Prediction

Prediction:
Determination of a possible future execution state.

Types of prediction.

- Branch Prediction

  Determination of branch outcome.

  Used for speculative instruction fetch and execution.

  In common use.

- Address Prediction

  Determination of load and store addresses.

  Used for hardware prefetching.

  Currently begin researched.

  Not covered in this set.

- Dependency Prediction

  Determine dependencies between load and stores.

  Used for speculative load bypassing (loading recently stored data before it reaches the cache).

  Early stage of research.

  Not covered in this set.

- Value Prediction

  Determine register values and results of loads and computation.

  Used in *trace caching*.

  Early stage of research.

Outline

Topics in this set (so far):

- One- and Two-level branch prediction.
- Avoiding aliasing (collisions) in branch predictor tables.
- Confidence Estimation (Outline only).
- Data Prediction (Outline only).

References at end.

Branch Prediction

Goal: Determine outcome of a branch knowing only address of instruction.

Reason: Keep processor busy.

Status: In routine use, well researched but possible room for improvement.

Most predictors are one of two types:

- One-level.
  Single branch history table (BHT).

- Two-level.
  Use outcome of recent branches to help index BHT.

One-Level Predictor

Covered in EE 4720 (and H & P).

Other names: *bimodal*, *2-bit counter*.

Rationale: determine if branches mostly taken or mostly not taken.

Uses direct mapped tag-less (usually) branch history table.

Maintain saturating counter for each branch.

Increment branch's counter if taken, decrement if not taken.

Predict taken if counter above a threshold.

Two-Level Predictors

Family of predictors described in depth by Yeh & Patt [Yeh 93].

Uses saturating counters similar to those in one-level predictors.

Saturating counter to use determined by branch instruction *and* outcome history (pattern).

Predictor consists of two tables.

Level One Table

Holds branch outcome patterns. (May consist of a single entry, sometimes called a shift register.)

Level Two Table

Holds saturating counters.

Two-Level Predictors, Continued

Three ways (more or less) of selecting table entry at each level.

Level-one selections:

G: Global: single pattern recorded for all branches.

S: Set: *set* based on branch; each set has own pattern.

P: Per-Branch: each branch has own pattern.

Level-two:

g: Global: only pattern used to index table.

s: Set: *set* based on branch; set and pattern index table.

p: Per-Branch: branch address and pattern index table.

Notation

$X\mathrm{A}y$, where $X \in \{$ G,S,P $\}$ and $y \in \{$ g,s,p $\}$.

Examples: GAg, PAg.

Two-Level Predictors, Continued

Sets can be defined many ways.

- Branch opcode (condition).
  May not work well.
- Region of memory of branch instruction.
  Does not work well.
- Low-order bits of branch address.
  Works well and a realistic way of implementing per-branch indexing.

Two-Level Predictor Performance Studies

Performance Studies

- Yeh & Patt [Yeh 93].
  Tested many two-level variations.
  High-cost configurations tested, would not apply to most systems.
- McFarling 93
  Tested several of Yeh & Patt's two-level predictors and others.
  Simulated practical tag-less direct mapped tables.

Performance Study: Yeh & Patt [Yeh 93].

Tested many two-level variations.

Simulated execution of SPEC89 benchmarks.

Set definition used: instruction address bits 10 and 11. (Correlate neighbors)

Branch tables: 1024 entry, 4-way set associative, LRU replacement, full tags. Reasonable when lots of memory is available, otherwise tag-less direct map configuration (used by McFarling) better.

On miss in per-address schemes predict forward not-taken, backward taken. (Can't do that with tag-less tables.)

Results

Note: PAp and GAp are called PAs and GAs in [Yeh 93], but use low-order branch bits, not bits 10 and 11.

Small table size (8k bits) prediction accuracy: PAp > GAp > SAs.

Large size (128k bits) prediction accuracy: GAp > PAp $\approx$ SAs

Performance Study: McFarling

Comparisons: [McFarling 93]

Ran SPEC89 benchmarks.
Tag-less direct mapped tables.

Results

Compared: GAp (gselect), GAg (global), PAg (local), one-level (bimodal).

(Yeh & Patt's terminology used, McFarling's terminology in parenthesis.)

GAp superior at small sizes and close second place at large sizes.

One-level almost as good at small sizes, not as good at larger sizes.

PAg performs second worst at small sizes, best at large sizes.

GAg worst at small sizes, well at larger sizes (but not as close as GAp and PAg)

Branch Behavior Effects on Prediction Performance

Here a *branch* is a particular branch instruction in a program.

The behavior of a branch determines ...
... how easily it can be predicted and ...
... how it interferes (aliases) with other branches.

One behavior: fraction of times branch taken [Chang 94].

Some branches mostly taken, some mostly not taken, some mixed direction.

