

Digital Design & Computer Arch.

Lecture 25a: Prefetching

Prof. Onur Mutlu

ETH Zürich

Spring 2021

3 June 2021

Readings for This Week and Last Week

■ Memory Hierarchy and Caches

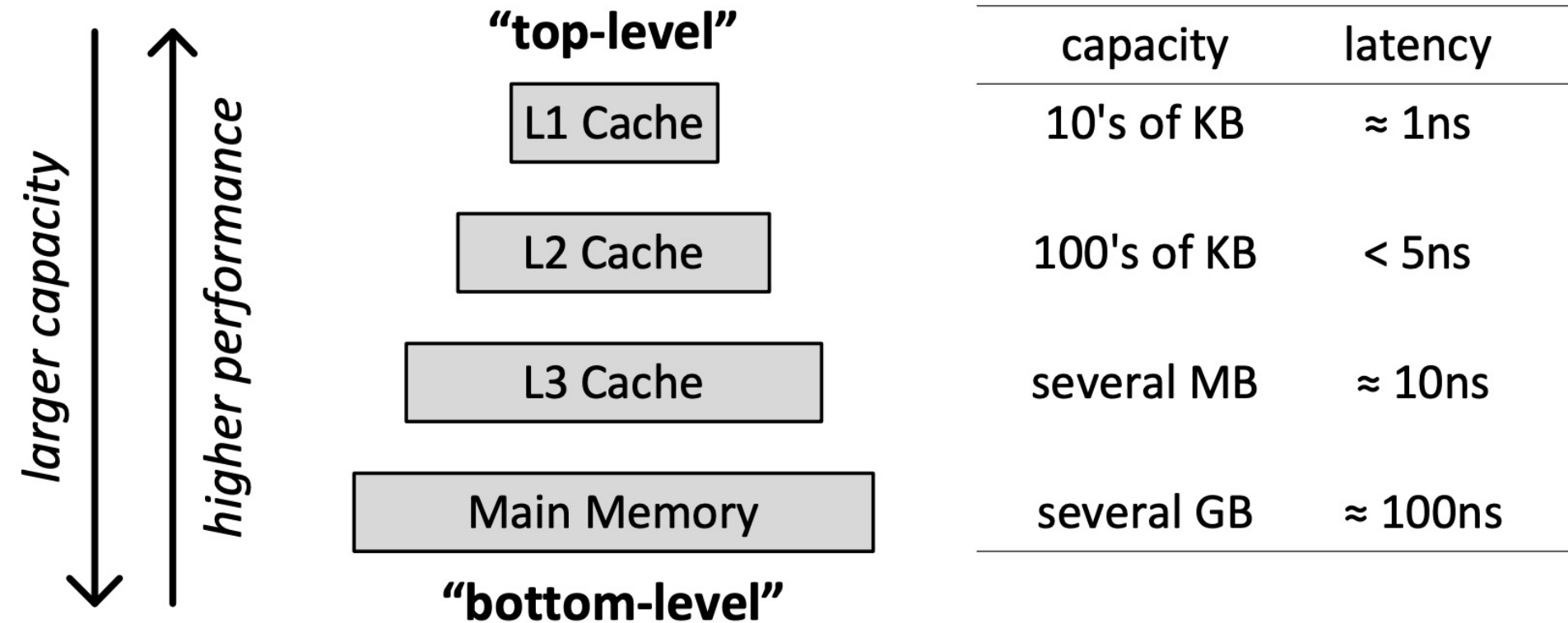
■ Required

- H&H Chapters 8.1-8.3
- Refresh: P&P Chapter 3.5
- Kim & Mutlu, "**Memory Systems**," Computing Handbook, 2014.
 - https://people.inf.ethz.ch/omutlu/pub/memory-systems-introduction_computing-handbook14.pdf

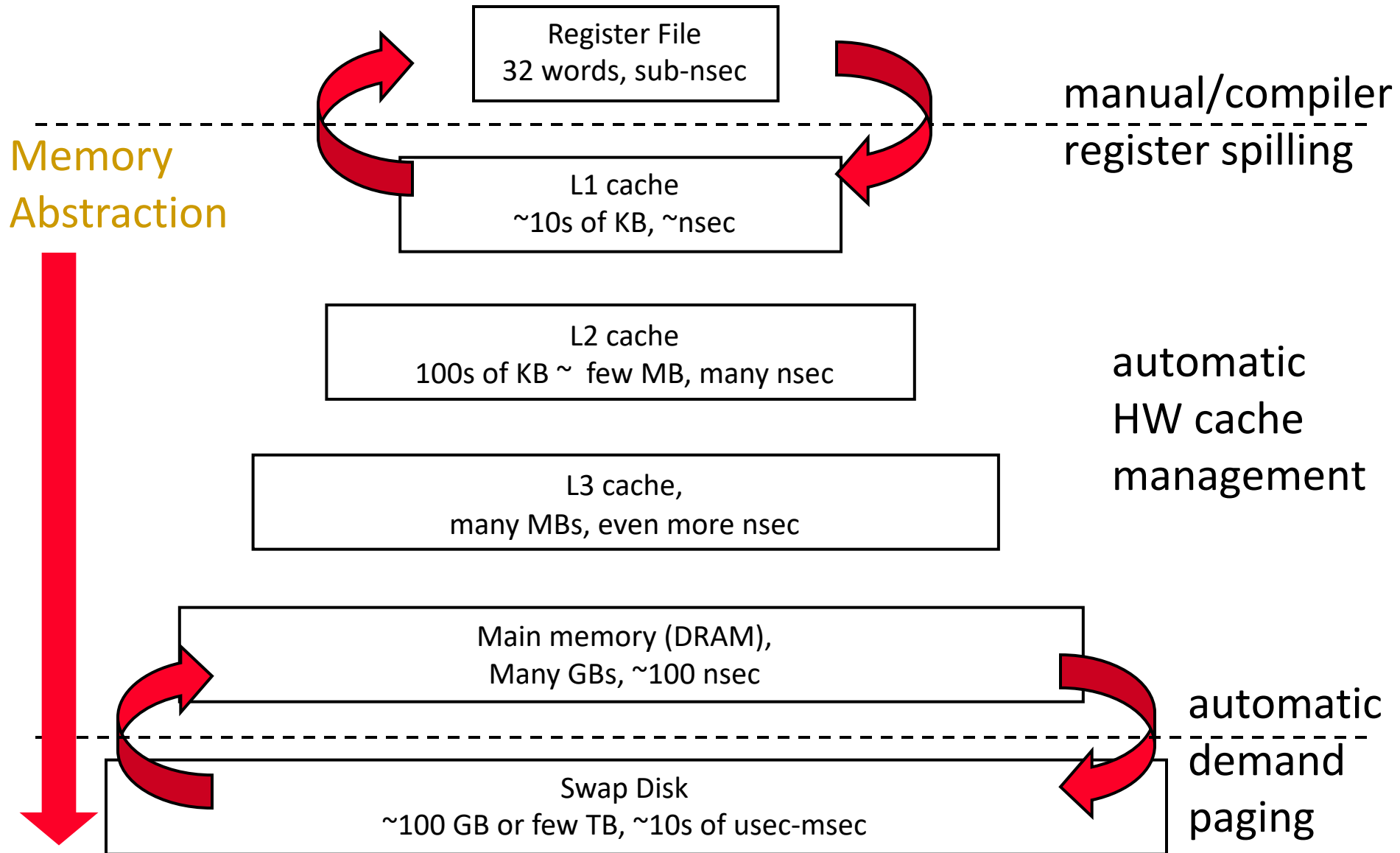
■ Recommended

- An early cache paper by Maurice Wilkes
 - Wilkes, "**Slave Memories and Dynamic Storage Allocation**," IEEE Trans. On Electronic Computers, 1965.
- An example prefetching paper
 - Mutlu et al., "**Runahead Execution: An Effective Alternative to Large Instruction Windows**," IEEE Micro, 2003.

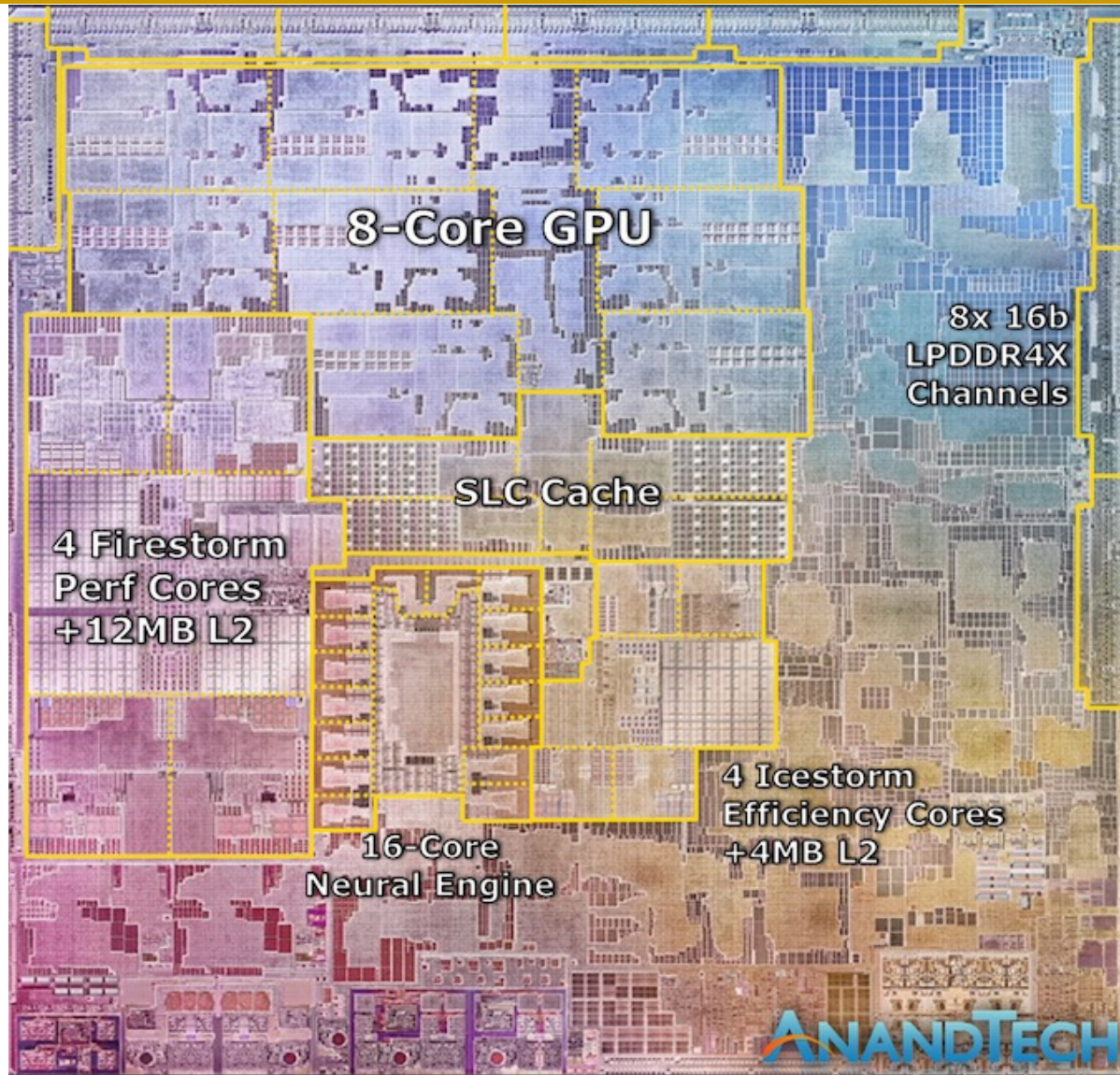
Recall: Memory Hierarchy Example



Recall: A Modern Memory Hierarchy

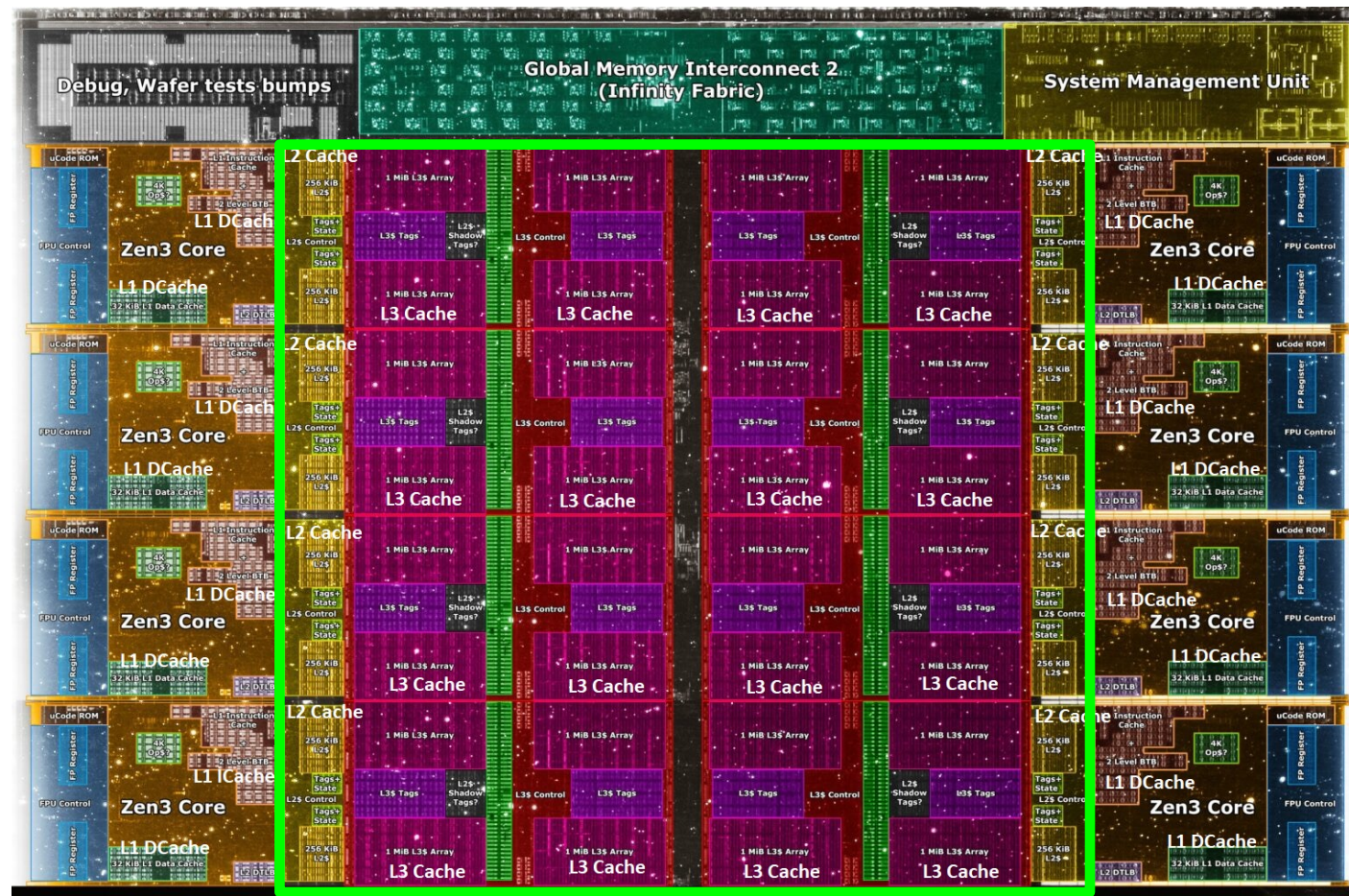


Recall: Deeper and Larger Cache Hierarchies



Apple M1,
2021

Recall: Deeper and Larger Cache Hierarchies



Core Count:

8 cores/16 threads

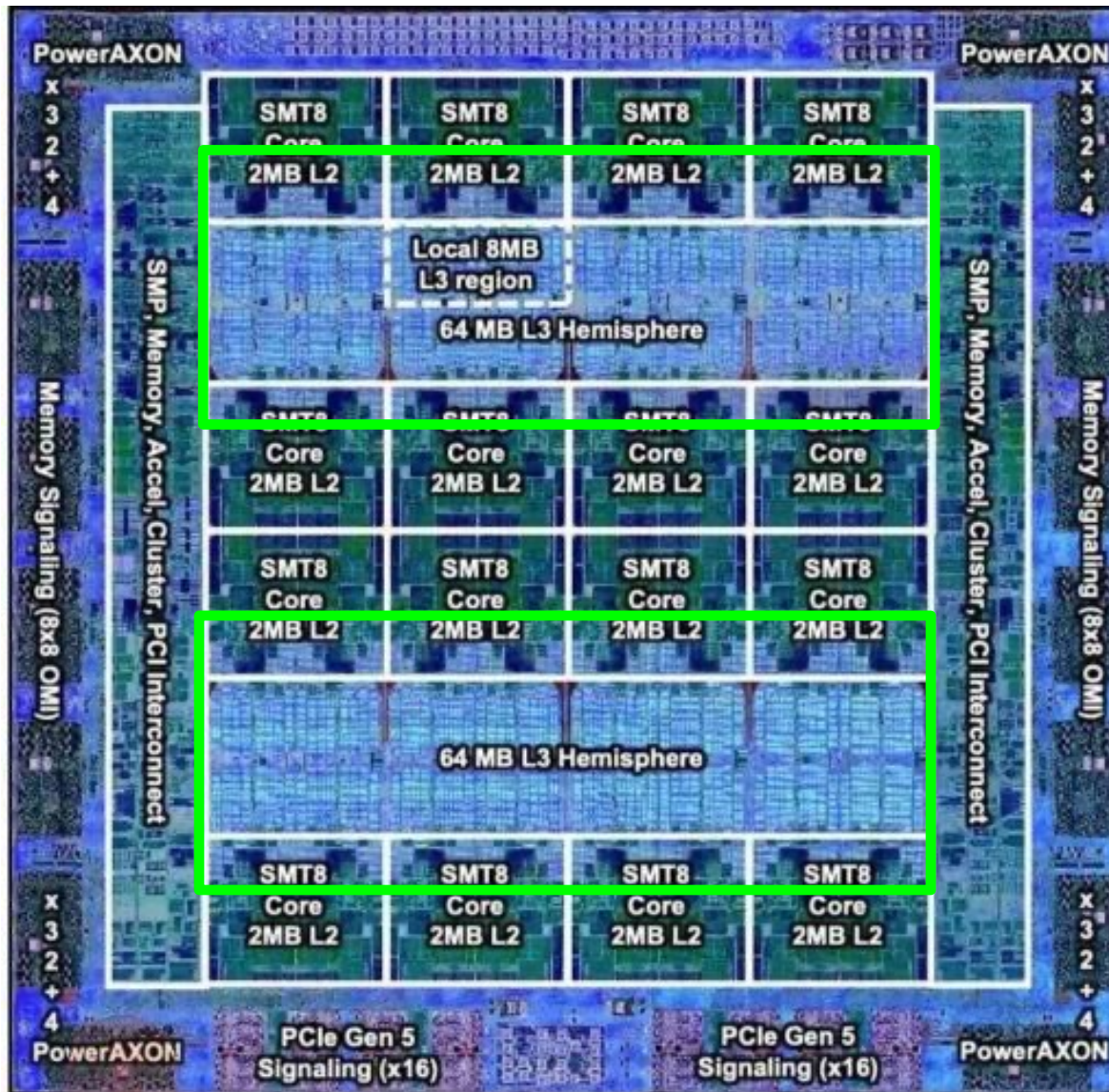
L1 Caches:
32 KB per core

L2 Caches:
512 KB per core

L3 Cache:
32 MB shared

AMD Ryzen 5000, 2020

Recall: Deeper and Larger Cache Hierarchies



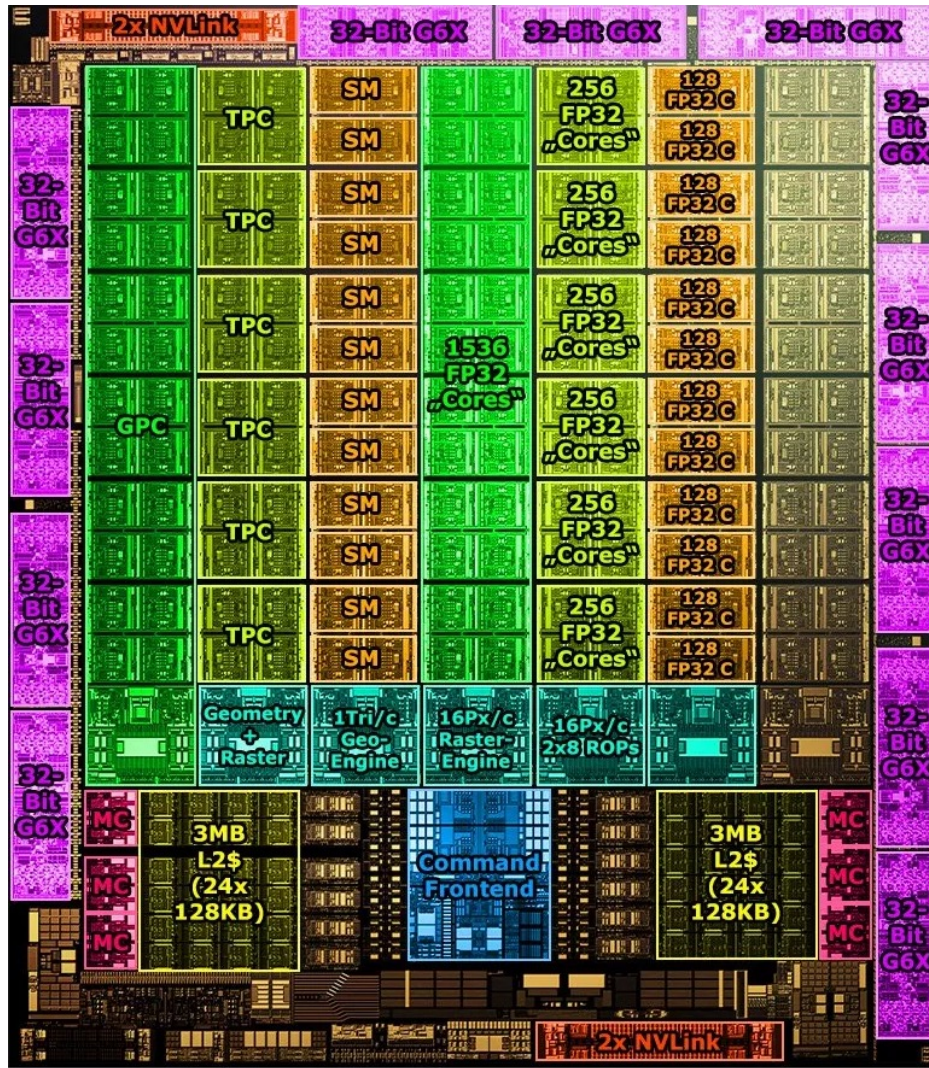
IBM POWER10,
2020

Cores:
15-16 cores,
8 threads/core

L2 Caches:
2 MB per core

L3 Cache:
120 MB shared

Recall: Deeper and Larger Cache Hierarchies



Nvidia Ampere, 2020

Cores:

128 Streaming Multiprocessors

L1 Cache or Scratchpad:

192KB per SM

Can be used as L1 Cache and/or Scratchpad

L2 Cache:

40 MB shared

Recall: How to Improve Cache Performance

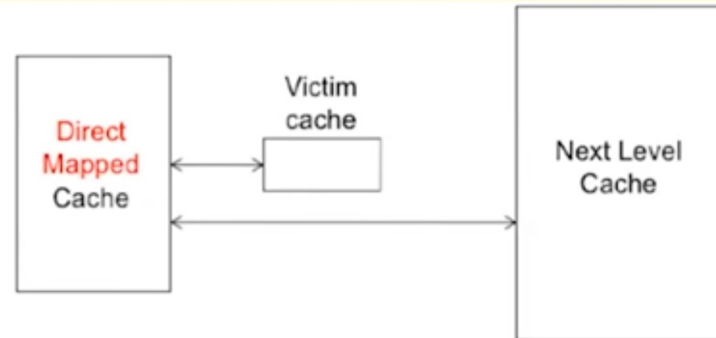
- Three fundamental goals
- **Reducing miss rate**
 - Caveat: reducing miss rate can reduce performance if more costly-to-refetch blocks are evicted
- **Reducing miss latency or miss cost**
- **Reducing hit latency or hit cost**
- The above three **together** affect performance

Recall: Improving Basic Cache Performance

- Reducing miss rate
 - ❑ More associativity
 - ❑ Alternatives/enhancements to associativity
 - Victim caches, hashing, pseudo-associativity, skewed associativity
 - ❑ Better replacement/insertion policies
 - ❑ Software approaches
 - ❑ ...
- Reducing miss latency/cost
 - ❑ Multi-level caches
 - ❑ Critical word first
 - ❑ Subblocking/sectoring
 - ❑ Better replacement/insertion policies
 - ❑ Non-blocking caches (multiple cache misses in parallel)
 - ❑ Multiple accesses per cycle
 - ❑ Software approaches
 - ❑ ...

Lectures on Cache Optimizations (I)

Victim Cache: Reducing Conflict Misses



- Jouppi, "Improving Direct-Mapped Cache Performance by the Addition of a Small Fully-Associative Cache and Prefetch Buffers," ISCA 1990.
- Idea: Use a small fully-associative buffer (victim cache) to store recently evicted blocks
 - + Can avoid ping ponging of cache blocks mapped to the same set (if two cache blocks continuously accessed in nearby time conflict with each other)
 - Increases miss latency if accessed serially with L2; adds complexity

Computer Architecture - Lecture 3: Cache Management and Memory Parallelism (ETH Zürich, Fall 2017)

6,392 views • Sep 29, 2017

49 1 SHARE SAVE ...

