Computer Science Tripos

Part II Project Proposal Coversheet

Please fill in Part 1 of this form and attach it to the front of your Project Proposal.

Part 1

Name: Marcus Alexander Karmi September

CRSID: maks2

College: Clare

Overseers: (Initials) R. M. & A. M. P

Title of Project: A parallel algorithm for all-pairs shortest paths that minimises data movement

Date of submission: 16.10.2021

Will Human Participants be used? No

Project Originator: Dr Jagdish Modi

Project Supervisor: Dr Jagdish Modi

Directors of Studies: Prof L. C. Paulson

Special Resource Sponsor:

Special Resource Sponsor:

Part 2

Overseer Signature 1: ----------

Overseer Signature 2: ----------

Overseers signatures to be obtained by Student Administration.

Overseers Notes:

Part 3

SA Date Received:

SA Signature Approved:

Part II Computer Science Project Proposal

A parallel algorithm for all-pairs shortest paths that minimises data movement

Marcus A. K. September, Clare College

Originator: Dr J. Modi

October 16, 2021

Project supervisor: Dr J. Modi

Director of studies: Prof L. C. Paulson

Project overseers: Prof R. Mantiuk & Prof A. M. Pitts

Introduction

The all-pairs shortest path (APSP) problem is relevant for minimising resource costs when travelling across a road network or a railway network. There are many known serial algorithms for solving this problem, but with a highly parallel processor, there might be more efficient solutions. After multiplying a graph's adjacency matrix with itself n times, it is possible to find the shortest paths of length n between all pairs of nodes. This can be used to construct an algorithm that solves APSP. Matrix multiplication is also a highly parallel problem, meaning there is a lot of potential gain from running it on a multiprocessor.

Many massively parallel processors have hundreds or thousands of CPUs, each with their own private memory. The CPUs then pass data and work to each other using interconnect channels. For problems where the input data is so large that it needs to be distributed across the CPUs private memory, the communication cost is often the performance bottleneck. Therefore, algorithms built for these multiprocessors need to minimize the amount of memory movement. For matrix multiplication, this can be done by employing techniques discovered by G. C. Fox, S. W. Otto, and L. E. Cannon [1, 2].

This project aims to develop an algorithm to solve APSP using matrix multiplication. Since I do not have access to a physical massively parallel processor, it will be simulated in Java. I will then parallelize the matrix multiplication step of the algorithm, and use above mentioned optimizations to minimize the data movement. If time allows, I intend to explore whether the advantage of parallelising APSP is still significant when the problem size is orders of magnitude larger than the number of CPUs in the multiprocessor. The matrix multiplication based algorithm could also be compared with a parallelised implementation of the Floyd-Warshall algorithm, as the two are similar in nature. Further extension work includes optimising the algorithm through graph compression and investigating the effect of different processor interconnect topologies.

Starting point

- I have experience with writing modular, object-oriented code in Java from the Object-Oriented Programming course and the Further Java course.
- I have some knowledge of parallel programming concepts from the Concurrent and Distributed Systems course. I also have a bit of experience using these concepts in code through the Further Java course.
- I know some of the principles behind parallel processors through the Computer Design course

- I am familiar with some route-planning algorithms like Dijkstra's algorithm through the Algorithms course

Resources required

I will use my own computer (an Acer Nitro Laptop), running both Ubuntu Linux 18.04 LTS and Windows 10, to both write and run all of my code. It has a quad-core CPU (an Intel i5-8300H @ 2.30Ghz), 8 GB of RAM, and an NVIDIA GTX 1050 Mobile. I accept full responsibility for this machine and I have made contingency plans to protect myself against hardware and/or software failure. In case of failure, I plan to use the machines provided by the MCS for development.

The source code, associated data, and my dissertation will be regularly version controlled using a private `git` repository hosted by GitHub. This repository will also regularly be copied over to the Computer Laboratory MCS server.

