ECSE 425 Lecture 13: More Branch Prediction

H&P Chapter 2

Administrative Notes

- Homework
  - Pick up Homework 1 (grades also on WebCT)
  - Homework 2 due today
  - Homework 3 out today, due October 17
- Midterm 1
  - 50 minutes, in class, October 12
  - Chapter 1, Appendix A, Chapter 2.1-2.3

Last Time

- Static prediction
  - Compiler profiling
- Dynamic prediction
  - 1-bit predictors

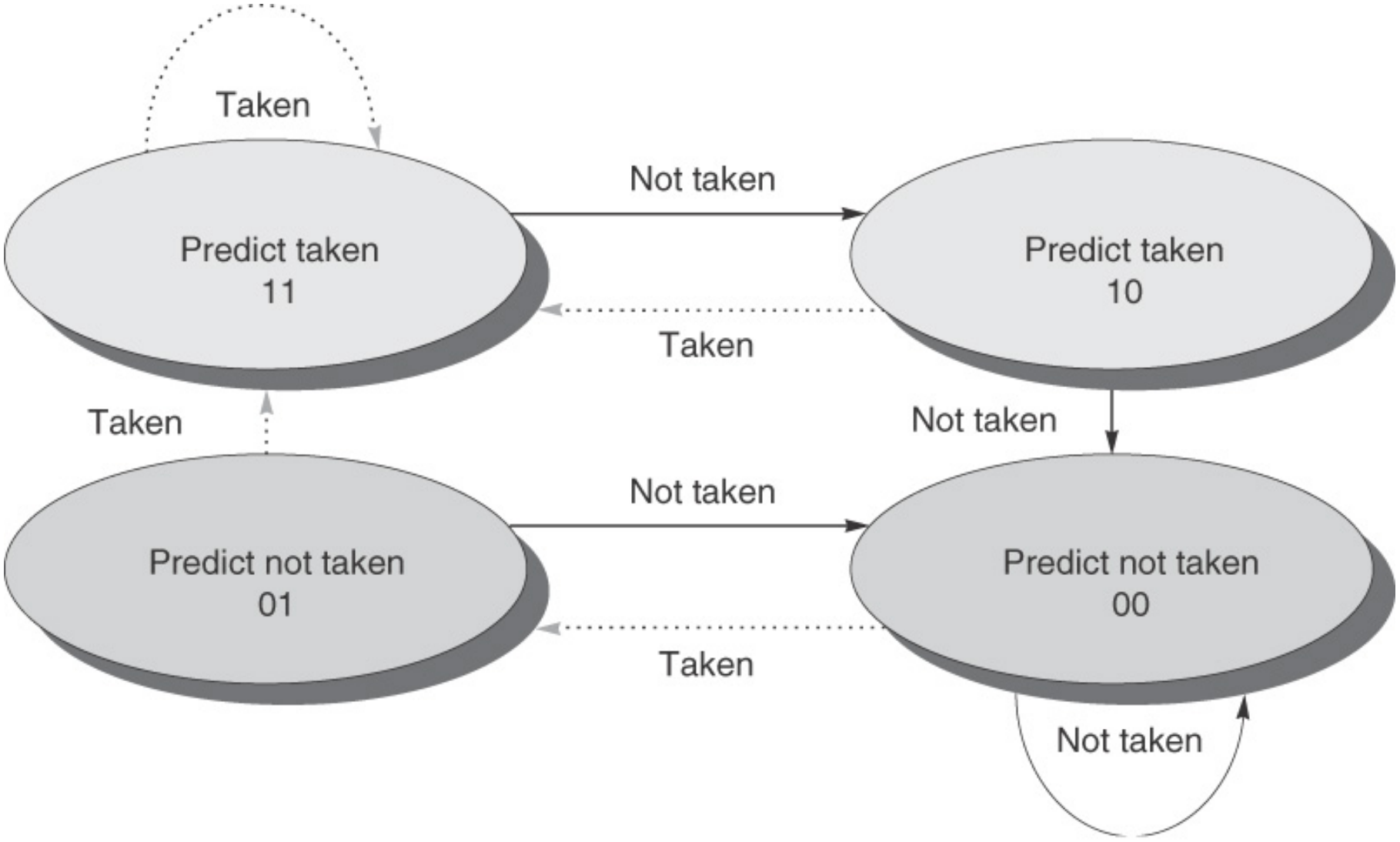
Today

- Chapter 2.3: Branch Prediction
- More dynamic prediction
  - 2-bit predictors
  - Correlating predictors
  - Tournament predictors

