU");  
 //@@ Место для вставки кода  
 wbTime\_stop(Copy, "Copying data from the GPU");  
  
 wbTime\_stop(GPU, "Doing GPU Computation (memory + compute)");  
  
 wbSolution(args, hostOutput, inputLength);  
  
 free(hostInput1);  
 free(hostInput2);  
 free(hostOutput);  
 return 0;  
}