

Tree Traversal Scheduling: A Global Scheduling Technique for VLIW/EPIC Processors

Huiyang Zhou

Matt Jennings (BOPS Inc.)

Tom Conte



TINKER Research Group

Department of Electrical & Computer Engineering

North Carolina State University

Presentation Outline

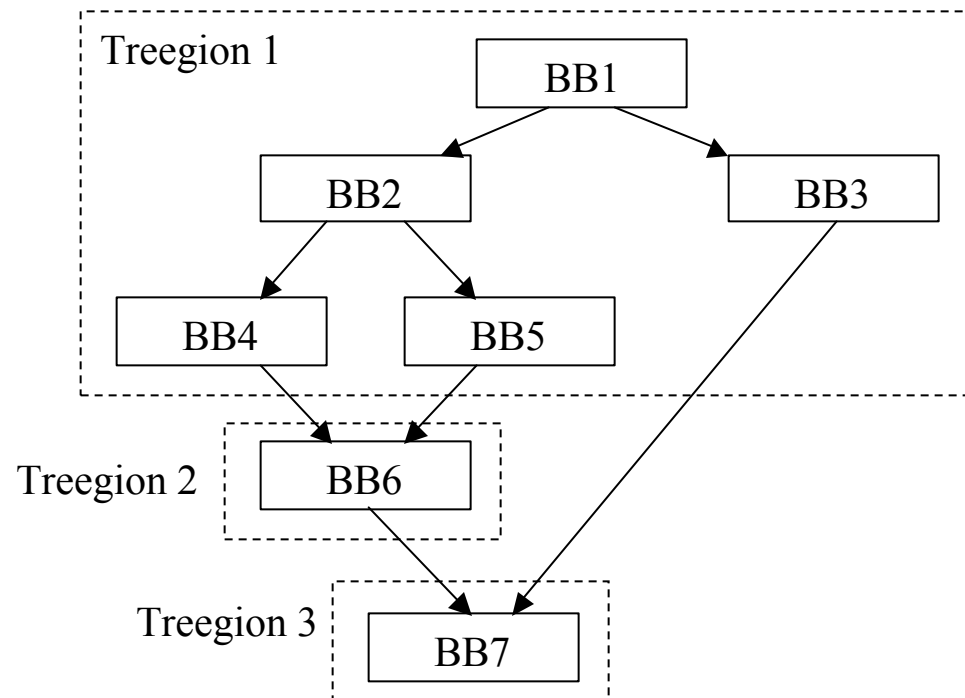
- Introduction
- Tree Traversal Scheduling (TTS) Algorithm
- Efficient Data Flow Analysis in TTS
- Simulation Methodology
- Results
- Conclusions

Introduction

- Global Scheduling
 - Arrange the order of the instructions to minimize the execution time and maintain the program semantics.
 - Schedule instructions beyond the basic block scope.
 - Containing two phases in Treegion framework:
 - **Treegion formation & treegion scheduling**
- Treegion
 - A single-entry / multiple-exit nonlinear region with CFG forming a tree (i.e., no merge points and back-edges in a treegion)
 - Basic scheduling unit in tree traversal scheduling (TTS)

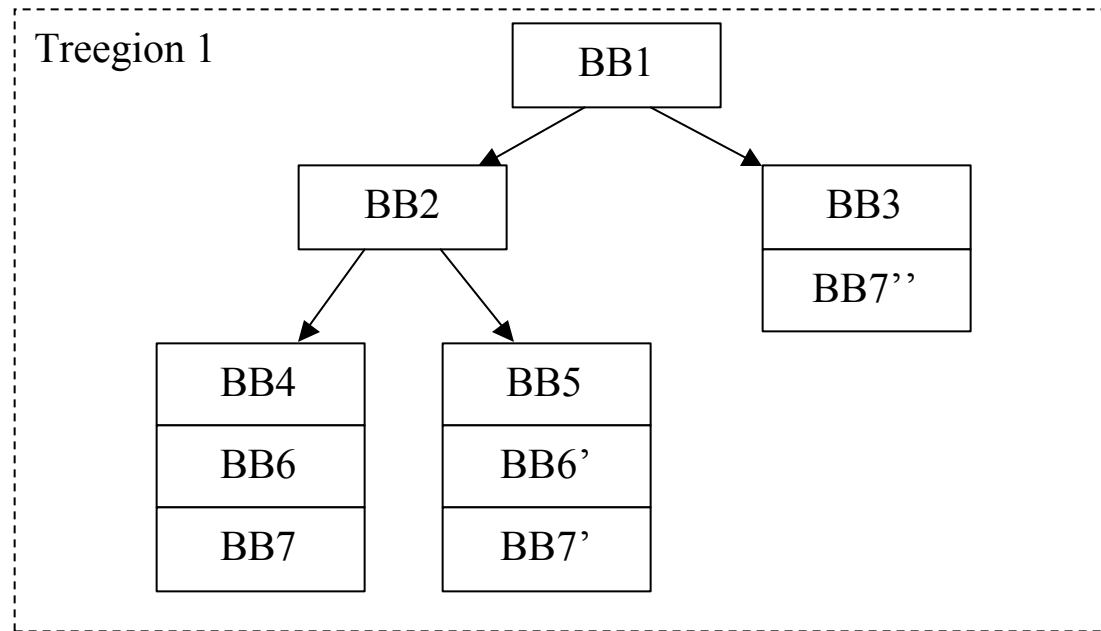
Introduction

- Treeregion Formation
 - Treeregion is formed only based on the program CFG.



Introduction

- Treeregion enlarge optimization: tail duplication at merge points



- Treeregion formation algorithm [W. A. Havanki, et.al. HPCA-4]

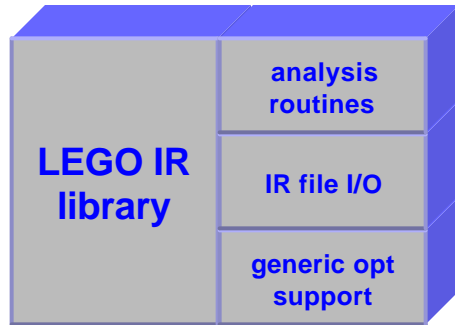
Introduction

- Treeregion Attributes
 - Only depending on the topology of the program's CFG, which makes it suitable for dynamic optimization
 - Containing multiple execution paths
 - Potential to speedup multiple paths
 - Large scheduling scope for ILP extraction
 - High resource utilization for wide issue processors
- LEGO: the ILP research compiler developed by Tinker Research Group at N. C. State University

www.tinker.ncsu.edu



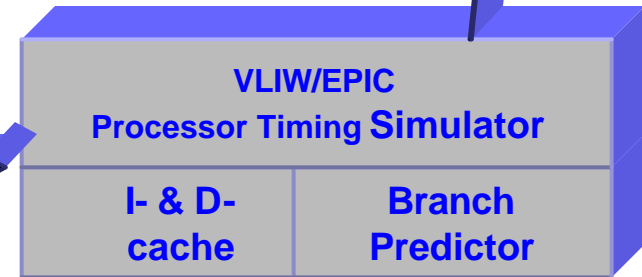
Jointly developed by:
 Prof. Tom Conte
 and his students



