Runahead Execution

A Short Retrospective

Onur Mutlu, Jared Stark, Chris Wilkerson, Yale Patt

HPCA 2021 ToT Award Talk

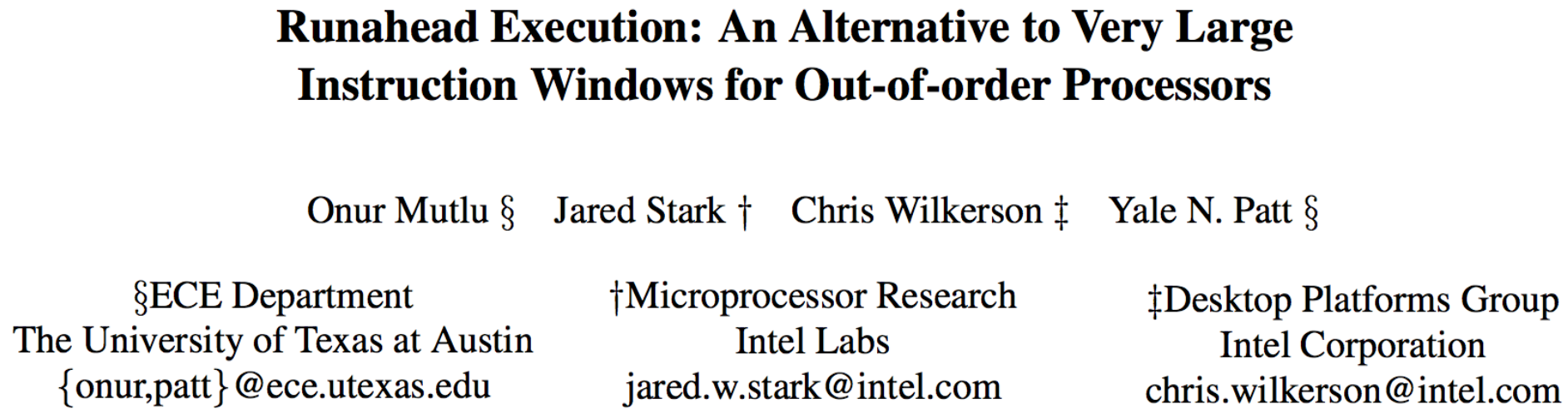
2 March 2021

Agenda

- Thanks
- Runahead Execution
- Looking to the Past
- Looking to the Future

Runahead Execution [HPCA 2003]

- 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**)*, Anaheim, CA, February 2003. Slides (pdf)
  One of the 15 computer architecture papers of 2003 selected as Top Picks by IEEE Micro.



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

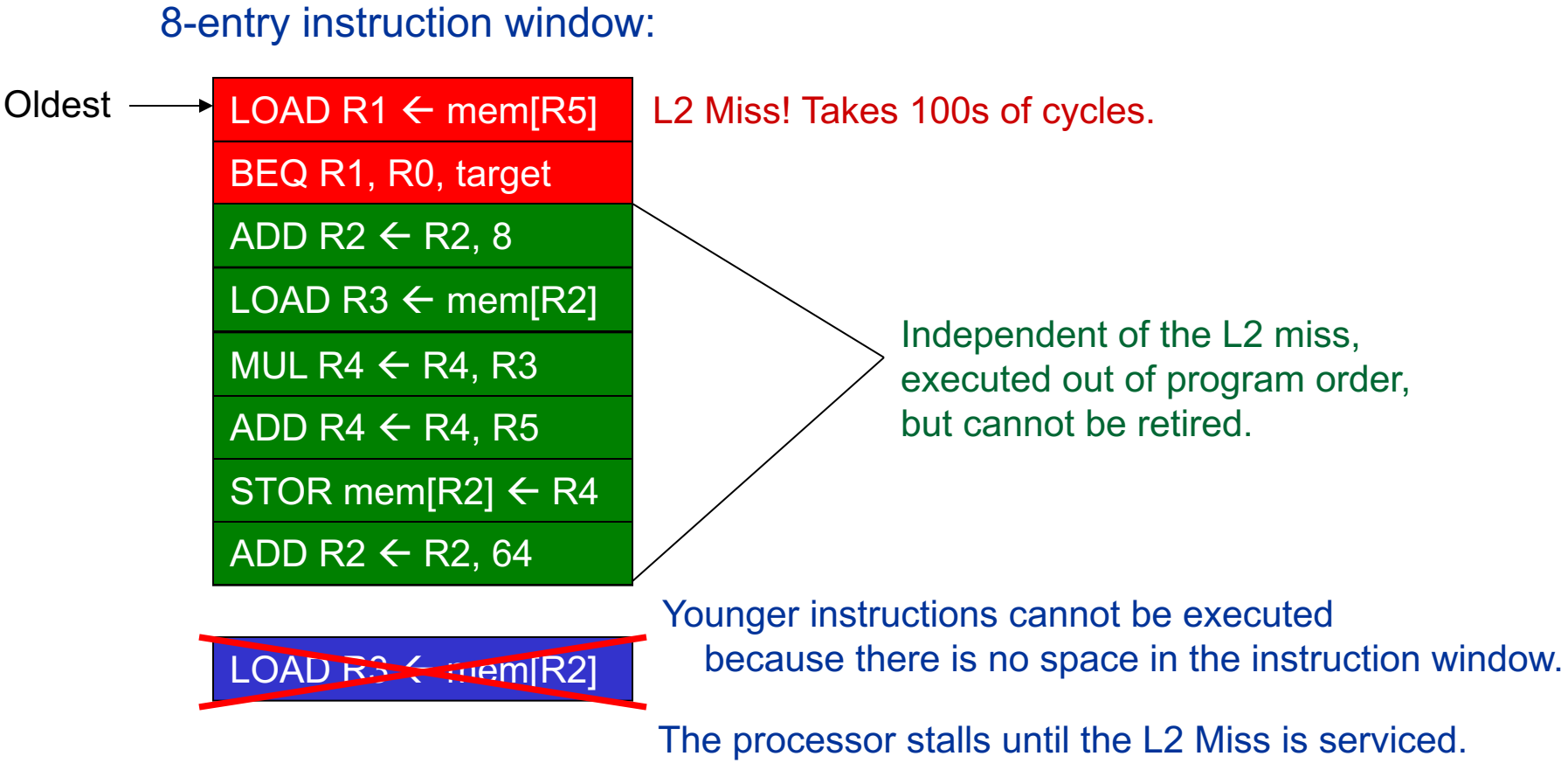
Onur Mutlu § Jared Stark † Chris Wilkerson ‡ Yale N. Patt §

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

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

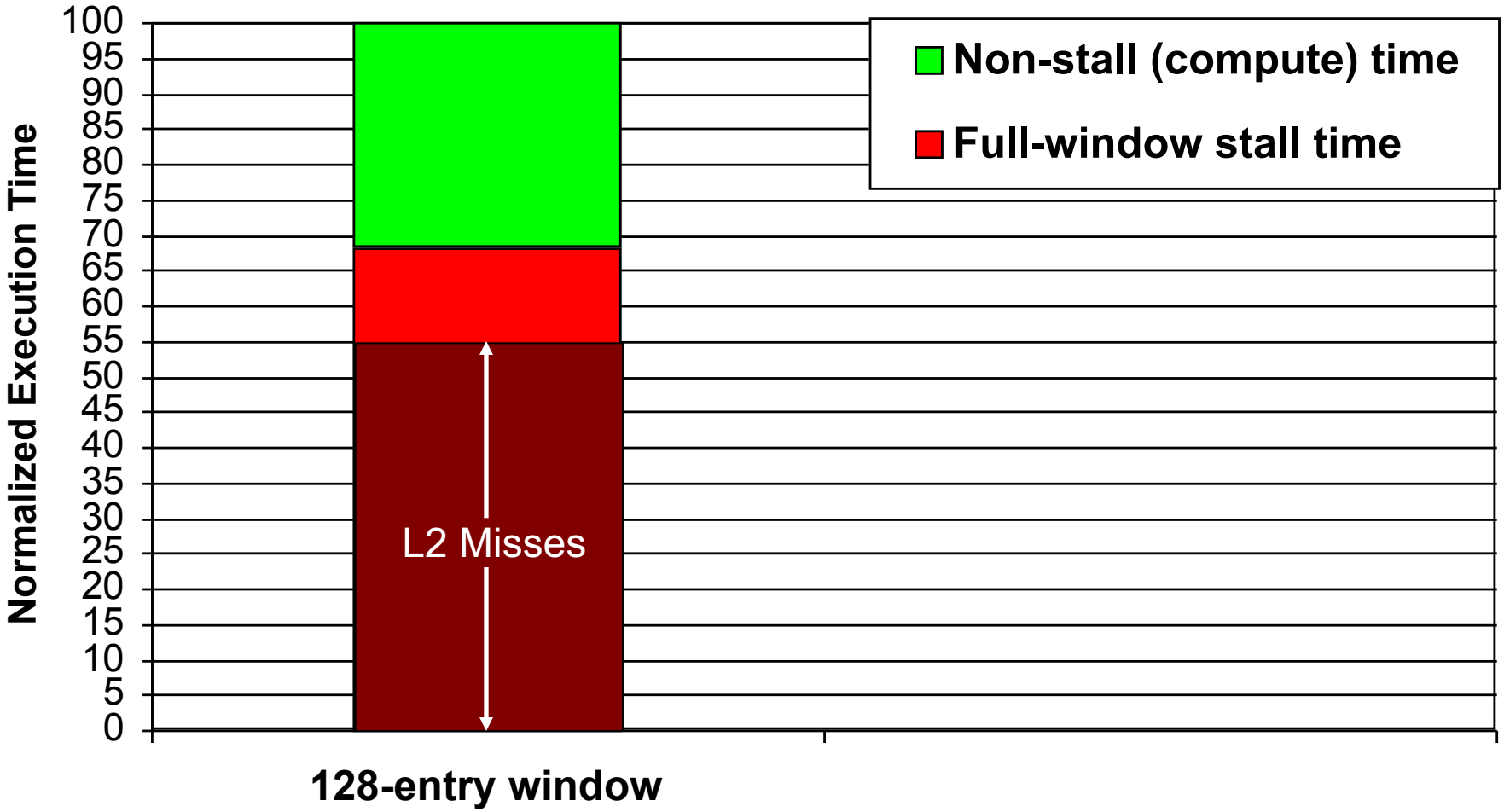
‡Desktop Platforms Group
Intel Corporation
chris.wilkerson@intel.com

