Computer Science Tripos – Part II – Project Proposal

Parallelising Sequence Alignment

B. Marshall, Sidney Sussex College
Originator: Mr P. Rugg
25 October 2019

Project Supervisor: Mr P. Rugg

Director of Studies: Mr M. Ireland

Project Overseers: Prof J. Daugman & Dr A. Madhavapeddy

Introduction

Sequence alignment is an important tool in Bioinformatics, used to find relationships between sequences of DNA, RNA, or proteins. The molecules in question are long polymers, made up of a sequence of different much smaller molecules. In DNA and RNA, there are four possible monomers called nucleotides, and in proteins there are twenty naturally occurring monomers called amino acids. Pairwise sequence alignment involves taking two sequences of monomers, and determining which monomers in each sequence correspond to one another and where monomers must have been inserted or deleted.

This is a valuable tool for Bioinformaticians, and finding sequence alignments is a computationally challenging problem. The different biological meanings of different types of sequence can be simplified into two sequences of characters, with some kind of scoring function to evaluate the quality of the alignment. Find an optimal alignment is often solved using a dynamic programming approach, building up possible alignments in a grid and the alignment with the best score is chosen when the grid as been populated. This is the Smith-Waterman algorithm. When evaluated sequentially, the execution time is $O(nm)$ where n and m are the lengths of the two strings. However, as the grid is populated the number of cells available to be evaluated grows, so the potential to parallelise execution grows as execution proceeds.

This can reduce execution time to $O(n)$, provided that there is sufficient hardware to run m threads parallel to one another.

This project aims to explore the differences between hardware architectures such as general purpose processors and graphics processing units, and as an extension FPGAs. Graphics processing units are capable of evaluating hundreds of instructions simultaneously, which I expect will give it a significant advantage over general purpose processing units. I expect an FPGA will prove to be an interesting platform to investigate, because of the potential to be able to make all of the calculations and decisions for a cell in the dynamic programming grid, in a single clock cycle. The hardware logic on an FPGA can be focussed on a single problem, unlike other platforms like GPUs where there will logic not being used by my software, which should allow the FPGA to consume less power.

I will produce software or hardware logic that find sequence alignments, and then compare the different implementations using metrics such as time taken to find alignments, electrical power usage and hardware costs.

Starting Point

I have pre-read the relevant sections of the Bioinformatics course (and those lectures will have been delivered by the end of the October), so I understand the basic algorithms underpinning sequence alignment. I have previous experience writing C code from the Programming in C and C++ course from Part IB, and wrote some during a summer internship as well. I have very little experience using GPU APIs such as CUDA, having written a very small amount of GLSL in the Introduction to Graphics course, so I will need to learn how to use CUDA as part of this project. During the ECAD and Architecture course I worked with an FPGA, and have written SystemVerilog as a part of that course, though I have no experience with FPGAs beyond that course.

I have read a few papers about parallelising sequence alignment, and it is worth noting that a similar project was undertaken by Benkrid et al. in 2012 [1]. However, their work is several years old and different conclusions might be drawn when using more modern hardware, as I intend to do. Additionally, I intend to evaluate performance more thoroughly, and investigate factors that may have an impact on that performance, such as alignment metrics or the impact of different compilers.

Work to be done

The implementation phase of the project breaks down into the following sections:

1. Implementing the Smith-Waterman algorithm to find sequence alignments using single-threaded C code.
2. Modifying the C code to make it multi-threaded.
3. Implementing the Smith-Waterman algorithm using CUDA to run on a GPU.

Extensions

My primary extension will be to implement the Smith-Waterman algorithm on an FPGA, and I hope to complete this extension. However, it is not a primary goal because of my unfamiliarity with FPGAs having only done ECAD and Architecture labs, and this will be much more complicated than anything I have attempted to implement on an FPGA before.

Another extension is to modify my C implementation to use vectorised instructions, exploiting SIMD parallelism in a similar way to my CUDA implementation.

Success criteria & Evaluation

The goal of this project is to compare the effectiveness of different computer processing technologies, particularly focussing on producing parallelised implementations. This focus on parallelism is to make the project relevant to modern machines, because GPUs are designed to perform hundreds of operations concurrently and modern CPUs all have multiple cores. This is in part due to Moore's Law slowing down, and parallelism is increasingly becoming the approach used to increase the performance of computers instead of shrinking silicon process nodes.

My success criteria are to produce implementations of the Smith-Waterman algorithm in single-threaded C, multi-threaded C, and CUDA, and to thoroughly evaluate their performance. This will involve finding how the time it takes to find sequence alignments changes for different lengths of sequences, and comparing the performance to approximate power usage. If I produce

an implementation to be run on an FPGA, I plan to evaluate that in a similar fashion. For all of the solutions I implement, I intend to implement a variety of alignment-scoring functions (simple edit distance, different gap penalties, substitution matrices such as BLOSUM, etc.), and compare the performance impact that these different functions can have on these different platforms. As extensions to the evaluation, I could investigate the impact of using different compilers and optimisation modes has on the runtime of my solutions.

Work Plan

As a student taking CST Part II 50% taking a similar number of Paper 8 and 9 courses in Michaelmas and Lent terms, I have a consistent lecture load throughout the year with no particularly quiet nor intense periods which students taking Part II Units of Assessment might experience. Therefore, I will not be accounting for differences in lecture load in my timetable. However, I am taking three papers at the end of the year so I intend to have handed in by dissertation by 24th April (in Week 1 of Easter term), giving me the whole term to focus on my last lecture series and revision.

14th Oct – 27th Oct

Write my proposal, with a draft handed in on 18th Oct and the final version handed in by 25th Oct. General project setup such as installing compilers, setting up version control arrangements, sorting out LaTeX typesetting for my proposal and dissertation.

