2D-Profiling
Detecting Input-Dependent Branches with a Single Input Data Set

Hyesoon Kim
M. Aater Suleman
Onur Mutlu
Yale N. Patt

HPS Research Group
The University of Texas at Austin
Motivation

- Profile-guided code optimization has become essential for achieving good performance.
  - Run-time behavior $\equiv$ profile-time behavior: Good!
  - Run-time behavior $\neq$ profile-time behavior: Bad!
Motivation

- Profiling with one input set is not enough!
  - Because a program can show different behavior with different input data sets
  - Example: Performance of predicated execution is highly dependent on the input data set
  - Because some branches behave differently with different input sets
Input-dependent Branches

- **Definition**
  - A branch is input-dependent if its misprediction rate differs by more than some Δ over different input data sets.

<table>
<thead>
<tr>
<th>Misprediction rate of Br. X</th>
<th>Inp. 1</th>
<th>Inp. 2</th>
<th>Inp.1 – Inp. 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>30%</td>
<td>29%</td>
<td>1%</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Misprediction rate of Br. Y</th>
<th>Inp. 1</th>
<th>Inp. 2</th>
<th>Inp.1 – Inp. 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>30%</td>
<td>5%</td>
<td>25%</td>
<td></td>
</tr>
</tbody>
</table>

- **Input-dependent branch**

- **Input-dependent br. ≠ hard-to-predict br.**
An Example Input-dependent Branch

- Example from Gap (SPEC2K):
  Type checking branch

```
TypHandle Sum (A,B)

If ((type(A) == INT) && (type(B) == INT)){ //input-dependent br
    Result = A + B;
    Return Result;
}
Return SUM(A, B):
```

- Train input set: A&B are integers 90% of the time
  - misprediction rate: 10%
- Reference input set: A&B are integers 42% of the time
  - misprediction rate: 30% (30%-10%) > Δ
Predicated Execution

(normal branch code)

if (cond) {
  b = 0;
}
else {
  b = 1;
}

(predicated code)

Eliminate hard-to-predict branches
but fetch blocks B and C all the time
Predicated Code Performance vs. Branch Misprediction Rate

Predicated code performs better

Normal branch code performs better

- Converting a branch to predicated code could hurt performance if run-time misprediction rate is lower than profile-time misprediction rate
Predicated Code Performance vs. Input Set

Predicated execution loses performance because of input-dependent branches

Measured on an Itanium-II machine
If We Know a Branch is Input-Dependent

- May not convert it to predicated code.
- May convert it to a wish branch.
  [Kim et al. Micro’05]

- May not perform other compiler optimizations or may perform them less aggressively.
  - Hot-path/trace/superblock-based optimizations
    [Fisher’81, Pettis’90, Hwu’93, Merten’99]
Our Goal

Identify input-dependent branches by using a single input set for profiling
Talk Outline

- Motivation
- 2D-profiling Mechanism
- Experimental Results
- Conclusion
Key Insight of 2D-profiling

**Phase behavior** in prediction accuracy is a good indicator of **input dependence**

![Graph showing branch prediction accuracy over time](image)

- **Phase 1**
- **Phase 2**
- **Phase 3**
Traditional Profiling

\[ \text{MEAN } \text{pr. Acc}(\text{brA}) \]

\[ \text{MEAN } \text{pr. Acc}(\text{brB}) \]

\[ \text{MEAN } \text{pr. Acc}(\text{brA}) \equiv \text{MEAN } \text{pr. Acc}(\text{brB}) \]

behavior of brA \equiv \text{behavior of brB}
2D-profiling

\[ \text{MEAN pr. Acc}(brA) \approx \text{MEAN pr. Acc}(brB) \]
\[ \text{STD pr. Acc}(brA) \neq \text{STD pr. Acc}(brB) \]

behavior of brA \neq \text{behavior of brB}

A: input-dependent br, B: input-independent br
2D-profiling Mechanism

- The profiler collects branch prediction accuracy information for **every static branch over time**

  \[
  \text{Slice 1} \quad \text{Slice 2} \quad \ldots \quad \text{Slice N}
  \]

  \[
  \text{mean Pr.Acc(brA, s1)} \quad \text{mean Pr.Acc(brA, s2)} \quad \ldots \quad \text{mean Pr.Acc(brA, sN)}
  \]

  \[
  \text{mean Pr.Acc(brB, s1)} \quad \text{mean Pr.Acc(brB, s2)} \quad \ldots \quad \text{mean Pr.Acc(brB, sN)}
  \]

  \[
  \text{Calculate MEAN (brA, brB, ...)}
  \]

  \[
  \text{Standard deviation (brA, brB, ...), PAM: Points Above Mean (brA, brB, ...)}
  \]

  \[
  \text{PAM:50%} \quad \text{PAM:0%}
  \]
Input-dependence Tests

- STD&PAM-test: Identify branches that have **large variations** in accuracy over time (phase behavior)
  - STD-test (STD > threshold): Identify branches that have **large variations** in the prediction accuracy over time
  - PAM-test (PAM > threshold): **Filter** out branches that pass STD-test due to a few outlier samples

- MEAN&PAM-test: Identify branches that have **low prediction accuracy** and **some time-variation** in accuracy
  - MEAN-test (MEAN < threshold): Identify branches that have **low prediction accuracy**
  - PAM-test (PAM > threshold): Identify branches that have **some variation** in the prediction accuracy over time

- A branch is classified as **input-dependent** if it passes either STD&PAM-test or MEAN&PAM-test
Talk Outline

- Motivation
- 2D-profiling Mechanism
- Experimental Results
- Conclusion
Experimental Methodology

- Profiler: PIN-binary instrumentation tool
- Benchmarks: SPEC 2K INT
- Input sets
  - Profiler: Train input set
  - Input-dependent Branches: Reference input set and train/other extra input sets
- Input-dependent branch: misprediction rate of the branch changes more than $\Delta = 5\%$ when input data set changes
  - Different $\Delta$ are examined in our TechReport [reference 11].
- Branch predictors
  - Profiler: 4KB Gshare, Machine: 4KB Gshare
  - Profiler: 4KB Gshare, Machine: 16KB Perceptron (in paper)
Evaluation Metrics

Coverage and Accuracy for input-dependent branches

Correctly Predicted Input-dependent br.

\[ \text{COV} = \frac{A \cap B}{A} = \frac{\text{Correctly Predicted}}{\text{Actual Input - dependent}} \]

Predicted Input-dependent br. (2D-profiler)

\[ \text{ACC} = \frac{A \cap B}{B} = \frac{\text{Correctly Predicted}}{\text{Predicted Input - dependent}} \]
Input-dependent Branches

% of branches that are input-dependent

- bzip2
- gzip
- twolf
- gap
- crafty
- gcc

Legend:
- 2 input sets
- 3 input sets
- 4 input sets
- 5 input sets
- 6 input sets
- 7 input sets
- 8 input sets
2D-profiling Results

Phase behavior and input-dependence are strongly correlated!
The Cost of 2D-profiling?

- 2D-profiling adds only 1% overhead over modeling the branch predictor in software
- Using a H/W branch predictor [Conte’96]
Conclusion

- 2D-profiling is a new profiling technique to find input-dependent characteristics by using a single input data set for profiling
- 2D-profiling uses time-varying information instead of just average data
- Phase behavior in prediction accuracy in a profile run → input-dependent
- 2D-profiling accurately identifies input-dependent branches with very little overhead (1% more than modeling the branch predictor in the profiler)
- Applications of 2D-profiling are an open research topic
  - Better predicated code/wish branch generation algorithms
  - Detecting other input-dependent program characteristics