The Impact of ILP Processors on Shared-Memory Multiprocessing

Sarita V. Adve

with

Vijay S. Pai, Parthasarathy Ranganathan

Hazim Abdel-Shafi, Jonathan Hall, and Tracy Harton

Department of Electrical and Computer Engineering
Rice University

`http://www-ece.rice.edu/~rsim`

`rsim@ece.rice.edu`

Motivation

Current microprocessors make aggressive use of

Instruction-Level Parallelism (ILP)

E.g., multiple issue, dynamic scheduling, non-blocking loads

For higher performance

Shared-memory multiprocessors built from ILP processors

But most shared-memory work assumes simple processors

Single issue, static scheduling, blocking loads

$\Rightarrow$ Need to re-examine architectural choices for current shared-memory systems

RSIM Project Goals

Performance

How effectively do multiprocessors exploit ILP features?

New techniques to improve performance?

Programmability and Portability

Memory consistency model used to involve a tradeoff

Programmability/portability vs. performance

Can ILP equalize performance of consistency models?

Simulation Techniques

Are current techniques appropriate for current systems?

New evaluation techniques for ILP multiprocessors?

Results – Performance [HPCA'97, ISCA'97]

Memory system a greater bottleneck for ILP multiprocessors

- Generally lower speedups with ILP multiprocessors
- Need to cluster load misses within instruction window

Software prefetching helps but

- System remains memory-bound
- Prefetching and load miss clustering may interact negatively

$\Rightarrow$ Need better latency tolerating and reducing methods

Results – Programmability and Portability [ASPLOS'96, SPAA'97]

ILP-based optimizations narrow performance gap between consistency models

But a significant gap still remains due to store latency

$\Rightarrow$ Relaxed consistency models still important

Results – Simulation Techniques [HPCA'97]

Current simulation models generally inaccurate

- Most common model: 26% to 192% error
- Best model: -8% to 73% error

But more detailed simulation models are slow

$\Rightarrow$ Need new techniques for accurate and fast evaluation

Outline

Experimental Methodology

Performance with ILP features

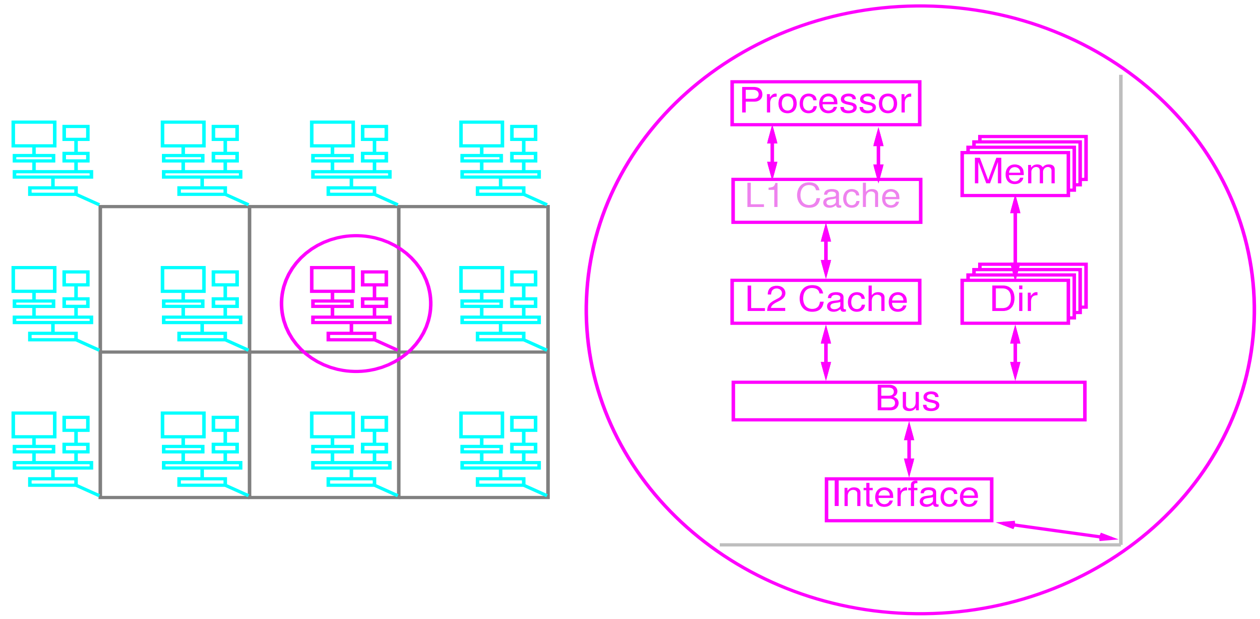
Software Prefetching

Memory Consistency Models

Simulation Techniques

Conclusions

Base Multiprocessor System (1 of 2)



Invalidate-based directory protocol

Release consistency

Base Multiprocessor System (2 of 2)

Representative latencies (contentionless)

L1 cache hit	1 cycle
L2 cache hit	10 cycles
Local memory	50 cycles
Remote memory	150-300 cycles

Cache sizes: 16K L1, 64K L2 (based on [Woo et al., 1995])

System size: 8 or 16 nodes (based on application scalability)

Processor Models

`Simple` processor

- Commonly used in simulation

`ILP` processor

- Aggressive ILP techniques
- Multiple issue, dynamic scheduling, non-blocking loads
- Resembles MIPS R10000

Compare multiprocessors with `Simple` and `ILP` processors

- Identical network and memory system

Simulator

RSIM – Rice Simulator for ILP Multiprocessors

Cycle-by-cycle instruction-level processor simulator

Discrete-event memory system simulator (RPPT)

Execution-driven $\Rightarrow$ Captures contention, synchronization

Applications

Applications

LU, FFT, Radix (SPLASH-2)

Mp3d, Water (SPLASH)

Erlebacher (Rice parallel compiler group)

LUopt, FFTopt (optimized SPLASH-2)

Input sizes

SPLASH/SPLASH-2: as in distribution, except LU

Erlebacher: 64 x 64 x 64 cube, block 8

Metrics

Execution time

Components of execution time

- CPU (Busy + non-memory instruction stall)
- Data Memory stall
- Synchronization stall

How to assign busy and stall time with overlapped instructions?

