Programming Assignment 1

Due April 18, 2012 12:50 PM

In this programming assignment you will use NVIDIA's Computing Unified Device Architecture (CUDA) language to implement a basic string cipher algorithm and the pagerank algorithm. In the process you will learn how to write general purpose GPU programming applications and consider some optimization techniques. You must turn in your own copy of the assignment as described below. You may discuss the assignment with your peers, but you may not share answers. You may turn in the assignment up to 24 hours late (Late Deadline: April 19, 2012 12:50 PM) with a 10% penalty. Please direct your questions about the assignment to the piazza forum.

1 CUDA

"C for CUDA" is a programming language subset and extension of the C programming language, and is commonly referenced as simply CUDA. Many languages support wrappers for CUDA, but in this class we will develop in C for CUDA and compile with `nvcc`.

The programmer creates a general purpose kernel to be run on a GPU, analogous to a function or method on a CPU. The compiler allows you to run C++ code on the CPU and the CUDA code on the device (GPU). Functions which run on the host are prefaced with `__host__` in the function declaration. Kernels run on the device are prefaced with `__global__`. Kernels that are run on the device are only to be called from the device are prefaced with `__device__`. cipher.cu has an example of a kernel declared with the `__global__` keyword.

The first step you should take in any CUDA program is to move the data from the host memory to device memory. The function calls `cudaMalloc` and `cudaMemcpy` allocate and copy data, respectively. `cudaMalloc` will allocate a specified number of bytes in the device main memory and returns a pointer to the memory block, similar to malloc in C. You should not try to dereference a pointer allocated with cudaMalloc from a host function, as it resides in a different address space you will not be able to access it (and the program will segfault).

The second step is to use `cudaMemcpy` from the CUDA API to transfer a block of memory from the host to the device. You can also use this function to copy memory from the device to the host. It takes four parameters, a pointer to the device memory, a pointer to the host memory, a size, and the direction to move data (`cudaMemcpyHostToDevice` or `cudaMemcpyDeviceToHost`). We have already provided the code to copy the string from the host memory to the device memory space, and to copy it back after calling your cipher kernel.

Kernels are launched in CUDA using the syntax `kernelName<<<...>>>(...)`. The arguments inside of the chevrons (`<<<...>>>`) specify the number of threads and thread blocks to be launched for the kernel. The arguments to the kernel are passed by value like in normal C/C++ functions. You need to call your cipher kernel from the main method.

There are some read-only variables that all threads running on the device possess. The three most valuable to you for this assignment are `blockIdx`, `blockDim`, and `threadIdx`. Each of these variables contains fields x, y, and z. blockIdx contains the x, y, and z coordinates of the thread block where this thread is located. blockDim contains the dimensions of thread block where the thread resides. threadIdx contains the indices of this thread within the thread block.

The CUDA language is complicated and intricate. We encourage you to consult the development materials available from NVIDIA, particularly the CUDA Programming Guide and the Best Practices Guide available at `http://www.nvidia.com/object/cuda_development.html`

2 What we give you

- `cipher.cu` - This is the starter code for the first part of the programming assignment. You will be implementing a simple Caesar Cipher. The serial host code is already implemented for you. You should modify this file.

- `pagerank.cu` - This is the starter code for the second part of the programming assignment. You will be implementing the PageRank algorithm developed by Sergey Brin and Larry Page. The serial host code is already implemented for you. You should modify this file.
- `Makefile` - make will build the cipher and pagerank binaries. make clean will remove the executables. You should be able to build and run the program when you first download it, however only the host code will run. You do not need to modify this file.
- `mobydick.txt` - This the text of Moby Dick by Herman Melville. You will use it to test your cipher algorithm for both correctness and performance.

These files can be found in `/afs/ir/class/cme213/assignments/pa1` on Leland machines (such as the corn and pup clusters).

3 Code implementation

Your first task is as follows:

1. Implement the cipher algorithm and pagerank algorithm using the CUDA API. You do not need to define any functions or kernels, but must fill in the starter code we have given you at the appropriate locations.

For this question, you have to turn in your code. We will run your code to check that the results are correct.

String Cipher Algorithm

The first documented use of encryption was done by Julius Caesar. It is a simple algorithm that substitutes every letter with the letter k spaces later in the alphabet, wrapping around. So if k = 3 (as it was in the original Caesar cipher) we would convert "zebra" to "cheud", as z → a → b → c, e → f → g → h, and so forth. While it took a very significant amount of time to be translated by the Romans, you will see that a GPGPU can do it very quickly.

