Steal Tree: Low-Overhead Tracing of Work Stealing Schedulers

Jonathan Lifflander (UIUC), Sriram Krishnamoorthy (PNNL), Laxmikant V. Kale (UIUC)

Annotated Example Program

```
fn() {
  s1;
  async {
    s5;
    async w;
    s6;
  }
  s2;
  finish {
    s7;
    async x;
    s8;
    async y;
    s9;
    async z;
    s10;
  }
  s3;
  async { s11; }
  s4;
}
```

Introduction

Work stealing is a popular approach to scheduling task-parallel programs, used in many programming models including OpenMP 3.0, Java Concurrency Utilities, Intel TBB, Cilk, and X10. The flexibility inherent in work stealing when dealing with load imbalance results in seemingly irregular computation structures, complicating the study of its runtime behavior. We present an approach to efficiently trace async-finish parallel programs scheduled using work stealing.

Tracing Algorithms

Rather than trace individual tasks, we exploit the structure of work stealing schedulers to coarsen the events traced. In particular, we construct a steal tree: a tree of steal operations that partitions the program execution into groups of tasks. We identify the key properties of two scheduling policies that enables the steal tree to be compactly represented.

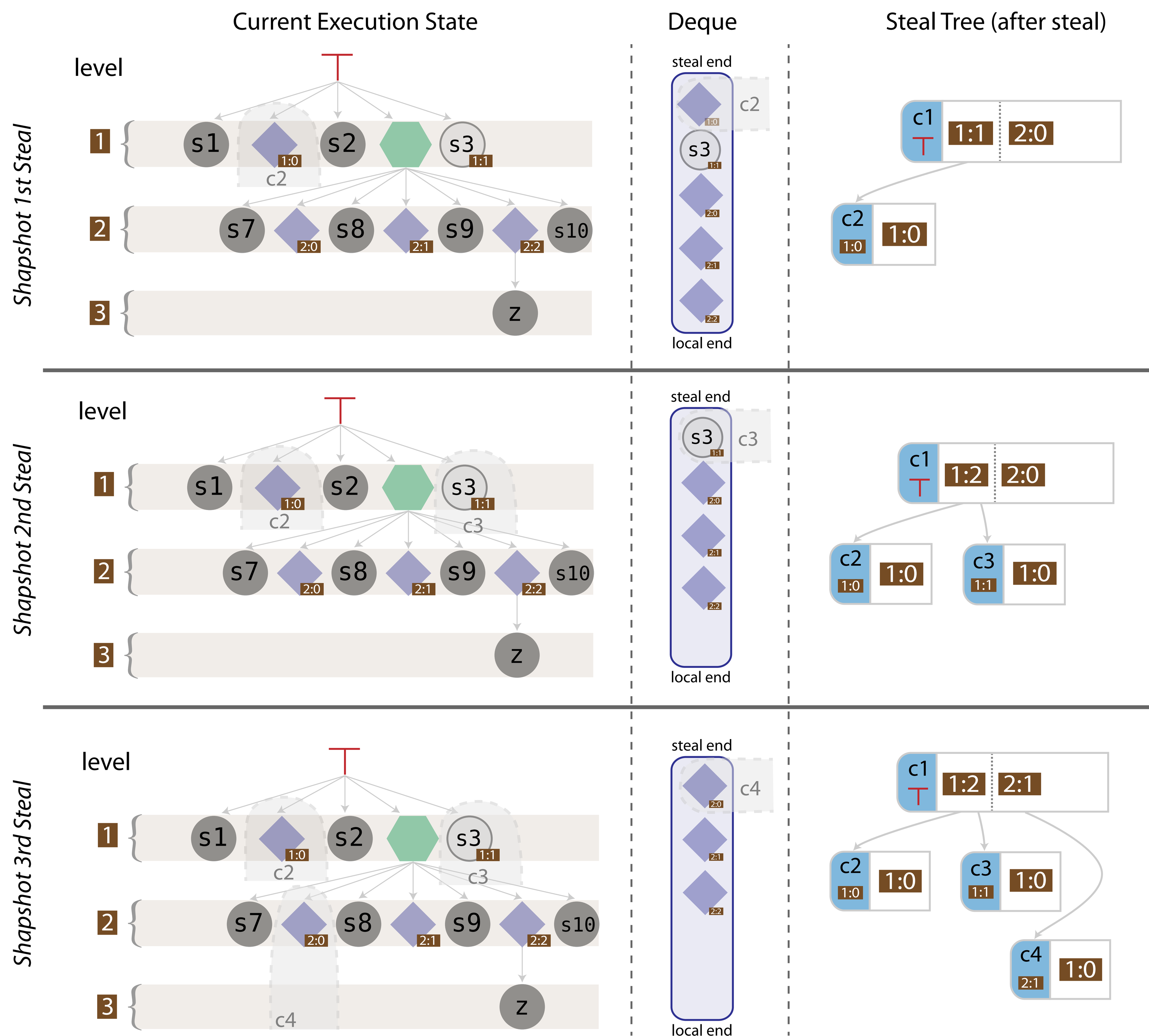
Primary Contributions

- Identifying the key properties of work-first and help-first schedulers to compactly represent the stealing relationships.
- Algorithms that exploit these properties to trace and replay async-finish programs by efficiently constructing the steal tree.
- Demonstration of low space overheads and within-variance perturbation of execution in tracing work stealing schedulers.
- Applying the tracing to two distinct contexts: data-race detection and retentive stealing

