Programming Assignment 2 — Due April 26, 2013 12:50 PM

In this programming assignment you will use NVIDIA's Compute Unified Device Architecture (CUDA) language to implement a basic string shift 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. Please direct your questions about the assignment to the piazza forum.

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 and that are only called from the device are prefaced with `__device__`.

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 return 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.

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 shift kernel.

Kernels are launched in CUDA using the syntax `kernelName<<<...>>>(...)`. The arguments inside of the chevrons (`<<<blocks, threads>>>`) specify the number of thread blocks and thread per block to be launched for the kernel. The arguments to the kernel are passed by value like in normal C/C++ functions.

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.

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`

Problem 1 String Shift

The purpose of this problem is to give you experience writing your first simple CUDA program. This program will help us examine how various factors can affect the achieved memory bandwidth. The program will take an input string and shift each character by constant amount. You will do this in three different ways:

- by shifting one byte at a time
- by shifting 4 bytes at a time
- and finally by shifting 8 bytes at a time

You should be able to take the files we give you and type `make` to build the `shift` executable. The executable takes 1 argument - the number of times to double the input file in size. For debugging it is

recommended to use a value of 0. The executable will run, but since the CUDA code hasn't been written yet (that's your job), it will report errors and quit.
For this problem we provide the following starter code:

- `shift_driver.cu` - This is the main file. We have already written most of the code for this assignment so you can concentrate on the CUDA code. We take care of loading the input file, computing the host solution and checking your results against the host reference. There is also a loop setup that will generate a table of results for the three kernels for a variety of sizes. You are free to modify this file, but it shouldn't be necessary.
- `studentSolutionShift.cu` - This is the file you will need to modify and submit. It already contains the necessary function headers - DO NOT change these. You should fill in the body of each function.
- `Makefile` - make will build the shift binary. 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 shift algorithm for both correctness and performance.

Question 1.1

25 points. Fill in the functions in `studentSolutionShift.cu` so that the program no longer reports any errors.

Question 1.2

5 points. Take the table that should be generated once you've correctly implemented everything and generate a plot of bandwidth in GB/sec vs. problem size in MB. For these tables, pass the argument 8 to the executable so that it doubles the input 8 times for the maximum size.

Question 1.3

10 points. Performance should increase significantly from the `char` to `uint` versions of the kernel. Why? Why does the performance not change much between the `uint` and `uint2` versions of the kernel?

Problem 2 PageRank

PageRank was the link analysis algorithm responsible (in part) 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

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

where A is a weighted adjacency matrix and $\mathbf{1}$ is a vector whose every entry is the value 1. Each entry in the vector π corresponds to the score for one node. The matrix A is sparse and each row i corresponds to the node n_i, the non-zero entries in this row correspond to the other nodes that are connected to n_i. We will choose the *average* number of connections for a node to be N and then have the actual number of connections per node vary from 1 to $2N - 1$.

In the actual algorithm this operation is performed until the change between successive π vectors is sufficiently small. In our case we will choose a fixed number of iterations to more easily compare performance across various numbers of nodes and edges. If you wish to learn more about the algorithm itself, check `http://en.wikipedia.org/wiki/PageRank`

For this problem, we provide the following starter code:

- `pagerank_driver.cu` - contains the code that sets up the problem and generates the reference solution. It also has a result generating loop that will generate a table of timing results for various numbers of edges and nodes. You may modify this file, but other than filling in the bandwidth calculation and a tiny required change to answer one of the questions, it shouldn't be necessary.

- `studentSolutionPagerank.cu` - this is the file you will need to modify and submit. Do not change the function headers but fill in the bodies and follow the hints/requirements in the comments.
- `Makefile` - make will build the pagerank binary. 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.

Question 2.1

35 points. Fill in the functions so that the program no longer reports any errors.

Question 2.2

10 points. What is the formula for the total number of bytes read from memory? Use the variables `num_nodes`, `num_edges` and `iterations`.

Edit the bandwidth calculation in the driver file (search for TODO) to reflect your answer to question 2.2.

Question 2.3

5 points. From the table of results, plot the memory bandwidth (GB/sec) vs. problem size for an average number of edges equal to 10. Make sure the plot is readable. You do not have to comment the plot.

Question 2.4

10 points. What does the memory access pattern look like for this problem? Using your answer to this question, explain the difference in bandwidth between Problem 1 and Problem 2.

Total number of points: 100

A Submission instructions

You should submit one zip file containing a folder named `FirstName_LastName_SUNetID_PA2`. Make sure to have a folder with this name otherwise your files will get mixed up with other students when we unzip your file. This folder should contain:

- `studentSolutionCipher.cu`
- `studentSolutionPagerank.cu`
- a pdf file named `Readme.pdf` containing your answers to the questions (and the plots)

The zip file should be named `PA2.zip`.

To run (and grade) your code, we will copy your source file in our directory and type:

- `make`
- `./shift` or `./pagerank`

B 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 (so they will be loaded when you connect on the cluster).

```
module add open64
module add cuda50
```

For the first time, after you changed the `.bashrc`, you need to logout and re-login (so that the changes are taken into account) or source the `.bashrc`.

You can now use `nvcc` to compile CUDA code.

Running

The cluster uses MOAB job control. The easiest way to run an executable is by using interactive job submission. First enter the command[1]

```
msub -I -l nodes=1:gpus=1
```

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

```
./pagerank
```

as you would on your personal computer.

C Advice and Hints

- You will need to use the bit shift `<<` and bitwise OR operators `|` to generate a suitable `uint` shift for shifting 4 bytes with one addition.
- 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 will be necessary to create a grid that uses more than one dimension to handle the larger sizes. Recall that the largest dimension of the grid in one dimension is 65,535.
- For debugging it will be helfpul to limit the number of cases being run to 1. In the shift problem do this by passing 0 as the parameter. In the pagerank problem change the values of `num_nodes` and `num_edges`.
- If you need some documentation on Cuda, you can look at the documents uploaded on Coursework (*Cuda Best Practice* and *Cuda Programming Guide*).
- In case nothing runs (for instance the device does not manage to allocate any memory), type in the command line `nvidia-smi`. If all the GPUs are in use, the error you get is possibly not because of your code. Try to logout and re-login. If the problem persists (you are assigned to a node that never have a free GPU), send me an email.
- For Problem 2, make sure you understand how the sparse matrix is encoded in memory. This will greatly help you figure out the code to write.

[1] Adding an alias to your `.bashrc` file may be a good idea if you want to avoid typing the following line too often.