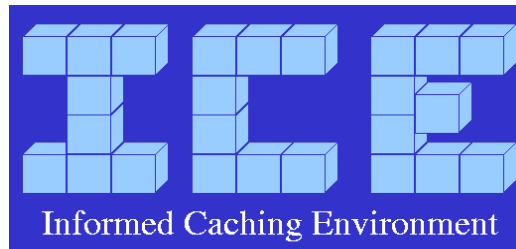


Annotated Memory References: A Mechanism for Informed Cache Management

Alvin R. Lebeck, David R. Raymond, Chia-Lin Yang

Mithuna S. Thottethodi

Department of Computer Science, Duke University



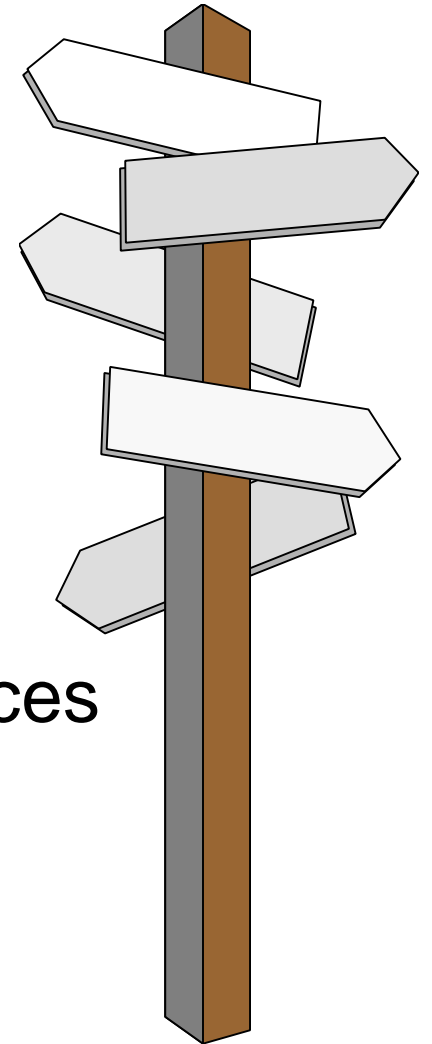
<http://www.cs.duke.edu/ari/ice>

Motivation

- Importance of cache performance
- Allow software to assist in cache management
- Issues in software assisted cache management and scope of this study
 - What information?
 - How is the information conveyed?
 - How is the information exploited?

Outline

- Motivation and scope
- The proposed mechanism
 - Static Instruction Annotation
 - The Tag Instruction
 - Overhead - Not more than 2%
- Utilizing Annotated Memory References
 - Retain/Release annotation
 - Word/Block annotation
 - Between 11% and 17% speedup
- Conclusion



