

15-740/18-740

Computer Architecture

Lecture 16: Runahead and OoO Wrap-Up

Prof. Onur Mutlu

Carnegie Mellon University

Fall 2011, 10/17/2011

Review Set 9

- Due this Wednesday (October 19)
- Wilkes, “**Slave Memories and Dynamic Storage Allocation,**” IEEE Trans. On Electronic Computers, 1965.
- Jouppi, “**Improving Direct-Mapped Cache Performance by the Addition of a Small Fully-Associative Cache and Prefetch Buffers,**” ISCA 1990.

- Recommended:
 - Hennessy and Patterson, Appendix C.2 and C.3
 - Liptay, “**Structural aspects of the System/360 Model 85 II: the cache,**” IBM Systems Journal, 1968.
 - Qureshi et al., “**A Case for MLP-Aware Cache Replacement,**” ISCA 2006.

Announcements

- Midterm I next Monday
 - October 24
- Exam Review
 - Likely this Friday during class time (October 21)
- Extra Office Hours
 - During the weekend – check with the TAs
- Milestone II
 - Is postponed. Stay tuned.

Course Feedback Analysis (I)

- 24 responses, 26 registered
- Course Pace:
 - Slow: 2
 - Right: 17
 - Fast: 4
 - Too fast: 1
- Lecture pace:
 - Slow: 5
 - Right: 15
 - Fast: 4

Course Feedback Analysis (II)

- Material difficulty:

- Easy: 2
- Right: 17
- Hard: 4
- Depends: 1

- Workload:

- Right: 5
- Heavy: 6.5
- Too heavy: 9

Course Pace Comments

- Fast pace allows the amount of material I would like to see covered.

Lecture Pace Comments

- Tangential discussions make lectures worth coming to instead of just having a rehash of slides
- Fast, but understandable
- Good that some key concepts are repeated twice

Material Difficulty Comments

- Depends on background. Coming from a different university, it is apparent some material is more familiar to those who took comp arch at CMU
- Lectures OK, some papers hard
- Started out with zero knowledge and have understood all concepts

Workload Comments

- Reviews take longer than homeworks ++
- Paper reviews are too intense
- Just the right amount

What to Change Comments?

- Start and finish on time ++
- Have what you write on board in the slides; have written lecture notes (latex'ed) +++++
- Reviews
 - Require only half the readings, let us choose the rest
 - No paper reviews (or no homeworks, not both)
 - Difficult to write reviews for overview papers
 - Fewer paper reviews ++
 - Time consuming
- More homeworks +++
- Some older papers are dry (and you cover them anyway)
- Fast feedback on homeworks +++
- Decrease % of exams on grade

What to Change Comments?

- Lectures
 - Two lectures a week +++
 - Add a recitation per week
- Simulator doesn't have documentation +++
- Include video lectures
- Course expectations high; I feel this is expected of students; nothing to change
- Provide estimates on how long we should be spending on the parts of the course
- Some questions on slides are unanswered

Other Comments

- A lot of other classes overlook classroom interaction; not this one
- Sometimes it is hard to hear questions and question is not repeated
- Go more in depth into topics
- The amount of work exceeds what I would expect from a 12 unit course ++
- Please don't murder me with the exam
- Lectures make sense even w/o comp arch background
- I am learning a lot. Very interesting. Write a textbook!

Action on Course Feedback

- Will have fewer lectures toward the end of the course
 - To enable you to focus on projects
- Will try to start and finish on time
- Will take into account other feedback as I described
- Will not murder you on the exam
 - Assuming you understand the material!

