An Improved Analytical Superscalar Microprocessor Memory Model

Xi Chen and Tor Aamodt

Electrical and Computer Engineering
University of British Columbia

June 22nd, 2008 (MoBS-2008)

Outline

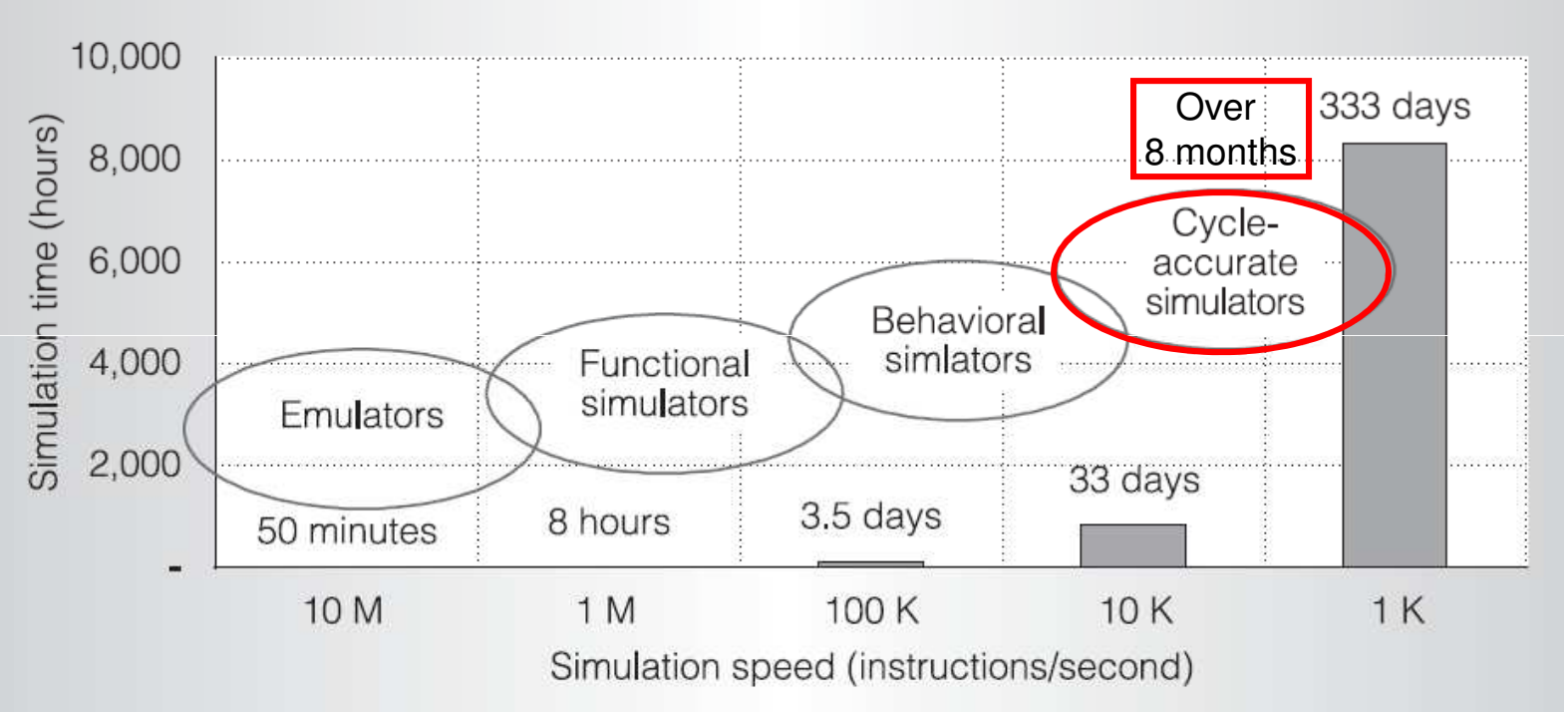
- Introduction
- Background
- Modeling Long Latency Memory Systems
  - Pending Hits
  - Accurate Hidden Miss Latency Estimation
  - A Limited Number of Outstanding Cache Misses
  - SWAM & SWAM-MLP
- Methodology
- Results
- Future Work and Conclusions

Introduction

- Processor design is a complicated task
- Cycle-accurate performance simulators are extensively used to explore design space
  - Creating and debugging a simulator takes time
  - Running detailed simulations is slow

Introduction

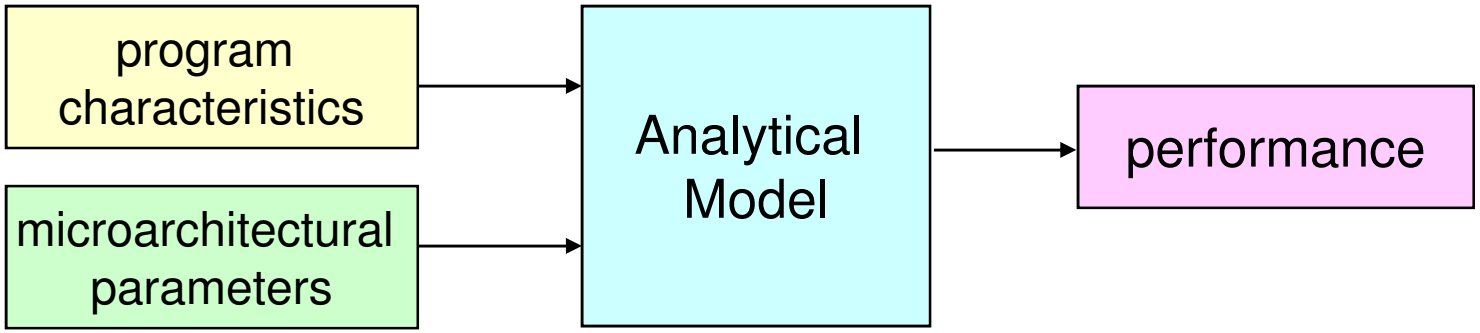
- Simulating future large-scale CMPs ?