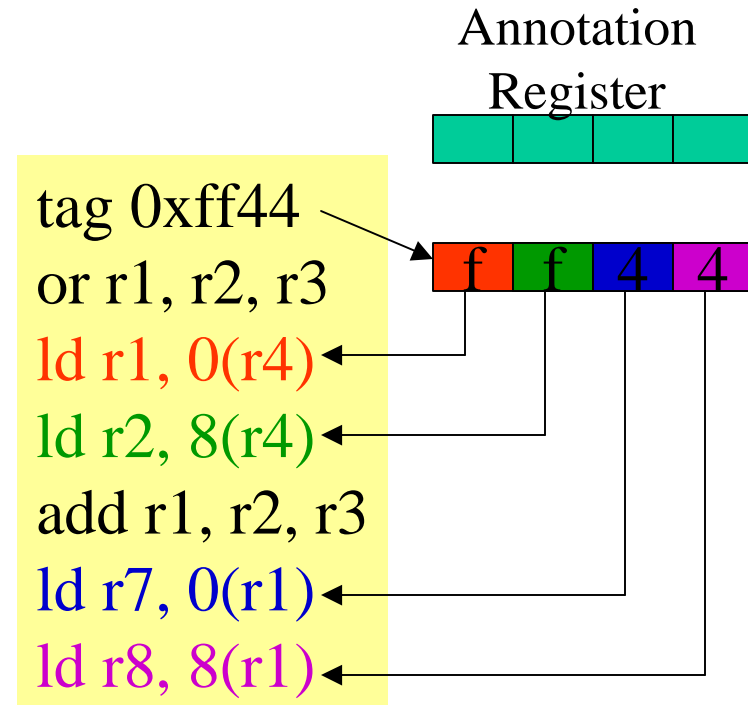
Design Decisions

What?	How?	Static	Dynamic
Instruction (PC)		✓	
Address (EA)			

- **Static Instruction** annotation
- One additional instruction (TAG) proposed
- Annotation Register

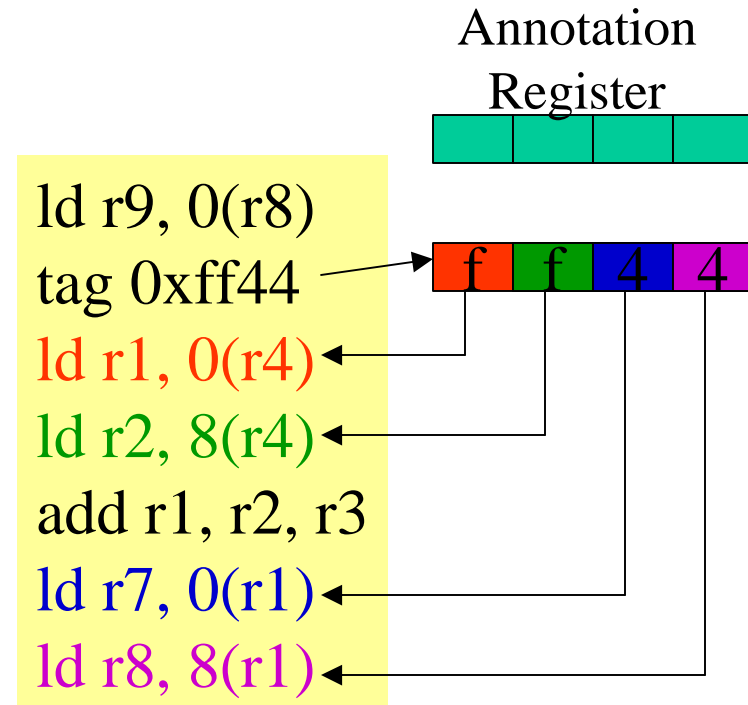
The Tag Instruction

- Tag instruction fills the Annotation Register
- Future n loads get k bits of annotation
- n : tag coverage
- 2^k : number of possible annotations
- Implementation issues in modern processors
 - Multiple Issue
 - Out-of-order Execution