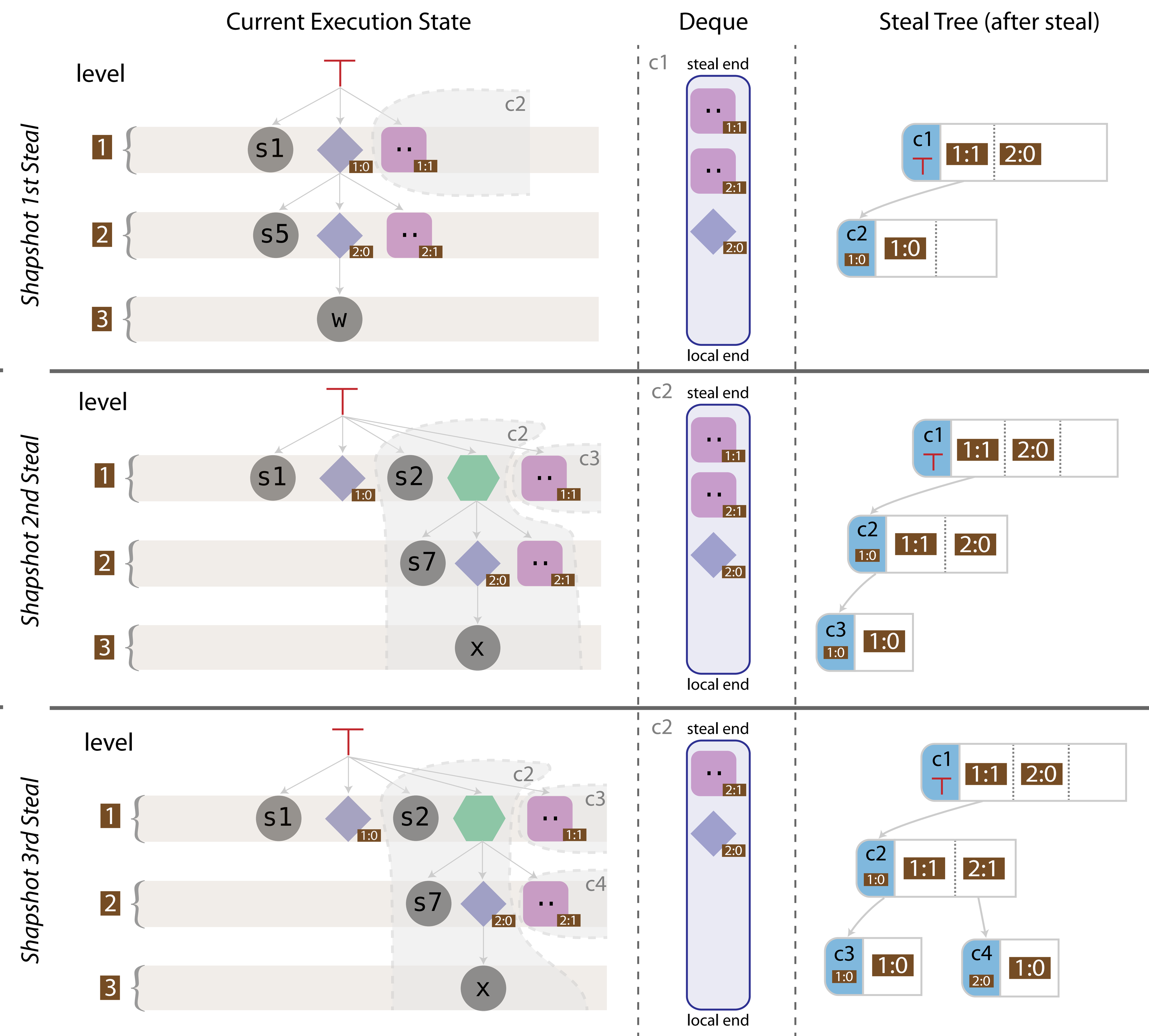
Applications

- We show a substantial reduction in the cost of data race detection using an algorithm that expoits our traces to maintain and traverse the dynamic program structure tree (DPST). In some cases, we observe up to a 70% reduction in tree traversals.
- Using our traces we are able to reduce the memory requirements for retentive stealing by not explictly enumerating tasks.