Small Windows: Full-Window Stalls



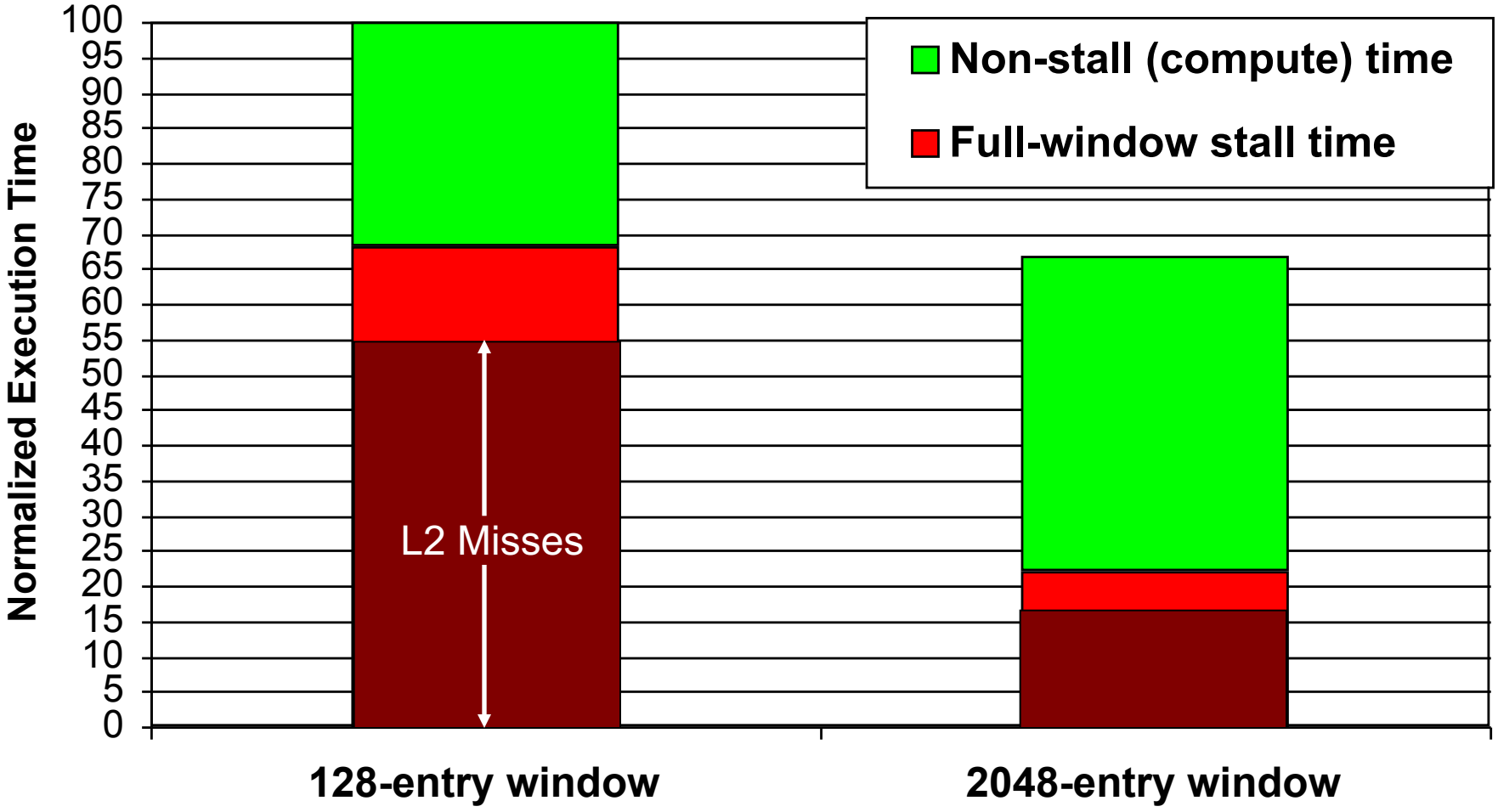
- Long-latency cache misses are responsible for most full-window stalls.

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

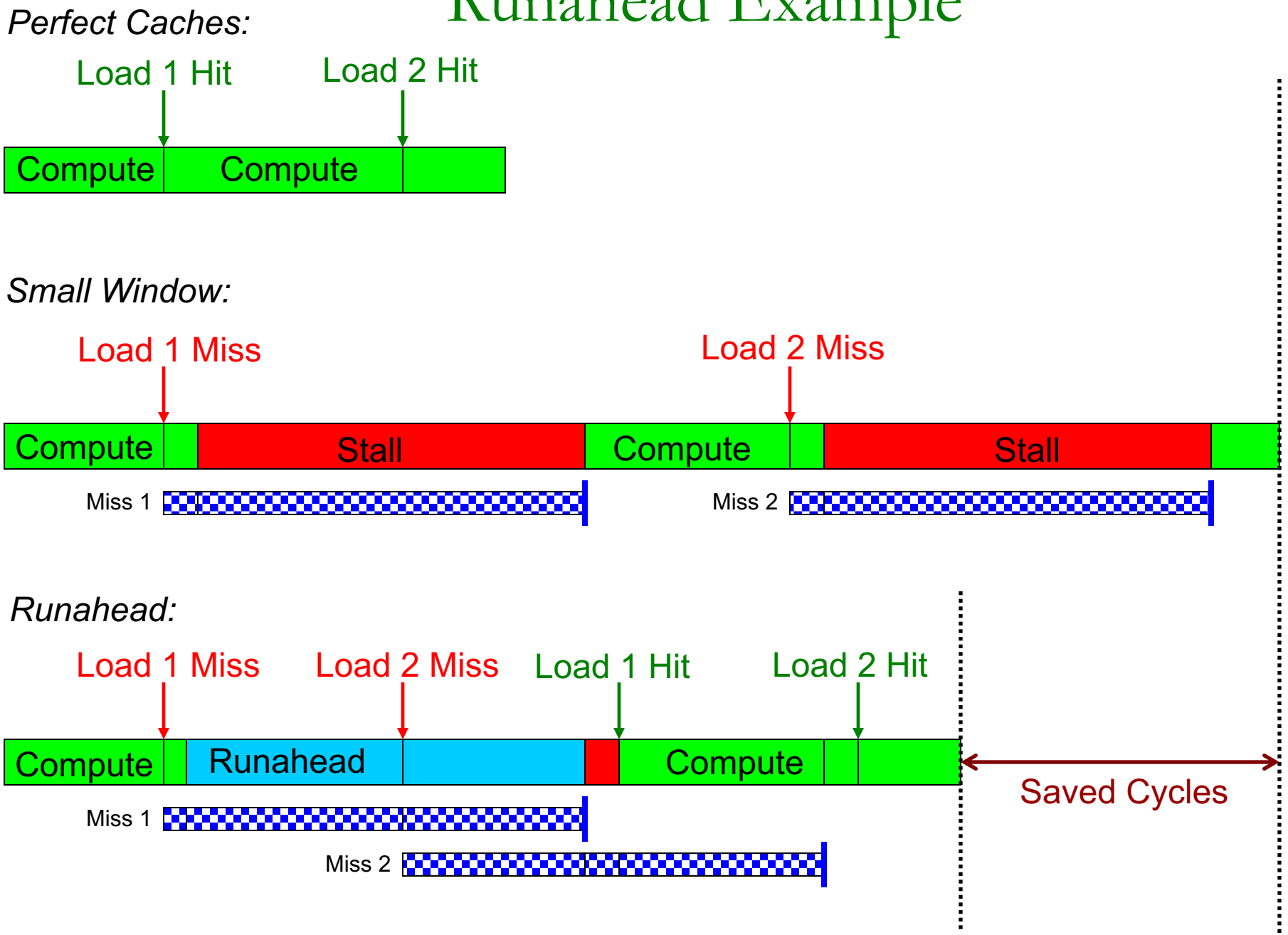
Runahead Execution

- 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



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.