Busy and Stall Cycles

Based on retirement unit

- Instructions retire in program order for precise interrupts

Let R = maximum retirement rate

Busy cycle

- R instructions retire

Stall cycle

- Less than R instructions retire

Assign stall cycle to first instruction that stalls retirement

Outline

Experimental Methodology

Performance with ILP features

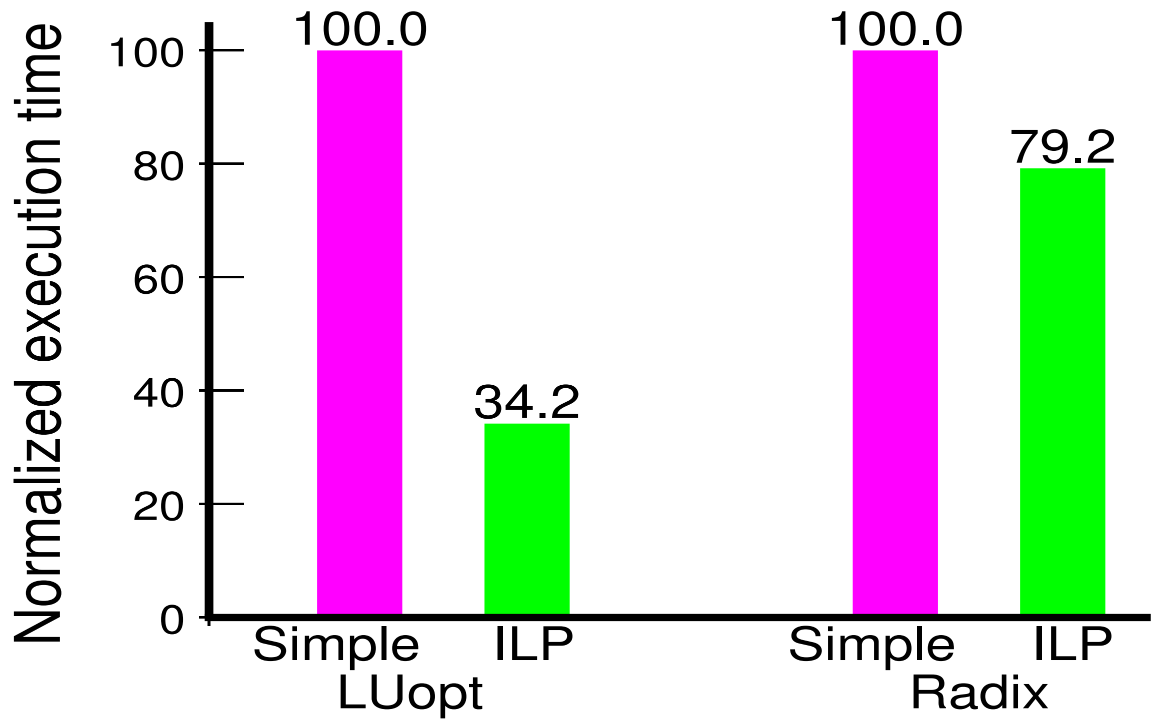
Software Prefetching

Memory Consistency Models

Simulation Techniques

Conclusions

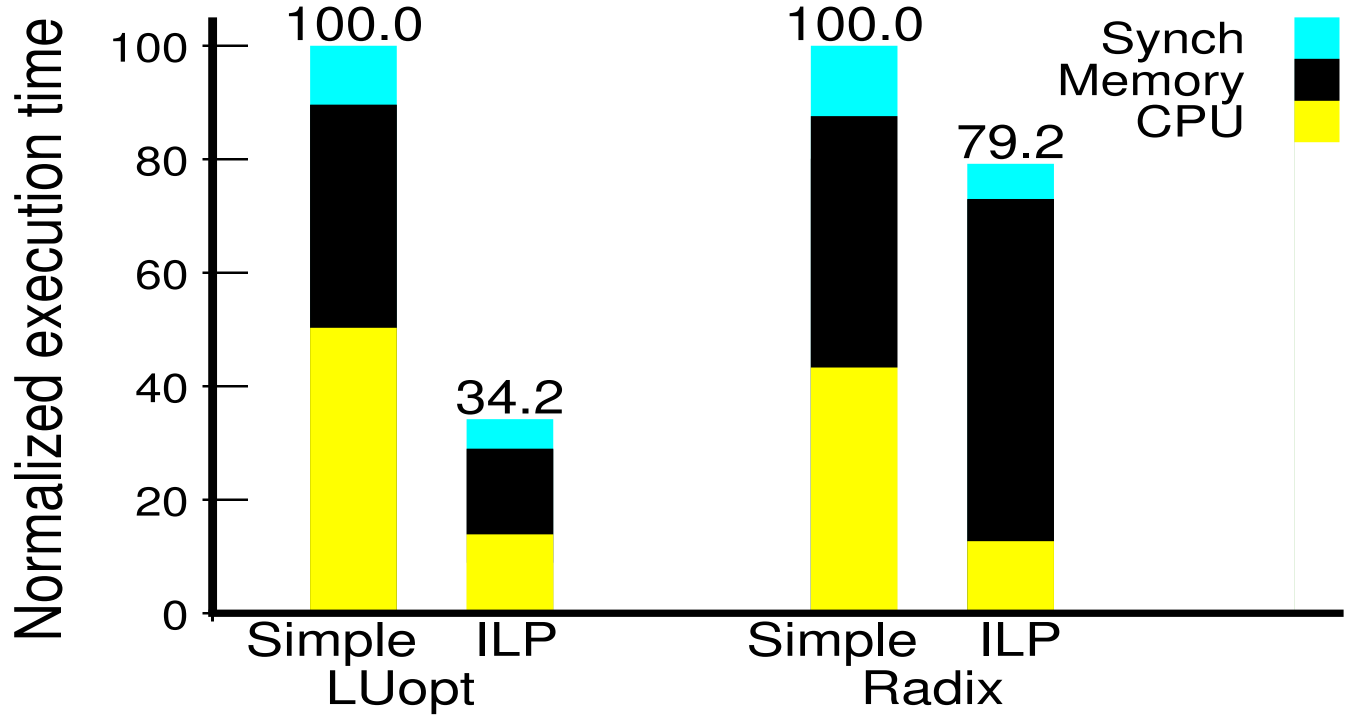
Impact of ILP on Multiprocessor Performance