Substance and structure of the project

The aim of this project is to implement a matrix multiplication based algorithm that solves APSP, and to parallelise the matrix multiplication step to run efficiently on a massively parallel processor with a distributed memory model. That is, the processor will have a large number of processing elements and each of these will have their own private memory. The core part of the project can be divided into several phases:

Data preparation and research The project will start of with a research phase where I look into parallel computing concepts, specifically the importance of minimizing memory movement, and techniques for parallelising matrix multiplication [2, 1]. Research will be done on techniques for evaluating the performance of parallel systems and algorithms, which will be used when planning the evaluation. Additionally, I will plan the different software modules of my project through use of UML diagrams.

The graphs I intend to use will be taken from road network datasets found at [3]. A set of graphs with a wide range of sizes is required for rigorous evaluation of the algorithm. To acquire this, I will create a program that extracts subsections of the datasets. The nodes in the graphs are labelled with their longitude and latitude, so this could for instance be done by removing all the nodes that fall outside of some arbitrarily drawn geographic circle.

APSP algorithm In this step, I will solve APSP using repeated matrix multiplication of the graph's adjacency matrix. The algorithm should also be able to reconstruct the list of nodes that make up the shortest paths. I will start with a serial matrix multiplication routine, and test the algorithm for correctness on small graphs. Later, the matrix multiplication routine will be parallelised and optimisations such as Fox-Otto's will be applied [1].

Simulating multiprocessor and preparing evaluation Since I do not have access to a massively parallel processor, I will simulate one using Java's `Thread` API. Each processing element will be represented by a Java `Thread`, and this will be abstracted away in this phase. Each simulated processing element will have its own private memory, consisting of a few Java variables. The only memory movement will be processing elements sending variables to each other or receiving data from some manager `Thread`. I do not intend to simulate the more complicated aspects of a processor, such as memory mapping, caches and job scheduling.

The framework I develop in this phase will also be used for my quantitative evaluations. One approach is to include a timer in each Java thread and take the maximum of these times after each computation phase. By doing this for all the computation phases of the algorithms – which have communication phases in-between – I would get an estimate of the computation time. For the communication time, I could count the number of memory transfers in the communication phase and estimate how long this would take by using realistic values for memory latency for existing multiprocessors.

Parallelising matrix multiplication In this phase I will implement Fox-Otto's algorithm using the framework developed in the previous phase [1]. After this, the implementation will be integrated into the APSP algorithm.

Possible extensions

1. Generalise the implementation of Fox-Otto's algorithm to also work when the number of processing elements is smaller than the problem size. Then move onto evaluating the advantage of parallelization as this ratio ($\#\ nodes\ /\ \#\ CPUs$) increases.
2. Optimise the APSP algorithm using graph compression techniques, such as removing redundant nodes.
3. Look into different processor network topologies, and how they affect the communication cost of running Fox-Otto's algorithm.

4. Parallelise Floyd-Warshall's algorithm and compare the computation and communication cost with the matrix multiplication-based APSP algorithm.

Success criteria

The project will be considered successful if all of the following criteria are met:

- Implemented an algorithm based on matrix multiplication that can find the length of the shortest path between all pairs of nodes in a graph, and it is able to give the list of nodes that make up such paths.
- Parallelised the matrix multiplication routine of the algorithm to run on a simulated massively parallel processor, where each processing element can send data to each other through simulated interconnects.
- The parallel matrix multiplication routine is optimised to minimise the amount of data movement between processing elements, which is done by using techniques such as Fox-Otto's algorithm [1].
- The evaluation of the algorithm demonstrates that parallel computation gives a high parallel efficiency and a significant speedup, compared with theoretical serial implementations, for solving APSP

Timetable

Week	Date	Description
1 – 3	7 Oct - 27 Oct	**Project proposal and research** • Incorporate feedback on project proposal draft and hand in the final project proposal • Research parallel computing concepts, parallel matrix multiplication algorithms like [1, 2], and how to evaluate parallel systems • Write small fragments of Java code that incorporates these concepts • Set up version control and sort out typesetting of dissertation **Milestones** • Project proposal handed in by October 18th • GitHub repository set up
4 – 5	28 Oct - 10 Nov	**Data preparation and planning** • Create program that extracts subsection of graphs from [3] • Plan evaluation of the algorithm, using knowledge obtained in the research phase • Plan the structure of the code, and how the components interact **Milestones** • Data extractor program written • UML diagrams of core project components created