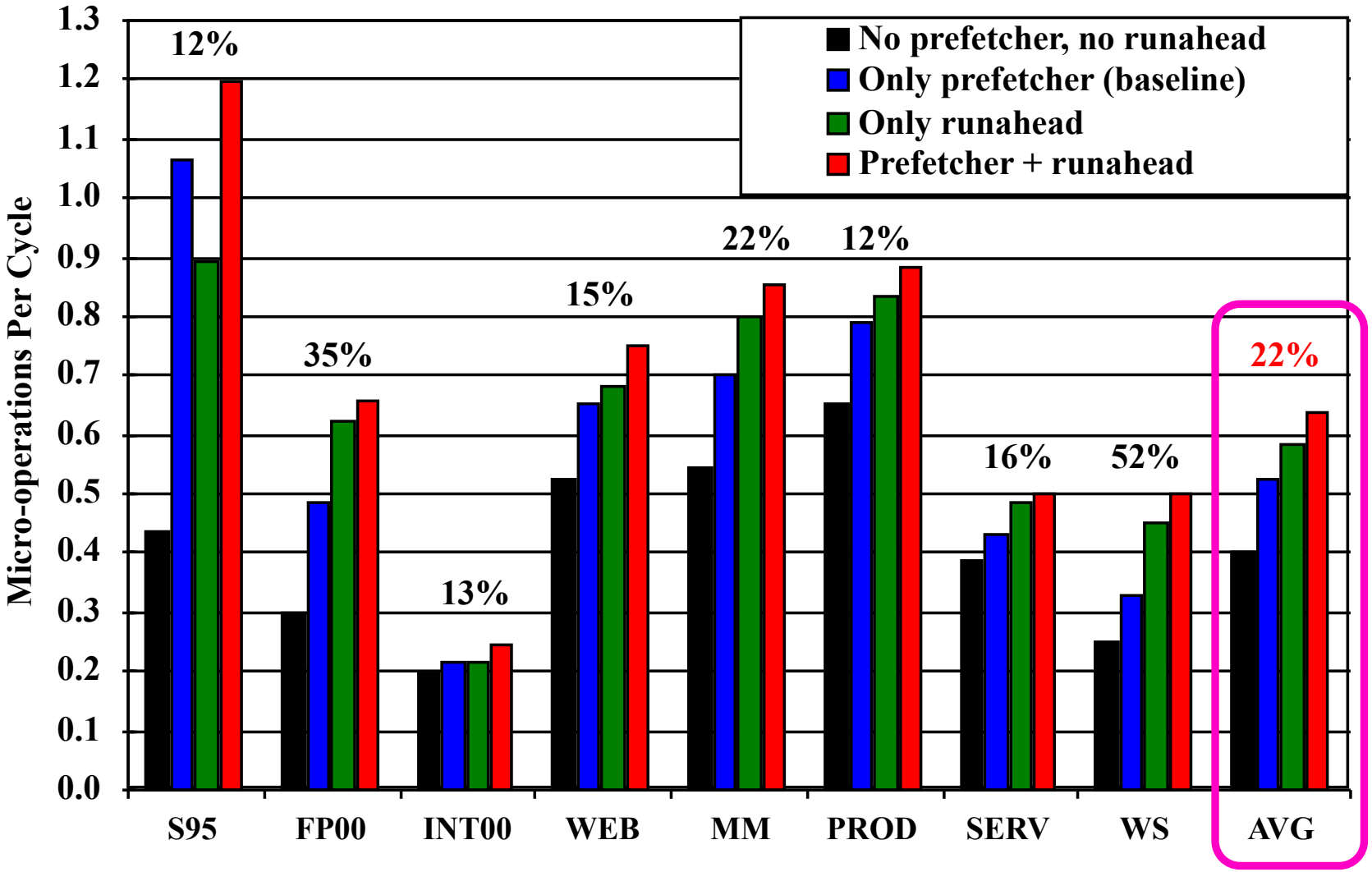
Runahead Execution Pros and Cons

- 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

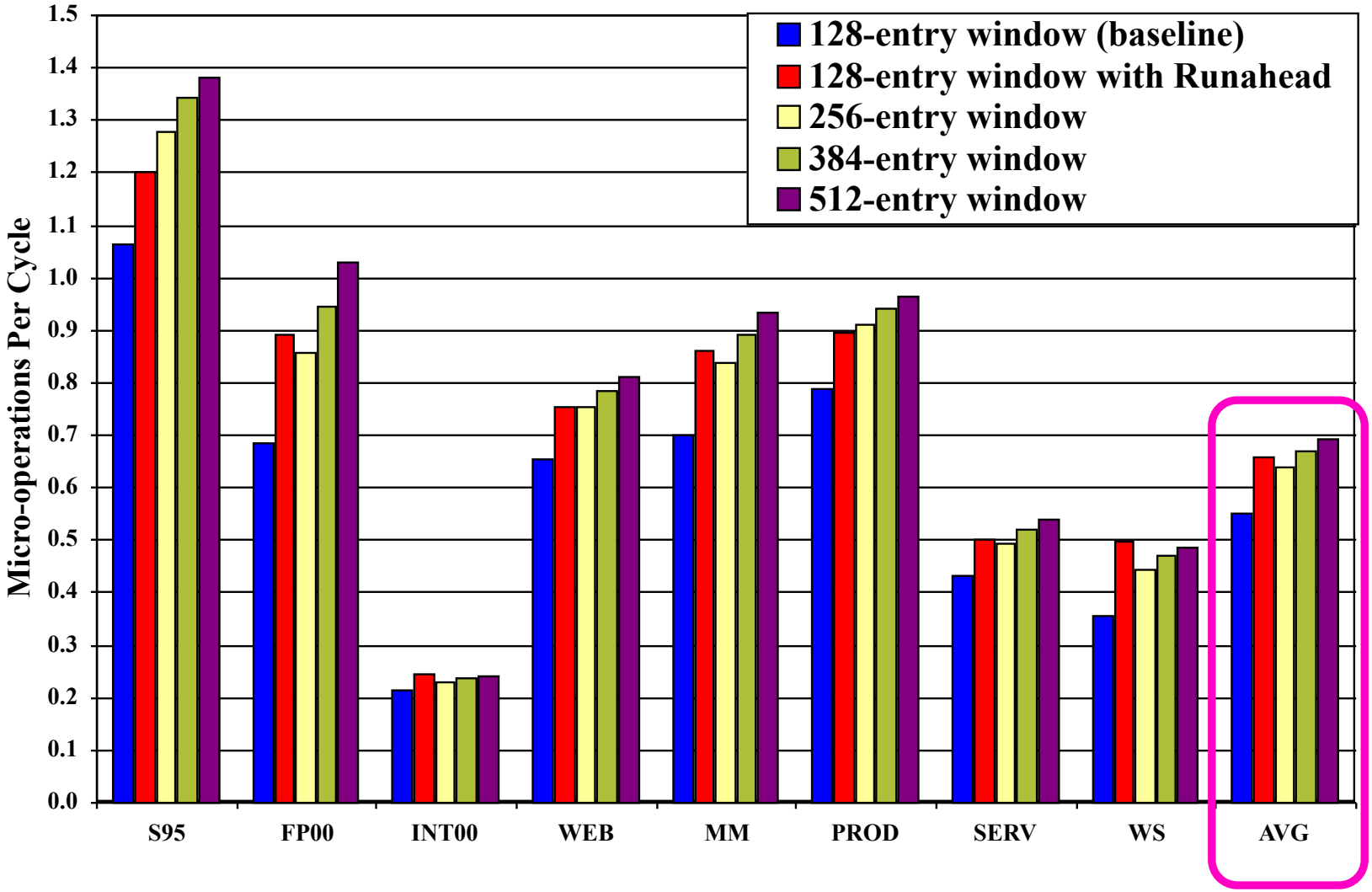
- Disadvantages/Limitations
  - -- Extra executed instructions
  - -- Limited by branch prediction accuracy
  - -- Cannot prefetch dependent cache misses
  - -- Effectiveness limited by available "memory-level parallelism" (MLP)
  - -- Prefetch distance (how far ahead to prefetch) limited by memory latency