Intermediate code:
 Elcor compatible
 PlayDoh semantics
 CFG-based



C (inline emulation)

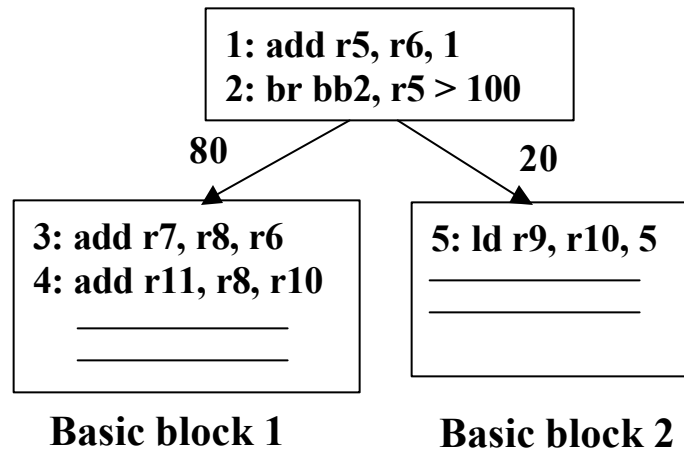


Tree Traversal Scheduling Algorithm

- Objective
 - Speedup each execution path in a tree region
 - If profile information is available, speedup up each path based on its execution frequency
- Common Scheduling techniques
 - List scheduling
 - Renaming
 - Speculative code motion

Speculation in a treeregion

- Over-aggressive speculation
 - May cause the delay to non-speculative instructions due to the contention of machine resources



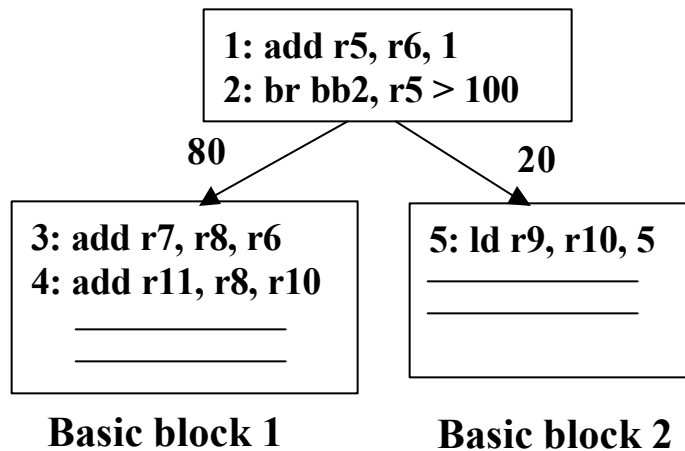
Sch_Time	ALU/BR	ALU/LD
Cycle n:	add r5,r6,1	add r7, r8, r6
Cycle n+1:	add r11, r8, r10	ld r9, r10,5
Cycle n+2:	br bb2, r5>100	

Average execution time: 4 cycles

- Over-conservative speculation
 - The operation latencies are not hidden enough. Less a problem for wide issue processors

Speculation in TTS

- Solve over-aggressive speculation by a cycle based scheduling with prioritizing the instructions according to:
 - (a) Execution frequency
 - (b) Exit count [Deitrich, et.al., MCIRO29] heuristic to resolve ties from (a), and
 - (c) Data dependence height to resolve ties from (b)
- Allow early schedule of branches even with downward code motion



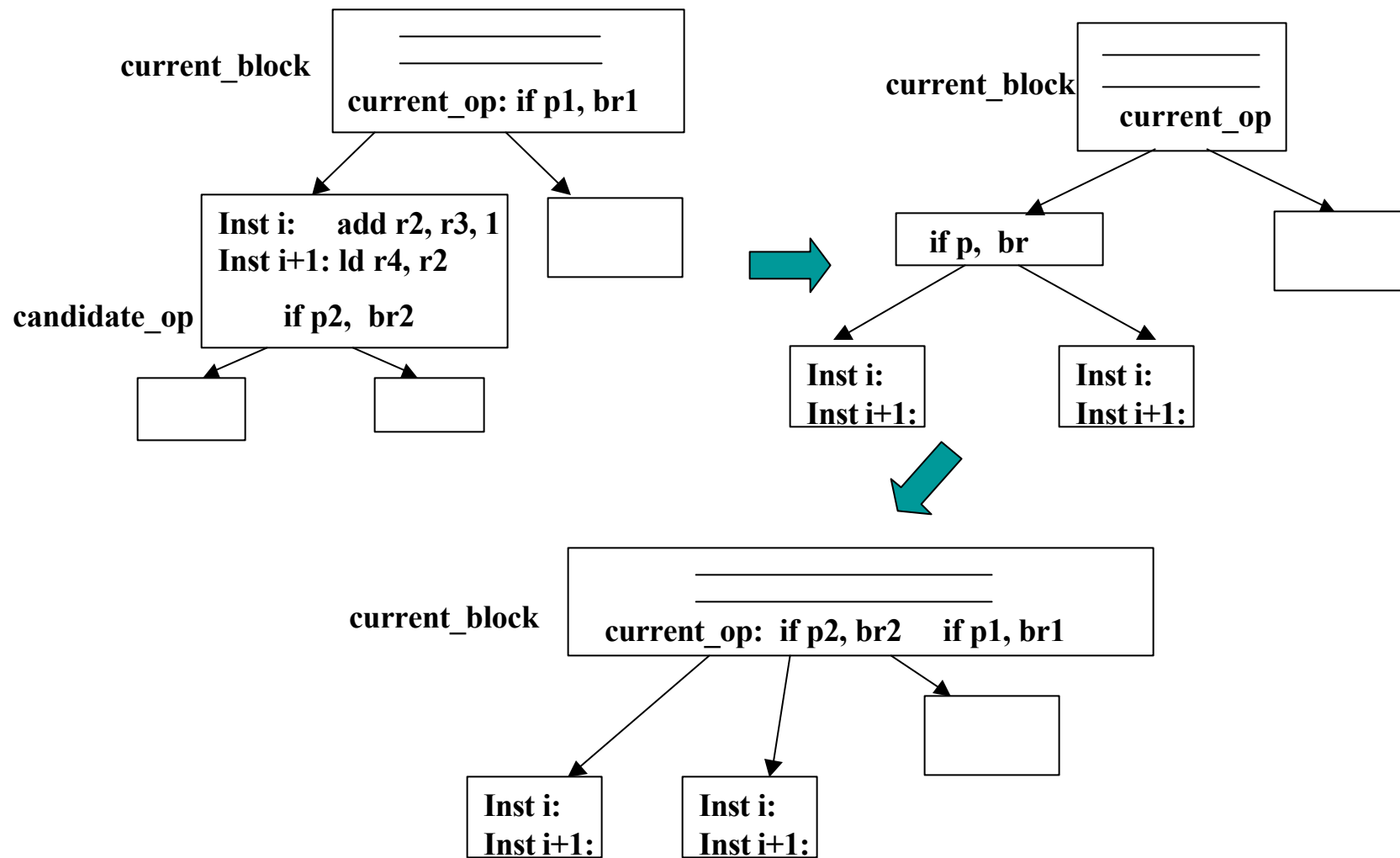
Sch_Time	ALU/BR	ALU/LD
Cycle n:	add r5,r6,1	add r7, r8, r6
Cycle n+1:	br bb2, r5>100	add r11,r8,r10
Cycle n+2:	(ld r9, r10,5)	

