Обращение к устройству

GPU Teaching Kit – Вычисления на графических ускорителях

# Цель работы

Цель данной лабораторной – дать студентам представление об аппаратных ресурсах CUDA наряду с их возможностями. Не требуется понимание кода, но важно понять процесс его компилирования и исполнения, который будет использоваться в следующих модулях.

# Инструкции

Код выполняет запросы к устройству GPU. Заостряйте внимание не на вызовах API, а на функциях, начинающихся с wb.

Функция wbLog производит печать на экран. Конкретно мы будем выводить следующие аппаратные характеристики:

* название карты GPU
* GPU computation capability
* максимальная размерность блока
* максимальная размерность сетки
* размер видеопамяти
* размер постоянной и разделяемой памяти
* размер варпа

Детали объясняются в первом и втором модулях курса.

# Инструкции по установке

Последнюю версию исходного кода лабораторной, наряду со скриптами сборки, можно найти в репозитории [Bitbucket](https://bitbucket.org/account/signin/?next=/hwuligans/gputeachingkit-labs/src/master/Module2). Инструкции к [Cmake](https://cmake.org/) 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.

./DeviceQuery\_Template

# Вопросы

1. Какая версия compute capability у архитектуры NVIDIA Fermi?
2. Какая максимальная размерность блока у GPU с compute capability 3.0?
3. Предположим, вы используете одномерную сетку и блок. Если максимальная размерность сетки на устройстве 65535, а максимальная размерность блока 512, какое максимальное количество нитей может быть запущено на GPU?
4. При каких условиях программист предпочтет не запускать максимальное количество нитей?
5. Что может помешать программе запустить максимальное количество нитей?
6. Что такое разделяемая память?
7. Что такое глобальная память?
8. Что такое константная память?
9. Что характеризует размер варпа в GPU?
10. Поддерживаются ли числа двойной точности в GPU версии 1.3?

# 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>  
  
//@@ Задача данного кода – познакомиться с процессом передачи  
//@@ данных. Не переживайте, если не понимаете деталей кода.  
  
int main(int argc, char \*\*argv) {  
 int deviceCount;  
  
 wbArg\_read(argc, argv);  
  
 cudaGetDeviceCount(&deviceCount);  
  
 wbTime\_start(GPU, "Getting GPU Data."); //@@ Запуск таймера  
  
 for (int dev = 0; dev < deviceCount; dev++) {  
 cudaDeviceProp deviceProp;  
  
 cudaGetDeviceProperties(&deviceProp, dev);  
  
 if (dev == 0) {  
 if (deviceProp.major == 9999 && deviceProp.minor == 9999) {  
 wbLog(TRACE, "No CUDA GPU has been detected");  
 return -1;  
 } else if (deviceCount == 1) {  
 //@@ WbLog предоставляет API логгирования (схожее с Log4J).  
 //@@ Функции логгирования wbLog принимает уровень  
 //@@ логгирования, который может быть одним из  
 //@@ OFF, FATAL, ERROR, WARN, INFO, DEBUG или TRACE, и  
 //@@ сообщение, которое нужно напечатать.  
 wbLog(TRACE, "There is 1 device supporting CUDA");  
 } else {  
 wbLog(TRACE, "There are ", deviceCount,  
 " devices supporting CUDA");  
 }  
 }  
  
 wbLog(TRACE, "Device ", dev, " name: ", deviceProp.name);  
 wbLog(TRACE, " Computational Capabilities: ", deviceProp.major, ".",  
 deviceProp.minor);  
 wbLog(TRACE, " Maximum global memory size: ",  
 deviceProp.totalGlobalMem);  
 wbLog(TRACE, " Maximum constant memory size: ",  
 deviceProp.totalConstMem);  
 wbLog(TRACE, " Maximum shared memory size per block: ",  
 deviceProp.sharedMemPerBlock);  
 wbLog(TRACE, " Maximum block dimensions: ",  
 deviceProp.maxThreadsDim[0], " x ", deviceProp.maxThreadsDim[1],  
 " x ", deviceProp.maxThreadsDim[2]);  
 wbLog(TRACE, " Maximum grid dimensions: ", deviceProp.maxGridSize[0],  
 " x ", deviceProp.maxGridSize[1], " x ",  
 deviceProp.maxGridSize[2]);  
 wbLog(TRACE, " Warp size: ", deviceProp.warpSize);  
 }  
  
 wbTime\_stop(GPU, "Getting GPU Data."); //@@ остановка таймера  
  
 return 0;  
}