2-bit predictors

- Mispredict twice in a row to change prediction
  - Count the number of ‘taken’ (not taken) outcomes
- Branch taken (not taken) twice in a row
  - Predict “taken” (not taken)
- Branch not taken (taken) once
  - Continue to predict “taken” (not taken)
- *n* prediction bits
  - 2*n*-1 mispredictions before prediction changes

2-Bit Branch Prediction



Example

- Consider a for loop executed again and again

```
Actual outcome:          …TTTTNTTTTNTTTTNTTTTN…
Predicted with 1-bit:    …NTTTTNTTTTNTTTTNTTTT…
Predicted with 2-bit:    …TTTTTTTTTTTTTTTTTTTT…
```

- 1-bit predictor: 60% accuracy
- 2-bit predictor: 80% accuracy

Iteration	Predictor Bits	Predicted Outcome	Actual Outcome	Update
1	10	T	T	11
2	11	T	T	11
3	11	T	T	11
4	11	T	T	11
5	11	T	N	**10**

Accuracy of 2-bit predictors

- 99-100% on heavy matrix code
- 90-95% on floating point code
- 80-90% on integer code (e.g. gcc)
- Branch predictor state: up to 4K entries
  - Statistics show no gain in accuracy beyond

Correlated Branches

- Why is the performance of integer code so low?
- We try to predict the behavior of branches in isolation
  - We assumed that branch behavior is not correlated
- However, branches are often related:

```
If (a == 2)                a->R1; b->R2; 0->R0
    a = 0                  DSUBUI   R3,R1,#2
If (b == 2)                BNEZ     R3,L1     ;branch one
    b = 0                  DADD     R1,R0,R0
If (a != b)        L1:     DSUBUI   R3,R2,#2
    …                      BNEZ     R3,L2     ;branch two
                           DADD     R2,R0,R0
                   L2:     DSUBU    R3,R1,R2
                           BEQZ     R3,L3     ;branch three
```

- "Local" predictors can't capture this behavior

Correlating Branch Predictors

- *Idea*: taken/not taken of recently executed branches is related to behavior of next branch (as well as the history of that branch behavior)
- *Simple predictor*
  - For each branch, maintain history of previous branch
  - 1-bit predictor for each possibility
    - If last branch was taken, take or don't take?
    - If last branch was not taken, take or don't take?

Example without Correlation

Simple Example

```
B1:    If (d == 0)
          d = 1
B2:    If (d == 1)
          …

       d->R1, 0->R0
       BNEZ      R1,L1
       DADDIU    R1,R0,#1
L1:    DADDIU    R3,R1,#-1
       BNEZ      R3,L2
       …
L2:
```

Performance of 1-bit Predictor

	Branch	Pred	Outcome	Update
d=2	B1	N	N	N
	B2	N	N	N
d=0	B1	N	T	T
	B2	N	T	T
d=2	B1	T	N	N
	B2	T	N	N
d=0	B1	N	T	T
	B2	N	T	T

Simple Correlating Predictor

Correlating Predictor States

Last branch *not taken*	Last branch *taken*
N	N
N	T
T	N
T	T

Predictor correlates past global behavior with future local behavior!

