Dynamic Path-Based

Branch Correlation

Ravi Nair

IBM Research

nair@watson.ibm.com

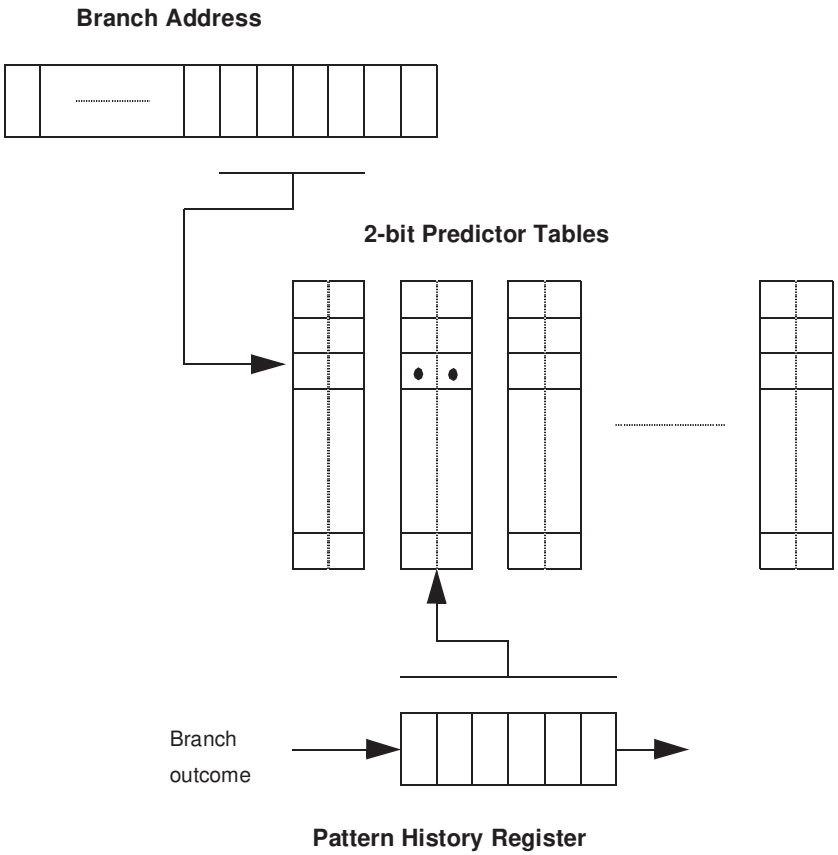
November 29, 1995 MICRO-28

Importance of Branch Prediction

- Late resolution of branch condition could result in pipeline stalling
- Penalty due to stalling can be alleviated by guessing branch outcome
- Two techniques
  - Static -- guess at compile time
  - Dynamic -- guess during execution
- Dynamic branch prediction
  - Takes advantage of local behavior of branch
  - Is usually more effective than static prediction
  - Does not require modification of program
  - Can be made better in succeeding generations of hardware
- Growing use of dynamic prediction
  - Simple predictors (1-bit and 2-bit-counter) generally
  - More complex schemes, eg. branch correlation, now evident

Branch Correlation

- Proposed by Yeh and Patt [1991], Pan, So, and Rahmeh [1992]
- Basic observation
  - The standard prediction scheme determines prediction entirely based on the instance of the branch in the program
  - Additional knowledge about the branch, or the state preceding the branch, could improve prediction accuracy
  - For example, use the pattern represented by the outcome of recent conditional branches