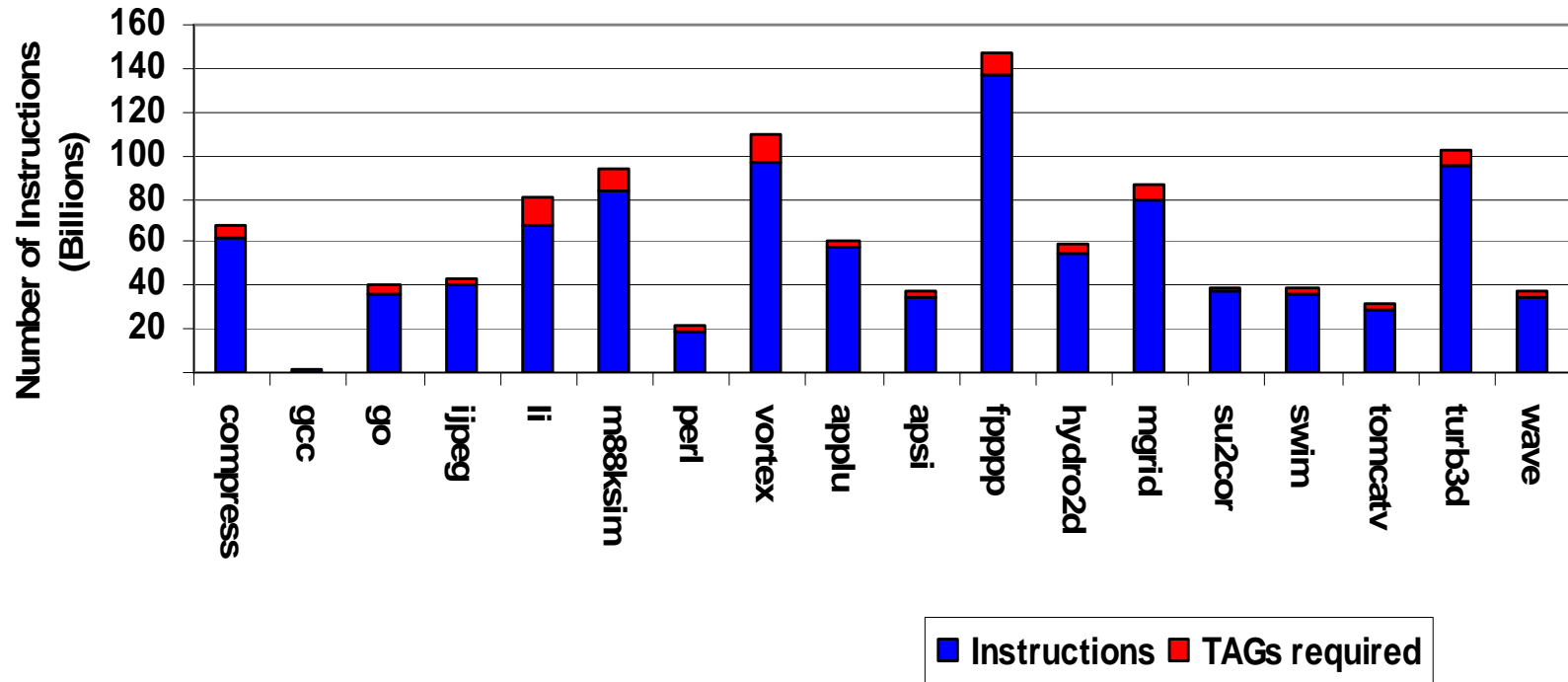


Implementation Issues

- Multiple Issue
 - WAR hazard
 - Dependence
- Out of order execution
 - Annotations associated with loads at decode time
 - Load/Store queue entries hold annotation bits