Time to simulate 1 second of LCMP (32-core, 4-thread per core) workload execution (Li Zhao et al., *Exploring Large-Scale CMP Architectures Using ManySim*, IEEE Micro, Issue 4, 2007)

Introduction

- Analytical Modeling
  - an alternative to cycle-accurate simulations

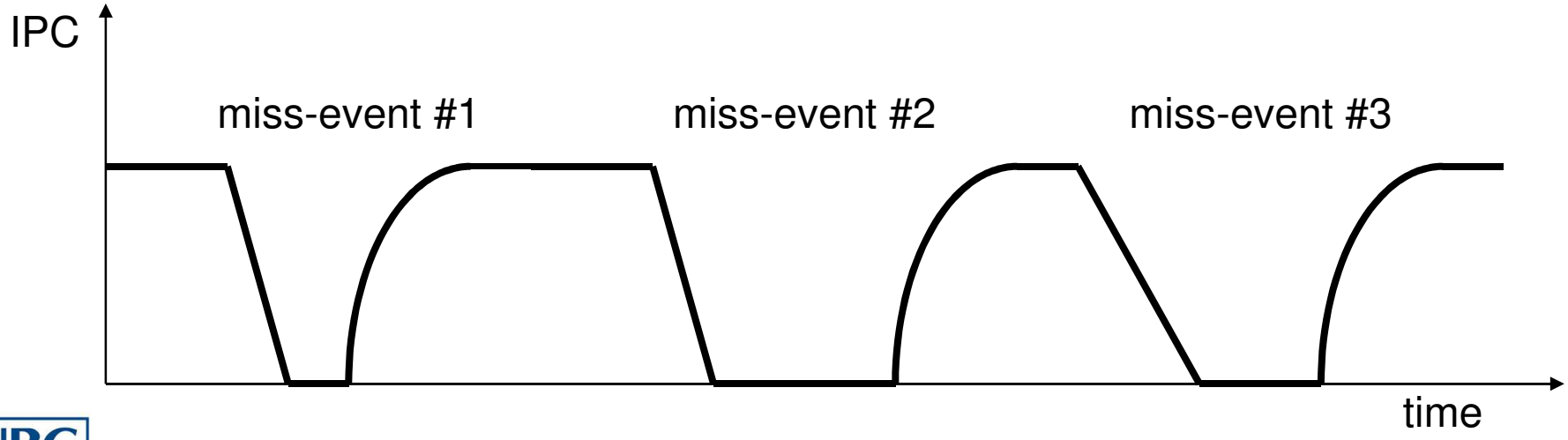


Introduction

- Analytical Modeling vs. Performance Simulations
  - Pros
    - Fast speed (orders of magnitude times faster)
    - Providing more insights for chip designers
  - Cons
    - Less accurate than performance simulations
    - Covering only major microarchitectural parameters

Background

- First-order Model
  - Tejas Karkhanis and James Smith. *A First-order Superscalar Processor Model*. ISCA'04.
  - Stable performance is disrupted by different types of miss-events



Background

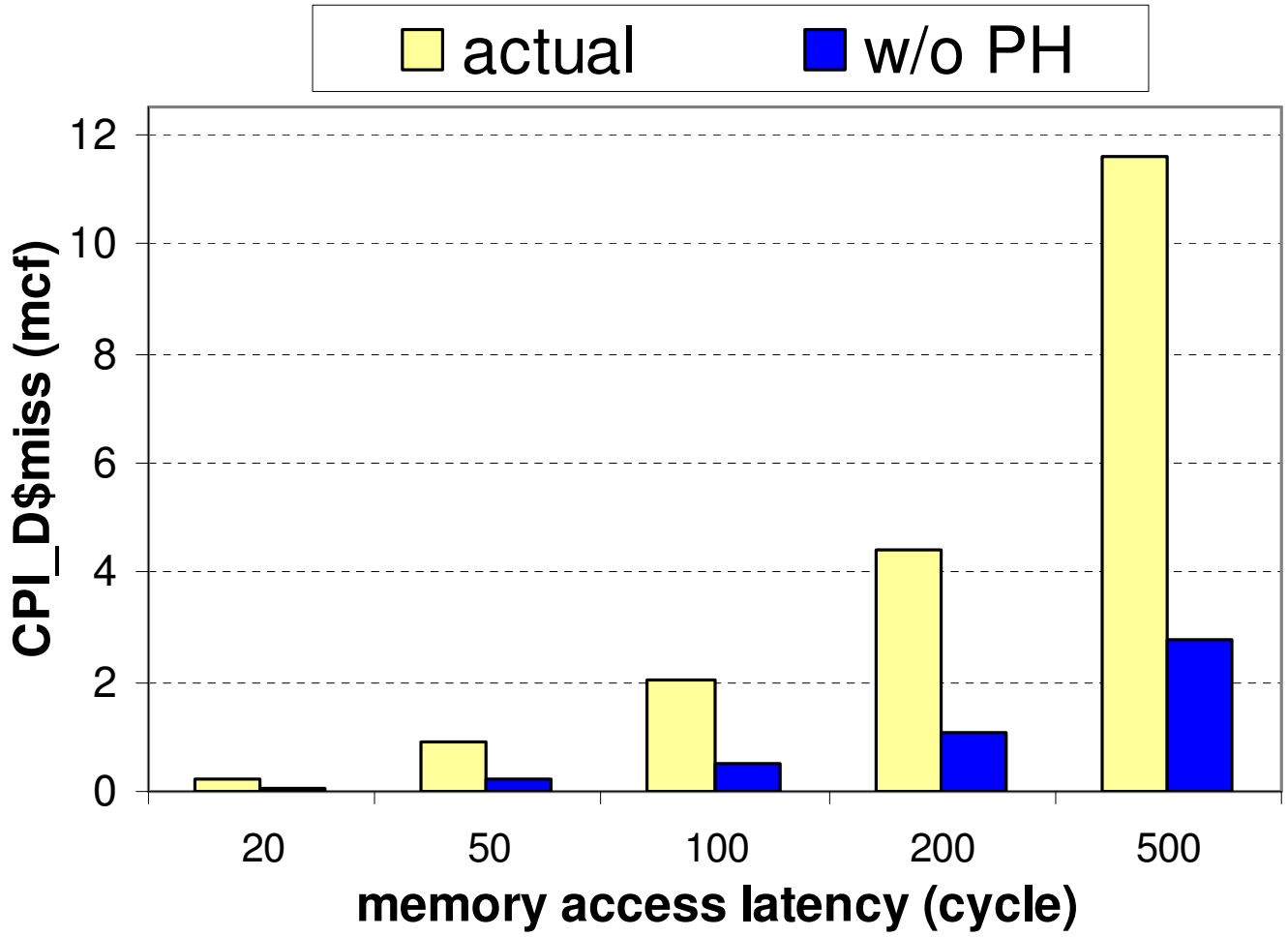
- First-order Model
  - The overall performance (CPI) is modeled by the sum of isolated CPI components due to each type of miss-event.

$$CPI_{modeled} = CPI_{base} + CPI_{bmsp} + CPI_{D\$miss} + CPI_{I\$miss}$$

- CPI_{base}: in absence of miss-events
- CPI_{bmsp}: branch mispredictions
- $CPI_{D\$miss}$: data cache misses
- $CPI_{I\$miss}$: instruction cache misses

  - The **baseline** in our paper is our careful re-implementation of the first-order model based upon the available details described in the ISCA'04 paper and its follow-up work.