- Asymptotically better results possible with correlation schemes

Path-based Branch Correlation

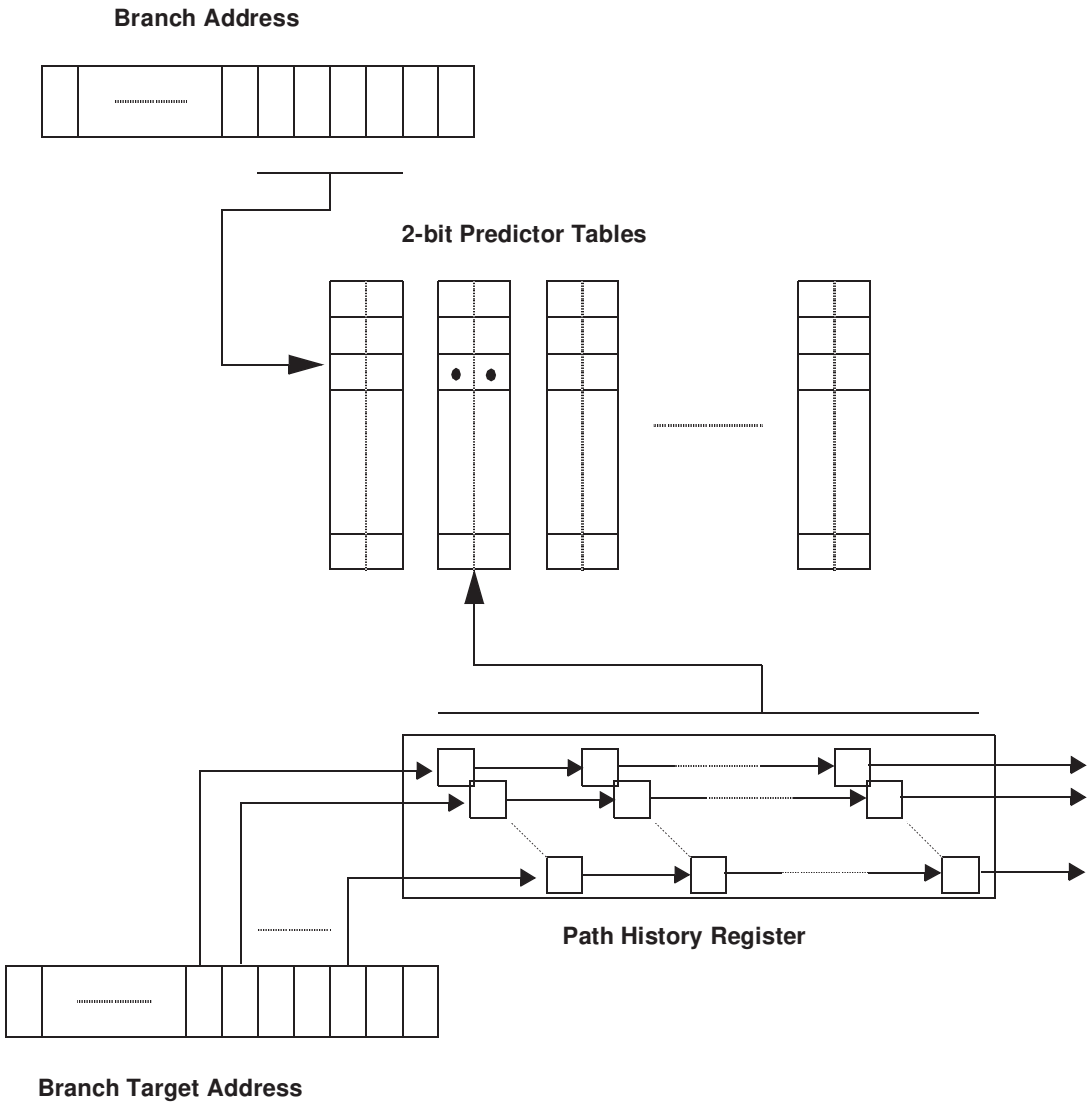
```
            bl      subr        ; Call subroutine 'subr'
            cmpi    cr0,r3,1    ; Does r3 contain 1?
A.          beq     cr0,target  ; If it does, go to target
            ......


      subr: ......
            cmpi    cr1,r5,r6   ; Does r5 equal r6?
B.          bne     label1      ; Do more if it doesn't
            li      r3,1        ; If it does, set r3 to 1
            blr                 ; and return
label1:     ......
            cmpi    cr2,r6,r7   ; Does r6 equal r7?
C.          bne     label2      ;
            li      r3,0        ; If it does, set r3 to 0
            blr                 ; and return
     label2: li     r3,1        ; If it does not, set r3 to 1
            blr                 ; and return
```

- The branch outcome at A cannot be predicted using the outcome at either B or C alone
- However, it can be predicted from the knowledge of which basic block the subroutine returned from
- Path-based correlation attempts to exploit more detailed information about the path leading to a branch

