Defensive Loop Tiling for Shared Cache

Bin Bao
Adobe Systems

Chen Ding
University of Rochester
“Unlike a bird, which can learn to fly better and better, existing programs are sort of dumb---the one millionth run of a program is typically not a bit better than the first-time run.” --- Professor Xipeng Shen @ W&M
Peer Interaction

- Interfering
  - Limited resources
- Collaborative
  - Parallel tasks
- Peers: threads, tasks, and independent programs
Co-Run Program Optimization

- Existing shared-cache optimization
  - Cache partitioning
- Job scheduling
- Task throttling
- Compiler optimization?
Loop Tiling --- A Matrix Multiplication Example

\[
\begin{align*}
&\text{for}(i = 0; i < N; i = i + 1) \\
&\text{for}(j = 0; j < N; j = j + 1) \\
&\text{for}(k = 0; k < N; k = k + 1) \\
&C[i][j] = beta \times C[i][j] + alpha \times A[i][k] \times B[k][j];
\end{align*}
\]

(a) Original code

\[
\begin{align*}
&\text{for}(jj = 0; jj < N; jj = jj + B_j) \\
&\text{for}(kk = 0; kk < N; kk = kk + B_k) \\
&\text{for}(i = 0; i < N; i = i + 1) \\
&\text{for}(j = jj; j < \min(jj + B_j, N); j = j + 1) \\
&\text{for}(k = kk; k < \min(kk + B_k, N); k = k + 1) \\
&C[i][j] = beta \times C[i][j] + alpha \times A[i][k] \times B[k][j];
\end{align*}
\]

(b) Tiled code
Tiling Strategy for Shared Cache

1. Tile for whole shared cache
2. Tile for part of shared cache
3. Tile for private cache only

???
Inclusion Victim Misses

- Inclusive cache
  - E.g. L3 cache in Intel Nehalem processor
- Inclusive victim [Jaleel et al. MICRO’10]
- A toy example: L1 cache size 2; L2 cache size 8

misses:    c             v             v
prog. 1:   a a a a a a a a a a a a a a a a a a a a ... 
prog. 2:   p q u v w x y z p q u v w x y z p ...
Matrix Multiplication Results on a Cache Simulator

- 2 cores
- Private 256KB L1 cache
- Shared 2MB L2 cache
- Matmul and streaming
Inclusion Victim Modeling

- Data usage
- Reused data, active period
- Cache interference
- Survival window

Inclusion Victim Modeling

\[ iv(p_1) = \frac{ap(p_1)}{sw(p_1 + p_2)} \times reuse(p_1) \]

<table>
<thead>
<tr>
<th>Misses</th>
<th>Program 1</th>
<th>Program 2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>c</td>
<td>v</td>
</tr>
<tr>
<td>a a a a a a a a a a a a a a a a a a a a a a ...</td>
<td>p q u v w x y z p q u v w x y z p ...</td>
<td></td>
</tr>
</tbody>
</table>

\[ ap(p_1) = 2 \times (255.9 KB) \]

\[ sw(p_1 + p_2) = 2 \times (255.9 KB) \]

\[ reuse(p_1) = \frac{5MB}{255.9 KB} \times 100\% = 19.66\% \]
Implementation in Open64 Compiler

- A cache cost function
- Example: matrix multiplication
  - Footprint
    \[ F_i = 8 \times (N \times B_k + B_j \times B_k + N \times B_j) \]
    \[ F_j = 8 \times (B_k + B_j \times B_k + B_j) \]
  - Reuse
    \[ reuse_j = F_j - (F_i - F_j)/N \]
Original cache miss equation

\[ CM_j = \frac{F_i}{N} + (\alpha \cdot \frac{R_i}{ecsz} + \beta \cdot \frac{|R_i - ecsz|^+}{ecsz}) \cdot reuse_j \]

Cache misses caused by inclusion victim

\[ IV_j = \frac{F_i}{scsz/\gamma} \cdot reuse_j \]

\( \gamma \) is the defensiveness parameter
Experimental Results

- PLUTO benchmarks
- Pin-based cache simulator
  - 256KB private L1, 2MB shared L2
- Intel Nehalem processor
  - private 32KB L1 and 256KB L2, shared 8MB L3
- Co-run peers
  - STREAM benchmark, in addition to PLUTO
Effect on private cache miss

Baseline: default tiling on solo-run

4 defensiveness values
Real Machine Performance

- Defensiveness parameter $\gamma = 4$

![Graph showing speedup of defensive tiling over Open64 default tiling as measured on Intel Nehalem. Each benchmark co-runs with 1 to 3 STREAM benchmarks. The four graphs show defensive tiling with $\gamma = 1$, 2, 4, 8.]

Symmetric Co-runs
We have tested each PLUTO benchmark running with one, two, or three of its own replicas. Figure 6 shows the result for the solo- and co-run tests when the defensiveness level is set to 4. The baseline is the default tiling in the solo run and in 2 to 4 symmetric co-runs. The first bar in each group shows the same speedup as those in Figure 5c.

All tests show improvements, although most are small and lower than the solo-run improvement. The results for other defensiveness levels are similar, thus we omit them for brevity.

Defensive tiling seems not effective since in all programs the lead over the default tiling is narrowed, often significantly. For explanation we need to examine the friendliness as defined and discussed in Section 2.4. The tiled programs have excellent locality, so they are among the friendly.
Comparison with Cache Oblivious Algorithm

- Recursive version matrix multiplication [Qing et al. PLDI 2000]
Summary and Future Work

- Defensive tiling
  - Self-aware -> Peer-aware
  - Reduce interference
- Currently investigating
  - Co-run with other programs, add adaptivity
  - Use the shared cache model to direct compiler optimization