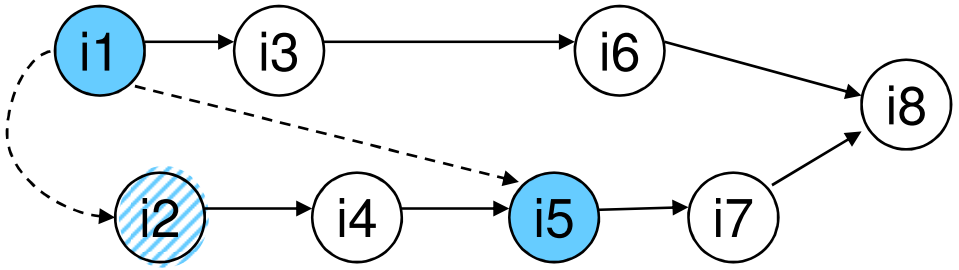
Pending Data Cache Hits



Contributions

- Pending Hits

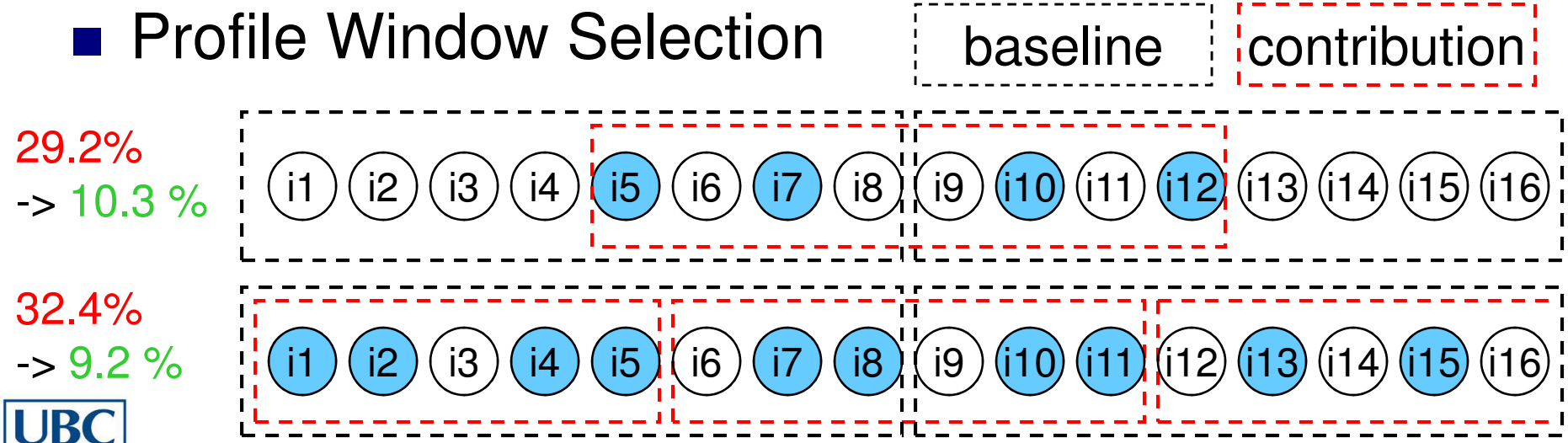
  error is reduced
  43.5% -> 27.5 %



- Accurate Hidden Miss Latency Estimation

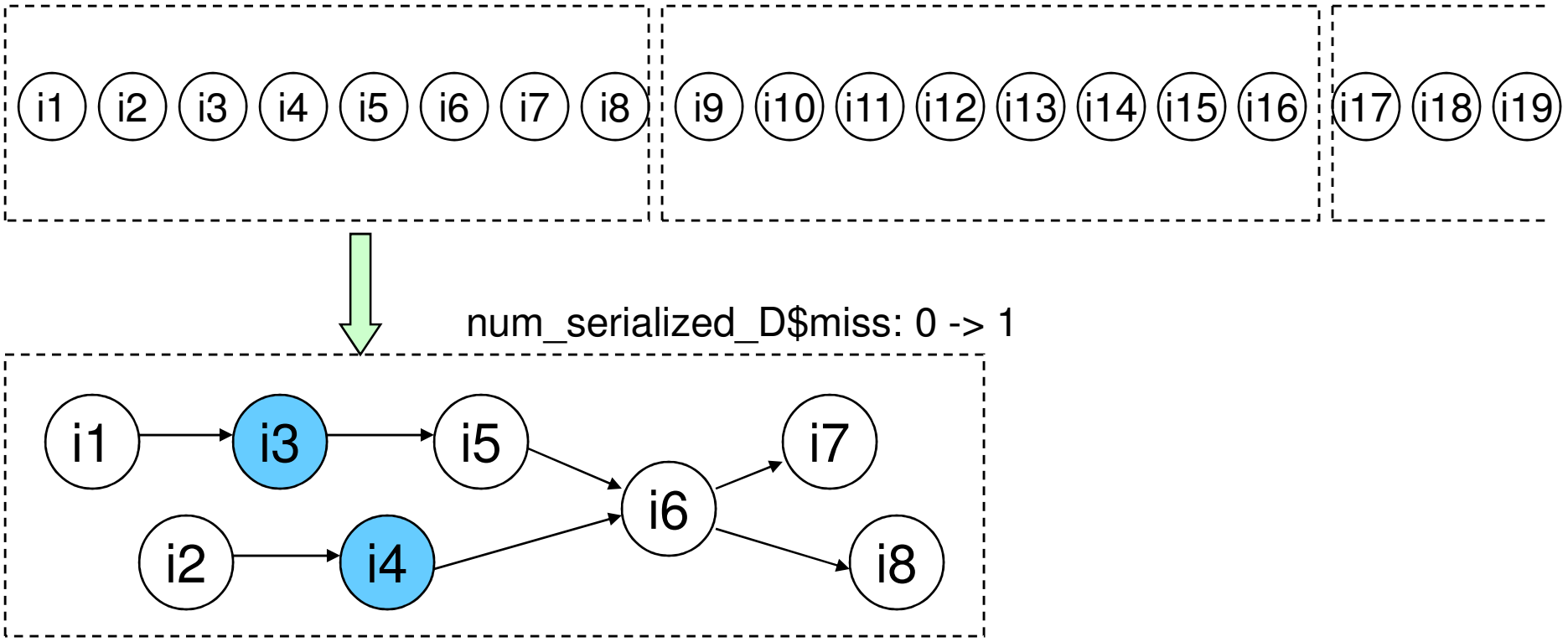
  15.5% -> 10.3 %

- Profile Window Selection



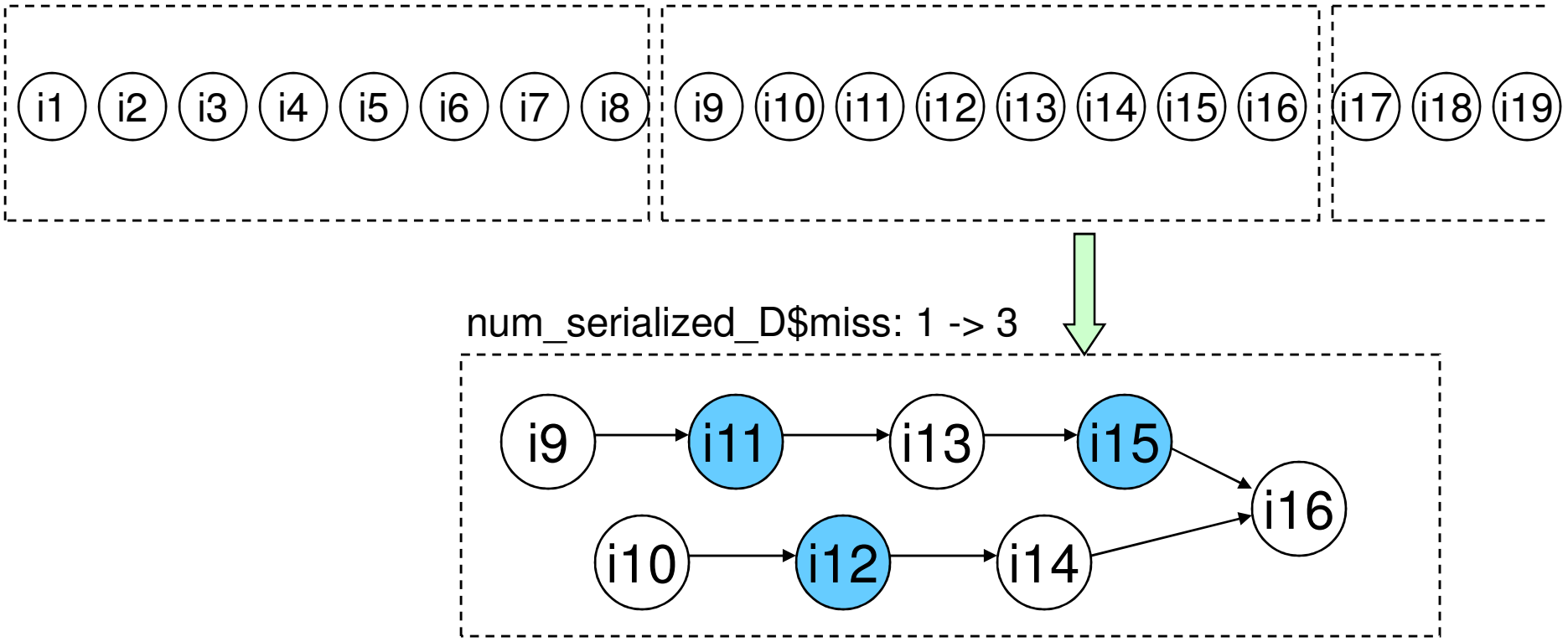
Modeling $CPI_{D\$miss}$: **Baseline**

- Assuming the instruction window size is eight



Modeling $CPI_{D\$miss}$: **Baseline**