Dynamic Path-Based Branch Correlation

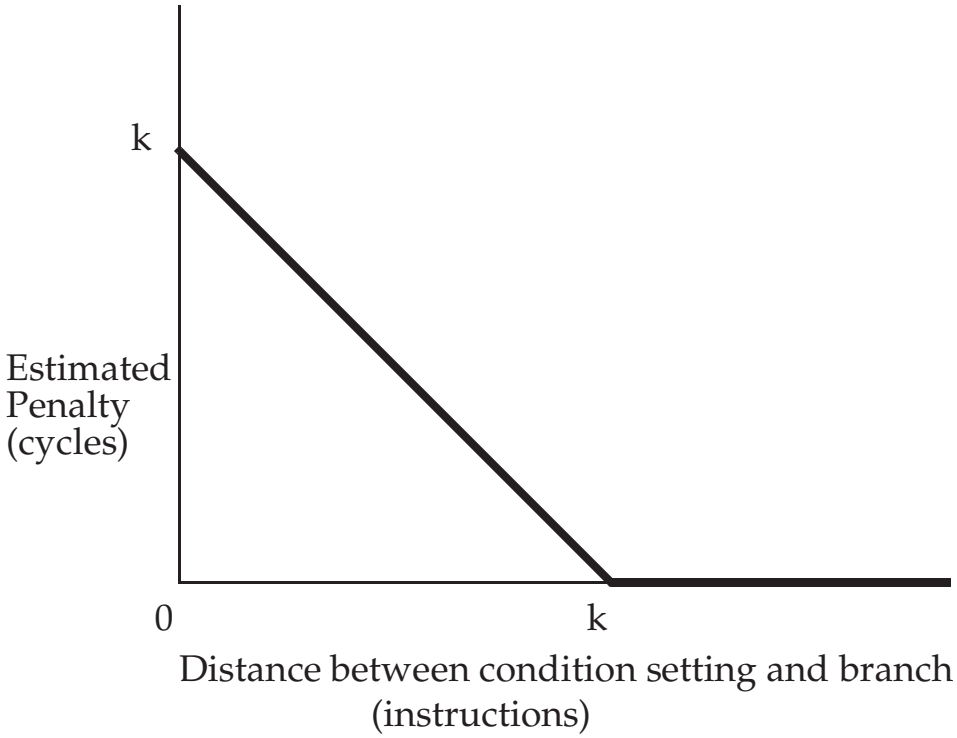
- In static path-based branch correlation [Young and Smith 1994], the compiler keeps track of basic block sequence information
- Dynamic -- hardware keeps track of basic block sequence



  - Advantage: Hardware can exploit local information
  - Disadvantage: Hardware limitations restrict path representation

