Accelerating Critical Section Execution with Asymmetric Multi-Core Architectures

M. Aater Suleman
University of Texas at Austin
suleman@hps.utexas.edu

Onur Mutlu
Carnegie Mellon University
onur@cmu.edu

Moinuddin K. Qureshi
IBM Research
mkquresh@us.ibm.com

Yale N. Patt
University of Texas at Austin
patt@ece.utexas.edu

ASPLOS 2009

Presented by Lara Lazier

Outline

- Introduction and Background
- Proposed Solution - ACS
- Methodology and Results
- Strengths
- Weaknesses
- Takeaways
- Questions and Discussion

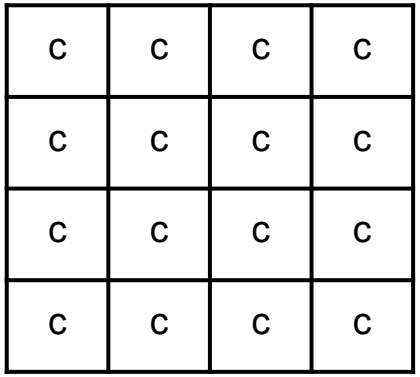
Executive Summary

- **PROBLEM: Critical sections** limit both performance and scalability
- **Accelerating Critical Sections (ACS):**
  - improves performance by moving the computation of the Critical Section to a larger and faster core.
  - First approach to accelerate critical sections in Hardware.
- **RESULTS:**
  - ACS improves performance on average by **34%** compared to a Symmetric CMP and by **24%** compared to a Asymmetric CMP
  - ACS improves scalability of 7 out of 12 Workloads.

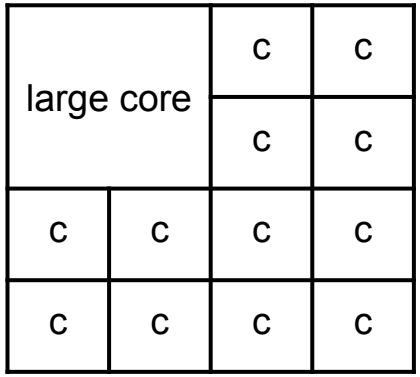
Introduction and Background

Background

- Large single core processors are complex and have high power consumption
- Chip-multiprocessors (CMP) are less complex and have less power consumption



SymmetricChipMultiProcessor



AsymmetricChipMultiProcessor

- To be able to extract high performance, programs must be split into threads

Background - Threads and Critical Sections

- Threads operate on different portions of the same problem
- Threads are not allowed to update shared data at the same time → **MUTUAL EXCLUSION**
- A **critical section** is a portion of a program that only one thread can execute at a given time

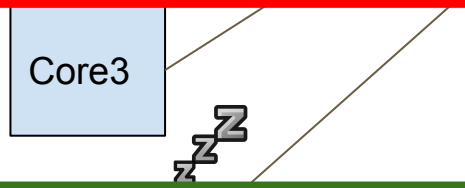
➡ Limits both performance and scalability

Amdahl's Law

$$Speedup = \frac{1}{(1 - P_{parallel}) + \frac{P_{parallel}}{N_{cores}}}$$

Problem overview and Goal

Contention increases when number of cores increases



The **goal** is to accelerate the execution of critical sections

Key Insight

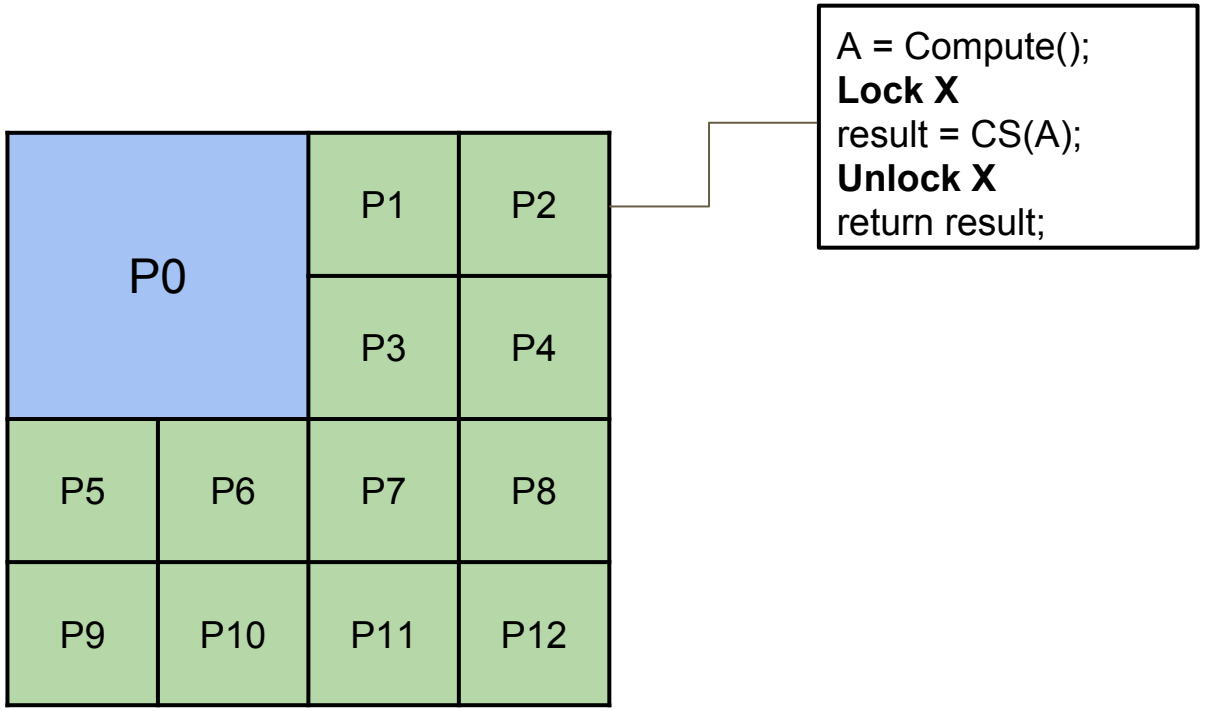
- Accelerating critical sections can provide significant performance improvement

