Characterizing the Impact of Predicated Execution on Branch Prediction

Scott A. Mahlke
Richard E. Hank
Roger A. Bringmann
John C. Gyllenhaal
David M. Gallagher
Wen-mei W. Hwu

Center for Reliable and High-Performance Computing
University of Illinois
Urbana-Champaign, IL

Problem Statement

- Performance of VLIW and superscalar processors dependent on the amount of extracted ILP
- Branches limit available ILP by imposing control flow dependences
- High frequency of branches in non-numerical applications
- ILP within basic blocks very small
  - 2-5 available instructions
  - Instructions have data dependences

Dynamic Instruction Mix - INT Benchmarks

Benchmark	Load	Store	IALU	FALU	Branch
008.espresso	0.228	0.027	0.520	0.000	0.225
022.li	0.323	0.137	0.226	0.000	0.313
023.eqntott	0.223	0.004	0.402	0.000	0.371
026.compress	0.201	0.098	0.450	0.000	0.252
072.sc	0.279	0.032	0.349	0.008	0.332
085.cc1	0.210	0.117	0.349	0.000	0.323
cccp	0.189	0.079	0.376	0.000	0.356
cmp	0.265	0.027	0.118	0.000	0.589
eqn	0.197	0.119	0.319	0.000	0.365
grep	0.202	0.092	0.146	0.000	0.560
lex	0.253	0.179	0.119	0.000	0.450
qsort	0.204	0.186	0.379	0.000	0.230
tbl	0.311	0.033	0.279	0.000	0.377
wc	0.123	0.131	0.331	0.000	0.415
yacc	0.240	0.035	0.349	0.000	0.376
Avg.	0.230	0.086	0.471	0.000	0.369

Problem Statement (Cont.)

- Branch prediction
  - Effective when continually predict correctly
  - Large penalty for mispredictions
  - Limited to extracting ILP along 1 path of control
- Limited branch handling capability per cycle may put automatic ceiling on ILP
  - 1 branch per cycle $\Rightarrow$ 2-4 max IPC
  - Execute multiple branches per cycle, but ...
    * Pipelined implementation is complex
    * Multiple branch predictions per cycle vs sacrifice BP accuracy
  - Duplicating arithmetic function units much simpler

Outline of this Talk

- Overview of predicated execution
  - Execution model
  - Architectural support
- Compiler support for predicated execution
  - Hyperblock formation
  - Case study of branch prediction effects
- Experimental results
- Conclusions

Predicated Execution

- Conditional execution of instructions based on a boolean source operand
- Execution model
  - load r1, r2, r3 (p)
  - if the value of p is 1, instruction executed normally
  - if the value of p is 0, instruction treated as a no_op
- Allows the compiler to remove branches using if-conversion

Architectural Extensions

- 3 major components

  1. Predicate register file - array of 1 bit registers
  2. Pipeline operation - convert instructions to NOP's if their predicate is false
     - Prevent issuing instructions with p=0
     - Prevent writeback of instructions with p=0
  3. Instruction set architecture
     - Predicate defining instructions - HPL Playdoh [Kathail, Schlansker, Rau 94]
     - Extra operand field to specify the predicate

HPL Playdoh Predicate Defining Instructions

pred_$< cmp >$ P1_$< type >$, P2_$< type >$, src1, src2 (P_{in})

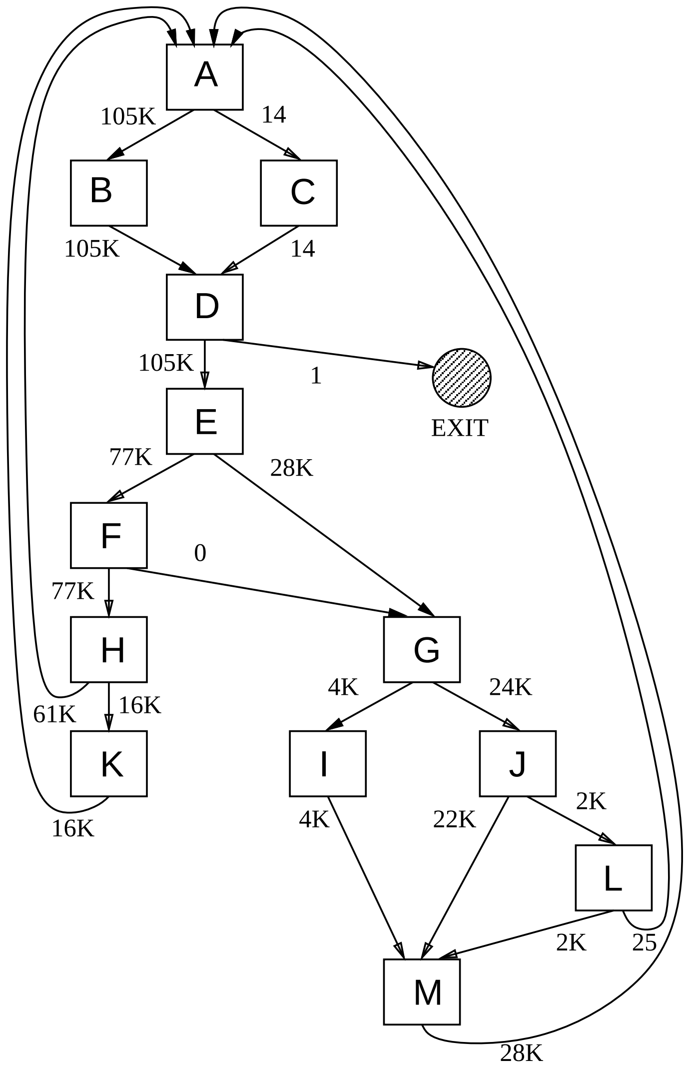
- $< cmp >$ – comparison type, $=, \neq, >, \leq, \ldots$
- $< type >$
  1. Unconditional ($U, \overline{U}$)
  2. Conditional ($C, \overline{C}$)
  3. OR-type ($OR, \overline{OR}$)
  4. AND-type ($AND, \overline{AND}$)
- P_{in} – input predicate

