Instruction Cache Locking inside a Binary Rewriter

Kapil Anand
Rajeev Barua
Outline

• Introduction
• Motivation
• Cache Locking Problem
• Algorithm
  – Binary Rewriter
• Experiment and Results
• Summary
Memory Architecture of Embedded Systems

Internal SRAM should be managed intelligently
Internal SRAM Techniques

- Hardware Cache
Software Involvement

- ScratchPad Memory

Compiler decides memory allocation
Software Hints

Cache

Internal SRAM

Software

Hardware Management Policies

Program Hints

External DRAM
Existing Cache Support

- Cache Hints in EPIC Architectures
  - Reuse distance based techniques suggested by Hollander et al

- Cache Locking
  - Intel Xscale, ARM Cortex, ARM9, ARM11
    - Coprocessor-based lock instructions for locking an address in the cache
Cache Locking

• **What is Cache Locking?**
  – The facility of locking one or more lines in the cache
  – An address, once locked in the cache, always results in a hit unless an unlocking operation is carried out

**Software Influence on Cache Replacement!!**
Cache Locking

- Current Uses
  - Adaptation of cache for multi-task real time systems
  - Improves worst case estimation

- Our Objective
  - Improve average case run-time of embedded applications
Motivation

Control Flow Graph

( ABD (ACD)^4 )^{10}

Execution Trace
Motivation

4 Word Direct Mapped Cache

(ABD(ACD)^4)^10
Without Cache Locking
Without Cache Locking

B C C ... B C

#Miss: 3

CPU

CACHE

DRAM

8000
8004
8008
8012
8016
8020
With Cache Locking
With Cache Locking

B C C ....... B C

#Miss: 2

Locked

HIT!!!

CPU

B

C
Cache Misses

- \(( ABD (ACD)4 )10\)

<table>
<thead>
<tr>
<th>Node</th>
<th># of Miss without locking</th>
<th># of Miss With locking</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>B</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>C</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>D</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Total</td>
<td>22</td>
<td>13</td>
</tr>
</tbody>
</table>
Cache Locking Problem

• Goal:

“Select the memory addresses to be locked in the instruction cache to minimize the total number of instruction cache misses”
Problem Formulation

• Each Cache Set – INDEPENDENT
• Define:
  – N : Associativity of Cache
  – K : Maximum number of lines which can be locked in a set
  – L : Number of lines to be locked in the set

\[ L \leq K \leq N \]

• For each cache set, Determine
  – L: The number of lines to be locked
  – LOCKLIST: The set of addresses to be locked

\[ |\text{LOCKLIST}| = L \]
Complexity of Problem

- Exponential number of solutions
- Greedy and Iterative solution
  - Static
Solution Visualization

Set of lines mapped to the cache set

0x0000
0x0020
0x0040
0x0060
0x0080
0x00A0
0x00C0
0x00E0
.....
.....
.....
.....
.....

COST?
One less Cache line

BENEFIT?
Only one miss

Select the line with maximum Net benefit

Continue till positive Net benefit

Calculate Net benefit for each line

Final set of lines to be locked

One Cache Set

Locked

THE A. JAMES CLARK SCHOOL OF ENGINEERING

UNIVERSITY OF MARYLAND
Time Model

- \( Time(A / B) \)
  - Total time to access A given the cache lines in set B have been locked

\[
Time(x_i / \text{LOCKLIST}) = \text{HIT}_{LL}(x_i) \times T_{HIT} + \text{MISS}_{LL}(x_i) \times T_{MISS}
\]

LOCKLIST – Set of lines locked in this set
LL – Number of lines locked in this set
Time Model

\[
Time(x_i / \text{LOCKLIST}) = \text{HIT}_{LL}(x_i) * T_{HIT} \\
+ \text{MISS}_{LL}(x_i) * T_{MISS}
\]

Suppose \( F(x_i) = \text{HIT}_{LL}(x_i) + \text{MISS}_{LL}(x_i) \quad \forall \ \text{LL} \)

\[
Time(x_i / \text{LOCKLIST}) = \text{HIT}_{LL}(x_i) * T_{HIT} \\
+ (F(x_i) - \text{HIT}_{LL}(x_i)) * T_{MISS}
\]
**Benefit Model**

