

Address-Branch Correlation: A Novel Locality for Long-Latency Hard-to-Predict Branches

Hongliang Gao, Yi Ma, Martin Dimitrov, Huiyang Zhou



School of Electrical Engineering and Computer Science
University of Central Florida



Why Long-Latency Hard-to-Predict Branches?

- Hard-to-Predict
- Long-Latency

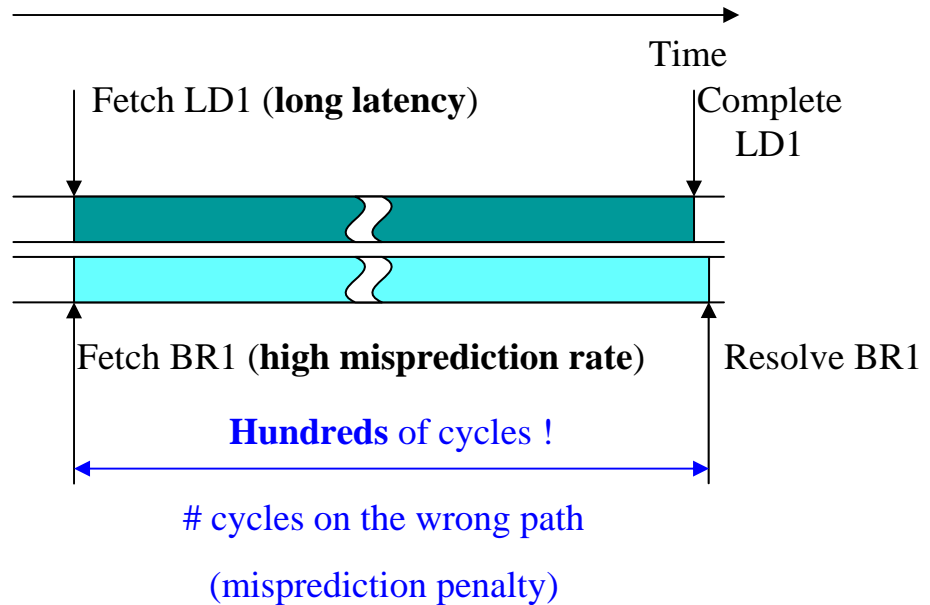
Cache miss, chained loads

Irregular patterns

LD1 \$r0, \$r2, [addr]

LD1 \$r1, [\$r2]

BR1 \$r1, Target



Critical performance bottleneck for processors with large instruction windows.



Overview

- Goal: Handle long-latency hard-to-predict branches.
- Address-Branch Correlation
 - A result of stable data structures or key components.
 - Using load addresses to predict branch outcomes.
- Benchmark study
 - Examples.
 - Performance potential.
- A design to exploit address-branch correlation.



Outline

- Motivation
- Overview
- Address-Branch Correlation
- Benchmark Study
- Address-Branch Correlation Based Predictor
- Evaluation
- Conclusion



Address-Branch Correlation

- Address → data addresses of load instructions
- Branch → dependent branch outcomes

A microbenchmark with random linked list insertion/deletion operations:

```

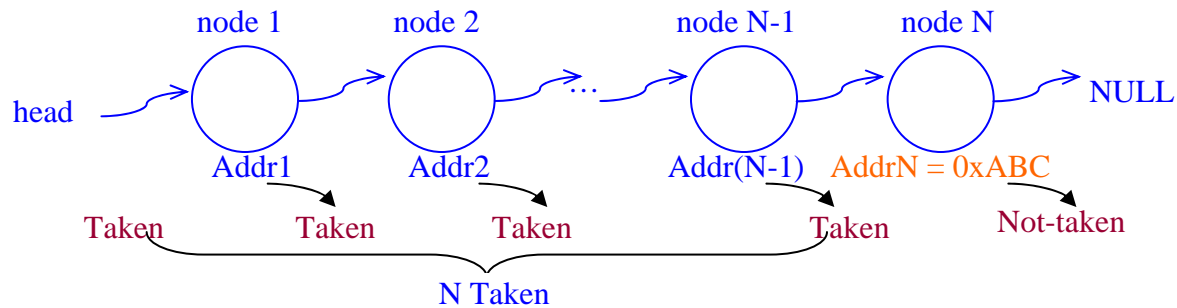
Initialization();
for (i = 0; i < 100000; i++) {
//Insert or delete a node at a random position except the end node
//Maximum length: 20
RandomOp();
node = head;
while( node ) {           //Branch1(while)
  node = node->next;      //Load 1 (node->next)
}
}

```

```

bne $r2, $r0, Target
...
lw $r2,0($r2)

```

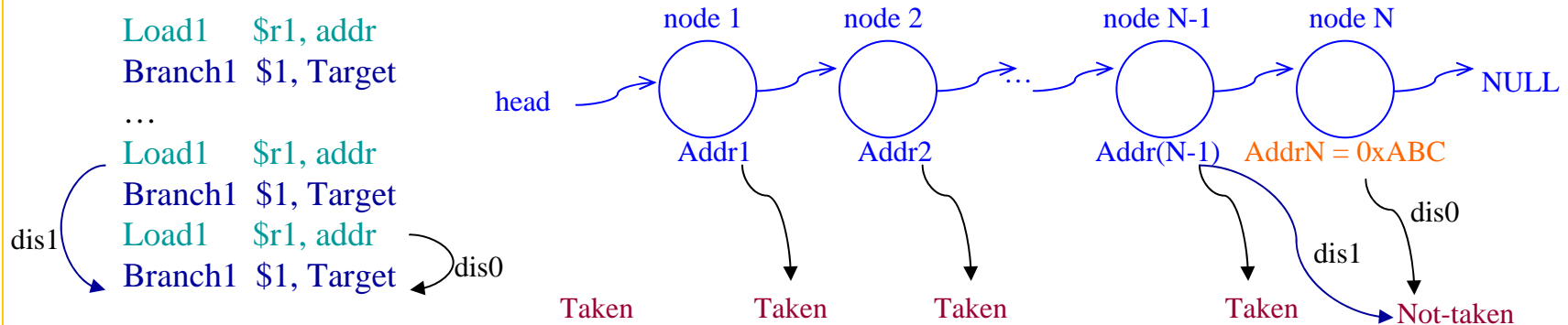


- Misprediction Rate of Branch1:16kB gshare: 9%;16kB TAGE: 6.5%



Address-Branch Correlation

- Correlation Distance



- When will they have stable correlation?
 - Key data component remains stable (e.g., the last node).
- How to exploit the correlation?
 - Use load address to predict branch outcome.
 - Use longer correlation distance.