Last Lecture

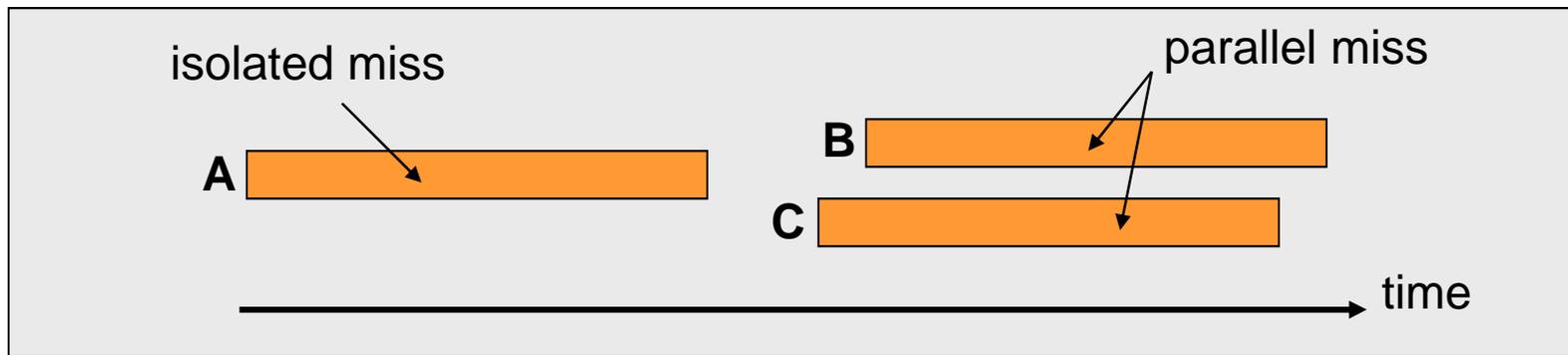
- Limitations of baseline runahead execution mechanism
- Memory-level parallelism
- Memory latency tolerance techniques
- Runahead execution performance
 - vs. Large instruction windows
- Limitations of baseline runahead execution
- Wrong path events
- Causes of inefficiency in runahead execution
- Address-value delta (AVD) prediction

Today

- Dual-core execution
- Load store handling in out-of-order versus runahead execution
- Research issues in out-of-order execution or latency tolerance
- Accelerated critical sections
- Caching, potentially

Review: Memory Level Parallelism (MLP)

- Idea: Find and service multiple cache misses in parallel
- Why generate multiple misses?



- Enables latency tolerance: **overlaps latency of different misses**
- How to generate multiple misses?
 - Out-of-order execution, multithreading, runahead, prefetching

Review: Memory Latency Tolerance Techniques

- Caching [initially by Wilkes, 1965]
 - Widely used, simple, effective, but inefficient, passive
 - Not all applications/phases exhibit temporal or spatial locality
- Prefetching [initially in IBM 360/91, 1967]
 - Works well for regular memory access patterns
 - Prefetching irregular access patterns is difficult, inaccurate, and hardware-intensive
- Multithreading [initially in CDC 6600, 1964]
 - Works well if there are multiple threads
 - Improving single thread performance using multithreading hardware is an ongoing research effort
- Out-of-order execution [initially by Tomasulo, 1967]
 - Tolerates cache misses that cannot be prefetched
 - Requires extensive hardware resources for tolerating long latencies

Review: 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, IEEE TC'06]