Average execution time: $3 + 2*0.2 = 3.4$ cycles

TTS Algorithm

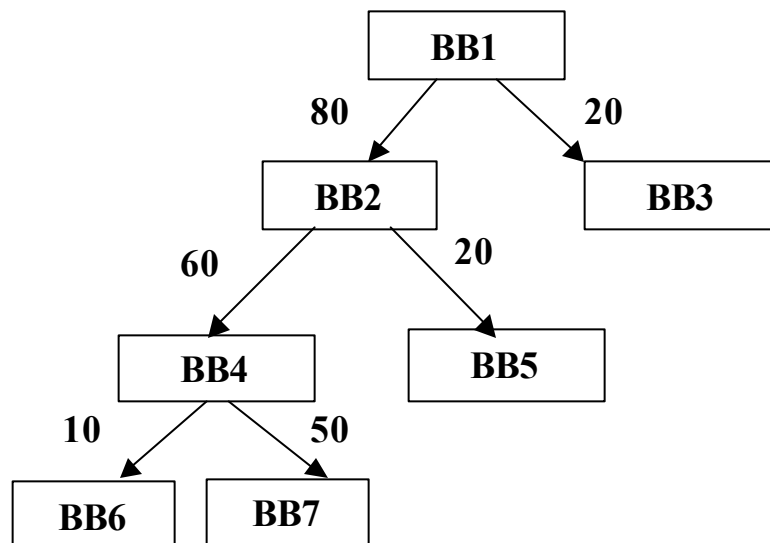
- Step 1. Construct the control/data dependence graph and perform instruction ordering.
- Step 2. Cycle scheduling of the instructions in a treegion
 - a. For each cycle, select the candidate operation according to the order of Step 1.
 - b. If machine resource is available for the candidate operation, check for whether the scheduling of the candidate is speculative.
 - c. For the speculative code motion, check whether the renaming is necessary to support the speculation.
 - d. If the candidate is a branch operation, downward code motion and multiway branch transformation may result.

Scheduling of Branches in TTS



Logic View of TTS

1. For a treegion, sort the basic blocks according to a depth-first traversal order with the child block selected with highest execution frequency.
2. Start list scheduling at the root basic block.
3. During the scheduling of a basic block, consider speculation for instructions dominated by this basic block.
4. After scheduling the block-ending branch, traverse to the next basic block and go back to 3.



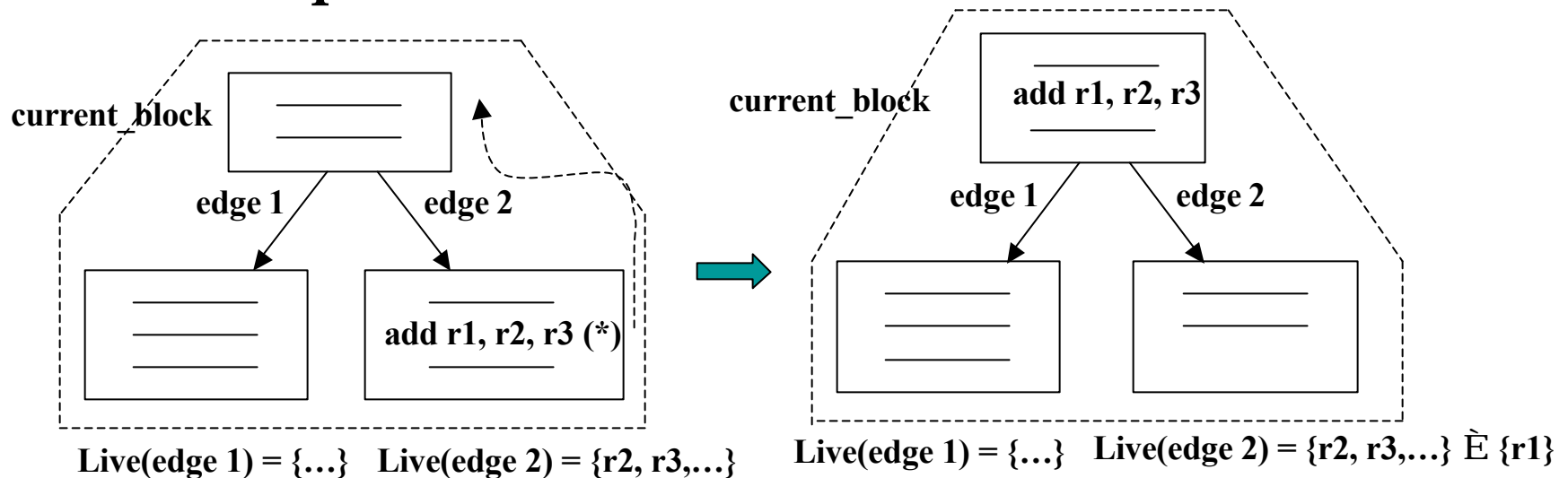
- Traversal order:
BB1, BB2, BB4, BB7, BB6, BB5, BB3
- High resource utilization from speculation of dominated instructions.
- Reducing resource contention: e.g., when scheduling BB4, the instructions in BB5 will not compete for the resources.

