A GPGPU Exercise on Blue Waters

Aaron Weeden, Shodor
2015
http://tinyurl.com/acca-cs-gpgpu

If you have not already done An MPI Exercise on Blue Waters, it is recommended you do so first.

Also, if you have not already read the slides for GPGPU: Terminology and Examples, it is recommend you do so first.

This exercise uses two examples of the same fire model, one written using OpenACC, the other written using CUDA. To see how the OpenACC version of the code compares to the serial version, download this file and open it in a web browser. The serial version is shown on the left, and the OpenACC version is shown on the right, with the differences highlighted. A similar file for the CUDA version is available here. There is also a file that compares the OpenACC version (left) with the CUDA version (right) here.

Both versions of the code use many files. You may find it helpful to reference this OpenACC flowchart and this CUDA flowchart to see how all the files interact.

For this exercise, you will need two terminal windows connected to Blue Waters. To connect to Blue Waters, follow the instructions in the *Logging In* section of A Blue Waters Usage Guide. Then, open a second window and repeat the process. On Windows, you can open a second PuTTY window by simply launching the program a second time. On Mac, you can open a second window in Terminal by typing **Command-N**.

Request interactive job

In the first window, request an interactive job for 1 hour using 1 node with 16 CPU cores and 1 GPGPU. This can be accomplished using the following command:

```
qsub -I -l nodes=1:ppn=16:xk -l walltime=01:00:00<ENTER>
```

This is slightly different from what we did in the OpenMP and MPI exercises. Here, we are asking for an XK node instead of an XE node. An XK node has 16 CPU cores and a GPGPU, as opposed to an XE node, which has 32 CPU cores but no GPGPU.

Update example code

While you wait for the interactive job to become available in the first window, in the second window, update the example fire model code in your home directory (you should already have a

directory called **fire** from doing the OpenMP and MPI exercises). **NOTE**: doing this update will replace the contents of the **acc** and **cuda** directories. If you have worked with these directory on your own since the last exercise, please make sure to make a backup of your files. After you have made any needed backups and are ready to replace the contents, the code can be replaced using the following commands:

With tabs:	**cp ~awee<TAB>f<TAB>/a<TAB>* ~/f<TAB>/a<TAB><ENTER>** **cp ~awee<TAB>f<TAB>/c<TAB>* ~/f<TAB>/c<TAB><ENTER>**
Without tabs:	**cp ~aweeden/fire/acc/* ~/fire/acc<ENTER>** **cp ~aweeden/fire/cuda/* ~/fire/cuda<ENTER>**

Change into the **fire** directory:

```
cd ~/fire<ENTER>
```

Confirm you are now in the **fire** directory:

```
pwd<ENTER>
```

You should get back **/u/training/<your username>/fire**

Create weak scaling plots

Inside the **acc** directory is a sample file, **scale-acc.out**, which is the output of running the **scale-acc.sh** batch script. Display the contents of that output file:

```
cat acc/scale-acc.out<ENTER>
```

The contents of the file are pairs of data: the number of GPGPU threads used, and the average wall time (in seconds) to run the program with that many threads. Highlight the contents of the file using your mouse/trackpad. On Windows, this causes PuTTY to copy what you have selected. On Mac, press **Command-C** to copy.

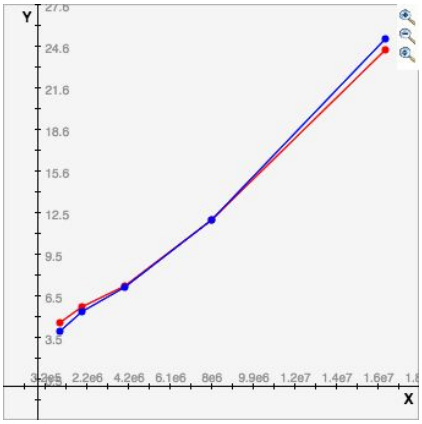
Go to this website in a web browser: http://shodor.org/interactivate/activities/SimplePlot/. In the text box below the word **Data**, type the word **redgraph** followed by a new line, and paste. Then on a new line, type the word **bluegraph** followed by a new line.

Back in the second terminal window, display the contents of a similar output file for the CUDA version of the program (as opposed to the OpenACC version):

```
cat cuda/scale-cuda.out<ENTER>
```

Copy and paste this data into the **Data** box in Simple Plot.

Click the button that says **`Plot/Update`**. You should get a graph that looks like the following, with the the x-axis measuring GPGPU thread counts and the y-axis measuring average wall time in seconds:



Notice that both the OpenACC and CUDA versions take about the same amount of time for each thread count. This hints that both versions use the GPGPU similarly.

Run interactively

These steps should be followed in the first window, the window in which you requested an interactive job.

Change directories to the example code directory:

```
cd ~/fire<ENTER>
```

Run the OpenACC program using the command below:

```
time aprun -n 1 acc/fire-acc -r 1300 -c 1300 -t 1300<ENTER>
```

This will run for a few seconds and print the final total percentage of trees that burned (100%) in a forest with 1300 rows, columns, and time steps. It will also print the **`real`**, **`user`**, and **`sys`** time. The **`real`** time is the wall clock time and is more reliable for measuring performance than the **`user`** and **`sys`** times.