Outline

- Motivation
- Overview
- Address-Branch Correlation
- Benchmark Study
- Address-Branch Correlation Based Predictor
- Evaluation
- Conclusion



Benchmark Study

- An example from *mcf*

```

long refresh_potential( network_t *net )
    ...
    while( node != root ) {
        while( node ) { //long-latency branch
            if( node->orientation == UP ) //long latency
branch
                node->potential = node->basic_arc->cost +
node->pred->potential;
            else { /* == DOWN */
                node->potential = node->pred->potential -
node->basic_arc->cost;
                checksum++;
            }
            tmp = node;
            node = node->child;
        }
    }
    ...
}

```

(a) C source code

```

...
00400808 lw $v0[2],28($a0[4]) // Load 1 “node-
>orientation”
00400810 bne $v0[2],$a3[7],00400848
                // Branch 1 “node->orientation == UP”
00400818 lw $v0[2],32($a0[4])
00400820 lw $v1[3],8($a0[4])
...
00400878 sw $v0[2],44($a0[4])
00400880 addu $v1[3],$zero[0],$a0[4]
00400888 lw $a0[4],12($a0[4]) // Load 2 “node =
node->child”
00400890 bne $a0[4],$zero[0],00400808 //Branch 2
“while( node )”

```

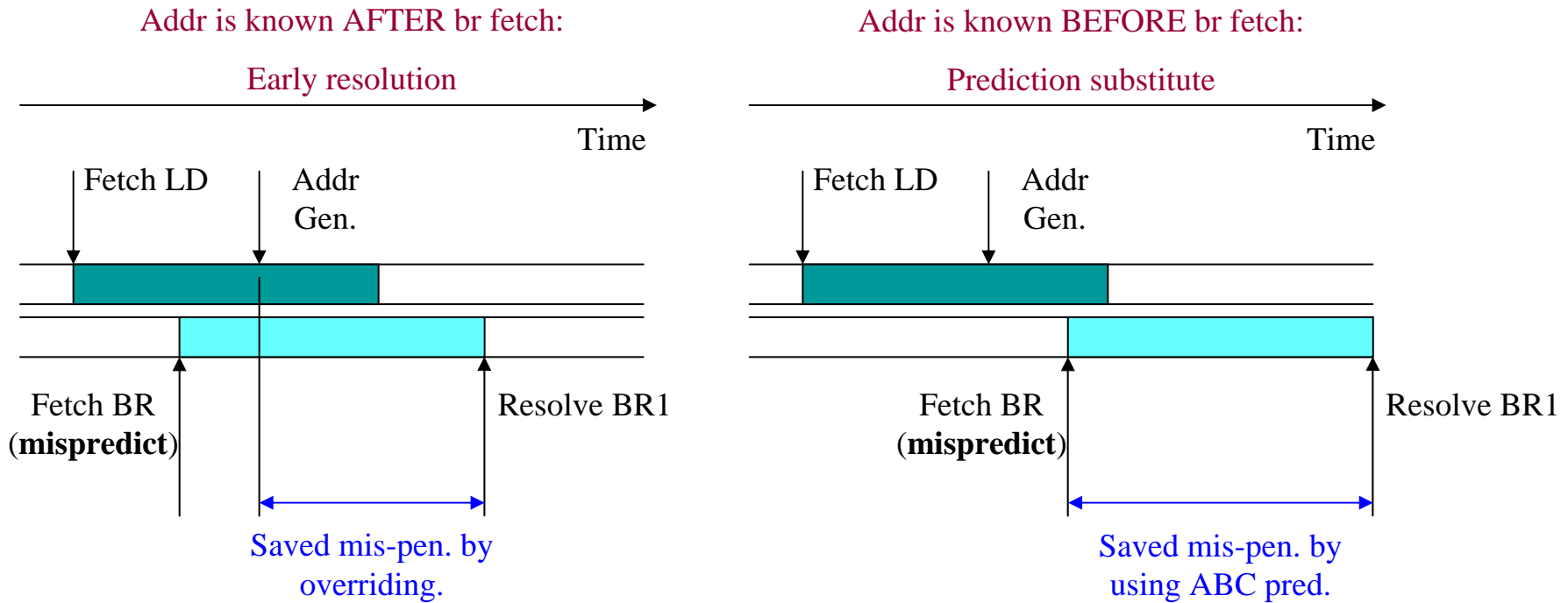
(b) Assembly code

	MR (16kB TAGE)	Avg. mis-pen.	% of total mis-pen.
Branch1	3.1%	542	9.5%
Branch2	17.4%	636	62.1%