- Asymmetric CMP can accelerate serial part using the large core

- Moving the computation of critical sections to the larger Core(s) could improve the performance

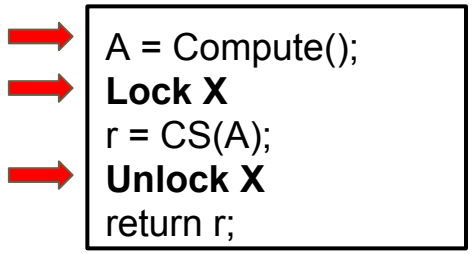
ACS
Accelerating Critical Sections

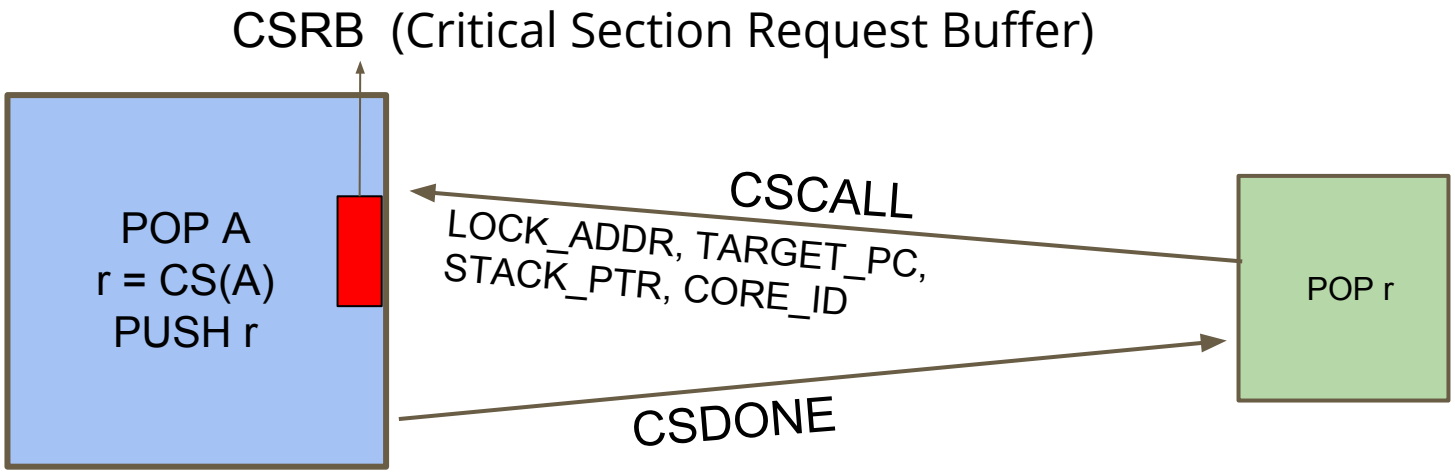
Overview



- Homogeneous ISA
- Asymmetric CMP with cache coherence

Implementation





- Lock and Shared Data do not need to be moved
- ACS reduces the number of L2 caches misses inside the critical sections by 20%