Comparison: Chang 94

Simulated two-level predictors ...
... using different outcome sizes (last 2 branches, last 4 branches, etc.) ...
... separately determining prediction accuracy for different branch behaviors.

In results below, note that outcome size is varied while holding cost constant.

Branch Behavior Effects on Prediction Performance, Continued

Results

Here, PAs and GAs use low-order branch address bits to index counter table, same as PAp and GAp as used above.

PAs effective at short to moderate outcome sizes on mostly not-taken (NT) branches.

PAs at short sizes less effective than GAs and gshare (described below).

GAs effective on mostly NT for short outcomes, but degrades fast.

Total size has large effect on performance.

gshare effective on mostly NT for short outcomes, but degrades and recovers.

For mixed-direction branches:

PAs most effective at short sizes, less effective at larger ones.

GAs and gshare less effective as short, more at larger sizes.

Conclusions

New predictor might select prediction method based on branch behavior.

Such predictors covered below.

Further Branch Predictor Performance Improvement

Better predictions.

Use different information (*e.g.*, call stack) or use outcomes and branch address different.

Nothing particularly effective discovered ... so far.

Reduce effects of aliasing.

Maybe less interesting than better predictions ...
... but important since aliasing has strong negative effect.

Reducing Aliasing

Approaches to Reducing Aliasing

Direct Solutions: Avoid collision, but still like GAp.

Set-associative BHT.

Modify how BHT addressed. For example, xor'ing in gshare.

Store counter in multiple tables. Retrieve all and vote.

Likely Branch Direction and other Solutions

Alternate scheme for branches that collide (mostly one way).

Agree: Detect collision, determine if other branch same or opposite.

Two tables, one for more taken, one for more not taken.

Dynamically adjust outcome size.

Alternate BHT Addressing

Goal: Spread out counters to avoid collisions.

Method: gshare. [McFarling 93]

Exclusive-or branch address and outcome history. (Covered above.)

Inexpensive and effective.

Method: set-associative BHT. (Used in [Yeh 93].)

Size of tag much greater than counter so not cost effective.

Method: Store counter in multiple tables. [Michaud 97 ISCA]

Counter written to several tables.

Each table indexed differently. (*E.g.*, outcome bits permuted before $\oplus$ ing.)

Retrieve all three and vote.

Rationale: mostly at most one counter aliases; voting discounts it.

Likely Branch Direction Solutions

Rationale for all: BHT entries for mostly one-way branches interfere with others.

Method: Agree [Sprangle 97]

Based on gshare.

Rationale: turn negative interference (aliasing) to positive.

Bit stored for each branch in special table, branch target buffer, or instruction cache.

Bit indicates how predictor interpreted: ...
... branch instruction "agrees" with counter (counter usually predicts branch) ...
... or branch instruction "disagrees" (counter usually predicts direction branch *won't* go).

Tricky part is setting bit ...
... initializing it to first branch outcome works well.

Results: Moderate improvement.

Likely Branch Direction Solutions

Method: Bi-mode. [Lee 97].

(Not same as McFarling's bimodal predictor).

Based on gshare.

Rationale: to minimize destructive aliasing place more-taken and more-not-taken branches in separate tables.

Use three tables:

Choice predictor table, indexed by branch address, used to determine if branch more taken or not taken.

Two other tables, *direction predictors*, indexed by outcome history and address.
One used for more taken, other for more not taken branches.

When branch encountered, choice-predictor counter retrieved ...
... its value used to select one of two direction predictor tables ...
... which is used to obtain counter for predicting branch.

Selected direction counter updated, choice counter usually updated.

Results: Moderate improvement.

Likely Branch Direction Solutions

Static Classification [Chang 94]

Two schemes described.

Common element: identify class of instruction, ...
... mostly one-way branches or mixed direction, ...
... *before* execution.

Static Branch Prediction of Mostly One Way (PG+gshare)

ISA must have branches with hint bits. (Hardware uses hint bits instead of predictor.)

Identify mostly one-way branches before execution, use hint bits for them.

Mixed-direction branches don't use hint bits ...
... use gshare to predict these.

Advantage: BHT used only for mixed-direction branches.

Multiple History Length (GAs.mhl) [Chang 94].

Like gshare, but use short outcome history for mostly one-way branches and full-size outcome history for others.

Determine instruction class before execution.

Number of outcome bits to use based on instruction class.

Comparison, PG+gshare and GAs.mhl

PG+gshare outperforms GAs.mhl, especially at small sizes.

Likely Branch Direction Solutions

Hybrid Predictors [McFarling 93, Chang 94].

Implement two predictors, update both for each branch.

Keep track of which is more accurate for each instruction.

Chang 94 and McFarling 93: Tested one-level 2bC/gshare and PAs/gshare.

