SAM: Optimizing Multithreaded Cores for Speculative Parallelism

MALEEN ABEYDEERA, SUVINAY SUBRAMANIAN, MARK JEFFREY,
JOEL EMER, DANIEL SANCHEZ

PACT 2017

Executive Summary

Analyzes the interplay between hardware multithreading and speculative parallelism

(eg: Thread Level Speculation and Transactional Memory)

Conventional multithreading causes performance pathologies on speculative workloads

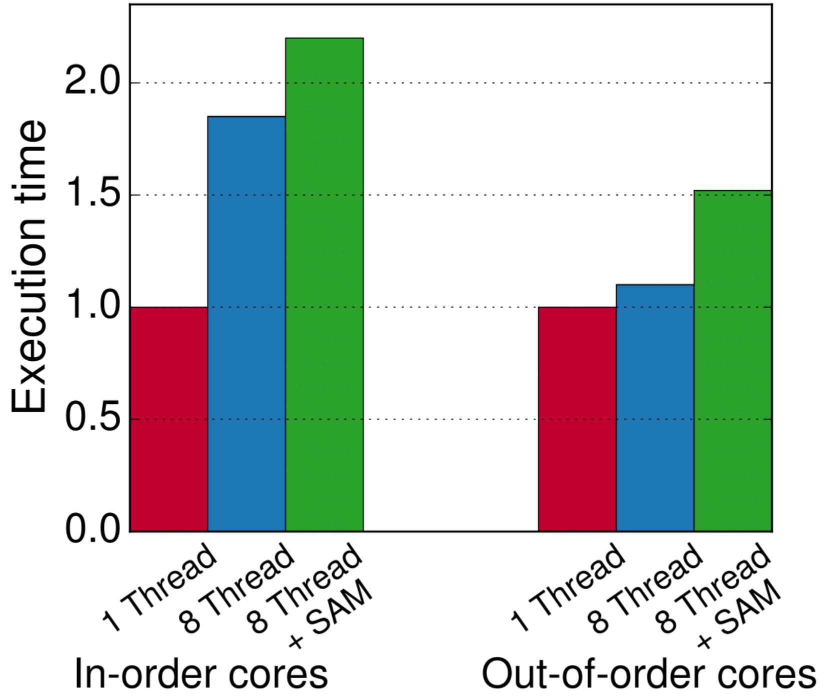
- Increase in aborted work
- Inefficient use of speculation resources

    Why? All threads are treated equally

Speculation Aware Multithreading (SAM)

- Prioritize threads running tasks more likely to commit

SAM makes multithreading more useful



Outline

Background on speculative parallelism

Pitfalls of speculative parallelism with conventional multithreading

SAM on in-order cores

SAM on out-of-order cores

Background on Speculative Parallelism

Parallelize tasks when the dependences are not known in advance
Hardware executes all tasks in parallel, aborting upon conflicts
Which task to abort? Conflict resolution policy

Speculative Parallelism

Ordered
e.g. Thread-Level Speculation (TLS)
(Program order dictates the conflict resolution order)

Unordered
e.g. Hardware Transactional Memory
(Any execution order is valid, but high-performance conflict resolution policies define an order)

Implicit order among all tasks in any speculative system

Baseline System - Swarm [Jeffrey, MICRO' 15]

Timestamped tasks

```
void desTask(Timestamp ts , GateInput* input) {
     Gate* g = input ->gate ();
     bool toggledOutput = g.simulateToggle(input);
     if ( toggledOutput ) {
         for (GateInput* i : g-> connectedInputs ()) {
             swarm::enqueue(desTask , ts+delay(g,i), i);
         }
     }
}
```

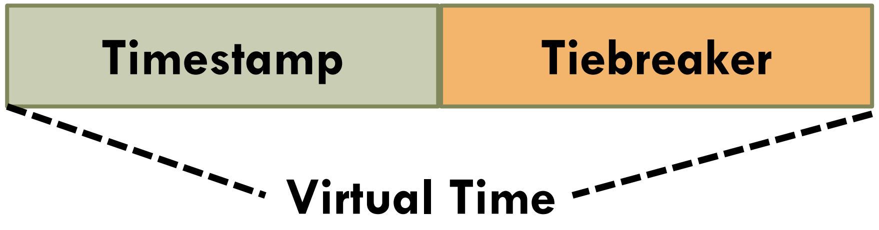
Tasks create children tasks (function ptr, timestamp, args)

