

## ILP vs. Consistency

### An Evaluation of Memory Consistency Models for Shared-Memory Systems with ILP Processors

Vijay S. Pai, Parthasarathy Ranganathan, Sarita V. Adve, and Tracy Harton

Department of Electrical and Computer Engineering  
Rice University  
Houston, Texas 77251-1892  
{vijaypai|parthas|sarita|harton}@rice.edu

CSE 240B

Dean Tullsen

CSE 240B

Dean Tullsen

## What's the big question?

## Architectural Parameters

- SC and RC
- Write-through vs write-back caches
- Load prefetching
- Write prefetching
- Speculative loads

CSE 240B

Dean Tullsen

CSE 240B

Dean Tullsen

## HW-generated Prefetching

- How does this hide consistency-related latency?
- For Loads?
- For Stores?
- Better with WT or WB cache?
- What about pointer chasing applications?

*CSE 240B*

*Dean Tullsen*

## Speculative Loads

- How does this hide consistency-related latency?
- For Loads?
- For Stores?
- Better with WT or WB cache?
- What about pointer chasing applications?

*CSE 240B*

*Dean Tullsen*

## RC with Prefetching?

*CSE 240B*

*Dean Tullsen*

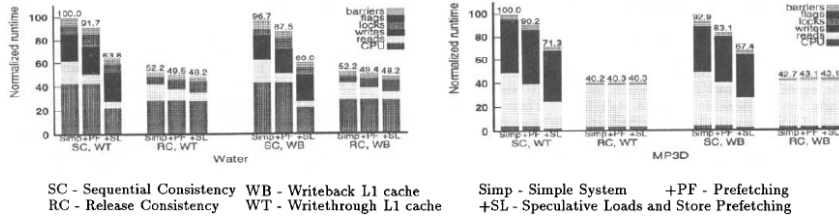
## RC with Speculative Loads?

*CSE 240B*

*Dean Tullsen*

## More aggressive protocol

- Originally – cannot overlap ownership request with shared read request.
- Optimization?
- Why Hard?
- Why helpful?



CSE 240B

Dean Tullsen

CSE 240B

Dean Tullsen

## Still a gap between RC and SC

- Why?

## Fuzzy Acquires

```

Code for Processor Pj
for(i=1; i < N+1; i++){
    WaitFlag(Flag[j][i])
    AcquireMemBar
    DoWork(X[i])
    ReleaseMemBar
    SetFlag(Flag[j+1][i])
}

/* Prologue code */
WaitFlag(Flag[j][1])

for(i=1; i < N; i++) {
    AcquireMemBar
    WaitFlag(Flag[j][i+1])
    DoWork(X[i])
    ReleaseMemBar
    SetFlag(Flag[j+1][i])
}

/* Epilogue code */
AcquireMemBar
DoWork(X[N])
ReleaseMemBar
SetFlag(Flag[j+1][N])
    
```

(a) (b)  
 Figure 4: Application of fuzzy acquires.

CSE 240B

Dean Tullsen

CSE 240B

Dean Tullsen

# Selective Acquires

CSE 240B

Dean Tullsen

# Is SC + ILP = RC?

Chris Gniady, Babak Falsafi, and T. N. Vijaykumar  
School of Electrical & Computer Engineering  
Purdue University  
1285 EE Building  
West Lafayette, IN 47907  
*impetus@ecn.purdue.edu, <http://www.ece.purdue.edu/~impetus>*

CSE 240B

Dean Tullsen

# Big Idea?

CSE 240B

Dean Tullsen

# Speculative Execution

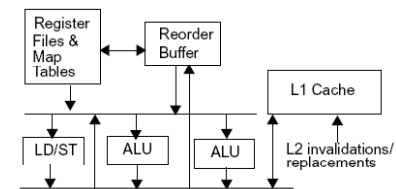


FIGURE 1: Speculative execution in current microprocessors.

CSE 240B

Dean Tullsen

## How do you

- Make speculative loads work in a SC system?
- Do write buffering in a SC system?

CSE 240B

Dean Tullsen

## In this paper

- What do you need to make SC match RC?

CSE 240B

Dean Tullsen

## Reordering stores

- What do you need to make this work in SC?

CSE 240B

Dean Tullsen

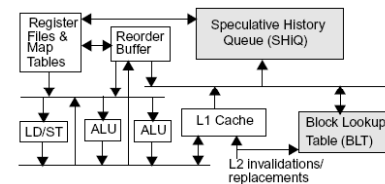


FIGURE 2: SC++ Hardware.

CSE 240B

	Relaxing Orders	Mechanisms to Guarantee Order	Potential for Order Violation
SC	loads bypass loads and stores	speculative execution using reorder buffer, load/store queue, and speculative placement of data in cache	lower
RC	loads and stores bypass each other between fences, loads bypass loads and stores across fences	fence instruction, speculative execution as in SC across fences	lower
SC++	loads and stores bypass each other	speculative execution using reorder buffer, load/store queue, and speculative placement of instructions in SHIQ, data addresses in BLT and data in cache	higher

Table 1: Comparison of implementations.

Dean Tullsen

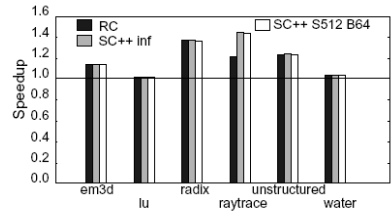


FIGURE 3: Comparison of SC, RC, and SC++.

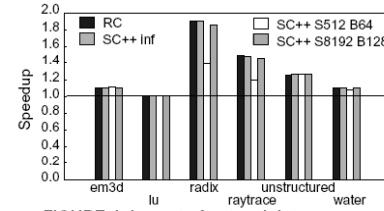


FIGURE 4: Impact of network latency.

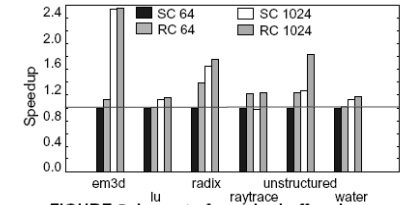


FIGURE 5: Impact of reorder buffer size.

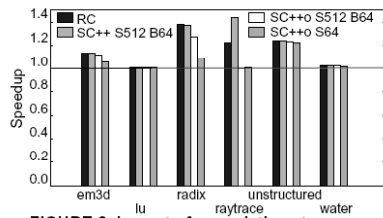


FIGURE 6: Impact of speculative stores.

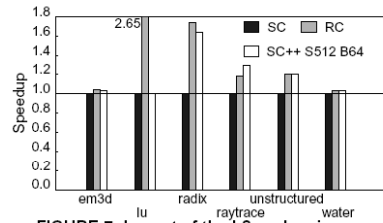


FIGURE 7: Impact of the L2 cache size.