ILP improves multiprocessor performance

BUT performance benefits vary with application (1.3X to 2.9X)

Impact of ILP on Multiprocessor Performance



CPU time: substantial, uniform benefits

Memory stall time: lower, variable benefits

Memory system a greater bottleneck despite ILP

Why is ILP Less Effective on Memory?

Factors affecting impact of ILP on memory

- Overlap
  - *Decreases* exposed memory latency
- Contention
  - *Increases* average memory latency
- Change in number of misses
  - Due to reordering and speculation in ILP

Memory Overlap (1 of 2)

Load misses need to overlap with each other because

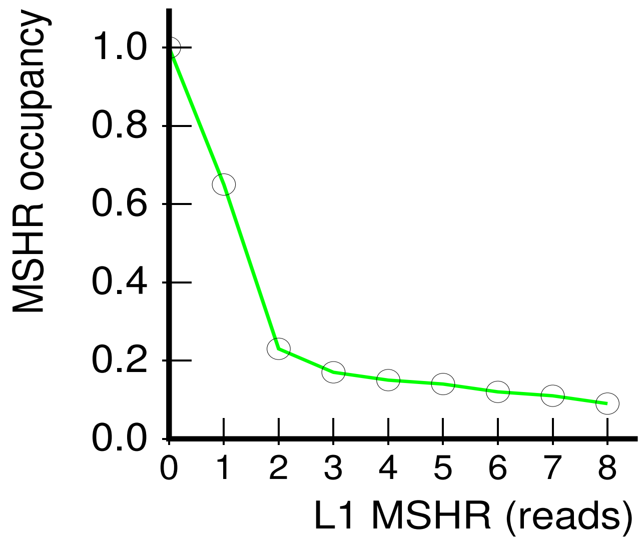
- Small instruction window (64 entries)
- Instructions retire in program order
- Load miss latencies much higher than other instructions

$\Rightarrow$ Need clustered load misses for effective overlap

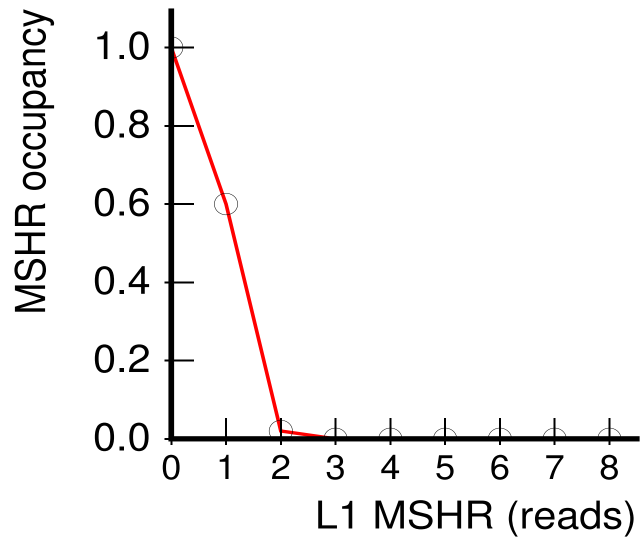
Use MSHR occupancy to measure overlap

- Writethrough, no write-allocate L1 $\Rightarrow$
  L1 MSHR occupancy gives load overlap

Memory Overlap (2 of 2)