- **NoLockTime**(\(x_i\))

\[
Time(x_i / \text{LOCKLIST}) = HIT_{LL}(x_i) \times T_{HIT} + (F(x_i) - HIT_{LL}(x_i)) \times T_{MISS}
\]

- **LockTime**(\(x_i\))

\[
Time(x_i/(\text{LOCKLIST} \cup \{x_i\})) = T_{MISS} + (F(x_i) - 1) \times T_{HIT}
\]

\[
\text{BenLock}(x_i) = \text{NoLockTime}(x_i) - \text{LockTime}(x_i)
\]
Cost

- **NoLockTime(xj)**

\[
Time(x_j / \text{LOCKLIST}) = \text{HIT}_{LL}(x_j) \cdot T_{HIT} + (F(x_j) - \text{HIT}_{LL}(x_j)) \cdot T_{MISS}
\]

- **LockTime(xj/\(x_i\))**

\[
Time(x_j / (\text{LOCKLIST} \cup \{x_i\})) = \text{HIT}_{LL} + 1(x_j) \cdot T_{HIT} + (F(x_j) - \text{HIT}_{LL} + 1(x_j)) \cdot T_{MISS}
\]

- **CostLock(xj/\(x_i\)) = LockTime(xj/\(x_i\)) − NoLockTime(xj)**

- **Total Cost for locking \(x_i\) = \(\sum_j\) CostLock(xj / \(x_i\))**
Benefit Cost Model

• Net Benefit of locking a line $x_i$
  $\text{BenLock}(x_i) - \text{CostLock}(x_i)$
Algorithm

- **Beginning:**
  - LOCKLIST is empty

- **(LL+1)th iteration :** LOCKLIST of LL lines
  - NetBenefit for each of the virtual cache line $x_i$
  - Select virtual cache line with maximum net benefit
  - Add to LOCKLIST

- **Iteration continued until**
  - We reach the limit of maximum cache lines
  - The net benefit becomes zero
Binary Rewriting

- Trampolines for inserting lock instructions

Old Code

New Code

Minimal Layout Modifications!!

Trampoline
## Bench Marks

<table>
<thead>
<tr>
<th>Application</th>
<th>Source</th>
<th>Lines of Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>BitCnsts</td>
<td>MiBench</td>
<td>543</td>
</tr>
<tr>
<td>QuickSort</td>
<td>MiBench</td>
<td>79</td>
</tr>
<tr>
<td>Susan</td>
<td>MiBench</td>
<td>1456</td>
</tr>
<tr>
<td>Jpeg</td>
<td>MiBench</td>
<td>19804</td>
</tr>
<tr>
<td>Lame</td>
<td>MiBench</td>
<td>15959</td>
</tr>
<tr>
<td>Dijkstra</td>
<td>MiBench</td>
<td>268</td>
</tr>
<tr>
<td>StringSearch</td>
<td>MiBench</td>
<td>3072</td>
</tr>
<tr>
<td>Blowfish</td>
<td>MiBench</td>
<td>3260</td>
</tr>
<tr>
<td>Rinjdael</td>
<td>MiBench</td>
<td>1017</td>
</tr>
<tr>
<td>Sha</td>
<td>MiBench</td>
<td>207</td>
</tr>
<tr>
<td>BasicMath</td>
<td>MiBench</td>
<td>7367</td>
</tr>
<tr>
<td>FFT</td>
<td>MiBench</td>
<td>278</td>
</tr>
<tr>
<td>Lout</td>
<td>MiBench</td>
<td>30689</td>
</tr>
<tr>
<td>ADPCM</td>
<td>MediaBench</td>
<td>411</td>
</tr>
<tr>
<td>G711</td>
<td>MediaBench</td>
<td>1173</td>
</tr>
</tbody>
</table>
Results

Improvement in Cache Miss Rate with variation in Cache size
Results

Improvement in Cache Miss rate with variation in associativity

- BitCnts
- Sha
- Susan
- Dijkstra
- Jpeg
- Qsort
- G711
- ADPCM
- Blowfish
- Lame
- Search
- FFT
- BasicMath
- Lout
- Rinjdal
- AVERAGE

