Cache Misses Prediction Using Stack Distances

Calin Cascaval and David A. Padua
Department of computer science
University of Illinois at Urbana-Champaign
cascaval, padua@cs.uiuc.edu
Outline

- Stack Distances as a Metric for Locality
- Run-time Instrumentation
- Compile-time Analysis
- Future Work
Stack Algorithms

\[ S_{t-1}(1) \]
\[ S_{t-1}(2) \]
\[ \ldots \]
\[ S_{t-1}(\Delta-1) \]
\[ S_{t-1}(\Delta) \]
\[ S_{t-1}(\Delta+1) \]
\[ \ldots \]
\[ S_{t-1}(\gamma_{t-1}) \]

\[ S_t(1) \]
\[ S_t(2) \]
\[ \ldots \]
\[ S_t(\Delta-1) \]
\[ S_t(\Delta) \]
\[ S_t(\Delta+1) \]
\[ \ldots \]
\[ S_t(\gamma_t) \]

\[ x_t \]
Stack Distances As Cache Misses

compute the number of cache hits and misses as follows:

\[ \text{hits}(C) = \sum_{\Delta=1}^{C} s(\Delta) \]

\[ \text{misses}(C) = \sum_{\Delta=C+1}^{\text{Inf}} s(\Delta) \]
Metric Validation

- Stack algorithm implemented as a library
- Polaris instrumented codes from the SPEC95 and Perfect Club benchmarks
- Measured actual number of cache misses using the hardware counters on the R10K processors (for both L1 and L2 caches)
Experiments Serial

Prediction Accuracy

- < 75%: 56%
- 75%-90%: 1%
- 90%-100%: 20%
- 100%-110%: 20%
- > 110%: 3%

August 27, 1998
Univ. of Illinois at Urbana-Champaign
Ocean

August 27, 1998

Univ. of Illinois at Urbana-Champaign
Solution Advantages

- Accurate in most cases.
- One pass through the trace estimates misses for any cache size.
- Architecture independent (except for the cache line size).
- Same model applicable to independent loops as well as entire programs.
- Easily applicable to parallel programs.
Run-time Solution Advantages

- Works in all cases
Run-time Solution Disadvantages

- It is run-time, therefore consumes CPU time
- Cannot easily identify references with bad locality
- Needs separate runs for different cache sizes/processors
- Assumes a fully associative cache (to reduce overhead)
Compile-time Solution

- Integrated within the Polaris parallelizing compiler
- Algorithm based on data dependence distance vectors to compute the stack distances for loops
- Computes symbolic expressions (based on loop bounds) for stack distances and number of references
Compile-time Solution (cont.)

- Preserves the advantages given by the run-time solution, adding
  - misbehaving reference identification
  - array size knowledge may improve accuracy
  - flexibility and locality information readily available in the compiler
- May need a run-time pass for unknown loop bounds or data depending on the input
Example

do j = 1, n
    do i = 1, n
        a(i,j) = b(i,1) + b(i+1,1)

<table>
<thead>
<tr>
<th>Distance</th>
<th>References</th>
</tr>
</thead>
<tbody>
<tr>
<td>$</td>
<td>\delta R^1_i + d - s</td>
</tr>
<tr>
<td>Inf</td>
<td>3N-(N-1)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Distance</th>
<th>References</th>
</tr>
</thead>
<tbody>
<tr>
<td>$</td>
<td>\delta R^1_j + d - s</td>
</tr>
<tr>
<td>$</td>
<td>\delta R^1_j + d - s</td>
</tr>
<tr>
<td>Inf</td>
<td>$N^2+N+1$</td>
</tr>
</tbody>
</table>
Future work

- Integrating the compile-time solution with the run-time instrumentation
- Solve the false sharing integration within the compiler approach
- Address some of the limitations: cache associativity, multi-word cache lines