■ Irresolvable branch mispredictions in runahead mode

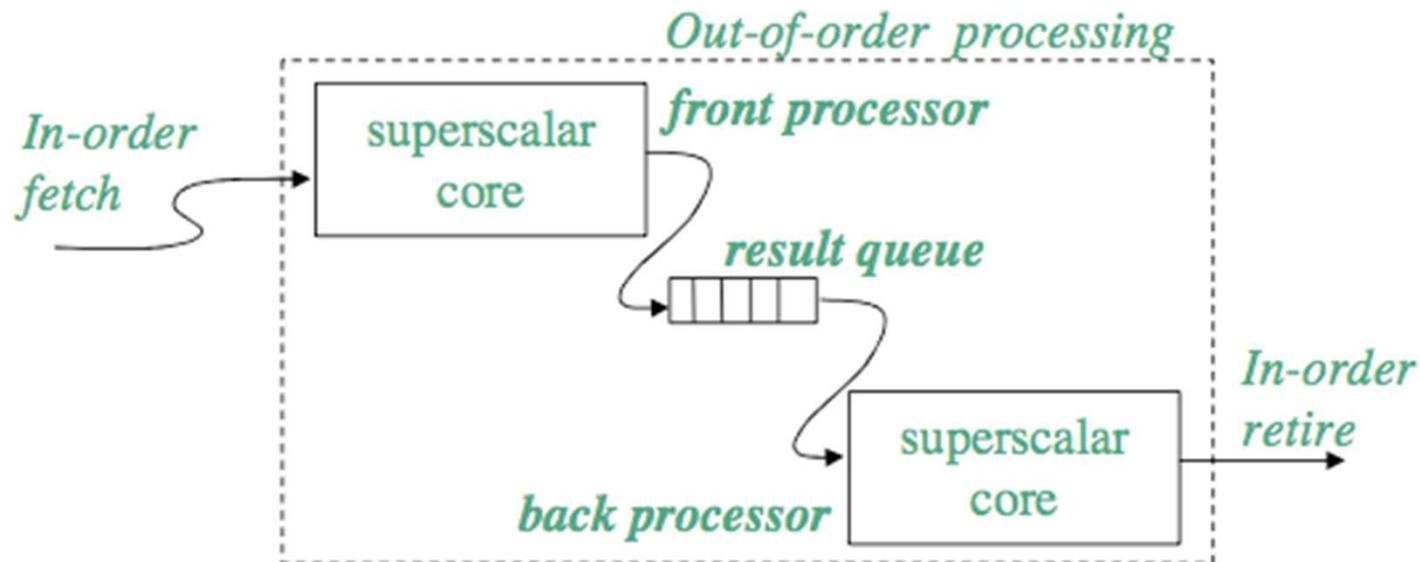
- Cannot recover from a mispredicted L2-miss dependent branch
 - **Wrong Path Events** [MICRO' 04]
-

Other Limitations of Runahead

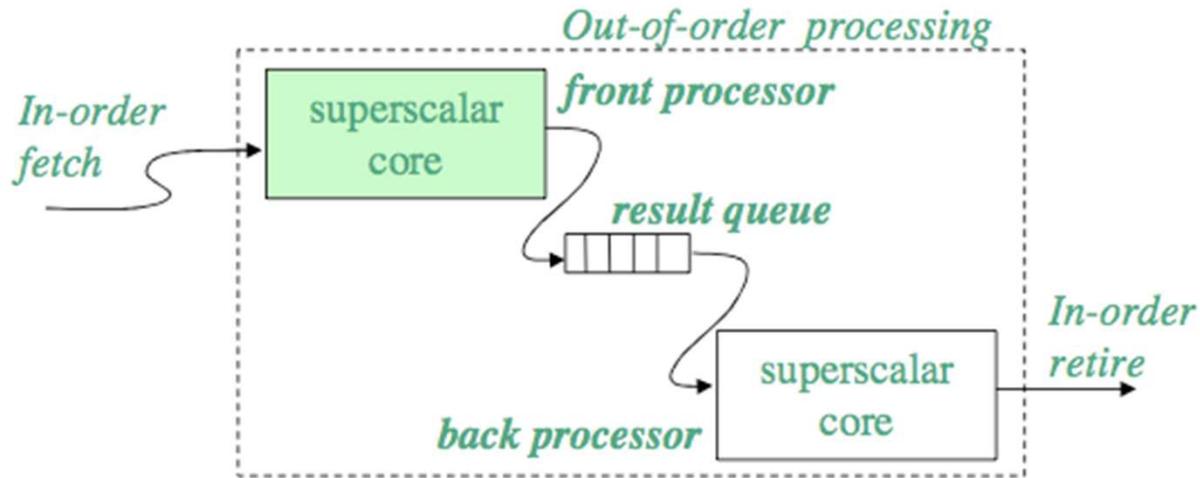
- What are the best instructions to execute during runahead mode?
- When to end runahead mode? What if the L2 misses are far apart from each other?

Dual Core Execution

- Idea: One thread context speculatively runs ahead on load misses and prefetches data for another thread context
- Zhou, “Dual-Core Execution: Building a Highly Scalable Single- Thread Instruction Window,” PACT 2005.

