Module 2 Lab

Device Query

GPU Teaching Kit – Accelerated Computing

OBJECTIVE

The purpose of this lab is to introduce the student to the CUDA hardware resources along with their capabilities. The student is not expected to understand the details of the code, but should understand the process of compiling and running code that will be used in subsequent modules.

INSTRUCTIONS

The code provided queries the GPU hardware on the system. Do not concentrate on the API calls, but on functions starting with `wb`.

The `wbLog` function logs results, specifically we log the following hardware features:

- GPU card's name
- GPU computation capabilities
- Maximum number of block dimensions
- Maximum number of grid dimensions
- Maximum size of GPU memory
- Amount of constant and share memory
- Warp size

The details are explained in the first and second modules of the teaching kit.

LOCAL SETUP INSTRUCTIONS

The most recent version of source code for this lab along with the build-scripts can be found on the Bitbucket repository. A description on how to use the CMake tool in along with how to build the labs for local development found in the README document in the root of the repository.

The executable generated as a result of compiling the lab can be run using the following command:

```
./DeviceQuery_Template
```

QUESTIONS

(1) What is the compute capability of the NVIDIA Fermi architecture?

ANSWER: **2.1.**

(2) What are the maximum block dimensions for GPUs with 3.0 compute capability?

ANSWER: **1024 in X, 64 in Y and Z.**

(3) Suppose you are launching a one dimensional grid and block. If the hardware's maximum grid dimension is 65535 and the maximum block dimension is 512, what is the maximum number threads can be launched on the GPU?

ANSWER: **65535 * 512 = 33553920**

(4) Under what conditions might a programmer choose not want to launch the maximum number of threads?

ANSWER: **There may not be enough data to effectively use the maximum number of threads.**

(5) What can limit a program from launching the maximum number of threads on a GPU?

ANSWER: **A program may be limited by shared memory or registers used across all threads.**

(6) What is shared memory?

ANSWER: **Fast read-write memory shared between all threads in a block.**

(7) What is global memory?

ANSWER: **Read-write memory that can be accessed by any thread.**

(8) What is constant memory?

ANSWER: **Fast read-only memory for values that do not change during kernel execution.**

(9) What does warp size signify on a GPU?

ANSWER: **The number of threads that execute in lockstep.**

(10) Is double precision supported on GPUs with 1.3 compute capability?

ANSWER: **Double precision is supported starting with CC 1.3.**

CODE TEMPLATE

The following code is suggested as a starting point for students. The code handles the import and export as well as the checking of the solution. Students are expected to insert their code is the sections demarcated with `//@@`. Students expected the other code unchanged. The tutorial page describes the functionality of the `wb*` methods.

```
#include <wb.h>

//@@ The purpose of this code is to become familiar with the submission
//@@ process. Do not worry if you do not understand all the details of
//@@ the code.

int main(int argc, char **argv) {
  int deviceCount;

  wbArg_read(argc, argv);

  cudaGetDeviceCount(&deviceCount);

  wbTime_start(GPU, "Getting GPU Data."); //@@ start a timer

  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 is a provided logging API (similar to Log4J).
        //@@ The logging function wbLog takes a level which is either
        //@@ OFF, FATAL, ERROR, WARN, INFO, DEBUG, or TRACE and a
        //@@ message to be printed.
        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."); //@@ stop the timer

  return 0;
}
```

CODE SOLUTION

The following is a possible implementation of the lab. This solution is intended for use only by the teaching staff and should not be distributed to students.

```
#include <wb.h>

//@@ The purpose of this code is to become familiar with the submission
//@@ process. Do not worry if you do not understand all the details of
//@@ the code.

int main(int argc, char **argv) {
  int deviceCount;

  wbArg_read(argc, argv);

  cudaGetDeviceCount(&deviceCount);

  wbTime_start(GPU, "Getting GPU Data."); //@@ start a timer

  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 is a provided logging API (similar to Log4J).
        //@@ The logging function wbLog takes a level which is either
        //@@ OFF, FATAL, ERROR, WARN, INFO, DEBUG, or TRACE and a
        //@@ message to be printed.
        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."); //@@ stop the timer

  return 0;
}
```