CUDA Programming on NVIDIA GPUs
Mike Giles

Practical 1: Getting Started

This practical gives a gentle introduction to CUDA programming using a very simple code. The main objectives in this practical are to learn about:

- the way in which an application consists of a host code to be executed on the CPU, plus kernel code to be executed on the GPU
- how to create different kinds of executable using the Makefile
- how to copy data between the graphics card (device) and the CPU (host)
- how to include error-checking, and perform simple debugging using emulation

The CUDA SDK comes with a "master" Makefile called `common.mk`. The user's Makefile references this and specifies various files to be compiled, identifying which are CUDA files and which are regular C++ files, and setting various compiler flags.

To execute the Makefile there are 4 options

- `make` creates a standard CUDA executable
- `make dbg=1` creates an executable with error-checking enabled
- `make emu=1` creates an executable to be run under emulation (on the CPU)
- `make emu=1 dbg=1` creates an emulation executable with error-checking

The executables usually get put in subdirectories called `../../bin/linux/release, ../../bin/linux/debug,` etc., but a modification to the user's Makefile puts them instead in `bin/release, bin/debug,` etc., as subdirectories of the directory holding the Makefile and the source files.

Finally, the command

```
make -n
```

(which can be combined, if wanted, with `dbg=1` and/or `emu=1`) is helpful in showing what `make` would do if the `-n` flag were omitted. This shows how it compiled each of the files into object files (using `nvcc` for the CUDA files and, usually, `gcc/g++` for the plain C/C++ files, and then linking them all together with the relevant libraries to form the executable.

What you are to do is:

1. Following the above directions, produce the four different versions of the `prac1a` executable, and run each one.

2. Read through the `prac1a.cu` source file and compare it to the `prac1b.cu` source file which adds in error-checking.

3. Look at `Makefile` to understand how it works, and then modify it to produce executables for `prac1b`.

4. Try introducing errors into `prac1b.cu`, such as setting `nblocks=0`, and see what happens.

5. Add in `printf` statements in the kernel code in `prac1b.cu`. to print out the values of `threadIdx.x` and `blockIdx.x`.

   What happens when you try to compile it without emulation?

   What happens when you compile and run it with emulation?

6. Copy `prac1b.cu` to `prac1c.cu` and modify it to add together two vectors which you initialise on the host and then copy to the device. This will require additional memory allocation and two `memcpy` operations to transfer the vector data from the host to the device.

   Make sure you get the correct results, and use debugging in emulation mode if necessary to figure out what is going on.

7. If you have spare time, look at the NVIDIA SDK examples in http://www.nvidia.com/object/cuda_sdks.html