Part II Computer Science Progress Report

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

Marcus A. K. September, Clare College
maks2@cam.ac.uk

February 1, 2022

Project supervisor: Dr J. Modi

Director of studies: Prof L. C. Paulson

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

Progress report

The project is currently on schedule, and work items have been completed as they were described on the timetable. There has not been any unexpected difficulties with implementation. Additionally, all the milestones of the work items leading up to, but not including, evaluation have been met. I am currently doing evaluation, which is in line with the timetable as the period 20 Jan – 2 Feb is set of for this.

I have implemented a python script and Java class that, respectively, downloads and transforms various graph datasets into the appropriate format to feed into the all-pairs shortest paths (APSP) algorithm. I have also finished the multiprocessor simulation; It now provides the programmer with an interface where they can program what a general processing element (PE) $PE(i, j)$ should do during its computation and communication phases, having access to methods such as `send(other_i, other_j, data)` and `broadcastRow(data)`. This description can then be passed onto a `Manager` which instantiates `Worker`s according to the description, loads the initial memory content, and runs them until they have completed a specified number of work phases. It also handles their communication using a `MemoryController` that I have implemented. I have also implemented the `FoxOtto` algorithm, which runs on the multiprocessor simulator. Additionally, I have finished implementing the main APSP algorithm, which uses `FoxOtto` min-plus matrix multiplication as a subroutine. I have also thoroughly tested the whole algorithm for correctness by manually creating small example graphs and verifying that the shortest paths found are correct, and I have also run it on a large graph with 250 nodes and compared the results with the output of a serial Dijkstra algorithm I am certain is correct.

Currently, I am working on the evaluation of the algorithm. I have implemented timing and communication-counting wrappers, and used these to estimate the computation and communication time required by the algorithm. I have also been researching how the algorithm implemented would map onto real multiprocessor hardware, and looked into what the communication latency and bandwidth are on such hardware. This will be used to create a more realistic estimate of the communication time. What remains to be done as part of evaluation is to do more measurements for graphs of various sizes to determine how the algorithm's performance scales.

Going forward, I will finish evaluation, then aim to complete two of my extensions, one of which I'm more than halfway done with: Graph compression through removal of 2-degree nodes. The other extension is generalizing the implementation to allow each PE to handle a sub-matrix of size greater than 1×1. Some weeks of February are set off for extensions, so this work will be according to the initial work plan.