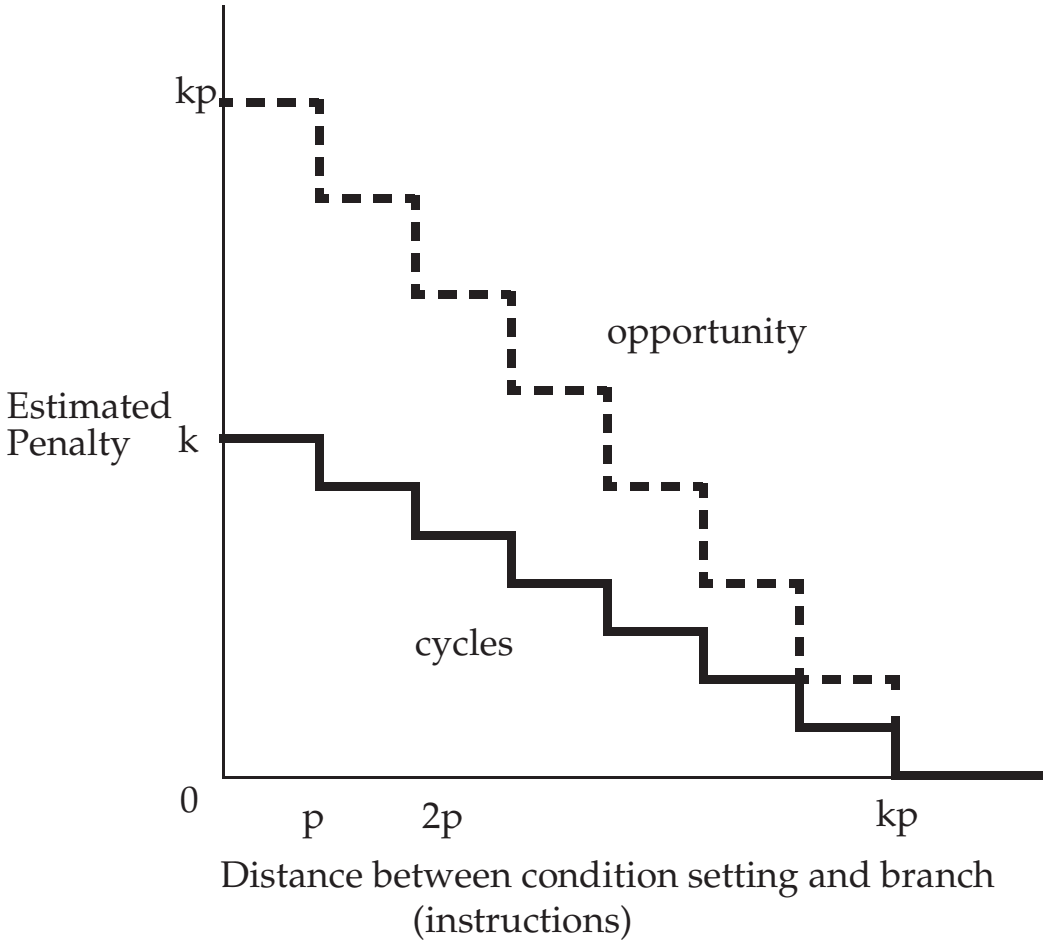
Performance metrics

- Misprediction rate is commonly used to evaluate prediction strategies
- Misprediction rate does not distinguish between branches which are resolved early vs. branches which are resolved late
- Branches resolved early need not be entered in the history table
- Branches resolved late incur variable penalty
- The *k-i* metric attempts to capture penalty, not misprediction rate.
  k = max. # of instructions to be flushed for a mispredicted branch
  i = # of instructions between condition setting instruction and branch



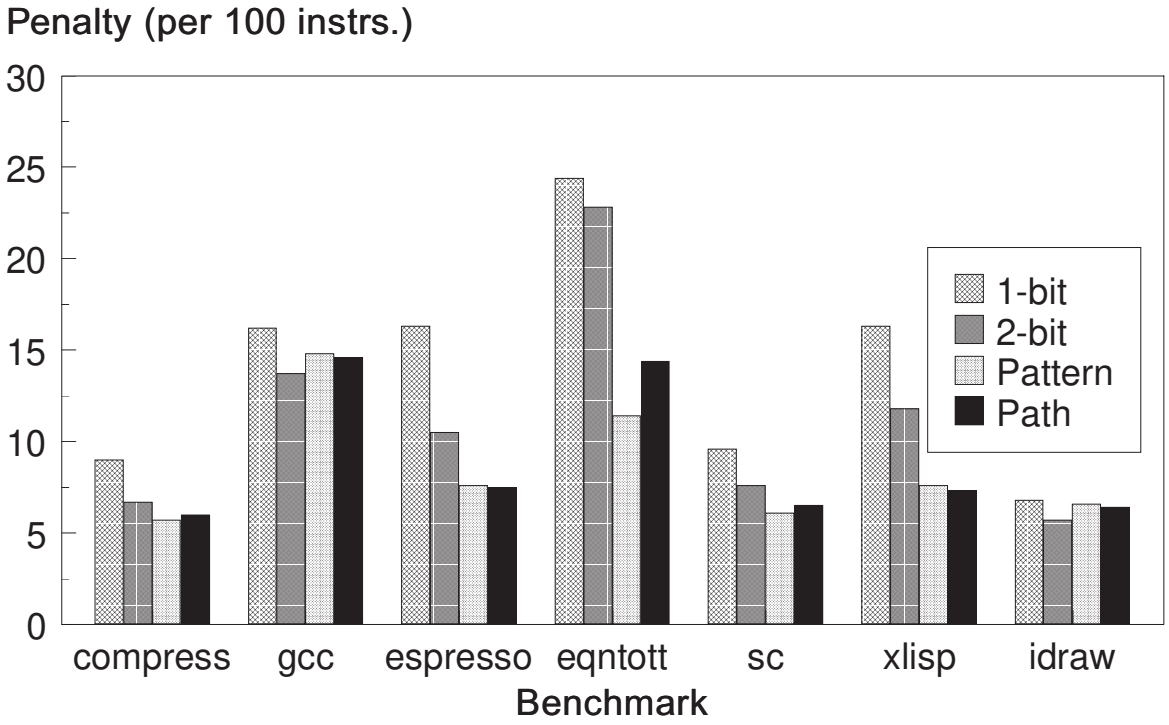
Conditional Branch Penalty in Superscalar Processors