Incremental Data Flow Analysis in TTS

- Motivation
 - The data flow analysis (liveness, reaching definition) obsolete due to code motions in TTS.
 - Recalculation takes too much computation time
 - Solution: incremental update (not accurate but conservative)
- Data flow analysis based on different categories of renaming (based on the renaming scope)
 - Speculation without renaming
 - Speculation with local renaming
 - Speculation with renaming with a copy
 - Speculation with global renaming

Data flow analysis for speculative code motion without renaming

• Example

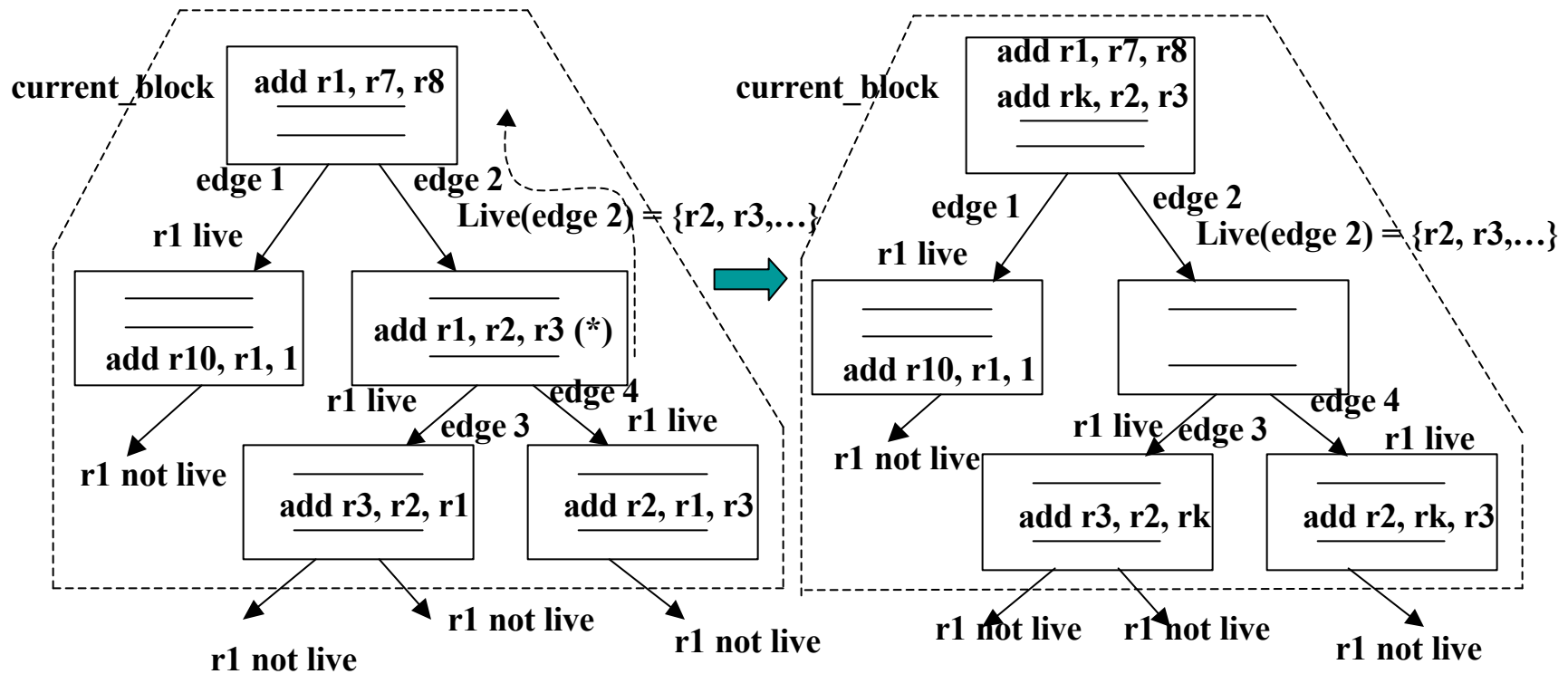


• Incremental update

- **liveness is extended for the destination operand and added to each edge that the instruction traverse**
- **Conservative liveness may cause unnecessary renaming (most of them are simple to process)**
- **No changes in reaching definitions**

Data flow analysis for speculative code motion with local renaming

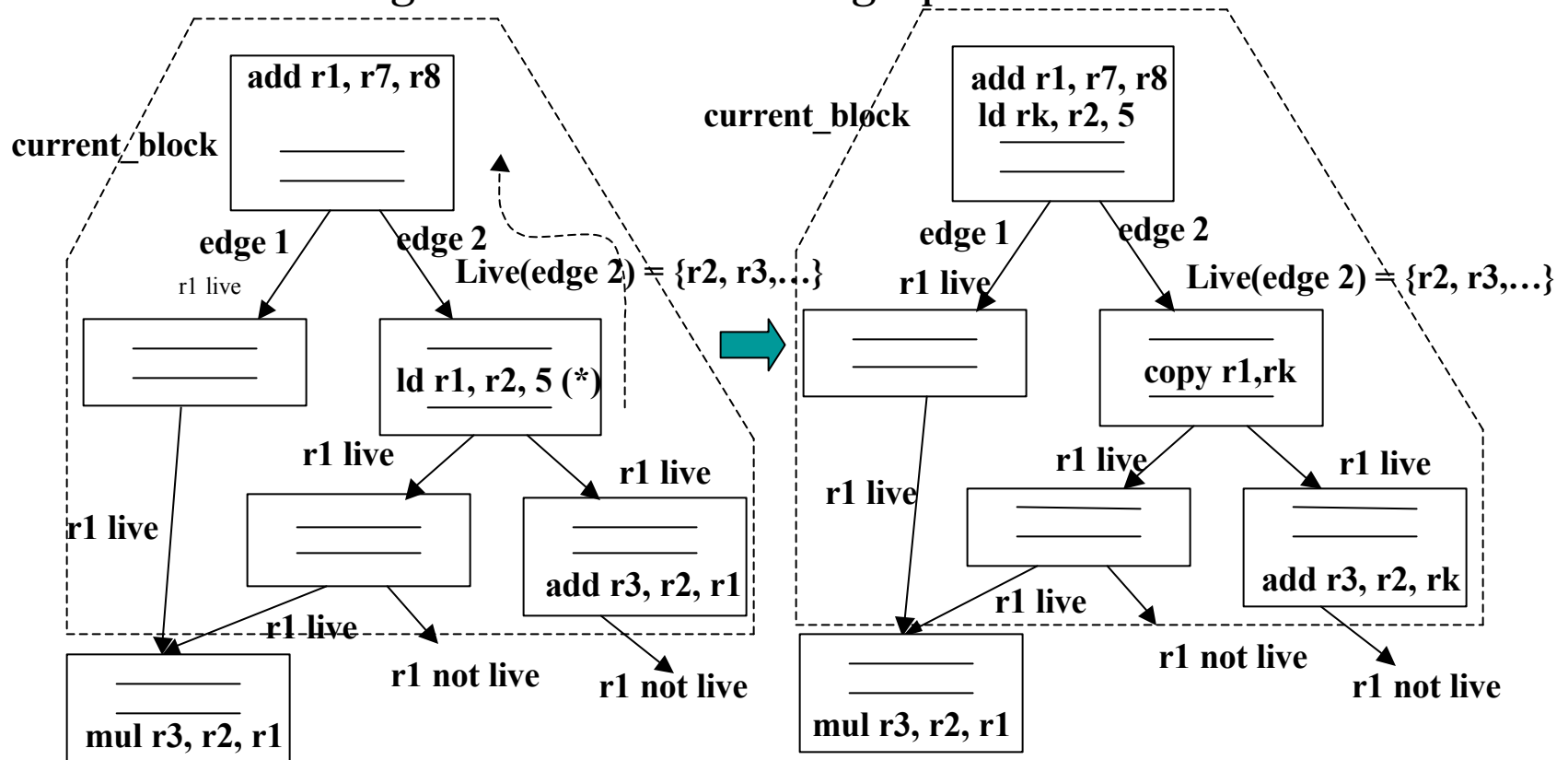
- Local renaming is used when the renaming scope is within the treeregion