Implementation

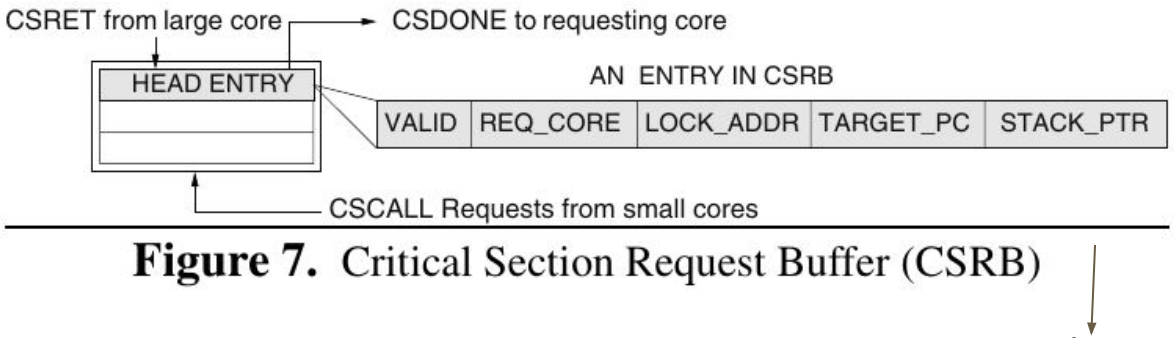
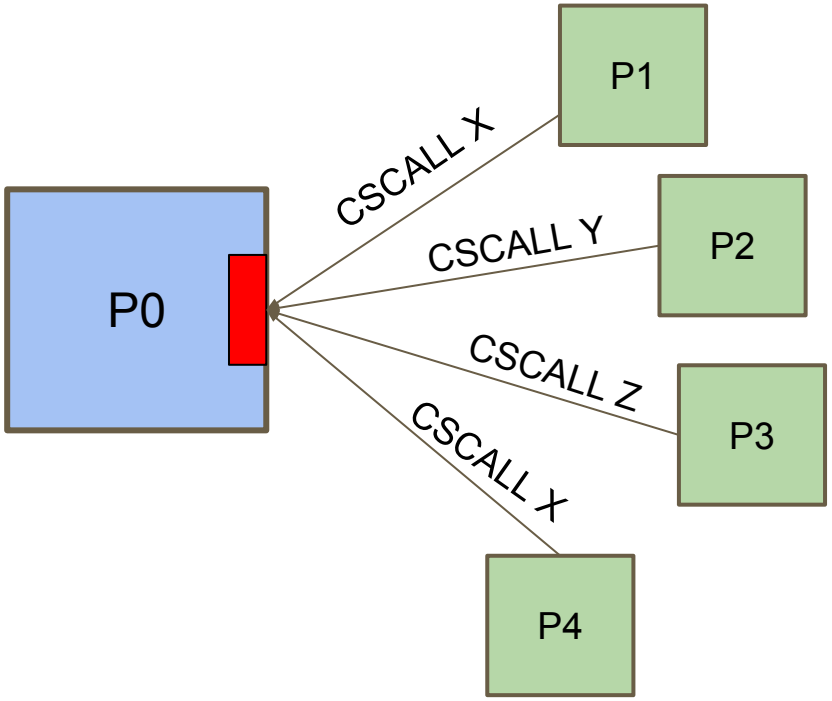
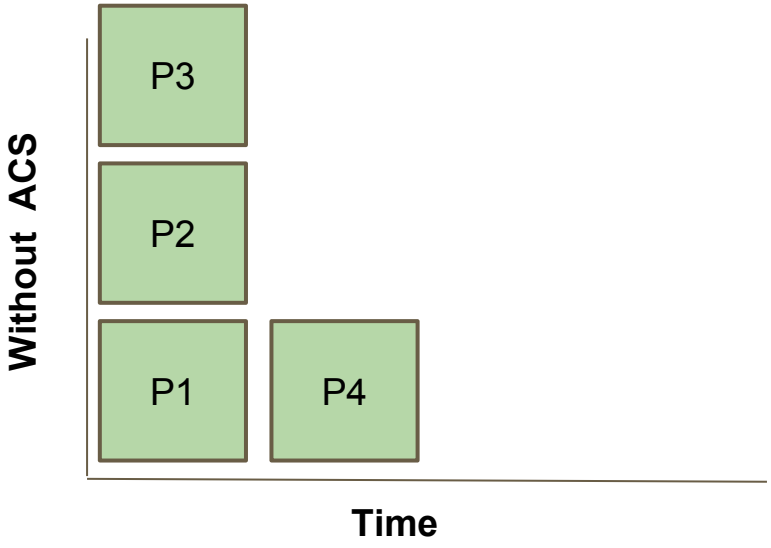


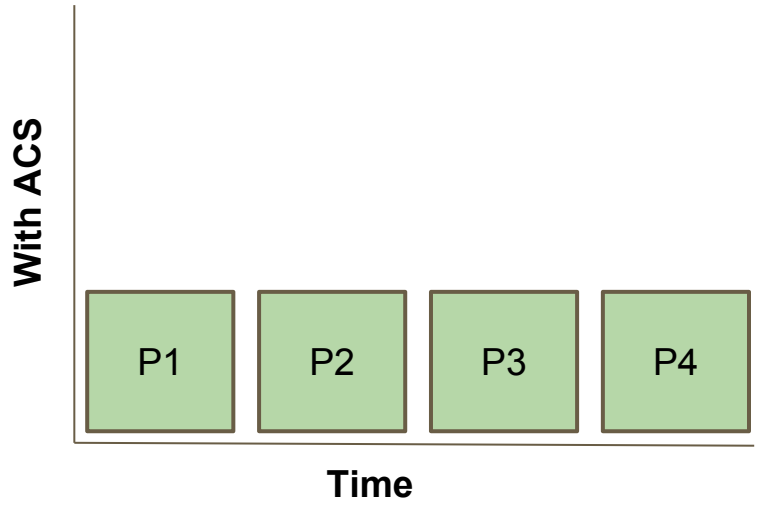
Figure 7. Critical Section Request Buffer (CSRB)

- ISA Support
  - CSCALL and CSRET
- Compiler Support:
  - insert CSCALL and CSRET and removes any register dependencies
    → *function outlining*
- Modification to small Core
  - support for executing instructions remotely
- Modification to larger Core
  - **CSRB (Critical Section Request Buffer)**
- Interconnect Extension
  - CSCALL and CSDONE
- OS support
  - allocates the large core to a single application
  - sets the same program context for all cores

False Serialization







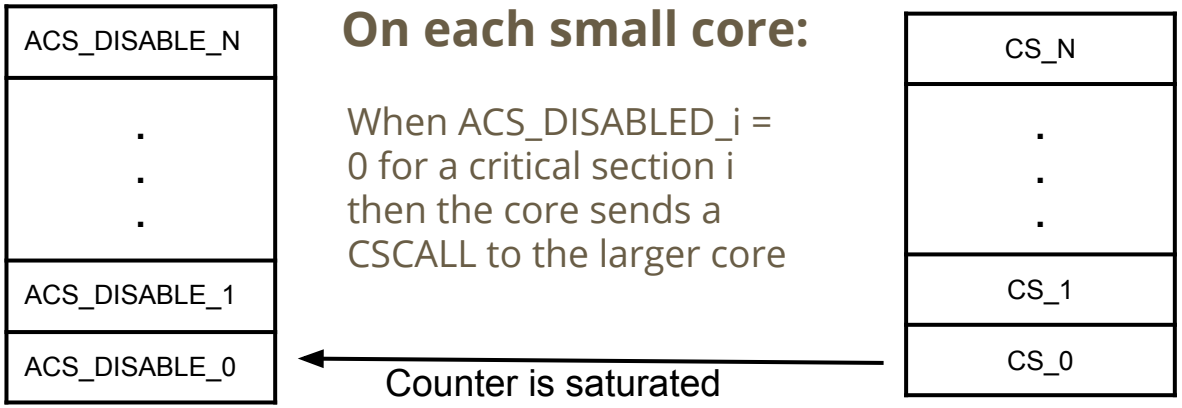
SEL

HOW TO SOLVE FALSE SERIALIZATION?