- The opportunity lost due to misprediction
  - Represents the number of useful instructions that could have been executed
  - It increases with the number of functional units in a superscalar machine, *p*



Performance Comparison

- For a given size there are several ways of partitioning the predictor table in either the pattern-based or the path-based correlation scheme
- Branch prediction mechanisms with approximately equal hardware (2048 bits)
  - 1-bit
  - 2-bit counter
  - Pattern-based (8 entries, 7-bit history)
  - Path-based (16 entries, 3-block history)
- All experiments assumed an $8 - i$ penalty model

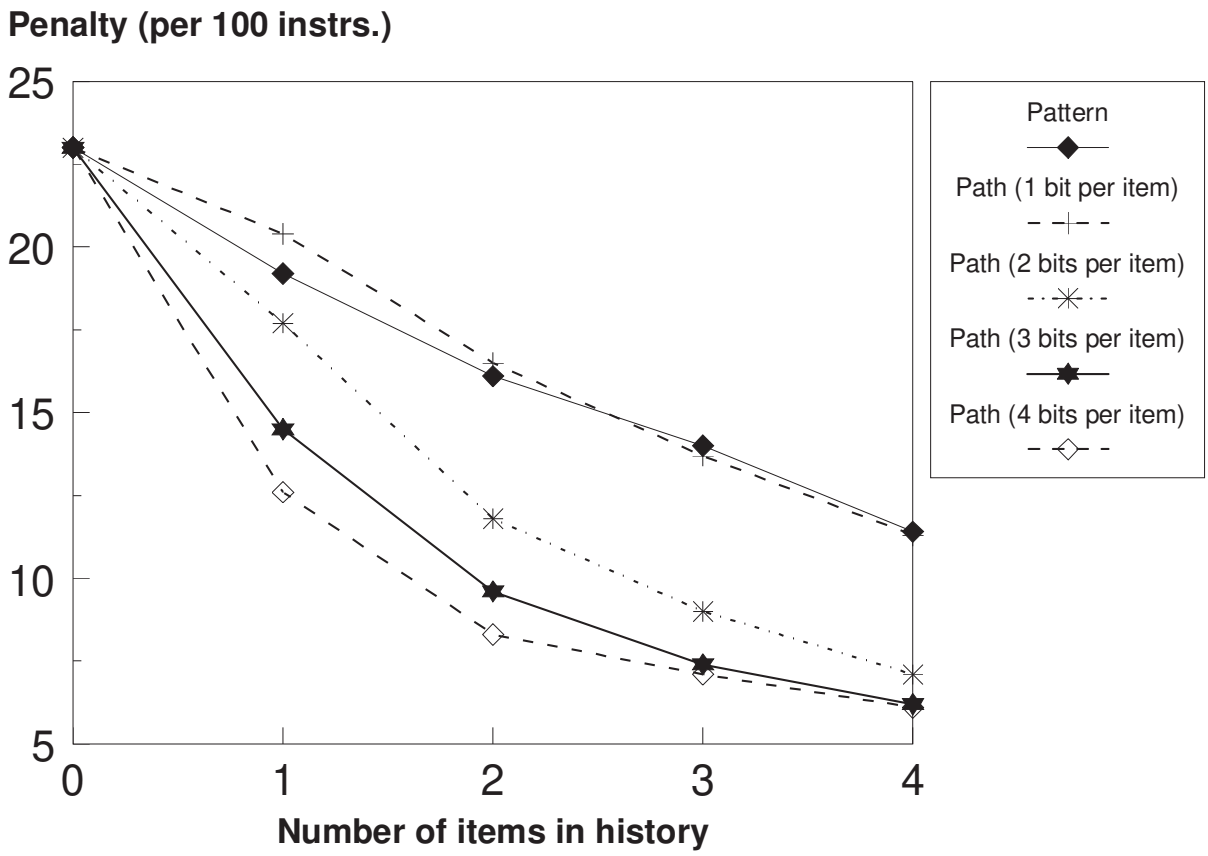


