Automatic Parallelization - Part 4

Y.N. Srikant

Department of Computer Science
Indian Institute of Science
Bangalore 560 012

NPTEL Course on Compiler Design
Given two array references (with $s$ dimensions and nested in loop nest of depth $d$):

$S_v : X(f_1(l_1, ..., l_d), f_2(l), ..., f_s(l))$

$S_w : X(g_1(l_1, ..., l_d), g_2(l), ..., g_s(l))$

- We test for both $S_v \delta^* S_w$ and $S_w \delta^* S_v$ simultaneously
- The particular type of dependence ($\delta$, $\overline{\delta}$, or $\delta^o$) depends on the position of references (lhs or rhs) and the direction of dependence

We first test to see if the array regions accessed by the two references intersect

- Intersection will occur when the subscript functions are equal simultaneously

\[
\begin{align*}
 f_1(i_1, ..., i_d) & = g_1(j_1, ..., j_d) \\
 f_2(i_1, ..., i_d) & = g_2(j_1, ..., j_d) \\
 & \vdots \\
 f_s(i_1, ..., i_d) & = g_s(j_1, ..., j_d)
\end{align*}
\]
Data Dependence Framework

- Test for intersection with a DV \((*, *, ..., *)\)
- If *independence* can be proven with this DV, then the regions accessed by the two references are disjoint
- Otherwise, one "*" in the DV is refined to "<", "=", and ">", and testing is continued with these three refined DV
- Thus, testing is done by hierarchical expansion of one "*" at a time
- If independence can be proven at any point in the hierarchy, then the DV beneath it need not be tested
Complement of a DV $\psi = (\psi_1, ..., \psi_d)$ is another DV $\psi^{-1} = (\psi_1^{-1}, ..., \psi_d^{-1})$, where each $\psi_k^{-1}$ is computed from $\psi_k$ as follows:

<table>
<thead>
<tr>
<th>$\psi_k$</th>
<th>&lt; = &gt; \leq \geq \neq *</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\psi_k^{-1}$</td>
<td>&gt; = &lt; \geq \leq \neq *</td>
</tr>
</tbody>
</table>

Product of two DVs $\psi^1 = (\psi^1_1, ..., \psi^1_d)$ and $\psi^2 = (\psi^2_1, ..., \psi^2_d)$ is defined to be $\psi = (\psi_1, ..., \psi_d) = \psi^1 \times \psi^2$, where $\psi_1 = \psi^1_1 \times \psi^2_1$, $\psi_2 = \psi^1_2 \times \psi^2_2$, ..., $\psi_d = \psi^1_d \times \psi^2_d$, and $\times$ is defined on DV elements as follows:

“.” means a null DV element
### Product of DV elements

<table>
<thead>
<tr>
<th>X</th>
<th>&lt;</th>
<th>=</th>
<th>&gt;</th>
<th>≤</th>
<th>≥</th>
<th>≠</th>
<th>*</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;</td>
<td>&lt;</td>
<td>.</td>
<td>.</td>
<td>&lt;</td>
<td>.</td>
<td>&lt;</td>
<td>&lt;</td>
</tr>
<tr>
<td>=</td>
<td>.</td>
<td>=</td>
<td>.</td>
<td>=</td>
<td>=</td>
<td>=</td>
<td>=</td>
</tr>
<tr>
<td>&gt;</td>
<td>.</td>
<td>.</td>
<td>&gt;</td>
<td>.</td>
<td>&gt;</td>
<td>&gt;</td>
<td>&gt;</td>
</tr>
<tr>
<td>¬</td>
<td>&lt;</td>
<td>=</td>
<td>.</td>
<td>≤</td>
<td>=</td>
<td>&lt;</td>
<td>&lt;</td>
</tr>
<tr>
<td>¬</td>
<td>.</td>
<td>=</td>
<td>&gt;</td>
<td>=</td>
<td>≥</td>
<td>&gt;</td>
<td>&gt;</td>
</tr>
<tr>
<td>≠</td>
<td>&lt;</td>
<td>.</td>
<td>&gt;</td>
<td>&lt;</td>
<td>&gt;</td>
<td>≠</td>
<td>≠</td>
</tr>
<tr>
<td>*</td>
<td>&lt;</td>
<td>=</td>
<td>&gt;</td>
<td>≤</td>
<td>≥</td>
<td>≠</td>
<td>*</td>
</tr>
</tbody>
</table>
- Compute product of different DV corresponding to various subscripts to get one DV
  \[ \Psi = \Psi_1 \times \Psi_2 \times \ldots \times \Psi_s \]
- If this combination produces any “·” entries, then there is no simultaneous intersection at all and so there can be no dependence
- To get the data dependence DV, we must intersect \( \Psi \) with the execution order DV:
  \[ \Psi_{v\rightarrow w} = \Psi \times \Omega_{v\rightarrow w} \]
- If this produces any “·” entries, there is no dependence from \( S_v \) to \( S_w \)
If all the entries are valid, we add the data dependence relation: $S_v \delta_{v \rightarrow w}^* S_w$ to the DDG.