SEL (SELective Acceleration of Critical Sections)

Storage Overhead of 36 bytes (16 counters of 6-bits and 16 ACS_DISABLE for each of the 12 small cores)

To implement SEL we need:



On each small core:

When ACS_DISABLED_i = 0 for a critical section i then the core sends a CSCALL to the larger core

On the large core:

Saturating counter for each critical section:

- if there are at least 2 CS in the CSRB + #CS
- if there is 1 CS in the CSRB -1

ACS_DISABLE bits a reset and the values of the saturating counter are halved every 10 million cycles

Performance Trade-offs

- **Faster critical sections vs. fewer threads**
  - Reduced parallel throughput
  + When the number of cores increases, loss of throughput decreases and increased contention benefits more from ACS

- **CSCALL/CSDONE signals vs. lock acquire/release**
  - The communication over the on-chip interconnect is an overhead
  + ACS keeps the lock at the large core and reduces cache misses

- **Cache misses due to private data vs. cache misses due to shared data**
  - worse private data locality
  + ACS eliminates the transfer of shared data by keeping it at the large core

Results

Methodology

Small core	2-wide In-order, 2GHz, 5-stage. L1: 32KB write-through. L2: 256KB write-back, 8-way, 6-cycle access
Large core	4-wide Out-of-order, 2GHz, 2-way SMT, 128-entry ROB, 12-stage, L1: 32KB write-through. L2: 1-MB write-back, 16-way, 8-cycle
Interconnect	64-bit wide bi-directional ring, all queuing delays modeled, ring hop latency of 2 cycles (latency between one cache to the next)
Coherence	MESI, On-chip distributed directory similar to SGI Origin [26], cache-to-cache transfers. # of banks = # of cores, 8K entries/bank
L3 Cache	8MB, shared, write-back, 20-cycle, 16-way
Memory	32 banks, bank conflicts and queuing delays modeled. Row buffer hit: 25ns, Row buffer miss: 50ns, Row buffer conflict: 75ns
Memory bus	4:1 cpu/bus ratio, 64-bit wide, split-transaction, pipelined bus, 40-cycle latency
Area-equivalent CMPs. Area = N small cores. N varies from 1 to 32	
SCMP	N small cores, One small core runs serial part, all N cores run parallel part, conventional locking (Max. concurrent threads = N)
ACMP	1 large core and N-4 small cores; large core runs serial part, 2-way SMT on large core and small cores run parallel part, conventional locking (Maximum number of concurrent threads = N-2)
ACS	1 large core and N-4 small cores; (N-4)-entry CSRB on the large core, large core runs the serial part, small cores run the parallel part, 2-way SMT on large core runs critical sections using ACS (Max. concurrent threads = N-4)

- Simulating CMPs using a cycle-accurate x86 simulator.
- The large core occupies the same area as **4** smaller cores and they are modeled after the **Intel Pentium-M.**
- The smaller cores are modeled after the intel **Pentium Processor.**

Workloads

The workloads are evaluated on:

- Symmetric CMP
- Asymmetric CMP with one large core with 2-way SMT
- Asymmetric CMP with ACS.

Locks	Workload	Description	Source	Input set
Coarse	ep	Random number generator	[7]	262144 nums.
	is	Integer sort	[7]	n = 64K
	pagemine	Data mining kernel	[31]	10Kpages
	puzzle	15-Puzzle game	[50]	3x3
	qsort	Quicksort	[11]	20K elem.
	sqlite	sqlite3 [3] database engine	[4]	OLTP-simple
	tsp	Traveling salesman prob.	[23]	11 cities
Fine	iplookup	IP packet routing	[49]	2.5K queries
	oltp-1	MySQL server [1]	[4]	OLTP-simple
	oltp-2	MySQL server [1]	[4]	OLTP-complex
	specjbb	JAVA business benchmark	[40]	5 seconds
	webcache	Cooperative web cache	[45]	100K queries