Explanations

- Both correlation schemes consistently outperform 1-bit and 2-bit schemes
- Simple path-based correlation does not appear to be superior to pattern based correlation for a fixed hardware cost
- Large working sets in *gcc* and *idraw* result in better performance for simpler predictors
  - Less collision in predictor table
- *eqntott* takes advantage of longer history information
  - Fewer number of history items in path correlation (3 vs. 7) causes path correlation to be worse for *eqntott*

Increasing Bits per History Item

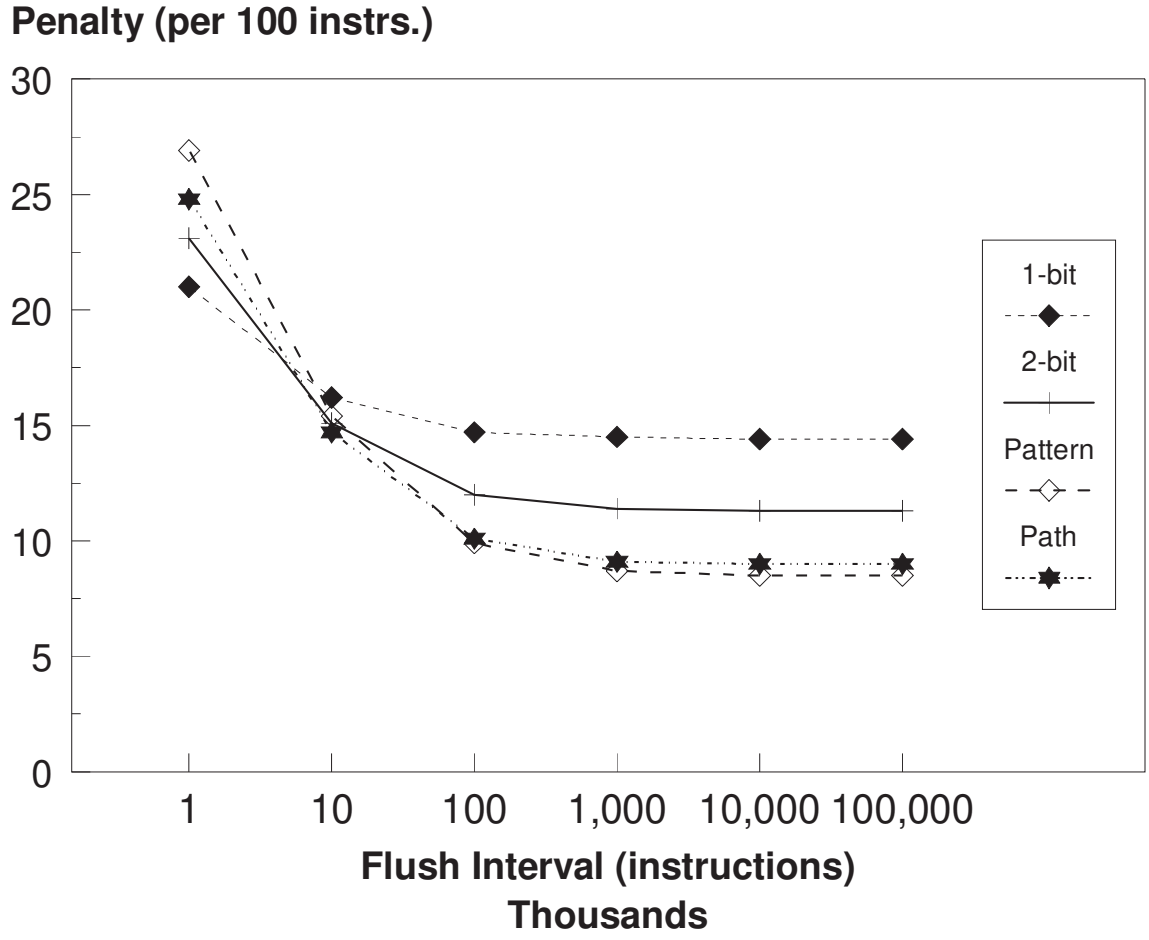
- What would happen if the number of bits allocated to each history item were increased?