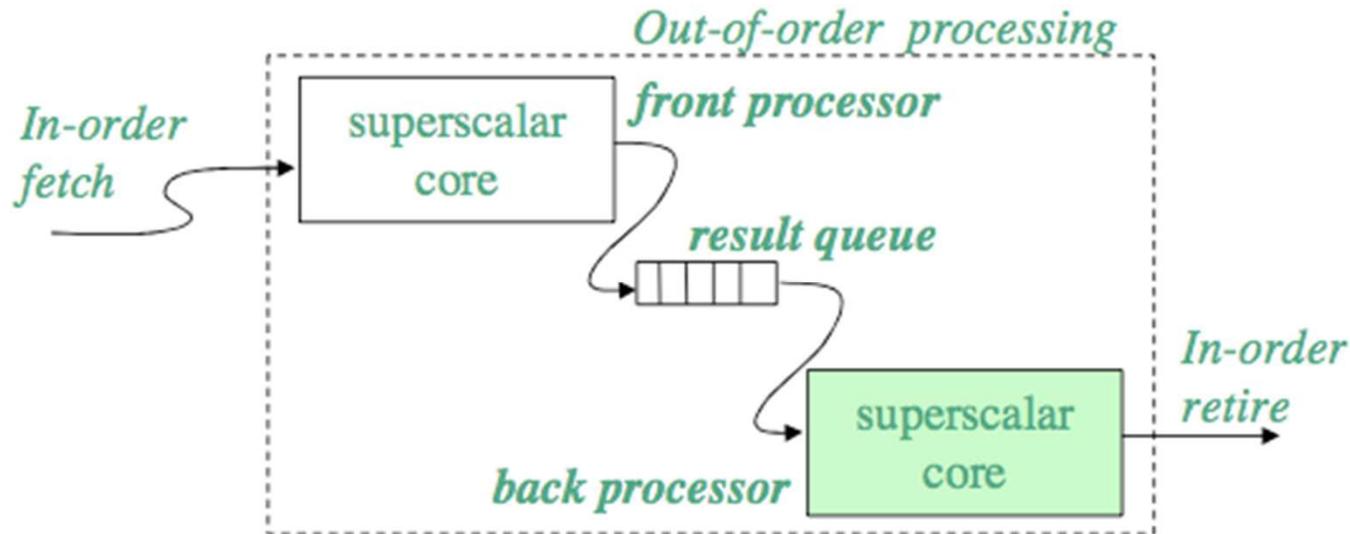


Dual Core Execution



- The front processor runs faster by invalidating long-latency cache-missing loads, similar to runahead execution
 - Load misses and their dependents are invalidated
 - Branch mispredictions dependent on cache misses cannot be resolved
- Highly accurate execution as independent operations are not affected
 - Accurate prefetches to warm up caches
 - Correctly resolved independent branch mispredictions

