**CSC 447: Parallel Programming for Multicore and Cluster Systems**

*Instructor: Haidar M. Harmanani*

Spring 2018

Lab 6

MPI and CUDA C Toolkit

Due: February 28, 2018

**Part I: MPI**

In Lab: Compile and execute the average program found in the Lab [repo](https://github.com/CSC447/Lab06). The main objevtive is to understand how is MPI written, compiled, and executed.

*PostLab (Due March 8) [100 Points]:* Using MPI, write a distributed-memory parallel Jacobi iteration. Make sure that there is only one process per processor (coarse granularity). Perform speedup measurements on 2, 4, 8, and 16 nodes in the High-Performance Lab. Your experiments should be carefully done; for example, you need to coordinate with your colleagues to make sure that they are not running tests. Make sure that you compute speedup relative to the sequential program, not the one-processor parallel program.

Bonus [25 Points]: Integrate the message-passing program with the shared-memory one you wrote last time. This means you will be running a thread-based shared-memory program on each node as well as a distributed-memory one between the nodes (simultaneously). Repeat the above speed-up computations. What do you notice?

**Part II [In Lab]: CUDA Toolkit and GPU Hardware**

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 [github](https://github.com/CSC447/Lab06). 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 Card in the Lab?
2. What are the maximum block dimensions for the GPUs in the Lab?
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?
4. Under what conditions might a programmer choose not want to launch the maximum number of threads?
5. What can limit a program from launching the maximum number of threads on a GPU?
6. What does warp size signify on a GPU?
7. Is double precision supported on GPUs with 1.3 compute capability?

# 

# Code Template

insert your code is the sections demarcated with

#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;  
}