- Implemented in Sun ROCK, IBM POWER6, NVIDIA Denver

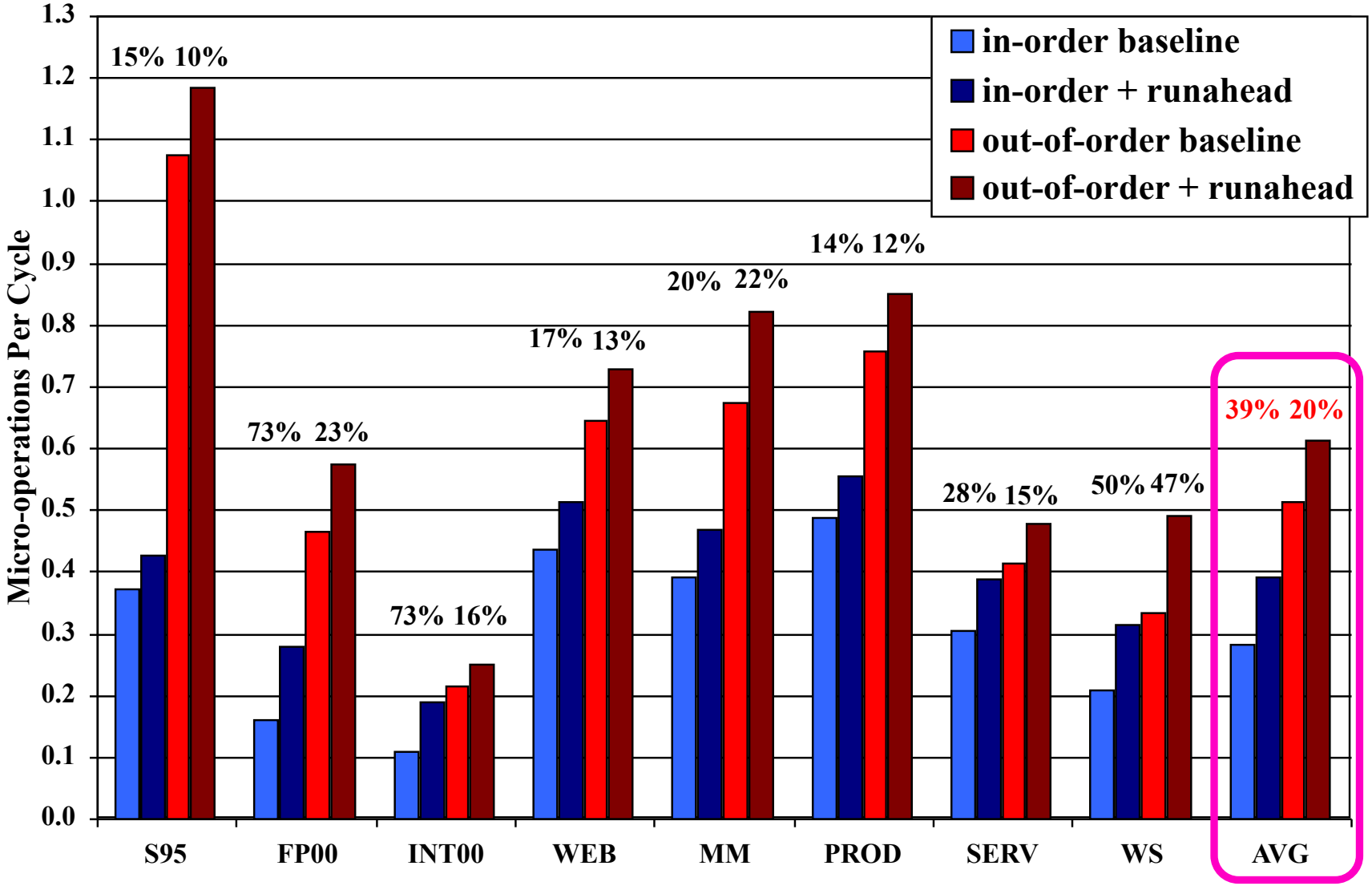
Performance of Runahead Execution



Runahead Execution vs. Large Windows

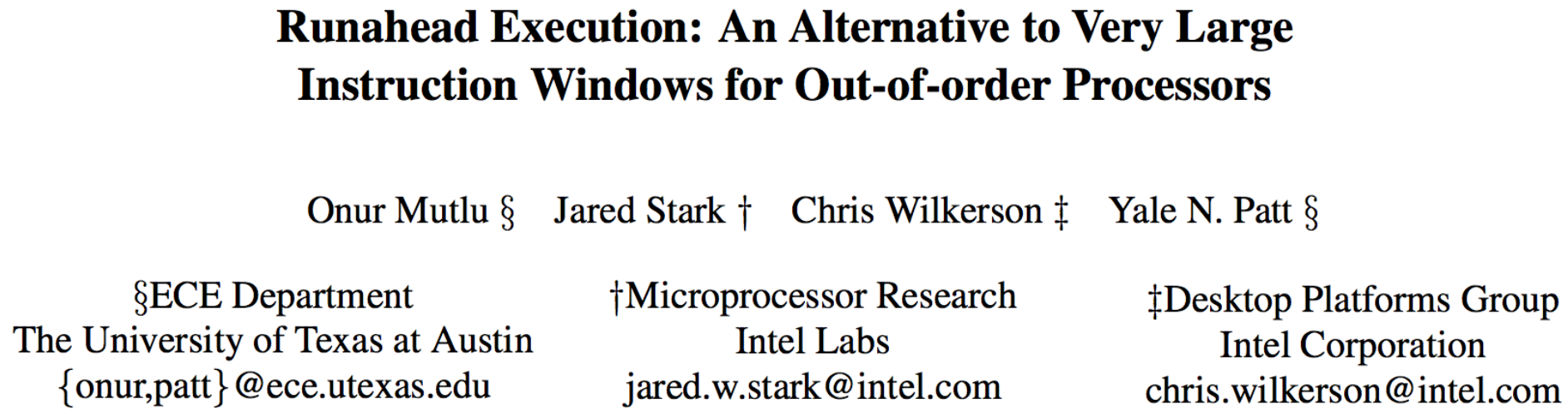


Runahead on In-order vs. Out-of-order



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***), Anaheim, CA, February 2003. Slides (pdf)
  One of the 15 computer architecture papers of 2003 selected as Top Picks by IEEE Micro.



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

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

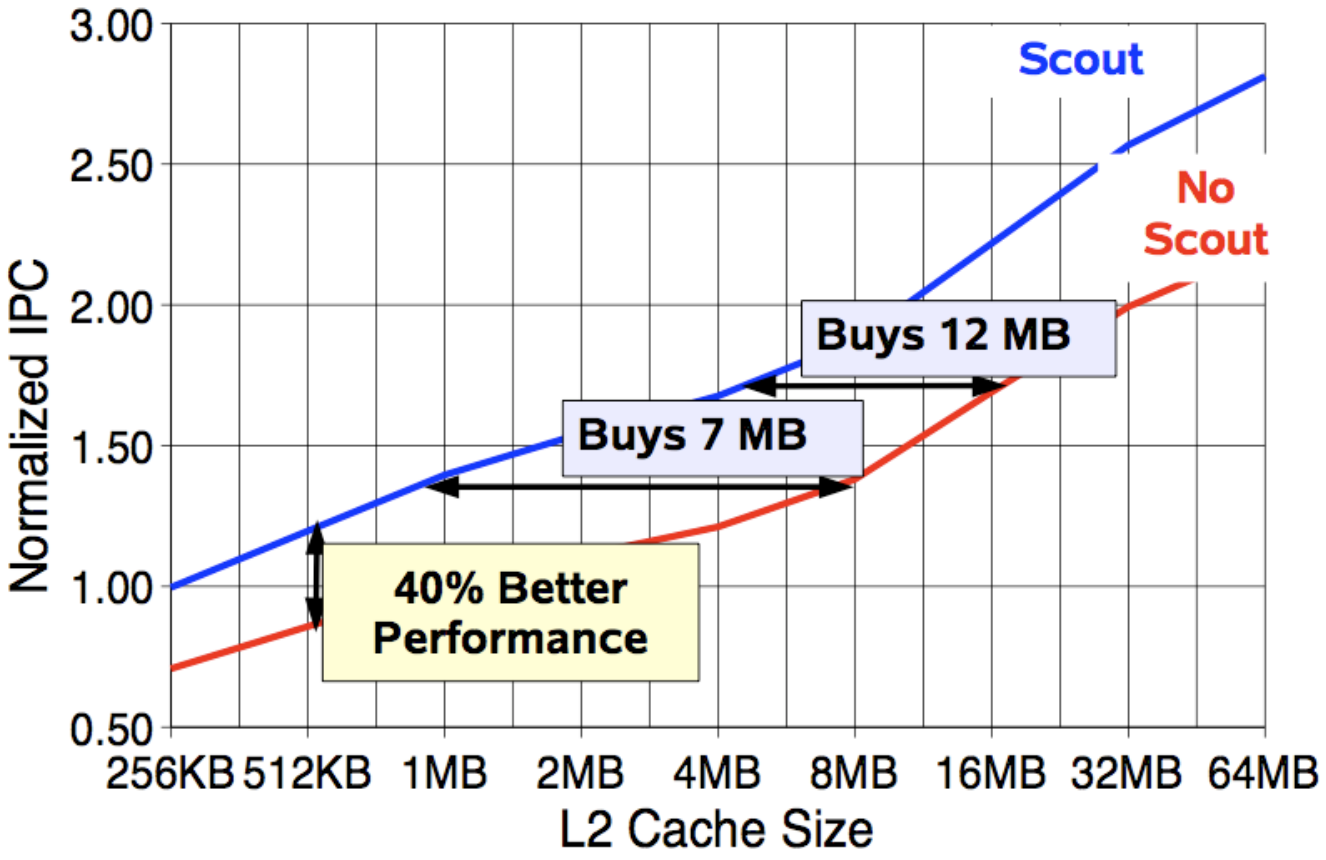
§ECE Department
The University of Texas at Austin
{onur,patt}@ece.utexas.edu

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

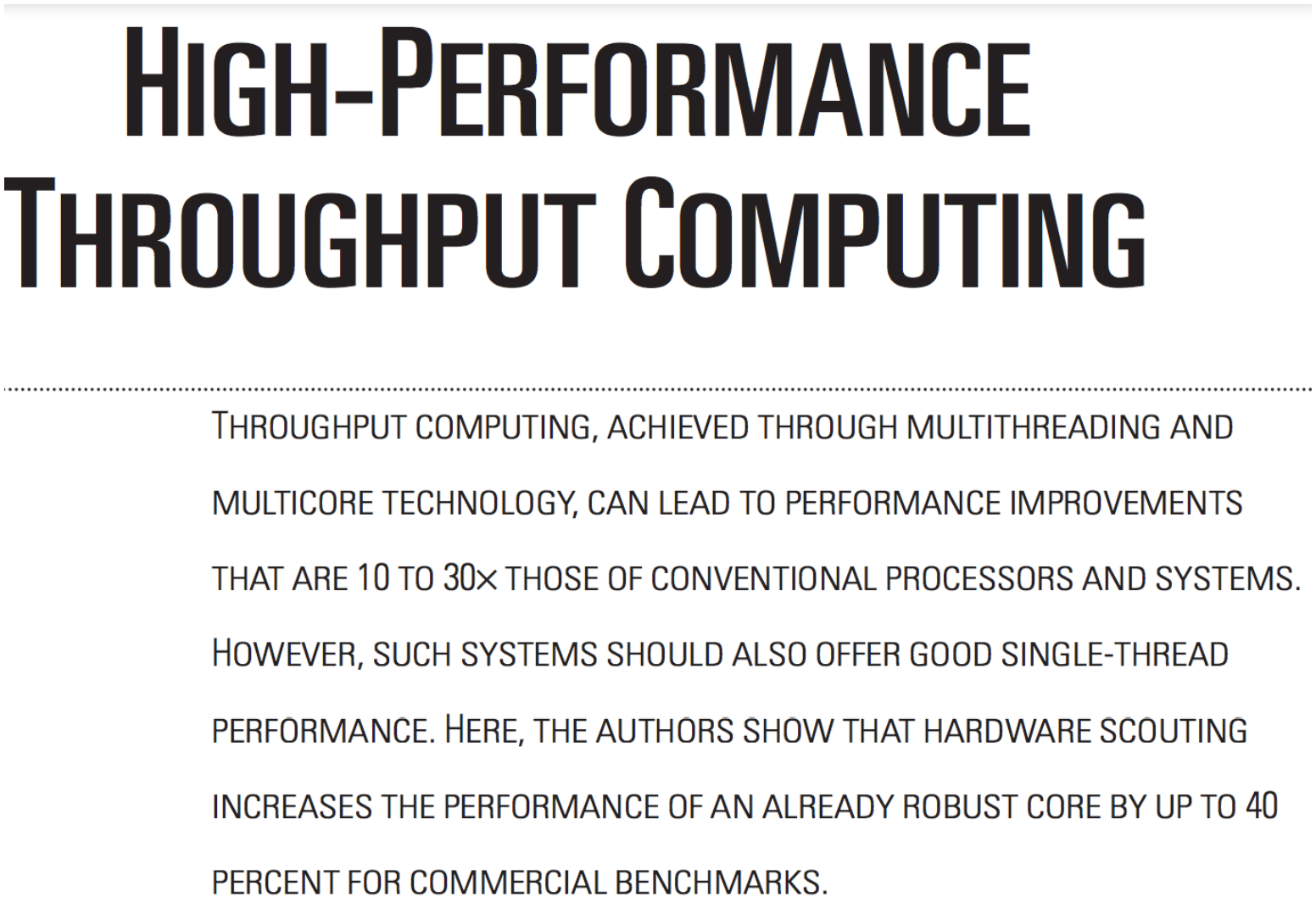
‡Desktop Platforms Group
Intel Corporation
chris.wilkerson@intel.com

Effect of Runahead in Sun ROCK

- Shailender Chaudhry talk, Aug 2008.