The actual type of dependence ($\delta$, $\bar{\delta}$, or $\delta^0$) will depend on the position of the references.

To check dependence from $S_w$ to $S_v$, we compute: $\Psi_{w \rightarrow v} = \Psi^{-1} \times \Omega_{w \rightarrow v}$.

If all the entries are valid, we add the data dependence relation: $S_w \delta_{w \rightarrow v}^* S_v$ to the DDG.
Given program:

```c
for I = 1 to 10 do {
    for J = 1 to 10 do {
        S1: A(I*10+J) = ...
        S2: ... = A(I*10+J-1)
    }
}
```

- Dependence equation: \(10I_1 + J_1 - 10I_2 - J_2 = -1\)
- GCD Test with \((*,*)\): GCD(10, 1, -10, -1) divides -1, which is true and hence dependence exists. Now we need to apply Banerjee's test
  - Lower Bound:
    \(LB^*_I = -90\) \(LB^{<}_I = -90\) \(LB^{=}_I = 0\) \(LB^{>}_I = 10\)
    \(LB^*_J = -9\) \(LB^{<}_J = -9\) \(LB^{=}_J = 0\) \(LB^{>}_J = 1\)
    \(UB^*_I = 90\) \(UB^{<}_I = -10\) \(UB^{=}_I = 0\) \(UB^{>}_I = 90\)
    \(UB^*_J = 9\) \(UB^{<}_J = -1\) \(UB^{=}_J = 0\) \(UB^{>}_J = 9\)
Data Dependence Framework Test Example - 2.2

\[(*,*)\]
\[LB_1^* + LB_J^* \leq B_0 - A_0 \leq UB_i^* + UB_J^*\]
\[-99 \leq -1 \leq 99\]

\[<,*)\]
\[-99 \leq -1 \leq -1\]
\[<,<)\]
\[-99 \leq -1 \leq -11\]
\[>^X\]

\[<,=)\]
\[-90 \leq -1 \leq -10\]
\[=^X\]

\[<,>\]
\[-89 \leq -1 \leq -1\]
\[>\sqrt{}\]

\[=,<)\]
\[-9 \leq -1 \leq -1\]
\[=^X\]

\[=,=)\]
\[-90 \leq -1 \leq -10\]
\[=^X\]

\[=,>)\]
\[-90 \leq -1 \leq -10\]
\[=^X\]
The dependence test returns two DVs: (<?,>) and (=,<)

There is only one subscript

Recall that $S_1 \Theta(=,\leq) S_2$, $S_2 \Theta(=,<) S_1$, $S_1 \Theta(<,\ast) S_2$, $S_2 \Theta(<,\ast) S_1$, are all possible

Intersect these with the execution order DVs

\[(<?,>) \times (<?,\ast) = (<?,>)\]
\[(=,\leq) \times (=,\leq) = (=,\leq)\]

Other products produce “·” values

Therefore we get: $S_1 \delta(=,<) S_2$ and $S_1 \delta(<,\ast) S_2$

There is no need to test $S_2 \delta^* S_1$, since not all entries are “·”
Concurrentization or Parallelization