Milestone: Final Proposal submitted.

28th Oct – 10th Nov

Implement Smith-Waterman algorithm as single-threaded C program, and find a subset of protein sequences from UniProt [2] for testing. These will be used to evaluate the correctness of all of the subsequent implementations I make.

Milestone: Single threaded C program written, with tests for correctness.

11th Nov – 24th Nov

Modify the single-threaded C to run using multiple threads. Begin writing the Preparation chapter of my dissertation.

Milestone: Multi-threaded C program written, tested for correctness.

25th Nov – 8th Dec

Write a basic CUDA implementation of the Smith-Waterman algorithm. Continue writing the Preparation chapter of my dissertation.

Milestone: Basic CUDA implementation, tested for correctness.

8th Dec – 21st Dec

Improve my CUDA implementation of the Smith-Waterman algorithm, considering branches and memory layouts. Finish first draft of the Preparation chapter. I doubt this will take the full two weeks, but it provides some space to catch up if things have slipped behind. If this is not necessary, I can begin work on the next milestone.

Milestone: Finished CUDA implementation, finished Preparation chapter draft.

6th Jan – 19th Jan

These slots are dedicated to extensions, but would also serve as slack time if things are running behind from the beginning of the project. Start with adding vectorised CPU instructions to my C code (which should require a similar approach to my CUDA application). After that, begin work on the FPGA extension: design logic to find small sequence alignments, small enough that the problem doesn't need to be split into subproblems. Sequences will be hard coded instead of taking them from an external source such as the ARM CPU on the same chip as the FPGA. This logic only needs to be simulated for this milestone. Begin work on progress report.

Milestone: Basic FPGA implementation made and testing in simulation.

20th Jan – 2nd Feb

Finish and rehearse progress report. Continue work on FPGA extension: Work on synthesising logic and running it on the FPGA. Add the ability

to for the ARM core on the same chip to deploy and collect work from the FPGA, using it as a hardware accelerator.

Assess at the end of this block how well the FPGA extension is going, and whether to continue with it.

Milestone: Hand in progress report. Synthesise logic from last milestone and run on the FPGA. Sort out the interface between ARM core and FPGA on the development board.

3rd Feb – 16th Feb

Work on the ability to divide up sequences on the FPGA so sequences of arbitrary length can be aligned, using a scaled down model in simulation. At the end of this work package, stop work on extensions regardless of their progress to focus on writing my dissertation.

Milestone: Deliver progress report. Implementation phase of project complete.

17th Feb – 1st Mar

Begin writing implementation chapter of my dissertation, and start evaluating my work. The range of test-sequence lengths required should be apparent from the approximate performance observed when testing the implementations. I will consult other papers to decide on lengths of test sequences that might represent a real-world work load as well, to add context to my results.

Milestone: Numerical evaluation of C code complete.

2nd Mar – 15th Mar

Finish writing draft of implementation chapter of my dissertation, and finish numerical evaluation. Begin writing Evaluation chapter.

Milestone: Implementation chapter drafted. Numerical evaluation of CUDA code (and FPGA logic if it is working) complete

16th Mar – 29th Mar

Finish writing the Evaluation chapter, and write Introduction and Conclusion chapters. Work on any feedback on my dissertation I receive.

Milestone: Introduction, Conclusion, and Evaluation chapters drafted.

30th Mar – 12th Apr

Finish any writing that may have slipped behind, and submit a draft to my supervisor and Director of Studies for feedback.

Milestone: None

13th Apr – 26th Apr

Final work based on feedback, then submit dissertation.

Milestone: Dissertation submitted.

Resources

I plan to use my own laptop for this project. It has a hexa-core CPU (an Intel i7-8750H), 16 GB RAM, 512 GB SSD and an NVIDIA GTX 1050 Ti Max-Q graphics chip, running Windows 10. I accept full responsibility for this machine and I have made contingency plans to protect myself against hardware and/or software failure.

All work towards the project and dissertation will be held under Git with a copy hosted on Github, and any work done each day will be pushed to this repository. Onedrive will also be used for real-time file mirroring, meaning I won't depend on a single cloud service.

In the event that I won't be able to use my laptop, I have an old laptop (dual-core CPU, 8 GB RAM, integrated Intel graphics chip) to use for development and writing, and I could use MCS machines as well. I will be able to pull my work from the cloud and continue working with minimal disruption.

Computationally difficult experiments may be slow on these devices, so the Computer Architecture group has agreed to provide me with access to a machine to evaluate my C implementation on.

Another important resource the Computer Architecture group has agreed to provide me with is an access to an FPGA. This is only required for an extension to the project, but time allowing it is something I would like to attempt. The DE1-SoC boards used in ECAD+Arch labs by Part IB students are ideal, with relatively small FPGA chips so I will be able synthesise my work on my laptop in reasonable periods of time.

References

[1] Khaled Benkrid et al. "High Performance Biological Pairwise Sequence Alignment: FPGA versus GPU versus Cell BE versus GPP". In: *International Journal of Reconfigurable Computing* 2012 (Feb. 2012). DOI: 10.1155/2012/752910. URL: http://dx.doi.org/10.1155/2012/752910.

[2] The UniProt Consortium. "UniProt: the universal protein knowledgebase". In: *Nucleic Acids Research* 46.5 (Feb. 2018), pp. 2699–2699. ISSN: 0305-1048. DOI: 10.1093/nar/gky092. eprint: http://oup.prod.sis.lan/nar/article-pdf/46/5/2699/24388239/gky092.pdf. URL: https://doi.org/10.1093/nar/gky092.