Instruction Overhead

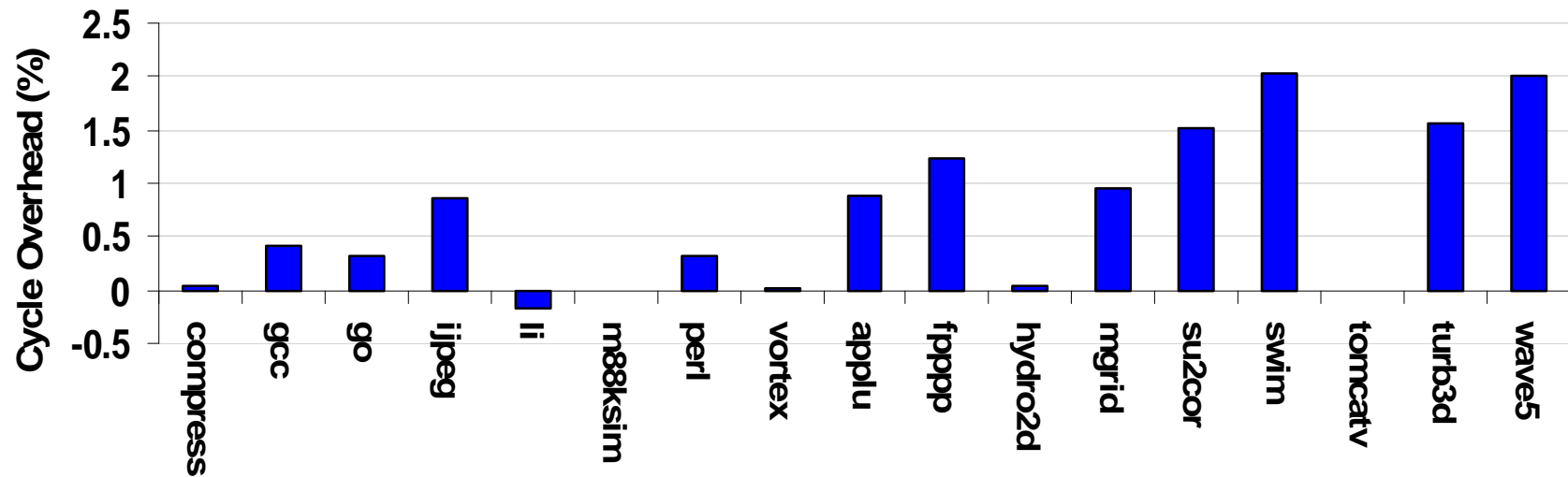


- All memory references annotated
 - 4 bits/annotation; Tag coverage of 6
- Instruction overhead
 - Integer codes : 5.5% to 16.2%
 - Floating Point codes : 6.2% to 7.7%

Cycle Overheads - Experiments

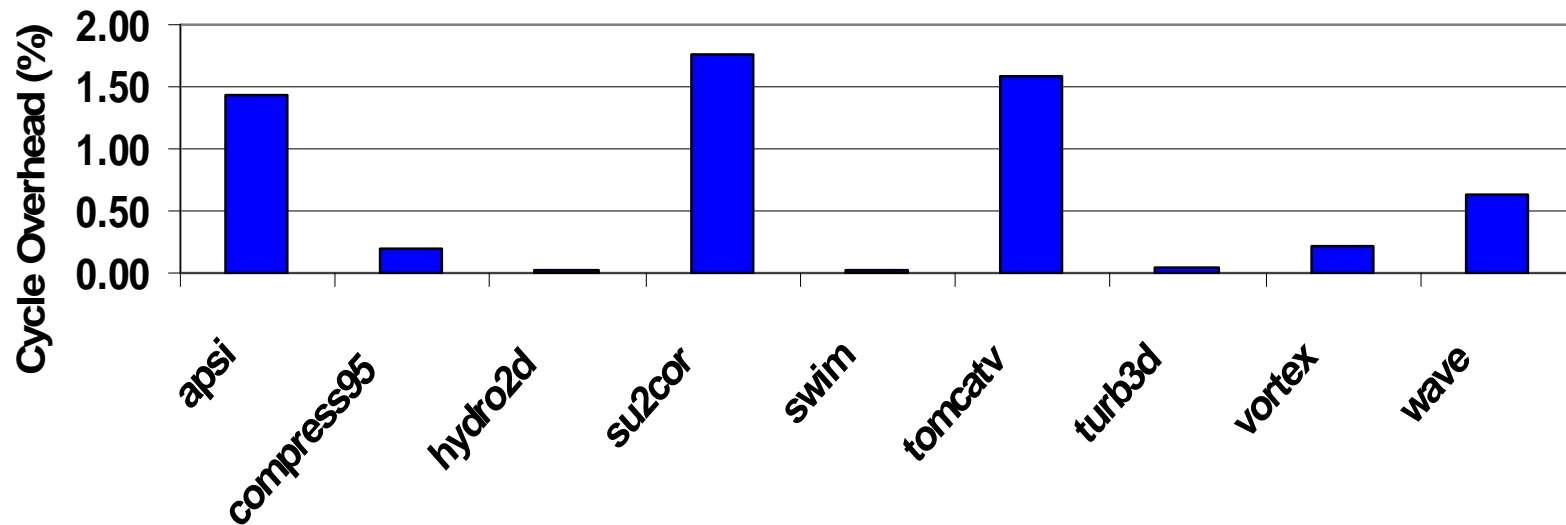
- Statically scheduled processors
 - ATOM based
 - 21164 issue policies
 - Perfect branch prediction
 - Ideal memory system
 - No inter-block dependencies
- Dynamically scheduled processors
 - SimpleScalar simulator
 - 4-way issue, 64 RUU, 32 LSQ
 - Simple overhead computation

