A Multi-component Branch Predictor Design for Low Resource Budget Processors

Moumita Das, Jadavpur University, Kolkata
Ansuman Banerjee, Indian Statistical Institute, Kolkata
Bhaskar Sardar, Jadavpur University, Kolkata
Introduction

- **Branch Predictor**: an important component of modern processors
  - Predicts branch direction at the fetch stage of the pipeline
  - Saves clock cycles as well as energy for deep pipelines
  - On a misprediction:
    - Pipeline flushed and new instructions brought in
    - Leads to loss of cycles and energy

- **Efficient and accurate predictor design** – an always important problem in computer architecture community
  - Prediction strategies in modern processors have reached a fair amount of sophistication over decades of research
    - Bimodal, Gshare, Gag, Perceptron, TAGE and its variants
  - Achieving last mile in prediction accuracy is still important
  - With new workloads, new challenges being envisioned
Motivation behind this research

- A single predictor component may not be well suited for all branches in a program
  - Branches ill-suited for a certain branch prediction policy often have better performance when run with another predictor
  - Multicomponent predictor design

- Multicomponent predictor designs in literature
  - Best predictor for each individual branch
  - Separate prediction history storage tables for each predictor
  - Popular Examples: Tournament predictor, Overriding predictor
Contributions of this work

- **A new multicomponent predictor design**
  - Use of multiple predictor components synergistically instead of a single one for prediction at runtime for better prediction accuracy
  - Does not necessarily switch to the best predictor for each branch

- **Ensuring a low storage budget design**
  - Sharing predictor tables between multiple predictor components
  - A heuristic to minimize inter-predictor interference

- **Experimental results on SPEC 2006**
  - Accuracy, energy and execution cycle
A multi-component predictor design

- A combination of 3 components: GShare, Gag and Bimodal predictors
  - Differ in their indexing functions
- Each predictor needs two main storage elements:
  - Branch History Register (BHR)
    - A shift register
    - Stores the branch outcomes of the most recent n branches
    - A 1 is recorded for a *taken* branch, 0 recorded for a *not taken* branch
  - Pattern History Table (PHT)
    - Stores the prediction information for all branches of a program
    - Contains a two bit saturating counter: MSB gives the final prediction
Going for a low storage budget design

■ Classical multi-component predictor design:
  ● Prediction accuracy improvement over single components
  ● Individual PHTs for individual predictor components

■ Our proposal for low storage budget: a single shared PHT
  ● Leads to prediction accuracy degradation with respect to single components

Accuracy comparison: shared vs individual PHT implementation
Addressing the accuracy degradation

- **Inter predictor interference**
  - Information stored by one predictor accessed and modified by others
  - Increase in Misprediction

- **Too many instances of predictor switching**
  - Switching between predictors to employ the best predictor for each branch does not necessarily help

- **Question:** Can we find a predictor usage sequence that minimizes mispredictions, and therefore, maximizes accuracy?
Complexity of the solution space

- Simulate all sequences of possible combinations of all branches for all predictors
- Example: 4 predictors and 4 branches
  - 256 combinations
  - Infeasible for program with more branches

Predictor Sequence Tree
Our multi-component predictor strategy

- **Attempts to reduce the instances of inter-predictor interference**
- **Does not necessarily switch predictors at each branch**
  - For each branch, evaluates the benefit of changing the predictor from the one currently being used
  - Changes the current predictor only if accuracy improvement is significant

```
Algorithm /* A_q denotes the prediction accuracy of predictor q */
R = predictor with maximum average prediction accuracy for a program
Initialize currentPredictor C = R
For each branch i in the program do
    Let P_i be the best predictor if available, or else P_i = R
    if |A_c - A_{P_i}| < θ then
        select C for prediction of i
    else
        select P_i for prediction of i
    update C = P_i
End Algorithm
```
Discussion on our proposal

- Can still manage with a single PHT shared between the components

- Accuracy degradation is not as significant as in the naive shared design

- Controls interference using lesser number of predictors
  - Total number of interfering predictors is usually less than the classical scheme
  - The worst case number of predictor switches in our case is upper bounded by the number of switches in a classical scheme
Experimental setup

- Tejas architectural simulator
- SPEC 2006 benchmark programs

- Shared PHT implementation
  - 32 KB PHT size – shared by three predictors

- Compared the performance benefits of our design with
  - A naïve shared PHT implementation (without our heuristic)
  - Split PHT implementation, with 16 KB PHT for each predictor
Comparing the number of interferences

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Interference (in shared implementation) (%)</th>
<th>Interference (using switching algorithm) (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>403.gcc</td>
<td>3.7</td>
<td>2.2</td>
</tr>
<tr>
<td>400.perlbench</td>
<td>4</td>
<td>3.3</td>
</tr>
<tr>
<td>429.mcf</td>
<td>1.2</td>
<td>0.5</td>
</tr>
<tr>
<td>458.sjeng</td>
<td>1</td>
<td>0.2</td>
</tr>
<tr>
<td>456.hmmr</td>
<td>1.1</td>
<td>0.05</td>
</tr>
<tr>
<td>447.dealII</td>
<td>1.0</td>
<td>0.1</td>
</tr>
<tr>
<td>464.h264ref</td>
<td>2.0</td>
<td>0.7</td>
</tr>
<tr>
<td>450.soplex</td>
<td>2.5</td>
<td>1.2</td>
</tr>
<tr>
<td>401.bzip2</td>
<td>3.4</td>
<td>0.4</td>
</tr>
</tbody>
</table>

- % of interferences in the naive shared implementation versus ours
- Achieved reduction in all cases
Prediction Accuracy Comparison

- Average accuracy improvement - around 2%-3%
Energy Comparison

- Improvement in execution time over a naive shared PHT implementation
- Improvement recorded over a SPLIT table implementation
Execution Time Comparison

- Improvement in execution time over a naive shared PHT implementation
- Improvement recorded over a SPLIT table implementation
Comments on our approach

- Leads to improvement in accuracy, energy and execution time

- Utilizes a static profile based selection scheme to identify and record the best predictor for each branch
  - Need to store this information for use at execution time
  - Decision to switch done at execution time

- Attempts to minimize negative interference
  - Benefits of positive interference lost as well

- Can implement this at run-time, switching at phases
  - Can keep track of best predictor information as execution progresses

Thank You!