6 – 8	11 Nov - 1 Dec	*Due to my unit of assessment, productivity during this slot might be lower.* **Simulating multiprocessor I** • Write simulation framework that allows running many CPUs in isolation, measuring their computation time • Write code fragments to test this **Simulating multiprocessor II** • Incorporate an interconnect topology into the framework • Allow the simulated CPUs in the multiprocessor to communicate by sending data to each other through this interconnect network • Create methods to find metrics for the communication cost • Write code fragments to test this **Milestones** • Simple multiprocessor simulation written • Simulation extended to support threads passing each other data • Quantitative computation and communication cost metrics are produced when running test fragments
9 – 11	2 Dec - 22 Dec	**APSP algorithm** • Write serial matrix multiplication routine • Implement algorithm to compute all pairs shortest path using matrix multiplication • Make algorithm reconstruct the list of nodes in the shortest paths • Test algorithm for correctness **Milestones** • APSP algorithm written • Algorithm produces correct results on small graphs
12 – 13	23 Dec - 5 Jan	*This time slot will work as slack time if the project is behind* **Christmas break**

14 – 15	6 Jan - 19 Jan	**Implement efficient parallel matrix multiplication and progress report** • Implement Fox-Otto's algorithm, using the multiprocessor simulation framework • Write progress report **Milestones** • Written parallel matrix multiplication algorithm • Progress report submitted by Friday February 4th
16 – 17	20 Jan - 2 Feb	**Evaluation** • Evaluate the performance of the parallel APSP algorithm, and compare it with a theoretical serial implementation • Compare the advantage of parallelism with theoretical serial implementations as the problem size increases relative to the number of processing elements (extension) **Milestones** • Theoretical comparison between parallel implementation and serial alternative has been made • Quantitative measurement of the performance of the parallel implementation has been done

18 – 20	3 feb - 16 Feb	*This timeslot is dedicated to extension work. If the project is running behind, this slot will instead serve as slack time.* **Extensions** • Generalise Cannon's or Fox-Otto's algorithm to work when the number of CPUs do not match the problem size • Do qualitative evaluation on the algorithm's performance as the ratio # $nodes$ / # $CPUs$ increases. • Optimise the algorithm using graph compression • Investigate the performance effect of different interconnect topologies • Parallelise Floyd-Warshall's algorithm and compare its performance with the core algorithm **Milestones** • Theoretical evaluation of the advantage of parallelism as the ratio # $nodes$ / # $CPUs$ increases. • Quantitative measurements of the algorithm's performance as this ratio changes have been made • The performance effect of graph compression has been measured quantitatively • A theoretical and/or quantitative evaluation of the effect of different interconnect topologies have been made • Written parallel implementation of Floyd-Warhsall's algorithm
21 – 22	24 Feb - 9 Mar	**Dissertation writing I** • Write first draft of dissertation • Send draft to Director of Studies **Milestones** • Completed first draft of dissertation

23 – 24	10 Mar - 23 Mar	**Dissertation writing II** • Incorporate feedback from first dissertation draft, producing a second draft **Milestones** • Completed second draft of dissertation
25 – 26	24 Mar - 6 Apr	**Dissertation writing III** • Repeatedly review the dissertation • Refine the layout and the diagrams **Milestones** • Submitted final dissertation by May 13th

References

[1] G.C Fox, S.W Otto, and A.J.G Hey. Matrix algorithms on a hypercube i: Matrix multiplication. *Parallel Computing*, 4(1):17–31, 1987.

[2] H. Gupta and P. Sadayappan. Communication efficient matrix-multiplication on hypercubes. Technical Report 1994-25, Stanford Infolab, 1994.

[3] F. Li. Real datasets for spatial databases: Road networks and points of interest. https://www.cs.utah.edu/~lifeifei/SpatialDataset.htm, 2005.