THE UNIVERSITY OF SYDNEY

High Performance Execution of Matrix Languages

Dominic Balasuriya (dbal7610@it.usyd.edu.au)
Supervisor: Dr. Bernhard Scholz (scholz@it.usyd.edu.au)
School of Information Technologies

FACULTY OF ENGINEERING & INFORMATION TECHNOLOGIES

1. Introduction

- We have achieved high performance execution of matrix operations using a new computation engine that runs in parallel on multicore CPUs.
- This is the first parallel system of its kind for Octave, an open source alternative to MATLAB and a language used by scientists and engineers for executing matrix operations.
- We have also developed a novel system for switching between matrix representations in a data dependence graph.
- By choosing the best representation for each operation, we minimise the overall execution time of all operations.
- This system has a number of general applications, including switching between different computation engines.

2. Background

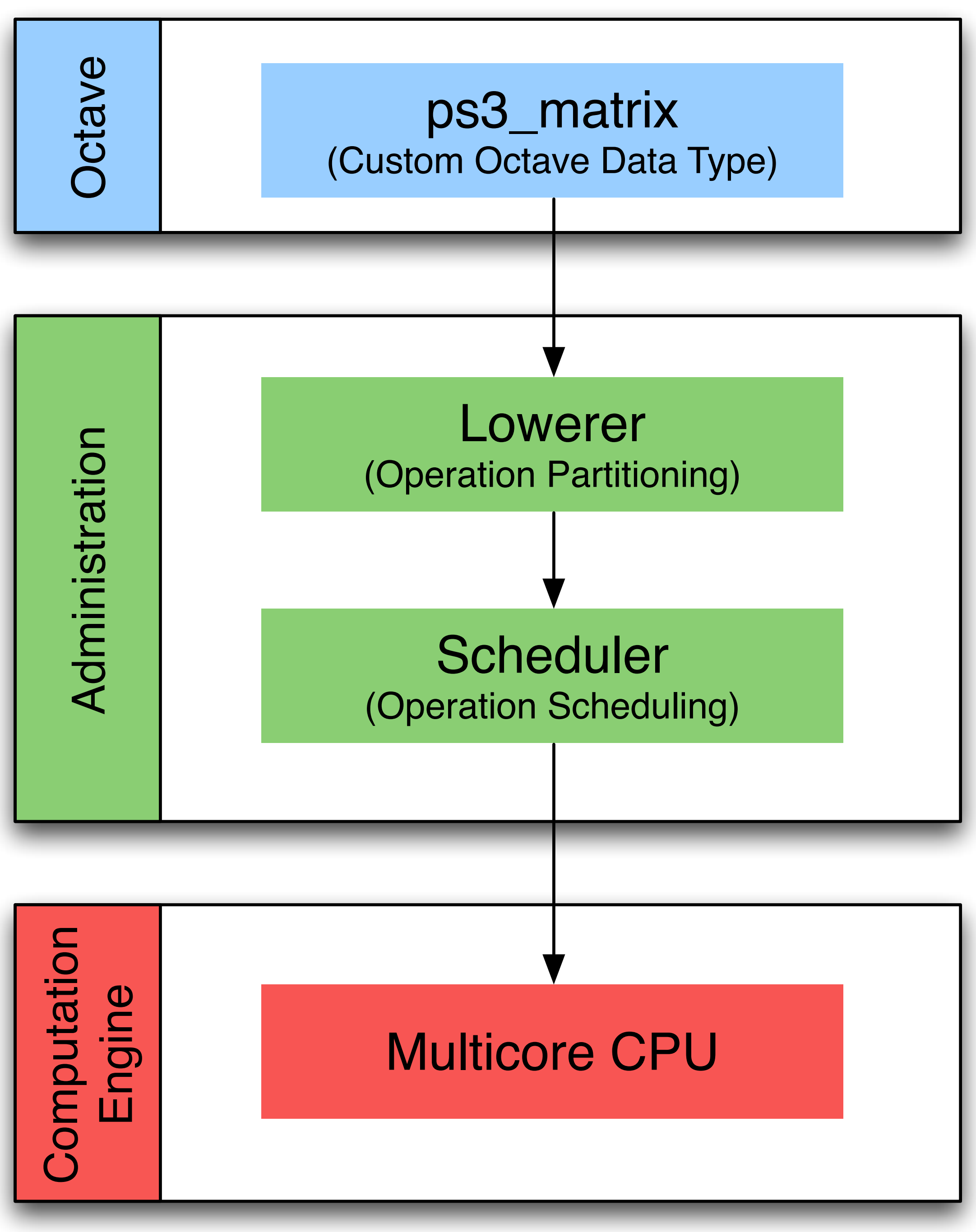
- This project extends work by Khoury et al. (2010), who created the PS^3 framework for accelerating Octave code using the Cell Broadband Engine. This is a multi-core chip used in the Sony's PS3 game console.