- Incremental update
 - No change to liveness and reaching definitions**

Data flow analysis for speculative code motion with renaming with a copy

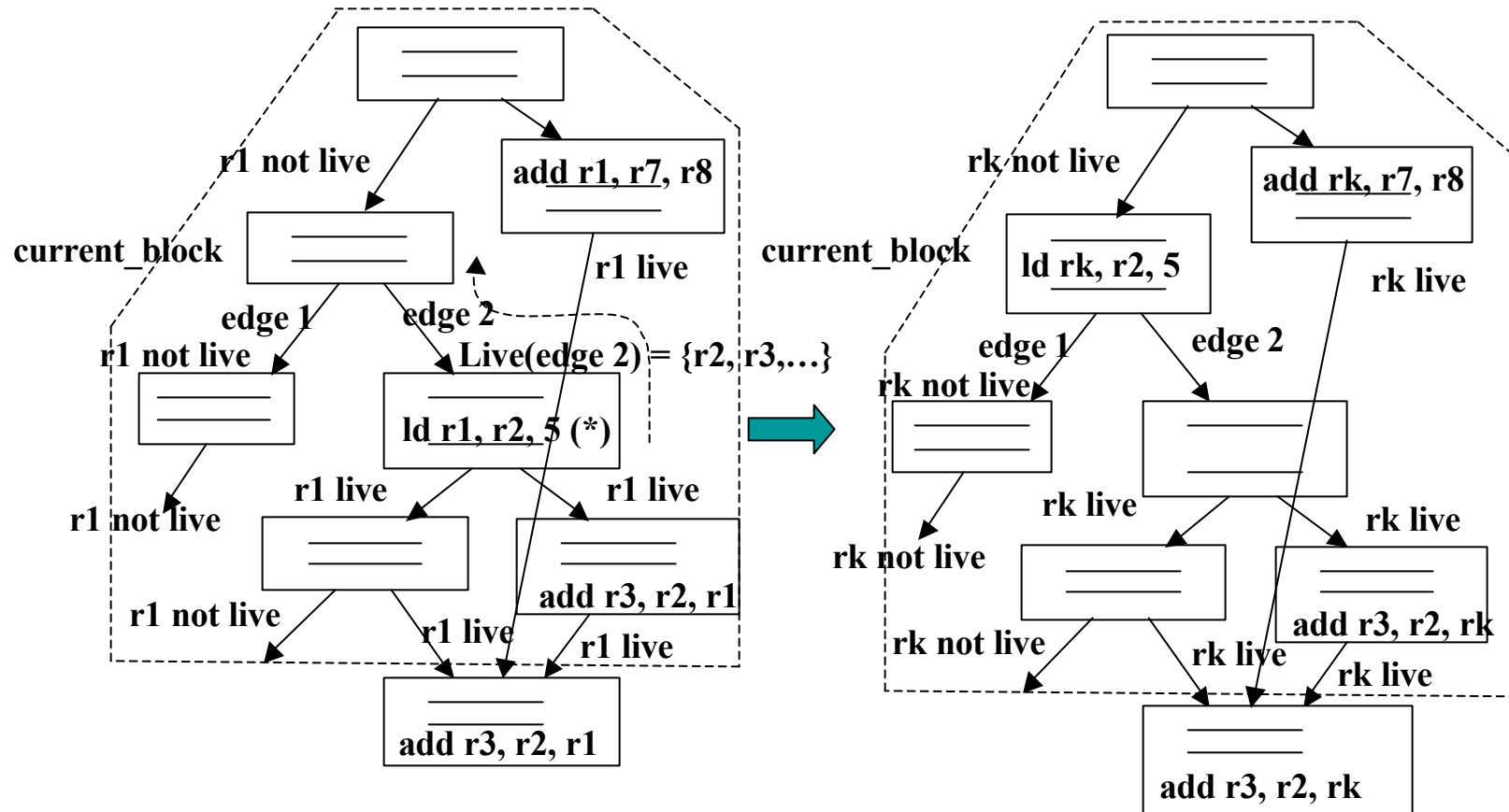
- Renaming with a copy is used when the operand to be renamed is live outside the treeregion and there is a 'merge' problem.



- Incremental update
 - No change in liveness and accurate update of reaching definition**

