Data Locality

Why Locality?

- Memory accesses are expensive

Loop Transformations to Improve Reuse

Key Insight

- Memory order ➔ memory locality

2. Compute the cost in cache lines accessed

3. Rank the loops based on their loop cost

For each loop l in a nest, consider l innermost loops

To calculate temporal and spatial reuse

Reuse on loop i

<table>
<thead>
<tr>
<th>Reuse on loop i</th>
<th>Reuse on loop j</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
</tr>
</tbody>
</table>

What reuse occurs in this loop nest?

- Group-reuse
  - caused by multiple references
  - caused by some reference

- Spatial locality
  - reuse of adjacent locations

- Temporal locality
  - reuse of a specific location

- Locality of reference
  - reuse of reference

Why locality affects
- Resizing registers, cache lines, TLB, etc.
  - except higher levels of memory hierarchy
Selecting a Loop Permutation

Cost of reference group for loop $k$

1. Select representative from reference group
2. Find cost (in cache lines) with $k$ innermost
3. Multiply by trip counts of outer loops
4. Sum of costs for reference groups

Matrix multiplication example

<table>
<thead>
<tr>
<th>RefGroups</th>
<th>J</th>
<th>K</th>
<th>I</th>
</tr>
</thead>
<tbody>
<tr>
<td>B(1)*</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A(1)*</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C(1)*</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B(2)*</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A(2)*</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C(2)*</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Matrix multiplication (execute time in seconds)

300 x 300

150 x 150

Sun Sparc
Intel i860
IBM RS/6k

Matrix multiplication (execute time in seconds)

300 x 300

150 x 150

Sun Sparc
Intel i860
IBM RS/6k
Dependence Analysis

Question

Do two references never/maybe/always access the same memory location?

Benefits

- improves alias analysis
- enables loop transformations

Motivation

- classic optimizations
- instruction scheduling
- data locality (register/cache reuse)
- vectorization, parallelization

Obstacles

- array references
- pointer references

A loop-independent dependence exists regardless of the loop structure. The source and sink of the dependence occur on the same loop iteration.

A loop-carried dependence is induced by the iterations of a loop. The source and sink of the dependence occur on different loop iterations.

Loop-carried dependences can inhibit parallelization and loop transformations
Dependence Testing

Given

\[
\begin{align*}
\text{do } & i_1 = L_1, U_1 \\
\vdots & \text{do } i_n = L_n, U_n \\
S_1 & = A(f_1(i_1, \ldots, i_n), \ldots, f_m(i_1, \ldots, i_n)) = \ldots \\
S_2 & = A(g_1(i_1, \ldots, i_n), \ldots, g_m(i_1, \ldots, i_n))
\end{align*}
\]

A dependence between statement $S_1$ and $S_2$, denoted $S_1 \delta S_2$, indicates that $S_1$, the source, must be executed before $S_2$, the sink on some iteration of the nest.

Let $\alpha$ & $\beta$ be a vector of $n$ integers within the ranges of the lower and upper bounds of the $n$ loops.

Does $\exists \alpha \leq \beta$, s.t.

\[
f_k(\alpha) = g_k(\beta) \quad \forall k, 1 \leq k \leq m
\]

Iteration Space

\[
\begin{align*}
\text{do } & I = 1, 5 \\
\text{do } & J = I, 6 \\
\text{enddo} \\
\text{enddo}
\end{align*}
\]

\[
\begin{array}{|c|c|c|c|c|c|}
\hline
1 & 2 & 3 & 4 & 5 & 6 \\
\hline
\end{array}
\]

\[
\begin{align*}
1 \leq I \leq 5 \\
I \leq J \leq 6
\end{align*}
\]

- lexicographical (sequential) order for the above iteration space is

\[
(1,1), (1,2), \ldots, (1,6) \\
(2,1), (2,2), \ldots, (2,6) \\
\vdots \\
(5,1), (5,2)
\]

- given $I = (i_1, \ldots i_n)$ and $I' = (i_1', \ldots i_n')$,

\[
I < I' \iff (i_1, i_2, \ldots i_k) = (i_1', i_2', \ldots i_k') \quad \& \quad i_{k+1} < i_{k+1}'
\]
**Distance Vectors**

\[
\text{do } I = 1, N \\
\text{do } J = 1, N \\
S_1 \ A(I,J) = A(I,J-1) \\
\text{enddo} \\
\text{enddo}
\]

\[
\text{do } I = 1, N \\
\text{do } J = 1, N \\
S_2 \ A(I,J) = A(I-1,J-1) \\
S_3 \ B(I,J) = B(I-1,J+1) \\
\text{enddo} \\
\text{enddo}
\]

Distance Vector = number of iterations between accesses to the same location
distance vector

\[S_1 \delta S_1 \]
\[S_2 \delta S_2 \]
\[S_3 \delta S_3 \]

---

**Loop Interchange**

\[
\text{do } I = 1, N \\
\text{do } J = 1, N \\
S_1 \ A(I,J) = A(I,J-1) \\
S_2 \ B(I,J) = B(I-1,J-1) \\
\text{enddo} \\
\text{enddo}
\]

\[
\text{do } J = 1, N \\
\text{do } I = 1, N \\
S_1 \ A(I,J) = A(I,J-1) \\
S_2 \ B(I,J) = B(I-1,J-1) \\
\text{enddo} \\
\text{enddo}
\]

\[\Rightarrow \text{loop interchange } \Rightarrow \]

\[\Rightarrow \text{loop interchange is safe iff} \]

- it does not create a lexicographically negative direction vector \((1,-1) \rightarrow (-1,1)\)

\[\Rightarrow \text{Benefits} \]
- may expose parallel loops, incr granularity
- reordering iterations may improve reuse