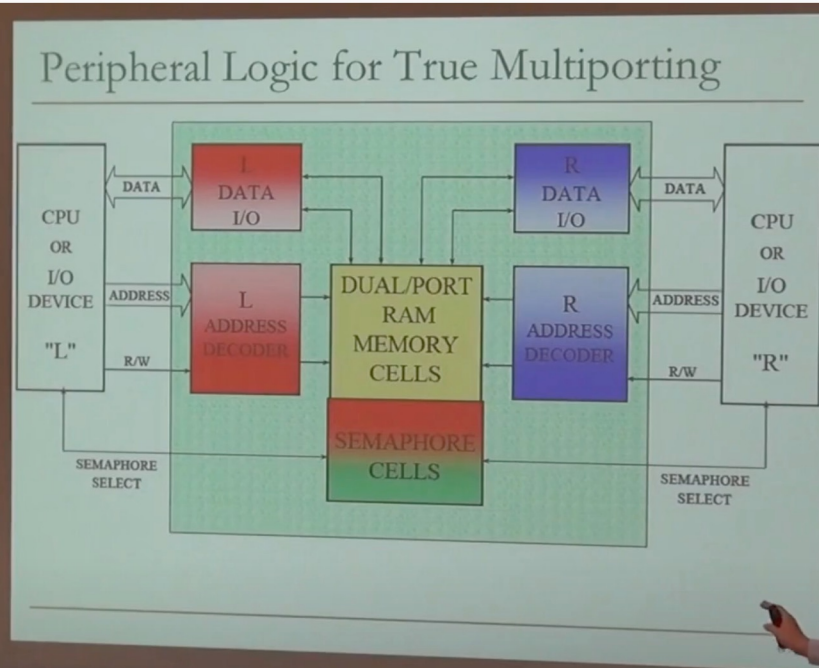


Onur Mutlu Lectures
16.3K subscribers

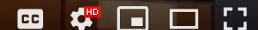
ANALYTICS

EDIT VIDEO

Lectures on Cache Optimizations (II)



1:18:05 / 1:28:10



ETH ZÜRICH HAUPTGEBÄUDE

Computer Architecture - Lecture 4a: Cache Design (ETH Zürich, Fall 2018)

1,437 views • Sep 29, 2018

15 0 SHARE SAVE ...



Onur Mutlu Lectures
16.3K subscribers

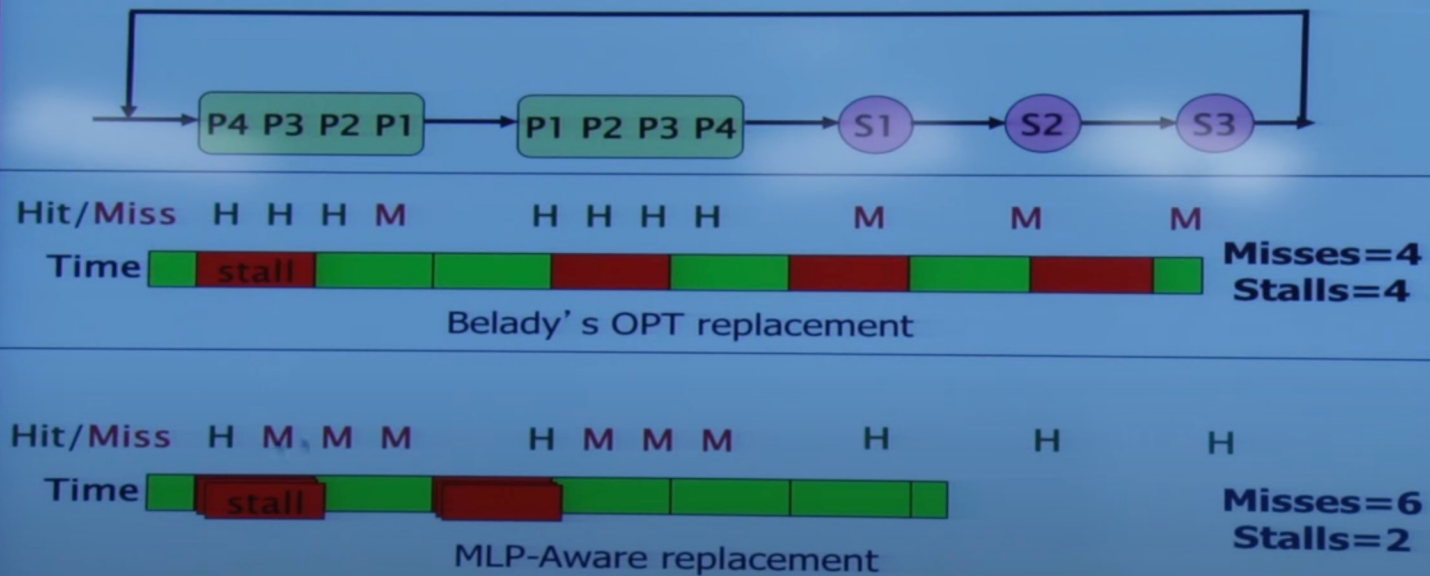
ANALYTICS

EDIT VIDEO

https://www.youtube.com/watch?v=55oYBm9cifl&list=PL5Q2soXY2Zi9JXe3ywQMhyIk_d5dl-TM7&index=6

Lectures on Cache Optimizations (III)

Fewest Misses \neq Best Performance



Lecture 19. High Performance Caches - Carnegie Mellon - Comp. Arch. 2015 - Onur Mutlu

9,737 views • Mar 5, 2015

63 1 SHARE SAVE ...



Carnegie Mellon Computer Architecture
23.2K subscribers

ANALYTICS

EDIT VIDEO

Lectures on Cache Optimizations

■ Computer Architecture, Fall 2017, Lecture 3

- ❑ Cache Management & Memory Parallelism (ETH, Fall 2017)
- ❑ https://www.youtube.com/watch?v=OyomXCHNJDA&list=PL5Q2soXY2Zi9OhoVQBX_YFIZywZXCPI4M_&index=3

■ Computer Architecture, Fall 2018, Lecture 4a

- ❑ Cache Design (ETH, Fall 2018)
- ❑ https://www.youtube.com/watch?v=55oYBm9cifI&list=PL5Q2soXY2Zi9JXe3ywQMh_ylk_d5dI-TM7&index=6

■ Computer Architecture, Spring 2015, Lecture 19

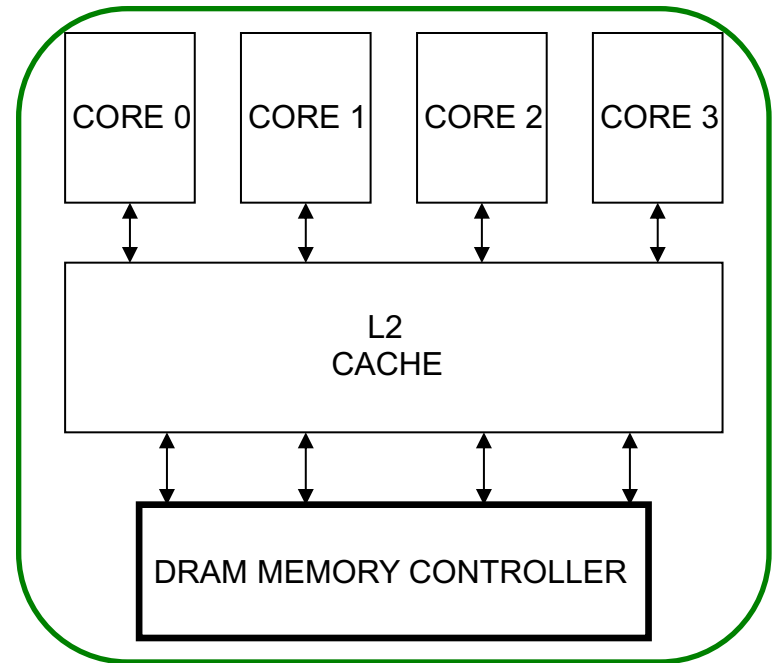
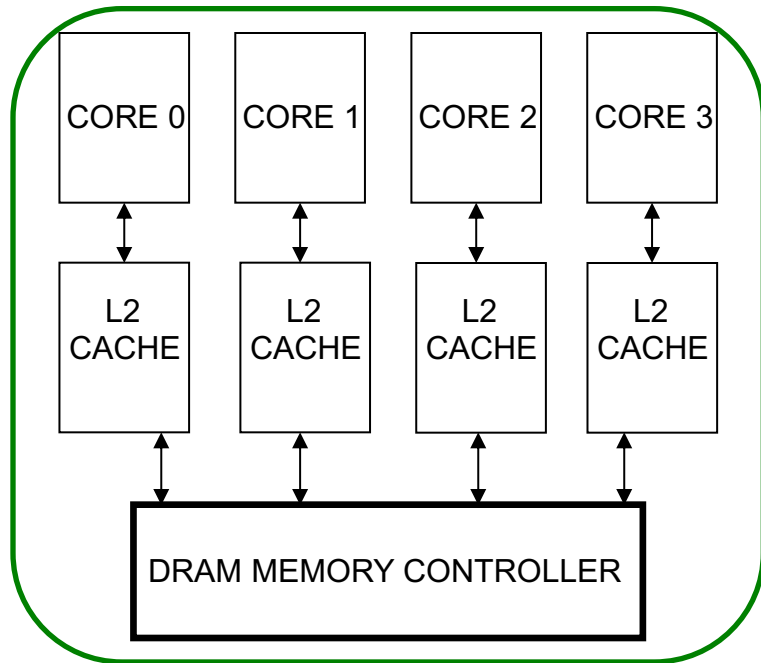
- ❑ High Performance Caches (CMU, Spring 2015)
- ❑ <https://www.youtube.com/watch?v=jDHx2K9HxIM&list=PL5PHm2jkkXmi5CxxI7b3JCL1TWybTDtKq&index=21>

Recall:

Multi-Core Issues in Caching

Recall: Private vs. Shared Caches

- **Private** cache: Cache belongs to one core (a shared block can be in multiple caches)
- **Shared** cache: Cache is shared by multiple cores



Lectures on Multi-Core Cache Management

Computer Architecture

Lecture 15:

Multi-Core Cache Management

Prof. Onur Mutlu
ETH Zürich
Fall 2017
15 November 2017



0:35 / 2:33:03



Computer Architecture - Lecture 15: Multi-Core Cache Management (ETH Zürich, Fall 2017)

934 views • Nov 17, 2017

13 0 SHARE SAVE ...

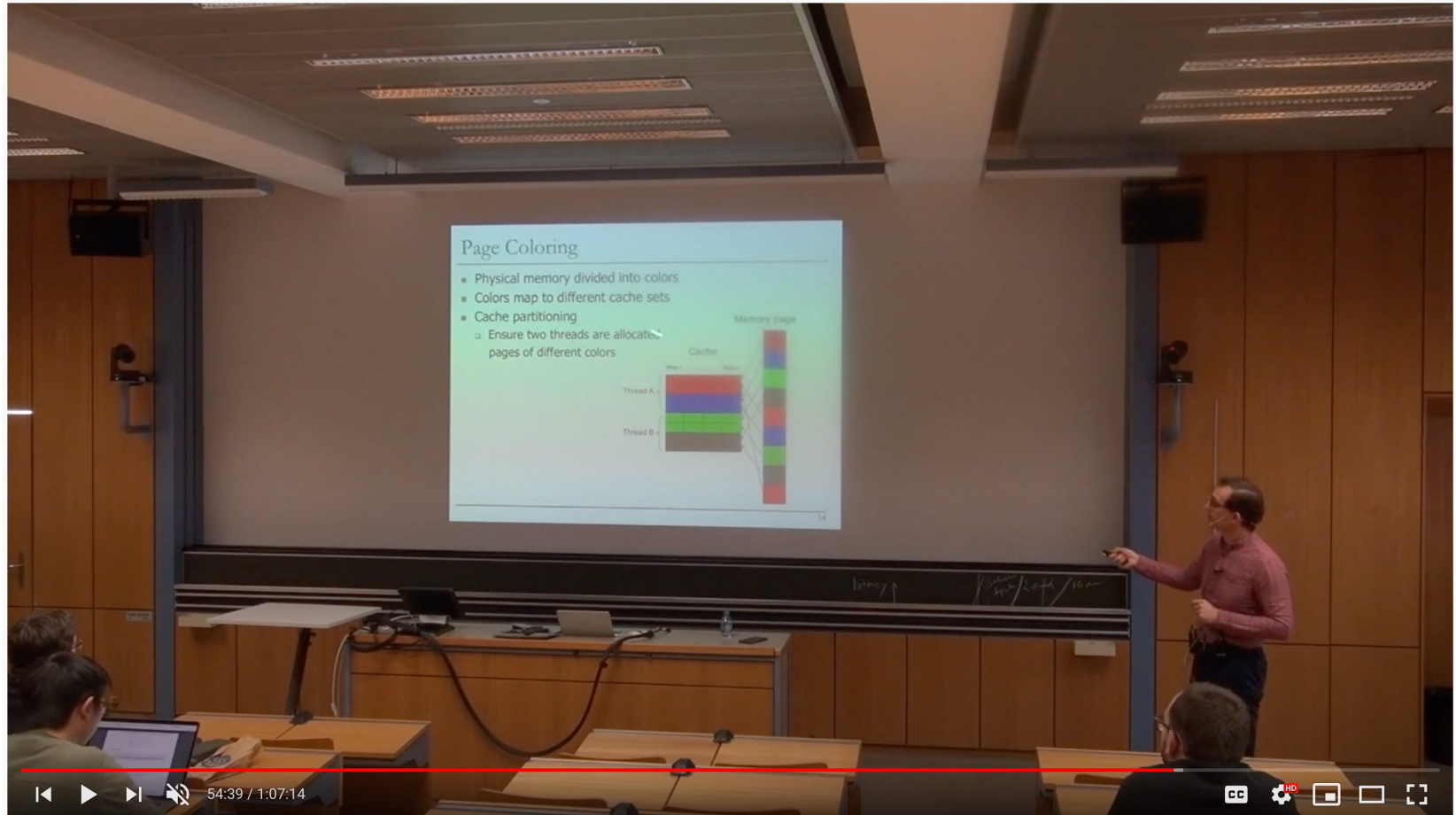


Onur Mutlu Lectures
16.5K subscribers

ANALYTICS

EDIT VIDEO

Lectures on Multi-Core Cache Management



ETH ZÜRICH HAUPTGEBÄUDE

Computer Architecture - Lecture 18b: Multi-Core Cache Management (ETH Zürich, Fall 2018)

744 views • Nov 23, 2018

12 0 SHARE SAVE ...



Onur Mutlu Lectures
16.5K subscribers

ANALYTICS

EDIT VIDEO

https://www.youtube.com/watch?v=c9FhGRB3HoA&list=PL5Q2soXY2Zi9JXe3ywQMhyIk_d5dl-TM7&index=29

Lectures on Multi-Core Cache Management

Approaches to Reuse Prediction

Use program counter or memory region information.

1. Group Blocks

PC 1 PC 2

A B S T

2. Learn group behavior

PC 1 PC 2

A B S T

3. Predict reuse

PC 1 C → C

PC 2 U → U

1. Same group ↗ same reuse behavior

2. No control over number of high-reuse blocks

77

ETH ZÜRICH HAUPTGEBÄUDE
Computer Architecture - Lecture 19a: Multi-Core Cache Management II (ETH Zürich, Fall 2018)

293 views • Dec 2, 2018

6 0 SHARE SAVE ...



Onur Mutlu Lectures
16.5K subscribers

ANALYTICS

EDIT VIDEO

Lectures on Multi-Core Cache Management

- **Computer Architecture, Fall 2018, Lecture 18b**
 - Multi-Core Cache Management (ETH, Fall 2018)
 - https://www.youtube.com/watch?v=c9FhGRB3HoA&list=PL5Q2soXY2Zi9JXe3ywQMhYlk_d5dI-TM7&index=29

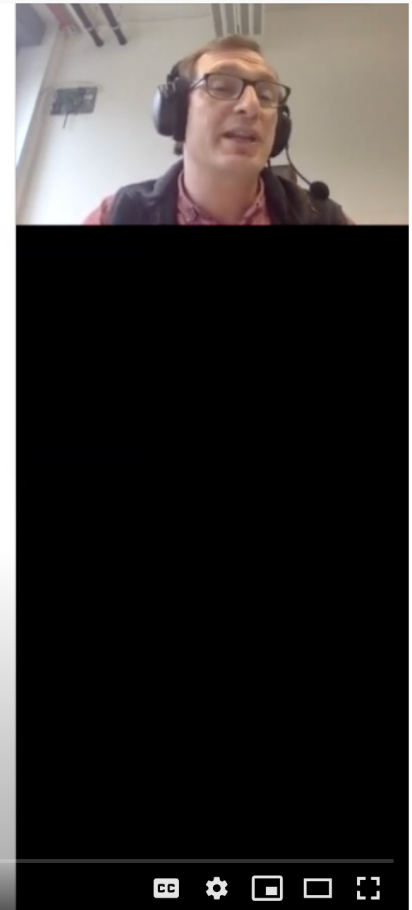
- **Computer Architecture, Fall 2018, Lecture 19a**
 - Multi-Core Cache Management II (ETH, Fall 2018)
 - https://www.youtube.com/watch?v=Siz86__PD4w&list=PL5Q2soXY2Zi9JXe3ywQMhYlk_d5dI-TM7&index=30

- **Computer Architecture, Fall 2017, Lecture 15**
 - Multi-Core Cache Management (ETH, Fall 2017)
 - https://www.youtube.com/watch?v=7_Tqlw8gxOU&list=PL5Q2soXY2Zi9OhoVQBXyFIZywZXCPI4M_&index=17

Lectures on Memory Resource Management

QoS-Aware Memory Systems: Challenges

- How do we **reduce inter-thread interference**?
 - ❑ Improve system performance and core utilization
 - ❑ Reduce request serialization and core starvation
- How do we **control inter-thread interference**?
 - ❑ Provide mechanisms to enable system software to enforce QoS policies
 - ❑ While providing high system performance
- How do we **make the memory system configurable/flexible**?
 - ❑ Enable flexible mechanisms that can achieve many goals
 - Provide fairness or throughput when needed
 - Satisfy performance guarantees when needed



ETH ZÜRICH HAUPTGEBÄUDE

Computer Architecture - Lecture 11b: Memory Interference and QoS (ETH Zürich, Fall 2020)

735 views • Oct 31, 2020

14 0 SHARE SAVE ...



Onur Mutlu Lectures
16.5K subscribers

ANALYTICS

EDIT VIDEO

Lectures on Memory Resource Management

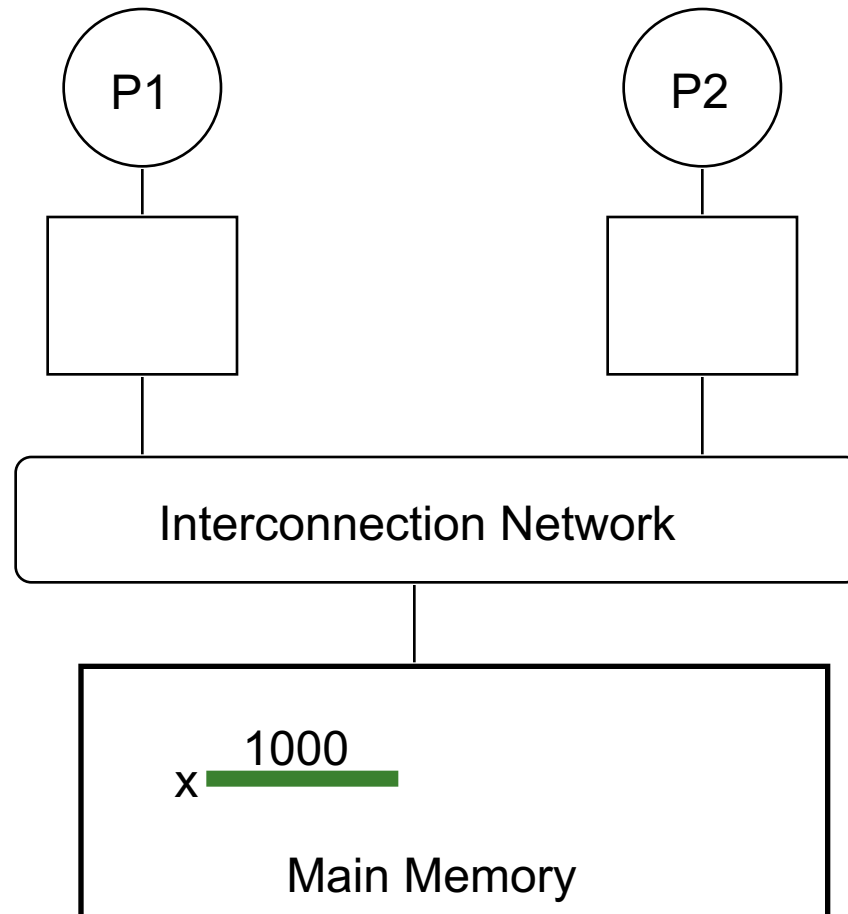
- **Computer Architecture, Fall 2020, Lecture 11a**
 - Memory Controllers (ETH, Fall 2020)
 - <https://www.youtube.com/watch?v=TeG773OgiMQ&list=PL5Q2soXY2Zi9xidyIgBxUz7xRPS-wisBN&index=20>
- **Computer Architecture, Fall 2020, Lecture 11b**
 - Memory Interference and QoS (ETH, Fall 2020)
 - <https://www.youtube.com/watch?v=0nnI807nCkc&list=PL5Q2soXY2Zi9xidyIgBxUz7xRPS-wisBN&index=21>
- **Computer Architecture, Fall 2020, Lecture 13**
 - Memory Interference and QoS II (ETH, Fall 2020)
 - <https://www.youtube.com/watch?v=Axye9VqQT7w&list=PL5Q2soXY2Zi9xidyIgBxUz7xRPS-wisBN&index=26>
- **Computer Architecture, Fall 2020, Lecture 2a**
 - Memory Performance Attacks (ETH, Fall 2020)
 - <https://www.youtube.com/watch?v=VJzZbwgBfy8&list=PL5Q2soXY2Zi9xidyIgBxUz7xRPS-wisBN&index=2>

Recall:

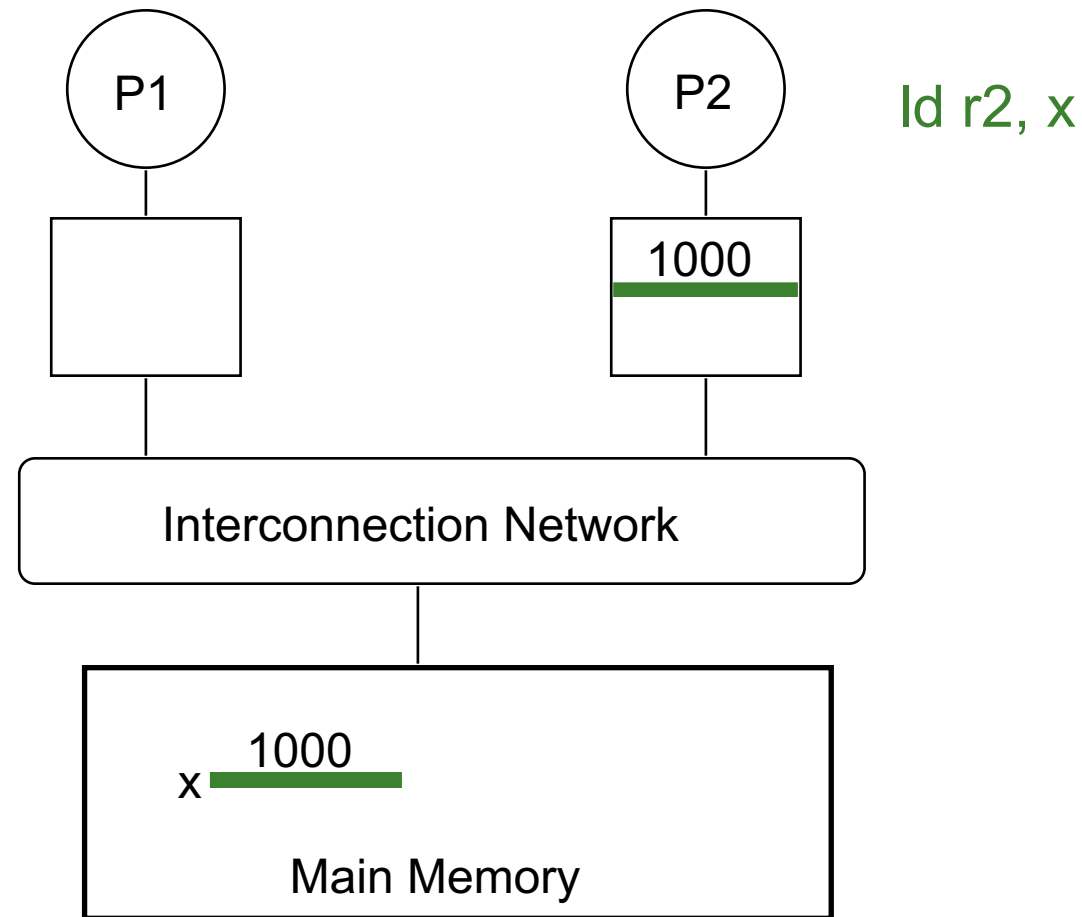
Cache Coherence

Recall: Cache Coherence

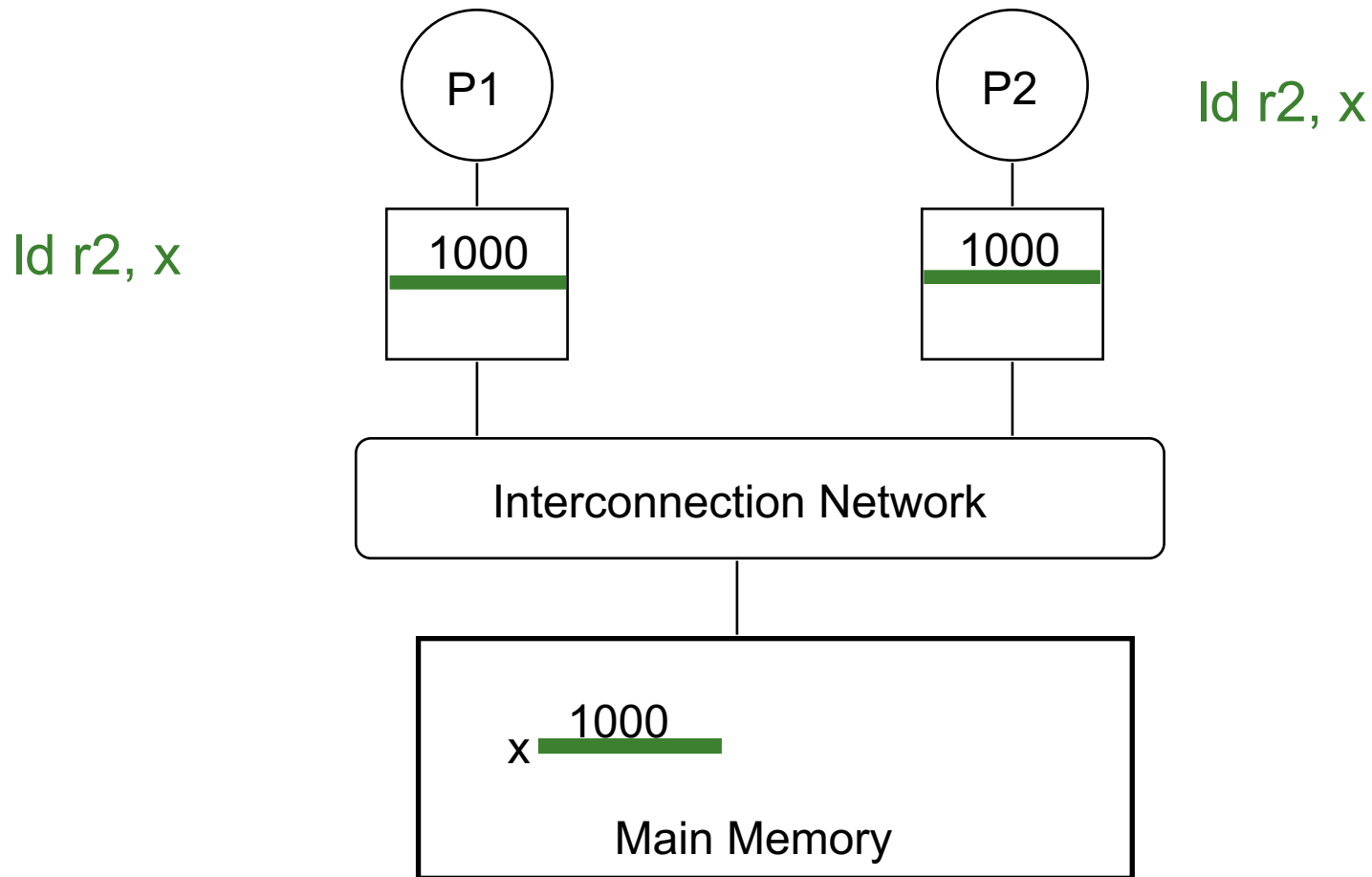
- Basic question: If multiple processors cache the same block, how do they ensure they all see a consistent state?



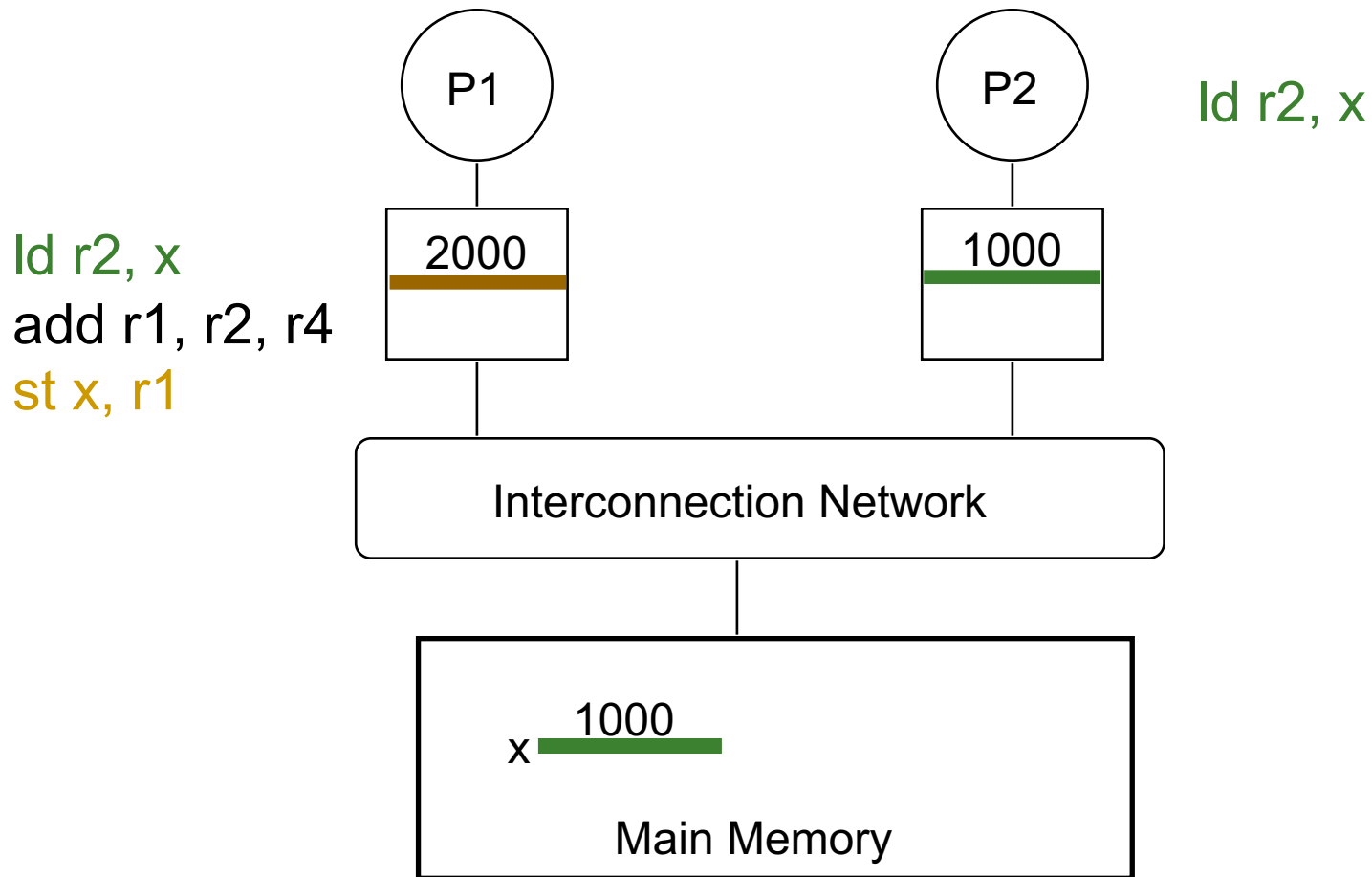
Recall: The Cache Coherence Problem



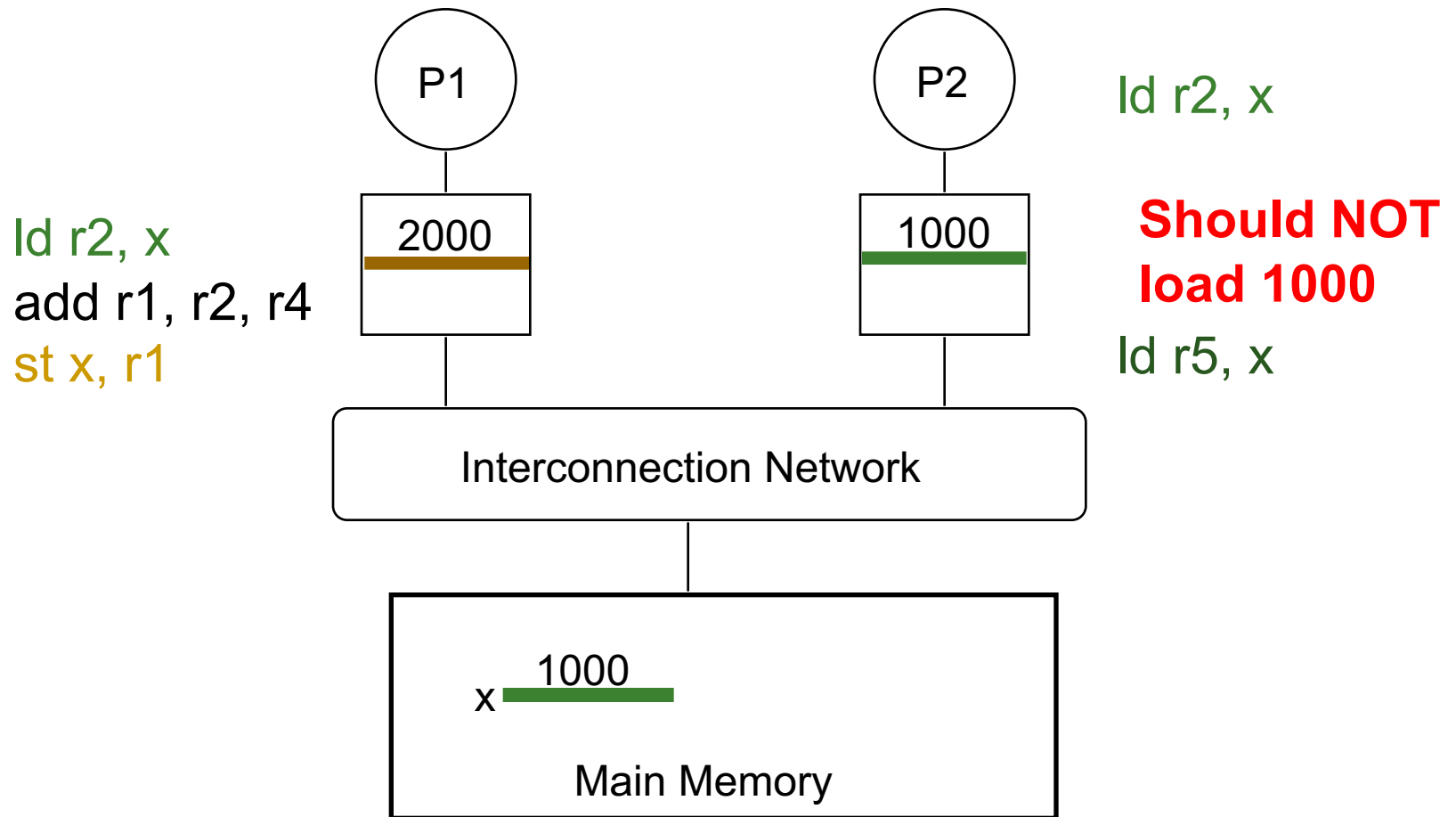
Recall: The Cache Coherence Problem



Recall: The Cache Coherence Problem

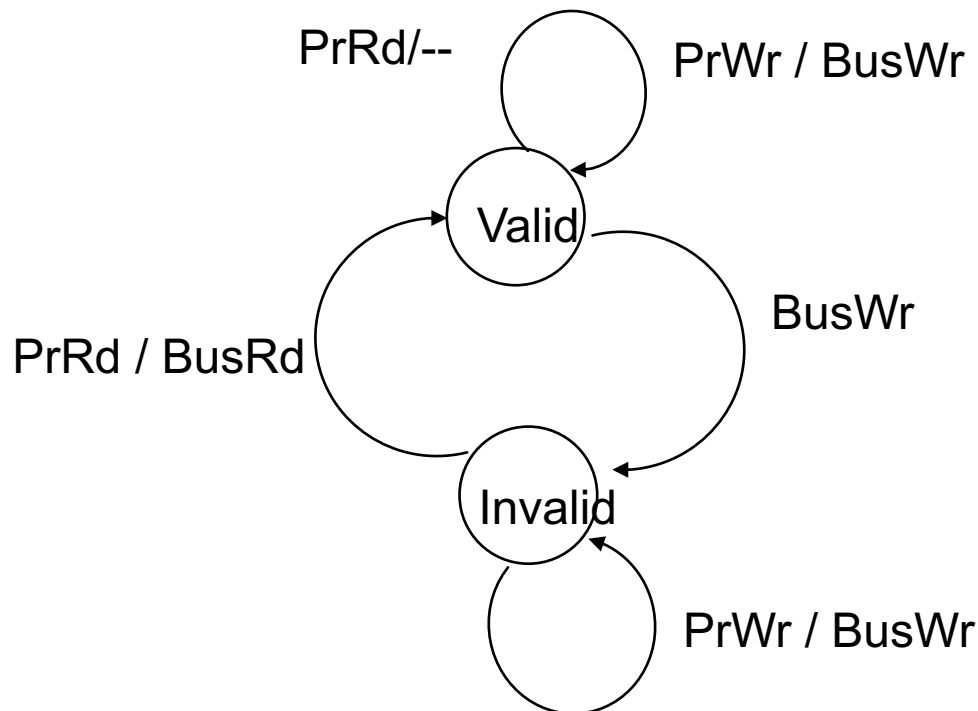


Recall: The Cache Coherence Problem



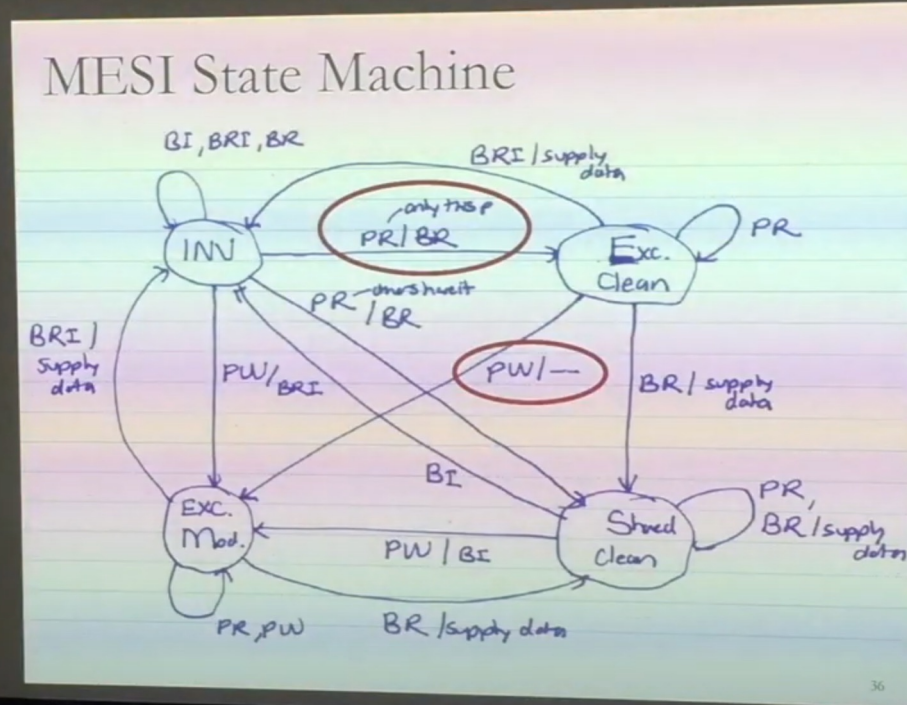
Recall: A Very Simple Coherence Scheme (VI)

- Idea: All caches “snoop” (observe) each other’s write/read operations. If a processor writes to a block, all others invalidate the block.
- A simple protocol:



- Write-through, no-write-allocate cache
- Actions of the local processor on the cache block: PrRd, PrWr,
- Actions that are broadcast on the bus for the block: BusRd, BusWr

Lecture on Cache Coherence



Computer Architecture - Lecture 21: Cache Coherence (ETH Zürich, Fall 2020)

1,419 views • Dec 4, 2020

27 0 SHARE SAVE ...



Onur Mutlu Lectures
16.3K subscribers

ANALYTICS

EDIT VIDEO

Lecture on Memory Ordering & Consistency

Diagram: Two processors, P1 and P2, are connected via an Interconnection Network to shared memory. P1 has a local cache for F2, and P2 has a local cache for F1. The memory contains F2 and F1. P1's view of memory is A (F1=1, F2=0) and B (F2=1, F1=0). P2's view of memory is X (F2=1, F1=0) and Y (F1=1, F2=0). Both views are in a Critical Section (CSL).

Timeline:

- 0: P1: A (Set F1=1), F1=1 complete from P1's view, A sent to memory. P2: X (Set F2=1), F2=1 complete from P2's view, X sent to memory.
- 1: P1: B (test F2=0), B (req F2) sent to mem., B (load F2) started. P2: Y (test F1=0), Y (req F1) sent to mem., Y (load F1) started.
- 50: Mem. sends F2(0) to P1, B (load F2) complete. Mem. sends F1(0) to P2, Y (load F1) complete.
- 51: P1 is in Critical Section(), P2 is in Critical Section().
- 100: Mem. completes F1=1 in memory, TOO LATE!. Mem. completes F2=1 in memory, TOO LATE!.

For P1: A appeared to happen before X

For P2: X appeared to happen before A

P1's VIEW: A → B → X, A → X

P2's VIEW: X → Y → A, X → A

BOTH CANNOT BE CORRECT! (from memory's perspective)

P1 and P2 saw an inconsistent order of operations in memory

Computer Architecture - Lecture 20: Memory Ordering (Memory Consistency) (ETH Zürich, Fall 2020)

976 views • Dec 4, 2020

22 0 SHARE SAVE ...



Onur Mutlu Lectures
16.5K subscribers

ANALYTICS

EDIT VIDEO

Lecture on Cache Coherence & Consistency

■ Computer Architecture, Fall 2020, Lecture 21

- ❑ Cache Coherence (ETH, Fall 2020)
- ❑ <https://www.youtube.com/watch?v=T9WlyezeaII&list=PL5Q2soXY2Zi9xidyIgBxUz7xRPS-wisBN&index=38>

■ Computer Architecture, Fall 2020, Lecture 20

- ❑ Memory Ordering & Consistency (ETH, Fall 2020)
- ❑ <https://www.youtube.com/watch?v=Suy09mzTbiQ&list=PL5Q2soXY2Zi9xidyIgBxUz7xRPS-wisBN&index=37>

■ Computer Architecture, Spring 2015, Lecture 28

- ❑ Memory Consistency & Cache Coherence (CMU, Spring 2015)
- ❑ <https://www.youtube.com/watch?v=JfjT1a0vi4E&list=PL5PHm2jkkXmi5CxxI7b3JCL1TWybTDtKq&index=32>

■ Computer Architecture, Spring 2015, Lecture 29

- ❑ Cache Coherence (CMU, Spring 2015)
- ❑ <https://www.youtube.com/watch?v=X6DZchnMYcw&list=PL5PHm2jkkXmi5CxxI7b3JCL1TWybTDtKq&index=33>

Prefetching

Prefetching

- Idea: Fetch the data before it is needed (i.e. pre-fetch) by the program
- Why?
 - Memory latency is high. If we can prefetch accurately and early enough we can reduce/eliminate that latency.
 - Can eliminate compulsory cache misses
 - Can it eliminate all cache misses? Capacity, conflict? Coherence?
- Involves predicting which address will be needed in the future
 - Works if programs have predictable miss address patterns

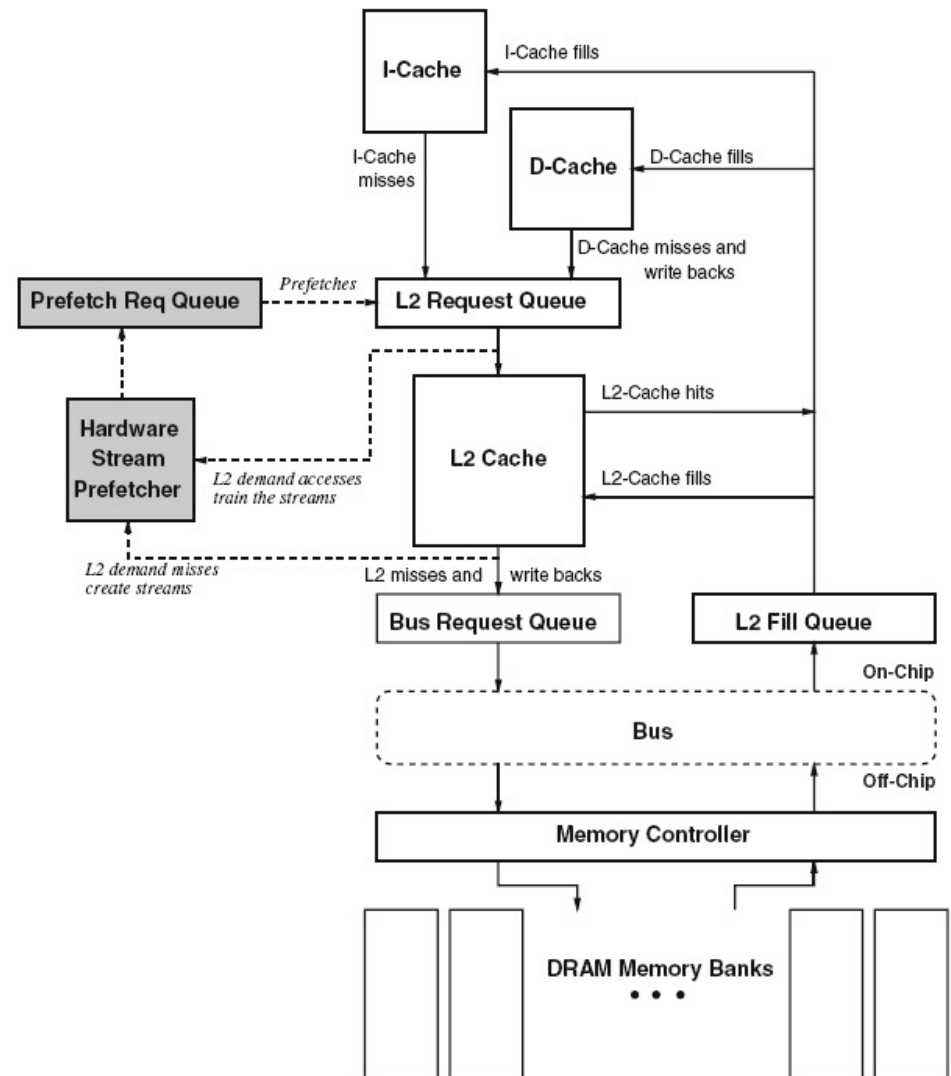
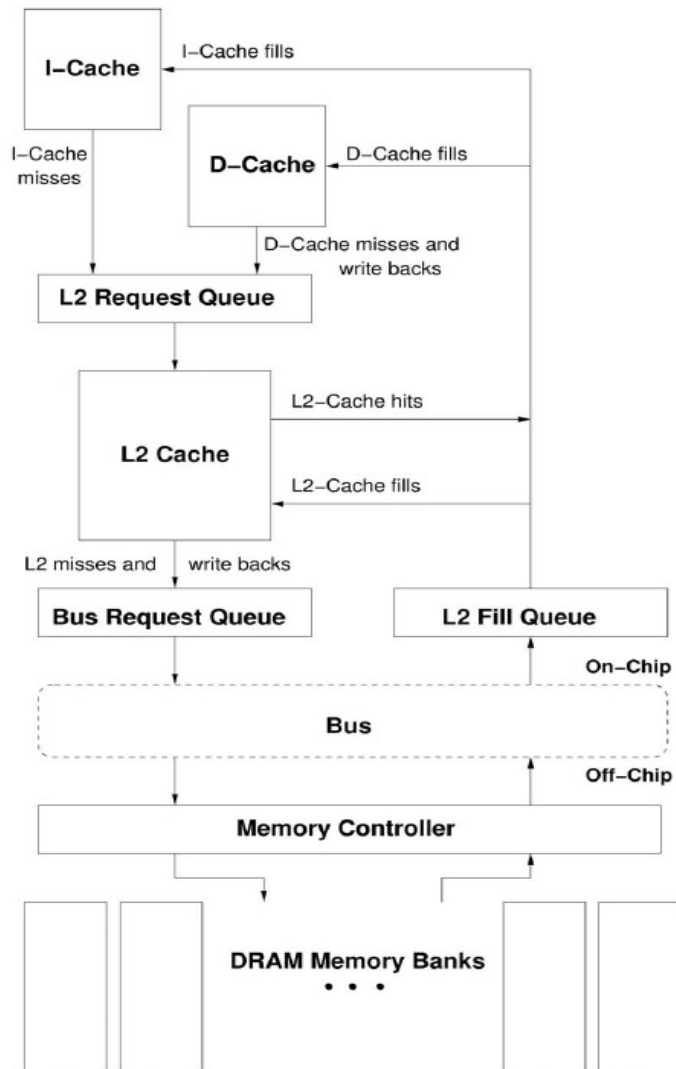
Prefetching and Correctness