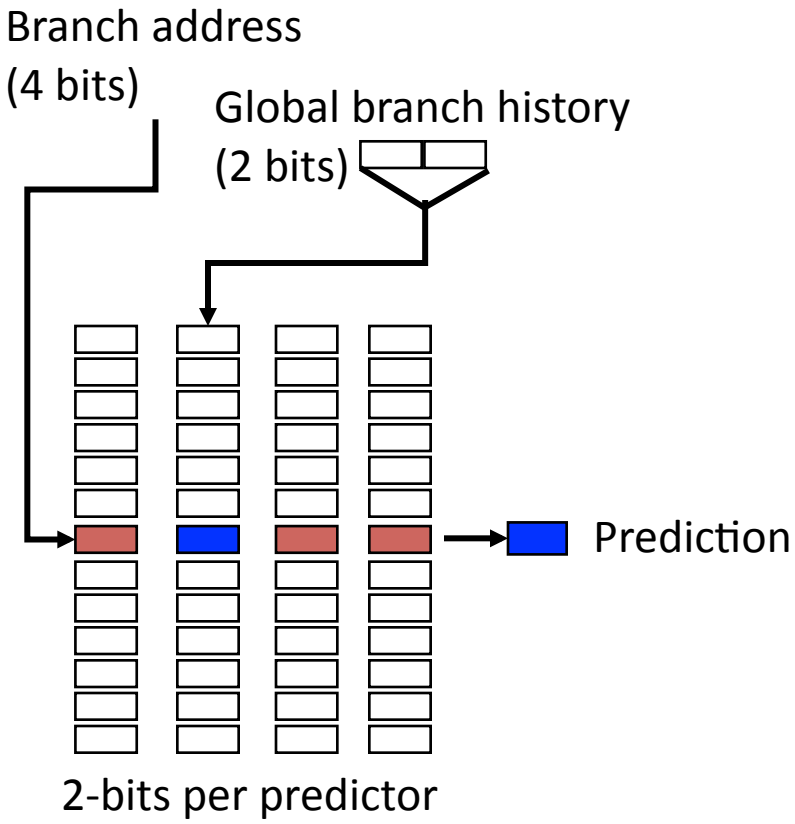
Correlating Predictor Perf.

	Branch	Pred Bits	Pred	Outcome	Update
d=2	B1	*N* N	N	N	N N
	B2	*N* N	N	N	N N
d=0	B1	*N* N	**N**	**T**	**T N**
	B2	N *N*	**N**	**T**	**N T**
d=2	B1	T *N*	N	N	T N
	B2	*N* T	N	N	N T
d=0	B1	*T* N	T	T	T N
	B2	N *T*	T	T	N T

```
B1: If (d == 0)
       d = 1
B2: If (d == 1)
       …
```

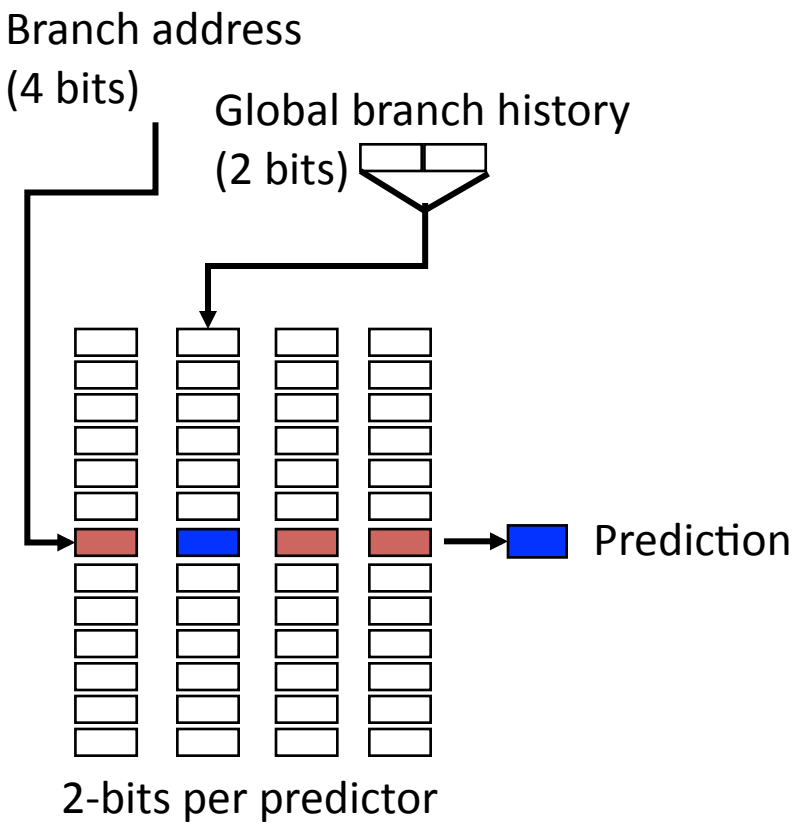
General Correlating Predictor

- LS branch address bits index the row
- Global branch history indexes the column
- (*m*, *n*) predictor
  - Tracks last *m* branches
  - Indexes 2^m predictors
  - Each predictor uses *n* bits
- (2,2) predictor
  - 2-bits of global history
  - 2-bits of local history
- (0,2) predictor
  - 2-bit saturating counter we've seen previously