Tasks appear to execute in timestamp order

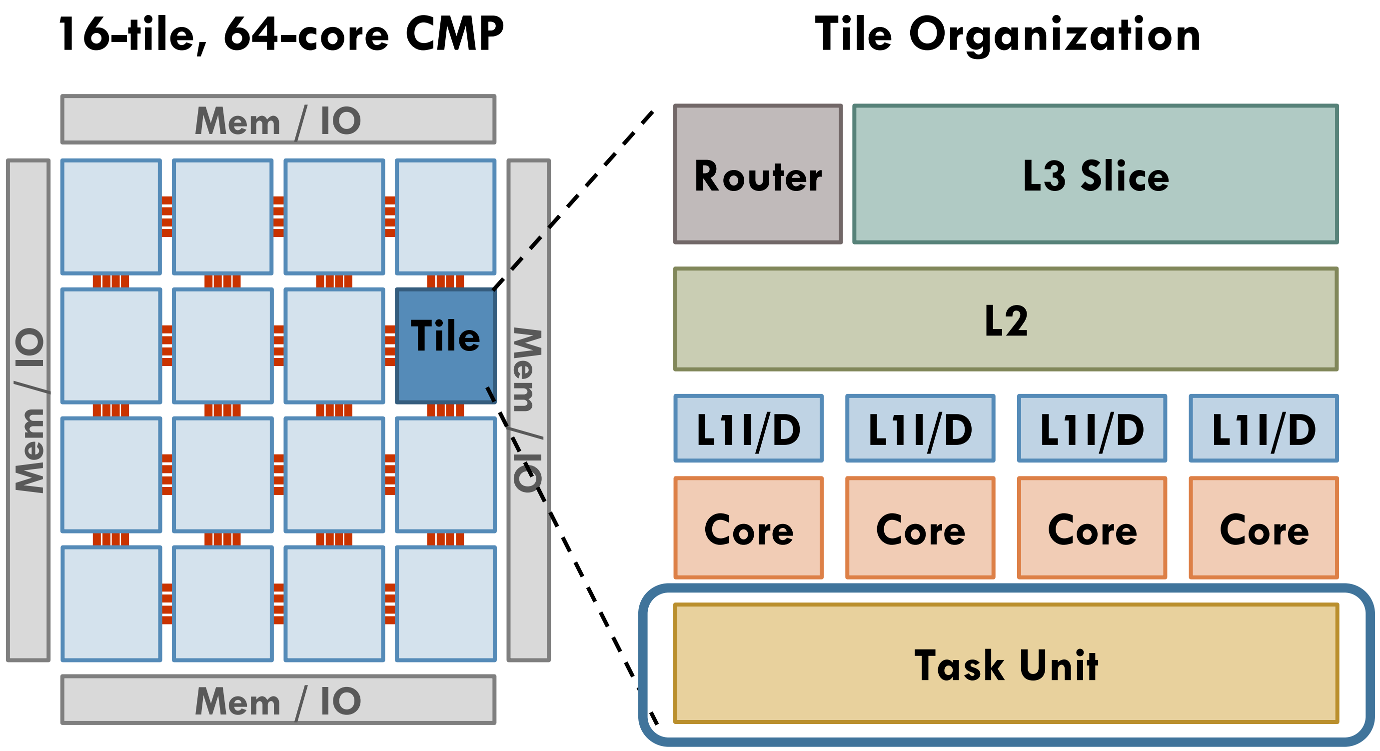
Unordered execution via equal timestamps

Swarm Microarchitecture

Equal timestamps:
global order via Virtual Time (VT)



