SAM: Optimizing Multithreaded Cores for Speculative Parallelism

MALEEN ABYEDEERA, SUVINAY SUBRAMANIEN, 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
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
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)
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]

```c
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);
        }
    }
}
```
Baseline System - Swarm [Jeffrey, MICRO’ 15]

```cpp
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);
        }
    }
}
```

Timestamped tasks

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)

- Timestamp
- Tiebreaker

Virtual Time
Swarm Microarchitecture

Equal timestamps: global order via Virtual Time (VT)

16-tile, 64-core CMP

Tile Organization

Router  L3 Slice

L2

L1I/D  L1I/D  L1I/D  L1I/D

Core  Core  Core  Core

Task Unit
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
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

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

Micro-ops issued from committed tasks
No ready micro-ops to issue

Execution Time

Threads per core

3.9x

vacation
Pitfalls of Speculation-Oblivious Multithreading

Insights:
1. Multithreading can be highly beneficial
   However, multithreading can also lead to:
2. Increased aborts
Insights:
1. Multithreading can be highly beneficial

However, multithreading can also lead to:
2. Increased aborts
3. Inefficient use of speculation resources

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

Micro-ops issued from committed tasks
Micro-ops issued from aborted tasks
Resource stalls

No ready micro-ops to issue
Insights:
1. Multithreading can be highly beneficial
2. Increased aborts
3. Inefficient use of speculation resources

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

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

SAM : OPTIMIZING MULTITHREADED CORES FOR SPECULATIVE PARALLELISM
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
SAM on in-order cores

SAM : OPTIMIZING MULTITHREADED CORES FOR SPECULATIVE PARALLELISM
SAM on in-order cores

Task Unit

Conflicts resolution
priority updates (Virtual Times)
SAM on in-order cores

SAM issue priorities (higher is better)

Virtual Times

- 52:9
- 52:7
- 17:1
- 95:4

SAM : OPTIMIZING MULTITHREADED CORES FOR SPECULATIVE PARALLELISM
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

Ordered Benchmarks

Unordered Benchmarks
SAM makes multithreading more effective.

Ordered Benchmarks
- bfs
- sssp
- astar
- color
- msf
- des
- nocsim
- silo

Unordered Benchmarks
- ssca2
- vacation-l
- vacation-h
- kmeans-l
- kmeans-h
- genome
- intruder
- yada

8 Thread Round Robin
1 Thread
SAM makes multithreading more effective

Ordered Benchmarks

Unordered Benchmarks

8 Thread SAM
8 Thread Round Robin
1 Thread
SAM makes multithreading more effective

Ordered Benchmarks

Unordered Benchmarks

8 threaded cores outperform single threaded cores by 1.85X

With SAM, the benefit increases to 2.33X
SAM makes multithreading more effective

8 threaded cores outperform single threaded cores by 1.85X

With SAM, the benefit increases to 2.33X
SAM makes multithreading more effective

Ordered Benchmarks

Unordered Benchmarks

With SAM, the benefit increases to 2.33X

8 threaded cores outperform single threaded cores by 1.85X
Why does SAM help?

SAM matches RR when there are no pathologies

SAM : OPTIMIZING MULTITHREADED CORES FOR SPECULATIVE PARALLELISM
Why does SAM help?

SAM matches RR when there are no pathologies

SAM reduces wasted work
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

<table>
<thead>
<tr>
<th>Micro-ops issued</th>
<th>Unused issue slots (reason)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Committed</td>
<td>Resource</td>
</tr>
<tr>
<td>Aborted</td>
<td>Not ready</td>
</tr>
<tr>
<td>Wrong path</td>
<td>Other</td>
</tr>
</tbody>
</table>

Baseline policy - ICount (IC)

### Execution time

<table>
<thead>
<tr>
<th></th>
<th>IC Saturated</th>
<th>SAM Saturated</th>
<th>IC Partitioned</th>
<th>SAM Partitioned</th>
<th>IC Shared</th>
<th>SAM Shared</th>
</tr>
</thead>
<tbody>
<tr>
<td>sssp</td>
<td>0.5</td>
<td>0.5</td>
<td>0.5</td>
<td>0.5</td>
<td>0.5</td>
<td>0.5</td>
</tr>
</tbody>
</table>

sssp – 8 threads

SAM : OPTIMIZING MULTITHREADED CORES FOR SPECULATIVE PARALLELISM
SAM tradeoffs with out-of-order cores

Baseline policy - ICount (IC)
SAM is more beneficial with dynamically shared ROBs
Reduces aborts + resource stalls

SAM : OPTIMIZING MULTITHREADED CORES FOR SPECULATIVE PARALLELISM
SAM tradeoffs with out-of-order cores

Baseline policy - ICount (IC)

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

But reduced pipeline efficiency
SAM tradeoffs with out-of-order cores

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

Micro-ops issued
- Committed
- Aborted
- Wrong path

Unused issue slots (reason)
- Resource
- Not ready
- Other

Evolution

Execution time

sssp-Dynamically Shared

SAM : OPTIMIZING MULTITHREADED CORES FOR SPECULATIVE PARALLELISM
Adaptive SAM policy

Hardware counters to track cycles
Adaptive SAM policy

Hardware counters to track cycles

- Aborted
- Resource
- Wrong path
- Not ready

Bar graph showing execution time for various metrics:
- Micro-ops issued
- Unused issue slots (reason)
- Committed
- Aborted
- Resource Not ready
- Other

Columns:
- sssp-Dynamically Shared
- IC
- Basic SAM

Values range from 0.0 to 1.0.
Adaptive SAM policy

Hardware counters to track cycles

![Chart showing execution time and reasons for aborts, resource unavailability, and other issues]
Adaptive SAM policy

Hardware counters to track cycles

Cycles lost to task level speculation
Adaptive SAM policy

Hardware counters to track cycles

Cycles lost to task level speculation > Cycles lost to pipeline inefficiencies

Use SAM

Use ICount

SAM : OPTIMIZING MULTITHREADED CORES FOR SPECULATIVE PARALLELISM
SAM on OoO cores (all benchmarks)

At 8 threads / core:

- Multithreading improves performance over single threaded cores by 1.1x
SAM on OoO cores (all benchmarks)

At 8 threads / core:
- Multithreading improves performance over single threaded cores by 1.1x
- With SAM, improvement rises to 1.5x
SAM on OoO cores (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
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