Automatic Loop Interchange

John R. Allen, Ken Kennedy
Rice University

Presented by: zihengw@eecs.umich.edu
Outline

• Introduction
• Data Dependence
• Reorder Transformation
• Dependence and Vectorization
• Valid Loop Interchange
• Level Movement with Interchange
• Loop Interchange and Vectorization
Introduction

• Loop interchange

Example: Product of Matrix A and B

\[
\begin{align*}
\text{DO 100 } & \text{J=1, 128} & \text{DO 100 } & \text{K=1, 128} \\
\text{DO 100 } & \text{I=1, 128} & \text{DO 100 } & \text{J=1, 128} \\
\text{DO 100 } & \text{K=1, 128} & \text{DO 100 } & \text{I=1, 128} \\
100 \text{ CONTINUE} & 100 \text{ CONTINUE} \\
\end{align*}
\]

\[
\begin{align*}
\text{DO 100 } & \text{K=1, 128} \\
\end{align*}
\]

• useful but not straightforward
Data Dependence

• Basic Definitions

• Data Dependence with Loops
  – Iteration vector
  – Direction vector
  – Two types of dependence

• Data Dependence with Array
II. Data Dependence

Basic Definitions

• Data Dependence and Control Dependence

• Definitions of Data Dependence

  – True dependence $S_1 \delta S_2$
    
    $S_1$: $X= \quad S_2$: $= X$

  – Antidependence $S_1 \delta^{-1} S_2$
    
    $S_1$: $= X \quad S_2$: $X= $

  – Output dependence $S_1 \delta^0 S_2$
    
    $S_1$: $X= \quad S_2$: $X= $
II. Data Dependence

Data Dependence with Loops

DO 100 I=1, 2
    DO 100 J=3, 4
        S1
    100 CONTINUE

• Iteration Vector: (2,3), etc
• Iteration Space: \{ (1,3), (1,4), (2,3), (2,4) \}
• Direction Vector
  Eg1: i=(1,2), j=(2,1), then D(i,j) = (< , >)
  Eg2: i=(2,1), j=(2,2), then D(i,j) = (= , <)
II. Data Dependence

Data Dependence with Loops (cont.)

• Loop Independent Dependence
  – S1 references location M on iteration $i$ and S2 references M on iteration $j$.
  – Every component of $D(i, j)$ is $=$ and
  – S2 lexically follows S1

• Example

  DO 100 I=1, 100
  S1 A(I)=
  S2 =A(I)
  100 CONTINUE

  for M=A(1), i=(1) and j=(1), D(i, j) = (=)
II. Data Dependence

Data Dependence with Loops (cont.)

• Loop Carried Dependence
  – S1 references location M on iteration i and S2 references M on iteration j and
  – \(D(i, j)\) contains an entry which is not =, the left most one of which is <.

• Example:

\[
\begin{align*}
&\text{DO 100 I=1, 100} \\
&S1 \quad \text{A(I)=} \\
&S2 \quad =\text{A(I-1)} \\
&\text{100 CONTINUE}
\end{align*}
\]

for M=A(1), i=(1) and j=(2), D(i, j) = (<)
II. Data Dependence

Data Dependence with Loops (cont.)

• Level of dependence
  – The level of a loop carried dependence is the index of the leftmost non = in its associated direction vector. \((S1 \delta_k S2)\)
  – Example:

    DO 100 I=1, 100
    DO 100 J=1, 100
    S1   A(I, J) = A(I-1, J)
    100 CONTINUE

    Direction vector: \((<, =)\) --> level 1 independence.
II. Data Dependence

Data Dependence with Array

• In general, it’s difficult to determine the existence of a dependence due to an array.
• A practical but not exact test for array dependence [Kenn 80, Alle 83, Bane 76]
  – $gcd$ test and Banerjee inequality
  – $S1 \delta_k S2$ only if both of the tests are passed
Reordering Transformation

• Property of the Reordering Transformations
  – They do not change the code execute
  – They do not destroy a dependence, but they would reverse it.

• Loop Transformation is Reordering Transformation

• Question: when will loop transformation reverse a dependence?
Reordering Transformation (cont.)

• Theorem 2:
  – If S2 has a loop independent dependence upon S1, it will be preserved by a common loop transformation that does not change their execute order within the loops.