Tasks execute out-of-order,
but commit in VT order



Commit queue: state of tasks waiting to commit

Outline

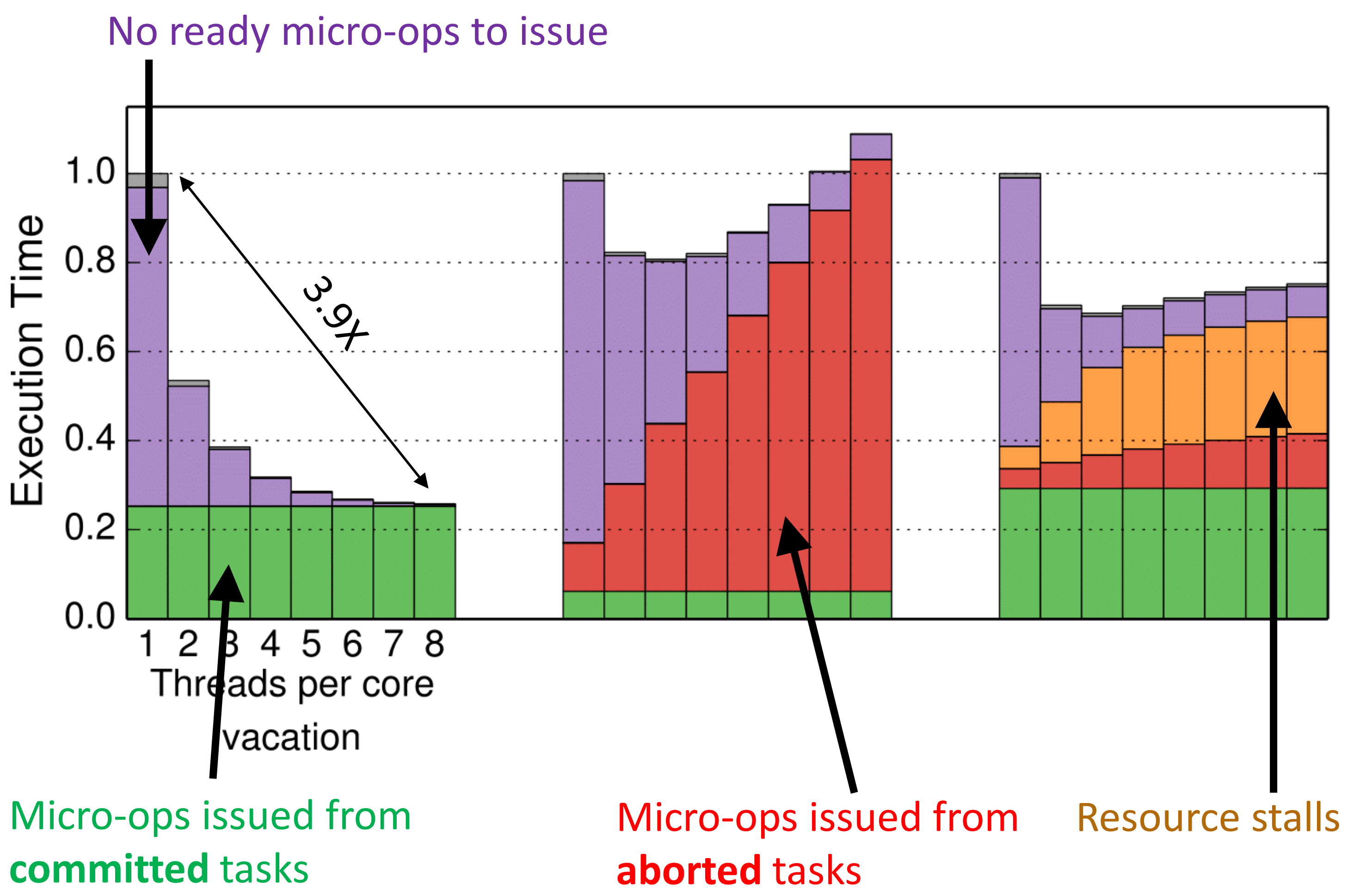
Background on speculative parallelism

Pitfalls of speculative parallelism with conventional multithreading

SAM on in-order cores

SAM on out-of-order cores

Pitfalls of Speculation-Oblivious Multithreading



System configuration:
64-core SMT system
In-order core with 2-wide issue
Speculation-oblivious round-robin order

Insights:

1. Multithreading can be highly beneficial

However, multithreading can also lead to:

2. Increased aborts
3. Inefficient use of speculation resources

Unlikely-to-commit tasks hurt the throughput of likely-to-commit ones

Speculation-Aware Multithreading

Prioritize threads according to their conflict resolution priorities

Reduce Aborts
(focus resources on tasks likely to commit)

Reduce Speculation Resource Stalls
(tasks commit early)