More on Runahead in Sun ROCK



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.

Chaudhry+, "High-Performance Throughput Computing," IEEE Micro 2005.

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

Chaudhry+, "Simultaneous Speculative Threading," ISCA 2009.

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).

Runahead Execution in NVIDIA Denver

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 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. Runahead 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).

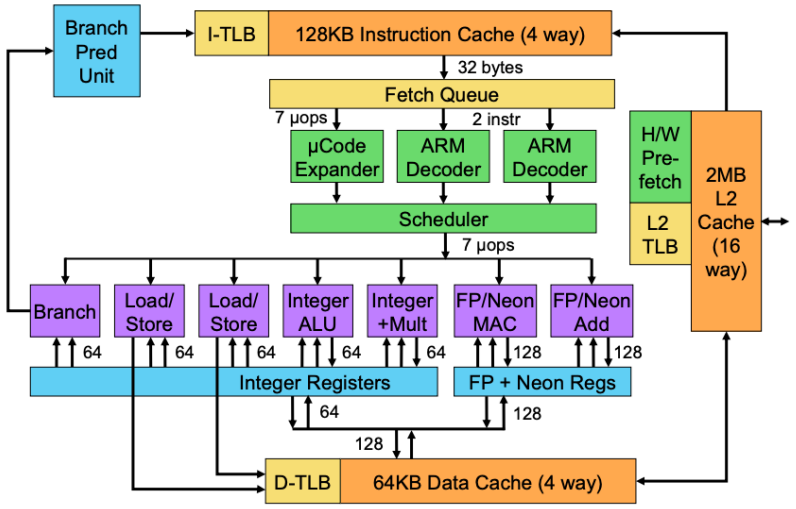


Figure 3. Denver CPU microarchitecture. This design combines a fairly

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

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

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.

Agenda

- Thanks
- Runahead Execution
- Looking to the Past
- Looking to the Future

At the Time…

- 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 non-MLP benefits
  - 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

Dundas+, "Improving Data Cache Performance by Pre-Executing Instructions Under a Cache Miss," ICS 1997.

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.

Glew, "MLP yes! ILP no!," ASPLOS WACI 1998.

Agenda

- Thanks
- Runahead Execution
- Looking to the Past
- 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 solutions

One Final Note

- Our purpose was (and is) to advance the state of the art
- That should be the sole purpose of any scientific effort
- There is a dangerous tendency (especially today) that increasingly goes against the scientific purpose
  - In reviews & paper acceptance decisions
  - In paper formatting rules
  - In publication processes
  - In the bureaucracy academics have created
  - …
- Conferences are not a competition → Let's respect science

Suggestions to Reviewers

- Be fair; you do not know it all
- Be open-minded; you do not know it all
- Be accepting of diverse research methods: there is no single way of doing research or writing a paper
- Be constructive, not destructive
- Enable heterogeneity, but **not** double standards

Do not block or delay scientific progress for non-reasons

Suggestion to Community

We Need to Fix the Reviewer Accountability Problem

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

Runahead Execution

A Short Retrospective

Onur Mutlu, Jared Stark, Chris Wilkerson, Yale Patt

HPCA 2021 ToT Award Talk

2 March 2021

Backup Slides

Citation for the ToT Award

Runahead Execution is a pioneering paper that opened up new avenues in dynamic prefetching. The basic idea of run ahead 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.

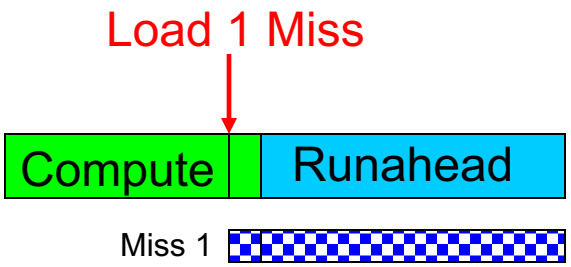
More on Runahead Execution

- 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.

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

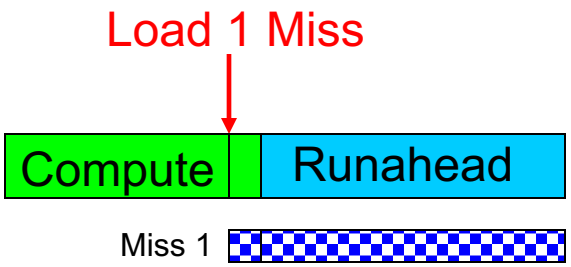
Instruction Processing in Runahead Mode



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

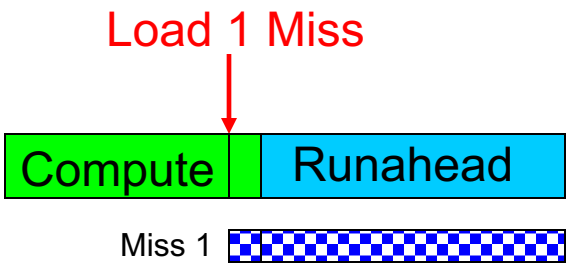
- It is purely speculative: Architectural (software-visible) register/memory state is NOT updated in runahead mode.
- L2-miss dependent instructions are identified and treated specially.
  - They are quickly removed from the instruction window.
  - Their results are not trusted.

L2-Miss Dependent Instructions



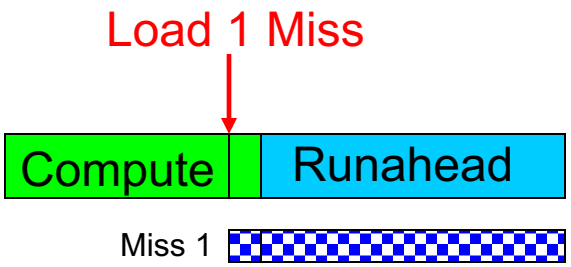
- Two types of results produced: INV and VALID
- INV = Dependent on an L2 miss
- INV results are marked using INV bits in the register file and store buffer.
- INV values are not used for prefetching/branch resolution.

Removal of Instructions from Window



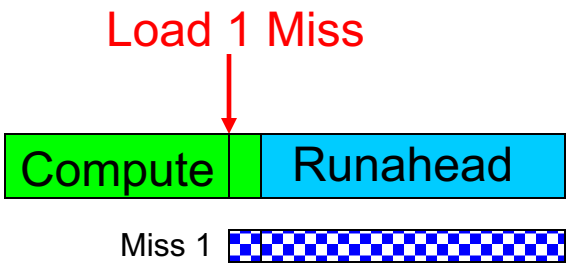
- Oldest instruction is examined for pseudo-retirement
  - An INV instruction is removed from window immediately.
  - A VALID instruction is removed when it completes execution.
- Pseudo-retired instructions free their allocated resources.
  - This allows the processing of later instructions.
- Pseudo-retired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

