Selective Fill Data Cache

Final Presentation

Anuj Dharia
Paul Rodriguez
Ryan Verret
Motivation

- Superscalar clock frequency and issue width continue to increase
- Cache sizes remain the same or decrease to accommodate the clock
- More frequent data accesses place greater strain on cache
- Must be more selective about what is allowed in cache
Hypothesis

Cache efficiency can be improved by selectively preventing infrequently used data blocks from filling the L1 cache. Data consistently evicted from the cache before a subsequent access ought not enter if it is to evict useful data.
Implementation

- Processor Core
- Bypass Buffer
- L1 Data Cache
- Cache Fill Policy Table
- L2 Cache

Data Path
Address Path
Control Path
Architecture Details

- L1 Data Cache
  - Used bits

- Cache Fill Policy Table
  - Direct mapped
  - Size proportional to number of sets
  - Contains threshold information

- Bypass Buffer
  - 2-way set associative
  - Variable size (optimally 1/16 the total cache size)
Base Configuration

8k Direct Mapped (VPR)

Number of Accesses

Misses
Hits

Reuse Distance

0 1 10 100 1000 10000 100000 1000000

1 10 100 1000 10000 100000 1000000

Number of Accesses

0 500000 1000000 1500000 2000000 2500000
Selective Fill Data Cache

8k Direct Mapped with Selective Fill Data Cache (VPR)

Number of Accesses

Misses

Hits
Victim Cache

8k Direct Mapped with Victim Cache (VPR)

Number of Accesses

Misses

Hits

Reuse Distance

1 10 100 1000 10000 100000 1000000
Bypass Buffer Analysis

Bypass Buffer Size Analysis For VPR Benchmark

Miss Rate

- 8k-direct
- 8k-2way
- 16k-direct
- 16k-2way

Fraction of D1 Cache Size
VPR Benchmark

VPR Benchmark Results

Configuration

- 8k-direct
- 8k-2way
- 16k-direct
- 16k-2way

Miss Rate

- Base Config.
- SFDC
- Victim Cache
MCF Benchmark

MCF Benchmark Results

<table>
<thead>
<tr>
<th>Configuration</th>
<th>Miss Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>8k-direct</td>
<td>0.17</td>
</tr>
<tr>
<td>8k-2way</td>
<td>0.17</td>
</tr>
<tr>
<td>16k-direct</td>
<td>0.155</td>
</tr>
<tr>
<td>16k-2way</td>
<td>0.14</td>
</tr>
</tbody>
</table>

- Base Config.
- SFDC
- Victim Cache
Parser Benchmark Results

<table>
<thead>
<tr>
<th>Configuration</th>
<th>Base Config.</th>
<th>SFDC</th>
<th>Victim Cache</th>
</tr>
</thead>
<tbody>
<tr>
<td>8k-direct</td>
<td>0.035</td>
<td>0.025</td>
<td>0.015</td>
</tr>
<tr>
<td>8k-2way</td>
<td>0.02</td>
<td></td>
<td>0.012</td>
</tr>
<tr>
<td>16k-direct</td>
<td>0.03</td>
<td></td>
<td>0.015</td>
</tr>
<tr>
<td>16k-2way</td>
<td>0.02</td>
<td></td>
<td>0.012</td>
</tr>
</tbody>
</table>
GZIP Benchmark

GZIP Benchmark Results

Configuration: 8k-direct, 8k-2way, 16k-direct, 16k-2way

Miss Rate:
- Base Config.
- SFDC
- Victim Cache
Future Work

- Implement both SFDC and victim cache together to see if improvement is complimentary
- Reevaluate the tradeoffs between SFDC and victim cache to take access time into account
- Eviction from cache fill policy table
- Dynamically determining optimal thresholds
Hypothesis Revisited

Cache efficiency can be improved for **most benchmarks** by selectively preventing infrequently used data blocks from filling the L1 cache. Data consistently evicted from the cache before a subsequent access ought not enter if it is to evict useful data.
Conclusions

- SFDC successfully improves cache performance.
- In terms of area only, a victim cache outperforms an SFDC.
- SFDC works better than a victim cache for larger reuse distances.
- Temporal locality can be better exploited with a more advanced cache architecture.
Questions?