Outline

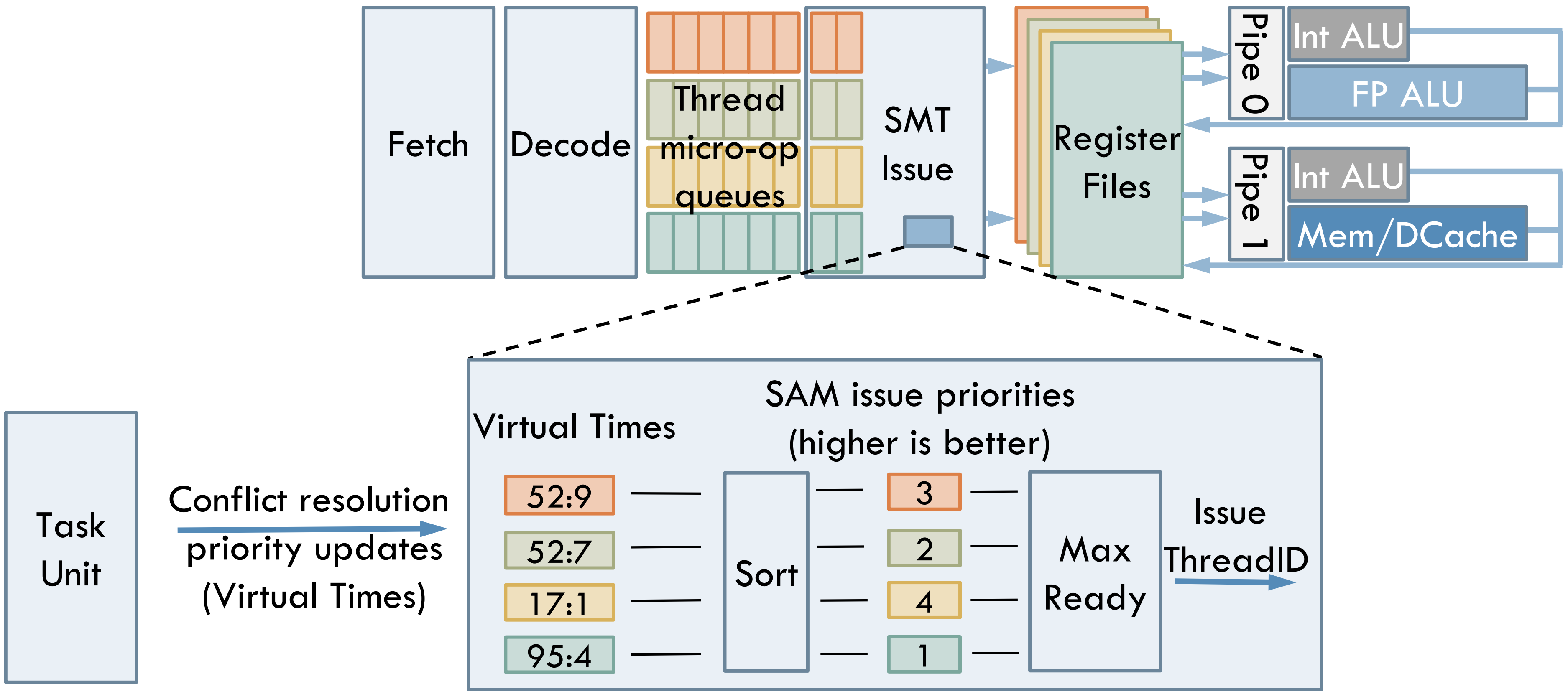
Background on speculative parallelism

Pitfalls of speculative parallelism with conventional multithreading

SAM on in-order cores

SAM on out-of-order cores

SAM on in-order cores



Experimental Methodology

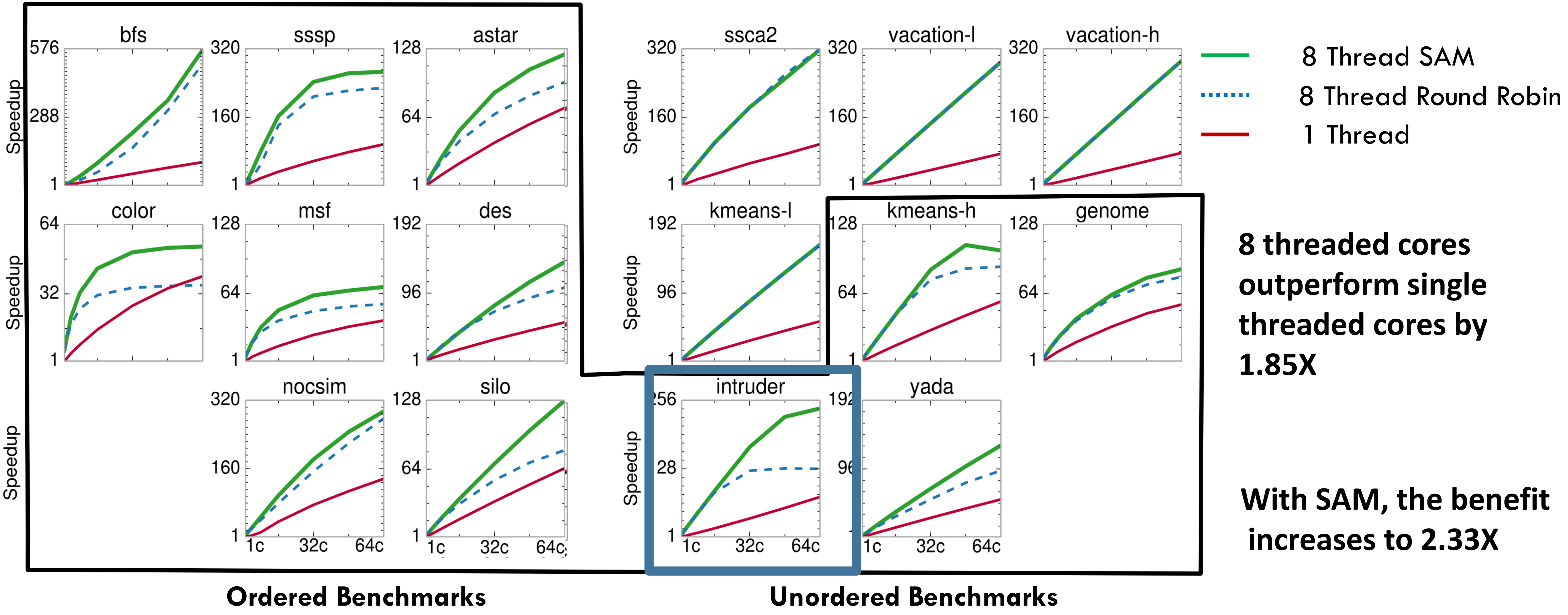
Baseline System

- Swarm + Wait-N-GoTM [Jafri et al. ASPLOS'13] conflict resolution techniques
- Cycle-accurate, event-driven, Pin-based simulator
- Model systems up to 64 cores
- Cores: 2 wide issue, up to 8 threads per core

Benchmarks

- Ordered : Swarm [Jeffrey et al. MICRO'15, MICRO'16] – 8 benchmarks
- Unordered : STAMP [Minh et al. IISWC' 08] – 8 benchmarks

