Frequently Used Trace Cache

[efurbish@hoss simplesim-2.0] > wc -l trace_cache.c
666

Advanced Computer Architecture,
under Dr. Scott Rixner

Marc Power
Eric Furbish
Indraneel Datta
Motivation

- Large trace caches suffer increased latency
- Lowers performance below ideal

We’d like to minimize the performance lost to latency.
To recover from the ill effects of latency in the trace cache, we need some low latency means of getting at the same data.

Normally, solve by implementing L1/L2 cache hierarchy

- Contention in the L1 can evict frequently used lines with infrequently used ones
- Contention could possibly be reduced by intelligently filling the L1

**Frequently Used Trace Cache (FUTC)** –

- Single-cycle latency, small L1
- Judiciously filled from L2 with frequently used lines

Given the new gizmo, we hoped:

\[ IPC(TC_s + FUTC_i) > IPC(TC_s + L1TC_i) > IPC(TC_s) \]
FUTC Implementation

- One saturating counter per trace cache line
  - Incremented on read hit
- Counters cleared on TC writes
  - Rewrites not propagated into FUTC
    - Saves read port
    - Maintains logical meaning of counters (apply to a specific trace)
- Lines with counters over threshold are promoted
  - Promoted on a read hit
  - Counter is not reset
Figure Modified from (E. Rotenbert, S. Bennett, J. Smith. Trace cache: a low latency approach to high bandwidth instruction fetching. Tech Report 1310, CS Dept., Univ. of Wisc.-Madison, 1996)
Area and Power Costs

- Size of FUTC (8-64KB is reasonable) in order to remain under single-cycle latency
- Counters consume 1-3KB for 1MB trace cache
  - Assumes threshold of 1-8 (1-3 bits)
- Power cost is minimal
  - No worse than L1 trace cache
  - Single-ported for speed and power
Methodology

1. Use “infinite” backend to stress fetch mechanism

2. Select ‘Optimal’ TC Size
   - Based on ideal (single-cycle) performance vs. performance with latency accounted for
   - Pick best candidate for improvement with FUTC

3. Run varying “approximate L1” sizes (8-64KB)
   - L1 = FUTC with threshold 0

4. Run varying FUTC sizes (8-64KB) with thresholds of 1, 2, 4 and 8

5. Compare TC/FUTC and L1/FUTC
Verification:

- Branch predictor accuracies of 90-99%
- Exactly the range we would expect
- Performance and trace cache hit rates very similar to trace cache paper
- Dispatch and commit streams are identical to unmodified sim-outorder
- Program flow is guaranteed to be correct
- It has seen billions of instructions without flaws
Results

IPC vs. FUTC size/Threshold

<table>
<thead>
<tr>
<th>Size/Threshold</th>
<th>0</th>
<th>0.2</th>
<th>0.4</th>
<th>0.6</th>
<th>0.8</th>
<th>1</th>
<th>1.2</th>
<th>1.4</th>
<th>1.6</th>
</tr>
</thead>
<tbody>
<tr>
<td>8KB</td>
<td>0</td>
<td>0.2</td>
<td>0.4</td>
<td>0.6</td>
<td>0.8</td>
<td>1</td>
<td>1.2</td>
<td>1.4</td>
<td>1.6</td>
</tr>
<tr>
<td>16KB</td>
<td>0</td>
<td>0.2</td>
<td>0.4</td>
<td>0.6</td>
<td>0.8</td>
<td>1</td>
<td>1.2</td>
<td>1.4</td>
<td>1.6</td>
</tr>
<tr>
<td>32KB</td>
<td>0</td>
<td>0.2</td>
<td>0.4</td>
<td>0.6</td>
<td>0.8</td>
<td>1</td>
<td>1.2</td>
<td>1.4</td>
<td>1.6</td>
</tr>
<tr>
<td>64KB</td>
<td>0</td>
<td>0.2</td>
<td>0.4</td>
<td>0.6</td>
<td>0.8</td>
<td>1</td>
<td>1.2</td>
<td>1.4</td>
<td>1.6</td>
</tr>
</tbody>
</table>

Test cases:
- test-fmath
- test-llong
- test-lswlr
- test-math
- test-printf
- ammp
- vpr
- mcf
FUTC Miss Rate vs FUTC Size/Threshold

Size/Threshold:
- 8KB
- 16KB
- 32KB
- 64KB

Miss Rate:
- 0.5
- 0.6
- 0.7
- 0.8
- 0.9
- 1.0
- 1.1

Test cases:
- test-fmath
- test-llong
- test-lswlr
- test-math
- test-printf
- ammp
- vpr
- mcf
Best-Case FUTC vs L1 and TC

Normalized IPC

- TC Only
- L1 TC
- FUTC

32KB
16KB
16KB
16KB

amm
mcf
test-printf
test-math
- Run larger programs
- Increased contention could make FUTC effective
- Use partial matching and inactive issue
- Higher TC hit rate = more promotions = more contention
Conclusions

- Hypothesis was partly correct
- Small auxiliary cache always helps over TC
- Contention in L1 outweighed by FUTC “warmup”
- New Hypothesis:
  - \( IPC(TC_s + L1\ TC_i) > IPC(TC_s + FUTC_i) > IPC(TC_s) \)
  - Really: \( IPC(TC_s + AUX-TC_i) > IPC(TC_s) \)