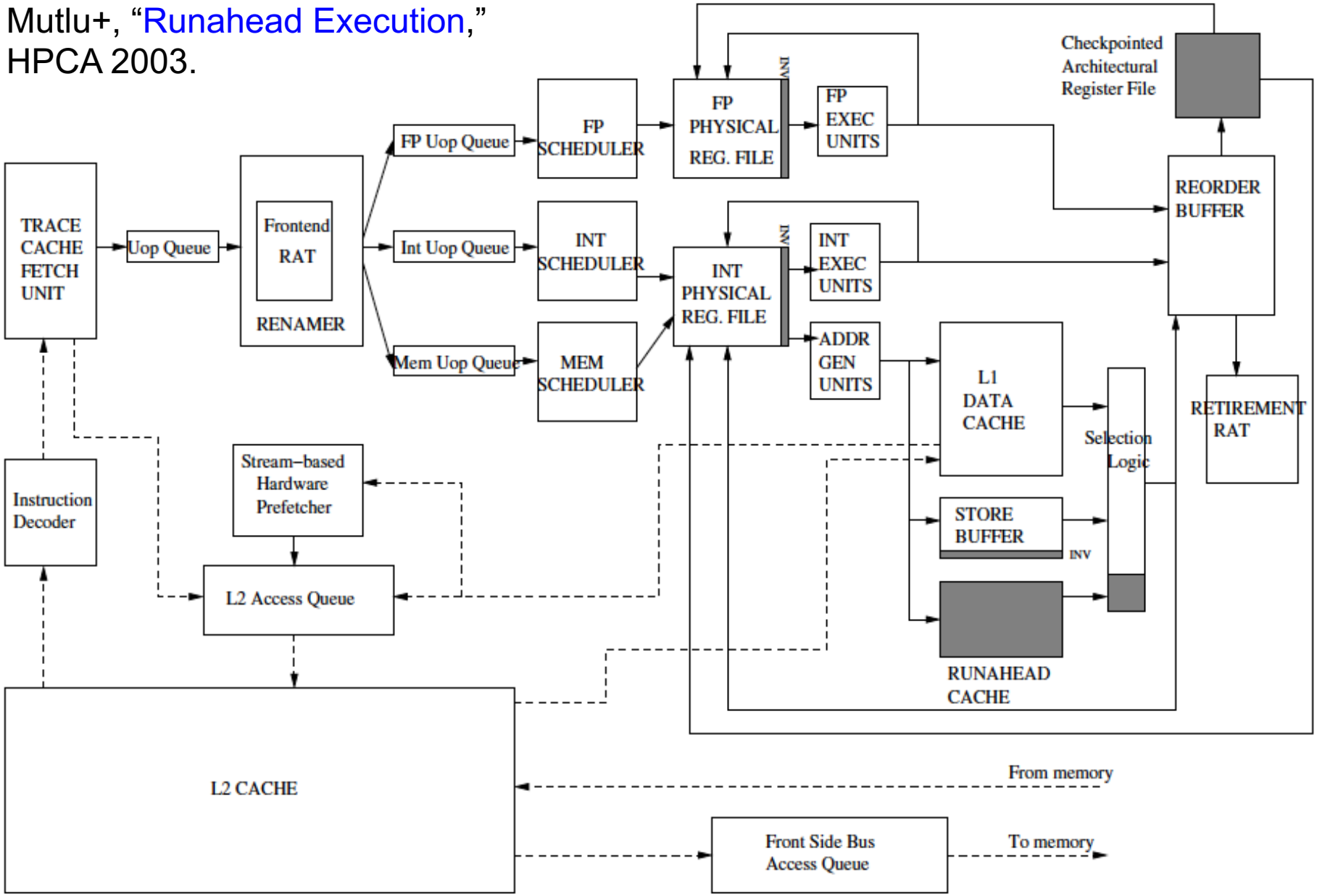
Branch Handling in Runahead Mode



- 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

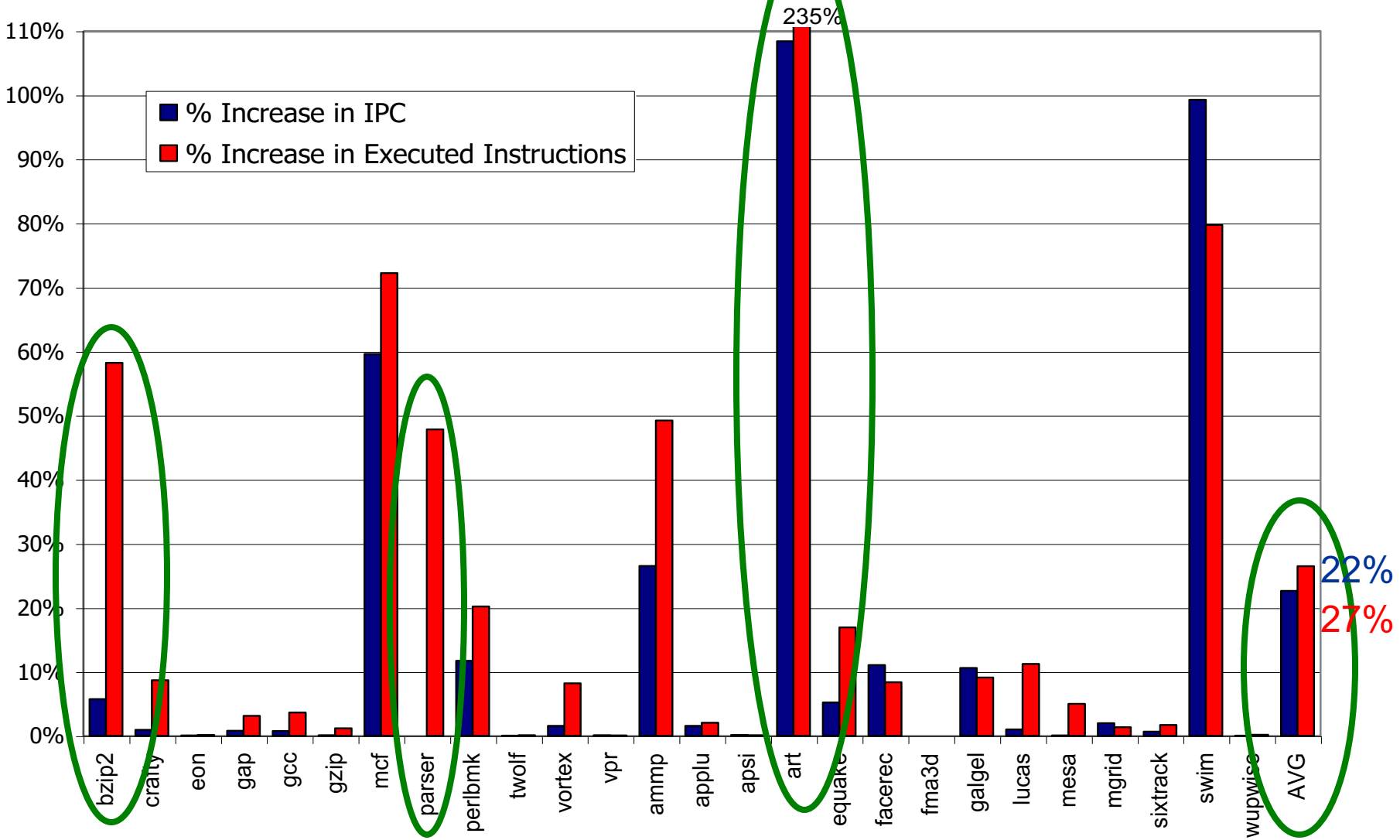
Mutlu+, "Runahead Execution," HPCA 2003.