LUopt



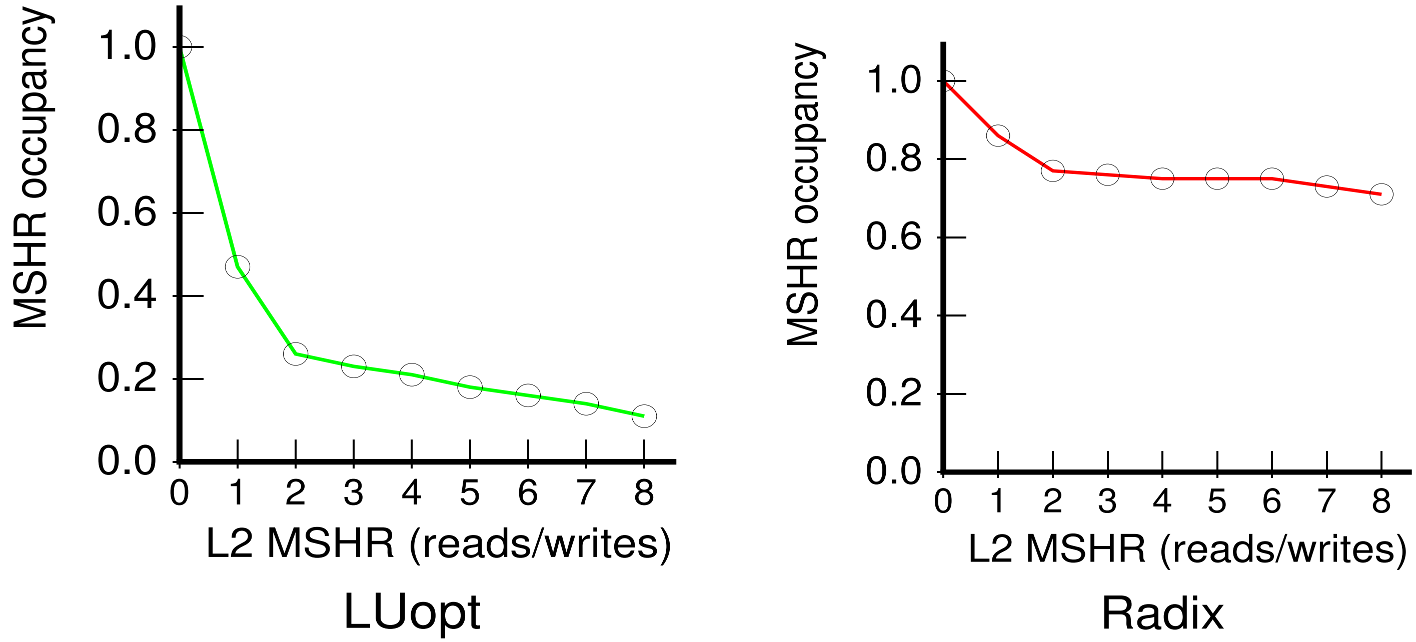
Radix

LUopt shows load miss clustering

$\Rightarrow$ LUopt sees memory speedup with ILP

Most apps show insufficient load miss clustering

Memory Contention



Radix sees contention from stores and no overlap of load misses

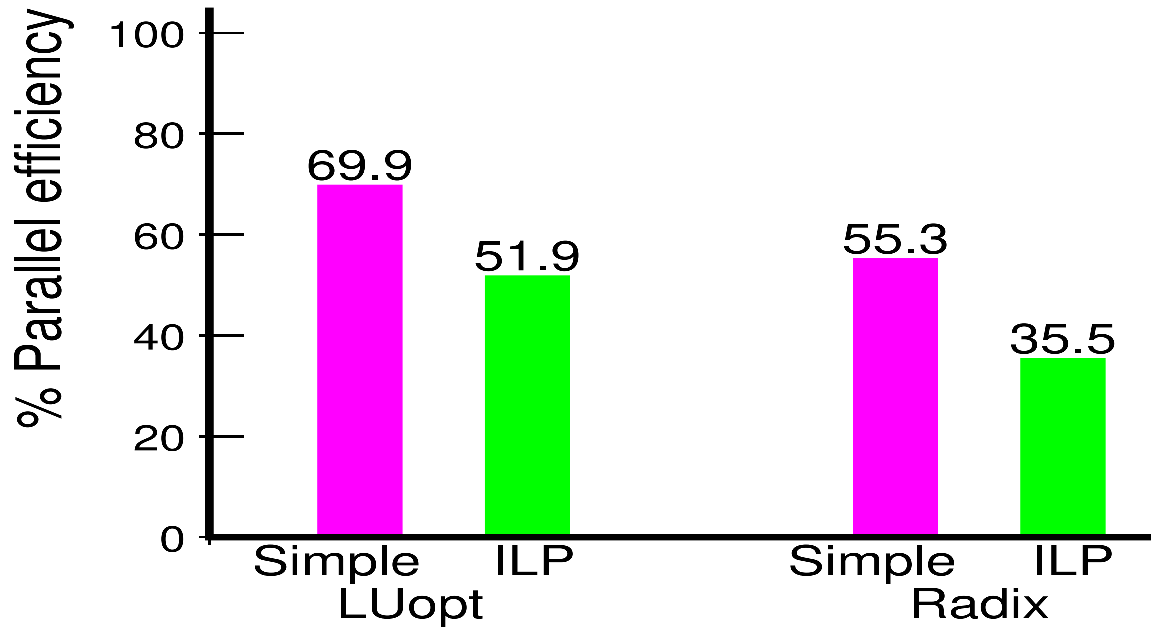
$\Rightarrow$ Radix sees memory slowdown with ILP

Lack of overlap dominates in most apps

Problems exacerbated in MPs due to longer, variable latencies

Impact of ILP on Parallel Efficiency

(Impact on Multiprocessors vs. Uniprocessors)



ILP generally degrades parallel efficiency

ILP less effective in multiprocessors

Why is ILP Less Effective in Multiprocessors?

Longer latencies

Must overlap load misses with other load misses

Dichotomy of local/remote latencies

Overlapped load misses must have similar latencies

Conclusions So Far

Memory system a greater bottleneck with ILP processors

- Need to cluster load misses
- Need other new and old techniques

Outline

Experimental Methodology

Performance with ILP features

Software Prefetching

Memory Consistency Models

Simulation Techniques

Conclusions

Software-Controlled Non-Binding Prefetching

Idea

- Insert prefetch instruction for line to be used later
- Line arrives in cache before demand access