Help-first Scheduling Policy

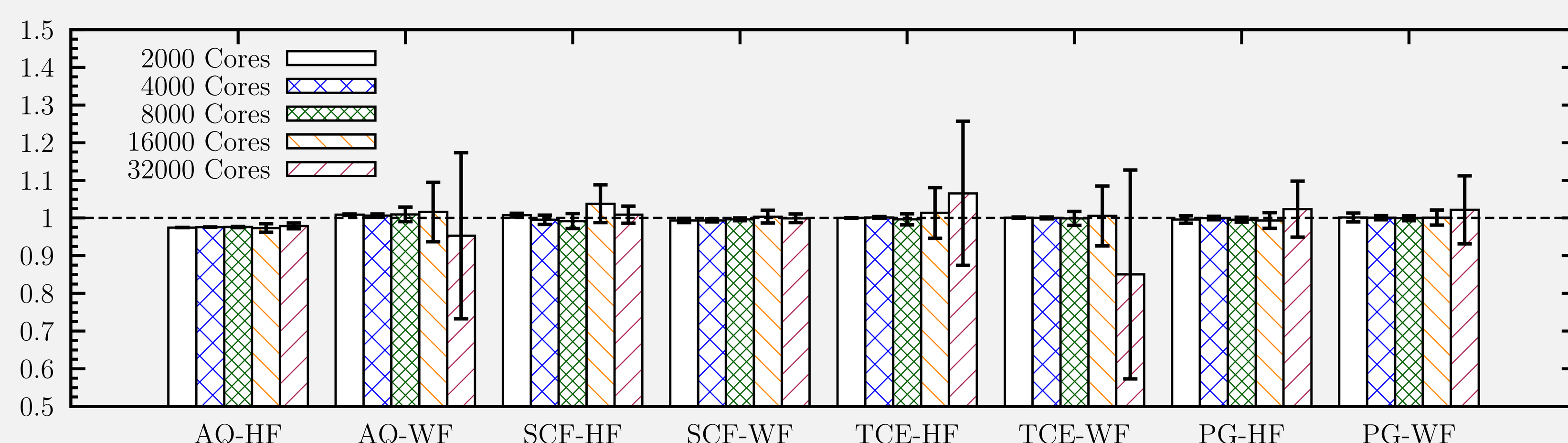


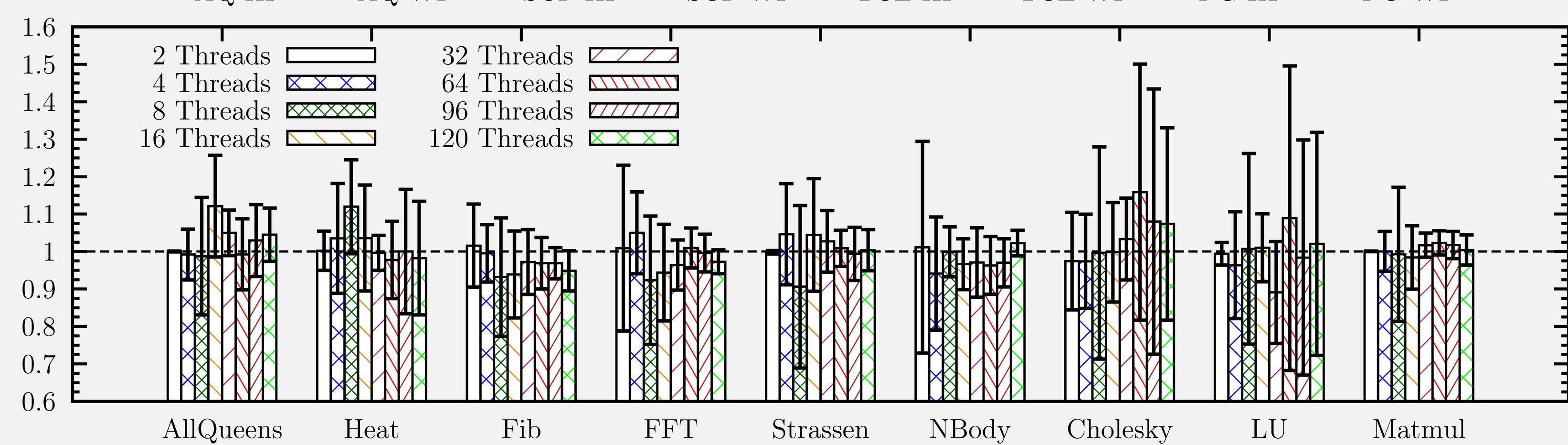
Work-first Scheduling Policy



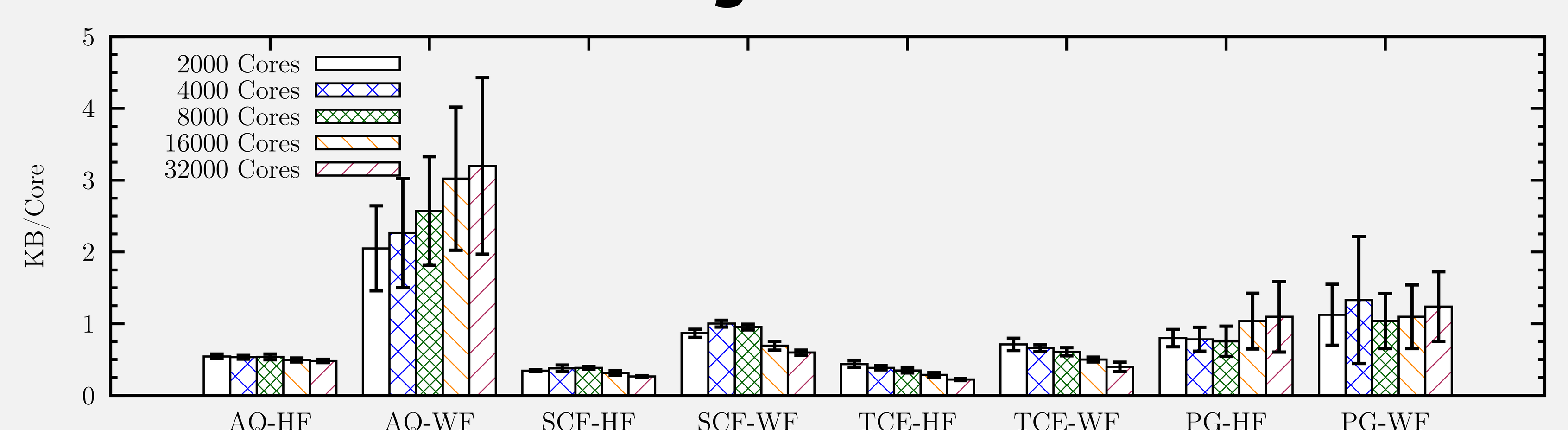