Compiler Support for Predicated Execution

- Hyperblock structure
  - Collection of connected basic blocks
  - Single block dominates all blocks (entry block)
  - Control flow may only enter hyperblock at entry block
  - Control flow may exit hyperblock at any number of locations
  - Control flow between blocks in hyperblock is eliminated
- Hyperblock formation - 5 step procedure

Hyperblock Formation Example (wc)

```
linect = wordct = charct = token = 0;
for (;;) {
   if (--(fp)->cnt < 0)
      c = filbuf(fp);
   else
      c = *(fp)->ptr++;
   if (c == EOF)
      break;
   charct++;
   if ((' ' < c) && (c < 0177)) {
      if (! token) {
           wordct++;
           token++;
      }
      continue;
   }
   if (c == '\n')
      linect++;
   else if ((c != ' ') && (c != '\t'))
      continue;
   token = 0;
}
```



Branch Characteristics of wc

- Dynamic branches

Loop		Non-loop		
Inner	Outer	Inner	Outer	StrLn
228K	2	344K	3	4

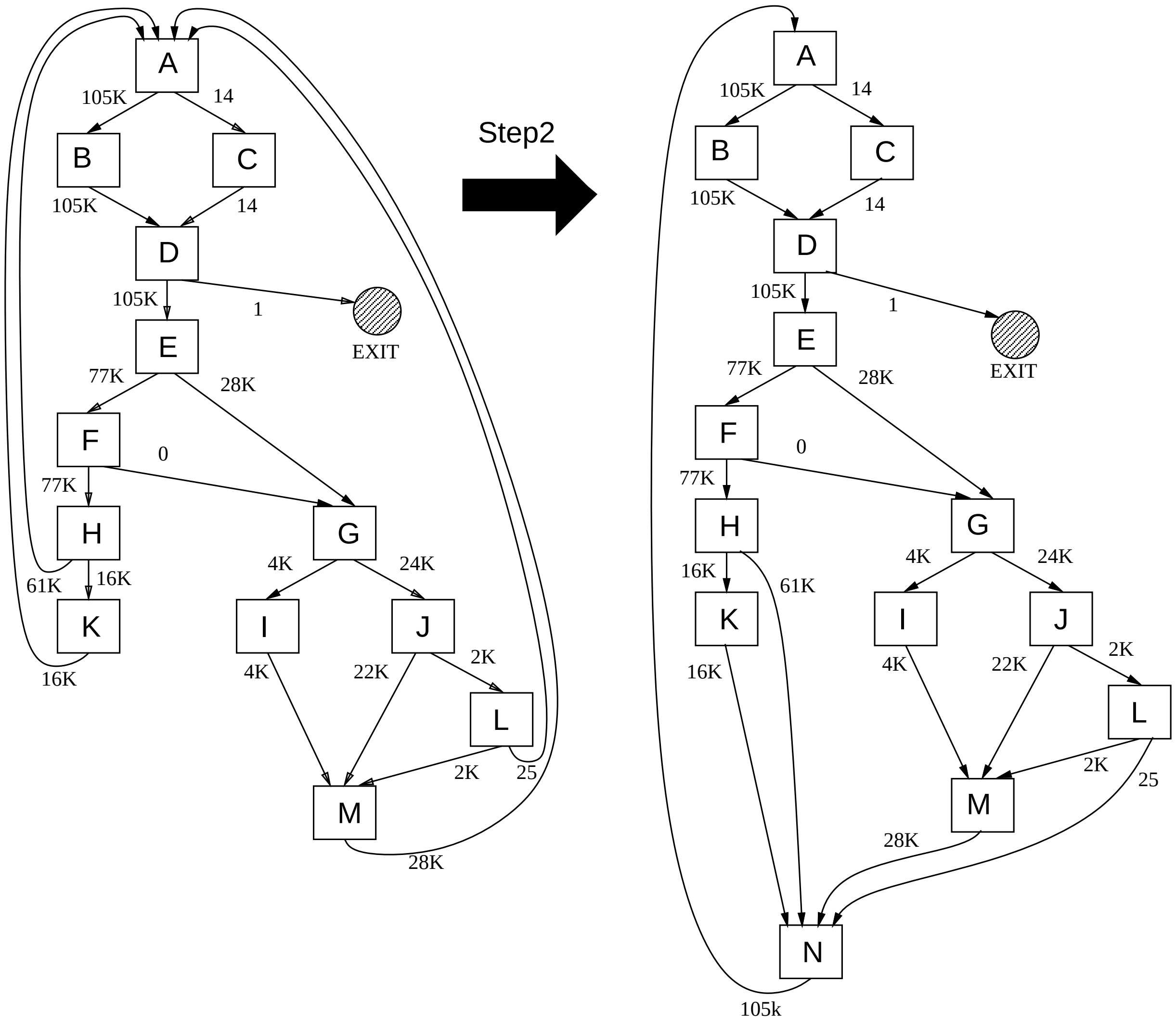
- Dynamic mispredictions

	Loop		Non-loop			
	Inner	Outer	Inner	Outer	StrLn	MPR
Counter	19.5K	1	32.6K	0	2	0.09
Profile	16.2K	1	32.9K	0	2	0.09
Btc	57.7K	1	37.1K	0	2	0.17