Successful in previous-generation (`Simple`) systems

- Less than 17% of time in data memory stalls in 4 of 5 apps

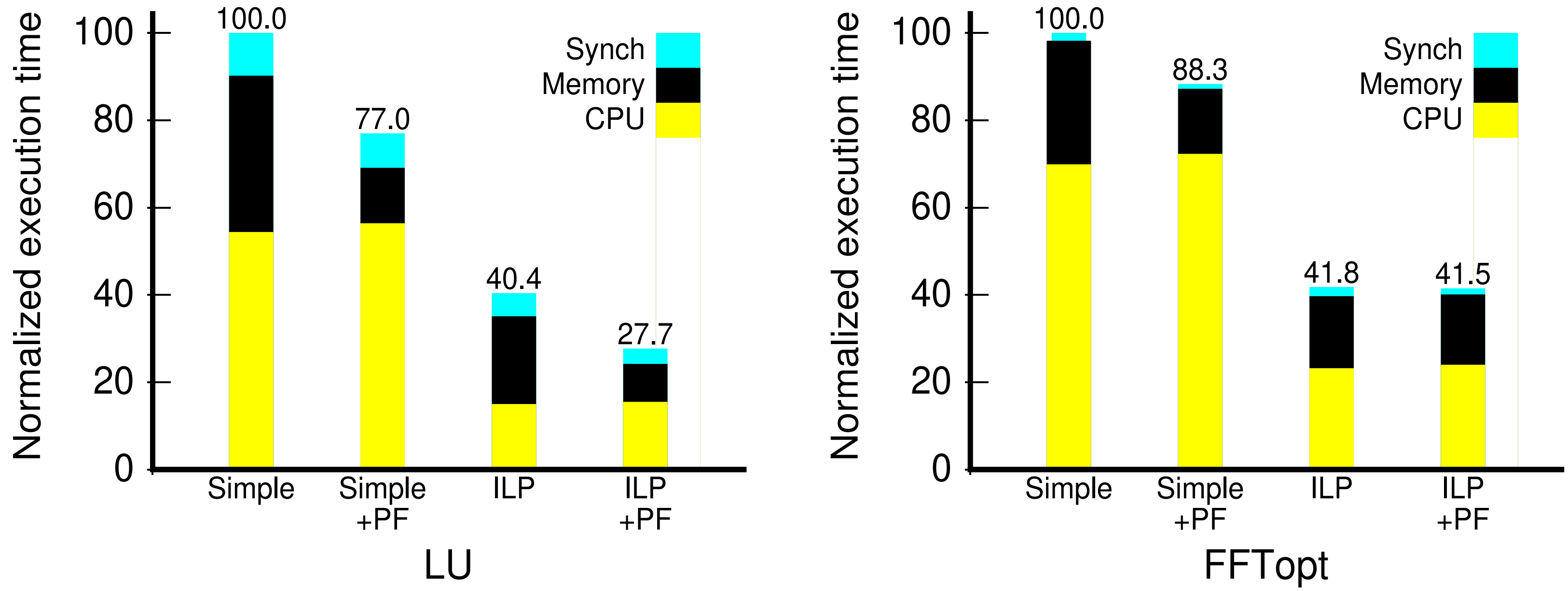
On `ILP` systems

- Prefetching can give overlap across instruction window

How does software prefetching interact with ILP?

Overall Results

Followed state-of-the-art algorithm by Mowry et al. [Mowry 94]



Prefetching improves `ILP` by 1.14X to 1.46X for 3 apps

But data memory stall time > 30% in 4 apps for `ILP`

Less than 17% in 4 apps for `Simple`

Interaction of ILP with Software Prefetching

Negative interactions

- Increased late prefetches
- Increased resource contention

Positive interaction

- Overlapped accesses

Negative effects dominate in our applications

Negative Interaction 1 – Late Prefetches

ILP sees reduced time between misses

Multiple issue, out-of-order speed up computation

Load clustering brings misses closer

⇒ Prefetch must be overlapped with more iterations

Smaller steady state for prefetched loop

⇒ Increased late prefetches

Longer latencies in ILP exacerbate the problem

Negative Interaction 2 – Resource Contention

`ILP` already stresses memory system more than `Simple`

Prefetching exacerbates contention

Prefetching cannot address contention before L1 cache
(e.g., address units, L1 ports)

Positive Interaction – Overlapped Accesses

Loads hard to prefetch may be overlapped in ILP

Prefetching needs to target less latency,
so may appear more effective

Seen in LU

Negative effects dominate for `ILP` in our applications

Conclusions So Far

Software prefetching improves execution time but

- ILP system still memory-bound
- Prefetching may interact negatively with load clustering

$\Rightarrow$ Need better latency reducing/tolerating mechanisms

- Producer-initiated communication [HPCA'97]
- Effective load miss clustering
- Better prefetching algorithm

Outline

Experimental Methodology

Performance with ILP features

Software Prefetching

Memory Consistency Models

Simulation Techniques

Conclusions

Memory Consistency Models

Memory consistency model

- Order in which memory operations must appear to execute
- Constrains overlap and re-ordering of memory operations
- Affects programmability and performance

Common Memory Consistency Models (1 of 2)

Sequential consistency (SC)

A processor's memory operations appear in program order

Most intuitive programming model

Earlier implementations serialized all ops of a processor

$\Rightarrow$ Lowest performance for earlier systems

Common Memory Consistency Models (2 of 2)

Release Consistency (RC)

Data operations may be overlapped or reordered

Ordering imposed only at synchronization

Best performance for earlier systems

But must identify data and synchronization

$\Rightarrow$ More complex programming model

Processor Consistency (PC)

Relaxes Store $\rightarrow$ Load order

Performance between SC and RC for earlier systems

Memory Consistency Models With ILP

ILP allows optimizations for consistency models [Gharachorloo+ 1991]