Results

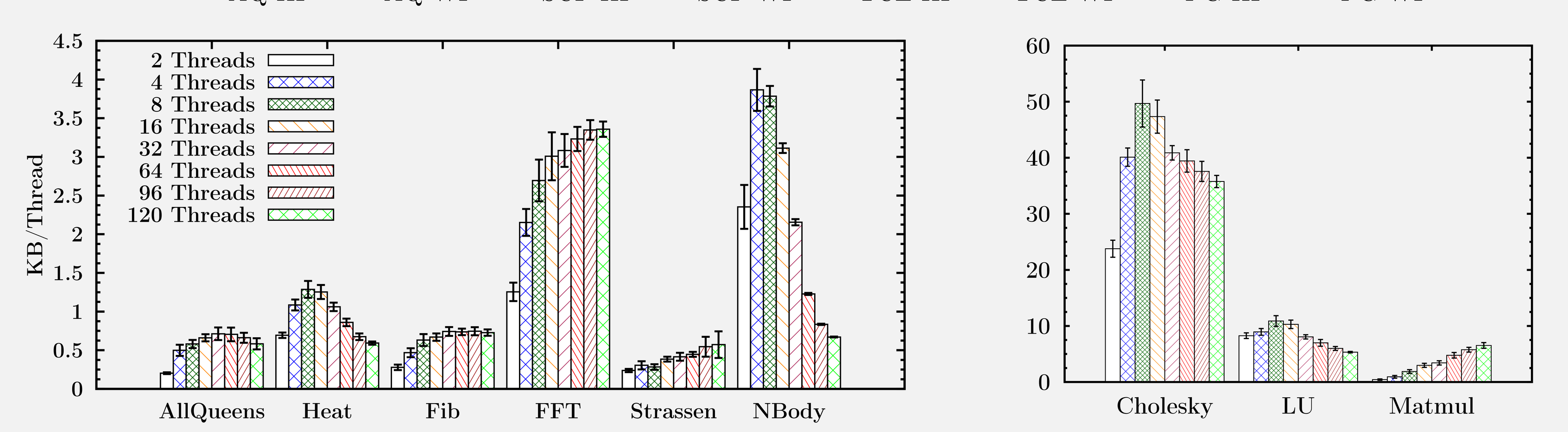
Execution Time Overhead



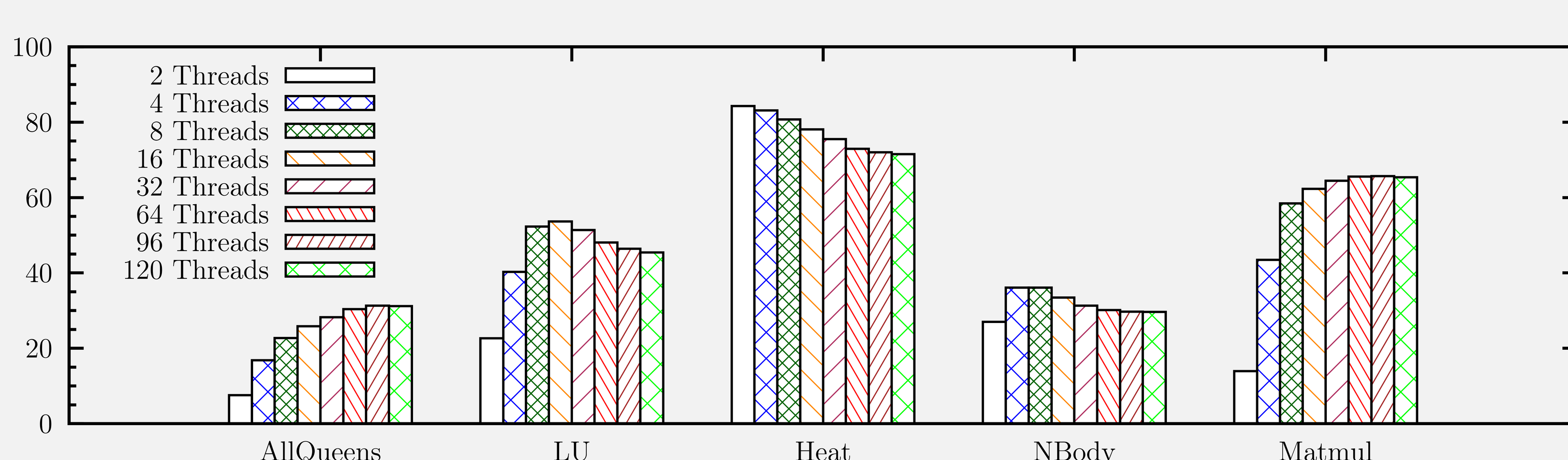


Storage Overhead