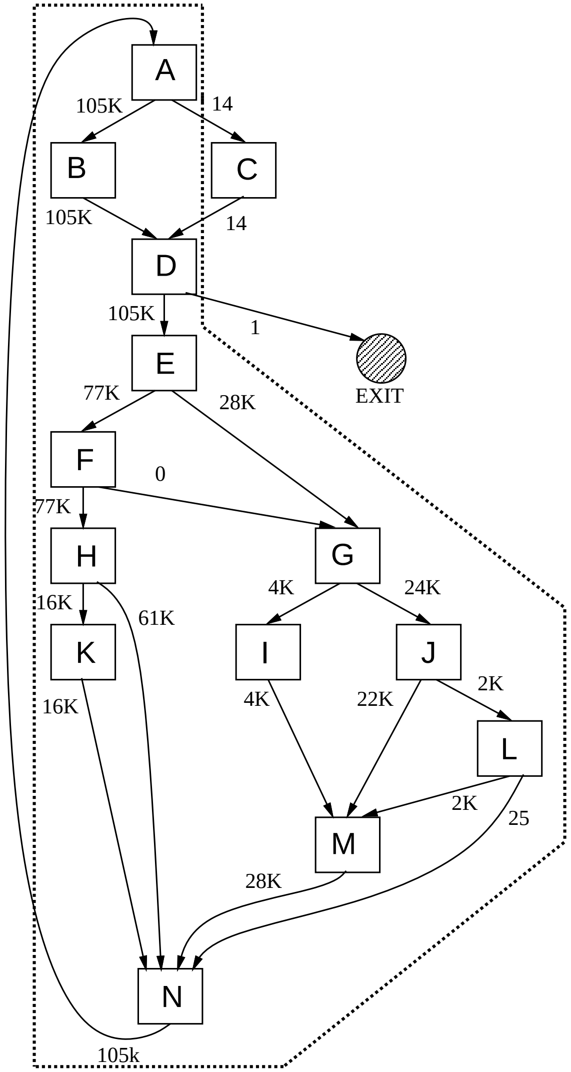
- Misprediction coverage

10	30	50	70	90	100
0.06	0.06	0.06	0.06	0.08	0.56

Hyperblock Formation Step 2

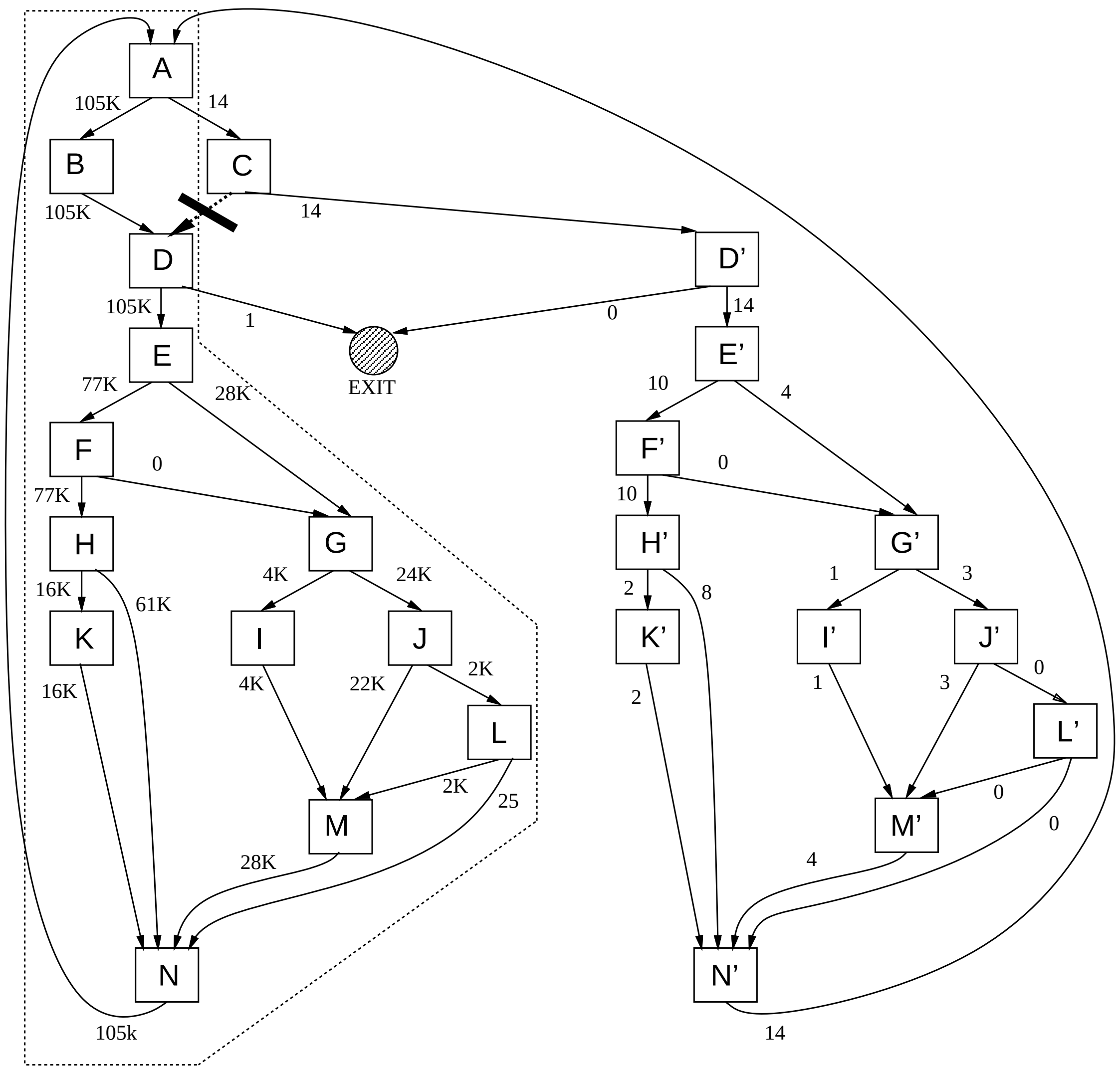


Hyperblock Formation Step 3



1. A-B-D-E-F-H-N
2. A-B-D-E-F-H-K-N
3. A-B-D-E-G-J-M-N
4. A-B-D-E-G-J-L-M-N
5. A-B-D-E-G-I-M-N
6. A-B-D-E-G-J-L-N
7. A-B-D
8. A-C-D-E-F-H-N
9. A-C-D-E-F-H-K-N
10. A-C-D-E-G-J-M-N
11. A-C-D-E-G-J-L-M-N
12. A-C-D-E-G-I-M-N
13. A-C-D-E-G-J-L-N
14. A-C-D
15. A-B-D-E-F-G-I-M-N
16. A-B-D-E-F-G-J-M-N
17. A-B-D-E-F-G-J-L-M-N
18. A-B-D-E-F-G-J-L-N
19. A-C-D-E-F-G-I-M-N
20. A-C-D-E-F-G-J-M-N
21. A-C-D-E-F-G-J-L-M-N
22. A-C-D-E-F-G-J-L-N