Run the CUDA program using the command below:

```
time aprun -n 1 cuda/fire-cuda -r 1300 -c 1300 -t 1300<ENTER>
```

You should expect to get back a similar **`real`** time.

Open the source code for the CUDA version of the program in vi:

```
vi cuda/fire-cuda.cu<ENTER>
```

Type the following to jump ahead to line 35 of the program:

```
:35<ENTER>
```

You should see the following line:

```
#define THREADS_PER_BLOCK 1024
```

Enter insert mode by typing **i**, and change the number **1024** to **512**. This number represents the number of threads per warp (32) times the number of warps per block (max 32). We have now halved the number of warps per block, which means we will have doubled the total number of blocks per grid (because the number of trees is the same). The line should now look like this:

```
#define THREADS_PER_BLOCK 512
```

Type **Esc** to exit insert mode, then type the following to save and quit:

```
:wq<ENTER>
```

Compile the program by running the following commands:

```
cd cuda<ENTER>
module load cudatoolkit<ENTER>
make<ENTER>
```

Modify the **scale-cuda.sh** script, which was used to generate the **scale-cuda.out** file:

```
vi scale-cuda.sh<ENTER>
```

Press **i** to enter insert mode, change the number of trials on line 5 from **5** to **1**, press **Esc** to exit insert mode, then enter the following to save and quit:

```
:wq<ENTER>
```

Run the script:

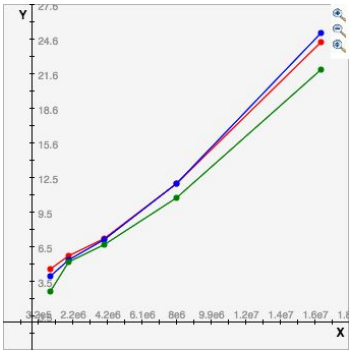
```
./scale-cuda.sh<ENTER>
```

Enter **y** to delete the **scale-cuda.out** file.

Display the contents of the file:

```
cat scale-cuda.out<ENTER>
```

In Simple Plot, create a graph with a new color (e.g. **greengraph**) and copy/paste the data. You should get something like the following:



That is, there should be a speedup when comparing the green graph (CUDA version with 512 threads per block) and the blue graph (CUDA version with 1024 threads per block).

Change the number of threads per block from **512** to **256**:

```
vi fire-cuda.cu<ENTER>
:35<ENTER>
```

When you finish, the line should look like the following:

```
#define THREADS_PER_BLOCK 256
```

Save and close the file. Recompile the executable:

```
make<ENTER>
```

Run the **scale-cuda.sh** script again:

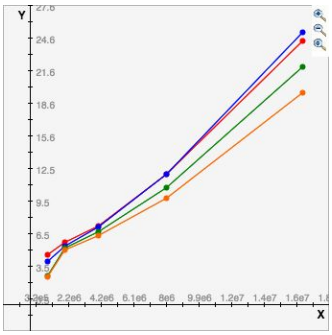
```
./scale-cuda.sh<ENTER>
```

Enter **y** to delete the **scale-cuda.out** file.

When the script finishes, display its contents:

```
cat scale-cuda.out<ENTER>
```

Copy and paste the data into Simple Plot using a new graph color (e.g. **orangegraph**). You should get something that looks like the following:



Notice that speedups continue as we continue to shape the thread blocks differently.

Change the number of threads per block to **128** and run the program with a problem size of **4000**:

```
make<ENTER>
time aprun -n 1 fire-cuda -r 4000 -c 4000 -t 4000<ENTER>
```

When the program runs, it creates one thread per tree. In this case, 16 million threads are created (4000 rows times 4000 columns). You should notice that the program runs quite quickly, but that the forest also burns **0.00%**, which is an error. The error occurs because the type of GPGPU we are using limits the number of thread blocks to 65,535 in each dimension. If we divide the number of threads including boundaries (4002 * 4002 = 16,016,004) by the current number of threads per block (128), we get a number of blocks larger than this number (125,125), thus the model silently fails. The maximum number of trees we can have if we use all 65,535 blocks with 128 threads per block is 8,388,480 (65,535 * 128). The square root of this is 2896, and if we subtract the 2 boundaries, this means the maximum problem size we can have for 128 threads per block is 2894.

For this particular program, there are problem sizes and process counts at which the MPI version is actually faster than the CUDA or OpenACC versions. See if you can find the number of MPI processes for which the program runs faster than the CUDA or OpenACC versions at problem sizes of 4000, 5000, 6000, and 7000.

Additional work

As with other versions of the program, an ASCII visualization can be generated using the **-o** option. This will not be covered in this exercise. If you would like to do this, see An OpenMP Exercise on Blue Waters for instructions, replacing the executable name **fire-omp** with **fire-acc** or **fire-cuda**.

Either in an interactive or batch job, you can explore running the **fire-acc** and/or **fire-cuda** executables with different parameters as explained in A Blue Waters Usage Guide**.**