- Does a misprediction in prefetching affect correctness?
- No, prefetched data at a “mispredicted” address is simply not used
- There is no need for state recovery
 - In contrast to branch misprediction or value misprediction

Basics

- In modern systems, prefetching is usually done in **cache block granularity**
- Prefetching is a technique that can reduce both
 - Miss rate
 - Miss latency
- Prefetching can be done by
 - Hardware
 - Compiler
 - Programmer
 - System

How a HW Prefetcher Fits in the Memory System



Prefetching: The Four Questions

■ What

- What addresses to prefetch (i.e., address prediction algorithm)

■ When

- When to initiate a prefetch request (early, late, on time)

■ Where

- Where to place the prefetched data (caches, separate buffer)
- Where to place the prefetcher (which level in memory hierarchy)

■ How

- How does the prefetcher operate and who operates it (software, hardware, execution/thread-based, cooperative, hybrid)

Challenge in Prefetching: What

- **What** addresses to prefetch
 - Prefetching useless data wastes resources
 - Memory bandwidth
 - Cache or prefetch buffer space
 - Energy consumption
 - These could all be utilized by demand requests or more accurate prefetch requests
 - **Accurate** prediction of addresses to prefetch is important
 - Prefetch accuracy = used prefetches / sent prefetches
- **How do we know what to prefetch?**
 - Predict based on past access patterns
 - Use the compiler's knowledge of data structures
- **Prefetching algorithm** determines what to prefetch

Challenges in Prefetching: How

- **Software** prefetching
 - ❑ ISA provides prefetch instructions
 - ❑ Programmer or compiler inserts prefetch instructions (effort)
 - ❑ Usually works well only for “regular access patterns”
- **Hardware** prefetching
 - ❑ Hardware monitors memory accesses
 - ❑ Memorizes or finds patterns/strides
 - ❑ Generates prefetch addresses automatically
- **Execution-based** prefetchers
 - ❑ A “thread” is executed to prefetch data for the main program
 - ❑ Can be generated by either software/programmer or hardware

Streaming Prefetcher in IBM POWER4

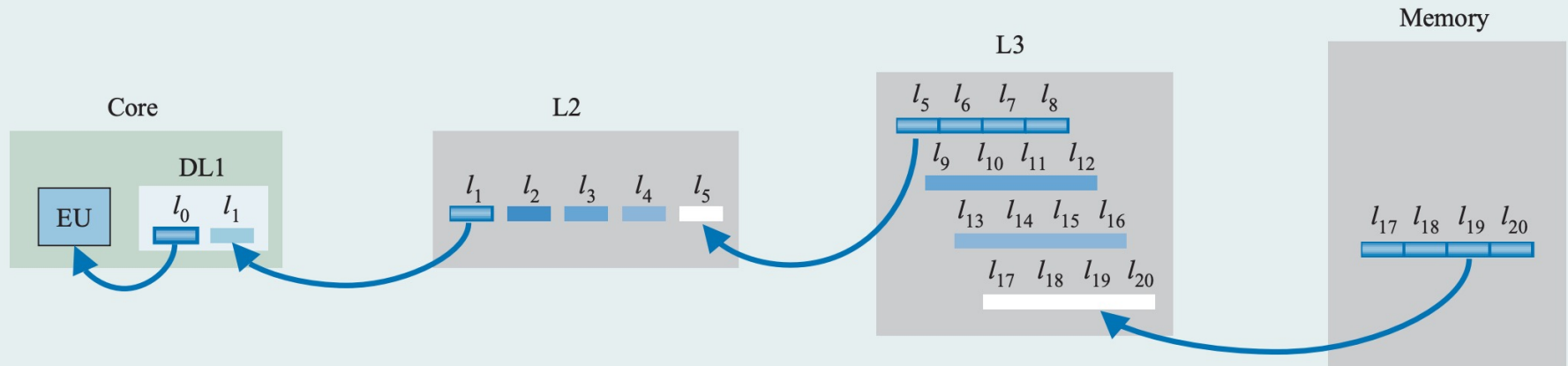


Figure 8

POWER4 hardware data prefetch.

Hardware data prefetch

POWER4 systems employ hardware to prefetch data transparently to software into the L1 data cache. When load instructions miss sequential cache lines, either ascending or descending, the prefetch engine initiates accesses to the following cache lines before being referenced by load instructions. In order to ensure that the data will be in the L1 data cache, data is prefetched into the L2 from the L3 and into the L3 from memory. **Figure 8** shows the sequence of prefetch operations. Eight such streams per processor are supported.

A Recommended Paper: Stream Prefetching

Improving Direct-Mapped Cache Performance by the Addition of a Small Fully-Associative Cache and Prefetch Buffers

Norman P. Jouppi

Digital Equipment Corporation Western Research Lab

100 Hamilton Ave., Palo Alto, CA 94301

Abstract

Projections of computer technology forecast processors with peak performance of 1,000 MIPS in the relatively near future. These processors could easily lose half or more of their performance in the memory hierarchy if the hierarchy design is based on conventional caching techniques. This paper presents hardware techniques to improve the performance of caches.

Miss caching places a small fully-associative cache between a cache and its refill path. Misses in the cache that hit in the miss cache have only a one cycle miss penalty, as opposed to a many cycle miss penalty without the miss cache. Small miss caches of 2 to 5 entries are shown to be very effective in removing mapping conflict misses in first-level direct-mapped caches.

Victim caching is an improvement to miss caching that loads the small fully-associative cache with the victim of a miss and not the requested line. Small victim caches of 1 to 5 entries are even more effective at removing conflict misses than miss caching.

Stream buffers prefetch cache lines starting at a cache miss address. The prefetched data is placed in the buffer and not in the cache. Stream buffers are useful in removing capacity and compulsory cache misses, as well as some instruction cache conflict misses. Stream buffers are more effective than previously investigated prefetch techniques at using the next slower level in the memory hierarchy when it is pipelined. An extension to the basic stream buffer, called *multi-way stream buffers*, is introduced. Multi-way stream buffers are useful for prefetching along multiple intertwined data reference streams.

Together, victim caches and stream buffers reduce the miss rate of the first level in the cache hierarchy by a factor of two to three on a set of six large benchmarks.

dous increases in miss cost. For example, a cache miss on a VAX 11/780 only costs 60% of the average instruction execution. Thus even if every instruction had a cache miss, the machine performance would slow down by only 60%! However, if a RISC machine like the WRL Titan [10] has a miss, the cost is almost ten instruction times. Moreover, these trends seem to be continuing, especially the increasing ratio of memory access time to machine cycle time. In the future a cache miss all the way to main memory on a superscalar machine executing two instructions per cycle could cost well over 100 instruction times! Even with careful application of well-known cache design techniques, machines with main memory latencies of over 100 instruction times can easily lose over half of their potential performance to the memory hierarchy. This makes both hardware and software research on advanced memory hierarchies increasingly important.

Machine	cycles per instr	cycle time (ns)	mem time (ns)	miss cost (cycles)	miss cost (instr)
VAX11/780	10.0	200	1200	6	.6
WRL Titan	1.4	45	540	12	8.6
?	0.5	4	280	70	140.0

Table 1-1: The increasing cost of cache misses

This paper investigates new hardware techniques for increasing the performance of the memory hierarchy. Section 2 describes a baseline design using conventional caching techniques. The large performance loss due to the memory hierarchy is a detailed motivation for the techniques discussed in the remainder of the paper. Techniques for reducing misses due to mapping conflicts (i.e., lack of associativity) are presented in Section 3. An

Important: Prefetcher Performance

- **Accuracy** (used prefetches / sent prefetches)
- **Coverage** (prefetched misses / all misses)
- **Timeliness** (on-time prefetches / used prefetches)

- **Bandwidth consumption**
 - Memory bandwidth consumed with prefetcher / without prefetcher
 - Good news: **Can utilize idle bus bandwidth (if available)**

- **Cache pollution**
 - Extra demand misses due to prefetch placement in cache
 - More difficult to quantify but affects performance

Outline of Prefetching Issues

- Why prefetch? Why could/does it work?
- The four questions
 - What (to prefetch), when, where, how
- Software prefetching algorithms
- Hardware prefetching algorithms
- Execution-based prefetching techniques and algorithms
- Prefetching performance
 - Coverage, accuracy, timeliness
 - Bandwidth consumption, cache pollution
- Prefetcher throttling
- Issues in multi-core

Recommended Paper

- Santhosh Srinath, Onur Mutlu, Hyesoon Kim, and Yale N. Patt,
"Feedback Directed Prefetching: Improving the Performance and Bandwidth-Efficiency of Hardware Prefetchers"
Proceedings of the 13th International Symposium on High-Performance Computer Architecture (HPCA), pages 63-74, Phoenix, AZ, February 2007. Slides (ppt)
One of the five papers nominated for the Best Paper Award by the Program Committee.

Feedback Directed Prefetching:

Improving the Performance and Bandwidth-Efficiency of Hardware Prefetchers

Santhosh Srinath^{†‡} Onur Mutlu[§] Hyesoon Kim[‡] Yale N. Patt[‡]

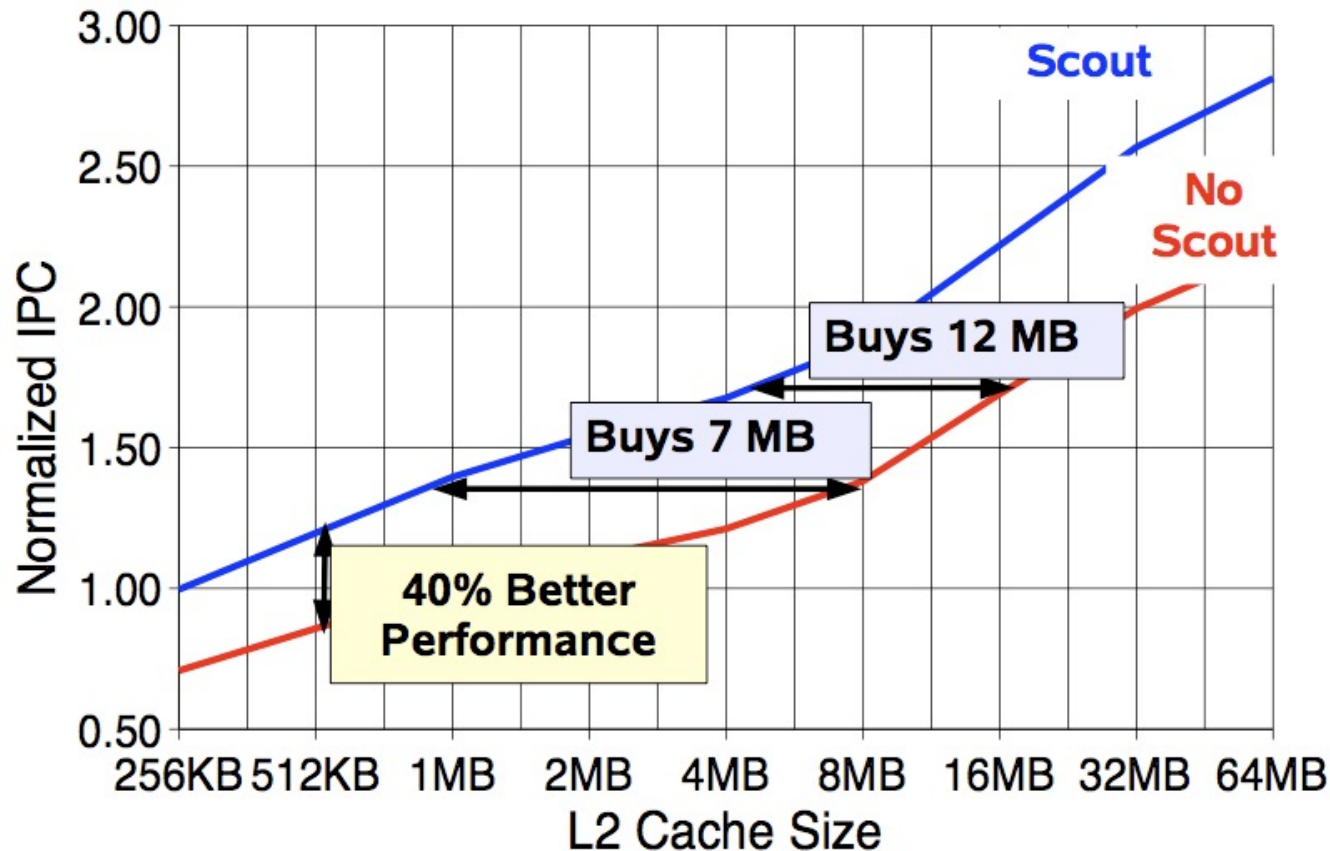
[†]Microsoft
ssri@microsoft.com

[§]Microsoft Research
onur@microsoft.com

[‡]Department of Electrical and Computer Engineering
The University of Texas at Austin
{santhosh, hyesoon, patt}@ece.utexas.edu

Effect of Runahead Prefetching in Sun ROCK

- Shailender Chaudhry talk, Aug 2008.

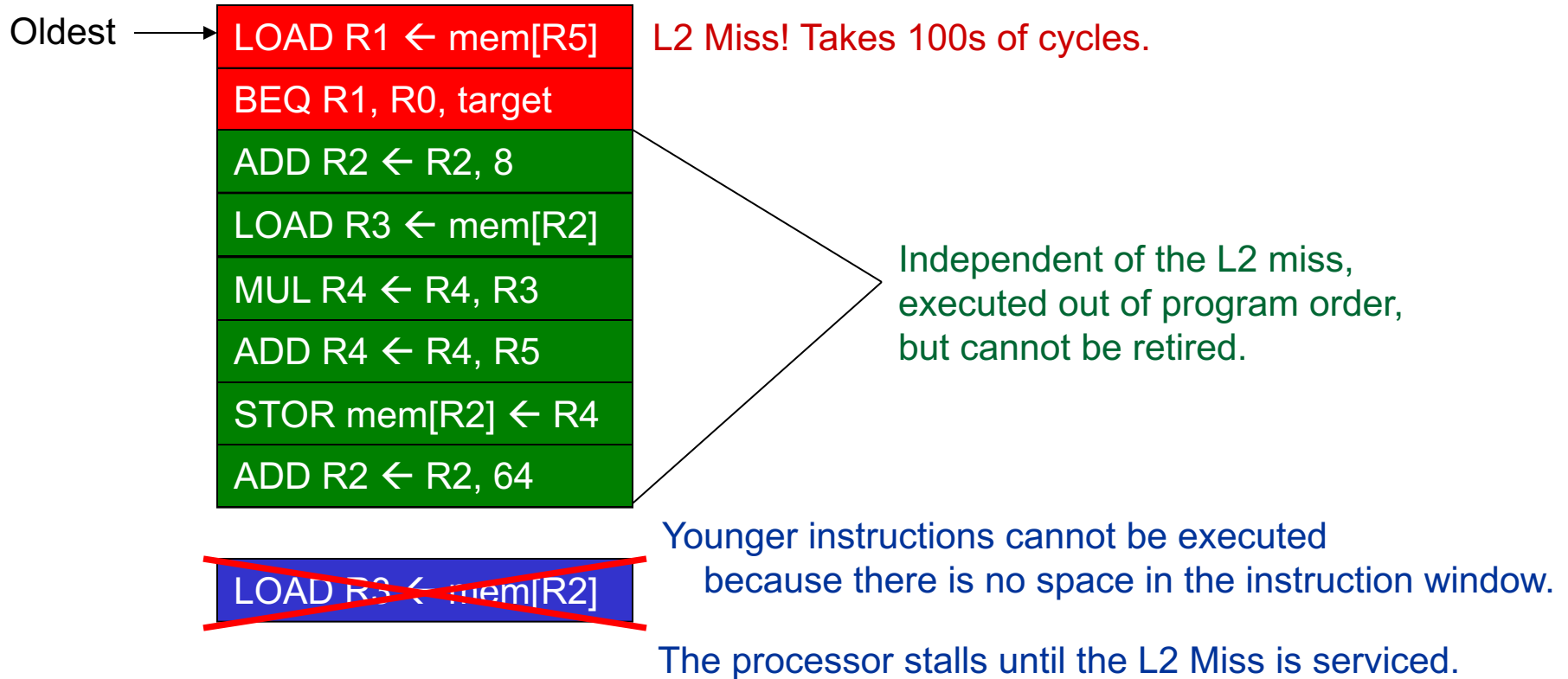


Effective prefetching can both improve performance and reduce hardware cost

An Example Prefetcher: Runahead Execution

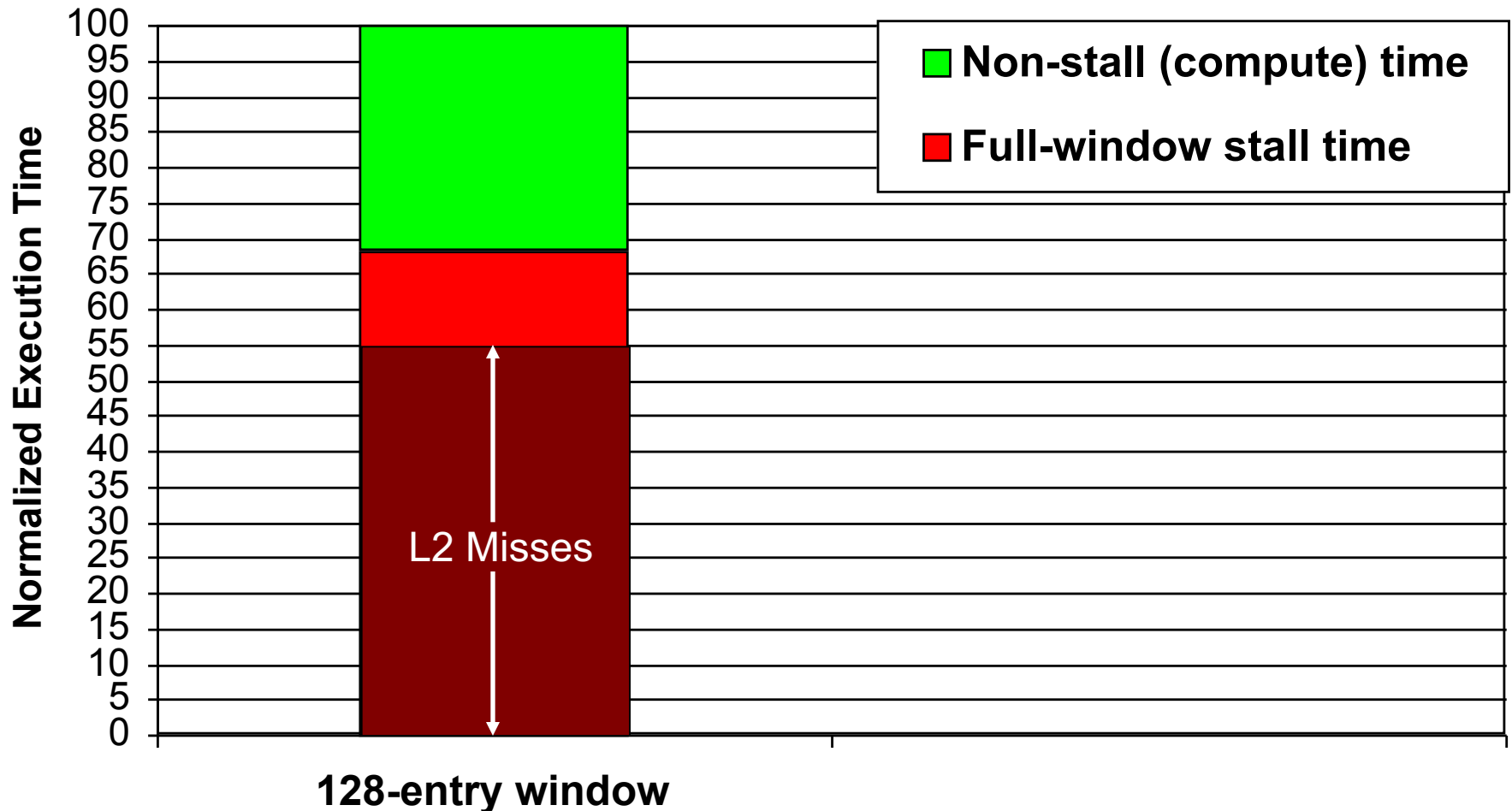
Small Windows: Full-Window Stalls

8-entry instruction window:



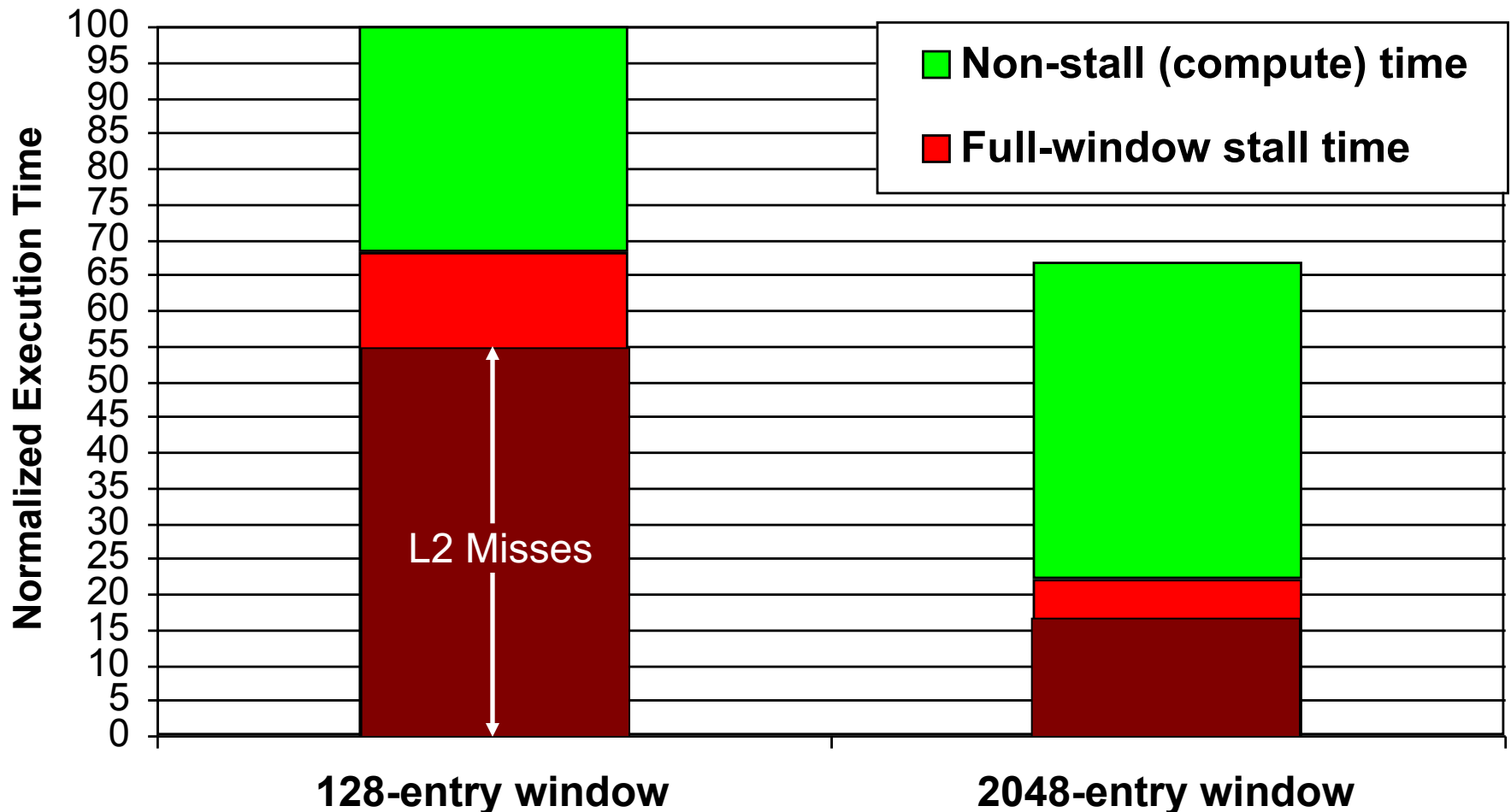
- Long-latency cache misses are responsible for most full-window stalls.