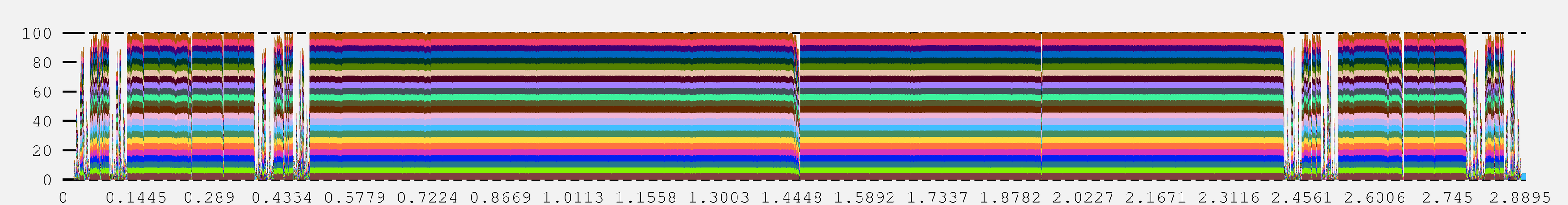




Percent Reduction DPST Traversals



Utilization Plot of Cilk LU



Presentation on Wednesday (June 19th) at 3:30pm in the Seattle 2&3 room