Universiteit van Amsterdam
BSc Informatica
Dr. Robert G. Belleman

Concurrency en
Parallel Programmeren
2012/2013

Assignment 6

Many-Core Programming with CUDA

Getting started with CUDA on the DAS-4

For these assignments you will be using the GPUs installed at the DAS-4 VU cluster. The hostname for this cluster is `fs0.das4.cs.vu.nl`. Use the same login and password as you do on the DAS-4 UvA cluster.

For using the GPU nodes, we need a little bit of configuration. Add the following line to your `.bashrc`:

```
module load cuda50/toolkit
module load prun
```

Now, log out and log in again.

If all is ok, you should be able to run the CUDA compiler now, so please try:

```
nvcc --version
```

This should print:

```
nvcc: NVIDIA (R) Cuda compiler driver
Copyright (c) 2005-2012 NVIDIA Corporation
Built on Fri_Sep_21_17:28:58_PDT_2012
Cuda compilation tools, release 5.0, V0.2.1221
```

For running jobs, use this command:

```
prun -v -np 1 -native '-l gpu=GTX480' <EXECUTABLE> [arg1 arg2 ...]
```

Try, for instance:

```
prun -v -np 1 -native '-l gpu=GTX480' \
$CUDA_SDK/bin/linux/release/deviceQuery
```

If your job doesn't start immediately, you can check the queue status with

```
preserve -long-list
```

Testing your setup

The framework contains a directory `vector-add` with sample code for a simple vector addition. Compile and run that code first. Build the code with `make` and then run it with

```
prun -v -np 1 -native '-l gpu=GTX480' ./vector-add
```

The result should be something like:

```
vector add timer   : avg = 541 ms, total = 541 ms, count =  1
results OK!
```

Study the code in `vector-add.cu`, and copy/paste parts of this code into your assignment later, so you don't have to start from scratch!

Assignment 6.1: Wave equation simulation

Reconsider the 1-dimensional wave equation studied in assignments 1.1, 2.1 and 3.1. Parallelize your sequential simulation code using CUDA. Use the source code for `vector-add` as a starting point.

Repeat the experiments of assignment 1.1: time the performance of your implementation by running experiments with different problem sizes, i.e. number of amplitude points being 10^3, 10^4, 10^5, 10^6, 10^7. For the number of time steps, take the same as you did with your earlier experiments so that you can compare the results. For now, fix the number of threads per block to 512. Report the new results with those of your implementation from assignments 1.1, 2.1 and 3.1.

Experiment with different thread block sizes, using 8, 16, 32, … 1024 threads per block (on a GTX480). Report your results and explain why some schedulings perform better than others.

Compare the accuracy of the results generated by your CUDA implementation with those generated by your sequential implementation. Report your findings and explain.

Assignment 6.2: Parallel reduction in CUDA

Design, implement and test a program that uses a CUDA kernel to determine the maximum value of an array of float values. Hint: use a tree-based approach as explained during the lecture. Validate the correct outcome of the CUDA kernel by comparing it with the outcome of a serial CPU algorithm.

Run experiments that time the performance of both the CPU and the GPU algorithm. Report the results then make at least one refinement to your CUDA algorithm so that it performs better.

Assignment due date: December 10, 2012, 12:00.