- Assuming the instruction window size is eight



Modeling $CPI_{D\$miss}$: **Baseline**

$$CPI_{D\$miss} = \frac{num\_serialized\_D\$miss \times mem\_lat}{total\_num\_instructions}$$

- *mem_lat*: main memory latency
- *total_num_instructions*: total number of instructions committed

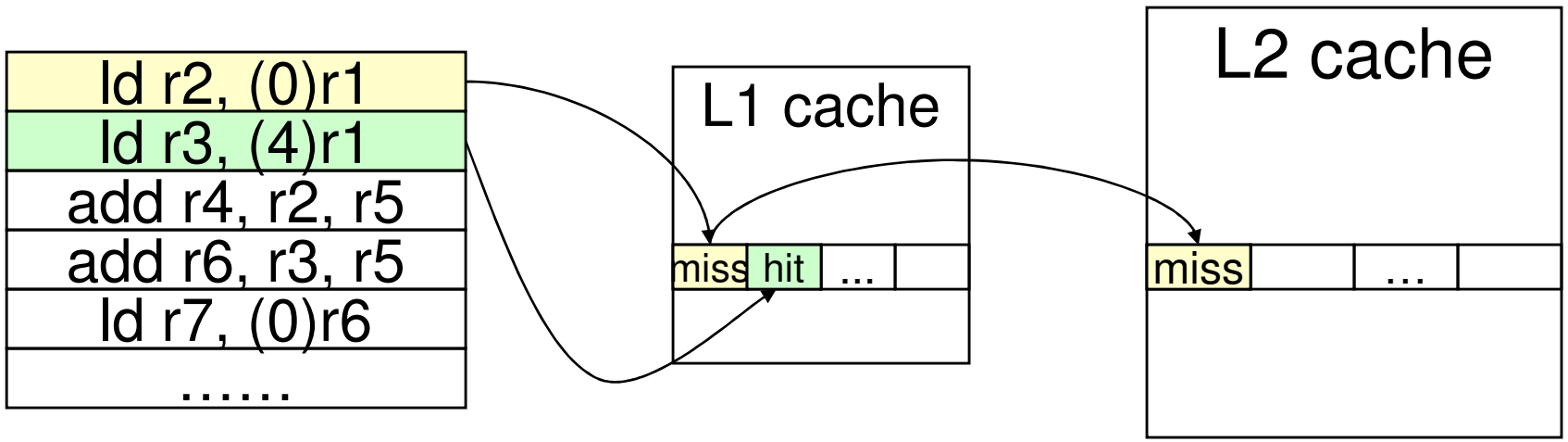
Pending Data Cache Hits

- A pending data cache hit results from a memory reference to a cache block for which a request has already been initiated by another instruction