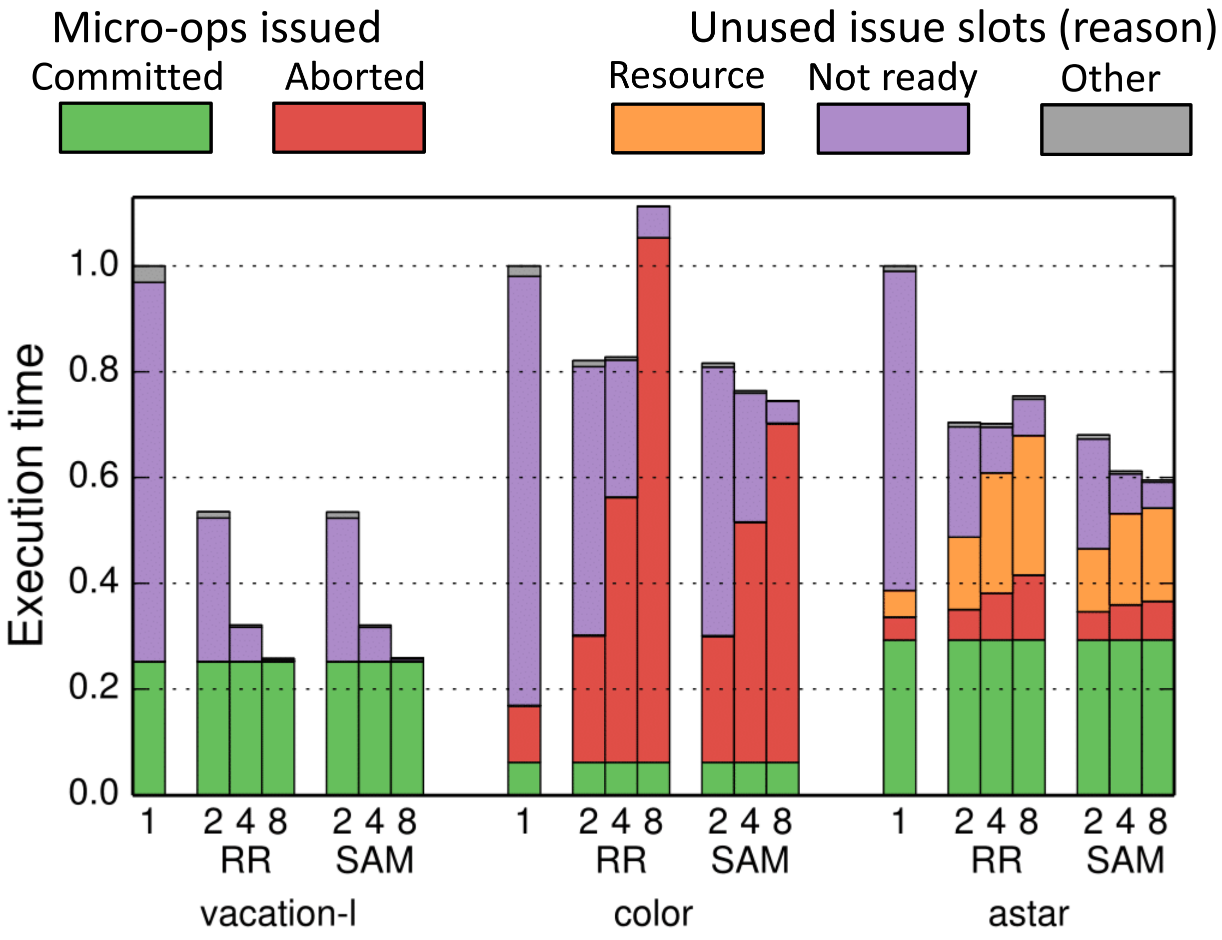
SAM makes multithreading more effective



8 threaded cores outperform single threaded cores by 1.85X

With SAM, the benefit increases to 2.33X

Why does SAM help?



SAM matches RR when there are no pathologies

SAM reduces wasted work

SAM reduces resource stalls

Outline

Background on speculative parallelism

Pitfalls of speculative parallelism with conventional multithreading

SAM on in-order cores

SAM on out-of-order cores

SAM on out-of-order cores

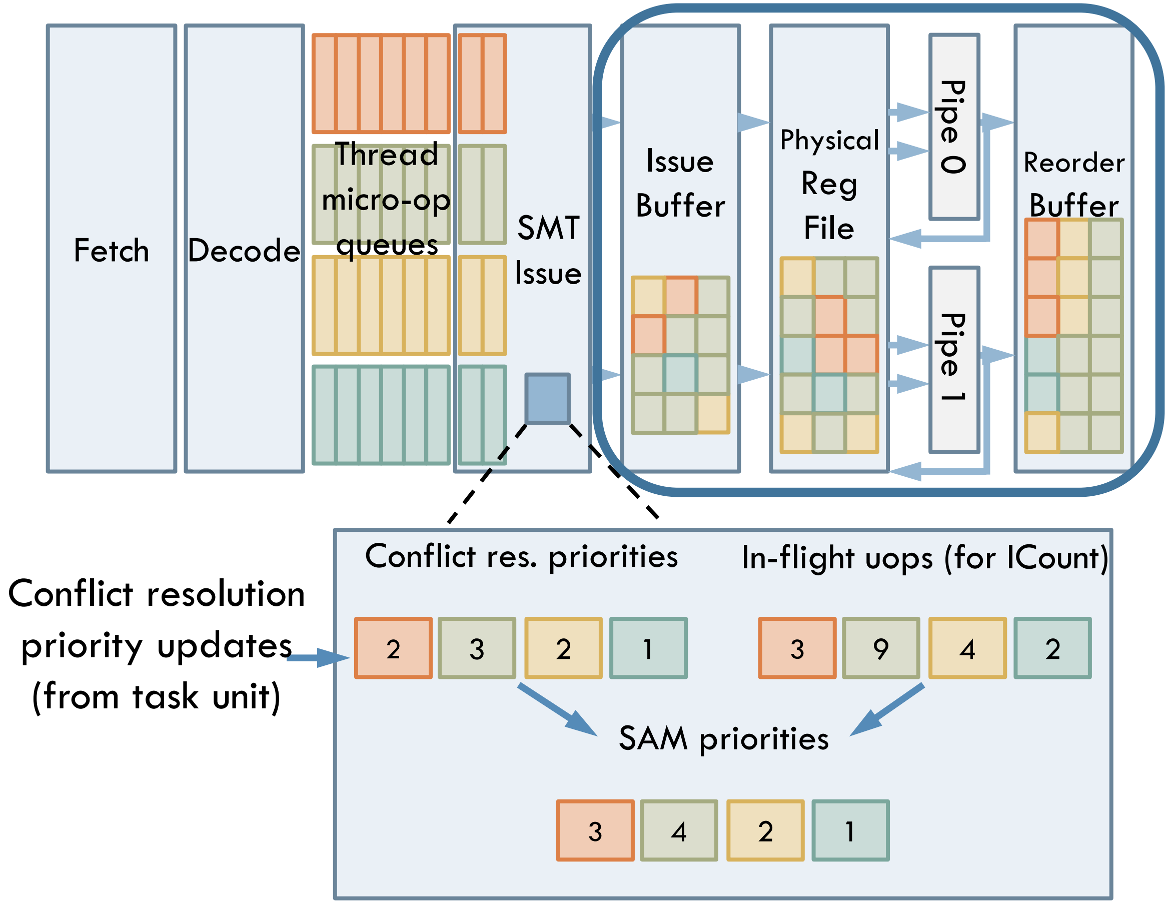
Unlike in-order cores, priorities affect pipeline efficiency

