DEI-FCTUC

Computação de Alto Desempenho – 2012/2013
Exercises #1: Introduction to OpenCL

Some of these exercises were taken or inspired from several sources including the Web.

1. First, install OpenCL on your system. This usually requires nothing in Mac OS X (from Snow Leopard), CUDA Toolkit + Drivers for Nvidia GPUs and APP SDK + drivers for ATI/AMD GPUs. You must also install PyOpenCL.

2. Write a program that uses the OpenCL framework to inspect the hardware enabled for OpenCL that exists on your computer. To simplify your task, you can resort to online documentation, like "OpenCL for programming GPUs", from Pawel Pomorski (available in the course material).

3. Try the following program in PyOpenCL: http://documen.tician.de/pyopencl/.

4. Now, write a program in PyOpenCL similar to what you wrote in C to get the available platforms and their data. You should get the platform name, vendor, version, etc. For each platform you should get the devices, their names, types, memory, clock frequency and number of cores.

5. Write a program (C or Python) and an OpenCL kernel to compute the square root of each element in an array. You should print the results in the `main()` function. You can assume that the number of elements in the array is smaller than the number of threads.

6. The idea of this exercise is to multiply matrices, an operation that is pretty common to run in parallel. You should go through a number of steps. In the first step you should improve the CPU code, e.g., by transposing matrices and using tiles, to take advantage of the cache. Then, you should resort to the GPU to improve your best execution time. You have some helper material to simplify the process: the OpenCL setup code is ready, although there is no kernel (you should notice the `FILENAME` macro definition). You can find a naïve multiplication for the CPU and a number of macros and functions to manipulate matrices. You also have a python script (version 3 or above) to generate matrices.

7. This exercise has two different levels of achievement. Refer to the document "Optimizing Parallel Reduction in CUDA" by Mark Harris (available in the course material). In the slides the author has a huge list of 2^{22} integers to add. To start you can write down the optimization steps Mark Harris did to get the best performance out of the GPU. Then, you may follow his steps and optimize an OpenCL kernel yourself. Keep track of all the changes and performance numbers you get along the way.