A Framework for Modeling and Optimization of Prescient Instruction Prefetch

ACM Sigmetrics 2003 – June 12, 2003

Tor M. Aamodt†‡, Pedro Marcuello§, Paul Chow‡, Antonio Gonzalez§
Per Hammarlund¶, Hong Wang†, John P. Shen†

†Microprocessor Research, Intel Labs
‡Dept. of Electrical and Computer Engineering, University of Toronto
§Intel Barcelona Research Center
¶Desktop Products Group, Intel Corp
Multithreading

Single chip, multiple flows of control

Question: How might a single-threaded application exploit this hardware capability?
Helper Threads

Use spare thread context(s) to reduce μArch bottlenecks. Typically do not need to satisfy all correctness constraints.

Related work on helper threads

- Helper threads: Chappell & Patt, Dubois & Song
- Slices: Zilles & Sohi, Roth & Sohi
- Data prefetch: Zilles & Sohi, Collins et al., Annavaram & Davidson, Luk, Moshovos et al., Liao et al.
- Branch prediction: Chappell & Patt

This work: first work to study using helper threads for instruction prefetch (may also help TC pre-building)
Existing/Proposed Techniques

- Traditional hardware - scalability
- Helper thread – a few “delinquent” instruction
- Runahead – need simultaneous I & D miss
Prescient Instruction Prefetch

Prefix

Infix

Postfix

spawn

target

I-cache Misses

Main Thread

Helper Thread

Tor M. Aamodt
Optimization of Prescient Instruction Prefetch

- Optimization problem can be divided into two parts
  1. Selection of SPAWN-TARGET pairs
  2. Optimization of resulting thread code, and hardware used to run it
- This paper focuses on the first issue only

Optimization Algorithms

Path Expression Mappings

Stochastic Path Analysis

HW Abstraction
HW Abstraction

Memory

Fully Associative I-Cache
(line size = 1 inst.)

Instruction Sequencer

Intra-procedural control flow = Markov Chain
Call/returns paired
HW Abstraction
Prescient Instruction prefetch

Instructions

\( o(s,t) \)  \( \text{slack}(i,s,t) \)

Time (cycles)

Tor M. Aamodt
Spawn-Target Selection Tradeoffs

- S&T highly correlated
- S and T should be far apart so slack is larger than memory latency.
- S->T instruction footprint should fit in I-cache; T->S should not.
## Quantifying Tradeoffs

<table>
<thead>
<tr>
<th>METRIC</th>
<th>Aspect Quantified</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reaching Probability</td>
<td>accuracy</td>
</tr>
<tr>
<td>Posteriori Probability</td>
<td>coverage</td>
</tr>
<tr>
<td>Expected Path Length</td>
<td>timeliness</td>
</tr>
<tr>
<td>Path Length Variation</td>
<td></td>
</tr>
<tr>
<td>Path Footprint</td>
<td>timeliness, necessity</td>
</tr>
</tbody>
</table>

Tor M. Aamodt
Spawn-Target Selection Algorithm

- **Inputs:** Profile data, estimated CPI
- **Compute metrics / spawn-target value function**
- **Select using greedy heuristic**
Path Expressions

• Regular expression describing all paths between two points.

\[ P(a, X) = A \cdot \left( \left( \left( B \cdot C \right) \cup \left( D \cdot E \right) \right) \cdot F \right)^* \cdot B \]

**Fast Path Expression Algorithm**

• [Tarjan 1981]: general approach to solving path problems efficiently.

• Examples: solving \( Ax = b \), shortest paths, data flow analysis.
Example: Reaching Probability

concatenation: \[ [R_1 \cdot R_2] = pq \]

union: \[ [R_1 \cup R_2] = p + q \]

closure: \[ [R_1^*] = \frac{1}{1 - p} \]

\[
P(a, X) = A \cdot \left( \left( \left( B \cdot C \right) \cup \left( D \cdot E \right) \right) \cdot F \right)^* \cdot B
\]

\[
[P(a, X)] = 0.98 \cdot \left( \frac{1}{1.0 - (0.1(0.0) + 0.90(1.0)) \cdot (0.999)} \right) \cdot 0.10
\]

\[
\approx 0.97
\]
## Mappings

<table>
<thead>
<tr>
<th>Concatenation $[R_1 \cdot R_2]$</th>
<th>Reaching Probability $pq$</th>
<th>Expected Path Length $X + Y$</th>
<th>Path Length Variance $v + w$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Union $[R_1 \cup R_2]$</td>
<td>$p + q$</td>
<td>$\frac{pX}{1 - p}$</td>
<td>$\frac{p(v + X^2) + q(w + Y^2)}{p + q} - \left(\frac{pX + qY}{p + q}\right)^2$</td>
</tr>
<tr>
<td>Closure $[R_1^*]$</td>
<td>$\frac{1}{1 - p}$</td>
<td>$\frac{pX}{1 - p}$</td>
<td>$\frac{p(v + X^2)}{1 - p} + \left(\frac{pX}{1 - p}\right)^2$</td>
</tr>
</tbody>
</table>

**Decompose problem:** $\sum E[X|\text{ follow } p \in R] \cdot P[\text{ follow } p \in R]$
Path Footprint

\[
F(x, y) = \frac{1}{RP(x, y)} \sum_v \text{size}(v) \cdot RP_{\alpha}(x, v | \neg y) \cdot RP_{\beta}(v, y)
\]
Accuracy: vs. Monte Carlo

Reaching Probability

Path Length Variation

Expected Path Length

Path Footprint
Accuracy: vs. Execution

- Reaching Probability
- Expected Path Length
- Path Length Variation
- Path Footprint
Spawn-Target Selection Algorithm

I-cache & edge profiling data estimated helper thread & main thread CPI

Partition large basic blocks

Summarize procedures

Next procedure in bottom-up traversal of call graph

Use fast path algorithm to find path expressions. Compute RP, PP, path length mean & variance.