- A single thread can clog core resources
- Increased wrong path execution

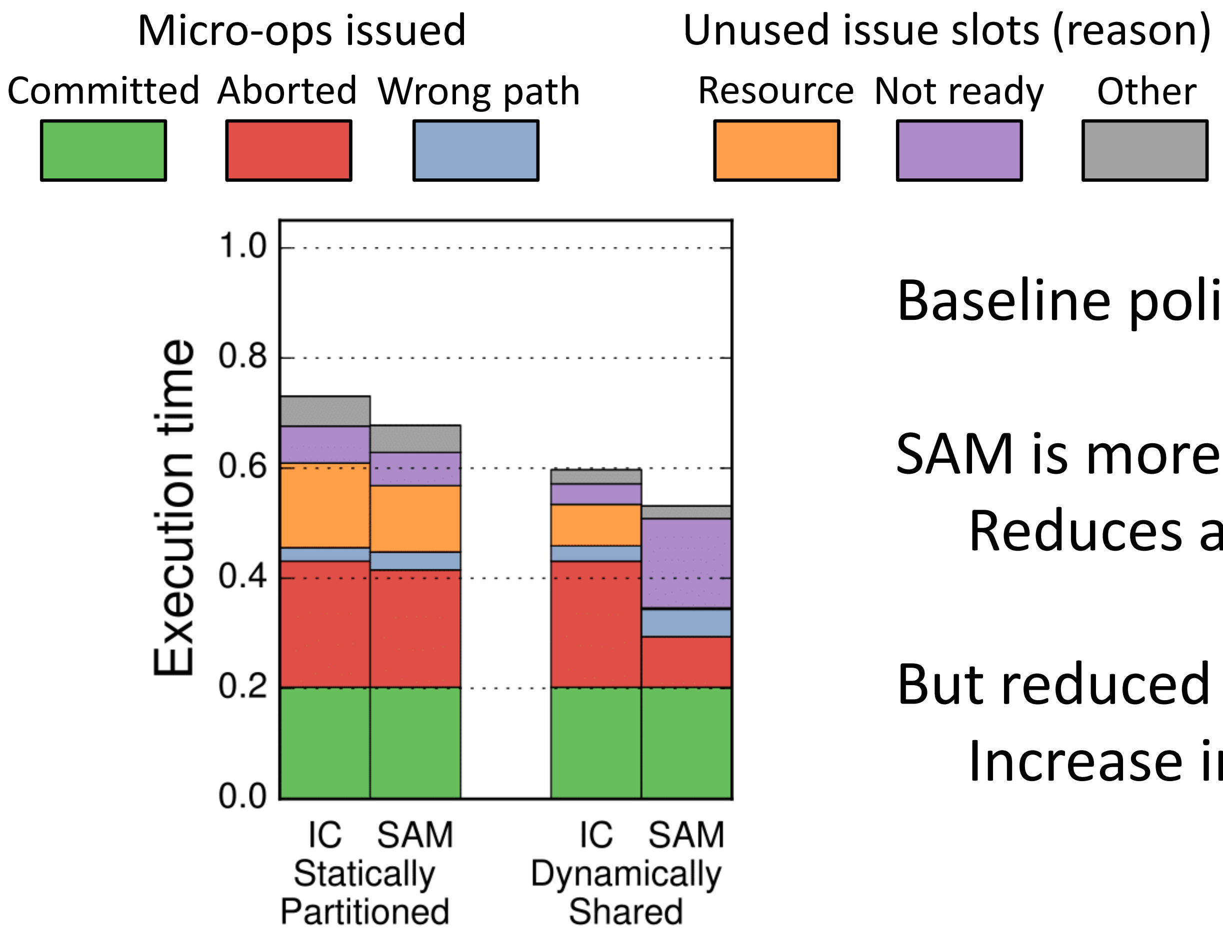
Despite these, prioritizing tasks is better

