CS61C Spring 2014 Project 3 Part 2: Digit Recognition Parallelization on GPU

TAs: Sagar Karandikar, Roger Chen

Updates

- The spec has been updated to note that you may perform determining if a computed distance is the minimum distance on the CPU.

Background

Your objective, just like in part 1, is to parallelize the digit recognition code you wrote for Project 1, but now with the GPU. If you're unsure of the operation you need to perform, please refer to the Project 1 Spec.

Architecture

What follows is information concerning the computers you will be working on. Some of it will be useful for specific parts of the project, but other elements are there to give you a real world example of the concepts you have been learning, so don't feel forced to use all of the information provided.

You will be developing on Dell Precision T5500 Workstation. They are packing not one, but two Intel Xeon E5620 microprocessors (codename Westmere). Each has 4 cores, for a total of 8 processors, and each core runs at a clock rate of 2.40 GHz.

All caches deal with block sizes of 64 bytes. Each core has an L1 instruction and L1 data cache, both 32 Kibibytes. A core also has a unified L2 cache (same cache for instructions and data) of 256 Kibibytes. The 4 cores on a microprocessor share an L3 cache of 12 Mibibytes. The L1, L2 caches are 8-way associative, while L3 is 16-way associative.

In this part however, we additionally have access to the Nvidia Tesla C1060 Graphics Co-processor installed on the Hive machines. We'll use this co-processor to perform all of our computation in this part of the project. For more information about the Tesla C1060, see Lab 10. You MAY NOT use the CPU to perform any of the heavy lifting.

Getting Started

For this project, you should work in groups of two (however, you may work solo). You are not allowed to show your code to other students, loved ones, pets, or any other individuals.

You should aim to start this project early. Since each Hive machine has only one C1060 co-processor, performance will decrease substantially when many people try to test as we approach the due date.

The Usual Disclaimers

Looking at solutions from previous semesters or code found on the internet is also strictly prohibited, unless we've explicitly given you access to a resource (see the resources section below). We have various tools at our disposal to detect plagiarism and they are very good at what they do.

Additionally, tampering with machines in any way to gain an unfair advantage - for example spamming other students' terminal sessions - is strictly prohibited (and considered abuse of campus resources). We can track anything that happens on your `cs61c-XX` account and you will be caught if others complain. Remember that you are responsible for anything that happens on your account: "my friend did it" is not an excuse.

Prepping Your Environment

To begin, copy the contents of the `~cs61c/proj/03-2` directory.

```
$ mkdir ~/proj3-2
$ cd ~/proj3-2
$ cp -r ~cs61c/proj/03-2/* ~/proj3-2
```

Project Files

The following files are provided (the only one you need to look at is `calc_dist.cu`):

- `Makefile`: Defines all the compilation commands.
- `benchmark.c`: Contains the benchmarking code that measures the performance of your program. Feel free to modify this to try additional sizes.
- `calc_dist.cu`: Unlike in Part 1, we've given you this file, since you'll likely rewrite most of your code. This is where you'll place your GPU solution. This is the only file you'll be allowed to turn in.
- `digit_rec.h`: Defines template width and height and function signatures related to digit recognition.
- `digit_rec.c`: Loads bitmap images and calls `calc_min_dist()` to calculate distances. **Running this will fail on your code for this part of the project, since we only require support for a subset of template sizes (see below).**
- `launch_gpu.cu`: A wrapper around calc_min_dist that allocates memory on the GPU and copies data from CPU addressible memory.
- `launch_gpu.h`: A header file for launch_gpu's use.
- `oracle.h`: Header file (function signatures) for the oracle. See `oracle.o` below.
- `oracle.o`: Computes the correct output for a given image and template. Allows you to check your output results by comparing against a guaranteed working solution.
- `test_digitrec.c`: Examines the calculated distances and prints out the digit recognition result, which is the digit with shortest distance to the test image.
- `utils.h`: The `Image` struct and utility function signatures.
- `utils.c`: Bitmap loading and printing functions.
- `templates/`: Contains the template images.
- `basic_tests/`: Contains test images for basic test cases (eg. only rotated, etc).

You will place your solution in `calc_dist.cu`. **This is the only file you will be permitted to submit, so you should place all of your code here. Code in other files will be overwritten or ignored.** More about your specific task is discussed in the following section.

Please post all questions about this assignment to Piazza, or ask about them during office hours.

Part 2: Due 4/23/2014 @ 23:59:59 (50pt)

Describing the computation:

- `NxM` is the size of your image. N is the width and M is the height.
- `TxT` is the size of your template. Templates are always square, so only one variable is required.