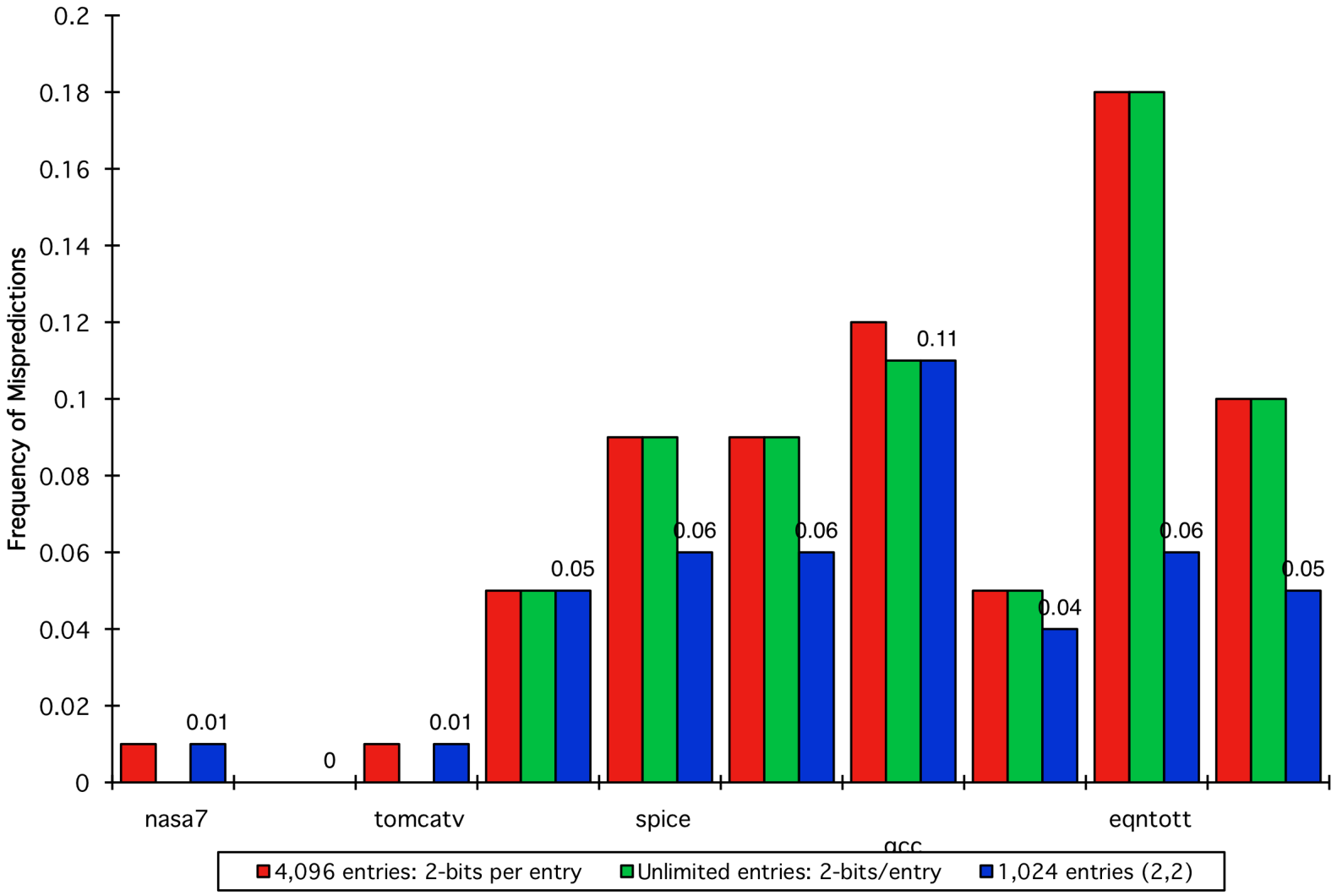


Correlating Predictor Example

- How many bits are in a (0,2) branch predictor with 4K entries?
- How many entries are in a (2,2) predictor with the same number of bits?