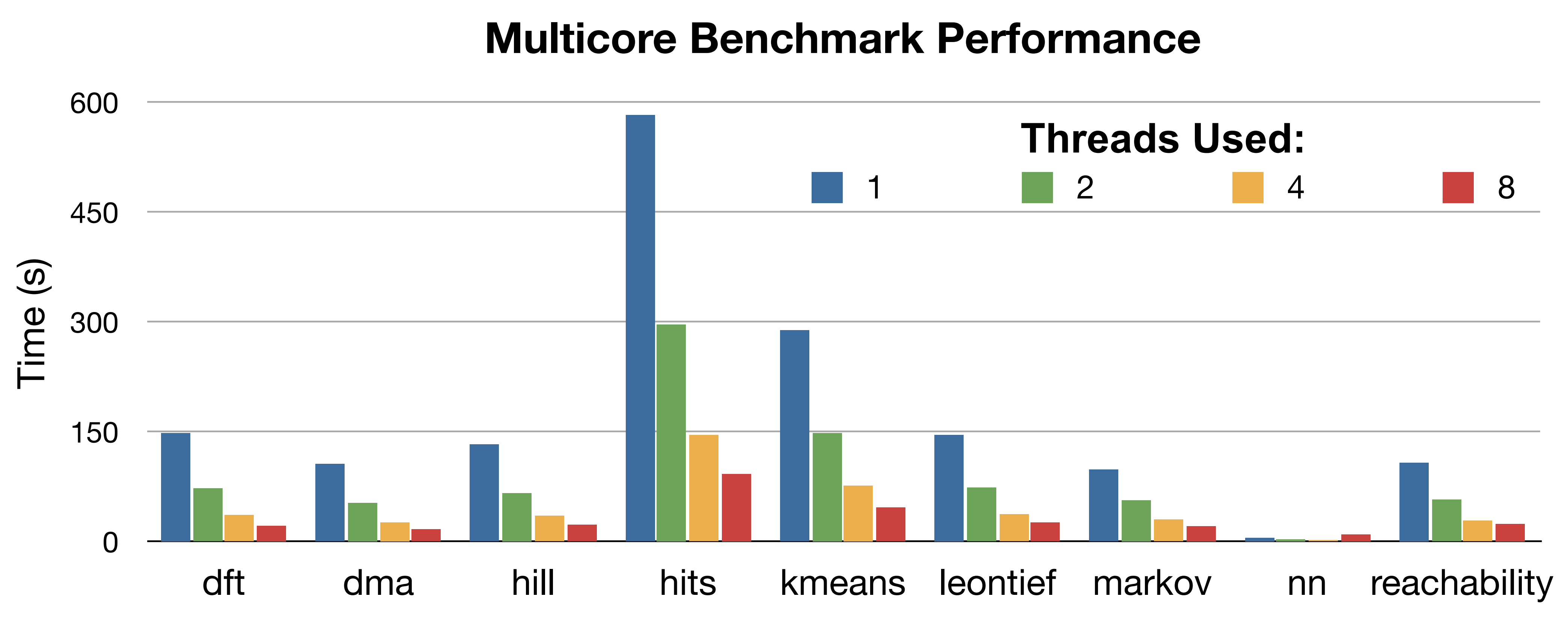
Motivation

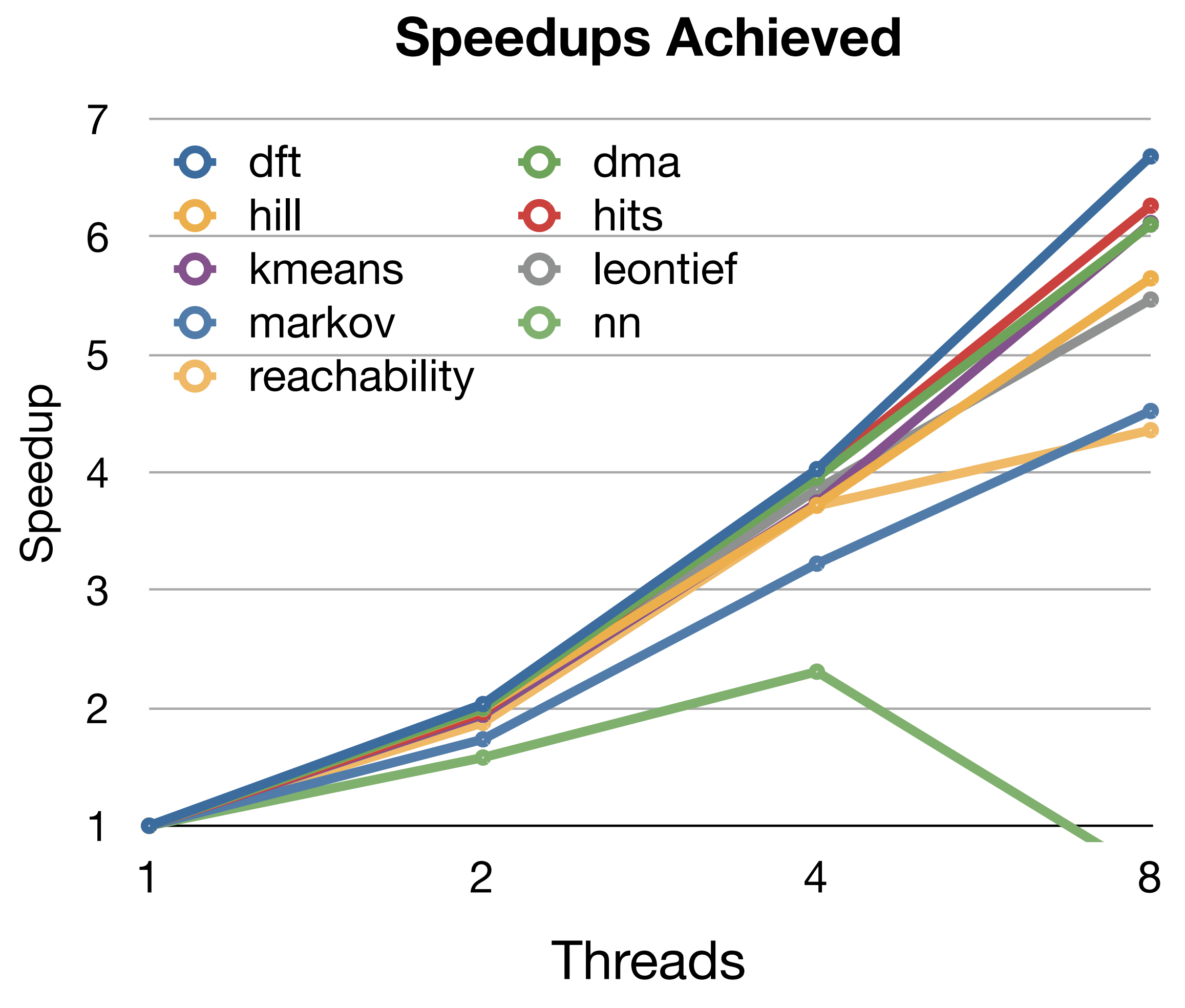
- The execution of matrix operations can benefit greatly from parallelisation. However, languages like MATLAB and Octave, which are used extensively by scientists and engineers, do not support parallel execution.
- Programming for parallel architectures is difficult because of issues such as race conditions and deadlocks. Octave users should not have to do this manually - therefore automatic parallelisation is needed.

The Framework

- The framework has three major components: Octave Extension, Administration and Computation Engine (see below).
- The Octave extension creates a data dependence graph from the Octave code. The optimal scheduling and partitioning of operations is then determined and the operations are then executed on the computation engine.







3. Multicore Computation Engine

Computation Engine

- The computation engine executes operations in parallel by using several CPU threads.
- The engine receives a vector of matrix operations from the scheduler, one for each thread.
- Each thread executes this sequence of operations, observing the dependencies between operations. If operands are not available, then the thread spin locks until it can proceed.
- Modifications were also made to the lowerer. Unlike the Cell's SPEs, which have limited memory, CPU threads have access to main memory. It is therefore possible to divide matrices into much larger blocks.
- Experiments were conducted to determine the architecture-specific optimal block size.

Results

- A series of nine benchmarks were used to test the system's performance on two 2.33GHz Intel Xeon E5245 Quad-Core CPUs. A block size of 10,000 floats was found to be optimal for this architecture.
- The results of these benchmarks (top of page) suggest that all benchmarks experienced substantial speedups. In particular, when fully utilising the Quad-Core CPU using four threads, the majority of benchmarks showed a speedup close to 4x.
- When eight threads were used, speedups of close to between 5 and 7x were achieved, which suggests exceptional utilisation of CPU resources.
- The exception to this was the nn (neural networks) benchmark, which slowed down when 8 threads were used.