Benchmark Study

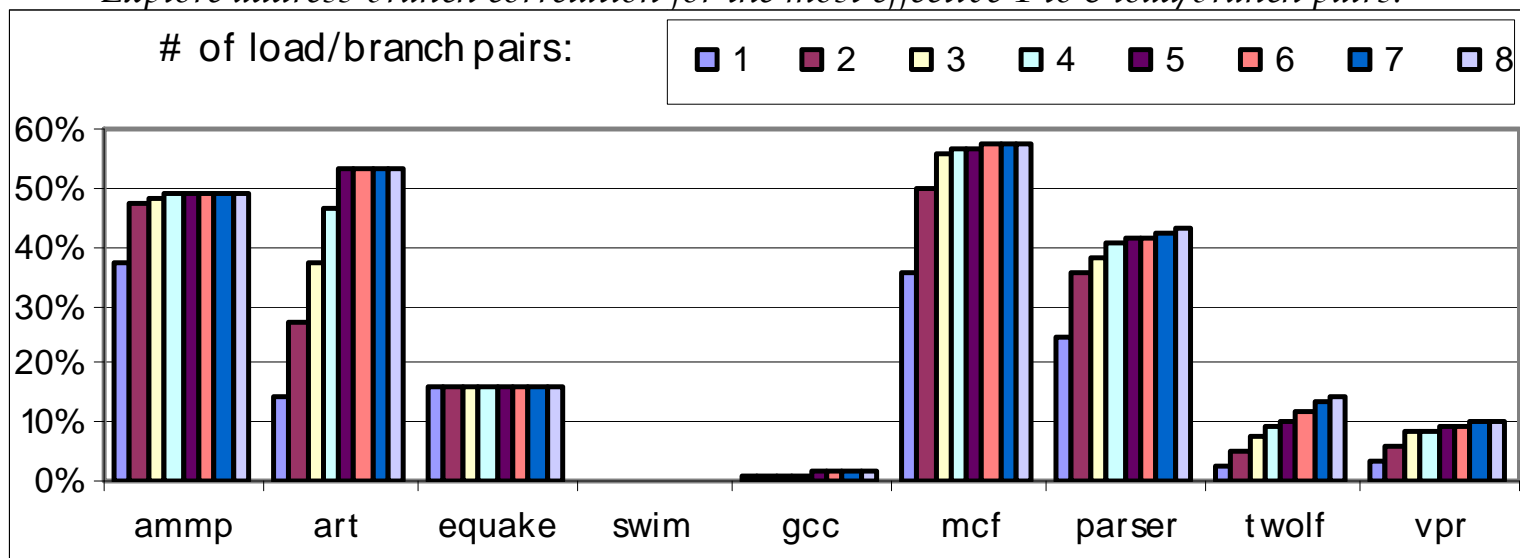
- Performance Potential
 - For each dependent load/branch pair, calculate reductions of misprediction penalty by exploring address-branch correlation for a correlation distance.





Benchmark Study

- Potential **misprediction penalty reductions** of memory-intensive benchmarks:
 - *Explore address-branch correlation for the most effective 1 to 8 load/branch pairs.*



- Observations:
 - A large portion of misprediction penalty could be saved for some benchmarks.
 - Only need to explore a small number of load/branch pairs.
- Correlation distance:
 - Optimal correlation distance is longer than 0.
 - Long correlation distance may hurt performance.



Outline

- Motivation
- Overview
- Address-Branch Correlation
- Benchmark Study
- Address-Branch Correlation Based Predictor
- Evaluation
- Conclusion

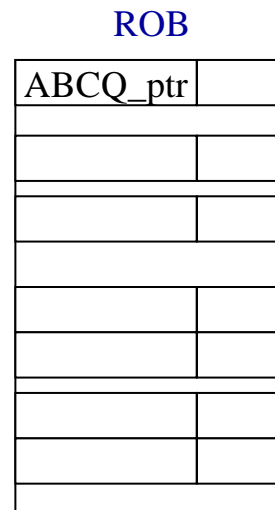
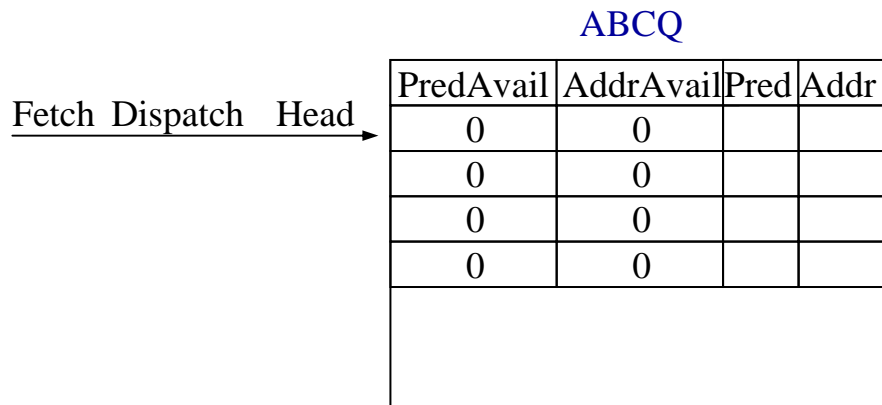


Address-Branch Correlation Based Predictor

- An example:

Address-Branch Correlation Prediction Table

Tag	Correlated	Pred
	1	
A	1	1
	1	





Address-Branch Correlation Based Predictor

- An example:

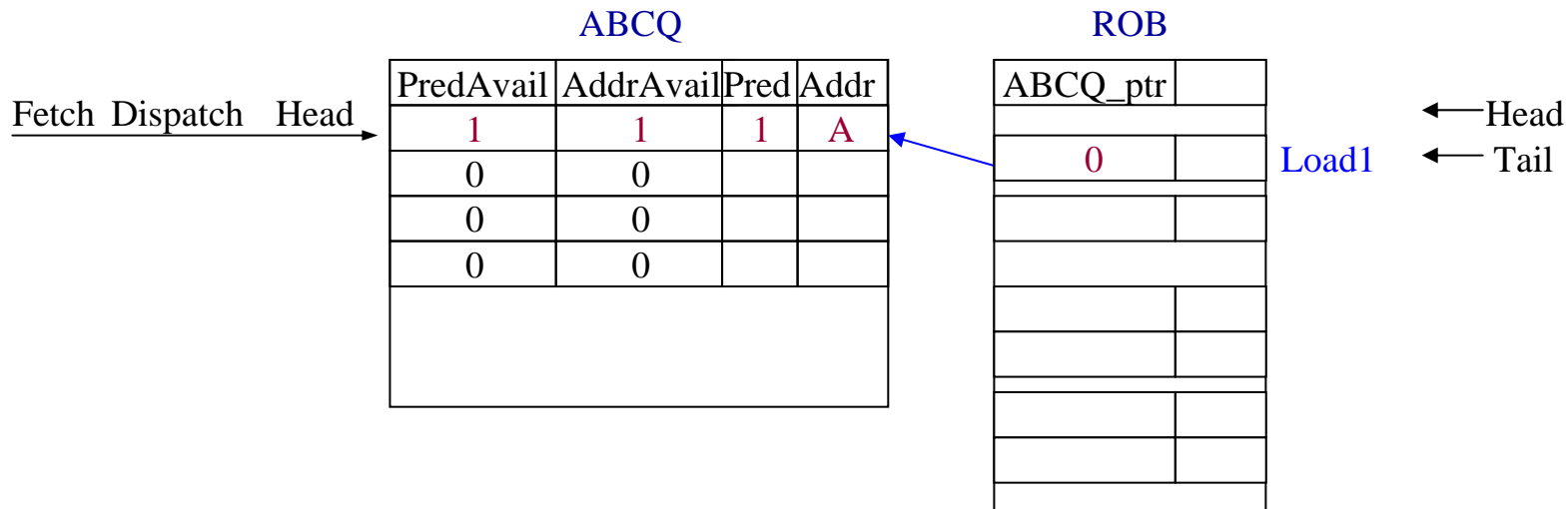
Load1 \$r1, 0xA

Address-Branch Correlation Prediction Table

Tag	Correlated	Pred
	1	
A	1	1
	1	

Dispatch Load1

Load1 Addr_GEN





Address-Branch Correlation Based Predictor

- An example:

Load1 \$r1, 0xA
 Branch1 \$r1, Target

Fetch Branch1

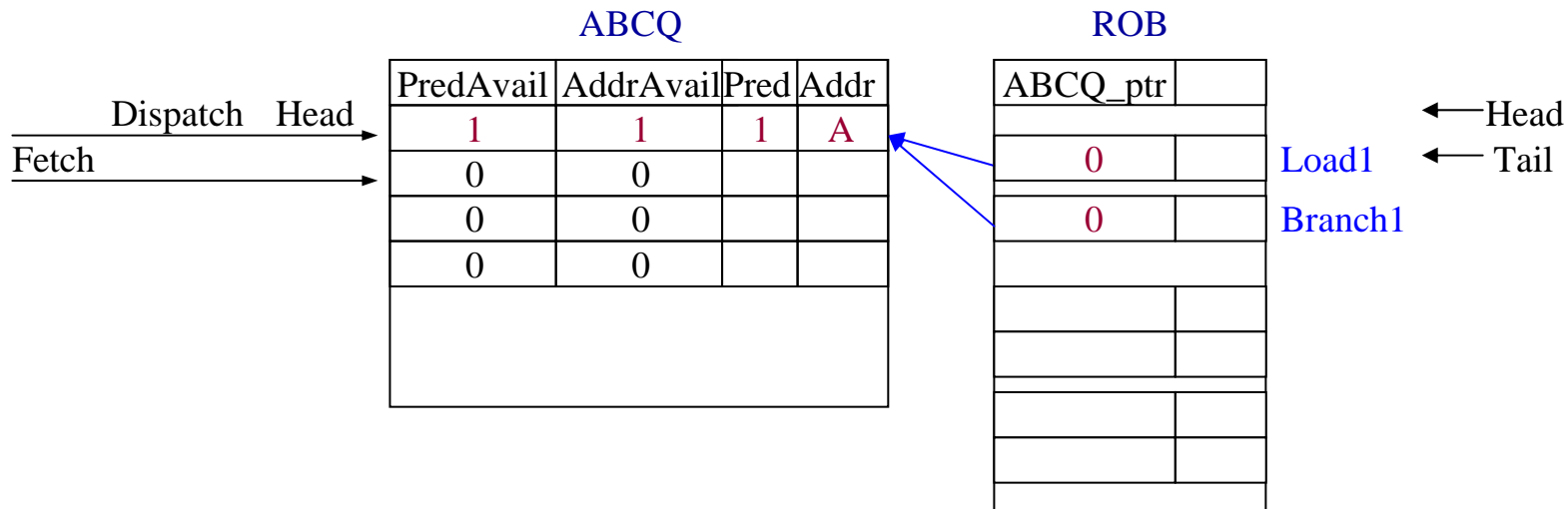
Dispatch Branch1

Retire Branch1

Address-Branch Correlation Prediction Table

Tag	Correlated	Pred
	1	
A	1	1
	1	

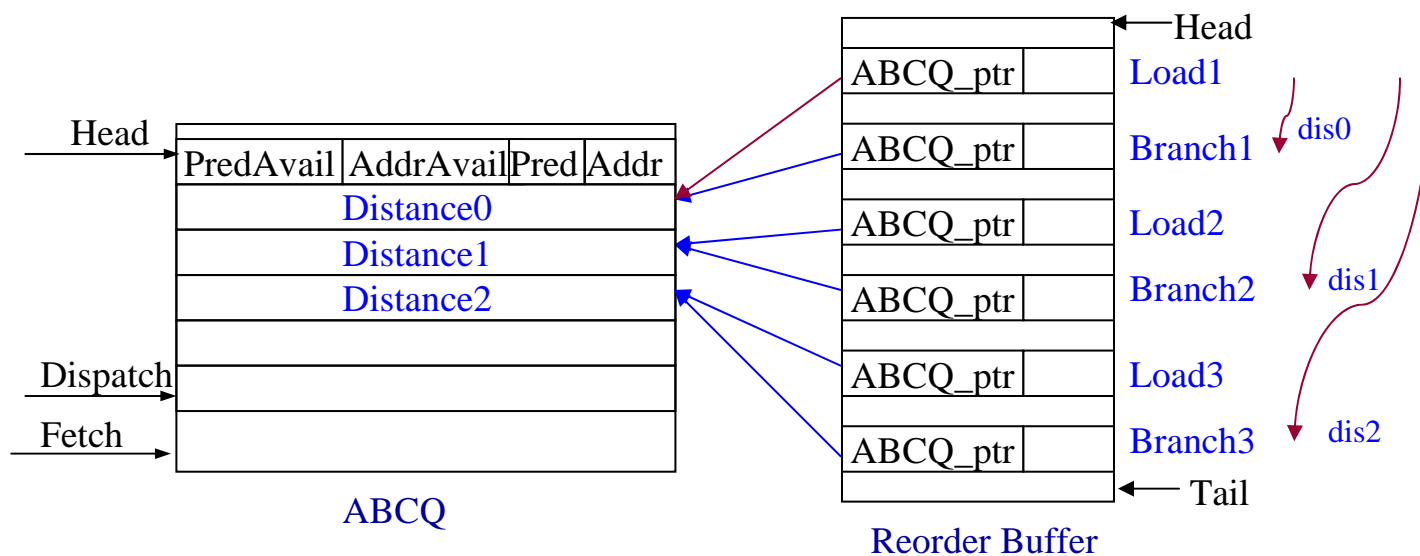
Correlated =
 1: if taken
 0: if not-taken