Impact of Long-Latency Cache Misses



512KB L2 cache, 500-cycle DRAM latency, aggressive stream-based prefetcher
Data averaged over 147 memory-intensive benchmarks on a high-end x86 processor model

Impact of Long-Latency Cache Misses



512KB L2 cache, 500-cycle DRAM latency, aggressive stream-based prefetcher
Data averaged over 147 memory-intensive benchmarks on a high-end x86 processor model

The Problem

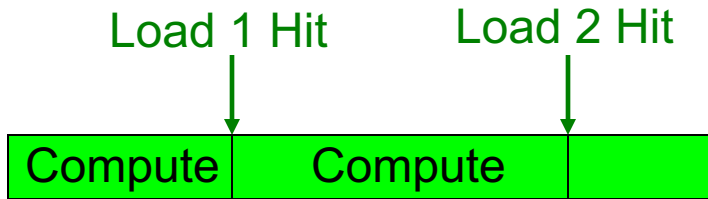
- Out-of-order execution requires large instruction windows to tolerate today's main memory latencies.
- As main memory latency increases, instruction window size should also increase to fully tolerate the memory latency.
- Building a large instruction window is a challenging task if we would like to achieve
 - ❑ Low power/energy consumption (tag matching logic, load/store buffers)
 - ❑ Short cycle time (wakeup/select, regfile, bypass latencies)
 - ❑ Low design and verification complexity

Runahead Execution

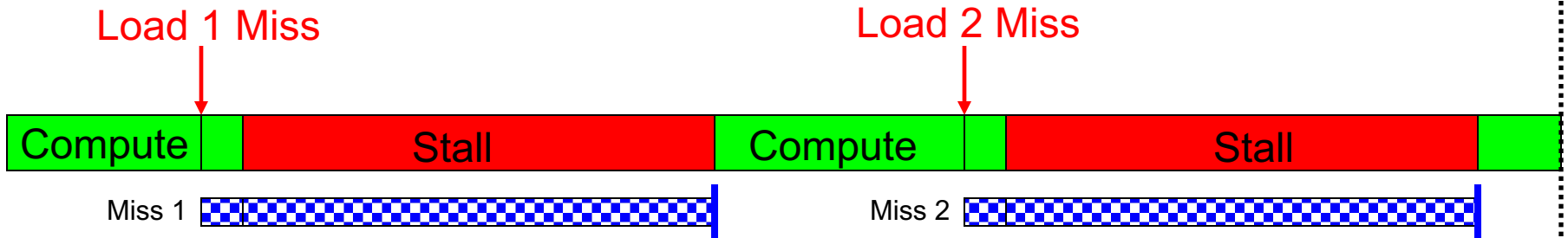
- A technique to obtain the memory-level parallelism benefits of a large instruction window
- When the oldest instruction is a long-latency cache miss:
 - **Checkpoint** architectural state and enter runahead mode
- In runahead mode:
 - **Speculatively pre-execute instructions**
 - **The purpose of pre-execution is to generate prefetches**
 - L2-miss dependent instructions are marked INV and dropped
- When the original miss returns:
 - **Restore checkpoint**, flush pipeline, resume normal execution
- Mutlu et al., “Runahead Execution: An Alternative to Very Large Instruction Windows for Out-of-order Processors,” HPCA 2003.

Runahead Example

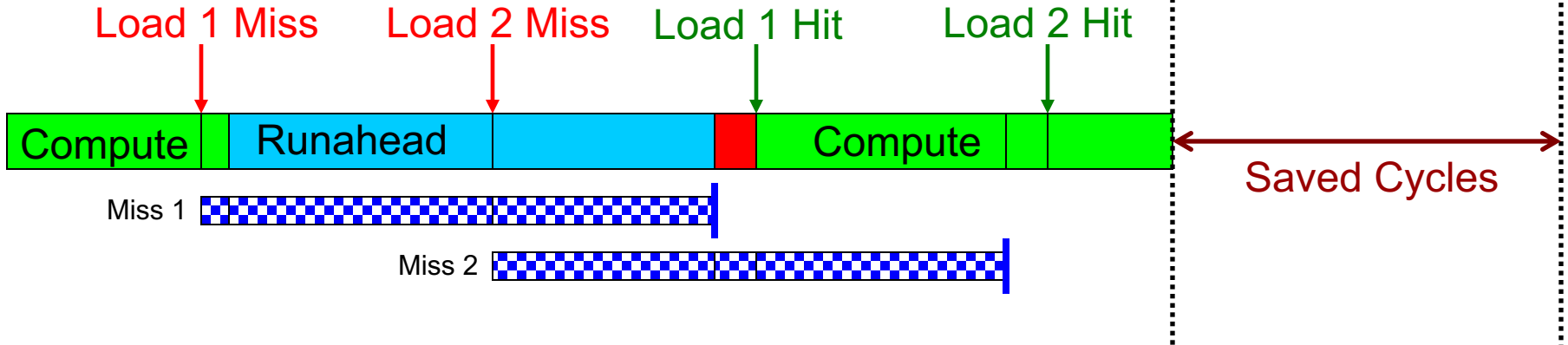
Perfect Caches:



Small Window:



Runahead:



Benefits of Runahead Execution

Instead of stalling during an L2 cache miss:

- Pre-executed loads and stores independent of L2-miss instructions generate **very accurate data prefetches**:
 - For both regular and irregular access patterns
 - **Instructions on the predicted program path are prefetched** into the instruction/trace cache and L2.
 - **Hardware prefetcher and branch predictor tables are trained** using future access information.
-

Runahead Execution Pros and Cons

■ Advantages:

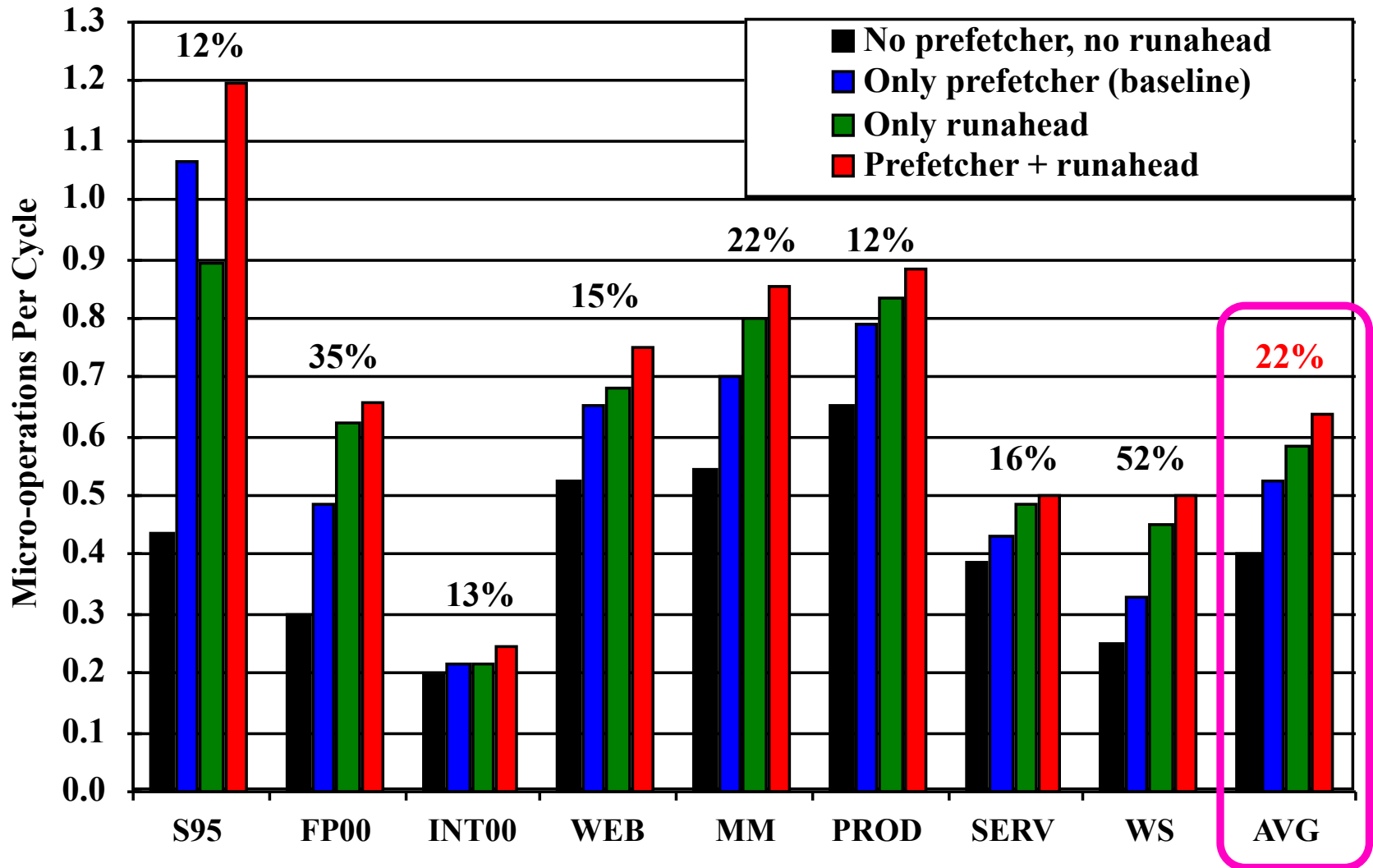
- + Very accurate prefetches for data/instructions (all cache levels)
 - + Follows the program path
- + Simple to implement: most of the hardware is already built in
- + No waste of context: uses the main thread context for prefetching
- + No need to construct a pre-execution thread

■ Disadvantages/Limitations

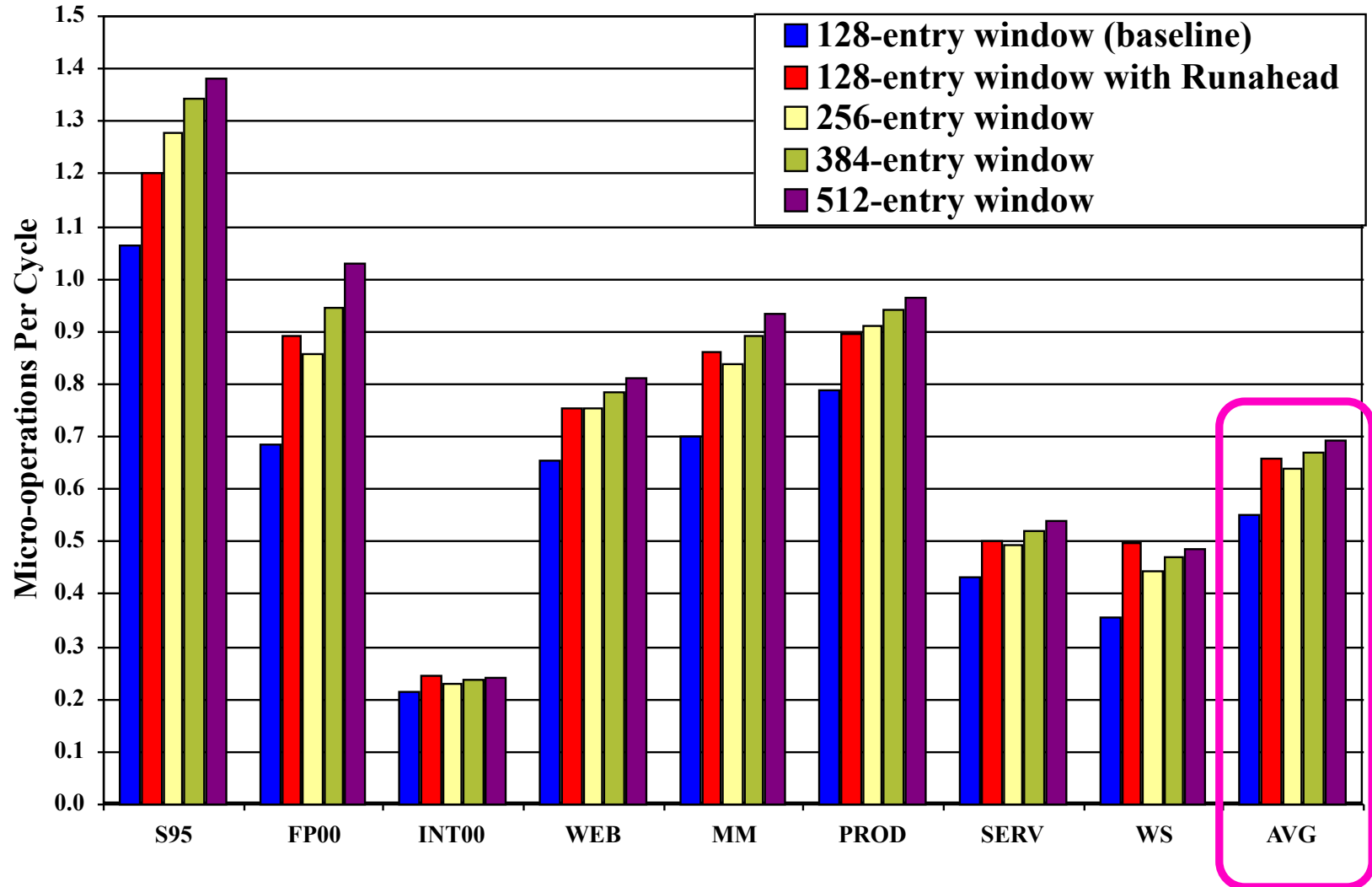
- Extra executed instructions
- Limited by branch prediction accuracy
- Cannot prefetch dependent cache misses
- Effectiveness limited by available “memory-level parallelism” (MLP)
- Prefetch distance (how far ahead to prefetch) limited by memory latency

■ Implemented in Sun ROCK, IBM POWER6, NVIDIA Denver

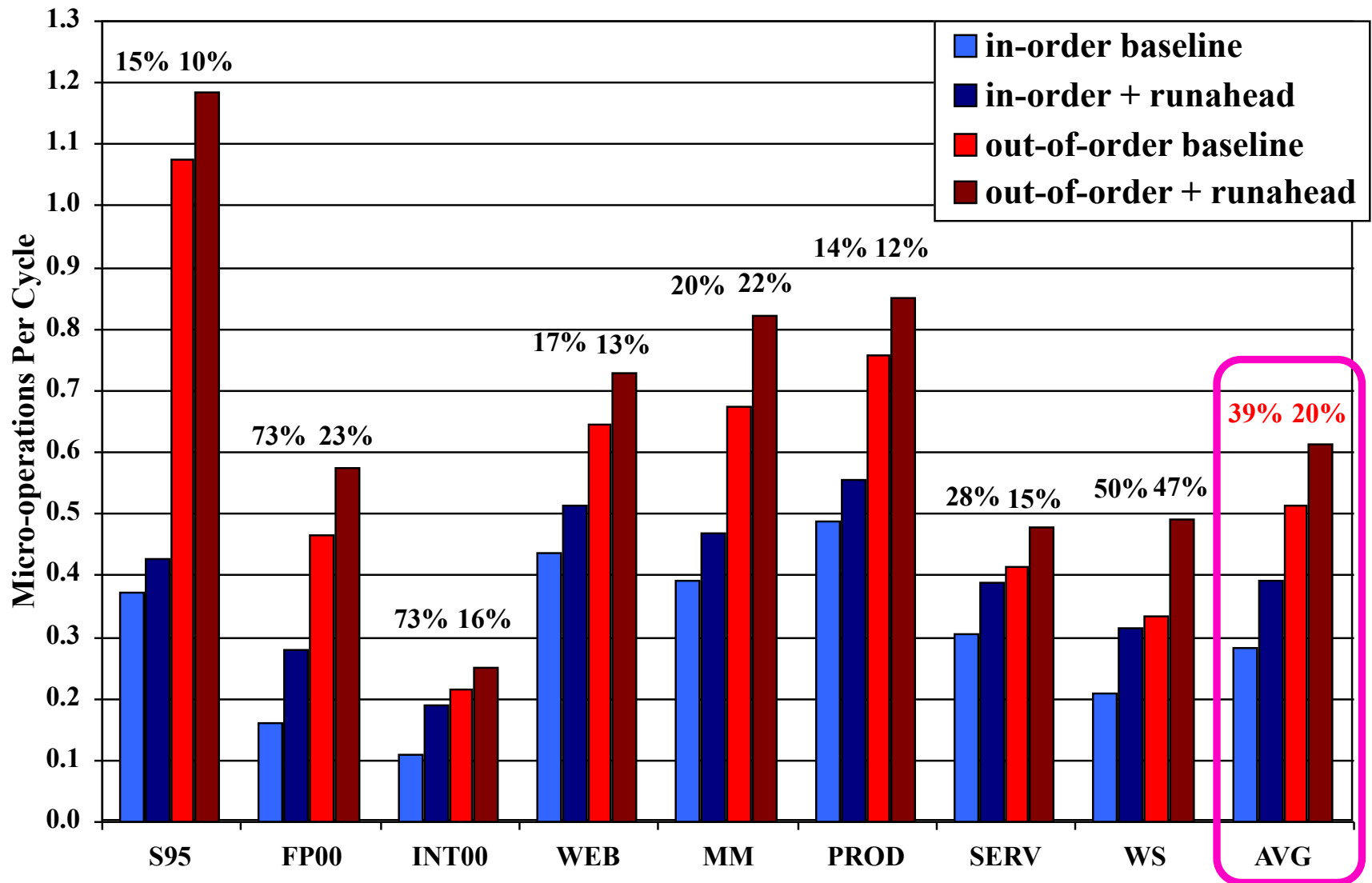
Performance of Runahead Execution



Runahead Execution vs. Large Windows



Runahead on In-order vs. Out-of-order



More on Runahead Execution

- Onur Mutlu, Jared Stark, Chris Wilkerson, and Yale N. Patt,
"Runahead Execution: An Alternative to Very Large Instruction Windows for Out-of-order Processors"

Proceedings of the 9th International Symposium on High-Performance Computer Architecture (HPCA), pages 129-140, Anaheim, CA, February 2003. [Slides \(pdf\)](#)

One of the 15 computer arch. papers of 2003 selected as Top Picks by IEEE Micro. HPCA Test of Time Award (awarded in 2021).

[\[Lecture Slides \(pptx\) \(pdf\)\]](#)

[\[Lecture Video \(1 hr 54 mins\)\]](#)

[\[Retrospective HPCA Test of Time Award Talk Slides \(pptx\) \(pdf\)\]](#)

[\[Retrospective HPCA Test of Time Award Talk Video \(14 minutes\)\]](#)

Runahead Execution: An Alternative to Very Large Instruction Windows for Out-of-order Processors

Onur Mutlu § Jared Stark † Chris Wilkerson ‡ Yale N. Patt §

§ECE Department

The University of Texas at Austin

{onur,patt}@ece.utexas.edu

†Microprocessor Research

Intel Labs

jared.w.stark@intel.com

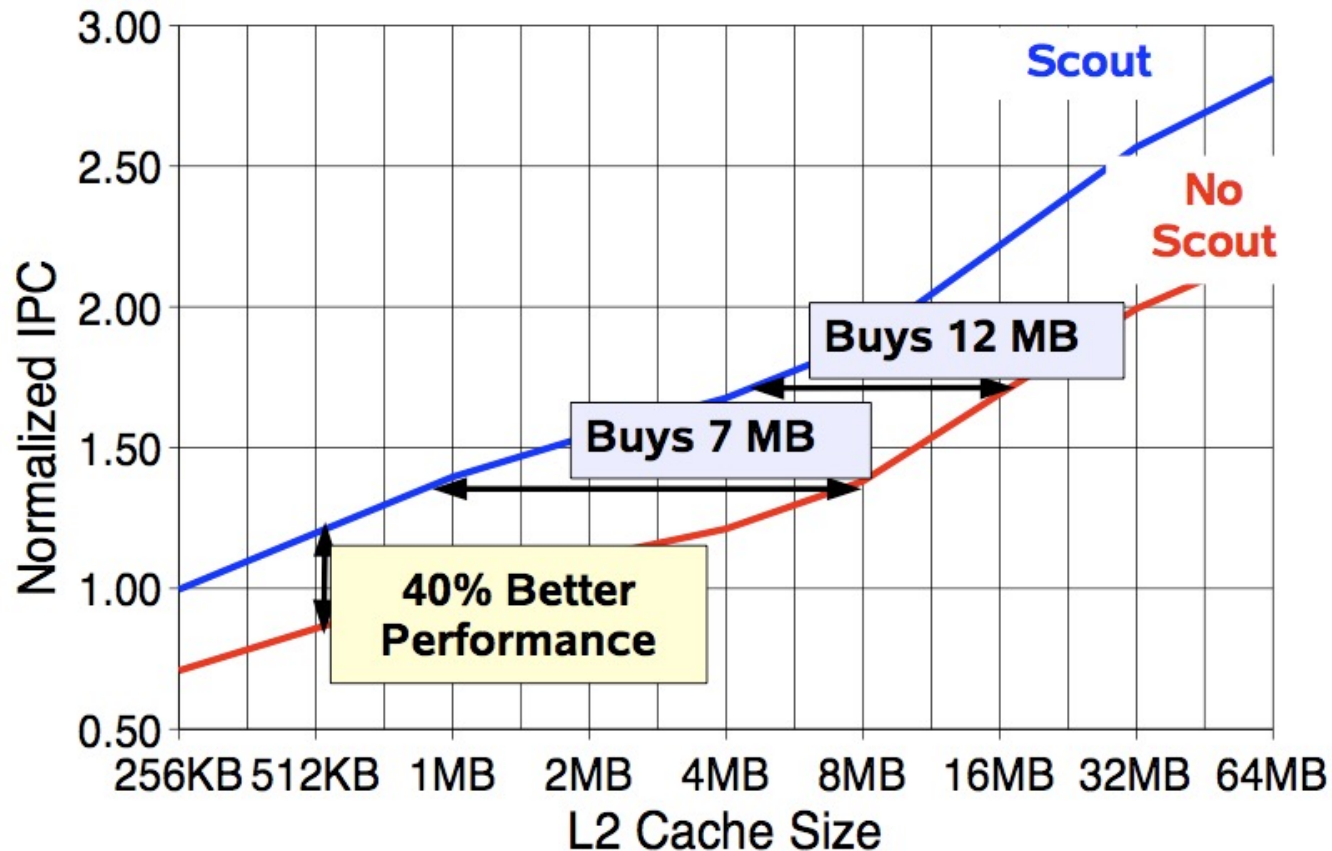
‡Desktop Platforms Group

Intel Corporation

chris.wilkerson@intel.com

Effect of Runahead in Sun ROCK

- Shailender Chaudhry talk, Aug 2008.



HIGH-PERFORMANCE THROUGHPUT COMPUTING

THROUGHPUT COMPUTING, ACHIEVED THROUGH MULTITHREADING AND MULTICORE TECHNOLOGY, CAN LEAD TO PERFORMANCE IMPROVEMENTS THAT ARE 10 TO 30× THOSE OF CONVENTIONAL PROCESSORS AND SYSTEMS. HOWEVER, SUCH SYSTEMS SHOULD ALSO OFFER GOOD SINGLE-THREAD PERFORMANCE. HERE, THE AUTHORS SHOW THAT HARDWARE SCOUTING INCREASES THE PERFORMANCE OF AN ALREADY ROBUST CORE BY UP TO 40 PERCENT FOR COMMERCIAL BENCHMARKS.

More on Runahead in Sun ROCK

Simultaneous Speculative Threading: A Novel Pipeline Architecture Implemented in Sun's ROCK Processor

Shailender Chaudhry, Robert Cypher, Magnus Ekman, Martin Karlsson,
Anders Landin, Sherman Yip, Håkan Zeffer, and Marc Tremblay
Sun Microsystems, Inc.
4180 Network Circle, Mailstop SCA18-211
Santa Clara, CA 95054, USA
{shailender.chaudhry, robert.cypher, magnus.ekman, martin.karlsson,
anders.landin, sherman.yip, haakan.zeffer, marc.tremblay}@sun.com

Runahead Execution in IBM POWER6

Runahead Execution vs. Conventional Data Prefetching in the IBM POWER6 Microprocessor

Harold W. Cain

Priya Nagpurkar

IBM T.J. Watson Research Center

Yorktown Heights, NY

{tcain, pnagpurkar}@us.ibm.com

Cain+, "Runahead Execution vs. Conventional Data Prefetching
in the IBM POWER6 Microprocessor," ISPASS 2010.

Runahead Execution in IBM POWER6

Abstract

After many years of prefetching research, most commercially available systems support only two types of prefetching: software-directed prefetching and hardware-based prefetchers using simple sequential or stride-based prefetching algorithms. More sophisticated prefetching proposals, despite promises of improved performance, have not been adopted by industry. In this paper, we explore the efficacy of both hardware and software prefetching in the context of an IBM POWER6 commercial server. Using a variety of applications that have been compiled with an aggressively optimizing compiler to use software prefetching when appropriate, we perform the first study of a new runahead prefetching feature adopted by the POWER6 design, evaluating it in isolation and in conjunction with a conventional hardware-based sequential stream prefetcher and compiler-inserted software prefetching.

We find that the POWER6 implementation of runahead prefetching is quite effective on many of the memory intensive applications studied; in isolation it improves performance as much as 36% and on average 10%. However, it outperforms the hardware-based stream prefetcher on only two of the benchmarks studied, and in those by a small margin.

When used in conjunction with the conventional prefetching mechanisms, the runahead feature adds an additional 6% on average, and 39% in the best case (GemsFDTD).

DENVER: NVIDIA'S FIRST 64-BIT ARM PROCESSOR

NVIDIA'S FIRST 64-BIT ARM PROCESSOR, CODE-NAMED DENVER, LEVERAGES A HOST OF NEW TECHNOLOGIES, SUCH AS DYNAMIC CODE OPTIMIZATION, TO ENABLE HIGH-PERFORMANCE MOBILE COMPUTING. IMPLEMENTED IN A 28-NM PROCESS, THE DENVER CPU CAN ATTAIN CLOCK SPEEDS OF UP TO 2.5 GHZ. THIS ARTICLE OUTLINES THE DENVER ARCHITECTURE, DESCRIBES ITS TECHNOLOGICAL INNOVATIONS, AND PROVIDES RELEVANT COMPARISONS AGAINST COMPETING MOBILE PROCESSORS.

Boggs+, "[Denver: NVIDIA's First 64-Bit ARM Processor](#)," IEEE Micro 2015.

Runahead Execution in NVIDIA Denver

Reducing the effects of long cache-miss penalties has been a major focus of the micro-architecture, using techniques like prefetching and run-ahead. An aggressive hardware prefetcher implementation detects L2 cache requests and tracks up to 32 streams, each with complex stride patterns.

Run-ahead uses the idle time that a CPU spends waiting on a long latency operation to discover cache and DTLB misses further down the instruction stream and generates prefetch requests for these misses.¹ These prefetch requests warm up the data cache and DTLB well before the actual execution of

the instructions that require the data. Run-ahead complements the hardware prefetcher because it's better at prefetching nonstrided streams, and it trains the hardware prefetcher faster than normal execution to yield a combined benefit of 13 percent on SPECint2000 and up to 60 percent on SPECfp2000.