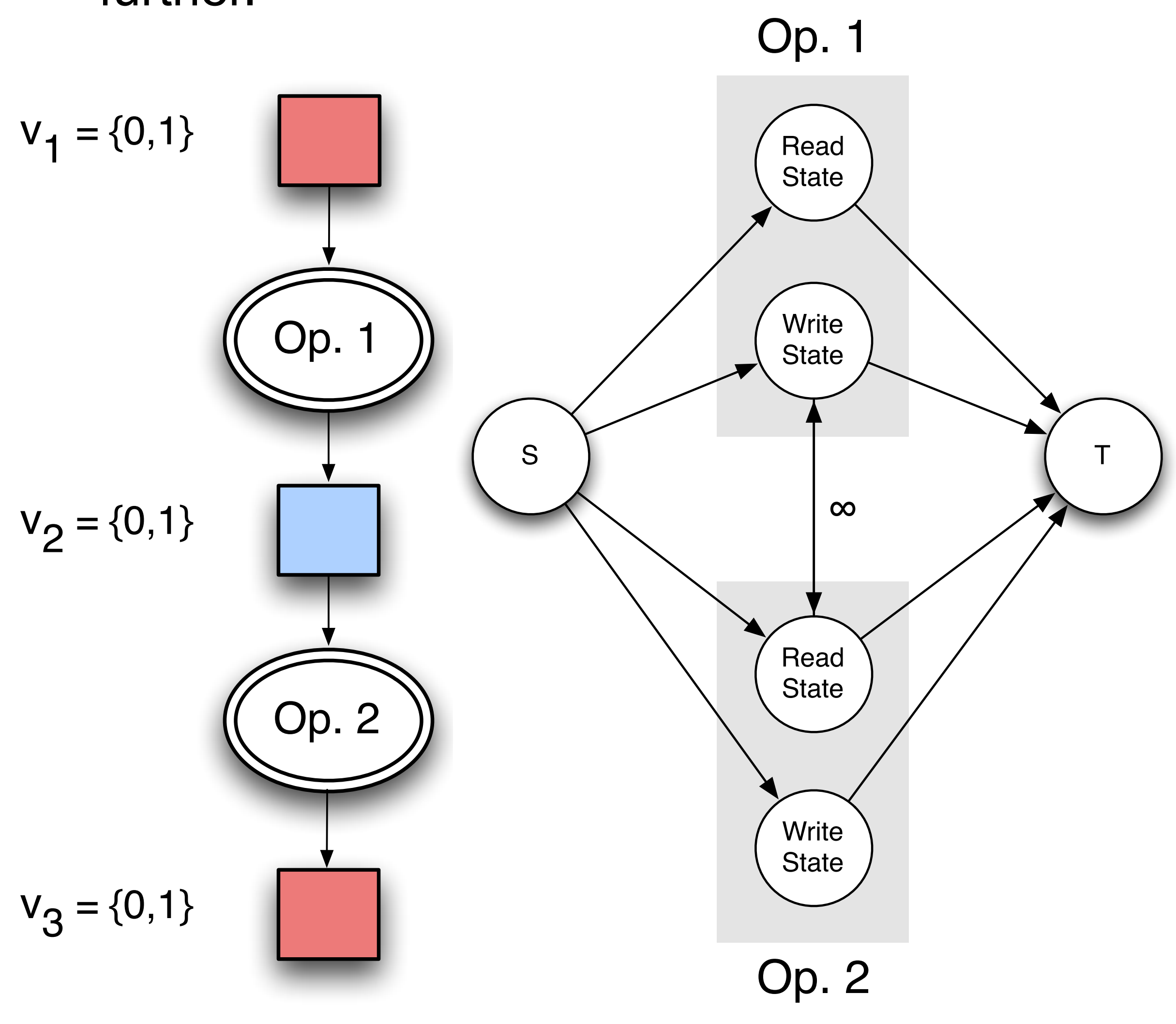
4. Switching Framework

Motivation

- It is possible to represent matrices in memory in multiple configurations. Due to CPU cache effects, the order in which matrix elements are accessed affects read and write times.
- We implemented two matrix representations: row-major and submatrix contiguous. Our hypothesis was that some operations would be read/written faster in one representation than the other, allowing us to switch between them.

The Switching Framework

- To determine the optimal reading and writing representation for each operation, we developed a framework that extends the metric labelling problem, using an ST-mincut to determine which representation should be used for each operation (see below).
- Unfortunately, the representations we chose to implement proved to have very similar performance across all operations.
- However, the novel switching framework that we have developed has far more general applications, including switching between different computation engines (e.g. CPU and GPU) and representations which are significantly different.
- We were able to use synthetic experiments to explore the efficiency of this framework further.



References

Khoury, R., Burgstaller, B. and Scholz, B. (2010) Accelerating the Execution of Matrix Languages on the Cell Broadband Engine Architecture. *IEEE Transactions on Parallel and Distributed Systems.*