Jake Orben

Final Progress Report

Dr. Papamicd

CSC 470

**Summary**

The task I was given was to take an algorithm and use GPU processing to parallelize the process of finding the edit distance between two strings of genetic code. In the end this task was beyond me, as I was unable to reconstruct an algorithm that could run the code in parallel. I went through many iterations of my code as well as a plethora of online sources, however, I was unable to pull a working program together.

**Introduction**

The field of parallel computing is a broad and interesting topic. The section of this field which I worked on was Graphic Processing Unit parallelization or GPU parallelization. This field requires the use of specialized languages, such as OpenCL or CUDA. OpenCL is an open source language that can be run on any dedicated GPU, whereas CUDA is focused on the Nvidia chipset. Nvidia is one of the two major dedicated GPU designers, the other being ATI. I personally used CUDA, as I have an Nvidia chipset and it allows for easier coding as well as better performance than OpenCL does using the Nvidia chipset.

**Report**

I began this report with both high hopes, as well as a touch of joy, as I have always wanted to learn CUDA, however I never had a reason to do so. I spent the first week of the project familiarizing myself with the language as well as creating some small programs to test my skills and gain an understanding of the concepts I would be working with. At the end of the week I received the algorithm that I would be working with and set straight to work.

I used several sources to complete this project, a vast amount of information came from the Internet and the rest came from Nvidia’s book on coding in CUDA. Many of the Internet resources showed me how to move code between the host and the “devices”. The host being the code run on regular RAM and processed by the CPU, the devices being a given GPU(s). Memory management here is quite important, as the GPU does not have nearly as much memory as the host does at its disposal, however, processes run in parallel are many times faster than code run on a single core of a CPU. One of the best examples I found involved turning a regular C program for calculating the addition of vectors into a CUDA program. In this tutorial, I learned a great deal about transferring data between the host and the kernel/device. The program originally called for a table of size 1024, however, with some work, I was able to figure out how to calculate a table of size 2048, as well as theoretically any multiple of the aforementioned numbers. The book from Nvidia was also quite helpful as it showed, from a beginner’s level, how to perform basic to advanced programming in the CUDA language. With all of this information I proceeded to coding for the actual program for edit distance.

I started off by looking at the source code for the Needleman method of calculating edit distance. This was a small challenge at first as I was rusty with the C programming language, but once I understood the concepts of the program, I began to parallelize it. I performed this task by using the CUDA programming language. In this language, it is possible to run a program in parallel across numerous “blocks” and their sub divisions “threads”. The threads are where the work is done while using a parallelized system, they complete tasks individually on their own, each one launching an instance of the “kernel”, which is where the majority of calculation is performed. The kernel is where one puts code that is to be run in parallel; the data generated from this method is then transferred back out to the “host” or main system. This is where I ran into several issues, as threads and blocks execute, their functions are not performed at exactly the same time. As a result, some of the data is misrepresented. If one thread needs data from another to continue, it will not wait, it will simply continue on as if it is executing normally. This causes issues, as required data is then misinterpreted, leading to a vast amount of erroneous data, making the output garbage. I tried a variety of methods to remedy this issue, from using a thread sync, which synchronizes all threads in a block, to moving the entire data structure into the GPU’s memory to speed up calculations. In spite of these efforts, I was unsuccessful, and still have incorrect output despite the many hours (well over 50) I have put into this project.

**Conclusion**

In summary, I spent a great deal of time trying to parallelize the Needleman method of calculating edit distance. However, my skills in the subject, along with the difficulty of the issue I was dealing with were beyond me. I still plan to work on this program until it works; however, I am currently at the deadline for writing the code. You can most likely expect an e-mail from me over winter break containing a working copy of the code required.