Hardware prefetching from instruction window

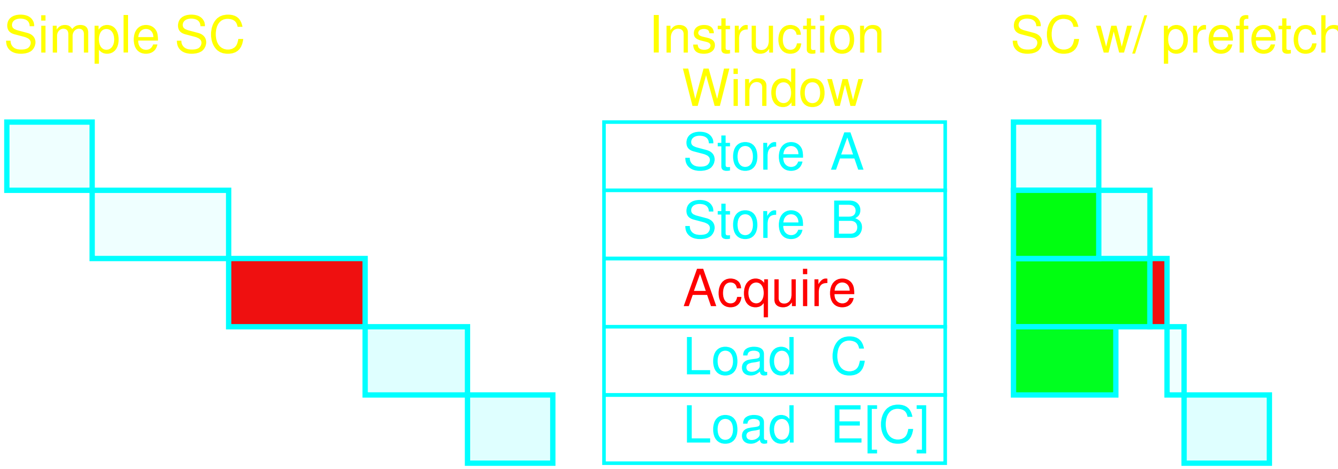
Speculative load execution

Conjecture: Optimizations equalize performance of SC, PC, RC

Optimized Implementations (1 of 2)

Hardware-controlled non-binding prefetching

Prefetch stalled instructions in window

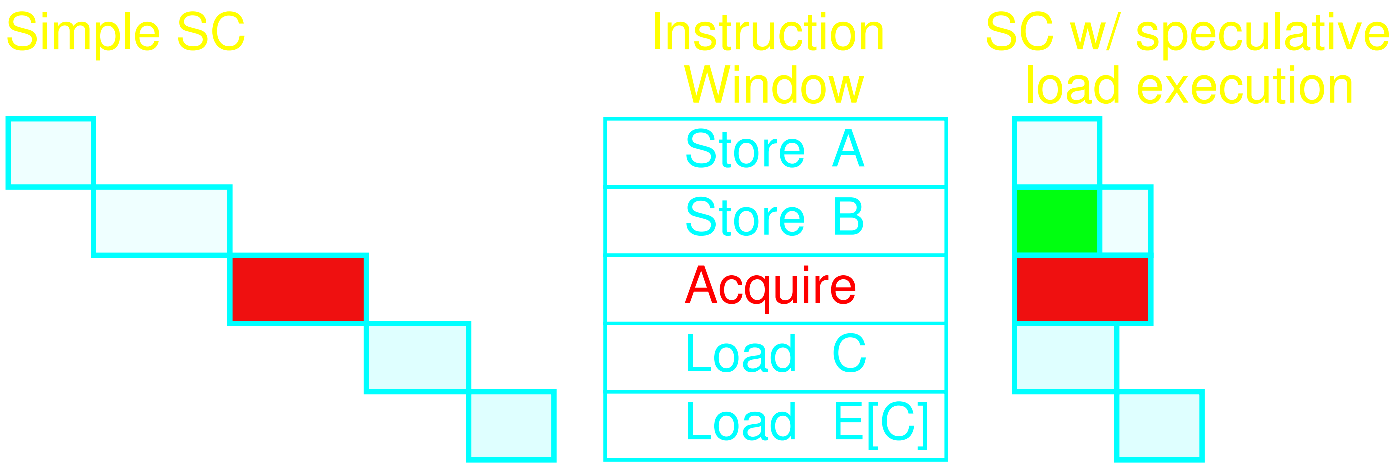


Limitation: Cannot use prefetched value

Optimized Implementations (2 of 2)