- Pending data cache hits are common due to spatial locality in applications

Pending Data Cache Hits

- A (traditional) cache simulator cannot identify pending cache hits due to the lack of timing information

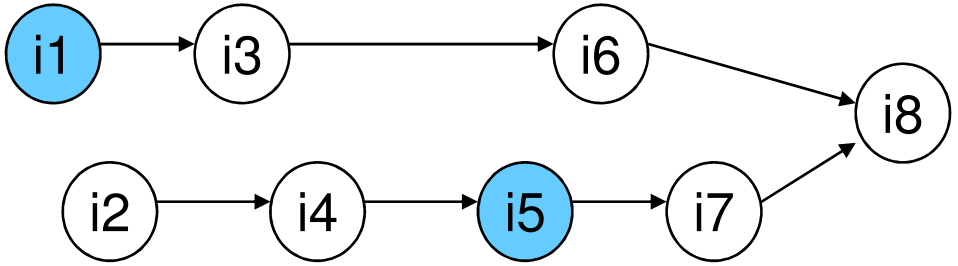


Pending Data Cache Hits

i1	ld r2, (0)r1
i2	ld r3, (4)r1
i3	add r4, r2, r5
i4	add r6, r3, r5
i5	ld r7, (0)r6
i6	or r8, r4, r5
i7	add r9, r5, r7
i8	or r10, r8, r9
i9	……

not considering pending hits (num_serialized_D$miss += **1**)

wrong value

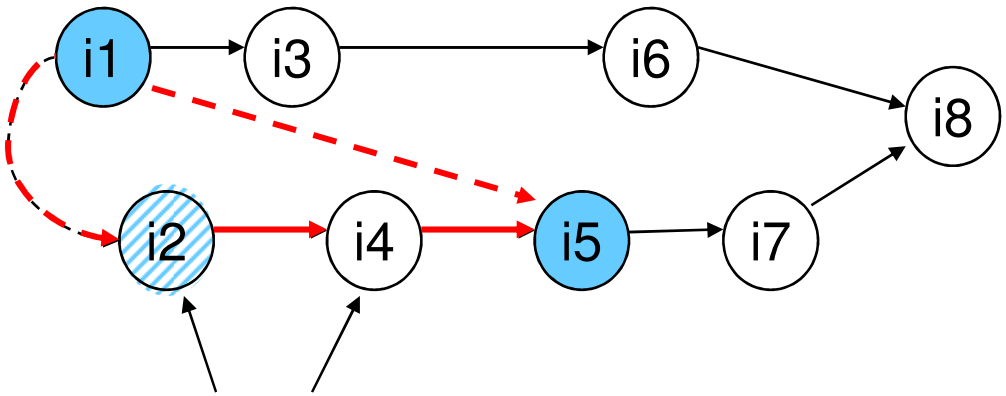


Modeling Pending Hits

i1	ld r2, (0)r1	miss
i2	ld r3, (4)r1	hit **(i1)**
i3	add r4, r2, r5	
i4	add r6, r3, r5	
i5	ld r7, (0)r6	miss
i6	or r8, r4, r5	
i7	add r9, r5, r7	
i8	or r10, r8, r9	
i9	……	

considering pending hits (num_serialized_D$miss += **2**)

correct value

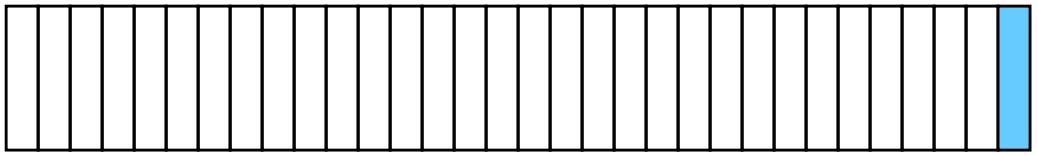


latency modeled by CPI_{base}

Error is reduced from **43.5%** to **27.5 %** with best fixed cycle compensation for miss latency

Compensating Overestimate [K&S ISCA'04, Karkhanis PhD Thesis]

- Part of the latency of a miss may be hidden



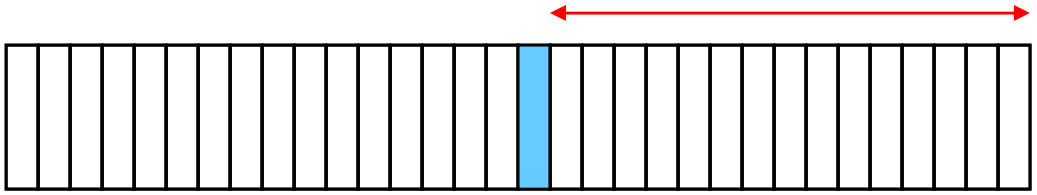
fetch ROB commit

when a miss issues, it can be (or near) the commit side of the ROB

$$CPI_{D\$miss} = \frac{num\_serialized\_D\$miss \times mem\_lat}{total\_num\_instructions}$$

Compensating Overestimate [K&S ISCA'04, Karkhanis PhD Thesis]

- Part of the latency of a miss may be hidden



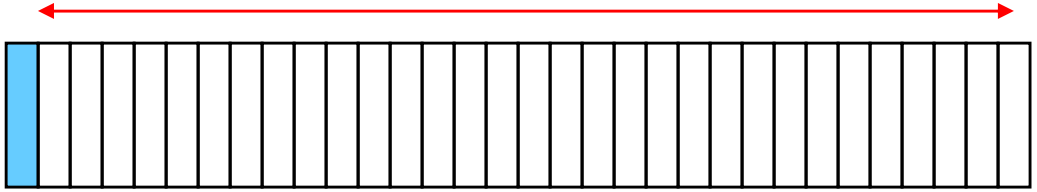
fetch ROB commit

when a miss issues, it can be (or near) the middle of the ROB

$$CPI_{D\$miss} = \frac{num\_serialized\_D\$miss \times (mem\_lat - \frac{ROB_{size}}{2 \times issue\_width})}{total\_num\_instructions}$$

Compensating Overestimate [K&S ISCA'04, Karkhanis PhD Thesis]