Statically Scheduled Processor



- All memory references annotated
 - 4 bits/annotation; Tag coverage of 6
- Cycle overhead
 - Integer Codes: 0% to 0.85%
 - Floating Point Codes: 0% to 2%

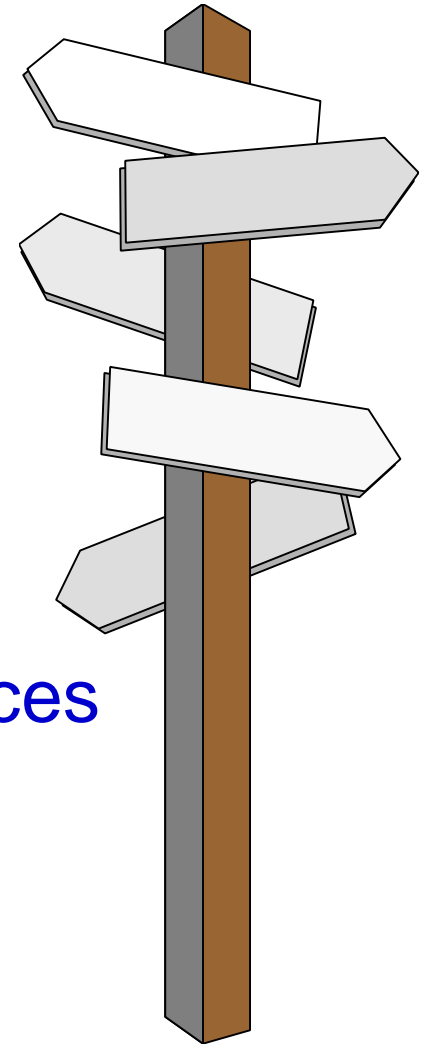
Dynamically Scheduled Processor



- All memory references annotated
 - 4 bits/annotation; Tag coverage of 6
- Cycle overhead
 - Integer Codes: 0% to 0.2%
 - Floating Point Codes: 0% to 1.76%

Outline

- Motivation and scope
- The proposed mechanism
 - Static Annotation
 - The Tag Instruction
 - Overhead - Not more than 2%
- Utilizing Annotated Memory References
 - Retain/Release annotation
 - Word/Block annotation
 - Between 11% and 17% speedup
- Conclusion

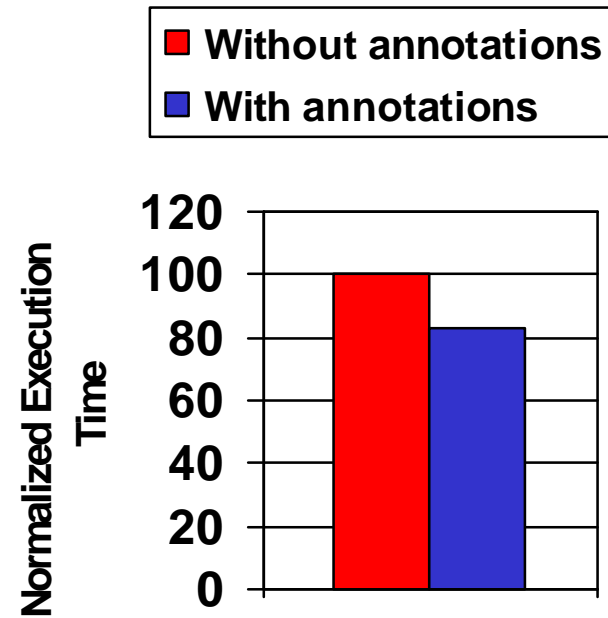


Utilizing Annotated Memory References

- Code inspection and manual insertion of annotations
- CProf tool to give insights of code operation
- Multimedia applications
 - epic, ijpeg, pegwit