- Roughly equal performance for a given hardware cost
- Potential for better prediction performance if a more sophisticated hashing function is used with path-based correlation

Context Switching

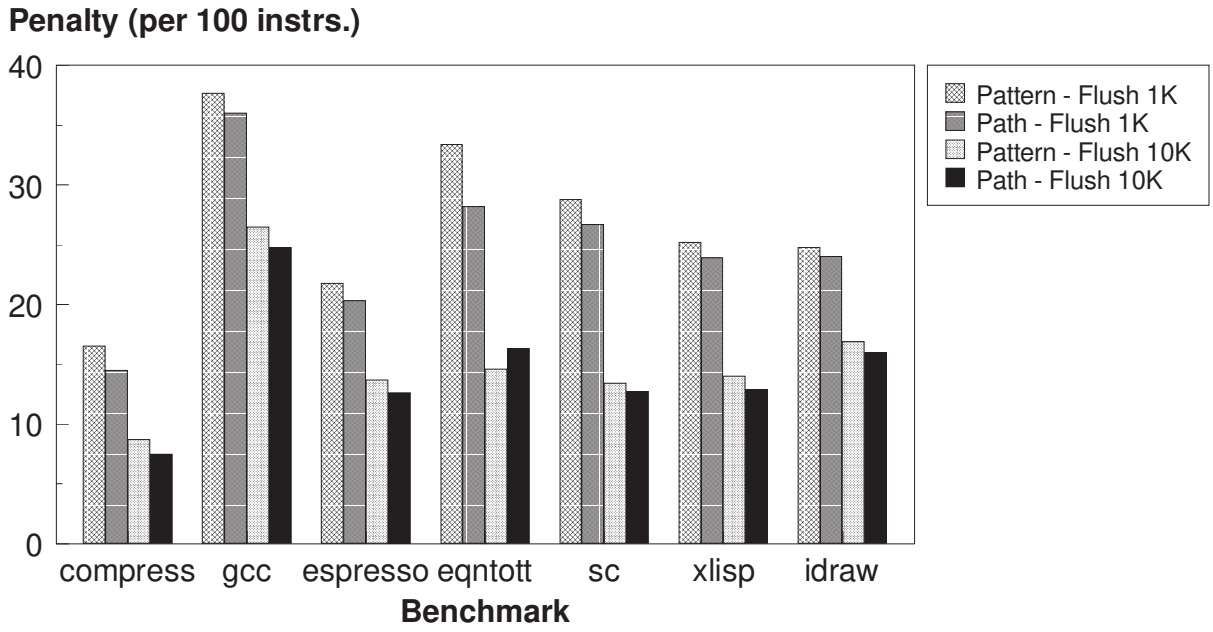
- Context switches could be disruptive to predictor performance
  - Short contexts do not provide opportunity to learn adequately
  - Learnt information is overwritten by information in new context



- Schemes with lower learning overhead have smaller variation in performance with context switching

- Path-based scheme better than pattern-based scheme at higher flush rates

Performance of Each Program with Context Switching



- Path-based scheme consistently better than patten-based scheme at these flush rates

Conclusions

- Path-based correlation provides roughly the same performance as pattern-based correlation for reasonable-sized tables (4K-64K bits)
- Path-based correlation tends to be better than pattern-based correlation in situations where learning overhead is important
- More work needs to be done in determining efficient hardware schemes to encode path information

Remarks

- Misprediction ratio does not adequately capture the performance penalty of a branch prediction scheme
- Should study branch prediction schemes in concert with I-cache fetch schemes
- "Asymptotically good" schemes
  - Usually imply
    - a lot of hardware
    - a large learning overhead
  - May not be cost-effective
    - for situations with high context-switching rates
    - for applications with very large working sets
    - where the workload incurs significant other penalties, eg. high D-cache penalties

Errata

- **Penalty model omitted:** All experiments were performed using the $k - i$ penalty model with $k = 8$.

- **Tables 3 and 4 have identical titles**: Table 3 represents "no context switching." Table 4 represents "flush rate of 1 per 10,000 instructions."