• Theorem 3:
  – Any reordering transformation which does not alter loops 1 through k preserves any level k dependence.
Dependence and Vectorization

• Vectorization = Executed together
• Vectorization scheme (*codegen* in [Kenn 80])
  – First attempt to generate code in parallel at the outermost level.
  – If dependence prevents that, run the outer loop sequentially (thereby satisfying the dependences carried by that loop) and try again one level deeper, ignoring dependences carried by the outer loop.
Valid Loop Interchange

• The previous scheme doesn’t work well with our matrix multiplication, for the dependence is carried by the innermost loop

• Solution: move the innermost loop outward

• Interchange Preventing

\[
\begin{align*}
\text{DO 100 I=1, 100} & \quad \text{DO 100 J=1, 100} \\
\text{DO 100 J=1, 100} & \quad \text{DO 100 I=1, 100} \\
S1 & \quad X(I+1, J+1) = X(I+2, J) \quad \rightarrow \quad S1 \quad X(I+1, J+1) = X(I+2, J) \\
100 \text{ CONTINUE} & \quad 100 \text{ CONTINUE}
\end{align*}
\]

consider $X(3,2)$: fetch on (1,2) store on (1,2) store on (2,1) fetch on (2,1)
Valid Loop Interchange (cont.)

- A dependence between S1 and S2 which prevents the interchange of loops $k$ and $p$ is denote as $S1 \gamma_p^k S2$
- Testing for Interchange Prevention Between Adjacent Loops: Theorem 5
  - (similar to Banerjee’s inequality)
  - ($S1 \gamma_p^k S2$ only if the inequality is valid)
Valid Loop Interchange (cont.)

• Loop Interchange Between Non-adjacent Loops
  – Deduction: there are $s$ loops between loop $k$ and $p$, then use $(2s+1)$ switches to interchange them
  – Result:

<table>
<thead>
<tr>
<th>Safe</th>
<th>Unsafe</th>
</tr>
</thead>
<tbody>
<tr>
<td>$k$</td>
<td>$k$</td>
</tr>
<tr>
<td>$&lt;$</td>
<td>$&lt;$</td>
</tr>
<tr>
<td>$&lt;$</td>
<td>$&lt;$</td>
</tr>
<tr>
<td>$&lt;$</td>
<td>$&lt;$</td>
</tr>
<tr>
<td>$&lt;$</td>
<td>$&lt;$</td>
</tr>
<tr>
<td>$=$</td>
<td>$&lt;=$</td>
</tr>
<tr>
<td>$&lt;=$</td>
<td>$&lt;=$</td>
</tr>
<tr>
<td>$&lt;=$</td>
<td>$&lt;=$</td>
</tr>
<tr>
<td>$&lt;=$</td>
<td>$&lt;=$</td>
</tr>
<tr>
<td>$&lt;=$</td>
<td>$&lt;=$</td>
</tr>
</tbody>
</table>

$*$
Level Movement with Interchange

• Loop interchange may affect the levels of the dependence within a section of code.
• What would not change?

Theorem 6:
If loops $k$ and $p$ ($k < p$) are validly interchanged, then all loops independent dependent, all dependence with level $< k$ and with level $> p$ are unaffected by the interchange.
Level Movement with Interchange

• What would change?
  – Theorem 7:
    if loops k and p (k<p) are validly interchanged, then all level p dependence in the original code become level k dependences in the transformed code.
Level p: (=, …, =, <, *)
Level Movement with Interchange

• What would change? (cont.)
  – Theorem 8:
    If loops $k$ and $p$ ($k<p$) are validly interchanged, then all level $k$ dependence in the original code become level $x$ dependences in the transformed code, where $k \leq x \leq p$.
  – interchange sensitive
    can be detected by the modification of the Banerjee’s inequality [Kenn80]
Level Movement with Interchange

– Theorem 9

In a valid loop interchange of loops $k$ and $p$, a level $x$ edge ($k < x < p$) will either remain a level $x$ edge or become a level $k$ edge.
Level Movement with Interchange

• Corollary:
  
  When loops $k$ and $p$ ($k < p$) are validly interchanged, all level $p$ dependences in the transformed code were level $k$ edges in the original code.

• By this, a loop may be continually moved inward by testing only the edges originally associated with that loop.
Loop Interchange and Vectorization

• How to do the loop interchange?
  – Directly
    • Calculate the sensitivity and preventability with respect to loop interchange for each dependence edge.
    • Disadvantage: high calculation expense
  – Corollary 2: (loop shift)
    • If there are no dependence at a level k, then the k loop is innermostable, i.e., it may be place as the innermost of all loops common to a block of statements
Summary

• Go back to our matrix multiplication example
  – the independence is on level 3
  – vectorize after loop shifts