Your task is to implement your `calc_min_dist` function on the GPU. To avoid solutions that are excessively inefficient, your code must achieve the performance specified for the kernels given below. However, your code should also work for any kernel whose width is a power of two in the range [16, 4096] (inclusive).

You can develop on any machine that you choose, but your solution will be compiled using the given Makefile and will be graded on the Hive machines. Therefore, you should perform all of your testing on a Hive machine (most other inst machines will not work anyway, since they lack `nvcc` and a compatible GPU), since performance is critical. **If your code does not compile on Hive, you will receive no credit.**

Checking Performance

To compile/run your improved program, do the following:

```
$ make
$ ./benchmark
```

Since we are dealing with very large templates and images, runs will take a rather long time (especially the last few sizes in the benchmark). A full run of the benchmark for a sufficiently performant solution will take a minute or two.

Disregarding performance, your code must be able to produce correct output for any valid (larger than the template) image size and templates with widths that are powers of two between [16, 4096] (inclusive bounds). Passing this requirement is worth 18 points.

Additionally, you should tune your solution to maximize performance for the cases in the table below. Each requirement in this rubric is worth 8 points, for a total of 32 points.

Performance Requirements

Template size, M	512	1024	2048	4096
N	612	1124	2148	4196
Performance Threshold (Gflops)	1	2	2.5	2.5

Intermediate Gflop/s point values are determined as follows: score = floor((your gflops / threshold)*(8 points)).

For this project, you may not perform any computation using values from the input image or template on the CPU. The only computation directly derived from values in the image/template that you may do on the CPU is selecting a minimum given a distance and the old minimum. Also, attempting to optimize by changing compile-time parameters is not allowed - we will be replacing your `Makefile` with our own when we grade your project.

Checking Correctness

You should make sure that your optimized code still produces the correct output. The `./benchmark` will compare the result of your `calc_min_dist` to a known working version in `oracle.o`, but **the benchmark does not guarantee correctness**. You should check correctness with the basic tests provided in the skeleton.

```
$ make check
```

Tips

Below are some things you **MUST** pay close attention to when programming on the GPU:

1. You should call `cudaThreadSynchronize()` after invoking the kernel. Normally, CUDA kernel invocations are asynchronous, meaning that program flow continues on the CPU after the kernel is launched on the GPU. Unless you know what you are doing, you should ALWAYS call this function after a kernel invocation.
2. You should also call `CUT_CHECK_ERROR("")` after invoking the kernel. This will print out information about errors that happen in the kernel. Not doing so may cause errors in the kernel to happen silently (for example stepping out of bounds of an array may produce no error, but your result will be incorrect).
3. In order to keep your computation manageable, you should take advantage of multiple dimensions of the grid (unlike in Lab 10).
4. You may wish to allocate "scratch space" on the GPU. You may do this with `cudaMalloc()`. See Lab 10 for more about `cudaMalloc()`.
5. Be sure to take advantage of the defined geometry of the GPU to simplify your computation and to allow computation on larger templates (e.g. 4096x4096).
6. Need to brush up on GPU Programming / need hints? See Lab 10 or the Resources section at the bottom of this page.

Extra Credit (Due Wednesday April 30th @ 23:59:59)

For this (optional) section, you may use any tools you have at your disposal to make your code as fast as you possibly can for varying sizes (not disclosed in advance). A small amount of extra credit will be given to the top performers. The only caveat is that your code must compile with the given Makefile (either the one given in part one or part two). For this section, you must submit either `calc_dist.c` or `calc_dist.cu`. If you're using the part 1 Makefile and `calc_dist.c`, submit with the command `submit proj3-ec-cpu`. If you're using the part 2 Makefile and `calc_dist.cu`, submit with the command `submit proj3-ec-gpu`. Solutions that do not compute the correct results will be immediately disqualified.

Submission

The full proj3-2 is due Wednesday. To submit the full proj3-2, enter in the following commands. You will only need to turn in `calc_dist.cu`.

```
$ cd ~/proj3-2
$ submit proj3-2
[Follow the prompts]
```

References

1. Nvidia GPU Computing Intro Slides
2. Nvidia CUDA Programming Guide
3. Specs by Compute Capability (Wikipedia) (The Tesla C1060 has Compute Capability 1.3)
4. Stack Overflow on Understanding CUDA geometry
5. Nvidia GPU Floating Point Accuracy Issues
6. Nvidia CUDA Reduction Slides