Update estimated # running helper threads, and I-cache miss coverage

Select next block

Select earliest target within \( \frac{1}{2} \) max prefetch distance

Select set of spawn-points (compute I-cache footprint on-demand)

Set of spawn points found: OUTPUT spawn pts., target, max. prefetch

Coverage of all basic blocks acceptable or no pair found

No suitable points

Done
Selection Algorithm Details

Loop over basic blocks (ranked by $E[\#i\text{-}misses]$)

1. Select target, then select spawn
   
   \[
   \text{value}(\text{spawn}) \propto \]
   
   $PP \cdot RP \cdot E[\text{postfix size}] \cdot P[\text{miss}] \cdot P[\text{evicted}] \cdot P[\# \text{ ht} < \# \text{ ctx}]$

2. Update coverage metrics
   
   \[
   \begin{align*}
   \# \text{ helper threads} &= \sum PP(s,i) \cdot P[\text{still running}] \\
   \# \text{ i-cache misses} &= PP(t,s) \cdot PP(t,i) \cdot P[\text{still running}] \cdot (\# \text{i-cache misses})
   \end{align*}
   \]
## Hardware Details

<table>
<thead>
<tr>
<th>Threading</th>
<th>SMT processor with 2, 4, or 8 hardware contexts</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pipelining</td>
<td>In-order 12-stage pipeline</td>
</tr>
<tr>
<td>Fetch</td>
<td>2 bundles from 1, or 1 bundle from 2 threads, prioritizing main thread, helpers ICOUNT</td>
</tr>
<tr>
<td>I-prefetch</td>
<td>next line prefetch (triggered on miss)</td>
</tr>
<tr>
<td></td>
<td>Stream prefetch triggered by compiler hints</td>
</tr>
<tr>
<td></td>
<td>(max. 4 outstanding prefetches per context)</td>
</tr>
<tr>
<td>Branch Pred.</td>
<td>2k-entry gshare. 256-entry 4-way assoc. BTB. Helper threads oracle branch prediction (always follow correct path)</td>
</tr>
<tr>
<td>Issue</td>
<td>2 bundles from 1, or 1 bundle from 2 threads</td>
</tr>
<tr>
<td></td>
<td>Prioritize main thread, helpers: round-robin</td>
</tr>
<tr>
<td>Caches</td>
<td>L1 (separate I&amp;D): 16KB  4-way 1-cycle</td>
</tr>
<tr>
<td></td>
<td>L2 (shared) : 256 KB  4-way 14-cycles</td>
</tr>
<tr>
<td></td>
<td>L3 (shared) : 3072 KB  12-way 30-cycles</td>
</tr>
<tr>
<td>Memory</td>
<td>230-cycles, TLB miss = 30 cycles</td>
</tr>
</tbody>
</table>
Performance Impact

![Bar chart showing speedup comparison across different tasks and conditions.]

- **Legend:**
  - ideal
  - 2t
  - 4t
  - 8t

- **Tasks:**
  - 145.fppp
  - 177.mesa
  - 186.crafty
  - 252.eon
  - 255.vortex
  - avg

- **Y-axis:** Speedup
  - Range: 1.0 to 1.3

- **X-axis:** Tasks

Tor M. Aamodt
Performance Impact

![Graph showing speedup comparison for various applications.](image-url)
Source of remaining I-cache misses

Miss Breakdown

- evicted
- too-slow
- no-context
- no-spawn
- no-target
# Helper Thread Characteristics

<table>
<thead>
<tr>
<th>benchmark</th>
<th>static # pairs</th>
<th>dyn. #pairs</th>
<th>infix region size</th>
<th>postfix region size</th>
</tr>
</thead>
<tbody>
<tr>
<td>145.fpppp</td>
<td>62</td>
<td>378528</td>
<td>622</td>
<td>162</td>
</tr>
<tr>
<td>177.mesa</td>
<td>34</td>
<td>210519</td>
<td>1186</td>
<td>255</td>
</tr>
<tr>
<td>186.crafty</td>
<td>166</td>
<td>560200</td>
<td>573</td>
<td>129</td>
</tr>
<tr>
<td>252.eon</td>
<td>152</td>
<td>407516</td>
<td>691</td>
<td>120</td>
</tr>
<tr>
<td>255.vortex</td>
<td>1348</td>
<td>438722</td>
<td>1032</td>
<td>142</td>
</tr>
</tbody>
</table>
Summary

- Limit study indicates Prescient Instruction Prefetch may yield speedups from 4.8% to 17%
- Introduce a framework for selecting spawn-target pairs based upon statistical analysis using path expressions.
- Future work
  - hardware & slice optimization (under review)
  - more sophisticated modeling (branch corr., phases)
  - optimization algorithms