Data flow analysis for speculative code motion with global renaming

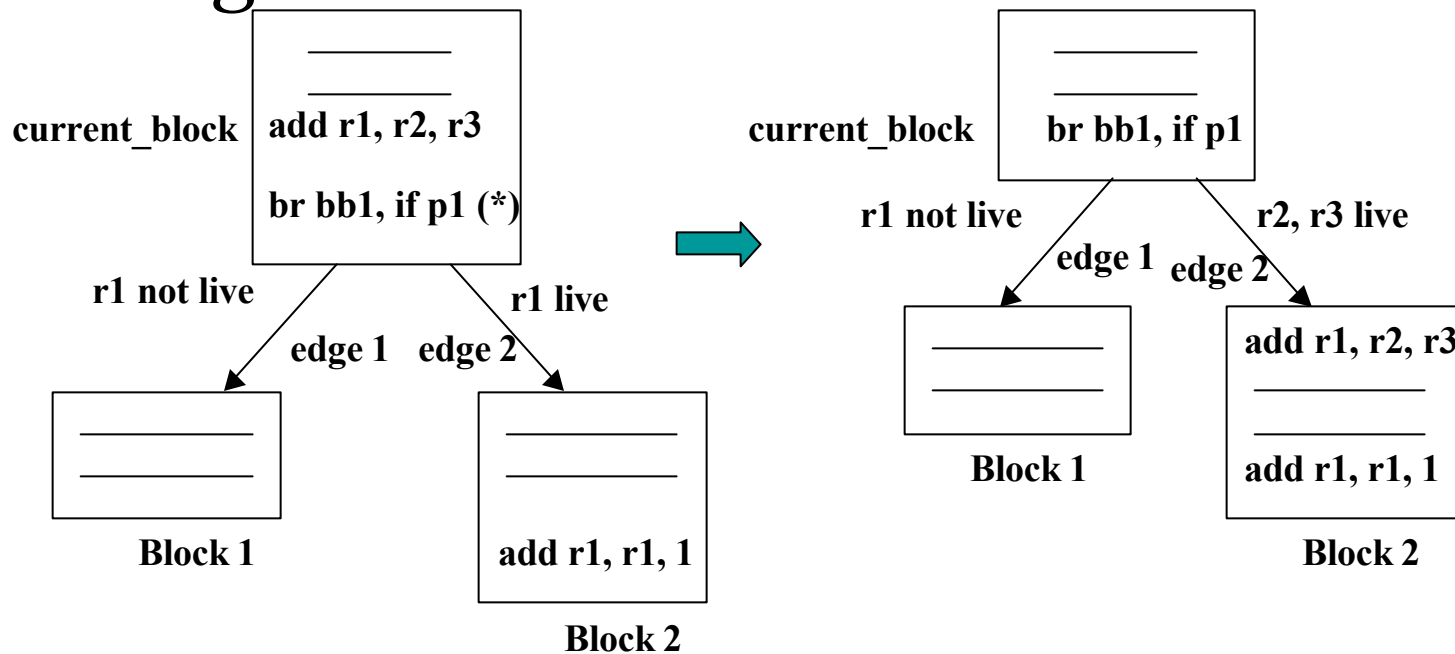
- **Global renaming is used when the renaming scope is beyond the treeregion and there is no merge problem**



- **Update**
 - **Recalculate the liveness at procedural scope. No changes in reaching definitions.**

Data flow analysis for downward code motion

- Result from the early schedule of block-ending branches



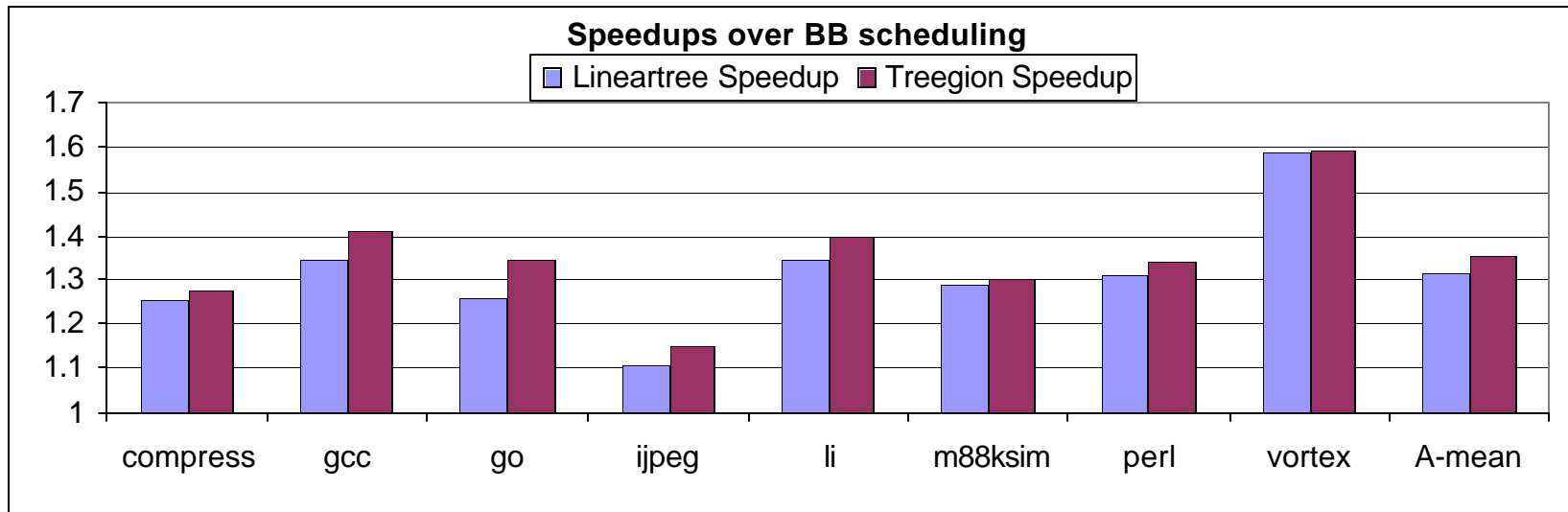
- Incremental update
 - For each downward moved instruction, add its source operands into liveness set and remove the its destination operand from liveness set.
 - Processed in reverse program order

VLIW/EPIC Processor Model Specification

	Specification
Execution	Dispatch/Issue/Retire bandwidth: 8; Universal function units: 8; Operation latency: ALU, ST, BR: 1 cycle; LD, floating-point (FP) add/subtract: 2 cycles; FP multiply/divide: 3 cycles
I-cache	Compressed (zero-nop) and two banks with 32KB each bank (Direct Mapped) [Conte et. al, MICRO29]. Line size: 16 operations with 4 bytes each operation. Miss latency: 12 cycles
D-cache	Size/Associativity/Replacement: 64KB/4-way/LRU Line size: 32 bytes Miss Penalty: 14 cycles
Branch Predictor:	G-share style Multi-way branch prediction [Menezes, et. al., PACT'97, Hoogerbrugge, PACT'00] Branch prediction table: 2^{14} entries; Branch target buffer: 2^{14} entries/8-way/LRU Branch misprediction penalty: 10 cycles