Better Block Replacement

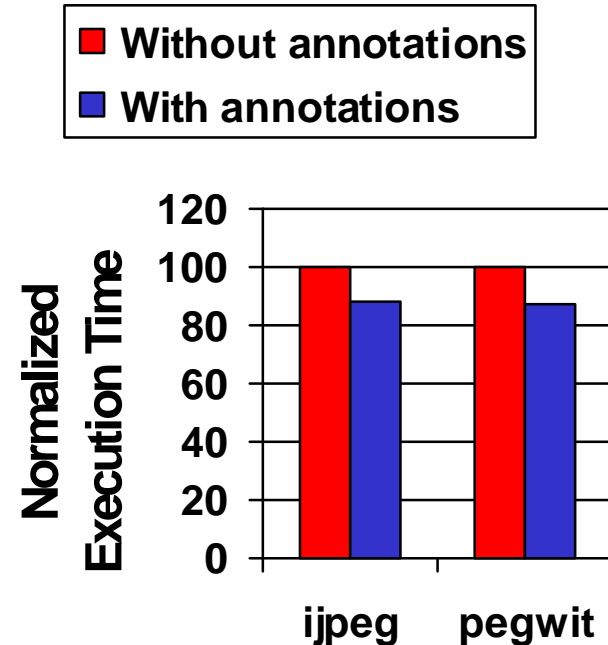
- Insight: some blocks should be retained even if LRU block
- Retain/Release annotations
- A block marked Retain cannot be replaced unless Released
- Bypass cache if no replacement candidate
- epic



4-way issue, OoO processor
64 RUU, 32 LSQ entries
8KB, 32 Byte block, Direct Mapped

Better Block Sizes

- Insight : Implicit prefetch of larger blocks hurts performance
- WordMode/BlockMode annotations
- WordMode annotated references bring in only a word and not the whole block
- pegwit and ijpeg



4-way issue, OoO processor
64 RUU, 32 LSQ entries
8KB, 32 Byte block, Direct Mapped

Conclusions

- Cache performance is critical
- Software can assist in managing caches
- We demonstrate a mechanism that allows software to help manage caches with
 - low overheads (under 2%), and
 - significant benefits (between 11% and 17% speedups)