Address-Branch Correlation Based Predictor

- Correlation distance





Outline

- Motivation
- Overview
- Address-Branch Correlation
- Benchmark Study
- Address-Branch Correlation Based Predictor
- Evaluation
- Conclusion

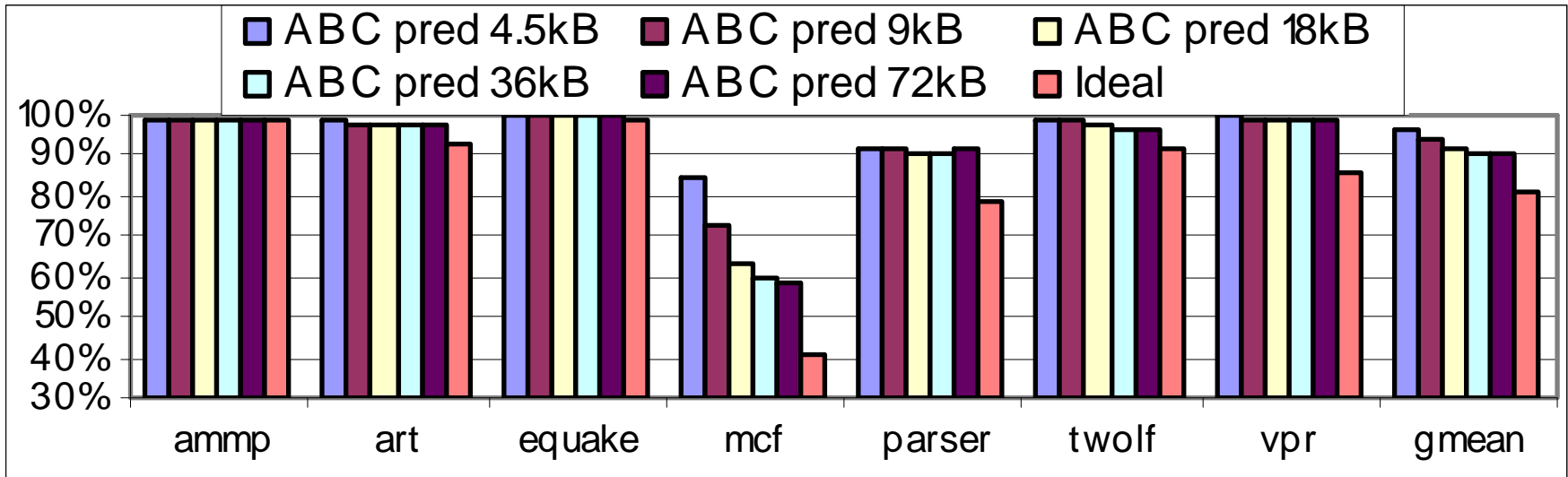


Experimental Methodology

- Memory intensive benchmarks (excluding *swim* and *gcc*)
- Processor Configuration:
 - I-Cache & D-Cache: 32kB, 2-way
 - L2 Cache: 1MB, 8-way
 - Mem latency: 300 cycles
 - Primary branch predictor: 16kB TAGE
 - Reorder buffer: 1024 entries
 - Dispatch/issue/retire bandwidth: 4-way
 - Prefetcher: Stride-based stream buffer

Performance Improvement

Execution Time (Normalized to 16kB TAGE without ABC):