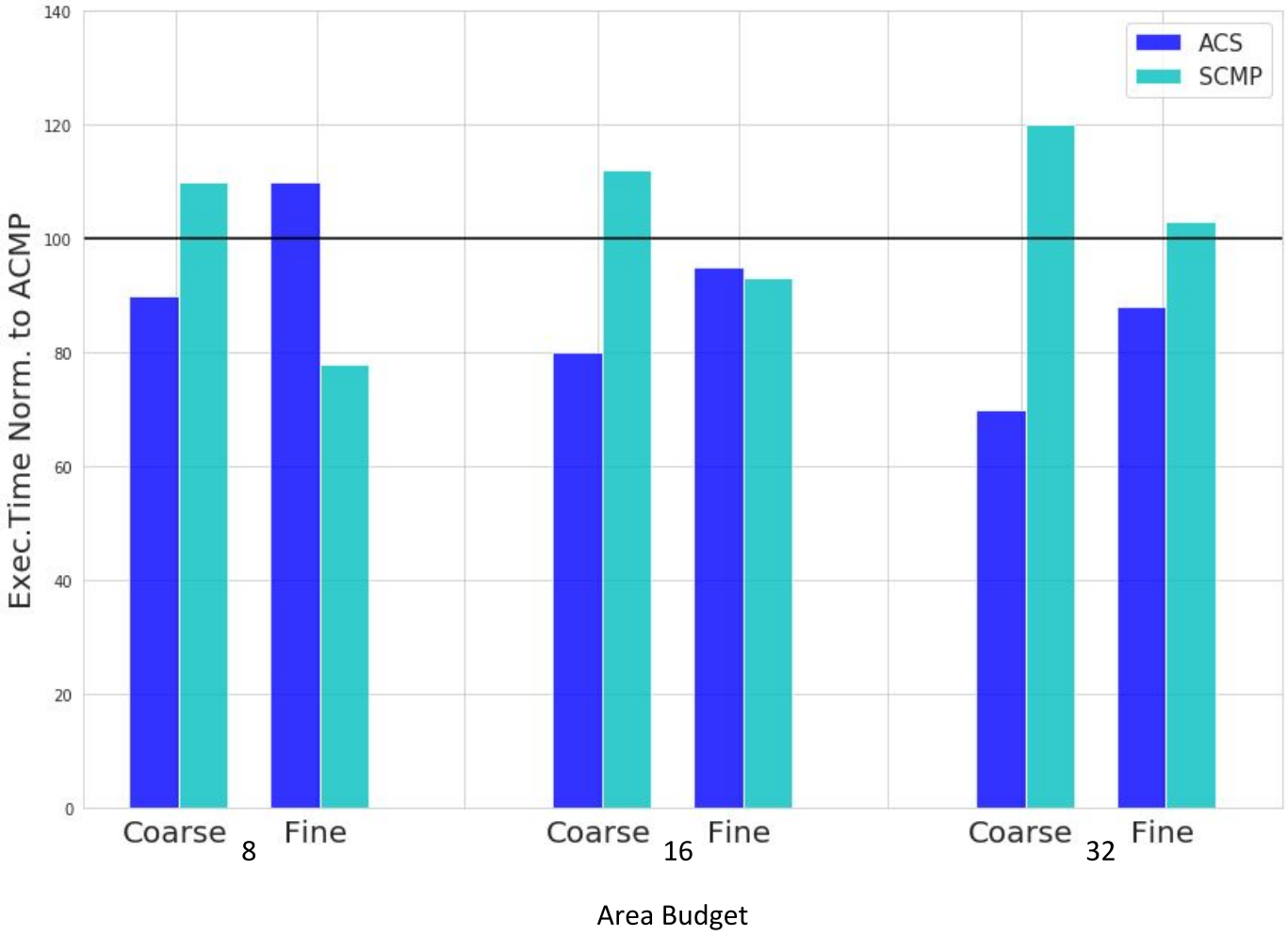
Results

- ACS reduces the average execution time by **34%** compared to an equal-area baseline with 32-Core SCMP.

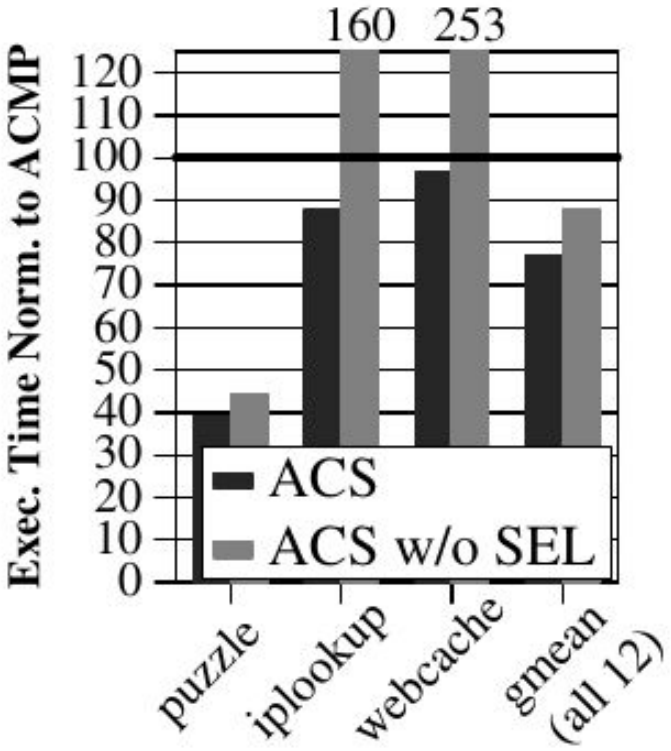
- ACS reduced the average execution time time by **23%** compared to an equal-area ACMP.

- ACS improves scalability of 7 workloads.

Coarse Grained vs. Fine Grained Results



SEL



On average, across all 12 workloads, ACS with SEL outperforms ACS without SEL by 15%.

Summary

Summary

- **PROBLEM: Critical sections** limit both performance and scalability
- **Accelerating Critical Sections (ACS):**
  - improves performance by moving the computation of the Critical Section to a larger and faster core.
  - First approach to accelerate critical sections in Hardware.
- **RESULTS:**
  - ACS improves performance on average by **34%** compared to a Symmetric CMP and by **24%** compared to a Asymmetric CMP
  - ACS improves scalability of 7 out of 12 Workloads.