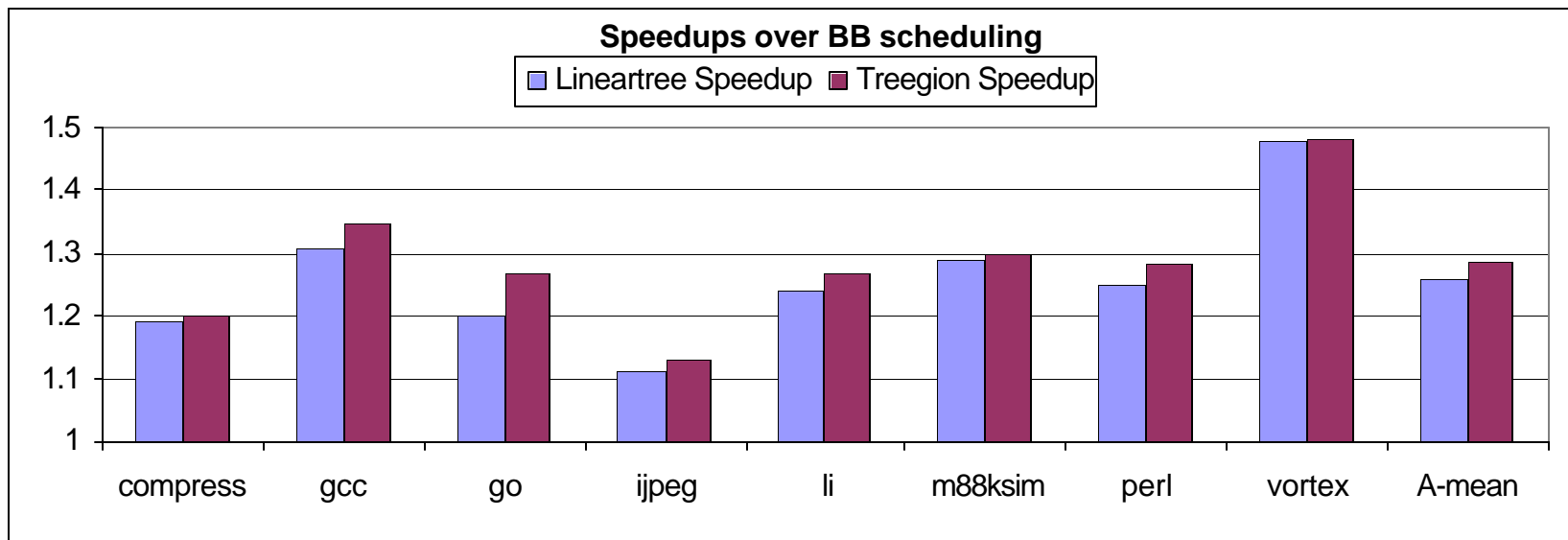
Speedup Results

- Speedup with ideal I-cache, D-cache, and branch predictor

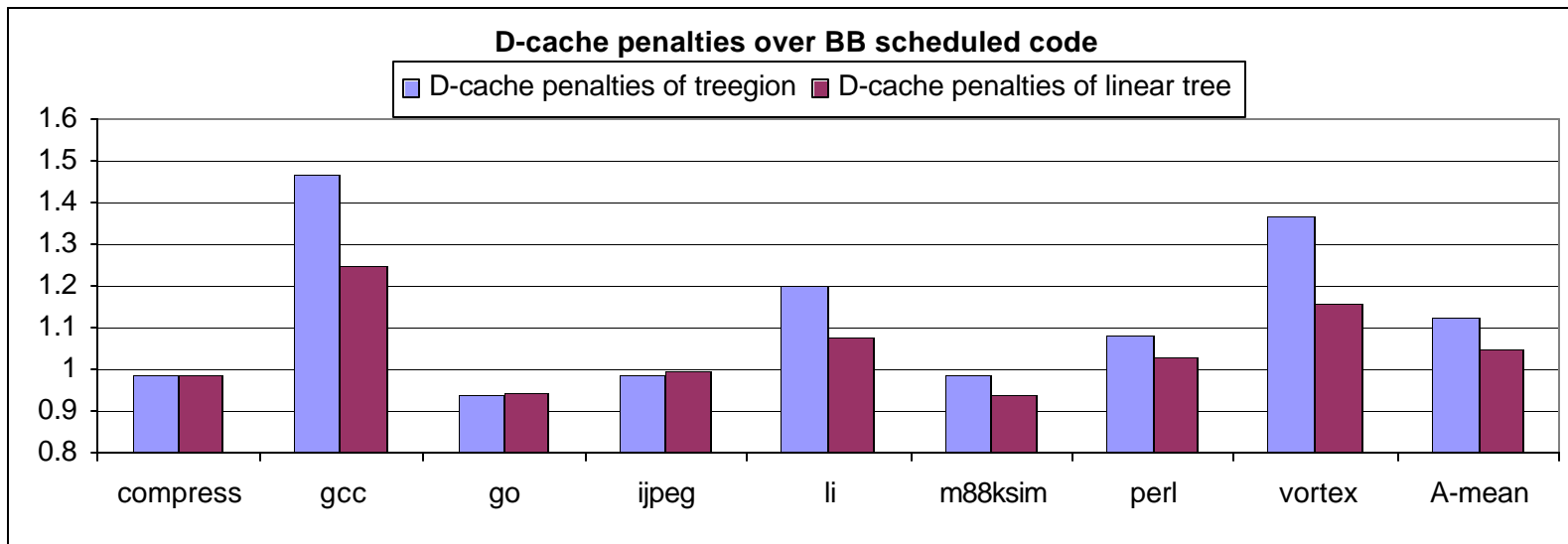
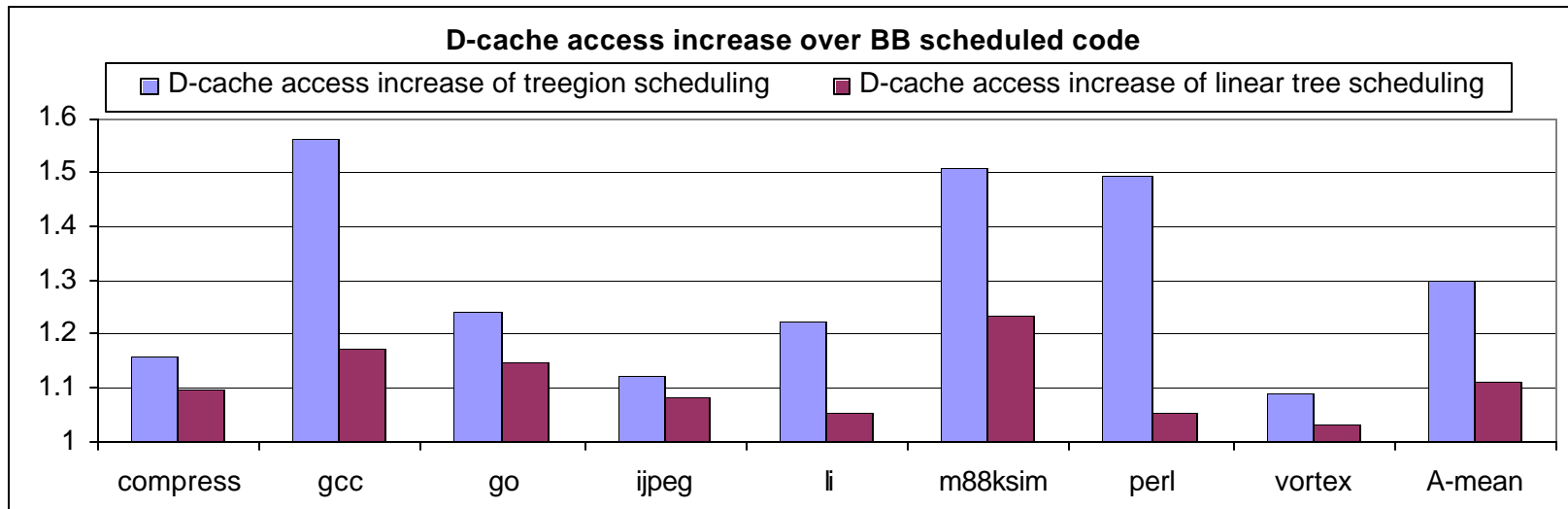