- Part of the latency of a miss may be hidden



fetch ROB commit

when a miss issues, it can be (or near) the fetch side of the ROB

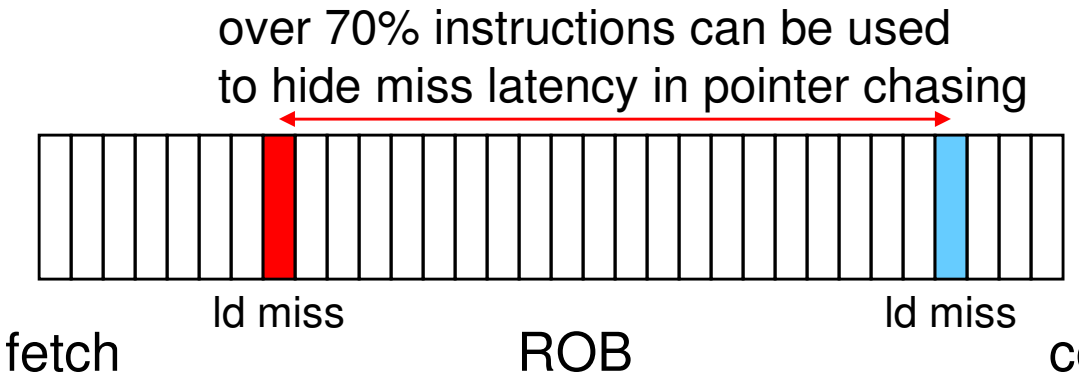
$$CPI_{D\$miss} = \frac{num\_serialized\_D\$miss \times (mem\_lat - \frac{ROB_{size}}{issue\_width})}{total\_num\_instructions}$$

Compensating Overestimate

- Fixed-cycle compensation used by prior work is not accurate for all the benchmarks we study.

- We propose to compensate the overestimate using the average distance between consecutive misses.

Compensating Overestimate



$$CPI_{D\$miss} = \frac{num\_serialized\_D\$miss \times D\$miss\_penalty - \frac{dist}{issuc\_width} \times num\_D\$miss}{total\_num\_instructions}$$

dist: average distance between two consecutive misses (the distance between two misses is saturated by the size of ROB)

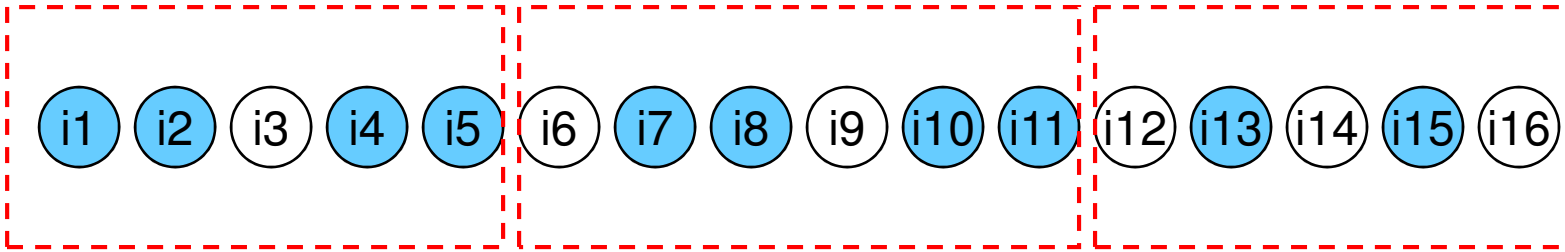
*error is reduced from **15.5%** (best fixed cycle compensation) to **10.3%***

Modeling a limited number of MSHRs

- The model thus far has assumed that the number of outstanding cache misses supported is unlimited.
- We propose a technique to model a limited number of outstanding cache misses supported.

Modeling a limited number of MSHRs

- We stop a profile step and update ***num_serialized_D$miss*** when the number of misses analyzed is equal to the number of Miss Status Holding Registers (MSHRs)