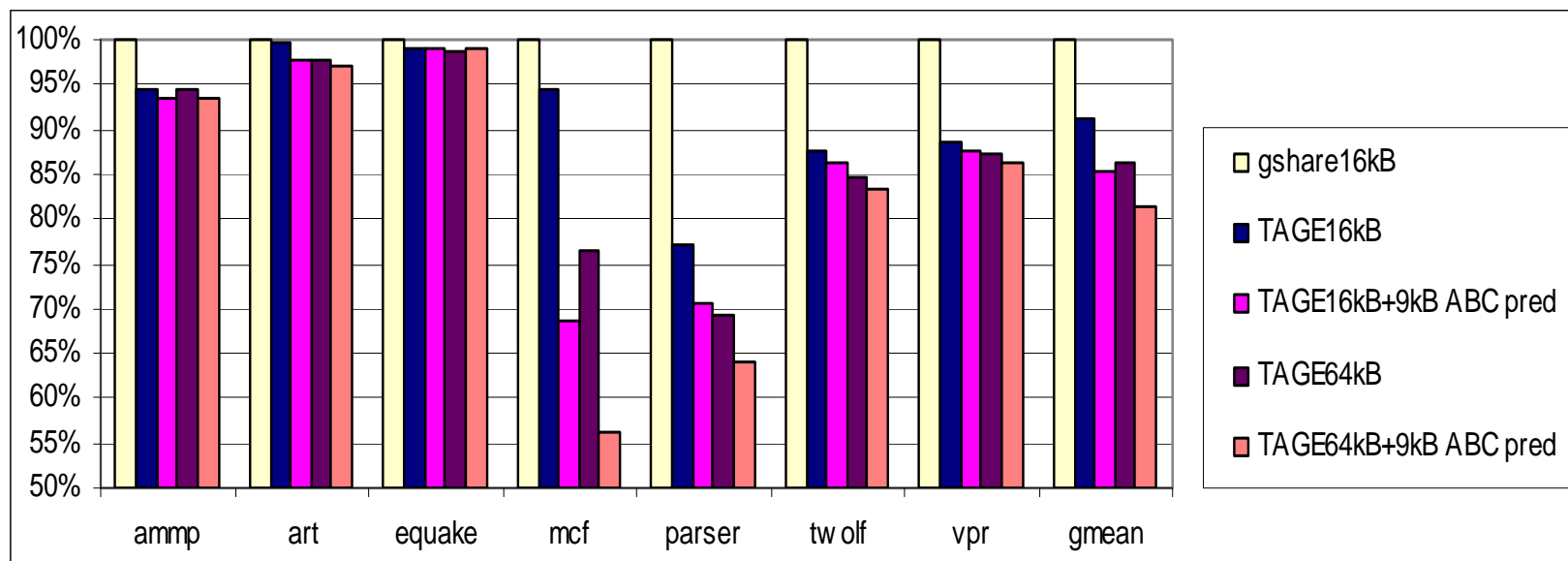


Augmenting a 16kB TAGE with a 9kB ABC predictor reduces the execution time by 6.3% (up to 27%).



Impact of Primary Branch Predictors

Execution Time (Normalized to gshare16kB):



A 16kB TAGE predictor with a 9kB ABC predictor outperforms a 64kB TAGE predictor



Misprediction Rates of Selected Branches

	ampp	art	equake	mcf	parser	twolf	vpr	amean
16kB TAGE	17.43%	9.66%	13.97%	10.64%	10.54%	20.09%	30.16%	16.07%
64kB TAGE	17.29%	5.28%	12.60%	6.37%	7.77%	18.46%	29.30%	13.87%
16kB TAGE + 9kB ABC	7.43%	4.45%	13.60%	4.37%	3.99%	16.07%	28.48%	11.20%



Conclusions

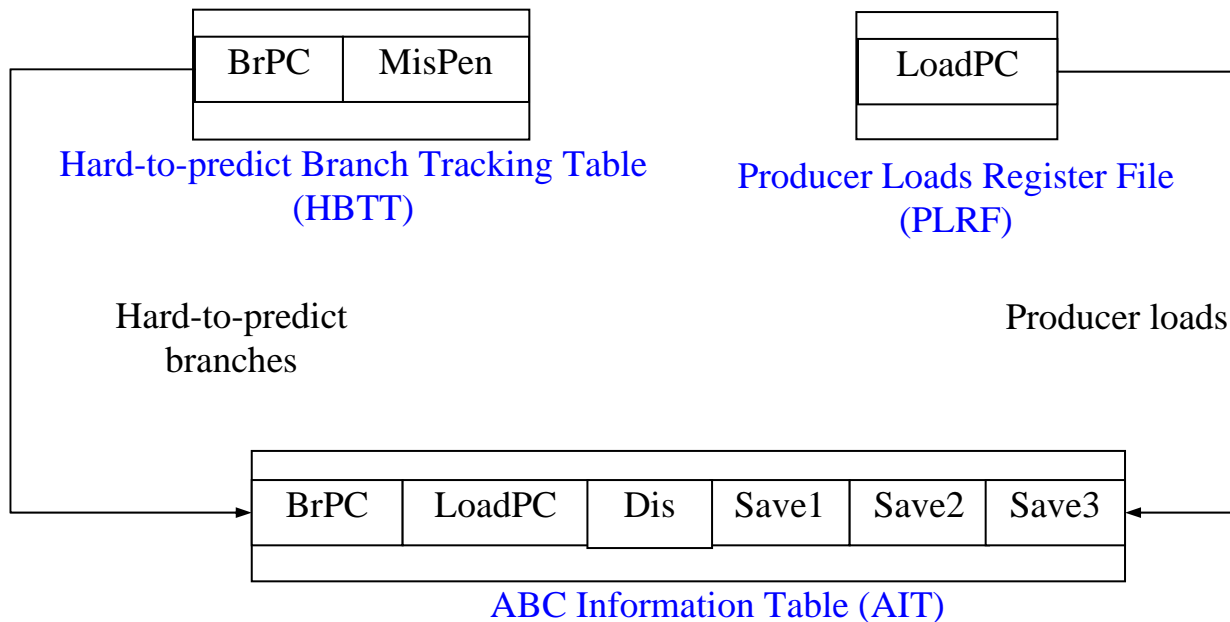
- Serious performance bottleneck:
 - Long-latency hard-to-predict branches.
- Address-Branch Correlation
 - A result of stable data structures or key components.
 - Using load addresses to predict branch outcomes.
 - Could be captured dynamically.
- Limitation
 - Only useful for applications with strong address-branch correlation.

Thank you and Questions?



School of Electrical Engineering and Computer Science
University of Central Florida

Backup - Capturing Load/Branch pairs





Backup - Training

- Length of the training period and the number of selected load/branch pairs:

	ampp	art	equake	mcf	parser	twolf	vpr
Training (M Insts.)	13	18	148	8	8	8	9
# of selected pairs	1	3	1	3	4	2	3