The core includes a hardware prefetch unit that Boggs describes as “aggressive” in preloading the data cache but less aggressive in preloading the instruction cache. It also implements a “run-ahead” feature that continues to execute microcode speculatively after a data-cache miss; this execution can trigger additional cache misses that resolve in the shadow of the first miss. Once the data from the original miss returns, the results of this speculative execution are discarded and execution restarts with the bundle containing the original miss, but run-ahead can preload subsequent data into the cache, thus avoiding a string of time-wasting cache misses. These and other features help Denver outscore Cortex-A15 by more than 2.6x on a memory-read test even when both use the same SoC framework (Tegra K1).

Boggs+, “Denver: NVIDIA’s First 64-Bit ARM Processor,” IEEE Micro 2015.

Gwennap, “NVIDIA’s First CPU is a Winner,” MPR 2014.

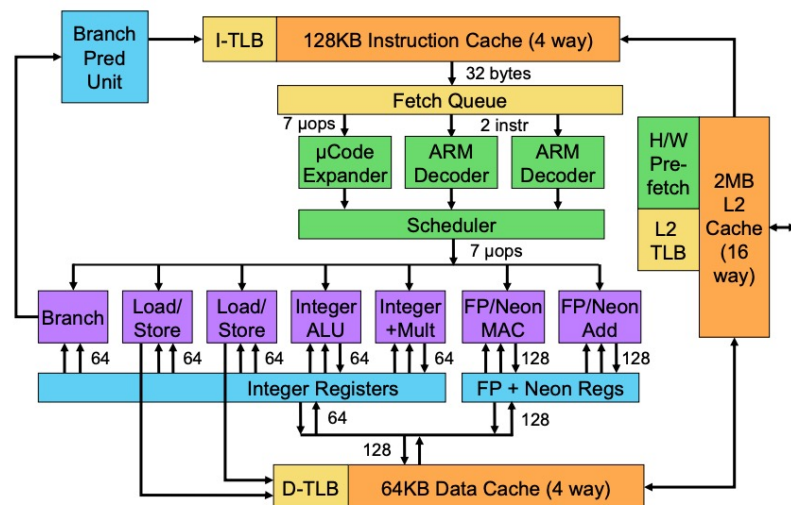


Figure 3. Denver CPU microarchitecture. This design combines a fairly

Runahead Enhancements

Runahead Enhancements

- Mutlu et al., “Techniques for Efficient Processing in Runahead Execution Engines,” ISCA 2005, IEEE Micro Top Picks 2006.
- Mutlu et al., “Address-Value Delta (AVD) Prediction,” MICRO 2005.
- Armstrong et al., “Wrong Path Events,” MICRO 2004.
- Mutlu et al., “An Analysis of the Performance Impact of Wrong-Path Memory References on Out-of-Order and Runahead Execution Processors,” IEEE TC 2005.

Limitations of the Baseline Runahead Mechanism

■ Energy Inefficiency

- ❑ A large number of instructions are speculatively executed
- ❑ **Efficient Runahead Execution** [ISCA'05, IEEE Micro Top Picks'06]

■ Ineffectiveness for pointer-intensive applications

- ❑ Runahead cannot parallelize dependent L2 cache misses
- ❑ **Address-Value Delta (AVD) Prediction** [MICRO'05]

■ Irresolvable branch mispredictions in runahead mode

- ❑ Cannot recover from a mispredicted L2-miss dependent branch
 - ❑ **Wrong Path Events** [MICRO'04]
 - ❑ **Wrong Path Memory Reference Analysis** [IEEE TC'05]
-

More on Efficient Runahead Execution

- Onur Mutlu, Hyesoon Kim, and Yale N. Patt,
"Techniques for Efficient Processing in Runahead Execution Engines"
Proceedings of the 32nd International Symposium on Computer Architecture (ISCA), pages 370-381, Madison, WI, June 2005. Slides (ppt) Slides (pdf)
One of the 13 computer architecture papers of 2005 selected as Top Picks by IEEE Micro.

Techniques for Efficient Processing in Runahead Execution Engines

Onur Mutlu Hyesoon Kim Yale N. Patt

Department of Electrical and Computer Engineering
University of Texas at Austin
{onur,hyesoon,patt}@ece.utexas.edu

More on Efficient Runahead Execution

- Onur Mutlu, Hyesoon Kim, and Yale N. Patt,
["Efficient Runahead Execution: Power-Efficient Memory Latency Tolerance"](#)

IEEE Micro, Special Issue: Micro's Top Picks from Microarchitecture Conferences (MICRO TOP PICKS), Vol. 26, No. 1, pages 10-20, January/February 2006.

EFFICIENT RUNAHEAD EXECUTION: POWER-EFFICIENT MEMORY LATENCY TOLERANCE

More Effective Runahead Execution

- Onur Mutlu, Hyesoon Kim, and Yale N. Patt,
"Address-Value Delta (AVD) Prediction: Increasing the Effectiveness of Runahead Execution by Exploiting Regular Memory Allocation Patterns"
Proceedings of the 38th International Symposium on Microarchitecture (MICRO),
pages 233-244, Barcelona, Spain, November 2005. [Slides \(ppt\)](#) [Slides \(pdf\)](#)
One of the five papers nominated for the Best Paper Award by the Program Committee.

Address-Value Delta (AVD) Prediction: Increasing the Effectiveness of Runahead Execution by Exploiting Regular Memory Allocation Patterns

Onur Mutlu Hyesoon Kim Yale N. Patt

Department of Electrical and Computer Engineering
University of Texas at Austin
{onur,hyesoon,patt}@ece.utexas.edu

More on Efficient Runahead Execution

- Onur Mutlu, Hyesoon Kim, and Yale N. Patt,
"Address-Value Delta (AVD) Prediction: A Hardware Technique for Efficiently Parallelizing Dependent Cache Misses"
IEEE Transactions on Computers (TC), Vol. 55, No. 12, pages 1491-1508, December 2006.

Address-Value Delta (AVD) Prediction: A Hardware Technique for Efficiently Parallelizing Dependent Cache Misses

Onur Mutlu, *Member, IEEE*, Hyesoon Kim, *Student Member, IEEE*, and
Yale N. Patt, *Fellow, IEEE*

Looking to the Past

At the Time... Early 2000s...

- Large focus on increasing the size of the window...
 - And, designing bigger, more complicated machines
- Runahead was a different way of thinking
 - Keep the OoO core simple and small
 - At the expense of some benefits (e.g., non-memory-related)
 - Use aggressive “automatic speculative execution” solely for prefetching
 - Synergistic with prefetching and branch prediction methods
- A lot of interesting and innovative ideas ensued...

Important Precedent [Dundas & Mudge, ICS 1997]

Improving Data Cache Performance by Pre-executing Instructions Under a Cache Miss

James Dundas and Trevor Mudge

Department of Electrical Engineering and Computer Science

The University of Michigan

Ann Arbor, Michigan 48109-2122

{dundas, tnm}@eecs.umich.edu

Abstract

In this paper we propose and evaluate a technique that improves first level data cache performance by pre-executing future instructions under a data cache miss. We show that these pre-executed instructions can generate highly accurate data prefetches, particularly when the first level cache is small. The technique is referred to as *runahead* processing. The hardware required to implement runahead is modest, because, when a miss occurs, it makes use of an otherwise idle resource, the execution logic. The principal hardware cost is an extra register file. To measure the impact of runahead, we simulated a processor executing five integer Spec95 benchmarks. Our results show that runahead was able to significantly reduce data cache CPI for four of the five benchmarks. We also compared runahead to a simple form of prefetching, sequential prefetching, which would seem to be suitable for scientific benchmarks. We confirm this by enlarging the scope of our experiments to include a scientific benchmark. However, we show that runahead was also able to outperform sequential prefetching on the scientific benchmark. We also conduct studies that demonstrate that runahead can generate many useful prefetches for lines that show little spatial locality with the misses that initiate runahead episodes. Finally, we discuss some further enhancements of our baseline runahead prefetching scheme.

are allocated by the software. This hybrid hardware-software technique was presented in [8]. Their instruction stride table (IST) selectively generates cache miss initiated prefetches for accesses chosen beforehand by the compiler. This resulted in multiprocessor performance for scientific benchmarks comparable in some cases to software prefetching, with an instruction stride table as small as 4 entries. The IST concept was subsequently combined with the prefetch predicates of [2] in [9]. Another hardware prefetching scheme that avoids the need for significant amounts of hardware is the “wrong path” prefetching described in [10]. This actually prefetches instructions from the not-taken path, in the expectation that they will be executed during a later iteration.

Most prefetching techniques, software- or hardware-based, tend to perform poorly on an important class of applications having recursive data structures such as linked-lists. A software technique that overcomes this limitation was presented recently in [11], in which software prefetches were inserted at subroutine call sites that passed pointers as arguments. Another pointer-based approach was described in [12]. This approach uses pointers stored within the data structures to generate software prefetches.

The runahead prefetching approach presented in this paper is a hardware approach, that requires only a modest amount of hardware, because, when a miss occurs, it makes use of an otherwise

An Inspiration [Glew, ASPLOS-WACI 1998]

MLP yes! ILP no!

Memory Level Parallelism, or why I no longer care about Instruction Level Parallelism

Andrew Glew

Intel Microcomputer Research Labs and University of Wisconsin, Madison

Problem Description: It should be well known that processors are outstripping memory performance: specifically that memory latencies are not improving as fast as processor cycle time or IPC or memory bandwidth.

Thought experiment: imagine that a cache miss takes 10000 cycles to execute. For such a processor instruction level parallelism is useless, because most of the time is spent waiting for memory. Branch prediction is also less effective, since most branches can be determined with data already in registers or in the cache; branch prediction only helps for branches which depend on outstanding cache misses.

At the same time, pressures for reduced power consumption mount.

Given such trends, some computer architects in industry (although not Intel EPIC) are talking seriously about retreating from out-of-order superscalar processor architecture, and instead building simpler, faster, dumber, 1-wide in-order processors with high degrees of speculation. Sometimes this is proposed in combination with multiprocessing and multithreading: tolerate long memory latencies by switching to other processes or threads.

I propose something different: build narrow fast machines but use intelligent logic inside the CPU to increase the number of outstanding cache misses that can be generated from a single program.

Solution: First, change the mindset: MLP, Memory Level Parallelism, is what matters, not ILP, Instruction Level Parallelism.

By MLP I mean simply the number of outstanding cache misses that can be generated (by a single thread, task, or program) and executed in an overlapped manner. It does not matter what sort of execution engine generates the multiple outstanding cache misses. An out-of-order superscalar ILP CPU may generate multiple outstanding cache misses, but 1-wide processors can be just as effective.

Change the metrics: total execution time remains the overall goal, but instead of reporting IPC as an approximation to this, we must report MLP. Limit studies should be in terms of total number of non-overlapped cache misses on critical path.

Now do the research: Many present-day hot topics in computer architecture help ILP, but do not help MLP. As mentioned above, predicting branch directions for branches that can be determined from data already in the cache or in registers does not help MLP for extremely long latencies. Similarly, prefetching of data cache misses for array processing codes does not help MLP – it just moves it around.

Instead, investigate microarchitectures that help MLP:

- (0) Trivial case – explicit multithreading, like SMT.
- (1) Slightly less trivial case – implicitly multithread single programs, either by compiler software on an MT machine, or by a hybrid, such as Wisconsin Multiscalar, or entirely in hardware, as in Intel's Dynamic Multi-Threading.
- (2) Build 1-wide processors that are as fast as possible: use circuit tricks, as well as logic tricks such as redundant encoding for numeric computation and memory addressing.
- (3) Allow the hardware dynamic scheduling mechanisms to use sequential algorithms implemented by this narrow, fast, processor, rather than limiting it to parallel algorithms implementable in associative logic.
- (4) Build very large instruction windows allowing speculation tens of thousands of instructions ahead. Avoid circuit speed issues by caching the instruction window. Remove small arbitrary limits on the number of cache misses outstanding allowed.
- (5) Further reduce the cost of very large instruction windows by throwing away anything that can be recomputed based on data in registers or cache.
- (6) Don't stall speculation because the oldest instruction in the machine is a cache miss. Let the front of the machine continue executing branches, forgetting data dependent on cache misses.
- (7) Parallelize linked data structure traversals by building skip lists in hardware – converting sequential data structures into parallel ones. Store these extra skip pointers in main memory.

Call such a processor microarchitecture a “super-non-blocking” microarchitecture.

Justification: The processor/memory trend is well known. Theoretically optimal cache studies show only limited headroom. Barring a revolution in memory technology, the Memory Wall is real, and getting closer. Multithreading and multiprocessing have some hope of tolerating memory latency, but only if there are parallel workloads. If single thread performance is still an issue, the only potentially MLP enhancing technologies are what I describe here, or data value prediction – and data value prediction seems to only do well for stuff that fits in the cache.

“Super-non-blocking” processors extends dynamic, out-of-order, execution to maximize MLP, but simplifies it by discarding superscalar ILP as unnecessary.

Looking to the Future

A Look into the Future...

- Microarchitecture is still critically important
 - And, fun...
 - And, impactful...
- Runahead is a great example of harmonious industry-academia collaboration
- Fundamental problems will remain fundamental
 - And will require fundamental (and creative) solutions

Citation for the Test of Time Award

- Runahead Execution is a pioneering paper that opened up new avenues in dynamic prefetching.
- The basic idea of runahead execution effectively increases the instruction window very significantly, without having to increase physical resource size (e.g. the issue queue).
- This seminal paper spawned off a new area of ILP-enhancing microarchitecture research.
- This work has had strong industry impact as evidenced by IBM's POWER6 - Load Lookahead, NVIDIA Denver, and Sun ROCK's hardware scouting.

Suggestion to Researchers: Principle: Passion

Follow Your Passion
**(Do not get derailed
by naysayers)**

Suggestion to Researchers: Principle: Resilience

Be Resilient

Principle: Learning and Scholarship

Focus on
learning and scholarship

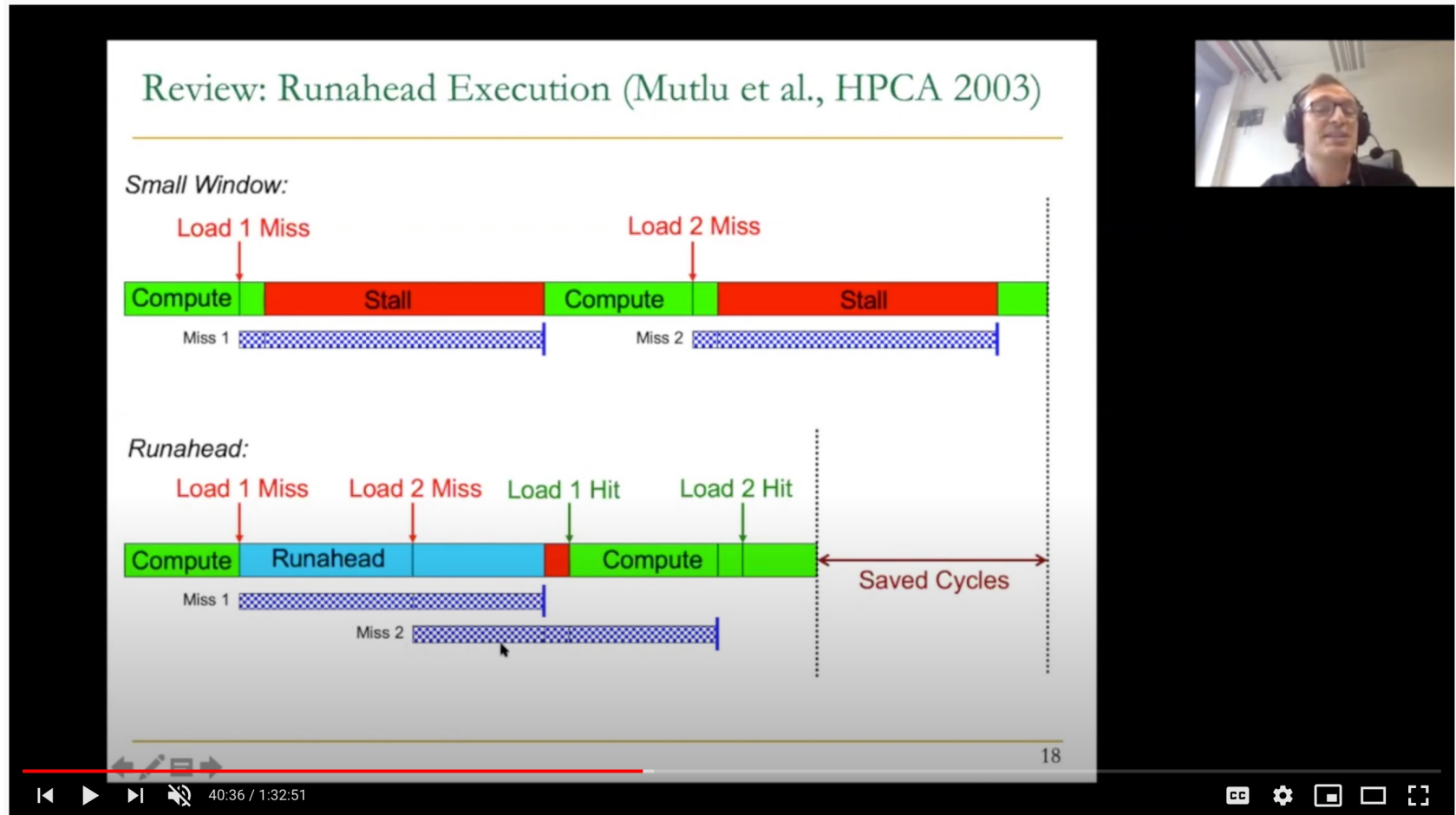
Principle: Learning and Scholarship

The quality of your work
defines your impact

More on Runahead Execution

- Lecture video from Fall 2020, Computer Architecture:
 - https://www.youtube.com/watch?v=zPewo6IaJ_8
- Lecture video from Fall 2017, Computer Architecture:
 - <https://www.youtube.com/watch?v=Kj3relihGF4>
- Onur Mutlu,
"Efficient Runahead Execution Processors"
Ph.D. Dissertation, HPS Technical Report, TR-HPS-2006-007, July 2006. [Slides \(ppt\)](#)
Nominated for the ACM Doctoral Dissertation Award by the University of Texas at Austin.

More on Runahead Execution (I)



Computer Architecture - Lecture 19a: Execution-Based Prefetching (ETH Zürich, Fall 2020)

395 views • Nov 29, 2020

14 0 SHARE SAVE ...



Onur Mutlu Lectures
16.5K subscribers

ANALYTICS

EDIT VIDEO

https://www.youtube.com/watch?v=zPewo6laJ_8&list=PL5Q2soXY2Zi9xidylgBxUz7xRPS-wisBN&index=34

More on Runahead Execution (II)

Runahead Execution in NVIDIA Denver

Reducing the effects of long cache-miss penalties has been a major focus of the micro-architecture, using techniques like prefetching and run-ahead. An aggressive hardware prefetcher implementation detects L2 cache requests and tracks up to 32 streams, each with complex stride patterns.

Run-ahead uses the idle time that a CPU spends waiting on a long latency operation to discover cache and DTLB misses further down the instruction stream and generates prefetch requests for these misses.¹ These prefetch requests warm up the data cache and DTLB well before the actual execution of the instructions that require the data. Run-ahead complements the hardware prefetcher because it's better at prefetching nonstrided streams, and it trains the hardware prefetcher faster than normal execution to yield a combined benefit of 13 percent on SPECint2000 and up to 60 percent on SPECfp2000.

The core includes a hardware prefetch unit that Boggs describes as "aggressive" in preloading the data cache but less aggressive in preloading the instruction cache. It also implements a "run-ahead" feature that continues to execute microcode speculatively after a data-cache miss; this execution can trigger additional cache misses that resolve in the shadow of the first miss. Once the data from the original miss returns, the results of this speculative execution are discarded and execution restarts with the bundle containing the original miss, but run-ahead can preload subsequent data into the cache, thus avoiding a string of time-wasting cache misses. These and other features help Denver outscore Cortex-A15 by more than 2.6x on a memory-read test even when both use the same SoC framework (Tegra K1).

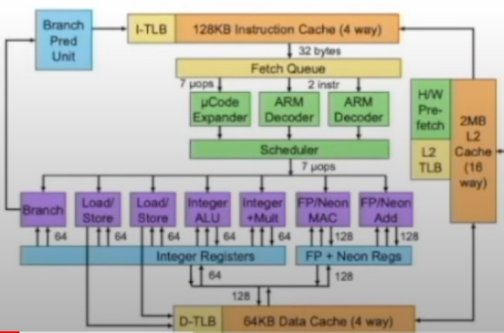


Figure 3. Denver CPU microarchitecture. This design combines a fairly

Boggs+, "Denver: NVIDIA's First 64-Bit ARM Processor," IEEE Micro 2015.

Gwennap, "NVIDIA's First CPU is a Winner," MPR 2014.



Onur Mutlu - Runahead Execution: A Short Retrospective (HPCA Test of Time Award Talk @ HPCA 2021)

1,162 views • Premiered Mar 6, 2021

50 0 SHARE SAVE ...



Onur Mutlu Lectures
16.5K subscribers

ANALYTICS

EDIT VIDEO

More Recommended Material on Prefetching

Lectures on Prefetching (I)

X86 PREFETCH Instruction

PREFETCHh—Prefetch Data Into Caches

Opcode	Instruction	64-Bit Mode	Compat/ Leg Mode	Description
OF 18 /1	PREFETCHT0 m8	Valid	Valid	Move data from m8 closer to the processor using T0 hint.
OF 18 /2	PREFETCHT1 m8	Valid	Valid	Move data from m8 closer to the processor using T1 hint.
OF 18 /3	PREFETCHT2 m8	Valid	Valid	Move data from m8 closer to the processor using T2 hint.
OF 18 /0	PREFETCHNTA m8	Valid	Valid	Move data from m8 closer to the processor using NTA hint.

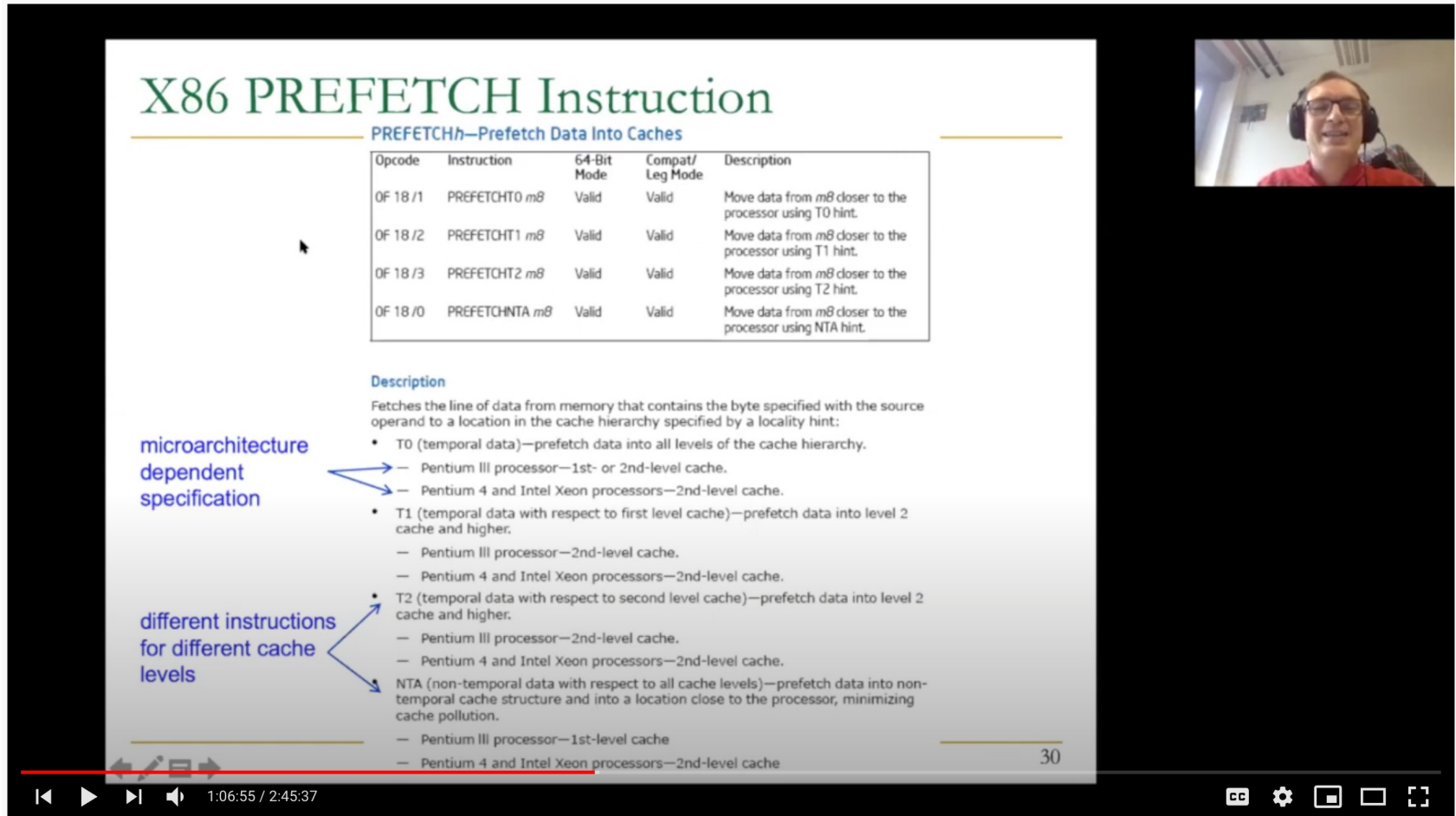
Description

Fetches the line of data from memory that contains the byte specified with the source operand to a location in the cache hierarchy specified by a locality hint:

- T0 (temporal data)—prefetch data into all levels of the cache hierarchy.
 - Pentium III processor—1st- or 2nd-level cache.
 - Pentium 4 and Intel Xeon processors—2nd-level cache.
- T1 (temporal data with respect to first level cache)—prefetch data into level 2 cache and higher.
 - Pentium III processor—2nd-level cache.
 - Pentium 4 and Intel Xeon processors—2nd-level cache.
- T2 (temporal data with respect to second level cache)—prefetch data into level 2 cache and higher.
 - Pentium III processor—2nd-level cache.
 - Pentium 4 and Intel Xeon processors—2nd-level cache.
- NTA (non-temporal data with respect to all cache levels)—prefetch data into non-temporal cache structure and into a location close to the processor, minimizing cache pollution.
 - Pentium III processor—1st-level cache
 - Pentium 4 and Intel Xeon processors—2nd-level cache

microarchitecture dependent specification

different instructions for different cache levels



30

Computer Architecture - Lecture 18: Prefetching (ETH Zürich, Fall 2020)

1,203 views • Nov 29, 2020

👍 26 💬 0 ➦ SHARE ⚙️ SAVE ⋮



Onur Mutlu Lectures
16.5K subscribers

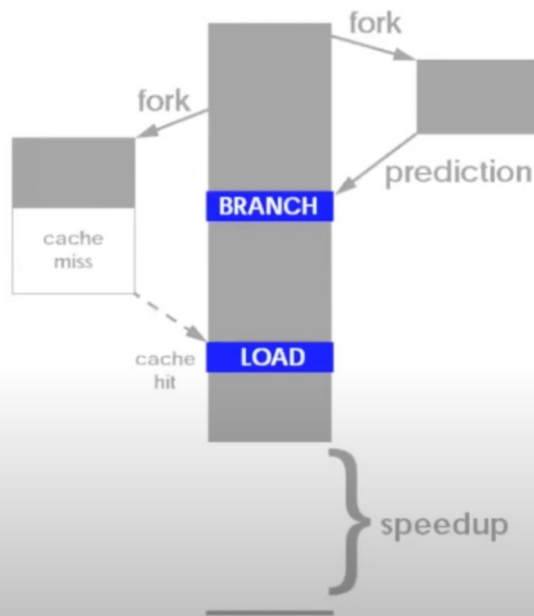
ANALYTICS

EDIT VIDEO

<https://www.youtube.com/watch?v=xZmDyj0g3Pw&list=PL5Q2soXY2Zi9xidylgBxUz7xRPS-wisBN&index=33>

Lectures on Prefetching (II)

Thread-Based Pre-Execution



- Dubois and Song, “**Assisted Execution**,” USC Tech Report 1998.
- Chappell et al., “**Simultaneous Subordinate Microthreading (SSMT)**,” ISCA 1999.
- Zilles and Sohi, “**Execution-based Prediction Using Speculative Slices**”, ISCA 2001.