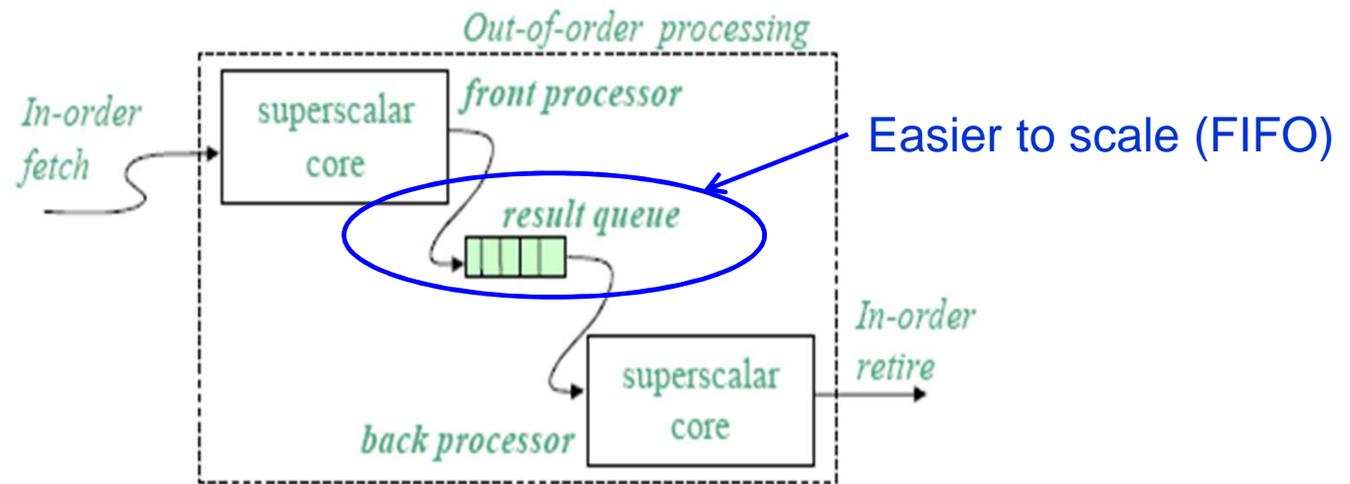
Dual Core Execution



- Re-execution ensures correctness and provides precise program state
 - Resolve branch mispredictions dependent on long-latency cache misses
- Back processor makes faster progress with help from the front processor
 - Highly accurate instruction stream
 - Warmed up data caches

Runahead and Dual Core Execution

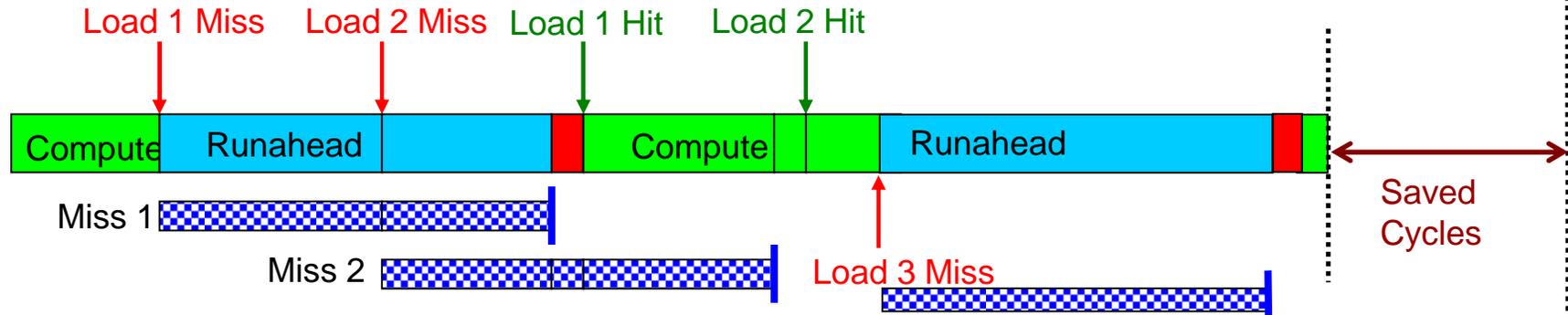
- Runahead execution:
 - + Approximates the MLP benefits of a large instruction window (no stalling on L2 misses)
 - Window size limited by L2 miss latency (runahead ends on miss return)
- Dual-core execution:
 - + Window size is not limited by L2 miss latency
 - Multiple cores used to execute the application; long misprediction penalty



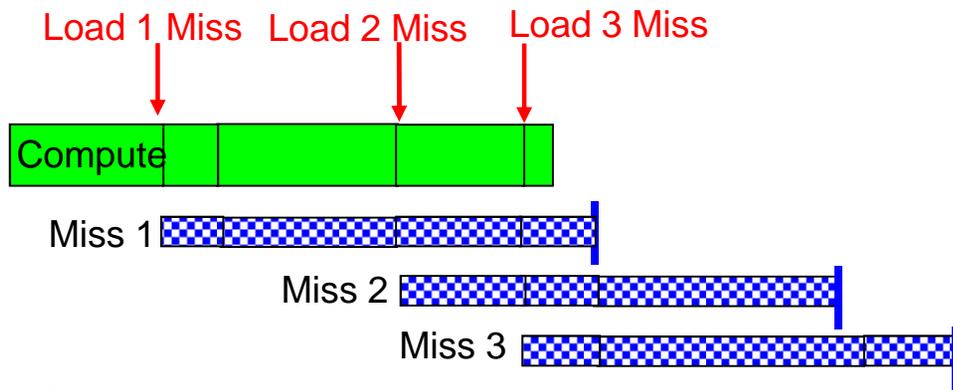
- Zhou, Dual-Core Execution: “**Building a Highly Scalable Single-Thread Instruction Window,**” PACT 2005.