Speculative load execution

Allows load values to be consumed out-of-order



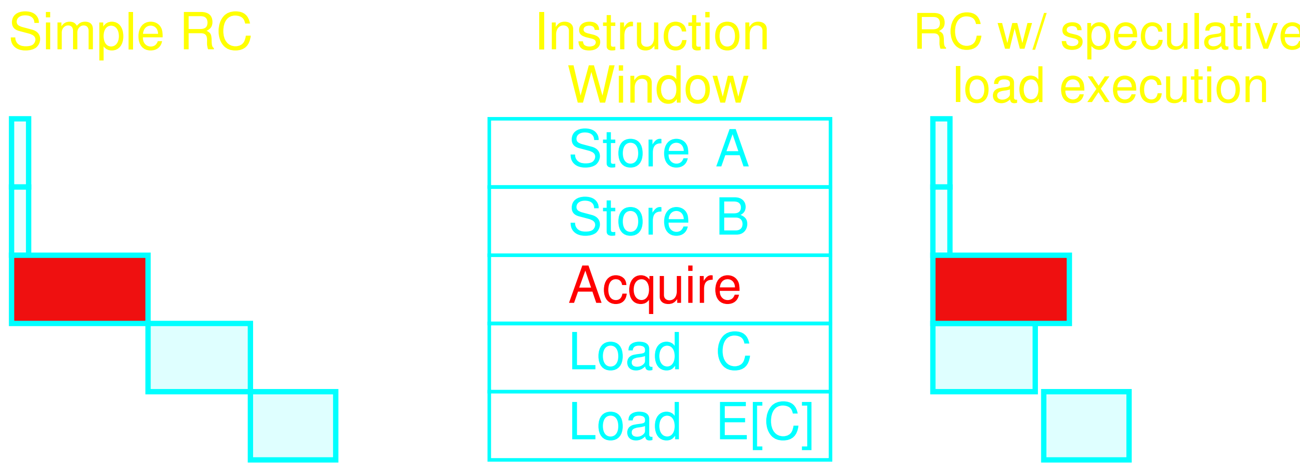
Value is wrong only if over-written

Invalidation/update message received

If violation, rollback instructions in window

Optimized Implementations with RC, PC

Techniques applicable to RC and PC



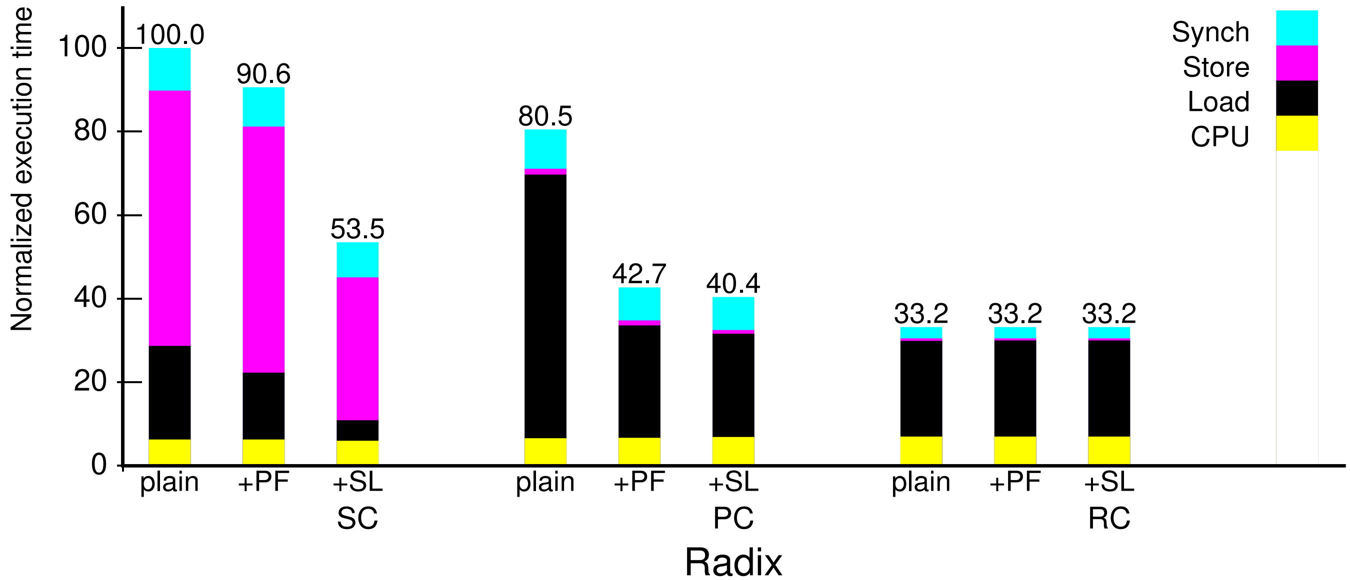
RC exploits optimizations only at synchronization

Less potential improvement

PC can exploit optimizations for

Load $\rightarrow$ Load, Load $\rightarrow$ Store, Store $\rightarrow$ Store

Results



Optimizations dramatically improve SC and PC

Little impact on RC

BUT PC still better than SC (15%, 24% for 2 apps)

RC still better than PC (18%, 23% for 2 apps)

RC still better than SC (18% to 37% for 3 apps)

Limitations of SC and PC

Load latency not a problem

Very few rollbacks

SC: Instruction window too small to overlap store prefetch

But with large window, early store prefetches hurt

PC: Memory queue too small

Early store prefetches exacerbate the problem

Clustering store with load miss may help

Conclusions for Consistency Models So Far

Is SC $\approx$ PC $\approx$ RC in performance?

- Optimizations dramatically improve SC and PC
- But PC still higher performance than SC
- But RC still higher performance than SC, PC
- RC higher performance *with less hardware complexity*

Outline

Experimental Methodology

Performance with ILP features

Software Prefetching

Memory Consistency Models

Simulation Techniques

Conclusions

Simulation Techniques

Most MP studies use direct-execution simulation

- Application runs directly on host
- Calls simulator on memory instructions
- Models simple processors with blocking loads

Approximations used

- Simple processors $\approx$ ILP (?)
- Simple + faster CPU $\approx$ ILP (?)
- Simple + faster CPU + faster L1 cache $\approx$ ILP (?)

Are simple approximations to ILP valid?