7

Computer Architecture - Lecture 19a: Execution-Based Prefetching (ETH Zürich, Fall 2020)

424 views • Nov 29, 2020

16 0 SHARE SAVE ...



Onur Mutlu Lectures
16.7K subscribers

ANALYTICS

EDIT VIDEO

Lectures on Prefetching (III)

Runahead Execution in NVIDIA Denver

Reducing the effects of long cache-miss penalties has been a major focus of the microarchitecture, using techniques like prefetching and run-ahead. An aggressive hardware prefetcher implementation detects L2 cache requests and tracks up to 32 streams, each with complex stride patterns.

Run-ahead uses the idle time that a CPU spends waiting on a long latency operation to discover cache and DTLB misses further down the instruction stream and generates prefetch requests for these misses.¹ These prefetch requests warm up the data cache and DTLB well before the actual execution of the instructions that require the data. Run-ahead complements the hardware prefetcher because it's better at prefetching nonstrided streams, and it trains the hardware prefetcher faster than normal execution to yield a combined benefit of 13 percent on SPECint2000 and up to 60 percent on SPECfp2000.

Boggs+, "Denver: NVIDIA's First 64-Bit ARM Processor," IEEE Micro 2015.

Gwennap, "NVIDIA's First CPU is a Winner," MPR 2014.

The core includes a hardware prefetch unit that Boggs describes as “aggressive” in preloading the data cache but less aggressive in preloading the instruction cache. It also implements a “run-ahead” feature that continues to execute microcode speculatively after a data-cache miss; this execution can trigger additional cache misses that resolve in the shadow of the first miss. Once the data from the original miss returns, the results of this speculative execution are discarded and execution restarts with the bundle containing the original miss, but run-ahead can preload subsequent data into the cache, thus avoiding a string of time-wasting cache misses. These and other features help Denver outscore Cortex-A15 by more than 2.6x on a memory-read test even when both use the same SoC framework (Tegra K1).

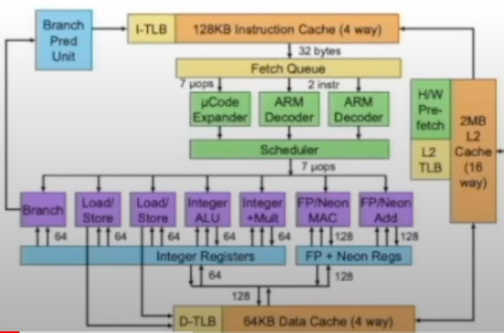
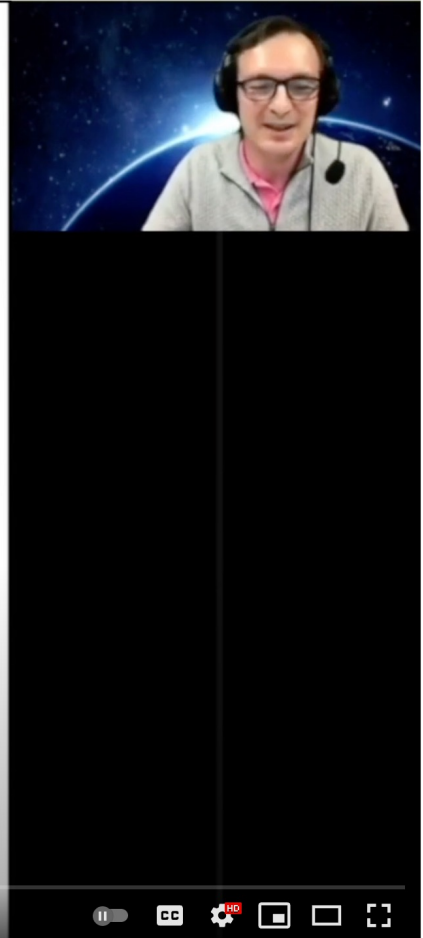


Figure 3. Denver CPU microarchitecture. This design combines a fairly



Onur Mutlu - Runahead Execution: A Short Retrospective (HPCA Test of Time Award Talk @ HPCA 2021)

1,162 views • Premiered Mar 6, 2021



Onur Mutlu Lectures
16.5K subscribers

👍 50 🗨️ 0 ➡️ SHARE ≡+ SAVE ...

ANALYTICS

[EDIT VIDEO](#)

Lectures on Prefetching (IV)

Software Prefetching (II)

```
for (i=0; i<N; i++) {  
    __prefetch(a[i+8]);  
    __prefetch(b[i+8]);  
    sum += a[i]*b[i];  
}
```

```
while (p) {  
    __prefetch(p->next);  
    work(p->data);  
    p = p->next;  
}
```

```
while (p) {  
    __prefetch(p->next->next->next);  
    work(p->data);  
    p = p->next;  
}
```

Which one is better?

- Can work for very regular array-based access patterns. Issues:
 - Prefetch instructions take up processing/execution bandwidth
 - How early to prefetch? Determining this is difficult
 - Prefetch distance depends on hardware implementation (memory latency, cache size, time between loop iterations) → portability?
 - Going too far back in code reduces accuracy (branches in between)
 - Need “special” prefetch instructions in ISA?
 - Alpha load into register 31 treated as prefetch (r31==0)
 - PowerPC *dcbt* (data cache block touch) instruction
 - Not easy to do for pointer-based data structures

40

Lecture 25: Prefetching - Carnegie Mellon - Computer Architecture 2015 - Onur Mutlu

5,216 views • Apr 3, 2015

39 0 SHARE SAVE ...



Carnegie Mellon Computer Architecture
23.3K subscribers

SUBSCRIBED



<https://www.youtube.com/watch?v=ibPL7T9iEwY&list=PL5PHm2jkkXmi5Cxxl7b3JCL1TWybTDtKq&index=29>

Lectures on Prefetching (V)

Address Correlation Based Prefetching (II)

The diagram illustrates the Address Correlation Based Prefetching (II) mechanism. It shows a 'Cache Block Addr' pointing to a table with 'Cache Block Addr (tag)' and a 'Prefetch' table. The 'Prefetch' table has columns for 'Candidate 1' and 'Confidence'. The 'Confidence' column contains a series of asterisks. To the right, there is a table with 'Candidate N' and 'Confidence'.

- Idea: Record the likely-next addresses (B, C, D) after seeing an address A
 - Next time A is accessed, prefetch B, C, D
 - A is said to be correlated with B, C, D
- Prefetch up to N next addresses to increase *coverage*
- Prefetch accuracy can be improved by using multiple addresses as key for the next address: (A, B) → (C)
(A,B) correlated with C
- Joseph and Grunwald, "Prefetching using Markov Predictors," ISCA 1997.

10

Lecture 26. More Prefetching and Emerging Memory Technologies - CMU - Comp. Arch. 2015 - Onur Mutlu

3,642 views • Apr 6, 2015

26 0 SHARE SAVE ...



Carnegie Mellon Computer Architecture
23.3K subscribers

SUBSCRIBED



Lectures on Prefetching

■ Computer Architecture, Fall 2020, Lecture 18

- ❑ Prefetching (ETH, Fall 2020)
- ❑ <https://www.youtube.com/watch?v=xZmDyj0g3Pw&list=PL5Q2soXY2Zi9xidyIgBxUz7xRPS-wisBN&index=33>

■ Computer Architecture, Fall 2020, Lecture 19a

- ❑ Execution-Based Prefetching (ETH, Fall 2020)
- ❑ https://www.youtube.com/watch?v=zPewo6IaJ_8&list=PL5Q2soXY2Zi9xidyIgBxUz7xRPS-wisBN&index=34

■ Computer Architecture, Spring 2015, Lecture 25

- ❑ Prefetching (CMU, Spring 2015)
- ❑ <https://www.youtube.com/watch?v=ibPL7T9iEwY&list=PL5PHm2jkkXmi5CxxI7b3JCL1TWybTDtKq&index=29>

■ Computer Architecture, Spring 2015, Lecture 26

- ❑ More Prefetching (CMU, Spring 2015)
- ❑ <https://www.youtube.com/watch?v=TUFins4z6o4&list=PL5PHm2jkkXmi5CxxI7b3JCL1TWybTDtKq&index=30>

Recommended Readings on Prefetching

- Onur Mutlu, Jared Stark, Chris Wilkerson, and Yale N. Patt,
"Runahead Execution: An Alternative to Very Large Instruction Windows for Out-of-order Processors"
Proceedings of the 9th International Symposium on High-Performance Computer Architecture (HPCA), pages 129-140, Anaheim, CA, February 2003. [Slides \(pdf\)](#)
One of the 15 computer arch. papers of 2003 selected as Top Picks by IEEE Micro. HPCA Test of Time Award (awarded in 2021).
[\[Lecture Slides \(pptx\) \(pdf\)\]](#)
[\[Lecture Video \(1 hr 54 mins\)\]](#)
[\[Retrospective HPCA Test of Time Award Talk Slides \(pptx\) \(pdf\)\]](#)
[\[Retrospective HPCA Test of Time Award Talk Video \(14 minutes\)\]](#)

Runahead Execution: An Alternative to Very Large Instruction Windows for Out-of-order Processors

Onur Mutlu § Jared Stark † Chris Wilkerson ‡ Yale N. Patt §

§ECE Department
The University of Texas at Austin
{onur,patt}@ece.utexas.edu

†Microprocessor Research
Intel Labs
jared.w.stark@intel.com

‡Desktop Platforms Group
Intel Corporation
chris.wilkerson@intel.com

Recommended Readings on Prefetching

- Onur Mutlu, Jared Stark, Chris Wilkerson, and Yale N. Patt,
["Runahead Execution: An Effective Alternative to Large Instruction Windows"](#)
IEEE Micro, Special Issue: Micro's Top Picks from Microarchitecture Conferences (MICRO TOP PICKS), Vol. 23, No. 6, pages 20-25, November/December 2003.

RUNAHEAD EXECUTION: AN EFFECTIVE ALTERNATIVE TO LARGE INSTRUCTION WINDOWS

Recommended Readings on Prefetching

- Santhosh Srinath, Onur Mutlu, Hyesoon Kim, and Yale N. Patt,
"Feedback Directed Prefetching: Improving the Performance and Bandwidth-Efficiency of Hardware Prefetchers"
Proceedings of the 13th International Symposium on High-Performance Computer Architecture (HPCA), pages 63-74, Phoenix, AZ, February 2007. Slides (ppt)
One of the five papers nominated for the Best Paper Award by the Program Committee.

Feedback Directed Prefetching:

Improving the Performance and Bandwidth-Efficiency of Hardware Prefetchers

Santhosh Srinath^{†‡} Onur Mutlu[§] Hyesoon Kim[‡] Yale N. Patt[‡]

[†]Microsoft
ssri@microsoft.com

[§]Microsoft Research
onur@microsoft.com

[‡]Department of Electrical and Computer Engineering
The University of Texas at Austin
{santhosh, hyesoon, patt}@ece.utexas.edu

Recommended Readings on Prefetching

- Onur Mutlu, Hyesoon Kim, and Yale N. Patt,
["Efficient Runahead Execution: Power-Efficient Memory Latency Tolerance"](#)
IEEE Micro, Special Issue: Micro's Top Picks from Microarchitecture Conferences (MICRO TOP PICKS), Vol. 26, No. 1, pages 10-20, January/February 2006.

EFFICIENT RUNAHEAD EXECUTION: POWER-EFFICIENT MEMORY LATENCY TOLERANCE

Digital Design & Computer Arch.

Lecture 25a: Prefetching

Prof. Onur Mutlu

ETH Zürich

Spring 2021

3 June 2021

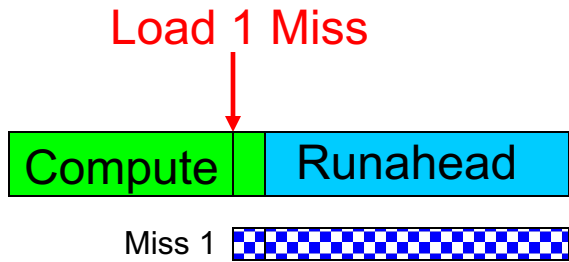
Backup Slides:

More on Runahead Execution

Runahead Execution Mechanism

- Entry into runahead mode
 - Checkpoint architectural register state
 - Instruction processing in runahead mode
 - Exit from runahead mode
 - Restore architectural register state from checkpoint
-

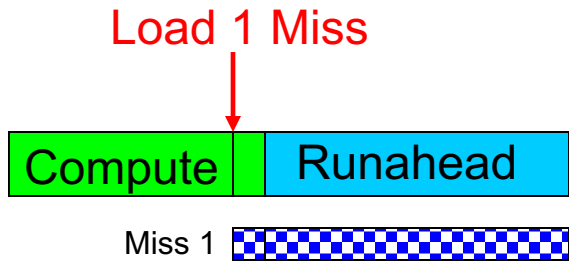
Instruction Processing in Runahead Mode



Runahead mode processing is the same as normal instruction processing, EXCEPT:

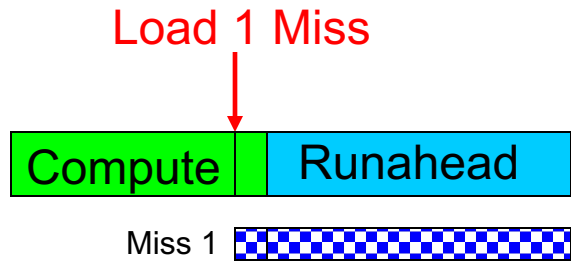
- It is purely speculative: Architectural (software-visible) register/memory state is NOT updated in runahead mode.
- L2-miss dependent instructions are identified and treated specially.
 - ❑ They are quickly removed from the instruction window.
 - ❑ Their results are not trusted.

L2-Miss Dependent Instructions



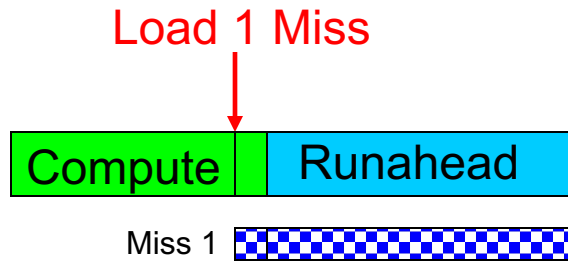
- Two types of results produced: INV and VALID
 - INV = Dependent on an L2 miss
 - INV results are marked using INV bits in the register file and store buffer.
 - INV values are not used for prefetching/branch resolution.
-

Removal of Instructions from Window



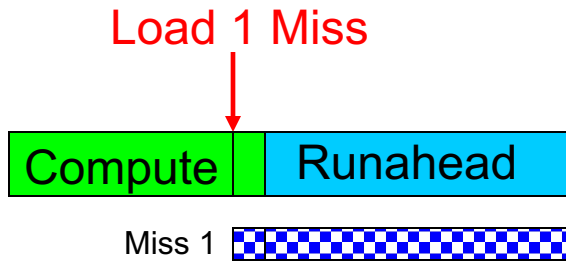
- Oldest instruction is examined for **pseudo-retirement**
 - ❑ An INV instruction is removed from window immediately.
 - ❑ A VALID instruction is removed when it completes execution.
- **Pseudo-retired instructions free their allocated resources.**
 - ❑ This allows the processing of later instructions.
- Pseudo-retired stores communicate their data to dependent loads.

Store/Load Handling in Runahead Mode



- A pseudo-retired store writes its data and INV status to a dedicated memory, called **runahead cache**.
 - Purpose: Data communication through memory in runahead mode.
 - A dependent load reads its data from the runahead cache.
 - Does not need to be always correct → Size of runahead cache is very small.
-

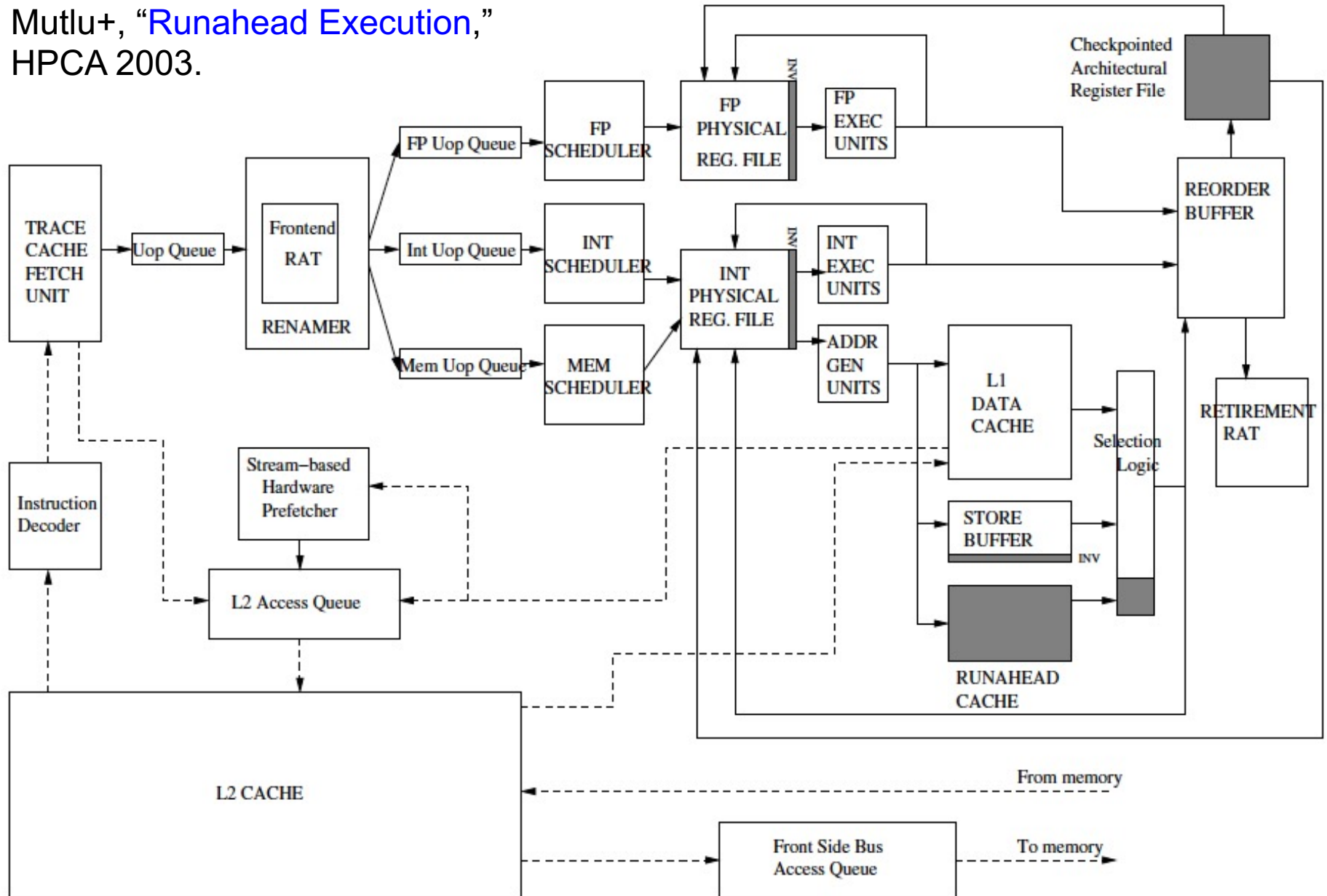
Branch Handling in Runahead Mode



- **INV branches cannot be resolved.**
 - A mispredicted INV branch causes the processor to stay on the wrong program path until the end of runahead execution.
- **VALID branches are resolved and initiate recovery if mispredicted.**

A Runahead Processor Diagram

Mutlu+, “Runahead Execution,”
HPCA 2003.