Runahead and Dual Core Execution

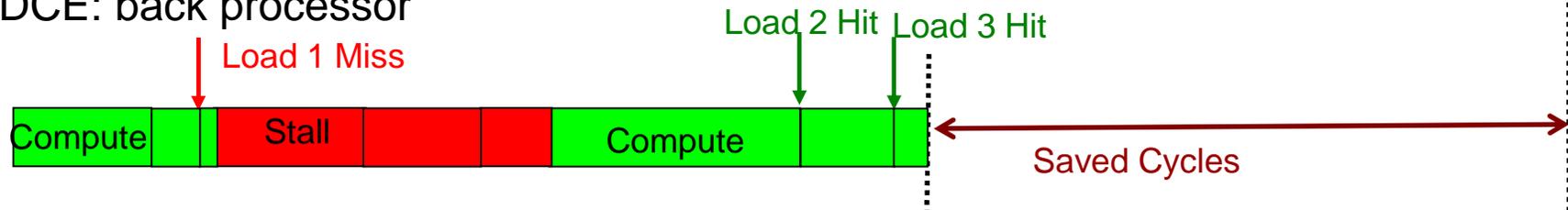
Runahead:



DCE: front processor



DCE: back processor



Handling of Store-Load Dependencies

- A load's dependence status is not known until all previous store addresses are available.
- How does the OOO engine detect dependence of a load instruction on a previous store?
 - Option 1: Wait until all previous stores committed (no need to check)
 - Option 2: Keep a list of pending stores in a store buffer and check whether load address matches a previous store address
- How does the OOO engine treat the scheduling of a load instruction wrt previous stores?
 - Option 1: Assume load independent of all previous stores
 - Option 2: Assume load dependent on all previous stores
 - Option 3: Predict the dependence of a load on an outstanding store

Store Buffer Design (I)

- An age ordered list of pending stores
 - un-committed as well as committed but not yet propoagated into the memory hierarchy
- Two purposes:
 - Dependency detection
 - Data forwarding (to dependent loads)
- Each entry contains
 - Store address, store data, valid bits for address and data, store size
- A scheduled load checks whether or not its address overlaps with a previous store

Store Buffer Design (II)

- Why is it complex to design a store buffer?
- Content associative, age-ordered, range search on an address range
 - Check for overlap of [load EA, load EA + load size] and [store EA, store EA + store size]
 - EA: effective address
- A key limiter of instruction window scalability
 - Simplifying store buffer design or alternative designs an important topic of research

Memory Disambiguation (I)