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

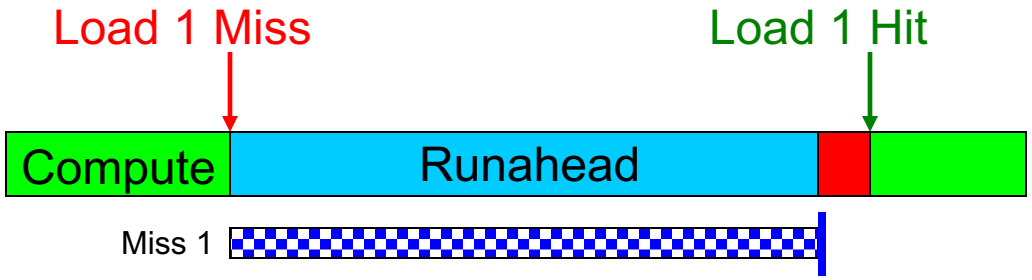
Overlapping Runahead Periods

- Two runahead periods that execute the same instructions

- Second period is inefficient

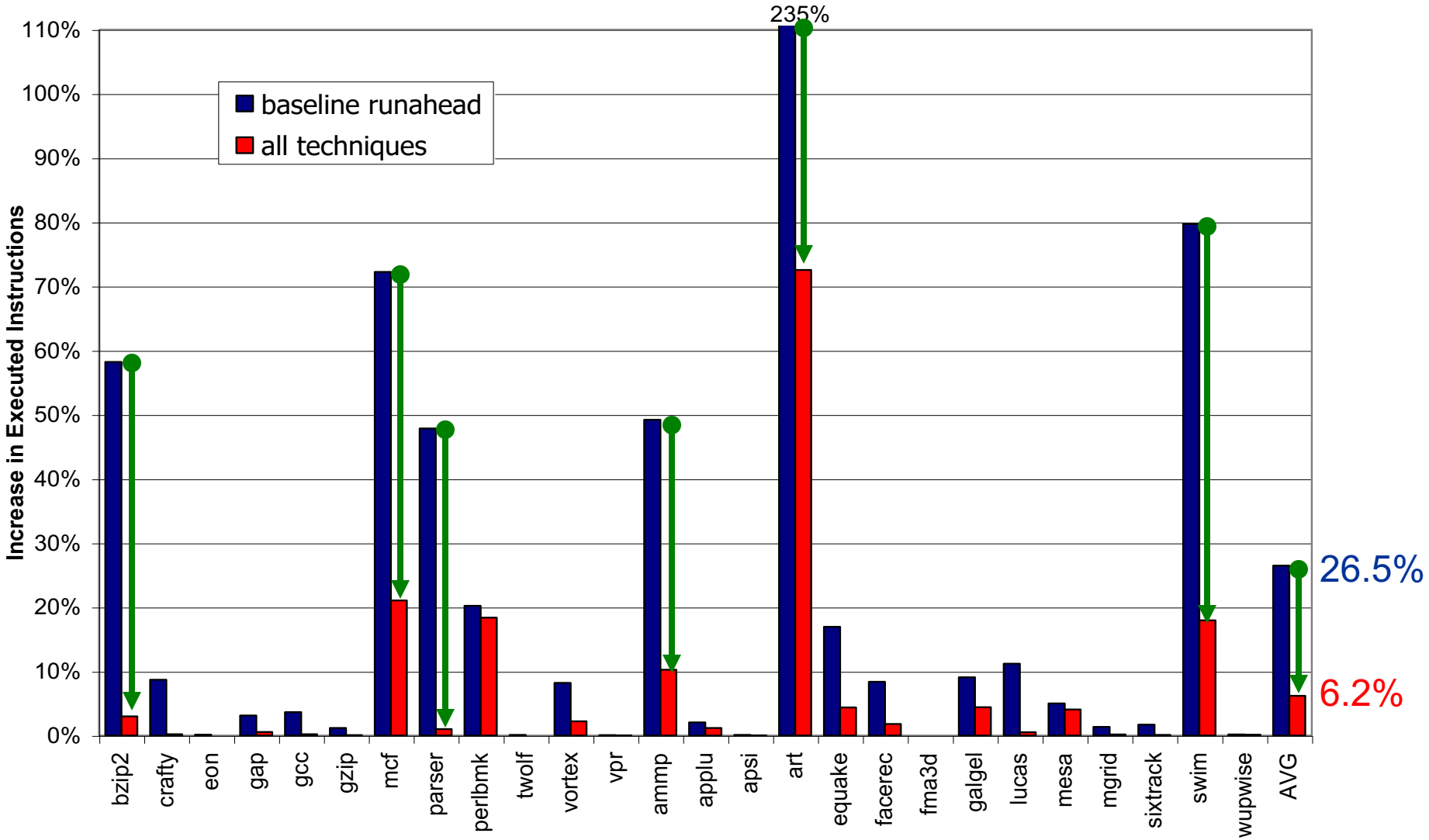
Useless Runahead Periods

- Periods that do not result in prefetches for normal mode

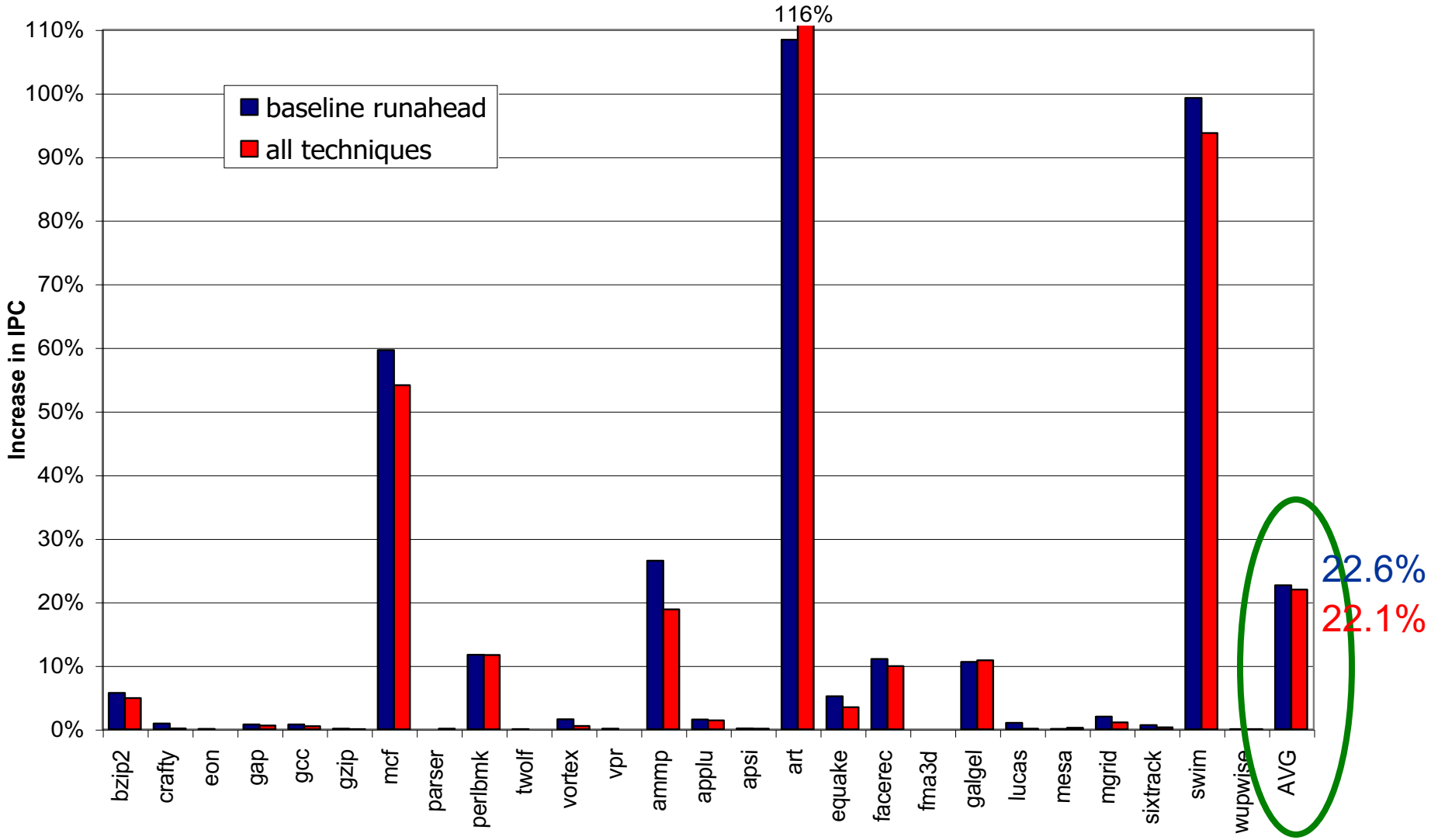


- They exist due to the lack of memory-level parallelism
- Mechanism to eliminate useless periods:
  - Predict if a period will generate useful L2 misses
  - Estimate a period to be useful if it generated an L2 miss that cannot be captured by the instruction window
    - Useless period predictors are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

Overall Impact on Executed Instructions

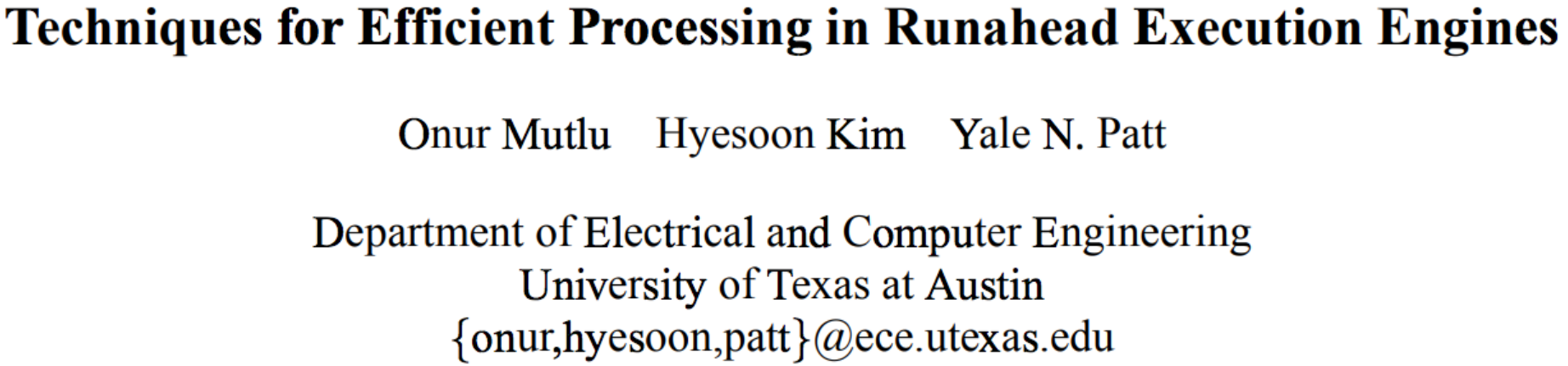


Overall Impact on IPC



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

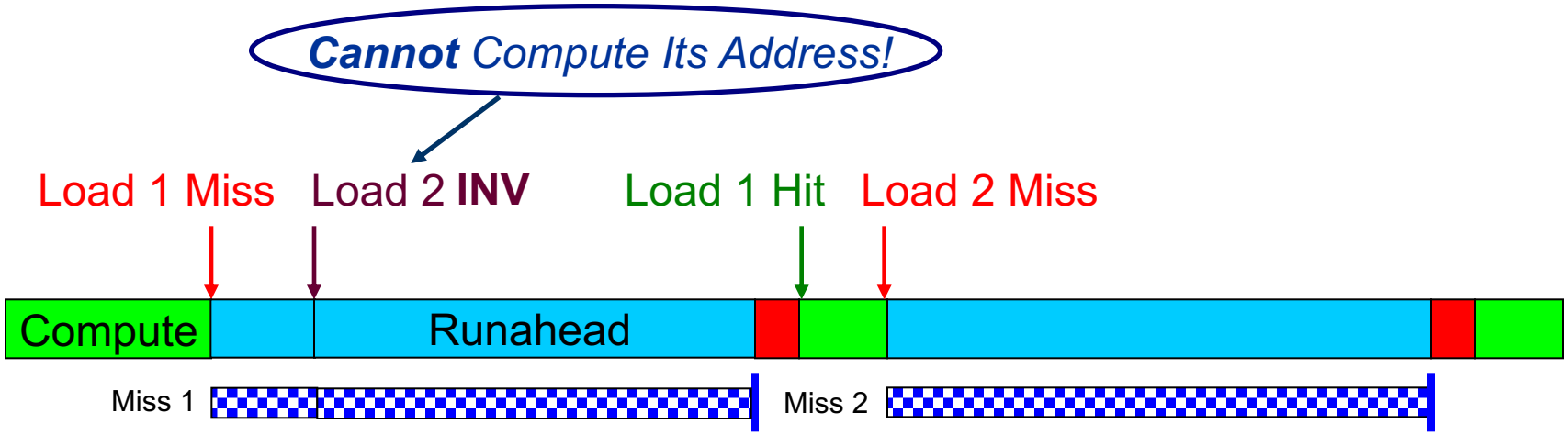
- Onur Mutlu, Hyesoon Kim, and Yale N. Patt,
  "Efficient Runahead Execution: Power-Efficient Memory Latency Tolerance"
  IEEE Micro, Special Issue: Micro's Top Picks from Microarchitecture Conferences (***MICRO TOP PICKS***), Vol. 26, No. 1, pages 10-20, January/February 2006.