Strengths

Strengths

- Novel, intuitive idea. First approach to accelerate critical sections directly in hardware
- The results are going to become more and more interesting
- Low hardware overhead
- The paper analyzes very well all possible trade offs
- The figures complement very well the explanations

Weaknesses

Weaknesses

- ACS only accelerate critical sections
- SEL might overcomplicating the problem. There might be some easier ideas that don’t need additional hardware
- The area budget to outperform both SCMP and ACMP make it less attractive for an everyday use
- Costly to implement : ISA, Compiler, interconnect…

Thoughts and Ideas

Thoughts and Ideas

- How would it work with more than one large core?

  Jose A. Joao, M. Aater Suleman, Onur Mutlu, and Yale N. Patt, **"Bottleneck Identification and Scheduling in Multithreaded Applications"** ***(ASPLOS '12)***

- How could we also accelerate other bottlenecks as barriers and slow pipeline stages?

  Jose A. Joao, M. Aater Suleman, Onur Mutlu, and Yale N. Patt, **"Bottleneck Identification and Scheduling in Multithreaded Applications"** ***(ASPLOS '12)***

- Improving locality in staged execution

  M. Aater Suleman, Onur Mutlu, Jose A. Joao, Khubaib, and Yale N. Patt, **"Data Marshaling for Multi-core Architectures"**, ***(ISCA '10)***

- Accelerating more (BIS with Lagging Threads):

  Jose A. Joao, M. Aater Suleman, Onur Mutlu, and Yale N. Patt, **"Utility-Based Acceleration of Multithreaded Applications on Asymmetric CMPs"** ***(ISCA '13)***

Takeaways

Key Takeaways

- The idea of moving specialized sections of computation to a different “core” (= accelerator, GPU...) has a lot of potential
- ACS is a novel way to accelerate critical section in hardware
- The key idea is very intuitive and easy to understand
- Software is not the only solution

Questions ?

Discussion Starters

1. Do you think the trend of specializing hardware is going to increase even more in future? What other things could be done?

2. Do you think this could create new security threats? Can you imagine a way modularity could increase security?

3. Could ACS be combined with MorphCore?

Accelerating Critical Section Execution with Asymmetric Multi-Core Architectures

M. Aater Suleman
University of Texas at Austin
suleman@hps.utexas.edu

Onur Mutlu
Carnegie Mellon University
onur@cmu.edu

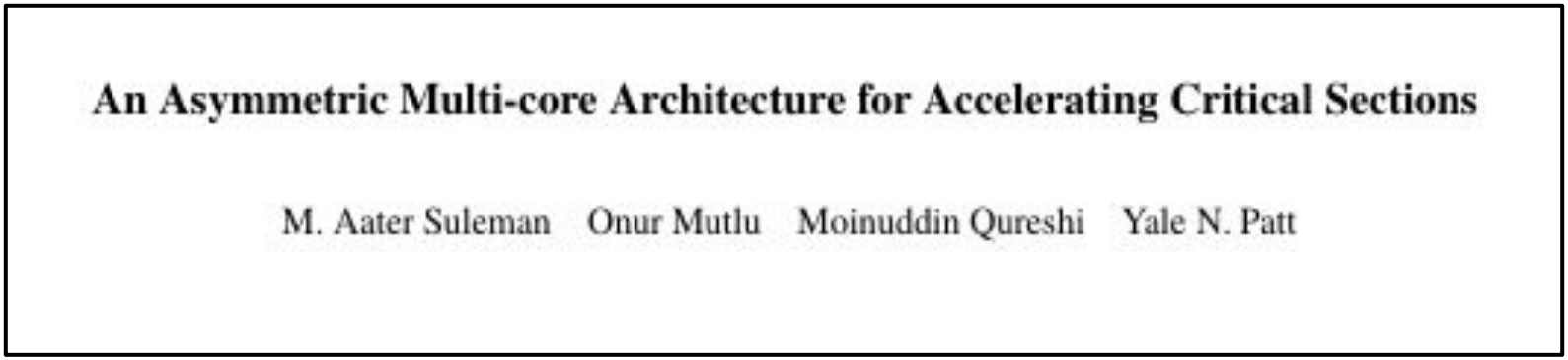
Moinuddin K. Qureshi
IBM Research
mkquresh@us.ibm.com

Yale N. Patt
University of Texas at Austin
patt@ece.utexas.edu

ASPLOS 2009

Presented by Lara Lazier

What more results?