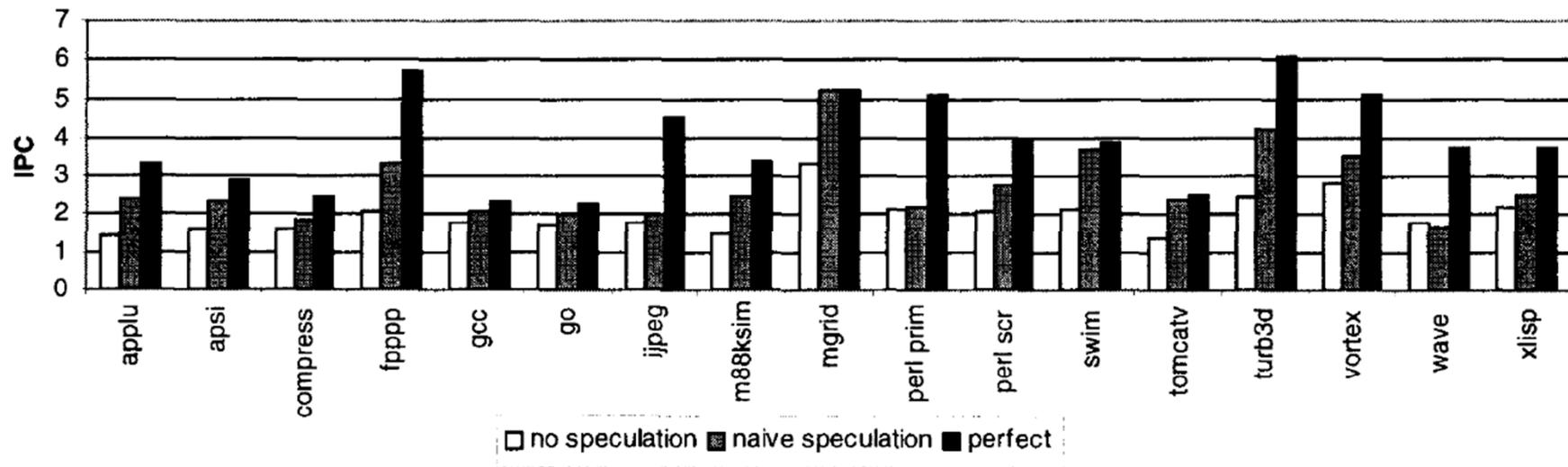
- Option 1: Assume load independent of all previous stores
 - + Simple and can be common case: no delay for independent loads
 - Requires recovery and re-execution of load and depends on misprediction

- Option 2: Assume load dependent on all previous stores
 - + No need for recovery
 - Too conservative: delays independent loads unnecessarily

- Option 3: Predict the dependence of a load on an outstanding store
 - + More accurate. Load store dependencies persist over time
 - Still requires recovery/re-execution on misprediction
 - Alpha 21264 : Initially assume load independent, delay loads found to be dependent
 - Moshovos et al., “**Dynamic speculation and synchronization of data dependences,**” ISCA 1997.
 - Chrysos and Emer, “**Memory Dependence Prediction Using Store Sets,**” ISCA 1998.

Memory Disambiguation (II)

- Chrysos and Emer, “**Memory Dependence Prediction Using Store Sets,**” ISCA 1998.



- Predicting store-load dependencies important for performance
- Simple predictors (based on past history) can achieve most of the potential performance

Speculative Execution and Data Coherence

- **Speculatively executed loads can load a stale value in a multiprocessor system**
 - The same address can be written by another processor before the load is committed → load and its dependents can use the wrong value

- **Solutions:**
 1. A store from another processor invalidates a load that loaded the same address
 - Stores of another processor check the load buffer
 - How to handle dependent instructions? They are also invalidated.
 2. All loads re-executed at the time of retirement

Open Research Issues in OOO Execution (I)

- Performance with simplicity and energy-efficiency
- How to build scalable and energy-efficient instruction windows
 - To tolerate very long memory latencies and to expose more memory level parallelism
 - Problems:
 - How to **scale or avoid scaling register files, store buffers**
 - How to **supply useful instructions into a large window** in the presence of branches
- How to approximate the benefits of a large window
 - MLP benefits vs. ILP benefits
 - Can the **compiler** pack more misses (MLP) into a smaller window?
- How to approximate the benefits of OOO with in-order + enhancements

Open Research Issues in OOO Execution (II)

- OOO in the presence of multi-core
- More problems: Memory system contention becomes a lot more significant with multi-core
 - OOO execution can overcome extra latencies due to contention
 - How to preserve the benefits (e.g. MLP) of OOO in a multi-core system?
- More opportunity: Can we utilize multiple cores to perform more scalable OOO execution?
 - Improve single-thread performance using multiple cores
- Asymmetric multi-cores (ACMP): What should different cores look like in a multi-core system?
 - OOO essential to execute serial code portions