Speedup Results

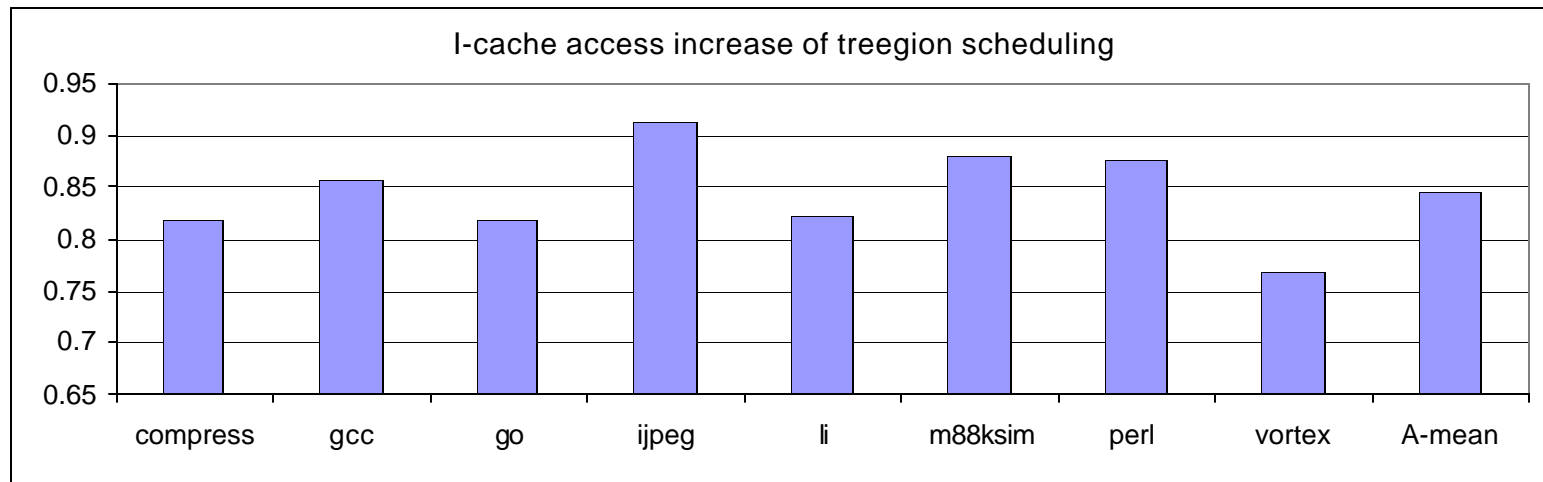
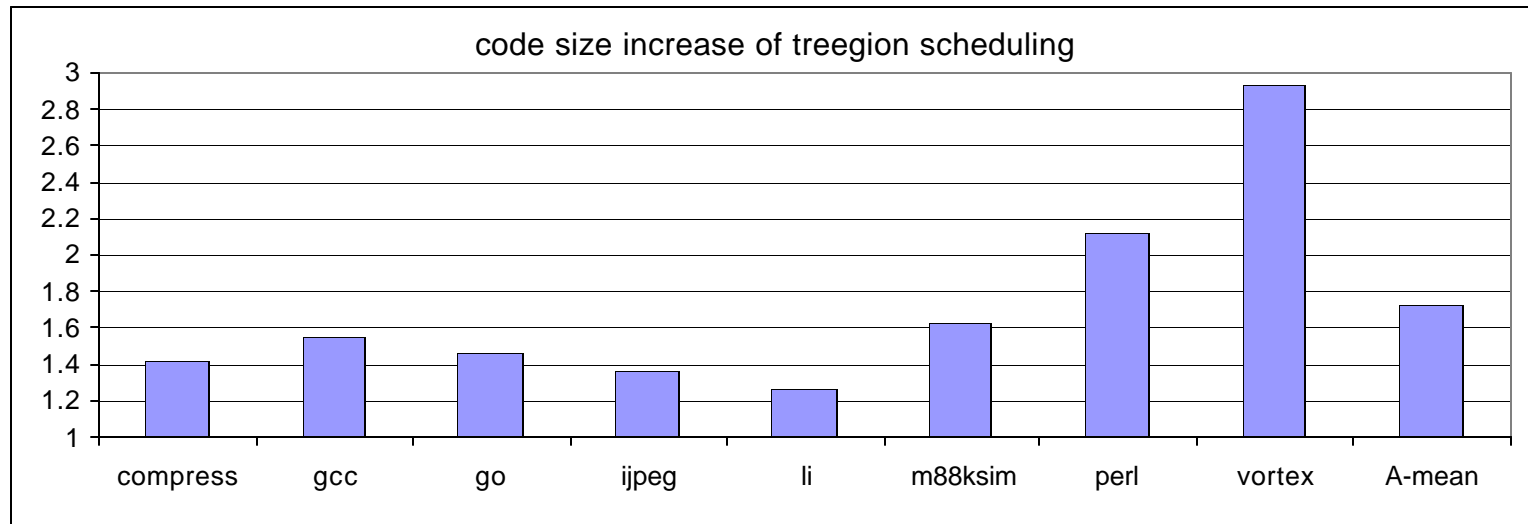
- Speedup with realistic I- & D- cache, Branch Predictor



D-Cache Performance



I-Cache Performance



Conclusion

- Significant speedup from tree traversal scheduling
- To fully take advantage of load speculation, the stall-on-use technique should be used in the in-order pipeline.
- Fewer multi-ops as a result of TTS results in fewer I-cache accesses while code expansion due to treeregion formation usually introduces higher miss rates

Contact Information

Huiyang Zhou

hzhou@eos.ncsu.edu

Matt Jennings

MattJ@bops.com

Tom Conte

conte@eos.ncsu.edu



TINKER Research Group
North Carolina State University
www.tinker.ncsu.edu