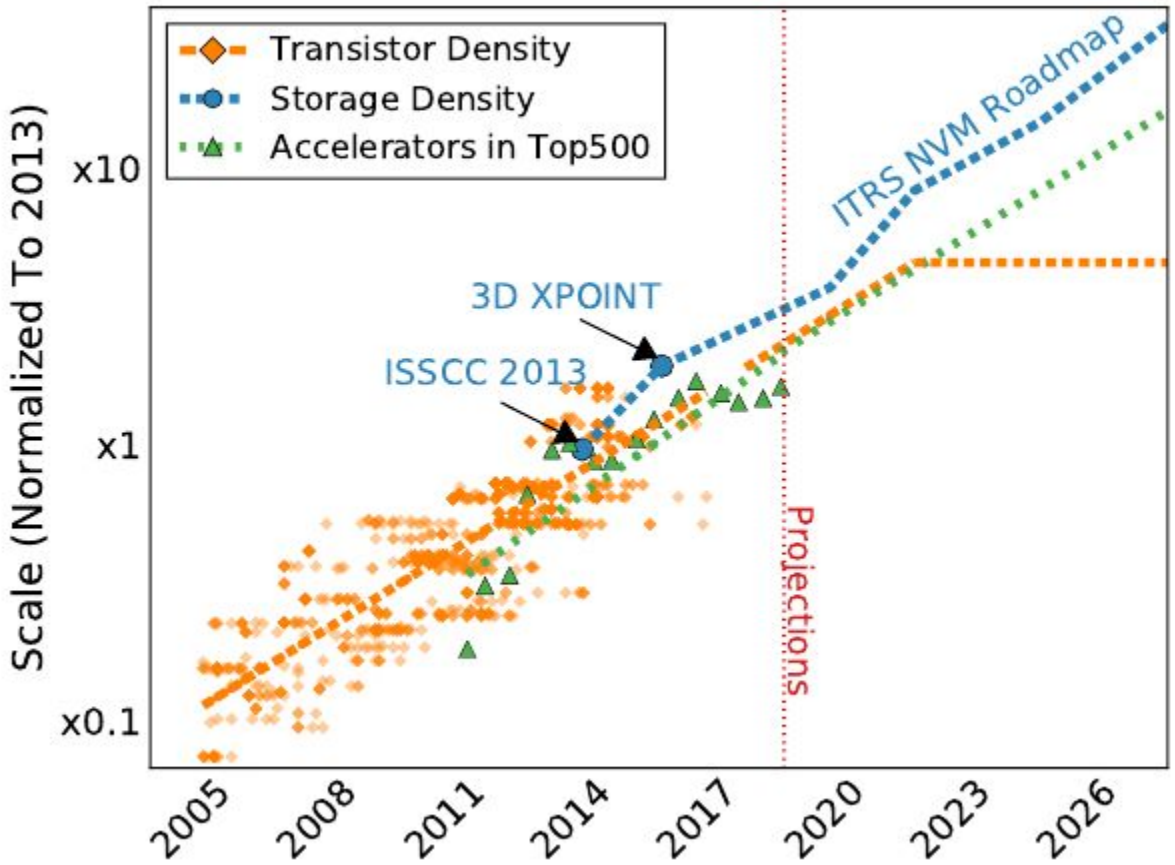


"An Asymmetric Multi-core Architecture for Accelerating Critical Sections"

HPS Technical Report, TR-HPS-2008-003, September 2008.

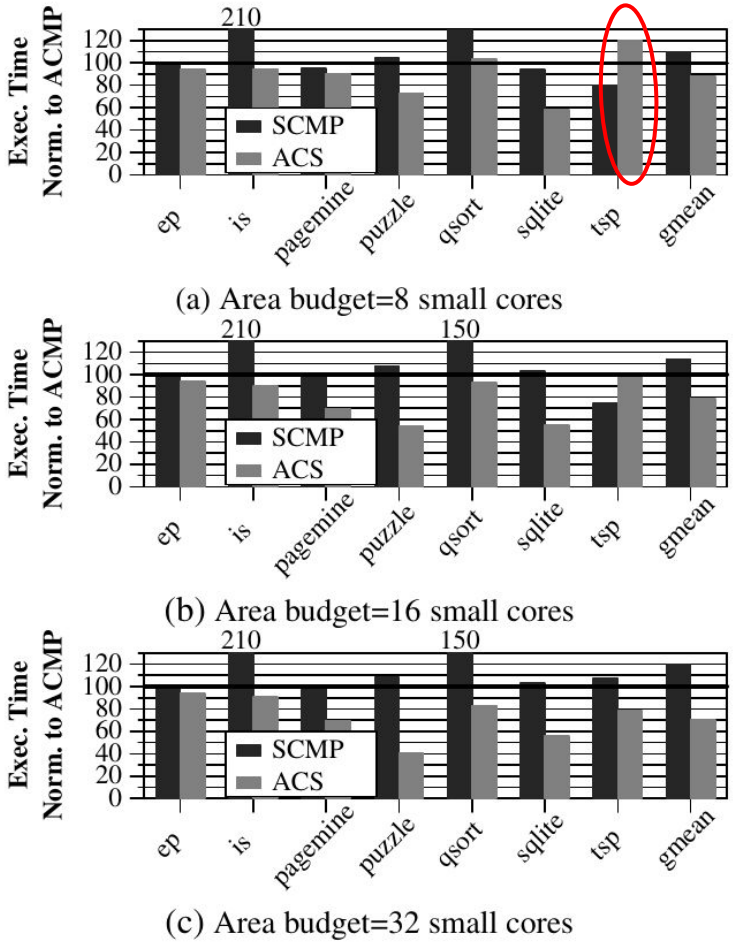
Backup Slides

Hardware specialization?



Adi Fuchs, David Wentzlaff, *"Scaling Datacenter Accelerators With Compute-Reuse Architectures"* (ISCA '18)

Coarse-Grained Workloads



(a) Area budget=8 small cores

(b) Area budget=16 small cores

(c) Area budget=32 small cores

Figure 8. Execution time of workloads with coarse-grained locking on ACS and SCMP normalized to ACMP

1. For an area budget of 8:

   ACS improves performance by **22%**compared to SCMP and **11%** compared to ACMP

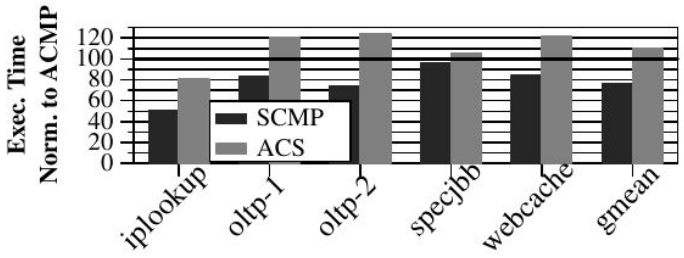
2. For an area budget of 16:

   ACS improves performance by **32%**compared to SCMP and **22%** compared to ACMP

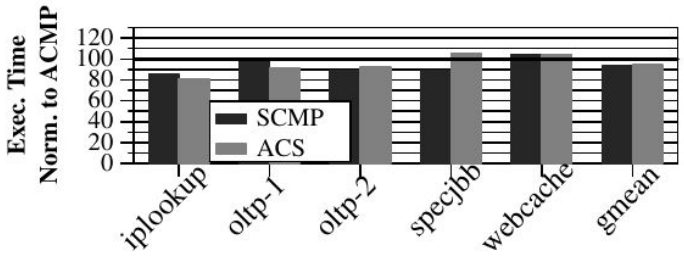
3. For an are budget of 32:

   ACS improves performance by **42%** compared to SCMP and **31%** compared to ACMP

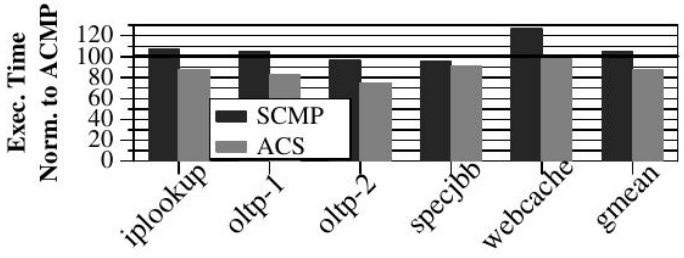
Fined Grained Workloads



(a) Area budget=8 small cores



(b) Area budget=16 small cores



(c) Area budget=32 small cores

- **The area budget required to outperform SCMP or ACMP for all workloads is less than or equal to 24 cores.**
- For an area budget of 8:

  ACS **increases** execution time for all workloads except for iploockup compared to SCMP
  and **increases** execution time for all workloads compared to ACMP
- For an area budget of **16:**

  ACS improves performance by **2%** compared to SCMP and **6%** compared to ACMP
- For an are budget of **32**:

  ACS improves performance by **17%** compared to SCMP and **6%** compared to ACMP

Scalability

- **For 7 out of 12 applications ACS improves Scalability**

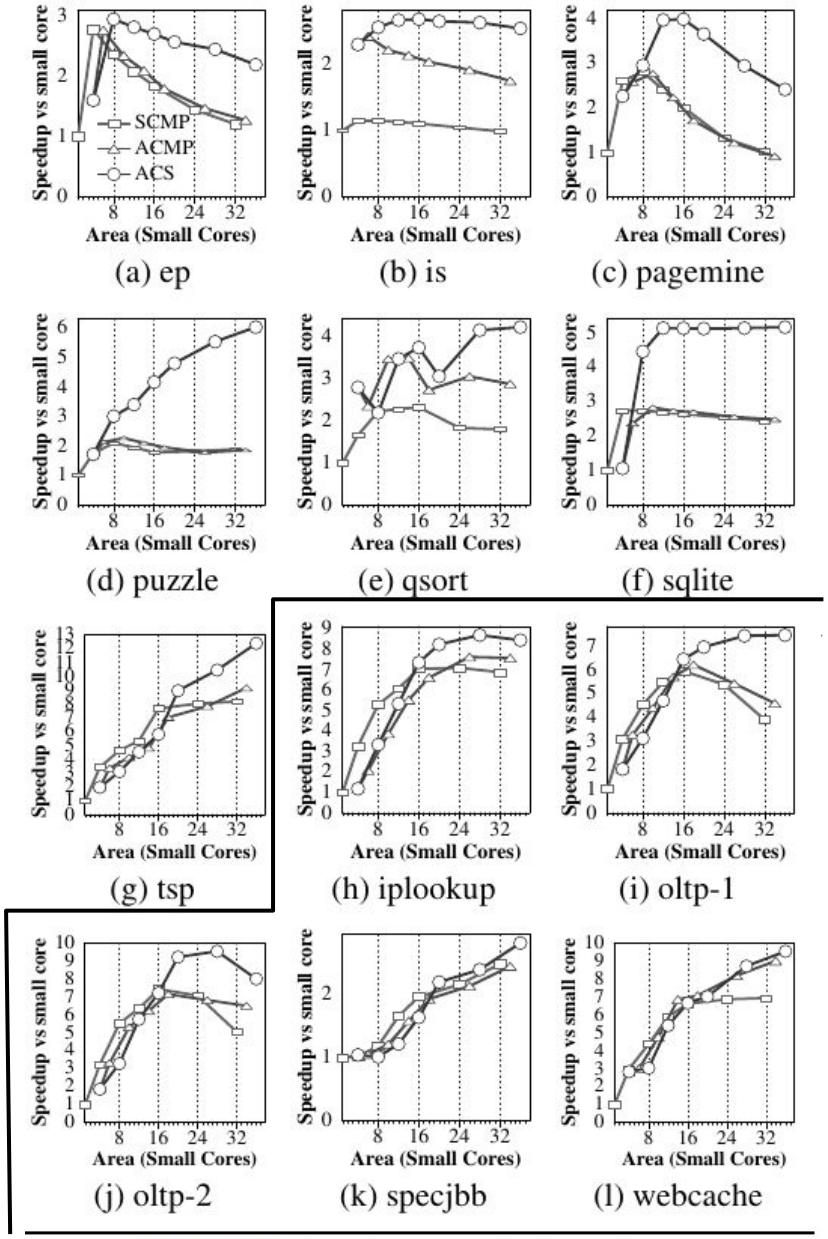


Figure 10. Speedup over a single small core