Summer school *Getting Moore from Multicores*

Lab Session Day 3

General Purpose Computing on GPUs

Rob van Nieuwpoort
rob@cs.vu.nl

Pieter Hijma
hphijma@few.vu.nl

Alessio Sclocco
a.sclocco@vu.nl

July 5, 2011

UNIVERSITY OF AMSTERDAM

Vrije Universiteit Amsterdam

1. Getting started with GPUs and the DAS-4

For information about the DAS-4 supercomputer, please go to:
`http://www.cs.vu.nl/das4/`

You can use the `display` command for displaying images.
For information about the special GPU node hardware in the DAS-4 go to the DAS-4 site, then select

"Users → Special Nodes"

For more information on the software for programming GPUs navigate to

"Users → GPUs"

The host name of the VU cluster we are using is: `fs0.das4.cs.vu.nl`

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

```
module load cuda40/toolkit 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 Thu_Apr__5_00:24:31_PDT_2012
Cuda compilation tools, release 4.2, V0.2.1221
```

For running jobs, you can use this command:
`prun -v -np 1 -native '-l gpu=GTX480' <EXECUTABLE >`

Try, for instance:
`prun -v -np 1 -native '-l gpu=GTX480' $CUDA_SDK/C/bin/linux/release/deviceQuery`

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

```
preserve -long-list
```

UNIVERSITY OF AMSTERDAM

Vrije Universiteit Amsterdam

2. Playing with CUDA

In principle, everything you need for the GPU hands-on session is in:

```
/home/rob/multimoore/day3
```

The documentation for CUDA is in `/home/rob/multimoore/day3/cuda-documentation`. Especially the CUDA programming guide (`CUDA_C_Programming_Guide.pdf`) is a good starting point and reference for learning and using CUDA. In general, the CUDA documentation is excellent, so use it! You can view it with the "evince" program.

Example code for the assignment is in `/home/rob/multimoore/day3`.
You can copy this code to your own account, and play with it:

```
cp -r /home/rob/multimoore/day3 .
```

In the directory `day3/vector-add` is some example code for a simple vector addition.
Let's try to compile and run that code first. You can compile the code with "`make`". Now, you should be able to 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!
```

You can look at 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!

UNIVERSITY OF AMSTERDAM

Vrije Universiteit Amsterdam

3. The assignment

Just like on the first days, the assignment is based on a simple image processing application. The pipeline will convert a color input image to grayscale, compute its histogram, enhance the contrast, and perform edge detection. Thus, for any input image, the filter outputs a gray image (of the same size as the input) with the detected edges. For simplicity and accuracy all operations are done in floating point.

Converting a color image to grayscale
Converting a color image to a grayscale image is straightforward. The Red, Green and Blue (RGB) components of each pixel are weighted and added together: gray = 0.3*R + 0.59*G + 0.11*B.

Contrast enhancement
Next, the application will enhance the contrast of the image. This is done in two steps. First, we have to calculate a histogram. The histogram counts how often each gray scale value (between 0 and 255) is used in the image. We will use the histogram to determine a weight that is used to scale the gray values of each pixel. Second, each pixel in the image is scaled with the weight.

Edge detection and the Sobel operator
Edge detection is a fundamental tool in image processing and computer vision, particularly in the areas of feature detection and feature extraction, which aim at identifying points in a digital image at which the image brightness changes sharply or more formally has discontinuities (from Wikipedia).
The Sobel operator is used in image processing, particularly within edge detection algorithms. Technically, it is a discrete differentiation operator, computing an approximation of the gradient of the image intensity function. At each point in the image, the result of the Sobel operator is either the corresponding gradient vector or the norm of this vector. The Sobel operator is based on convolving the image with a small, separable, and integer valued filter in horizontal and vertical direction and is therefore relatively inexpensive in terms of computations. On the other hand, the gradient approximation which it produces is relatively crude, in particular for high frequency variations in the image (from Wikipedia).
Mathematically, the operator uses two 3 x 3 kernels which are convolved with the original image to calculate approximations of the derivatives - one for horizontal changes, and one for vertical. If we define A as the source image, and Gx and Gy are two images which at each point contain the horizontal and vertical derivative approximations, the computations are as follows (* here denotes the 2-dimensional convolution operation):

$$\mathbf{G}_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{bmatrix} * \mathbf{A} \quad \text{and} \quad \mathbf{G}_x = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} * \mathbf{A}$$

The x-coordinate is here defined as increasing in the "right"-direction, and the y-coordinate is defined as increasing in the "down"-direction. At each point in the image, the resulting gradient approximations can be combined to give the gradient magnitude, using:

$$\mathbf{G} = \sqrt{\mathbf{G}_x{}^2 + \mathbf{G}_y{}^2}$$

The sequential code is given in the directory `day3/seq-filters`. Please read the code carefully, and try to understand it. The assignment is to port this code to CUDA, and to parallelize it. A set of input images is provided in `day3/input-images`. You can again use "make" to compile the sequential version, and run it with:

```
prun -v -np 1 ./filters /home/rob/multimoore/day3/input-images/20.bmp
```

This should result in some timing results on your screen, and three output images (one for each step of the program): `gray.bmp`, `histogram.bmp`, `contrast.bmp`, `convolution.bmp`. You can look at the results with the "`display`" command.

Now, you can start your own CUDA version with the code in `day3/cuda-filters`, which is a copy of the sequential code, with a CUDA makefile. You can directly edit the file `cuda-filters.cu`. Don't forget that you can copy/paste some code from the vector-add example to get started.

Enjoy!

UNIVERSITY OF AMSTERDAM

Vrije Universiteit Amsterdam