PageRank

PageRank was the link analysis algorithm responsible for the success of Google. It generates a score for every node in a graph by considering the number of in links and out links of a node. We are going to compute a simplified model of pagerank, which, in every iteration computes the pagerank score as a vector π and updates π as

$$\pi(t+1) = \frac{1}{2}\,\pi(t)\;A + \frac{1}{2}\,\mathbf{1}$$

where A is a weighted adjacency matrix and $\mathbf{1}$ is a vector whose every entry is the value 1. We perform this update step for twenty iterations and the final pagerank vector is our result.

Hints

- The cipher asks you to make some optimization to the code by considering how you can more intelligently perform the computation. Think about how data is represented at the bit level, and how arithmetic operations are computed.
- In order to perform a batch update, we use two pagerank vectors in our algorithm and switch their roles on every iteration (reading from one and writing to the other)
- It may be necessary to create a grid that uses more than one dimension. Recall that the largest dimension of the grid in one dimension is 65,535.

Scoring

The grading for problem 1 is based on the correctness of the code. The points will be given as follows.

Caesar Cipher (20%). Your program should write the data to `device_output_array`. Our code will then check your result against the host implementation and confirm the answer is correct. If the program works correctly, it should output "All CUDA Versions matched reference output. Outputting ciphered text" to stdout followed by the enciphered Moby Dick text.

PageRank (30%). Your program should write the data to `h_graph_nodes_result`, which is a pointer to memory on the host. We will then check for errors. If your program outputs "Worked! CUDA and reference output match." Then you have the same pagerank vector as the host code.

4 Bandwidth

For the second part of the homework, please include a PDF file with the answers to the following questions:

2. Caesar Cipher (20%)
   - (a) Generate plots with char, uint, and uint2 on the same plot. Create a plot for
     - i. Bandwidth vs. vector length
     - ii. Bandwidth vs. block size
   - (b) Explain the shape of your plots. You only need to plot enough data points to show the general trend.
   - (c) What is the bandwidth that is achieved on the copies from the host to the device, and from the device to the host? How do they compare?
3. PageRank (30%)
   - (a) Calculate the bandwidth when the average number of edges is 8.
     - i. How does this compare to the bandwidth in the Caesar Cipher? Explain any differences you observe.
     - ii. Does changing the block size make a difference? Why or why not?
   - (b) Plot bandwidth vs. avg_edges ranging from 2 to 20 edges.
     - i. Explain the shape of the curve.
     - ii. Does the size of the vector have a significant impact on the bandwidth?

5 Submission instructions

You should submit the complete source code for your working solution and a brief pdf file (no more than one or two pages excluding plots) explaining the correctness and bandwidth utilization (as described above). **Do not resubmit the mobydick.txt file to us.**

To submit:

- Create a directory, for example `pset1`, on your Leland account.
- Copy the files you would like to submit in the directory `pset1`. Do not include auxiliary files that are not needed.
- From within the director `pset1`, run the shell command
  ```
  corn:> /afs/ir/class/cme213/bin/submit pa1
  ```
  This will send the entire content of the directory to us and will label it `pa1`.

A timestamp is also included. We will use your last submission before the midnight deadline for grading. Please send a message to the course staff through piazza if you encounter a problem.

6 Hardware available for this class

Machines

We will be using the icme-gpu teaching cluster, which you can access with ssh:

```
ssh sunet@icme-gpu1.stanford.edu
```

You should have received an email with your username (your SUNet id) and a temporary password. Upon logging in for the first time, please change your password by entering the command passwd.

We have only provided accounts for those enrolled in the course. If you are auditing the course, we may consider a special request for an account.

To learn more about these machines and how to use them, visit
`http://icme.stanford.edu/Computer Resources/gpu.php`
The tutorial is very helpful if you are unsure how to proceed.

Compiling

To use nvcc on the icme cluster, you must copy and paste these lines to your .bashrc file:

```
module add open64
module add openmpi
module add cuda40
module add torque
```

You can now use nvcc to compile CUDA code. We have provided a makefile which will compile cipher and pagerank by typing make.

Running

The cluster uses MOAB job control. The easiest way to run an executable is by using interactive job submission. First enter the command

```
msub -I nodes=# :ppn=# :gpus=#
```

where # correspond to the number of nodes, processors, and gpus respectively. For this assignment, you can use 1 for every value.

You can then run any executable in the canonical fashion. For example

```
./pagerank
```

as you would on your personal computer.