- 1 Way
- 2 Way
- 4 Way
- 8 Way
Results

![Graph showing Initial Miss Rate for various benchmarks. The x-axis represents benchmarks including BitCts, Sha, Susan, Dijkstra, Jpeg, G711, ADPCM, Blowfish, Qsort, Lame, Search, FFT, BasicMath, Lout, and Rindal. The y-axis represents the percentage of initial miss rate ranging from 0 to 25. The graph indicates an increasing trend in initial miss rate for Rindal.]
Results

Improvement in Execution Time with variation in Cache Size
Results

![Graph showing initial miss rate for various benchmarks.]
Results

The graph shows the initial miss rate and improvement in execution time for various benchmarks. The benchmarks include BitCn, Sha, Susan, Dijkstra, Jpeg, G711, ADPCM, Blowfish, Qsort, Lame, Search, FFT, BasicMath, Lout, and Rinjpal. The x-axis represents the benchmarks, and the y-axis shows the percentage. The blue line indicates the initial miss rate, while the green line shows the improvement in execution time.
Summary

• We present a new technique for cache locking
  – Uses Binary Rewriter

• First instruction cache locking method for average case run-time improvement

• Provides 13.5% improvement in execution time for memory access constrained benchmarks through a “pure software” method on existing commercial hardware
• MERCI !!!!!
Need for Approximation

\[ Time(x_j / (LOCKLIST \cup \{ x_i \})) = HIT_{LL + 1(x_j)} + \sum \text{HIT} + \]
\[ (F(x_j) - HIT_{LL + 1(x_j)}) \times T_{MISS} \]

Can’t be accurately determined!!!!

• Decreased hit rate can’t be calculated accurately

• Approximate value by locking a dummy (unused) virtual cache line

• Always provides conservative estimates for future hit rate – GUARANTEED IMPROVEMENT
Problem Formulation

- Each set -- INDEPENDENT
- N way set associative cache
- K – maximum number of locked lines in a set
- M – number of cache lines getting mapped to this set

Problem:
- determining L: the number of lines to be locked
- selecting L virtual cache lines out of M candidates
Objective

- Objective of previous Cache Locking
  - improve the worst case system behavior

- Our objective
Example

ABDACD ....... ABDACD
Example

ABDACD ....... ABDACD

HIT!!!
Cache Locking Interface

- Cache Locking Interface
  - Line Locking
    - Virtual Cache Line = Addr/Words-Per-Line
Solution Visualization

Determine L

Select L Lines

N-L set associative cache

\begin{align*}
0x0000 & \\
0x0020 & \\
0x0040 & \\
0x0060 & \\
0x0080 & \\
0x00A0 & \\
0x00C0 & \\
0x00E0 & \\
\ldots & \\
\ldots & \\
\ldots & \\
\ldots & \\
\ldots & \\
\end{align*}
Cost-Benefit Model

• If $x_i$ is locked
  
  – $x_i$ will observe only one compulsory miss
  **BENEFIT!!!**

  – One less cache line is available for remaining lines
  **COST!!!**
Greedy Formulation

- Objective Function

\[ Total\ Time = \sum_{i} Time(x_i / LOCKLIST) \]

- Find LOCKLIST to minimize total time
- Greedy and iterative solution for selecting one line at a time
\[ \text{Significant for 2kb} \]

\[ \text{Significant for 4kb} \]

\[ \text{Significant for 8kb, 16kb} \]
Types of Software Involvement

• Program Level Cache Control
  – Column Caching [Rudolph et al]
  – ReadMe/EvictMe Bits [Want et al]

Hardware schemes not present in current commercial architectures!!!
Why Different Problem?

- Didn’t consider benefit, cost of locking
- Lock the line which has maximum number of accesses
- Number of lines to lock?
  - Might result in large number of wrong decisions
Example

```c
for(){
    access A
} // 50 iterations

for(){
    access B
} // 20 iterations

for(){
    access C
} // 30 iterations
```

- 2 way set associative cache
- Lock A and C
- All accesses to B – misses
- Right Solution :: Don’t Lock anything
- OUR MODEL FINDS THIS SOLUTION