Branch History Table Prediction Accuracy

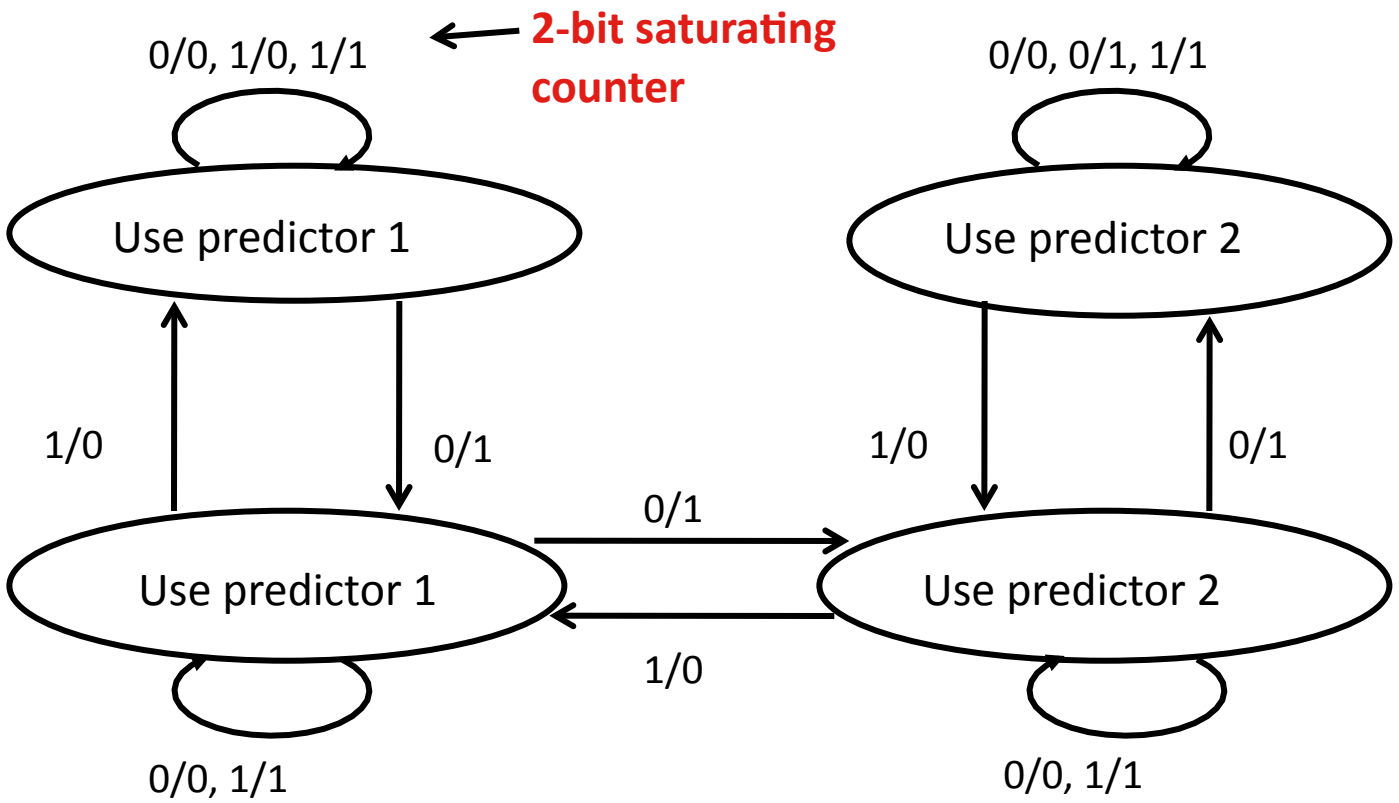


Tournament Prediction: Best of Both

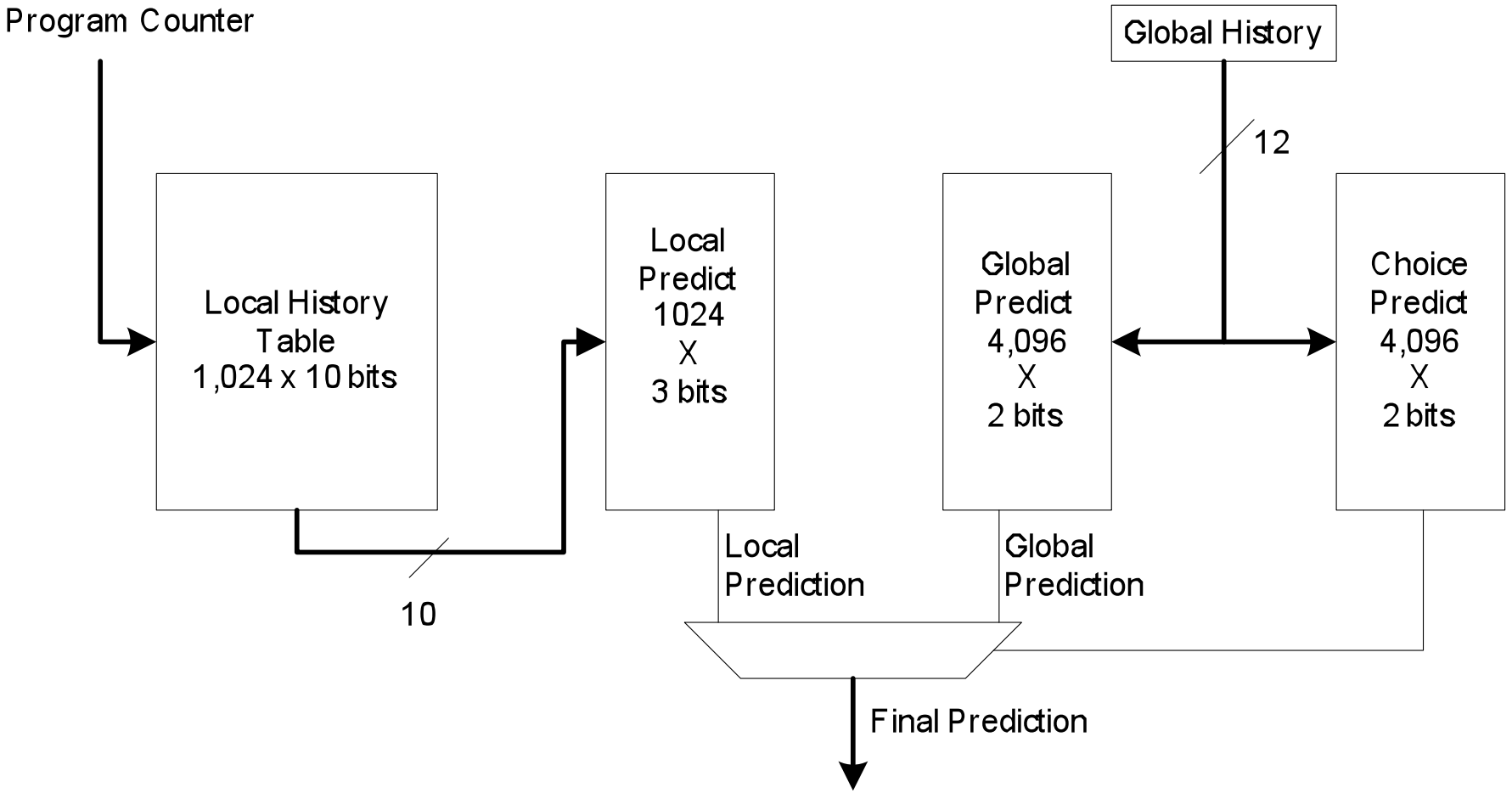
- 1-bit predictor failed to capture simple loop behavior
  - Increase local state to account for frequently taken branches
- 2-bit predictor failed to capture all branch behavior
  - Add global state to improve performance
- Correlating predictors
  - Prefer global history to local history … why not leverage both?
- Tournament predictors use two predictors
  - One based on global information
  - One based on local information
  - A selector dynamically chooses between the two
- Pentium4 and Power5 – 30Kb tournament predictors

Tournament Predictor

- Dynamically combine local and global predictors
  - Use 2-bit saturating counter for selector
  - Must miss twice before changing the predictor
- Different branches may prefer global, local, or a mix