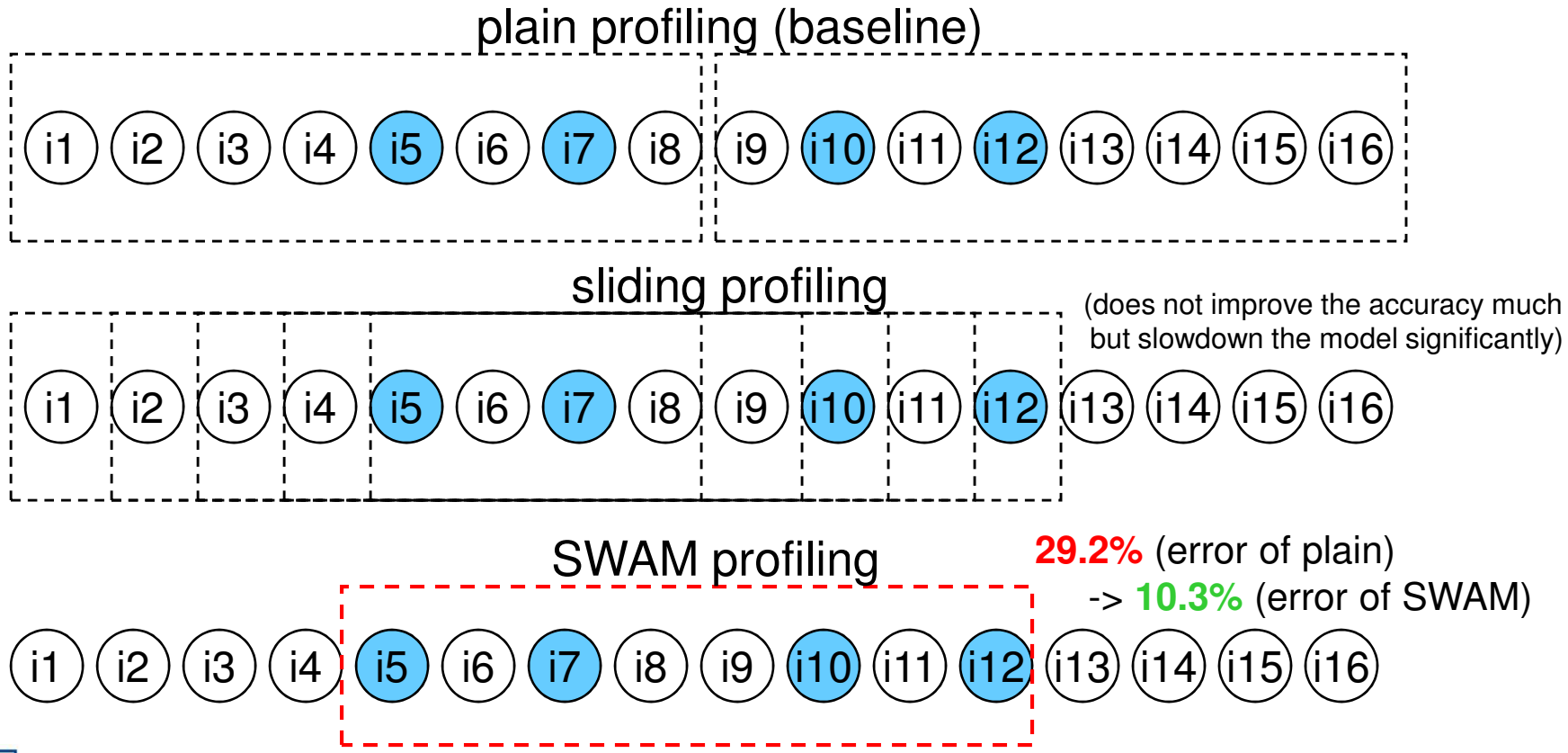


$ROB_{size} = 8, N_{MSHR} = 4$

With plain profiling, error reduces from **32.4%** to **23.9%** for 8 MSHRs

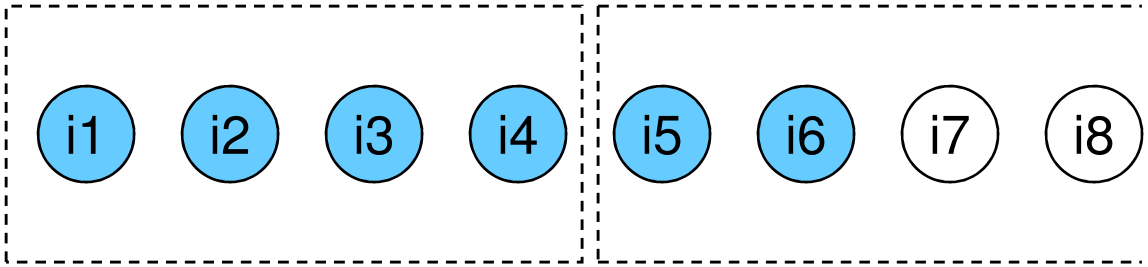
Start-with-a-miss (SWAM) Profiling

- Each profile step starts with a cache miss



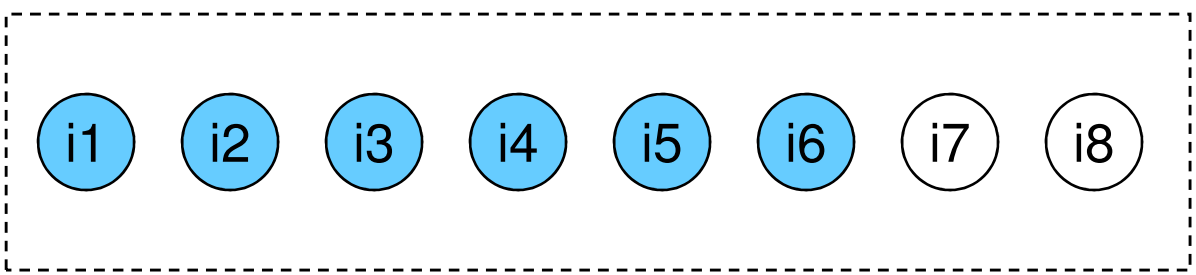
SWAM-MLP

SWAM (ROB_{size} = 8, N_{MSHR} = 4)

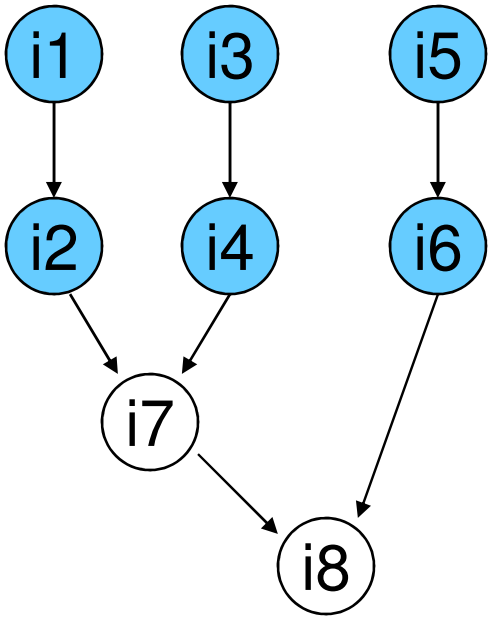


num_serialized_miss += 4 (wrong value)

SWAM-MLP



num_serialized_miss += 2 (correct value)



