StatCache
A Statistical Cache Profiler
—
Low Overhead Data Locality Analysis
Erik Berg, Hagersten
Uppsala University, Sweden
{erikberg, eh}@it.uu.se

The Problem ...

What kind of profiler do you need?
Software developers Hardware developers

<table>
<thead>
<tr>
<th></th>
<th>Software developers</th>
<th>Hardware developers</th>
</tr>
</thead>
<tbody>
<tr>
<td>2-way $ better than 4-way $?</td>
<td>Yes!</td>
<td></td>
</tr>
<tr>
<td>Change loop blocking factor?</td>
<td>Yes!</td>
<td>Can’t do that anyway...</td>
</tr>
<tr>
<td>Larger BTB?</td>
<td>Yes!</td>
<td></td>
</tr>
<tr>
<td>LRU or RND?</td>
<td>Yes!</td>
<td></td>
</tr>
<tr>
<td>Fast turnaround?</td>
<td>Yes!</td>
<td></td>
</tr>
<tr>
<td>Portable software?</td>
<td>Definitely!</td>
<td>Hardly!(or...)</td>
</tr>
<tr>
<td>Identify $-trashing arrays?</td>
<td>Yes!</td>
<td>?</td>
</tr>
</tbody>
</table>

What is Accuracy Really?
- Detailed model
- Variations in the execution
- Complete execution / Short sample
- Reduced input data
- Wrong system setup
### There's Always a Trade-off

<table>
<thead>
<tr>
<th>Source code</th>
<th>Spatial locality?</th>
<th>Accuracy</th>
<th>Long runs</th>
<th>Realistic workload?</th>
<th>Accurate system setup?</th>
<th>Spatial locality?</th>
<th>HW counters</th>
</tr>
</thead>
<tbody>
<tr>
<td>No</td>
<td>No</td>
<td>Very High</td>
<td>No way!</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>All - but inflexible</td>
</tr>
<tr>
<td>Well…</td>
<td>High</td>
<td>High</td>
<td>Hard</td>
<td>50 days...</td>
<td>Maybe</td>
<td>No/Yes</td>
<td>Hard</td>
</tr>
<tr>
<td>Yes</td>
<td>5 days...</td>
<td>Med</td>
<td>Yes, but painful</td>
<td>Yes!</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Yes</td>
<td>Yes</td>
<td>Low</td>
<td>Yes!</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Yes</td>
<td>Yes</td>
<td>All - but inflexible</td>
<td>Yes</td>
<td>No</td>
<td>Yes!</td>
<td>No way!</td>
<td>No</td>
</tr>
</tbody>
</table>

### Outline

- The problem
  - Statistical cache model
  - Implementation
  - StatCache vs. simulation
  - Using StatCache

### StatCache overview

- A probabilistic cache model instead of cache simulation
- Based on sparse sampling (i.e., fast)
- Single run – arbitrary cache size & cache line size

### Only 20% overhead

- 171.swim and 188.ammp

![Graph showing miss ratio vs. cache size](chart.png)
Traditional Model

```
set  A,%r1
ld    [%r1],%r0
st     %r0,[%r1+8]
add  %r1,1,%r1
ld    [%r1+16],%r0
add  %r0,%r5,%r5
st     %r5,[%r1+8]
```

Memory ref:
1: read A
2: write B
3: read C
4: write B

---

StatCache Model

```
set  A,%r1
ld    [%r1],%r0
st     %r0,[%r1+8]
add  %r1,1,%r1
ld    [%r1+16],%r0
add  %r0,%r5,%r5
st     %r5,[%r1+8]
```

Memory ref:
1: read A
2: write B
3: read C
4: write B

---

Run-Time Statistics: Reuse Distance

Sline: A B E B D B A F D .. .

Memory ref:

1 2 3 4 5 6 7 8 9 10 11 12

(Note: This is not "stack distance")

---

Reuse Distance Histogram

(Note: This is not "stack distance")
Run-Time Statistics: Sampling

- Sampling possible
- Efficient implementation possible:
  - Solaris watchpoint support
  - Performance counter (DC_rd)

- Cannot directly read out miss ratios → post-processing “math”

Run-Time Statistics: Sampled Histogram

Run-Time Statistics: Summary

- Easy to measure
- Sampling possible
- Efficient implementation possible:
  - Solaris watchpoint support
  - Performance counter (DC_rd)

- Cannot directly read out miss ratios → post-processing “math”

StatCache Model

- CPU
- Run-time statistics
- Probabilistic cache model
- Memory ref: 1:read A
  2:write B
  3:read C
  4:write B
  [...]

StatCache runtime and post-processing system

Code:
```c
set A, %r1
ld [%r1], %r0
st %r0, [%r1+8]
add %r1, 1, %r1
ld [%r1+16], %r0
add %r0, %r5, %r5
st %r5, [%r1+8]
[...]
```
Probabilistic Cache Model

- Fully-associative cache
- Random-replacement policy (working on direct mapped)

Find the probability that a load or store instruction causes a cache miss without knowledge of exact cache content.