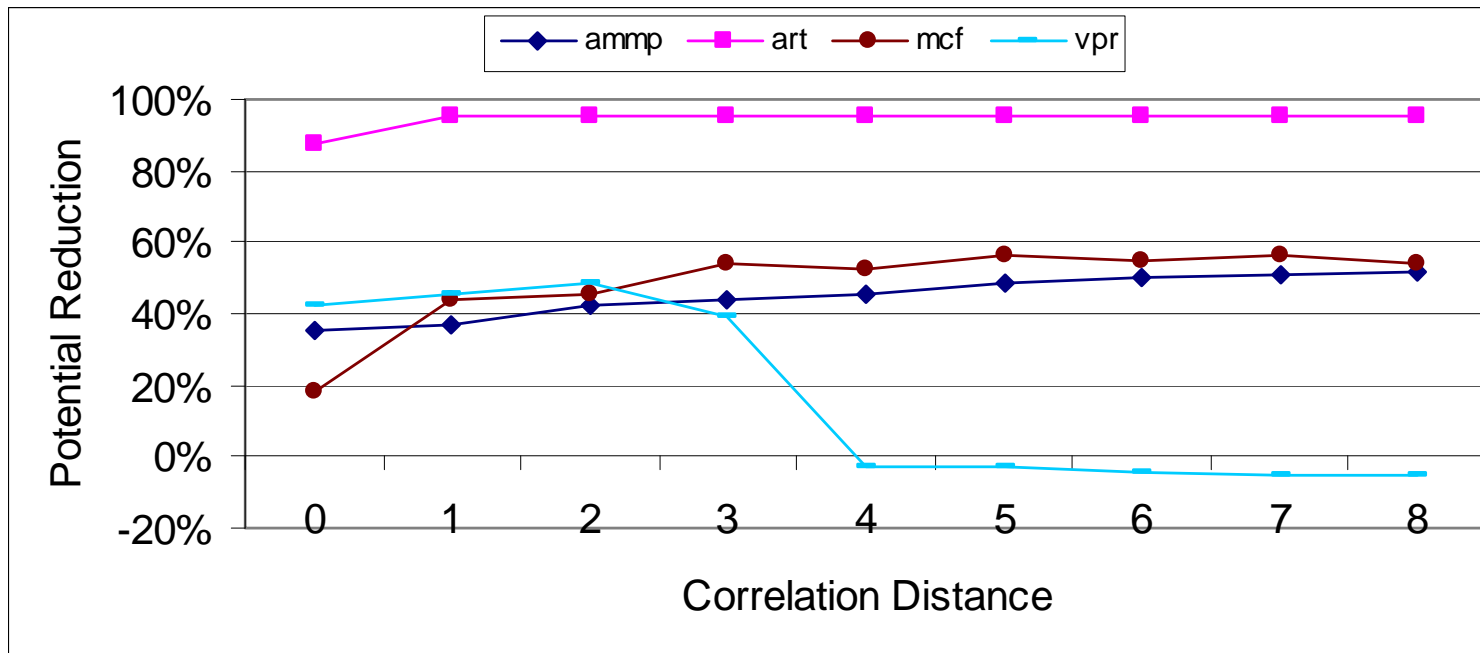


Backup – Hardware Cost

- HBTT:
 - 16 entries, each entry: a 32-bit PC and a 24-bit saturating counter.
- AIT:
 - 5 entries, each entry: 32-bit PCs, a 3-bit counter to store correlation distance and three 21-bit saturating counters.
- PLRF:
 - 64 entries, each entry: a 32-bit PC. The overall storage requirement of those three tables is 3594 bits (449 bytes).
- HBTT + AIT + PLRF: 449 bytes.
- ABCQ:
 - 64 entries, each entry: three 1-bit fields and an address field (12 bits).
 - 5 ABCQs, total: 880 bytes.

Backup – Correlation Distance

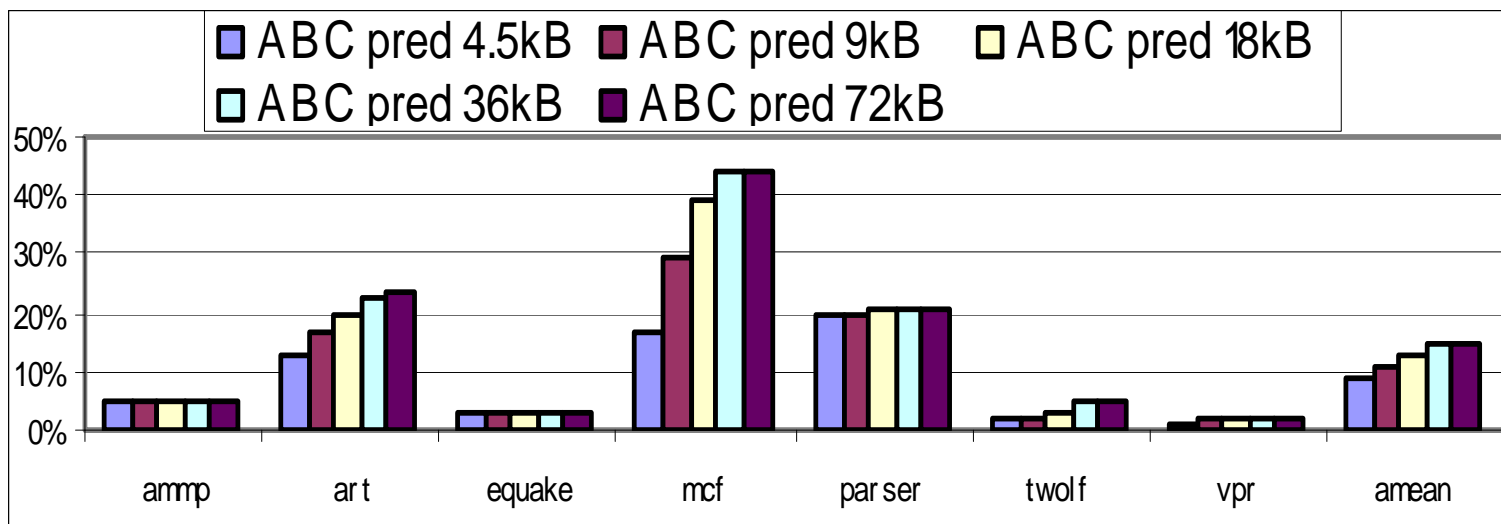
- Load/Branch pair with largest potential misprediction penalty reduction:





Backup – Reduction of Misprediction Penalty

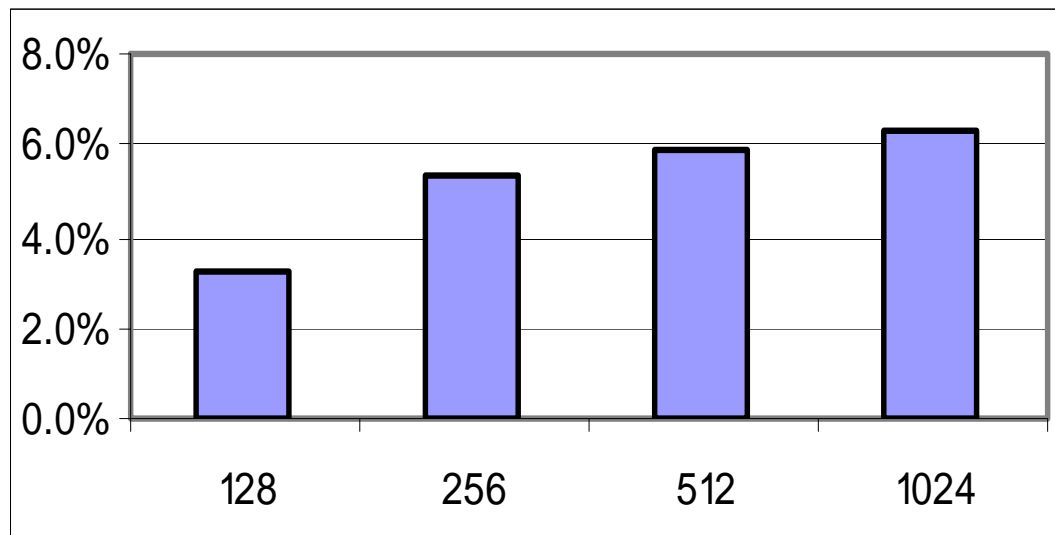
- The real reductions of misprediction penalty for selected branches:





Backup – Sensitivity Study (ROB)

- Performance improvements of 16kB TAGE + 9kB ABC:



ROB Size

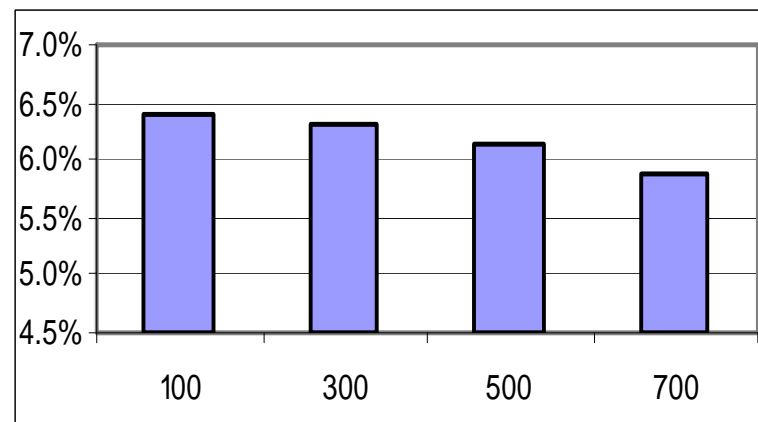


Backup – Sensitivity Study (Memory and Cache)

- Performance improvements of 16kB TAGE + 9kB ABC:



L2 Size

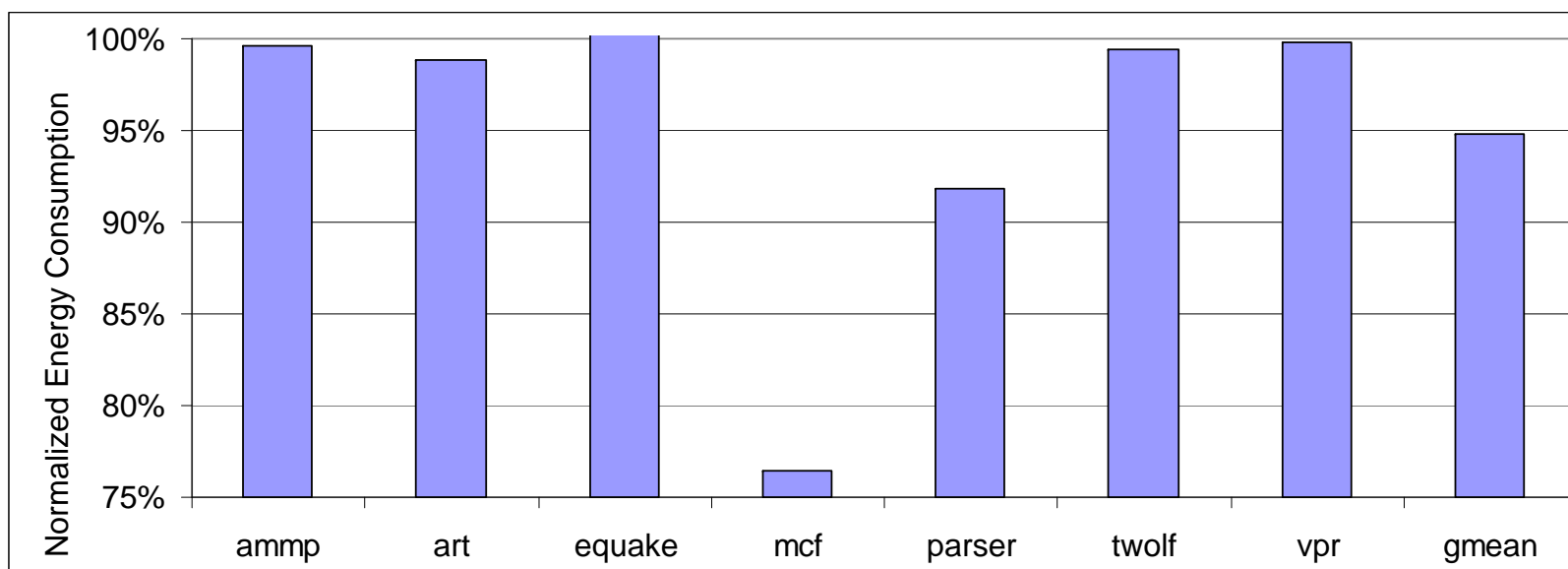


Memory Latency (cycles)



Backup – Energy

- Energy Consumption (16kB TAGE + 9kB ABC, normalized to 16kB TAGE):





Backup – GTL

- Execution Time (Normalized to GTL):