Limitations of the Baseline Runahead Mechanism

■ Energy Inefficiency

- ❑ A large number of instructions are speculatively executed
- ❑ **Efficient Runahead Execution** [ISCA'05, IEEE Micro Top Picks'06]

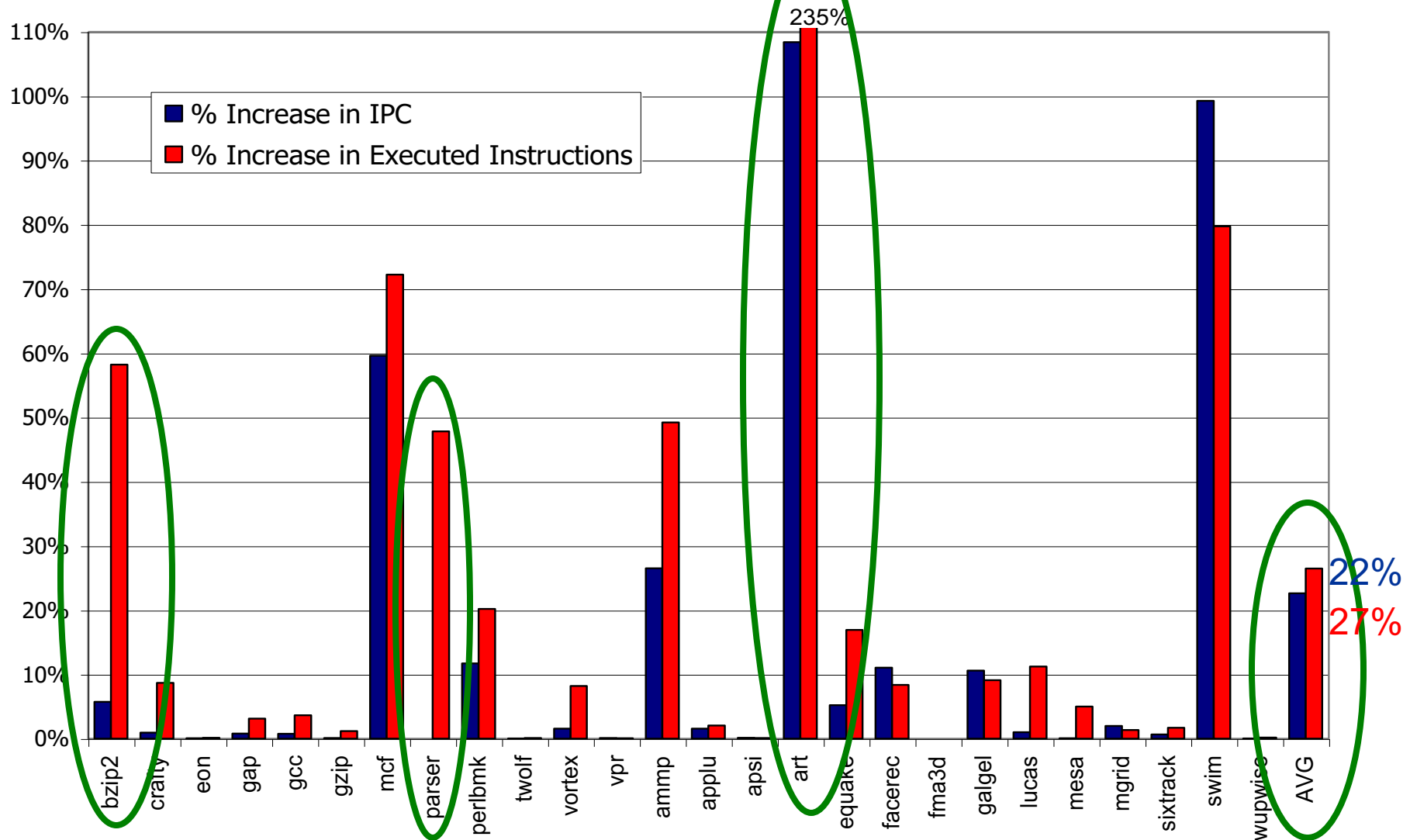
■ Ineffectiveness for pointer-intensive applications

- ❑ Runahead cannot parallelize dependent L2 cache misses
- ❑ **Address-Value Delta (AVD) Prediction** [MICRO'05]

■ Irresolvable branch mispredictions in runahead mode

- ❑ Cannot recover from a mispredicted L2-miss dependent branch
 - ❑ **Wrong Path Events** [MICRO'04]
 - ❑ **Wrong Path Memory Reference Analysis** [IEEE TC'05]
-

The Efficiency Problem

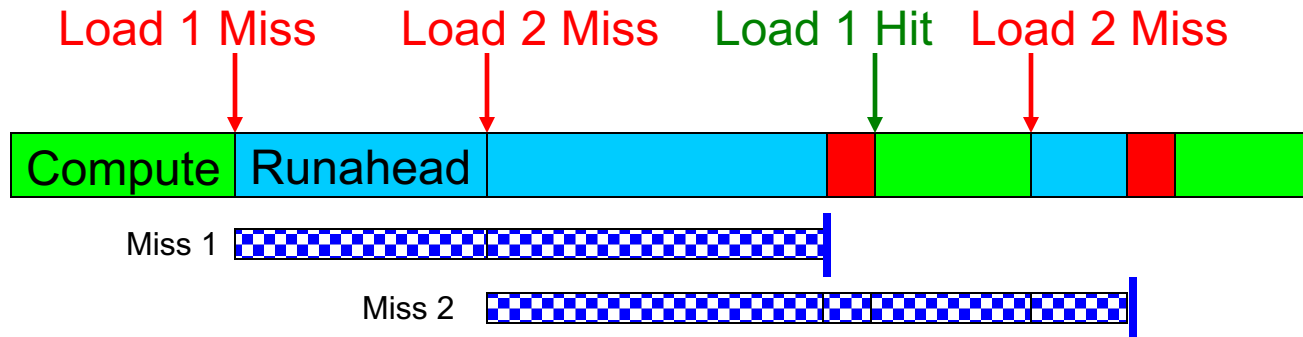


Causes of Inefficiency

- Short runahead periods
 - Overlapping runahead periods
 - Useless runahead periods
 - Mutlu et al., “Efficient Runahead Execution: Power-Efficient Memory Latency Tolerance,” ISCA 2005, IEEE Micro Top Picks 2006.
-

Short Runahead Periods

- Processor can initiate runahead mode due to an already in-flight L2 miss generated by
 - the prefetcher, wrong-path, or a previous runahead period



- Short periods
 - are less likely to generate useful L2 misses
 - have high overhead due to the flush penalty at runahead exit
-

Overlapping Runahead Periods

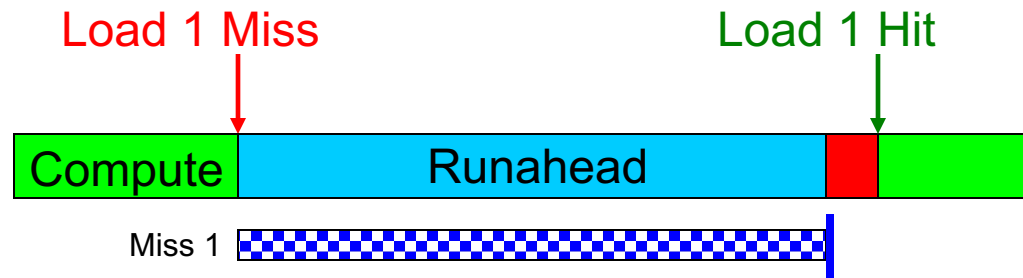
- Two runahead periods that execute the same instructions



- Second period is inefficient
-

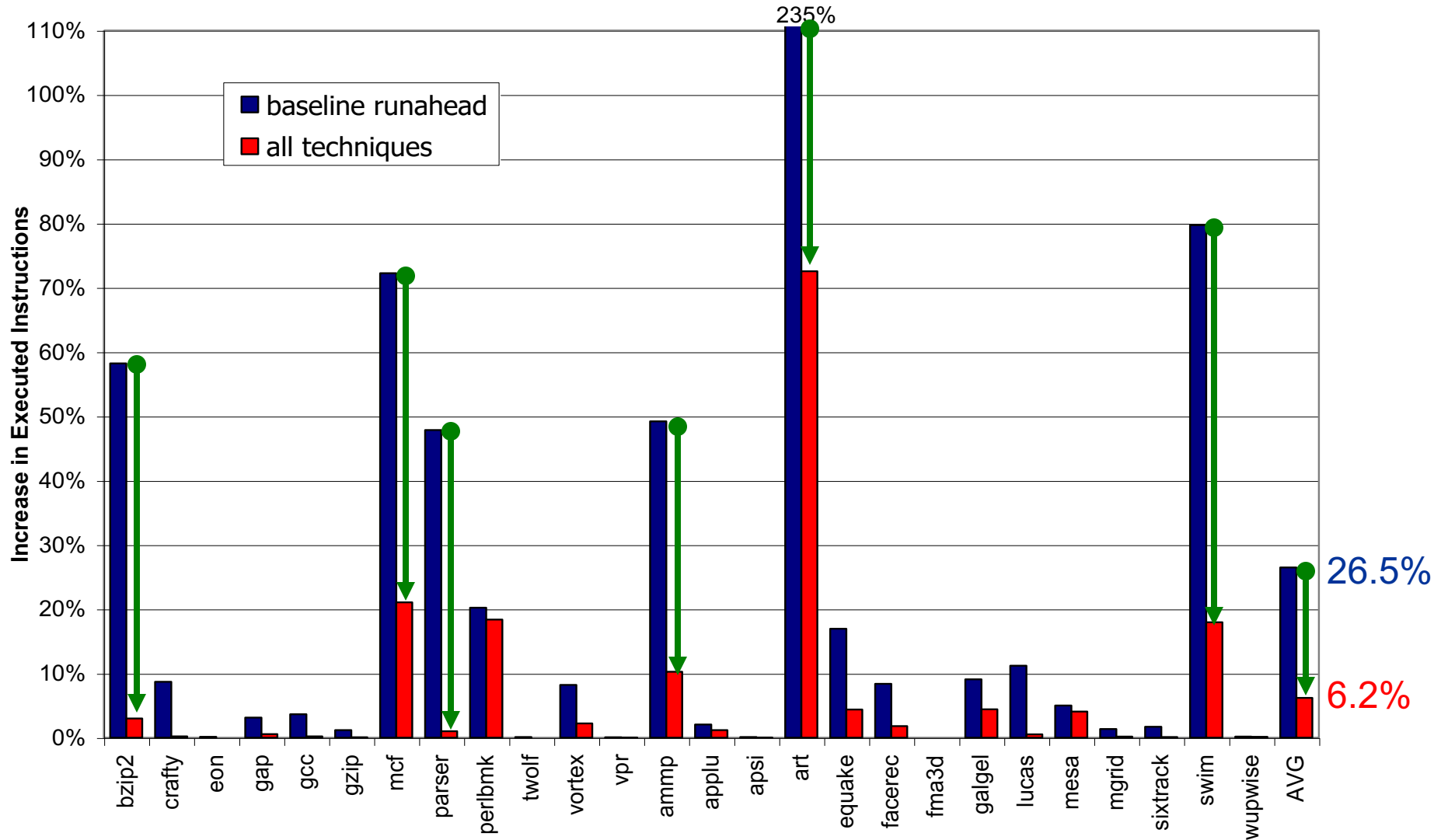
Useless Runahead Periods

- Periods that do not result in prefetches for normal mode

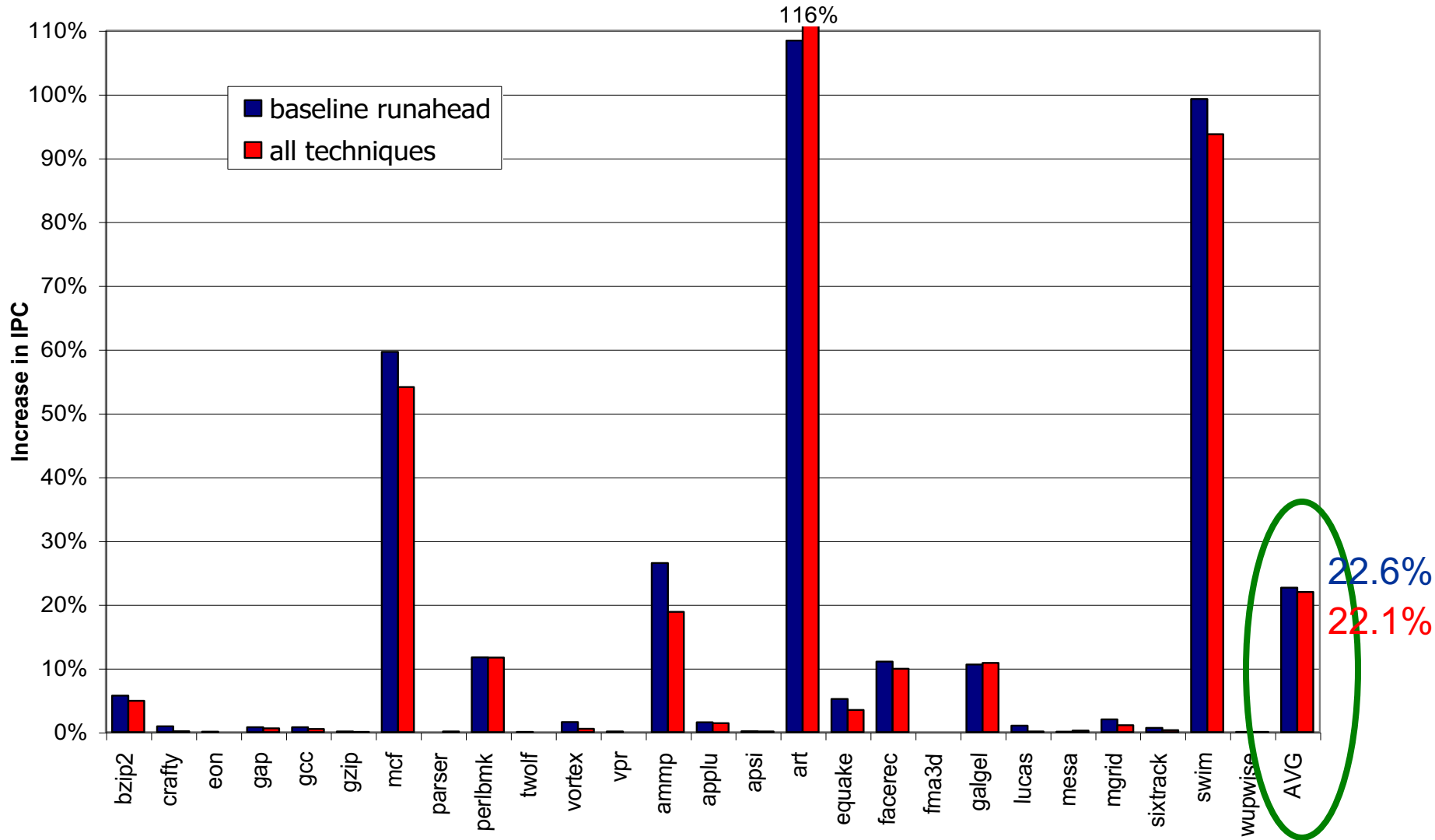


- They exist due to the lack of memory-level parallelism
 - Mechanism to eliminate useless periods:
 - Predict if a period will generate useful L2 misses
 - Estimate a period to be useful if it generated an L2 miss that cannot be captured by the instruction window
 - Useless period predictors are trained based on this estimation
-

Overall Impact on Executed Instructions



Overall Impact on IPC



More on Efficient Runahead Execution

- Onur Mutlu, Hyesoon Kim, and Yale N. Patt,
"Techniques for Efficient Processing in Runahead Execution Engines"
Proceedings of the 32nd International Symposium on Computer Architecture (ISCA), pages 370-381, Madison, WI, June 2005. [Slides \(ppt\)](#) [Slides \(pdf\)](#)
One of the 13 computer architecture papers of 2005 selected as Top Picks by IEEE Micro.

Techniques for Efficient Processing in Runahead Execution Engines

Onur Mutlu Hyesoon Kim Yale N. Patt

Department of Electrical and Computer Engineering

University of Texas at Austin

{onur,hyesoon,patt}@ece.utexas.edu

More on Efficient Runahead Execution

- Onur Mutlu, Hyesoon Kim, and Yale N. Patt,
"Efficient Runahead Execution: Power-Efficient Memory Latency Tolerance"
IEEE Micro, Special Issue: Micro's Top Picks from Microarchitecture Conferences (MICRO TOP PICKS), Vol. 26, No. 1, pages 10-20, January/February 2006.

EFFICIENT RUNAHEAD EXECUTION: POWER-EFFICIENT MEMORY LATENCY TOLERANCE

Limitations of the Baseline Runahead Mechanism

■ Energy Inefficiency

- ❑ A large number of instructions are speculatively executed
- ❑ **Efficient Runahead Execution** [ISCA'05, IEEE Micro Top Picks'06]

■ Ineffectiveness for pointer-intensive applications

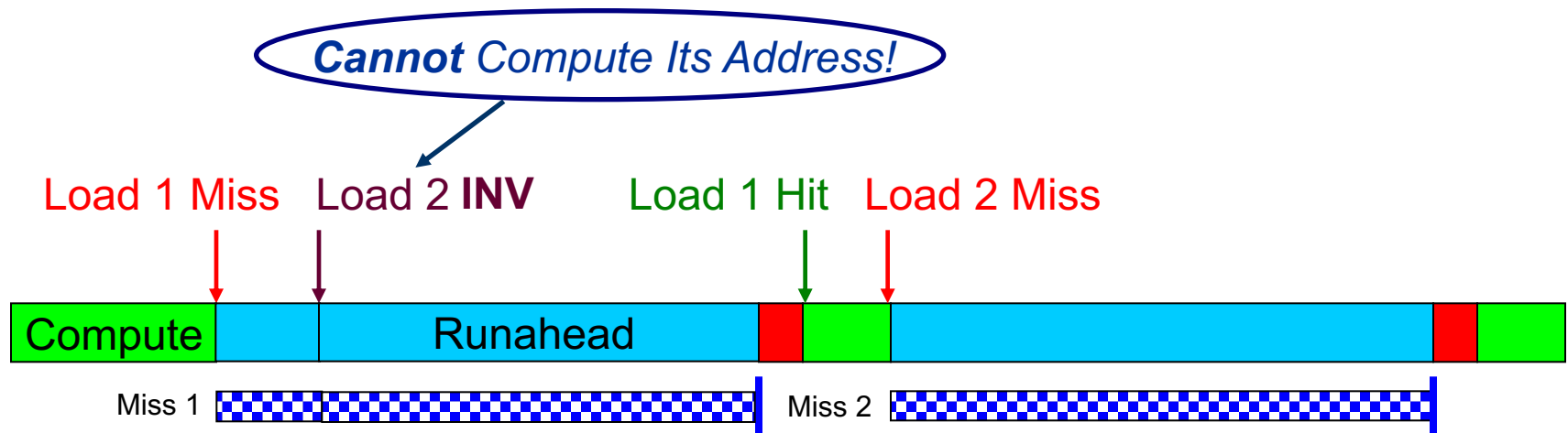
- ❑ Runahead cannot parallelize dependent L2 cache misses
- ❑ **Address-Value Delta (AVD) Prediction** [MICRO'05]

■ Irresolvable branch mispredictions in runahead mode

- ❑ Cannot recover from a mispredicted L2-miss dependent branch
 - ❑ **Wrong Path Events** [MICRO'04]
 - ❑ **Wrong Path Memory Reference Analysis** [IEEE TC'05]
-

The Problem: Dependent Cache Misses

Runahead: **Load 2 is *dependent* on Load 1**

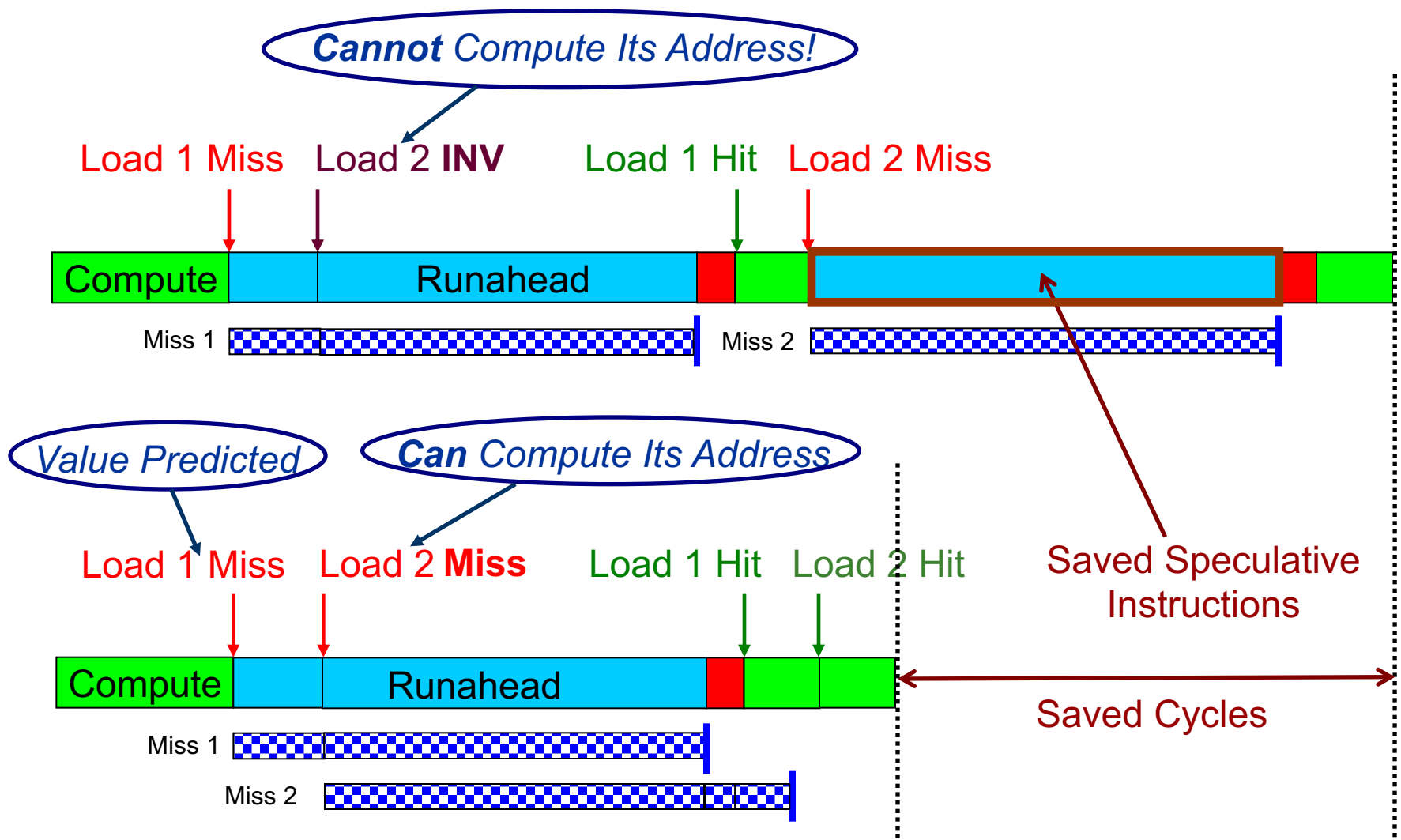


- Runahead execution cannot parallelize dependent misses
 - ❑ wasted opportunity to improve performance
 - ❑ wasted energy (useless pre-execution)
- Runahead performance would improve by 25% if this limitation were ideally overcome

Parallelizing Dependent Cache Misses

- **Idea:** Enable the parallelization of dependent L2 cache misses in runahead mode with a low-cost mechanism
 - **How:** Predict the values of L2-miss **address (pointer) loads**
 - **Address load:** loads an address into its destination register, which is later used to calculate the address of another load
 - as opposed to **data load**
 - **Read:**
 - Mutlu et al., “Address-Value Delta (AVD) Prediction,” MICRO 2005.
-

Parallelizing Dependent Cache Misses



More on AVD Prediction

- Onur Mutlu, Hyesoon Kim, and Yale N. Patt,
"Address-Value Delta (AVD) Prediction: Increasing the Effectiveness of Runahead Execution by Exploiting Regular Memory Allocation Patterns"
Proceedings of the 38th International Symposium on Microarchitecture (MICRO),
pages 233-244, Barcelona, Spain, November 2005. [Slides \(ppt\)](#) [Slides \(pdf\)](#)
One of the five papers nominated for the Best Paper Award by the Program Committee.

Address-Value Delta (AVD) Prediction: Increasing the Effectiveness of Runahead Execution by Exploiting Regular Memory Allocation Patterns

Onur Mutlu Hyesoon Kim Yale N. Patt

Department of Electrical and Computer Engineering
University of Texas at Austin
{onur,hyesoon,patt}@ece.utexas.edu

More on AVD Prediction (II)

- Onur Mutlu, Hyesoon Kim, and Yale N. Patt,
"Address-Value Delta (AVD) Prediction: A Hardware Technique for Efficiently Parallelizing Dependent Cache Misses"
IEEE Transactions on Computers (TC), Vol. 55, No. 12, pages 1491-1508, December 2006.

Address-Value Delta (AVD) Prediction: A Hardware Technique for Efficiently Parallelizing Dependent Cache Misses

Onur Mutlu, *Member, IEEE*, Hyesoon Kim, *Student Member, IEEE*, and
Yale N. Patt, *Fellow, IEEE*

Limitations of the Baseline Runahead Mechanism

■ Energy Inefficiency

- ❑ A large number of instructions are speculatively executed
- ❑ **Efficient Runahead Execution** [ISCA'05, IEEE Micro Top Picks'06]

■ Ineffectiveness for pointer-intensive applications

- ❑ Runahead cannot parallelize dependent L2 cache misses
- ❑ **Address-Value Delta (AVD) Prediction** [MICRO'05]

■ Irresolvable branch mispredictions in runahead mode

- ❑ Cannot recover from a mispredicted L2-miss dependent branch
 - ❑ **Wrong Path Events** [MICRO'04]
 - ❑ **Wrong Path Memory Reference Analysis** [IEEE TC'05]
-

Wrong Path Events

- David N. Armstrong, Hyesoon Kim, Onur Mutlu, and Yale N. Patt, **"Wrong Path Events: Exploiting Unusual and Illegal Program Behavior for Early Misprediction Detection and Recovery"**
Proceedings of the 37th International Symposium on Microarchitecture (MICRO), pages 119-128, Portland, OR, December 2004. [Slides \(pdf\)](#)[Slides \(ppt\)](#)

Wrong Path Events: Exploiting Unusual and Illegal Program Behavior for Early Misprediction Detection and Recovery

David N. Armstrong Hyesoon Kim Onur Mutlu Yale N. Patt

Department of Electrical and Computer Engineering
The University of Texas at Austin
{dna,hyesoon,onur,patt}@ece.utexas.edu

Effects of Wrong Path Execution (I)

- Onur Mutlu, Hyesoon Kim, David N. Armstrong, and Yale N. Patt,
"Understanding the Effects of Wrong-Path Memory References on Processor Performance"
Proceedings of the 3rd Workshop on Memory Performance Issues (WMPI), pages 56-64, Munchen, Germany, June 2004. [Slides](#)
[\(pdf\)](#)

Understanding The Effects of Wrong-Path Memory References on Processor Performance

Onur Mutlu Hyesoon Kim David N. Armstrong Yale N. Patt

Department of Electrical and Computer Engineering
The University of Texas at Austin
{onur,hyesoon,dna,patt}@ece.utexas.edu

Effects of Wrong Path Execution (II)

- Onur Mutlu, Hyesoon Kim, David N. Armstrong, and Yale N. Patt, **"An Analysis of the Performance Impact of Wrong-Path Memory References on Out-of-Order and Runahead Execution Processors"** *IEEE Transactions on Computers (TC)*, Vol. 54, No. 12, pages 1556-1571, December 2005.

An Analysis of the Performance Impact of Wrong-Path Memory References on Out-of-Order and Runahead Execution Processors

Onur Mutlu, *Student Member, IEEE*, Hyesoon Kim, *Student Member, IEEE*,
David N. Armstrong, and Yale N. Patt, *Fellow, IEEE*