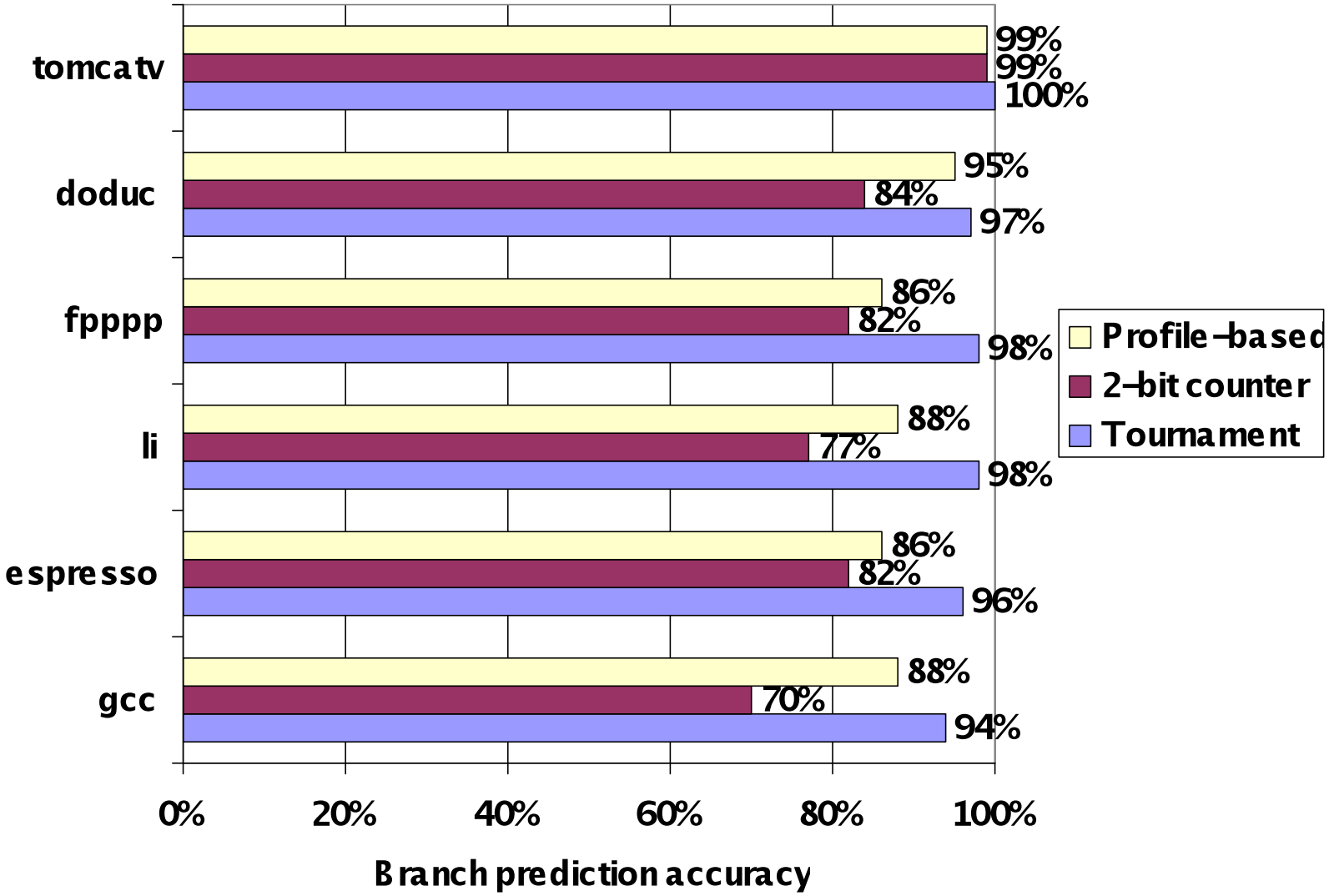
Alpha 2164 Tournament Predictor



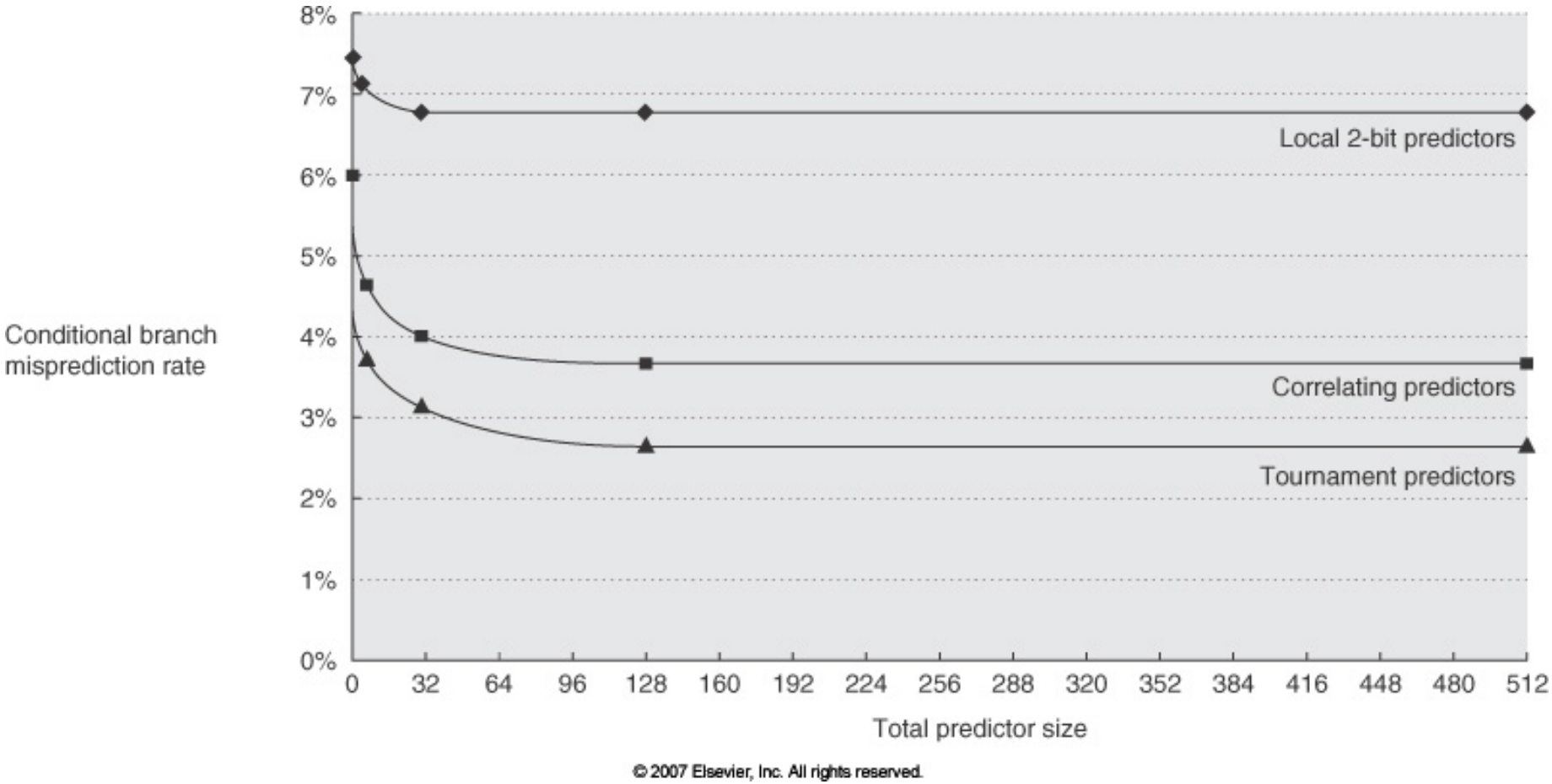
Alpha 21264 Tournament Predictor

- 4K Entry Choice Predictor
  - 2-bit saturating counters select between global and local predictors
- 4K Entry Global Predictor
  - Indexed by the 12-bit branch history
  - Each entry in the global predictor is a standard 2-bit predictor
- 2-level Local Predictor
  - 1K Entry Local History Table
    - 10-bits each, corresponds to the most recent 10 outcomes for the entry
  - 1K Entry Local Prediction Table
    - 3-bit saturating counters
    - Indexed by local history pattern
- Total size: 4K*2 + 4K*2 + 1K*10 + 1K*3 = 29K bits!
- Highly accurate branch prediction

Tournament Predictor Accuracy



Predictor Accuracy vs. Size (SPEC89)



Summary

- 2-bit predictor
  - Additional state improves loop branch accuracy
- Correlating branch predictor
  - Recent branches correlate with next branch
  - Different branches, or different encounters with one
- Tournament predictor
  - Dedicate more resources
  - Different predictors compete
  - Dynamically pick the most accurate one

Next Time

- Dynamic Scheduling
  - Chapter 2.4