Need for aggressive prioritization affects core design

- Shared, not partitioned ROBs



SAM tradeoffs with out-of-order cores



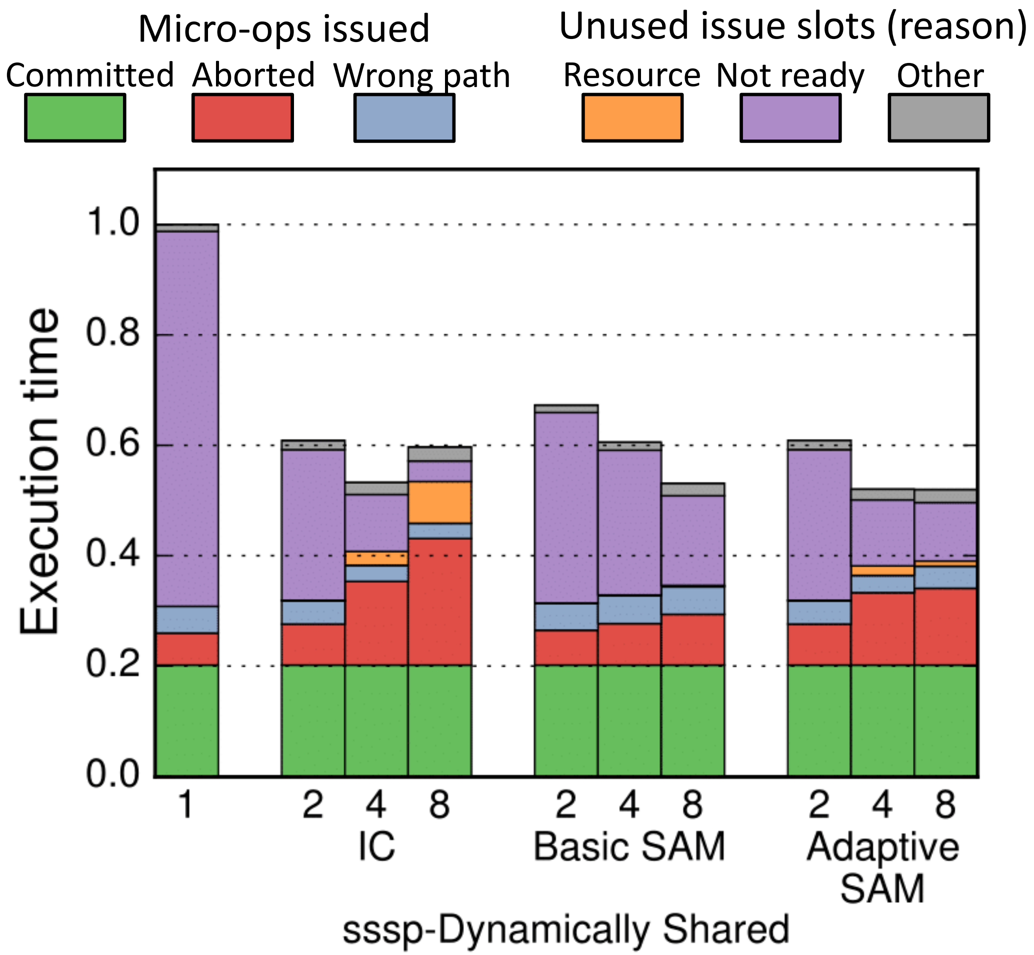
sssp – 8 threads

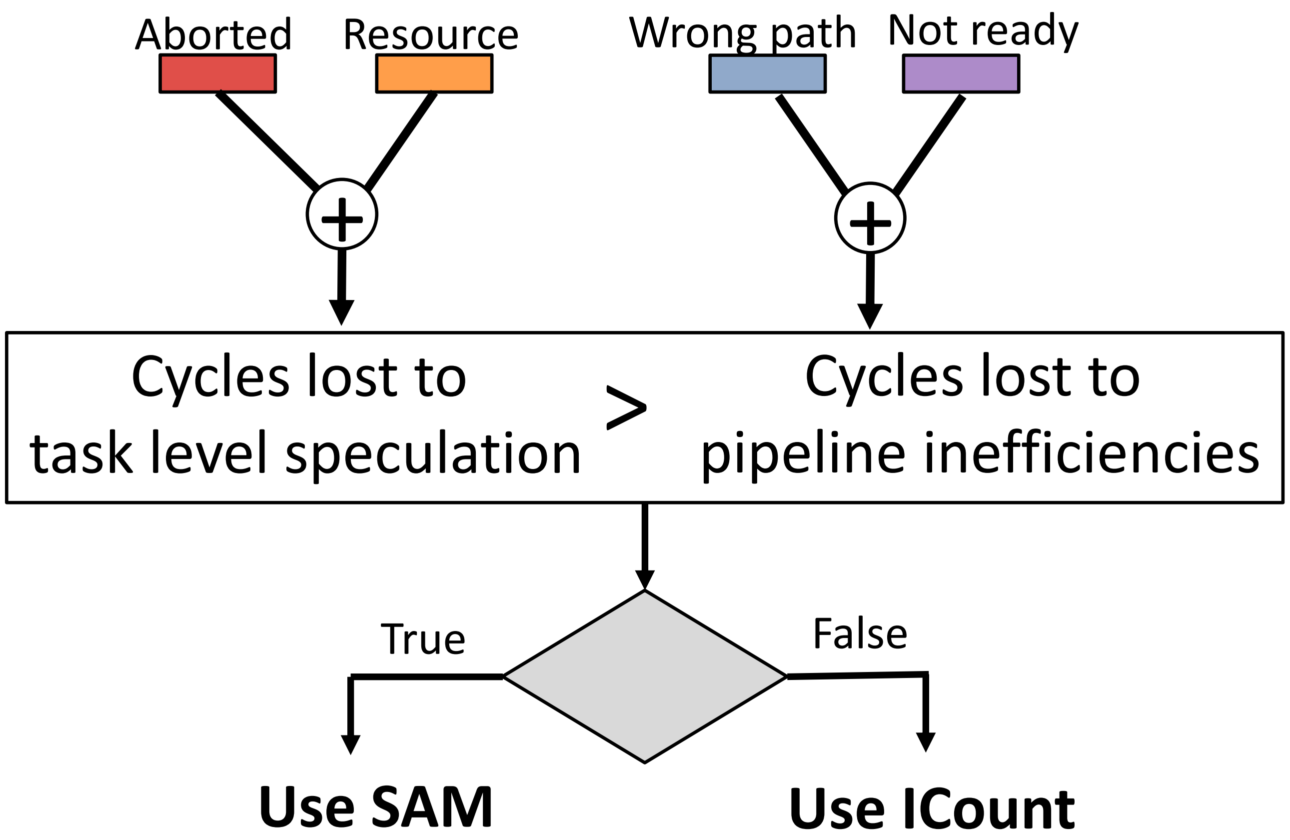
Baseline policy - ICount (IC)

SAM is more beneficial with dynamically shared ROBs
- Reduces aborts + resource stalls