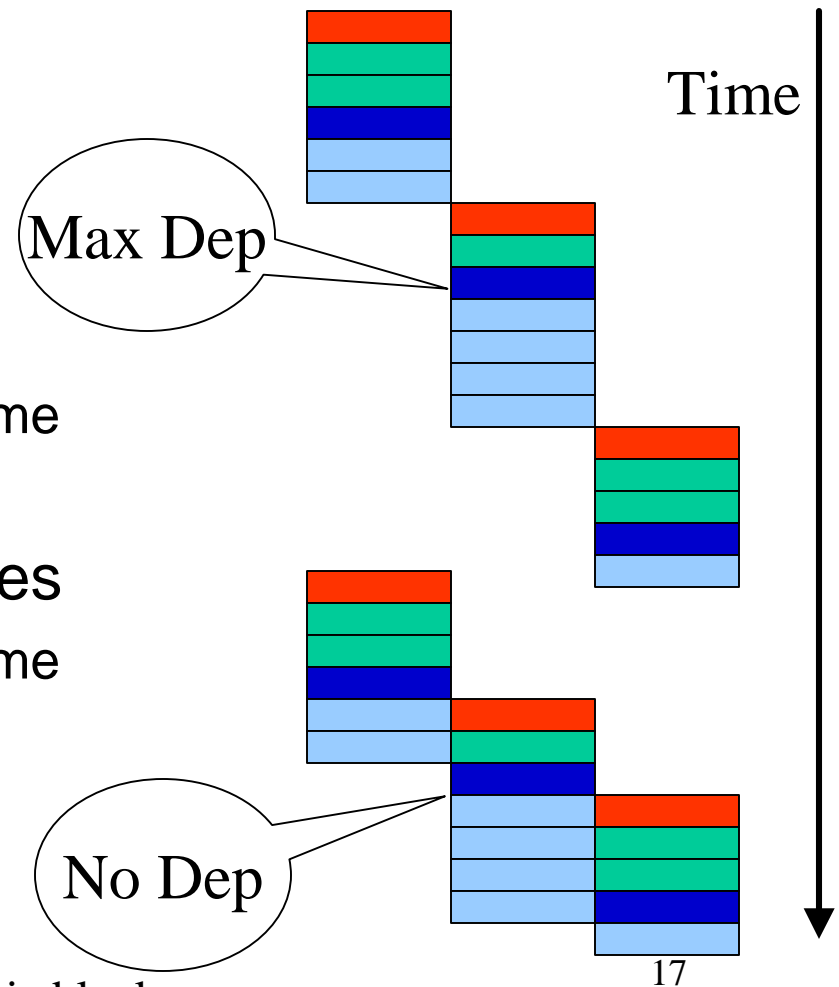
Backup Slide : I-Cache Effects

- Maximum dynamic code expansion is 1.16
 - Worst case : if all memory references are annotated
- Approx. 25% increase in cache misses for ill behaved codes [Lebeck and Wood, '94]
- In practice
 - Far fewer annotated memory references