Miss probability function

\[ f(n) = 1 - (1 - 1/L)^n \]

- \( n \) = number of cache misses since last touched
- \( L \) = number of cache lines in the cache

Miss Ratio Formula

\[ R \cdot N = h(1)f(R) + h(2)f(2R) + h(3)f(3R) + ... \]

Solve for \( R \) to get miss ratio. (With numerical method)
StatCache Summary

Collect run-time statistics by sampling

Estimate miss ratios

Present results for arbitrary cache size

Implementation

We need to do three things

1. Select samples: Overflow trap from perf.counter (DC_rd)
2. Detect reuse: Solaris watchpoint support:
   \texttt{write("/proc/self/ctl", addr,$linesize)}
3. Measure reuse distance: Using the perf.counter again (DC_rd)

StatCache vs. simulation
Performance

<table>
<thead>
<tr>
<th>Benchmark SPEC</th>
<th>Mem.rd&amp;wr (billions)</th>
<th>Run time (min, sec)</th>
<th>Overhead (min, sec)</th>
<th>Overhead (percent)</th>
</tr>
</thead>
<tbody>
<tr>
<td>wupwise</td>
<td>171</td>
<td>12m 51s</td>
<td>2m 17s</td>
<td>18</td>
</tr>
<tr>
<td>swim</td>
<td>120</td>
<td>24m 32s</td>
<td>2m 35s</td>
<td>11</td>
</tr>
<tr>
<td>mgrid</td>
<td>328</td>
<td>33m 49s</td>
<td>4m 43s</td>
<td>14</td>
</tr>
<tr>
<td>applu</td>
<td>194</td>
<td>25m 1s</td>
<td>1m 39s</td>
<td>6.6</td>
</tr>
<tr>
<td>art</td>
<td>38</td>
<td>3m 43s</td>
<td>3m 15s</td>
<td>87</td>
</tr>
<tr>
<td>equake</td>
<td>121</td>
<td>16m 42s</td>
<td>2m 29s</td>
<td>21</td>
</tr>
<tr>
<td>ammp</td>
<td>246</td>
<td>15m 12s</td>
<td>2m 54s</td>
<td>19</td>
</tr>
<tr>
<td>apsi</td>
<td>237</td>
<td>16m 30s</td>
<td>3m 45s</td>
<td>23</td>
</tr>
</tbody>
</table>

Using StatCache

- Evaluate and compare optimizations
- Measure data locality:
  - spatial and temporal
- Identify poor data structure layout and/or access patterns
- Locate code with poor cache behavior
- Workload characterization

Blocked Matrix Multiplication: X = Y·Z

- Total blocksize = 60
- Total blocksize = 20
- Y blocksize = 20
- Y blocksize = 60
Varying the cache-line size

Spatial Locality Example

**swim**: Spatial Use $\approx 1.0$
Well-behaved access pattern gives good spatial locality.

**ammp**: Spatial Use $\approx 0.0$
Poor spatial locality caused by structure field-by-field initialization.

for ( i = 0; ... )
a[i].field = ... ;

Spatial Optimization of Equake

Unoptimized memory layout:

Optimized memory layout:

Equake Spatial Locality
Conclusions

• Fast:
  - only 20% overhead and
  - low sampling rate
• Arbitrary cache size/line size
• No code instrumentation
• Code and data structure mapping possible

Current limitations
• Fully associative, random repl.

Work in Progress/Plans

• Direct mapped caches
• Run any SPARC executable
• Use DWARF debugging
  • Identify data structures
  • Source code mapping
• Use for optimal prefetching
• MP

Why is speed so important?

![Miss ratio vs Cache size graph]

- AMMP (reference input)
- AMMP (reduced input)
Cache Model: Random vs. LRU

Cache Hit Function

• Fully-associative cache, random replacement
• L cache lines:

Probability that data remains in cache

\[ (1 - \frac{1}{L})^n \]

\[ n = \# \text{ cache misses since last touched} \]

Probabilistic Cache Model

\[ E(\text{Total misses}) = p_1 + p_2 + \ldots + p_N \approx g(1) + g(2) + \ldots + g(N) \]

where \( N = \# \text{memory accesses} \)
Great! We have a relationship between reuse distances and miss probabilities.

But, too many unknowns.

$$p_1 + p_2 + \ldots + p_N \approx g(1) + g(2) + \ldots + g(N)$$

where

$$g(i) = f(p_{i-A(i)} + p_{i-A(i)+1} + \ldots + p_{i-1})$$

Solution: Assume a constant miss ratio $R$ [for a region of accesses]

This means that:

$$\frac{(p_i + p_{i+1} + \ldots + p_{j-2} + p_{j-1})}{j-i} \approx R$$

where $0 < i < j \leq N$

Using the histogram info...

$A(i)$ is the reuse distance of ref. $i$

Histogram over $A(i)$ values, where $i = \{1..N\}$:

$$h(j) = \text{number of references that have reuse distance } j,$$
that is, satisfy $A(i) = j$
Run-Time Statistics: Sampled Histogram

Probabilistic Cache Model

Probabilistic Cache Model

Probabilistic Cache Model
Miss rates varying over time...

- Split time in time slots
- Generate histogram for each time slot at run-time
- Calculate the miss ratio for each time slot:
  \[ h(1)f(R) + h(2)f(2R) + h(3)f(3R) + \ldots = RN \]
- Take average miss ratio of all time slots

StatCache Model

Code:
```
set   A,%r1
ld    [%r1],%r0
st     %r0,[%r1+8]
add  %r1,1,%r1
ld    [%r1+16],%r0
add  %r0,%r5,%r5
st     %r5,[%r1+8]
[...]
```

Memory ref:
1: read A
2: write B
3: read C
4: write B
[...]

Simulation vs. StatCache

Simulation vs. StatCache

Graphs showing miss ratio (% vs cache size (bytes)) for different simulations and StatCache models.