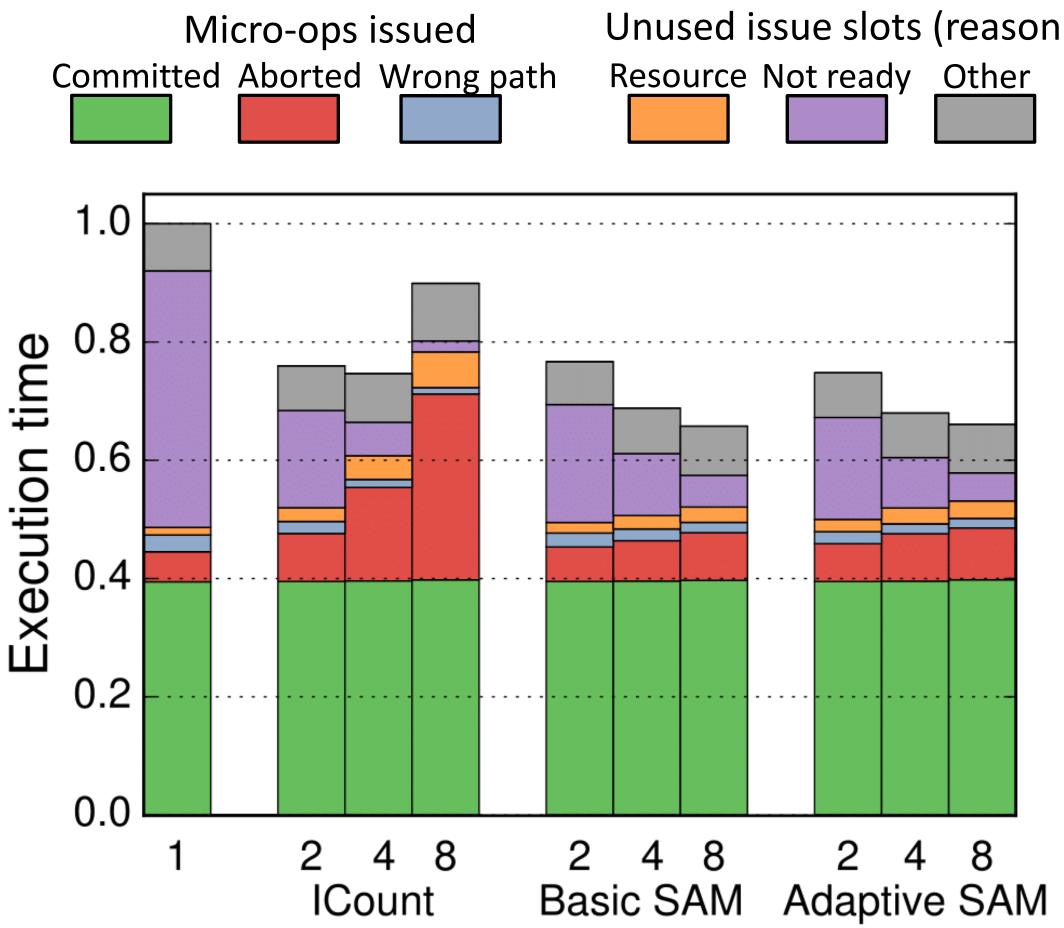
But reduced pipeline efficiency
- Increase in wrong-path issues + not-ready stalls

Adaptive SAM policy





SAM on OoO cores (all benchmarks)



Average over all benchmarks

At 8 threads / core:

- Multithreading improves performance over single threaded cores by 1.1x
- With SAM, improvement rises to 1.5x

Adaptive policy slightly increases performance at 2 and 4 threads

Conclusion

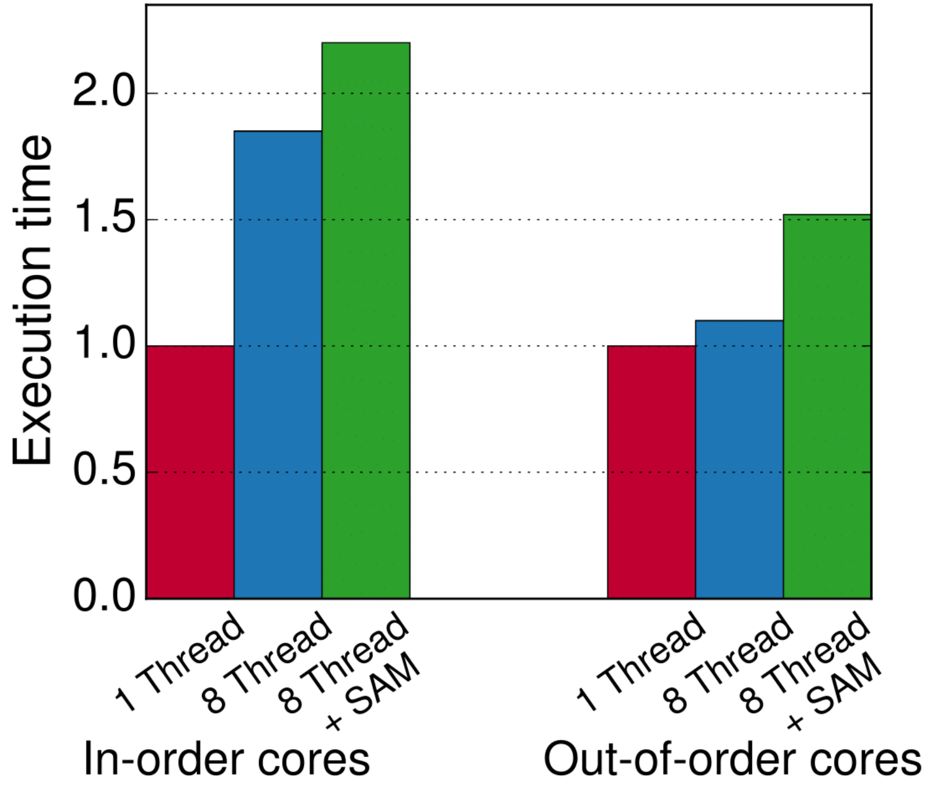
Conventional multithreading causes performance pathologies on speculative workloads

- Increase in aborted work
- Inefficient use of speculation resources

Speculation Aware Multithreading (SAM)

Prioritize threads running tasks more likely to commit

SAM makes multithreading more useful



Questions?

Conventional multithreading causes performance pathologies on speculative workloads

- Increase in aborted work
- Inefficient use of speculation resources

Speculation Aware Multithreading (SAM)

Prioritize threads running tasks more likely to commit

SAM makes multithreading more useful