Formulating The Problem of
Compiler-Assisted Cache Replacement

Hongbo Yang
Agenda

- Background: Memory hierarchy, ISA with cache hints
- Problem definition: How should compiler give cache hint to minimize cache miss rate?
- Case Study for Relationship Between Reference Window and Cache Misses
- Problem Formulation
- Performance Result
- Summary
Cache Size Becoming Larger and Larger

1993: Pentium
- 8 KByte I-cache
- 8 KByte D-cache

1997: Pentium-II
- 16 KByte L1I, 16 KByte L1D
- 512 KByte off-die L2

2002: Itanium-2
- Level 1: 16K KByte I-cache, 16 KByte D-cache
- Level 2: 256 KB
- Level 3: integrated 3 MB or 1.5 MB
Memory Hierarchy

- It is critical to make data needed close to the CPU to sustain CPU performance.
- Cache pollution is a severe problem that prevents us from doing so.
Cache Pollution

• Access sequence: a, b, c, a

  
<table>
<thead>
<tr>
<th>a</th>
<th>c</th>
<th>c</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>b</td>
<td>a</td>
</tr>
</tbody>
</table>

  capacity miss!

• Access sequence: a, b, c, a, b, c

  
<table>
<thead>
<tr>
<th>a</th>
<th>c</th>
<th>c</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>b</td>
<td>a</td>
<td>a</td>
</tr>
<tr>
<td>b</td>
<td>c</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

  capacity miss! capacity miss! capacity miss!
Using “nt” (Non-Temporal) Cache Hint

- Access sequence: a, b, c, a
  Access type: normal, nt, nt, normal

- Access sequence: a, b, c, a, b, c
  Access type: normal, nt, nt, normal, nt, nt

- Better than the previous slide!
Problem Statement

Problem: For a loop nest $S$, determine array references in the loop body that should give "nt" hint thus number of cache misses of loop execution is minimized.

Notes:

- We consider array references only since they are cache-hog
- We differentiate data reference and data access here: a data reference appears lexically in program, a data access is an instantiation of some data reference at runtime
Case Study

DO 110 J = 1, 128, 4
   DO 110 K = 1, 64
      DO 110 I = 1, 256
         C(I,K) = C(I,K) + A(I,J) * B(J,K) + A(I,J+1) * B(J+1,K) +
                  A(I,J+2) * B(J+2,K) + A(I,J+3) * B(J+3,K)
   110 CONTINUE

• The four array references of A doesn't overlap
• We measured the cache occupancy of A(I,J), which means the number of cache blocks that holds data accessed by A(I,J)
Relationship Between Cache Occupancy and Miss Rate
Observations and Analysis

- **Observations**
  - Cache miss rate is inversely proportional to cache occupancy.
  - Cache miss rate of reference $A(I,J)$ is 0 when cache occupancy of this reference is 256.

- **Why?**
  - Let’s analyze data reuse of this loop nest!
Data Reuse Analysis: Notations

• Access matrix

Subscripts of $A(I,J)$ can be represented as

$$\begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix} \begin{pmatrix} J \\ K \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

DO 110 J = 1, 128, 4
   DO 110 K = 1, 64
      DO 110 I = 1, 256
         C(I,K) = C(I,K) + A(I,J) * B(J,K) + A(I,J+1) * B(J+1,K) +
            A(I,J+2) * B(J+2,K) + A(I,J+3) * B(J+3,K)
      110 CONTINUE

110 CONTINUE
Data Reuse Analysis: Notations

• Reuse vector

For a reference \( H \cdot \vec{i} + \vec{c} \), data accessed at iteration \( \vec{i} \)
will be reused at iteration \( \vec{j} \) only if \( H \cdot (\vec{j} - \vec{i}) = \vec{0} \)
Data Reuse of A(I,J)

• Reuse vector

For $A(I,J)$ with subscripts

$$
\begin{pmatrix}
0 & 0 & 1 \\
1 & 0 & 0 \\
1 & 0 & 0
\end{pmatrix}
\begin{pmatrix}
J \\
K \\
I
\end{pmatrix}
+ \begin{pmatrix}
0 \\
0 \\
0
\end{pmatrix},
$$

$$
\begin{pmatrix}
0 & 0 & 1 \\
1 & 0 & 0 \\
1 & 0 & 0
\end{pmatrix}
\begin{pmatrix}
0 \\
1 \\
0
\end{pmatrix}
= \begin{pmatrix}
0 \\
0 \\
0
\end{pmatrix},
$$

thus reuse vector is

$$
\begin{pmatrix}
0 \\
1 \\
0
\end{pmatrix}
$$

DO 110 J = 1, 128, 4

DO 110 K = 1, 64

DO 110 I = 1, 256


110 CONTINUE
Data Reuse Analysis: Notations

• Reference Window

From iteration \( i \) to iteration \( j \) where \( H \cdot (j - i) = 0 \)

is satisfied, how many array elements are accessed by the same array reference?