Hyperblock Formation Step 4



Hyperblock Formation Step 5

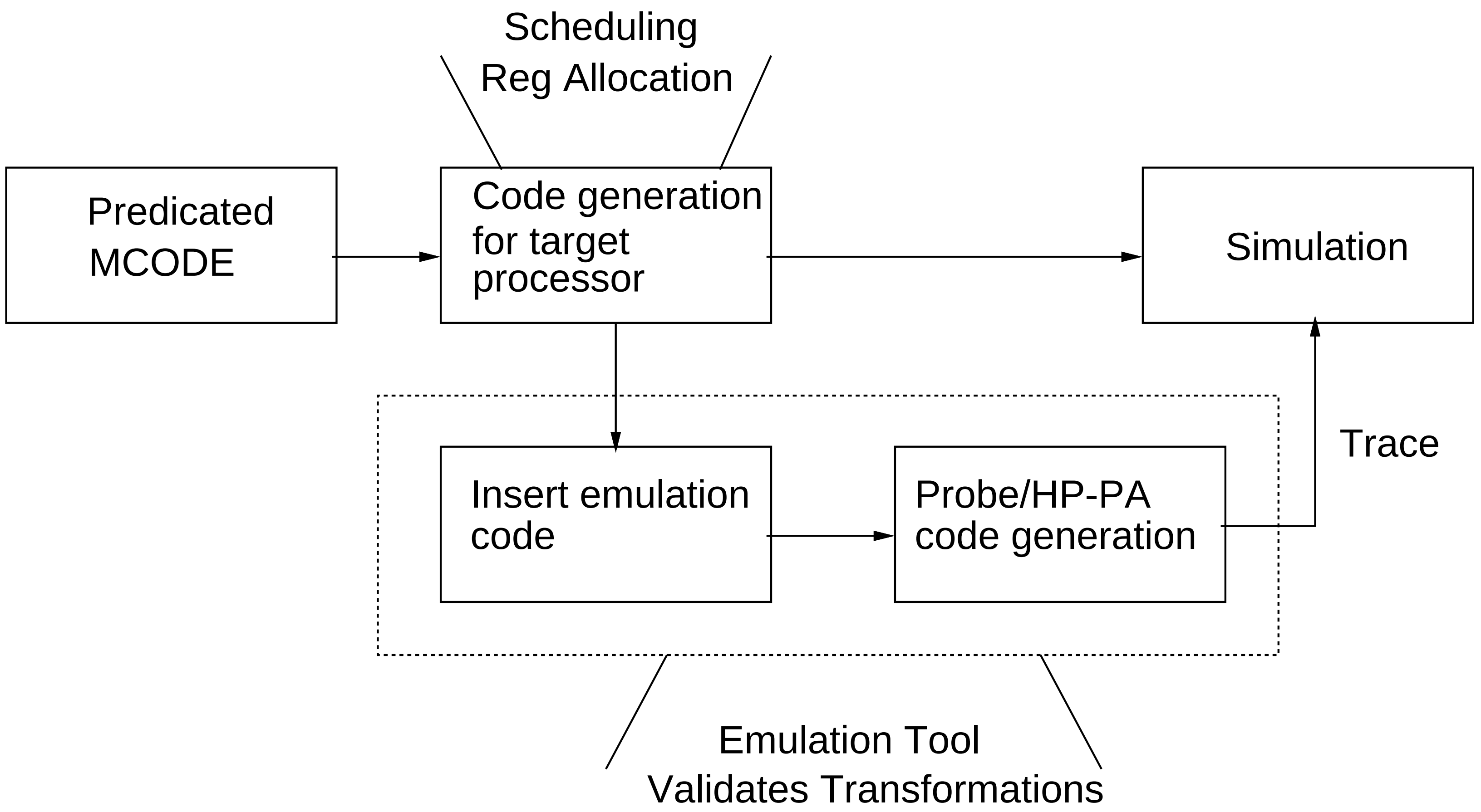
Loop:
```
pred_clr  p4, p6
ld  r98, mem(r3 + 0)
add  r27, r98, -1
st  mem(r3 + 0), r27
blt  r98, 1, LC
ld  r30, mem(r3 + 4)
add  r29, r30, 1
st  mem(r3 + 4), r29
ld  r4, mem(r30 + 0)
beq  r4, -1, EXIT
ld  r33, mem(r73 + 0)
add  r32, r33, 1
st  mem(r73+0), r32
pred_ge  p4(OR), p1(Ū), 32, r4
pred_ge  p4(OR), p2(Ū), r4, 127  (p1)
pred_eq  p3(U), -, 0, r2 (p2)
pred_eq  p6(OR), p5(Ū), r4, 10  (p4)
pred_eq  p7(U), -, r4, 10 (p4)
pred_eq  p6(OR), p8(Ū), r4, 32  (p5)
ld  r36, mem(r72+0)  (p3)
add  r35, r36, 1  (p3)
st  mem(r72+0), r35  (p3)
add  r2, r2, 1  (p3)
ld  r39, mem(r71 + 0)  (p7)
add  r38, r39, 1  (p7)
st  mem(r71 + 0), r38  (p7)
pred_eq  p6(OR), -, r4, 9  (p8)
mov  r2, 0  (p6)
jump  Loop
```

Peformance Summary for Wc

8-issue, 1BR per cycle
Perfect I/D caches

	Base	Pred
Dynamic Branches	572K	315K
Mispredictions	52K	56
MPR	9.0%	0.0%
Distance betw Brs	2.0	6.8
Distance betw MPs	46	56K
Performance (8 issue)	637K	289K