12.8% (error of SWAM) to **9.2%** (error of SWAM-MLP) for 8 MSHRs

23.2% (error of SWAM) to **9.9%** (error of SWAM-MLP) for 4 MSHRs

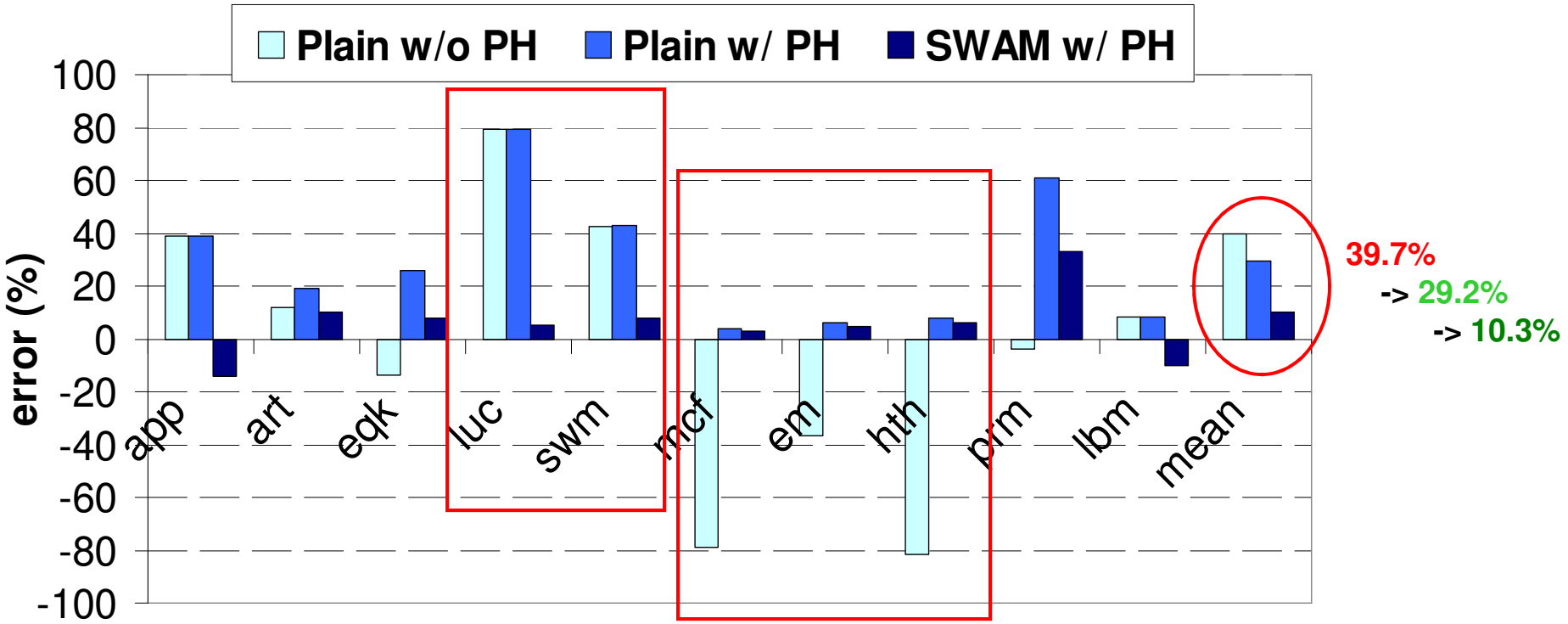
Methodology

- We modified SimpleScalar and used a set of memory intensive benchmarks from SPEC 2000 and OLDEN suite whose cache misses per thousand instructions (MPKI) is higher than 10 for our cache configurations

Machine Width	4
ROB Size	256
LSQ Size	256
Branch Predictor	4KB gShare
L1 I-Cache	16KB, 32B/line, 4-way, 1-cycle latency
L1 D-Cache	16KB, 32B/line, 4-way, 2-cycle latency
L2 Cache	128KB, 64B/line, 8-way, 10-cycle latency
Memory Latency	200 cycles

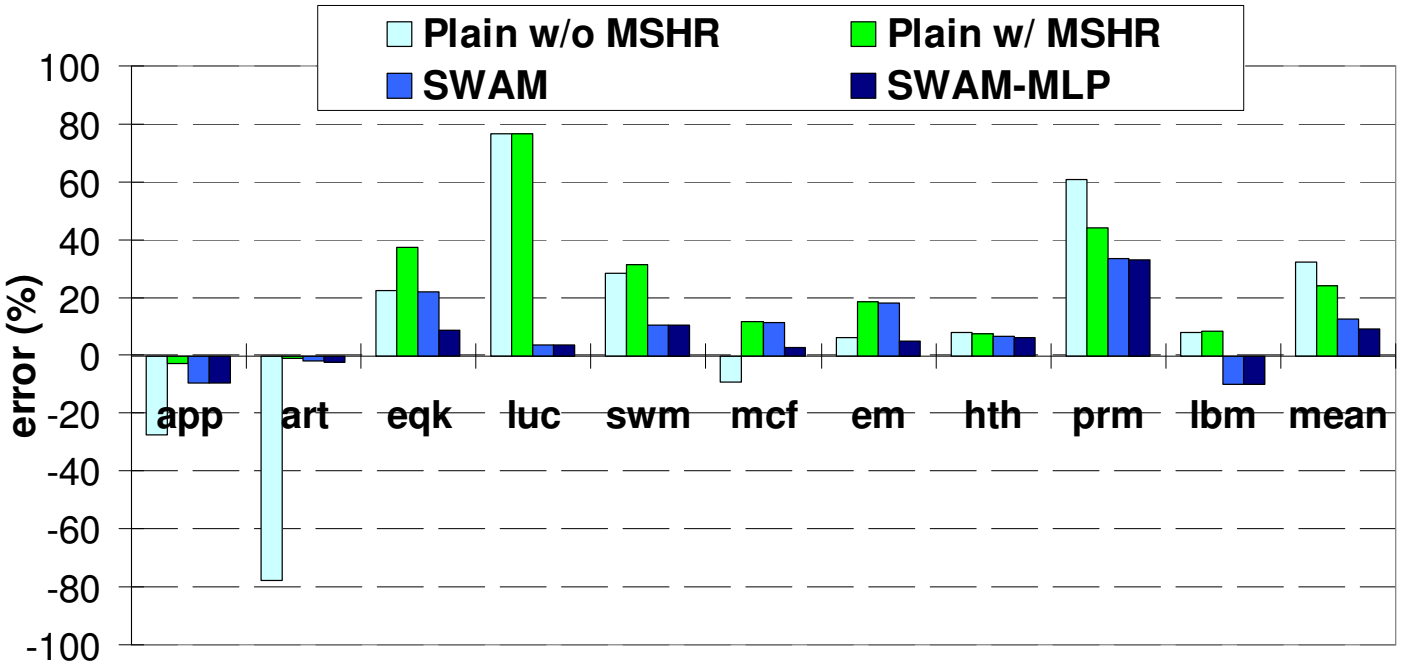
Results

- Unlimited number of outstanding cache misses supported



Results

- Modeling a limited number of MSHRs
  - SWAM-MLP further decreases error of SWAM from 12.8% to 9.2% (8 MSHRs, e.g., Prescott), from 23.2% to 9.9% (4 MSHRs)



- Our improvements do **not** slowdown the first-order model

Current / Future Work

- Analytically modeling the performance impact of hardware data prefetching (by the pending hits caused by data prefetches)
- Analytically modeling the throughput of fine-grain multithreaded microprocessors (e.g., Sun's Niagara)
- Extending the analytical model for CMPs with superscalar, out-of-order execution cores
- Analytically modeling the performance of SMT superscalar cores

Conclusions

- Modeling Long Latency Memory System
  - Pending Data Cache Hit
  - Accurate Hidden Miss Latency Estimation
  - Modeling MSHRs
  - SWAM & SWAM-MLP

- Overall our improvements reduce the error of our baseline from 39.7% to 10.3%. The error is less than 10% when modeling MSHRs. Our improvements do not slow down the first-order model.