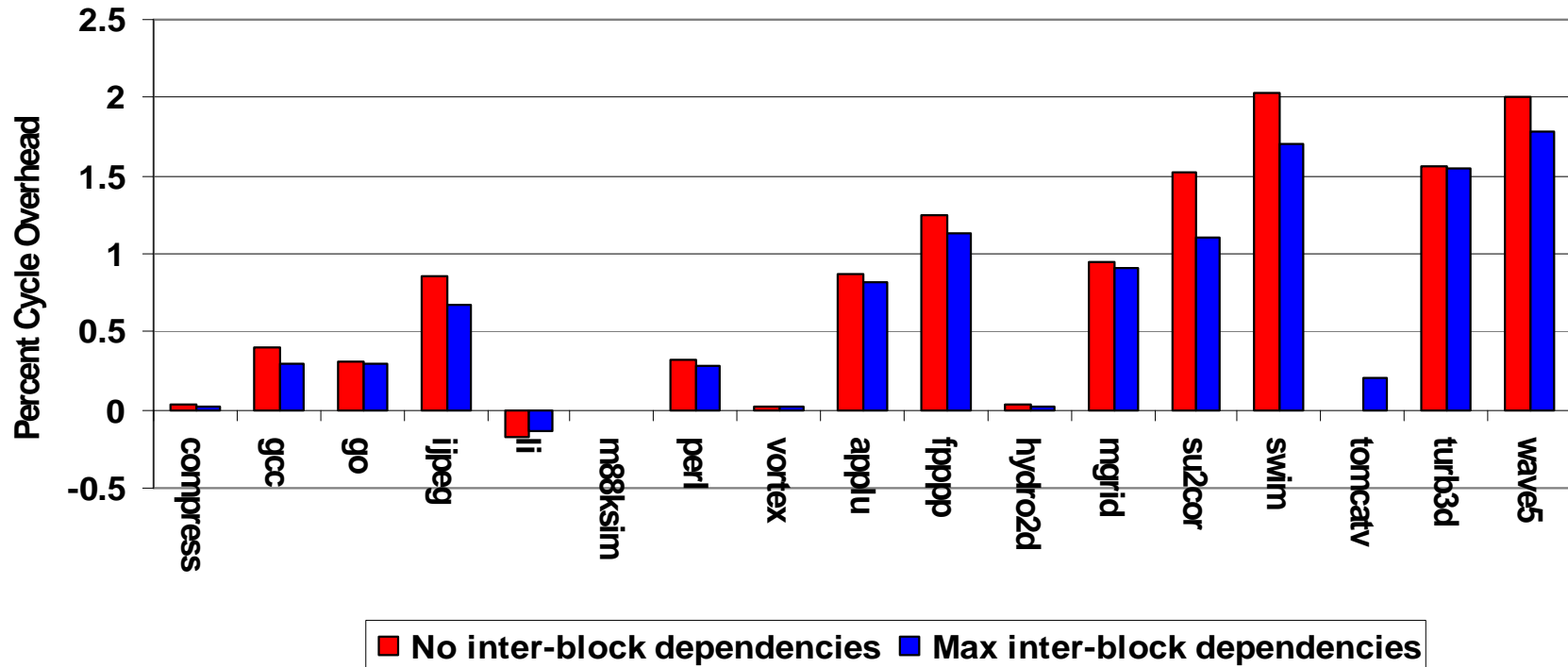
Cycle Time Overhead

- Dependencies across basic blocks:
 - Maximum inter-block dependencies
 - Upper bound of execution time
 - Lower bound of overhead
 - No inter-block dependencies
 - Lower bound of execution time
 - Upper bound of overhead

- Issue of first instruction in basic block
- Issue of last instruction in basic block
- Continued execution of instructions in basic block



Statically Scheduled Processor



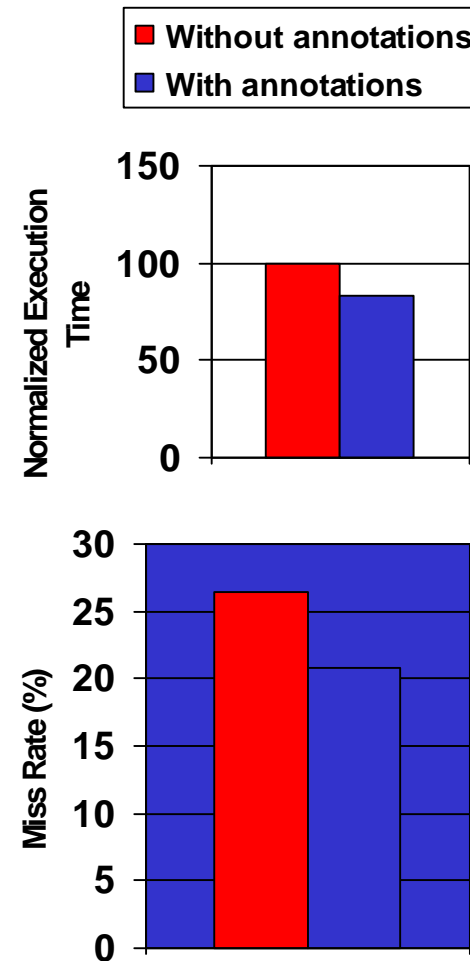
- Integer Codes: 0% to 0.8%
- Floating Point Codes: 0% to 2%

Design Decisions

What?	How?	Static	Dynamic
Instruction (PC)		Abraham et al, '93 Tyson et al, '95	Tyson et al, '95
Address (EA)			McFarling et al. '92 Rivers et al. '96 Johnson et al. '97 Inoue et al. '99

Better Block Replacement

- Insight: some blocks should be retained even if LRU block
- Retain/Release annotations
- A block marked Retain cannot be replaced unless Released
- Bypass cache if no replacement candidate
- epic



Better Block Sizes

- Insight : Implicit prefetch of larger blocks hurts performance
- WordMode/BlockMode annotations
- WordMode annotated references bring in only a word and not the whole block
- pegwit and ijpeg