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

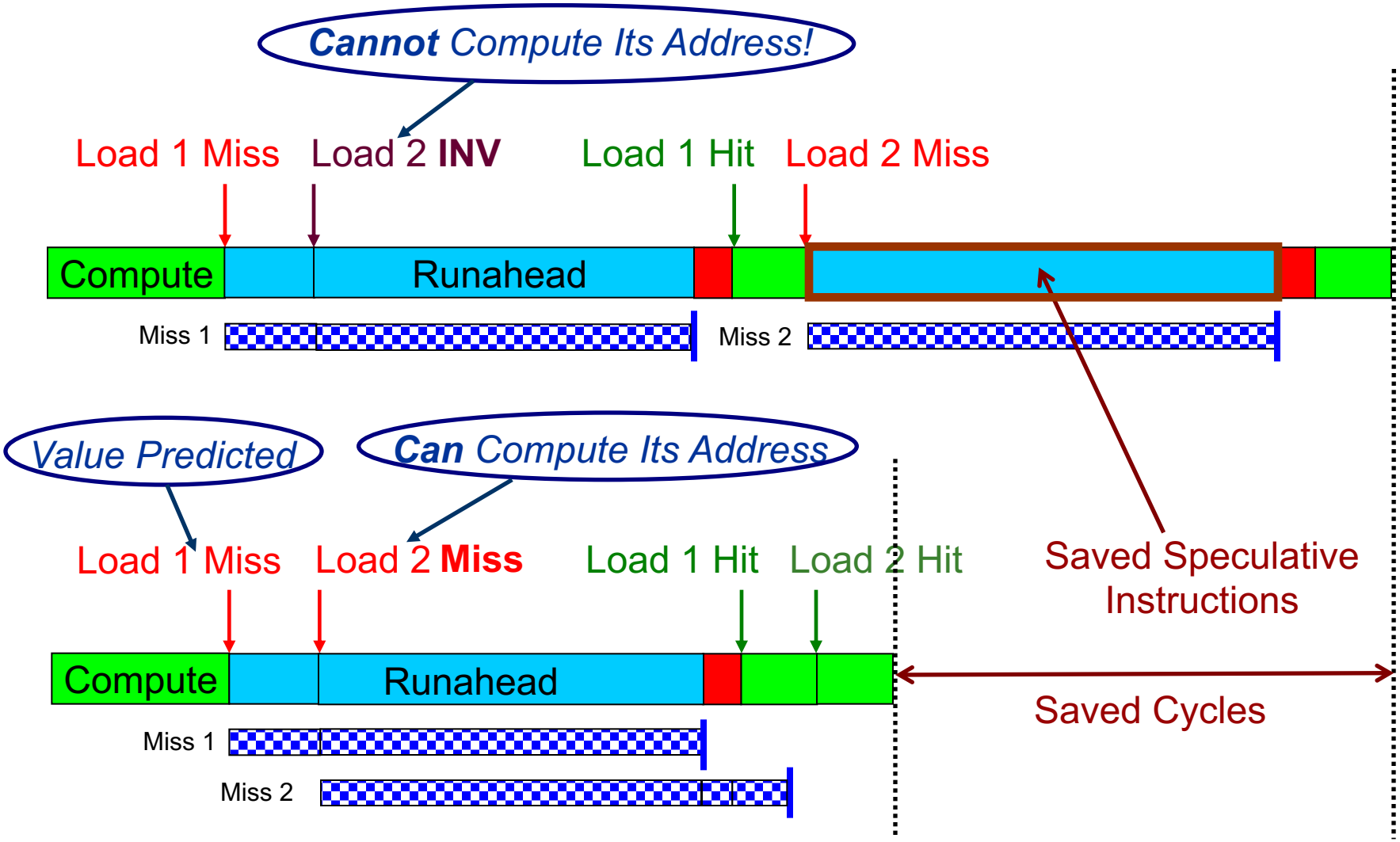
Parallelizing Dependent Cache Misses

- 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.

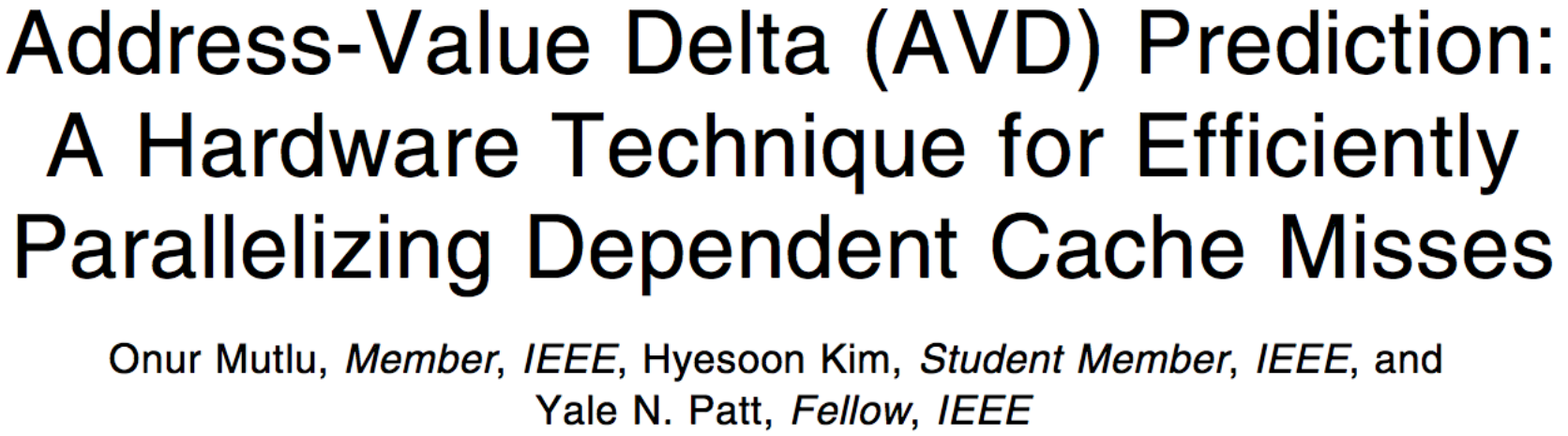
Address-Value Delta (AVD) Prediction: Increasing the Effectiveness of Runahead Execution by Exploiting Regular Memory Allocation Patterns

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

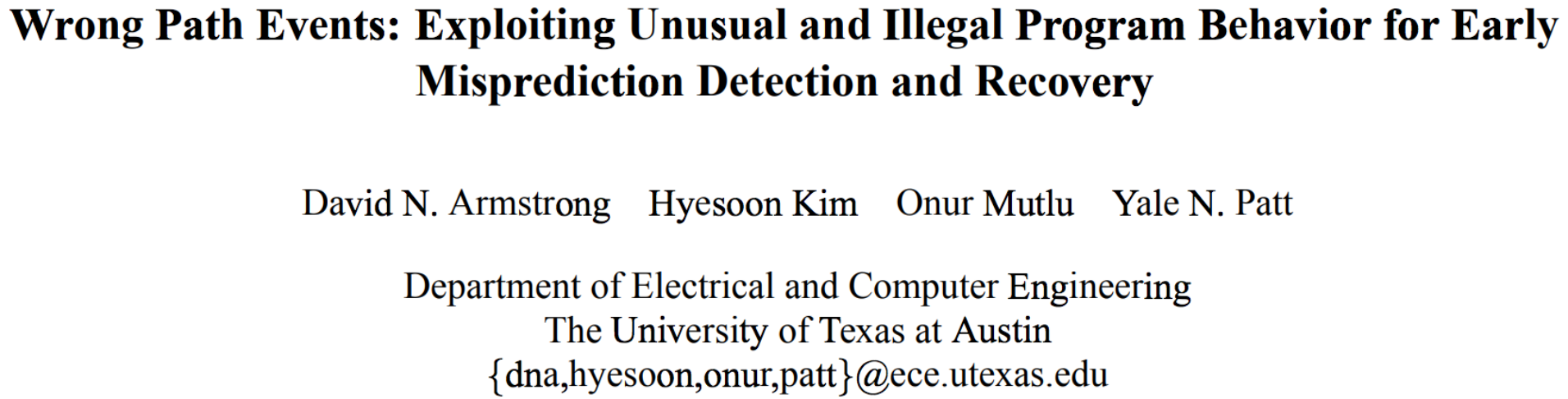
Onur Mutlu, *Member*, *IEEE*, Hyesoon Kim, *Student Member*, *IEEE*, and Yale N. Patt, *Fellow*, *IEEE*

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"** *Proceeedings 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)

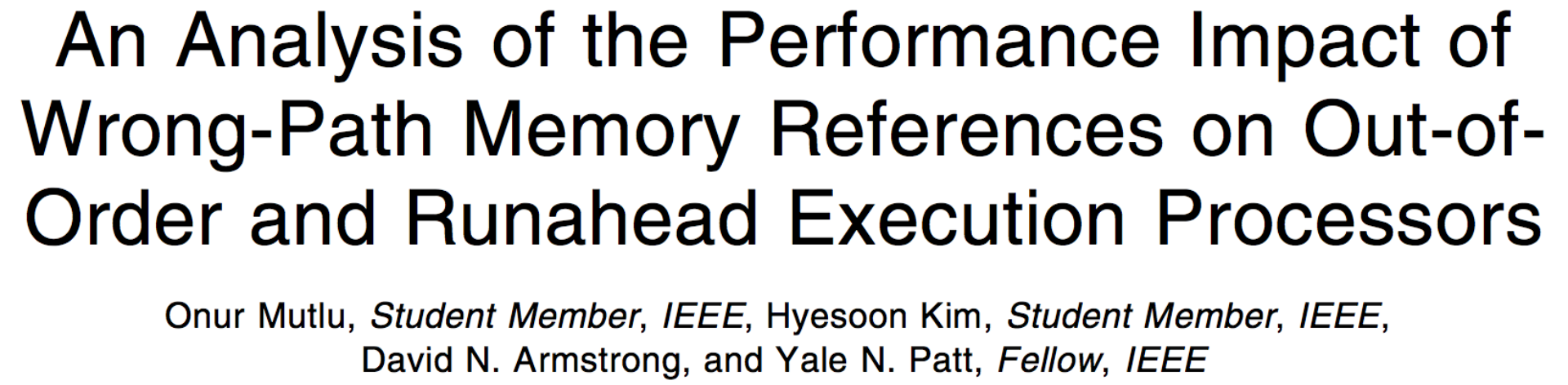
Understanding The Effects of Wrong-Path Memory References on Processor Performance

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*