Simulation Models Reported

Simple

Most common simulation model

Same clock speed as ILP system

Best approximate

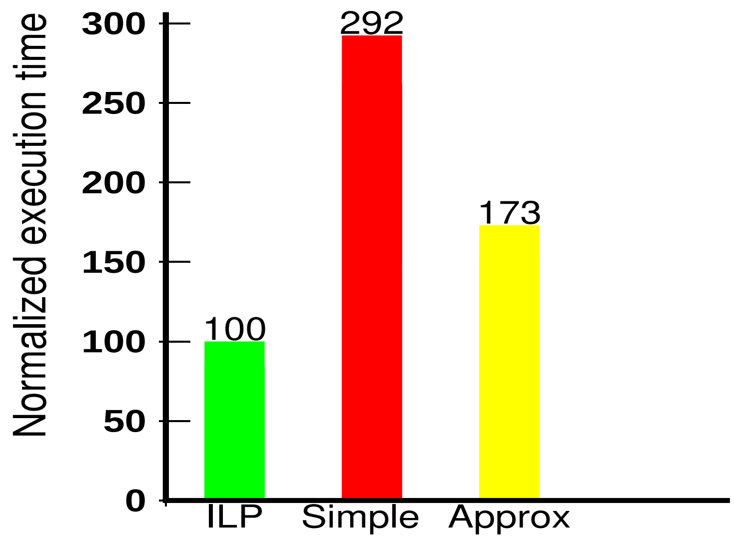
Simple processor

Clock speed $=$ 4$\times$ILP clock speed

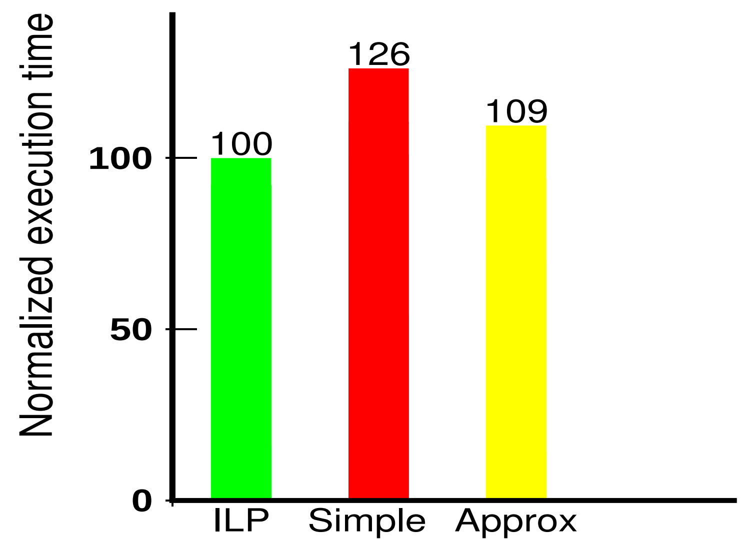
L1 cache hits also sped up

Rest of memory system unchanged

Effectiveness of Simulation Models



LUopt



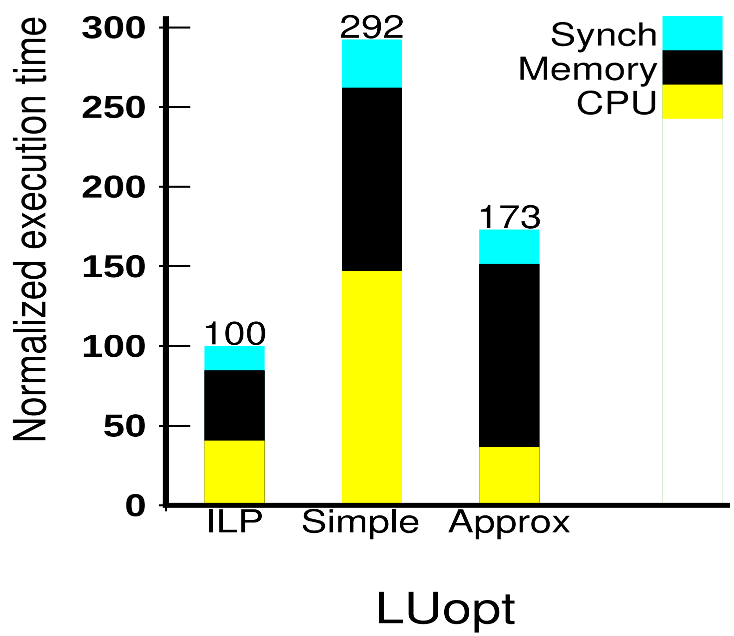
Radix

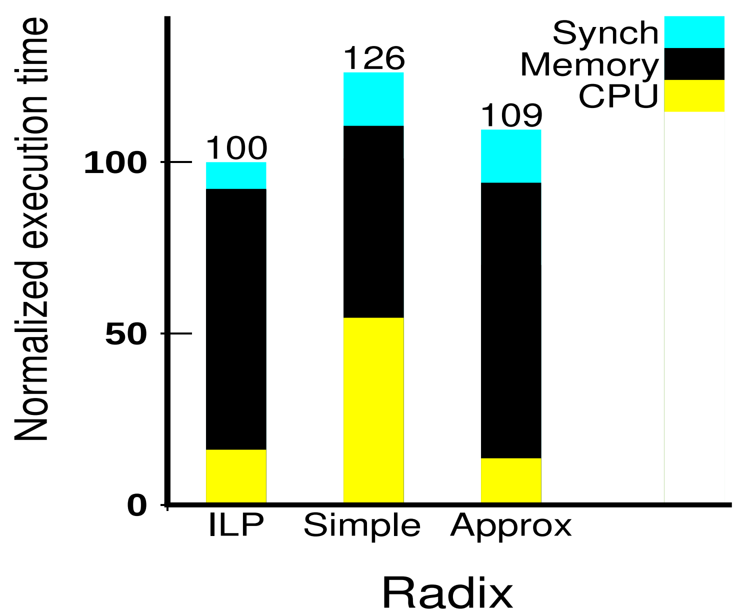
Simple processor based models can give large, variable errors

Most common model: 26% to 192% error

Best model: -8% to 73% error

Effectiveness of Simulation Models (Cont.)





Key source of error

Simple models do not account for memory overlap in ILP

High memory overlap $\Rightarrow$ greater errors

Conclusions for Simulation Techniques

Simple processor models can give large errors

But detailed simulation slow

$\Rightarrow$ Need novel methods for ILP multiprocessors

- Smarter direct execution
- Parallel simulation
- Sampling
- Selective analytical models

Summary

Performance

- Memory system a larger bottleneck for ILP multiprocessors
  - Need to cluster load misses
- Software prefetching helps but not enough
  - Miss clustering can decrease impact of prefetching

Programmability and Portability

- PC still has advantage over SC
- RC still has advantage over SC, PC

Simulation Techniques

- Simple processor models can give large errors

Ongoing Work

Performance

Producer-initiated communication, load miss clustering, better prefetching

Programmability/Portability

Alleviate negative impact of early store prefetches in SC, PC

Store clustering for SC, PC

Simulation Techniques

Smarter direct execution, parallel simulation, sampling, selective analytical models

Incorporate operating system functionality

New applications: Database, signal processing, communications