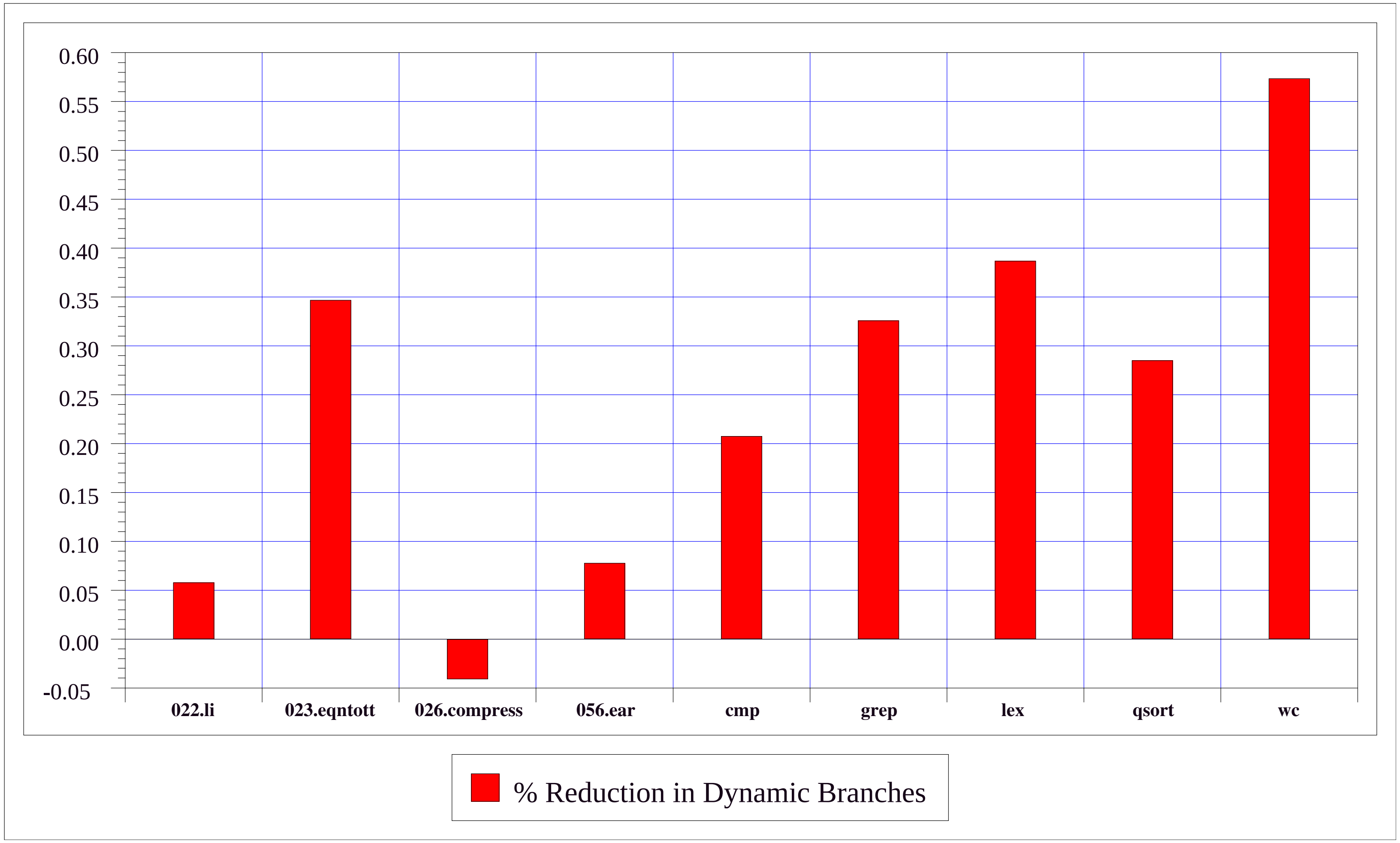
Experimental Evaluation



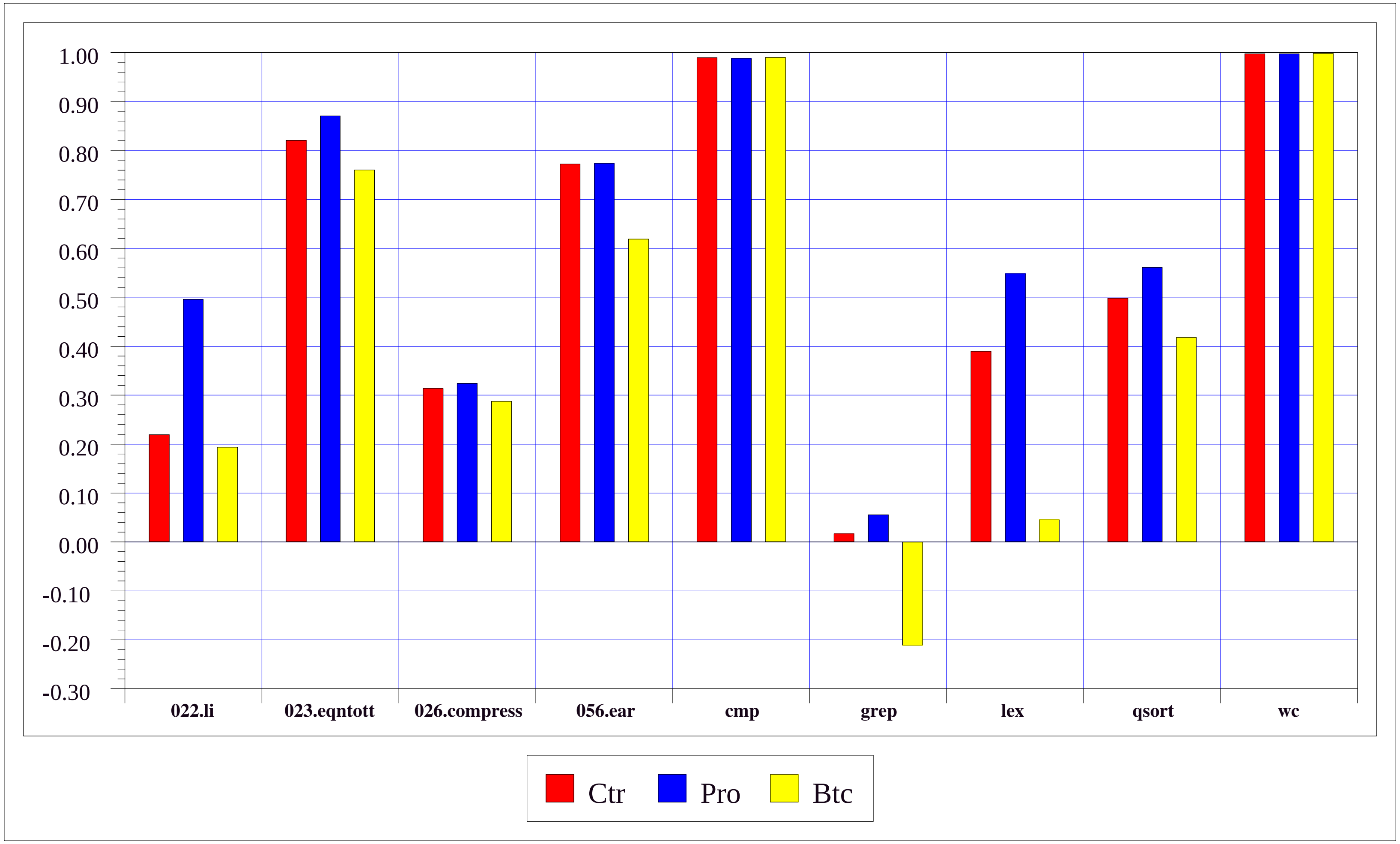
Processor Model

- In-order issue superscalar processor
- Uniform function units, except branches
- Extended version HP PA-RISC instruction set
- Instructions latencies of HP PA-RISC 7100
- 64 integer, 64 floating point, 64 predicate registers
- I/D caches: 64K, directed mapped, blocking, 64 byte blocks, 12 cycle miss penalty
- BTB: 1k entries, direct mapped, 2 bit counter, 2 cycle misprediction penalty

Reduction in Dynamic Branches



Reduction in Branch Mispredictions



Distance Between Branches

	Base		Pred	
	Branches	Mispred Branches	Branches	Mispred Branches
022.li	3.5	43	3.7	62
023.eqntott	2.1	28	4.8	195
026.compress	7.3	78	8.7	137
056.ear	4.7	129	5.3	492
cmp	2.6	433	3.6	43940
grep	0.9	126	1.2	100
lex	1.6	160	6.4	290
qsort	8.5	54	14.7	128
wc	2.0	46	6.8	28464
Average	3.7	122	6.1	8201

Branch Characteristics of lex

- 45% of dynamic instructions are branches
- Low branch misprediction rate (2%)
- Hyperblock formation concentrates on reducing branch frequency
- Dynamic branch breakdown

	Loop		Non-loop			
	Inner	Outer	Inner	Outer	StrLn	Total
Base	5917K	883K	252K	7000K	36K	14107K
Pred	6255K	956K	73K	1312K	29K	8648K

Branch Characteristics of lex (Cont.)

