Simula PyOpenCL Workshop (A. Klöckner)

Introductory Lab

This lab assignment is intended to help you get your feet wet with the basics of Python, numpy and GPU programming.

The problems get harder as you move along, and there is more than enough work here to keep participants at any level busy for the allotted time. So don't worry if you don't make it all the way to the end, and feel free to jump to the things that interest you most. Do come back and retry the problems that you didn't have time for.

Getting Started

Use of the GPU cluster is kindly provided by Tim Warburton at Rice University.

Follow these steps to get started:

1. Using an SSH client, log into `simula@haamster.rice.edu`.

   On Linux and OS X, this means that you need to type

   ```
   ssh simula@haamster.rice.edu
   ```
2. The password will be announced in class. Type it in.
3. Continue onward by typing "`ssh box`".
4. Run "`./which-gpu`" to determine which GPU you should run on.

   Important: Please run on the CPU ('AMD'/'Intel' platforms in the menu shown by PyOpenCL) while debugging and then run on the GPU only to get timings. This will help system responsiveness for everybody. Thanks!
5. Make your own directory by typing "`mkdir` *`firstname.lastname`*", where you substitute your first and last names into the directory name.
6. Change to that directory: "`cd` *`firstname.lastname`*".
7. Type "`nano hello-world.py`" (without the quotes) to start an editor on a new file.
8. Enter "`print "Hello World!"`", save, and quit.
9. In the terminal, type "`python hello-world.py`" and hit Enter. You've just run your first Python script on the machine–congratulations! Time to get started on the assignment.

One more request: GPUs can produce gigabytes of data very quickly. When writing files to disk, please take into account that storage space and disk bandwidth is shared among everybody in the room.

Problem 1: Python, Numpy and Project Euler

If we list all the natural numbers below 10 that are multiples of 3 or 5, we get 3, 5, 6 and 9. The sum of these multiples is 23.

Find the sum of all the multiples of 3 or 5 below $n = 1000$.

a) Write an iterative Python program to do this. (*Hints:* `range(5)` generates the list `[0,1,2,3,4]`, and '%' is modulo.)

b) Write a `numpy`-vectorized program to do this.

c) What happens if you increase n above? What is the relative speed of each solution? What is the scaling with n? Look up Python's `time`[1] library to do your timing. (*Hint:* Use `xrange`, not `range` for large values of n.)

d) (Slightly advanced:) The solution to this problem can be expressed in a single line (of 50-odd characters) in Python. How? (*Hints:* Search the Python documentation for the words 'generator expression'. Python has a function '`sum`'.)

Problem 2: Simple GPU Programming with PyOpenCL

Write a PyOpenCL code that solves Problem 1. You may start from the simple example in the PyOpenCL documentation[2].

a) Think of a way to parallelize this. In your first attempt, you may use some OpenCL and some host-side computation.

b) Gradually increase n. You likely want to make sure that you are using 64-bit integers (always '`long int`' in OpenCL) beyond a certain value of n to avoid integer overflow.

   What performance results do you expect? Benchmark your solution using synchronization (`queue.finish()`) and Python's `time`[3] library. Do your measurements match what you expect? Also try measuring using PyOpenCL `Events`. Are the results any different?

c) Investigate the use of `pyopencl.array` and `pyopencl.elementwise`.

d) Benchmark the array-based solution. What do you observe? In each of your codes, what machine feature determines your performance?

e) Investigate the use of `pyopencl.reduction` to move the computation entirely onto the GPU.

Hint: Use your solution to Problem 1 to check your answers.

Problem 3: Better know a GPU

Let us try and measure a few properties of the GPU you are using.

a) Try to measure the transfer bandwidth between host and device. Is it the same in both the reading and writing direction?

b) Try to measure the device memory bandwidth. How does this number depend on alignment? How on the stride from one work item to the next? Does the number you measured match the published peak numbers? (see Wikipedia[4])

c) Can you measure the maximum flop rate? What obstacles do you encounter?

Hint: See this article[5] for inspiration.

[1] `http://docs.python.org/library/time.html`
[2] `http://documen.tician.de/pyopencl`
[3] `http://docs.python.org/library/time.html`
[4] `http://is.gd/fldxGv`
[5] `http://dx.doi.org/10.1109/ISPASS.2010.5452013`