DO 110 J = 1, 128, 4
  DO 110 K = 1, 64
    DO 110 I = 1, 256
      C(I,K) = C(I,K) + A(I,J) * B(J,K) + A(I,J+1) * B(J+1,K) +
               A(I,J+2) * B(J+2,K) + A(I,J+3) * B(J+3,K)
  110 CONTINUE
Reference Window of A(I,J)

- From iteration \[
\begin{pmatrix}
  j \\
  k \\
  i
\end{pmatrix}
\]
  to
\[
\begin{pmatrix}
  j \\
  k+1 \\
  i
\end{pmatrix},
\]
  256 elements are accessed by A(I,J).

- Formula is given in “Strategies for cache and local memory management by global programming transformation” by Gannon, Jalby and Gallivan, JPDC Vol 5, No 5, Oct 1988

   DO 110 J = 1, 128, 4
   DO 110 K = 1, 64
   DO 110 I = 1, 256
      C(I,K) = C(I,K) + A(I,J) * B(J,K) + A(I,J+1) * B(J+1,K) +
              A (I,J+2) * B(J+2,K) + A(I,J+3) * B(J+3,K)
   110 CONTINUE
Problem formulation

Maximize

$$\sum_{i=1}^{m} b_i$$

Within the constraint:

$$\sum_{i=1}^{m} (\text{Ref}_\text{Win} (i) \cdot b_i) < C$$

<table>
<thead>
<tr>
<th>C</th>
<th>effective cache size</th>
</tr>
</thead>
<tbody>
<tr>
<td>m</td>
<td>number of array references</td>
</tr>
<tr>
<td>b_i</td>
<td>1 if it is normal, 0 if it is non-temporal</td>
</tr>
</tbody>
</table>

This is a knapsack problem!
Problem Formulation for MXM Example

Maximize
\[ b_1 + b_2 + b_3 + b_4 + b_5 + b_6 + b_7 + b_8 + b_9 \]

Within the constraint:
\[ 128K \cdot b_1 + 2K \cdot b_2 + 1 \cdot b_3 + 2K \cdot b_4 + 1 \cdot b_5 + 2K \cdot b_6 + 1 \cdot b_7 + 2K \cdot b_8 + 1 \cdot b_9 < 8K \]

DO 110 J = 1, 128, 4
   DO 110 K = 1, 64
      DO 110 I = 1, 256
         C(I,K) = C(I,K) + A(I,J) * B(J,K) + A(I,J+1) * B(J+1,K) +
                  A(I,J+2) * B(J+2,K) + A(I,J+3) * B(J+3,K)
   110 CONTINUE
How to Handle Group-Reuse

DO 10  T = 1, IT
  DO 10  I = 1, M
    DO 10  J = 1, N
      L(I, J) = (A(I,J-1) + A(I,J+1) + A(I-1,J) + A(I+1,J)) / 4
    10  CONTINUE

Reuse Graph. Each node is a reference. Arc is a possible reuse. The vector adjacent to each arc is reuse vector.
Pruning the Reuse Graph

DO 10  T = 1, IT
    DO 10  I = 1, M
        DO 10  J = 1, N
            L(I, J) = (A(I,J-1) + A(I,J+1) + A(I-1,J) + A(I+1,J)) / 4
        10  CONTINUE
How to Handle Group-Reuse

Maximize \( b_1 + b_2 \)

Subject to: \( (N - 1) \cdot b_1 + 2 \cdot b_2 < C \)

The number adjacent to each arc is size of the reference window.
Results: MXM

Conventional cache vs Cache with Hints

Cache miss rate (%)

4K 8K 16K 32K
L1-D cache size

Conventional Cache
Evict-Me cache
Results: VPENTTA

![Graph showing cache miss rate comparison between conventional cache and cache with NT hint]

- **Cache Miss Rate (%):**
  - **4K:**
    - Conventional cache: [Value]
    - Cache with NT hint: [Value]
  - **8K:**
    - Conventional cache: [Value]
    - Cache with NT hint: [Value]
  - **16K:**
    - Conventional cache: [Value]
    - Cache with NT hint: [Value]
  - **32K:**
    - Conventional cache: [Value]
    - Cache with NT hint: [Value]
Results: TOMCATV

![Graph showing cache miss rate(%)]

- **Conventional cache**
- **Cache with nt hint**

The graph illustrates the cache miss rate for different cache sizes (4K, 8K, 16K, 32K) and two cache configurations: conventional cache and cache with nt hint. The cache miss rate decreases as the cache size increases for both configurations.
Analysis of Degradation

Avg Reference Times of Regular Cache Lines

Avg Reference Times of nt Cache Lines
Summary

• Motivation
  - Managing limited cache space judiciously is critical for program performance.
  - Hardware-only LRU replacement algorithm is inadequate for many cases.
• ISA Solution: Cache hints given by compiler
• Problem: What memory reference should be given “nt” hint?
• Solution: Knapsack problem formulation