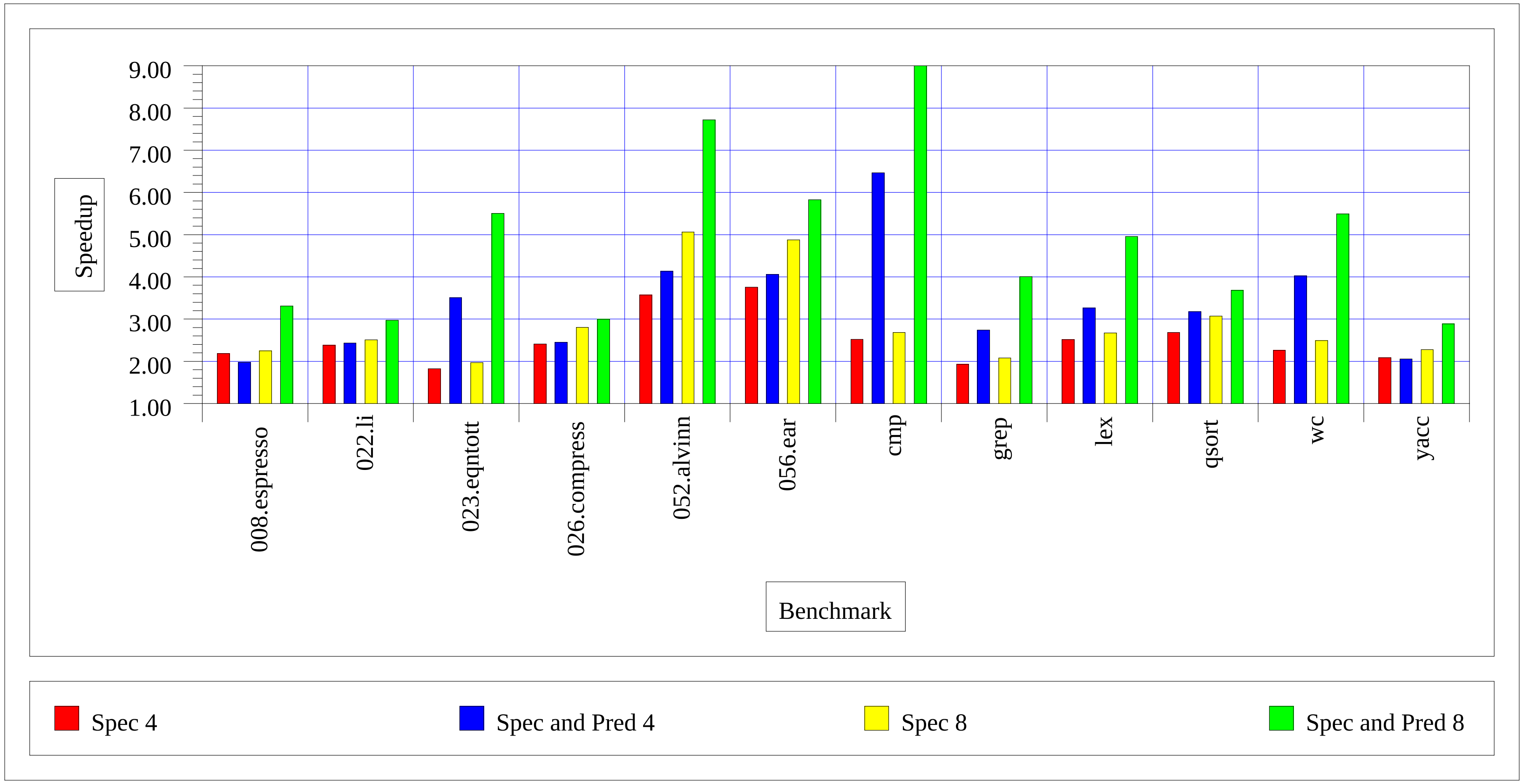
- Dynamic mispredictions

		Loop		Non-loop			
		Inner	Outer	Inner	Outer	StrLn	MPR
Base	Counter	39.9K	9.1K	7.9K	160K	4.9K	0.02
	Profile	93.0K	28.2K	10.5K	269K	5.2K	0.03
	Btc	76.6K	43.5K	10.2K	244K	9.4K	0.03
Pred	Counter	79.1K	14.1K	0.6K	35.7K	3.5K	0.02
	Profile	83.0K	17.1K	23.1K	51.7K	4.8K	0.02
	Btc	174K	93.4K	2.0K	85.5K	8.3K	0.04

- Misprediction coverage

	10	30	50	70	90	100
Base	0.00	0.00	0.00	0.01	0.06	0.29

Effect of Predicated Execution Support
(1 Branch per cycle, Perfect I/D cache)



Conclusions

- Predicated execution
  - Enables the compiler to remove branches using conditional instructions
  - Requires moderate amount of architectural support
- Hyperblock techniques
  - Eliminate branches $\Rightarrow$ reduce branch processing requirements
  - Remove unbiased branches $\Rightarrow$ reduce mispredictions
  - Exploiting ILP along multiple paths on control
- Predication and speculation are viable techniques for executing non-numeric programs on wide issue ILP processors