- Hybrid predictors better than GAs (and probably gshare).
- PAs/gshare is slightly better than 2bC/gshare at larger sizes.
- Chang 94: PG+gshare much better at small sizes, slightly worse than 2bC/gshare and PAs/share at larger sizes.

Likely Branch Direction Solutions

Method: Variable History Length [Juan 98]

- Based on gshare
- Adjust length of outcome history based on past performance.
- A table of prediction accuracies is maintained.
- Each entry describes accuracy of a particular size.
- Pointer to current size and best size maintained.
- At regular intervals table updated and new best size chosen.

Results: small improvement in prediction accuracy.

Improving Accuracy

Method: Path History

Rationale: path important for prediction, outcomes cannot distinguish all paths.

Instead of outcomes, shift register stores path.

Path consists of addresses of last several branches.

To save space, only a few address bits per branch are used.

Results:

- Not much better than gshare.

Confidence Estimation

Confidence Estimation
Determining the likelihood that a branch prediction is correct.

Used to avoid costly mistakes. (Uses covered in later set.)

Not yet widely used.

Confidence Estimation Measures

- PVP, PVN, Sensitivity, Specificity

Confidence Estimators

- JRS: Estimate high confidence after several consecutive correct predictions.
- Counter: High confidence if BHT counter saturated.
- Fixed Pattern: For PAp, etc: assign high confidence only to pre-determined branch (local) outcome patterns.
- Static: Determine predictable branches before run.

See [Grunwald 97] for details.

Data Value Prediction

Value Prediction
Determining the value of a register or other result without reading it, usually before it is produced.

Possible Applications: (All covered later.)

- Cache Prefetching. (Predict addresses.)
- Dependency Prediction. (Does a load depend on a store?)
- Speculative Execution. (Predict program output. `:-)`)
- Trace Caching. (Avoid true dependency stalls. Lower cost because less bypassing needed.)

Methods (See [Sazeides 97]).

- Stride: Assume constant difference between consecutive values produced by an instruction.
- Context: Assume values produced follow repeating pattern.

Branch Prediction References

Description of two-level schemes.

Yeh 93: Tse-Yu Yeh and Yale N. Patt, "A comparison of dynamic branch predictors that use two levels of branch history," in *Proceedings of the International Symposium on Computer Architecture,* May 1993, pp. 257–266.

Modifications to two-level schemes, and hybrid predictors.

McFarling 93: Scott McFarling, "Combining branch predictors," Digital Equipment Corporation WRL Technical Note TN-36, June 1993.

Analysis of two-level predictor behavior and hybrid predictors.

Chang 94: Po-Yung Chang, Eric Hao, Tse-Yu Yeh, and Yale Patt, "Branch classification: a new mechanism for improving branch predictor performance," in *Proceedings of the Proceedings of the 27th annual international symposium on microarchitecture,* November 1994, pp. 22–31.

Use of path history for branch prediction.

Reches 98: Shlomo Reches and Shlomo Weiss, "Implementation and analysis of path history in dynamic branch prediction schemes," *IEEE Transactions on Computers,* vol. 47, no. 8, pp. 907-912, August 1998.

A gshare-like predictor with separate tables for more taken and not-taken branches.

Lee 97: Chih-Chieh Lee, I-Cheng K. Chen, and Trevor N. Mudge, "The bi-mode branch predictor," Proceedings of the Thirtieth Annual IEEE/ACM International Symposium on Microarchitecture, December 1997, pp. 4–13.

A gshare-like predictor with a dynamically adjusted outcome history size.

Juan 98: Toni Juan, Sanji Sanjeevan, and Juan J. Navarro, "Dynamic history-length fitting: a third level of adaptivity for branch prediction," in *Proceedings of the International Symposium on Computer Architecture,* June 1998, pp. 155–166.

Exploit aliasing by determining if branch direction agrees with others.

Sprangle 97: Eric Sprangle, Robert S. Chappell, Mitch Alsup, and Yale N. Patt, "The agree predictor: a mechanism for reducing negative branch history interference," in *Proceedings of the International Symposium on Computer Architecture,* June 1997, pp. 284–291.

Confidence Estimation and Data Prediction References

Performance Analysis of Confidence Estimators

Grunwald 98: Dirk Grunwald, Artur Klauser, Srilatha Manne, and Andrew Pleszkun, "Confidence estimation for speculation control," in *Proceedings of the International Symposium on Computer Architecture,* June 1998, pp. 122–131.

Accuracy of Some Data Predictors

Sazeides 97: Yinnakis Sazeides and James E. Smith, "The predictability of data values," Proceedings of the Thirtieth Annual IEEE/ACM International Symposium on Microarchitecture, December 1997, pp. 248–258.