- If all the dependence relations in a loop nest have a direction vector value of “=” for a loop, then the iterations of that loop can be executed in parallel with no synchronization between iterations.
- Any dependence with a forward (<) direction in an outer loop will be satisfied by the serial execution of the outer loop.
- If an outer loop L is run in sequential mode, then all the dependences with a forward (<) direction at the outer level (of L) will be automatically satisfied (even those of the loops inner to L).
- However, this is not true for those dependences with (=) direction at the outer level; the dependences of the inner loops will have to be satisfied by appropriate statement ordering and loop execution order.
Concurrentization Examples

for $l = 2$ to $N$ do {
    for $J = 2$ to $N$ do {
        $S1$: $A(l,J) = B(l,J) + 2$
        $S2$: $B(l,J) = A(l-1, J-1) - B(l,J)$
    }
}

$S1 \subseteq (\leq, \leq) S2$, $S1 \subseteq (\geq, \geq) S2$, $S2 \subseteq (\leq, \leq) S2$

<table>
<thead>
<tr>
<th>$l = 1$</th>
<th>$l = 2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$J = 1$</td>
<td></td>
</tr>
<tr>
<td>$A(2,2) = A(1,1)$</td>
<td>$A(3,2) = A(2,1)$</td>
</tr>
<tr>
<td>$J = 2$</td>
<td></td>
</tr>
<tr>
<td>$A(2,3) = A(1,2)$</td>
<td>$A(3,3) = A(2,2)$</td>
</tr>
<tr>
<td>$J = 3$</td>
<td></td>
</tr>
<tr>
<td>$A(2,4) = A(1,3)$</td>
<td>$A(3,4) = A(2,3)$</td>
</tr>
</tbody>
</table>

If the $l$ loop is run in serial mode then, the $J$ loop can be run in parallel mode.

<table>
<thead>
<tr>
<th>$l = 1$</th>
<th>$l = 2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$J = 1$</td>
<td></td>
</tr>
<tr>
<td>$A(2,2) = A(2,1)$</td>
<td>$A(3,2) = A(3,1)$</td>
</tr>
<tr>
<td>$J = 2$</td>
<td></td>
</tr>
<tr>
<td>$A(2,3) = A(2,2)$</td>
<td>$A(3,3) = A(3,2)$</td>
</tr>
<tr>
<td>$J = 3$</td>
<td></td>
</tr>
<tr>
<td>$A(2,4) = A(2,3)$</td>
<td>$A(3,4) = A(3,3)$</td>
</tr>
</tbody>
</table>

The $J$ loop cannot be run in parallel mode. However, the $l$ loop can be run in parallel mode.

Y.N. Srikant
Automatic Parallelization
Loop Transformations for increasing Parallelism

- Recurrence breaking
  - Ignorable cycles
  - Scalar expansion
  - Scalar renaming
  - Node splitting
  - Threshold detection and index set splitting
  - If-conversion

- Loop interchanging
- Loop fission
- Loop fusion
Any single statement recurrence based on $\delta$ may be ignored.

The program:

```
for I = 2 to 100 do {
S: X(I-1) = F(X(I))
}
```

has the dependence $S \delta S$, but it can be vectorized as follows:

$X(1:99) = F(X(2:100))$
Scalar Expansion

Not vectorizable or parallelizable

```plaintext
for l = 1 to N do {
    S1: T = A(l)
    S2: A(l) = B(l)
    S3: B(l) = T
}
```

Vectorizable due to scalar expansion

```plaintext
for l = 1 to N do {
    S1: Tx(l) = A(l)
    S2: A(l) = B(l)
    S3: B(l) = Tx(l)
}
```

Parallelizable due to privatization

```plaintext
forall l = 1 to N do {
    private temp
    S1: temp = A(l)
    S2: A(l) = B(l)
    S3: B(l) = temp
}
```

Acyclic DDG

Cyclic DDG
Scalar Expansion is not always profitable

Not vectorizable or parallelizable

for l = 1 to N do {
    S1:   T = T + A(l) + A(l+2)
    S2:   A(l) = T
}

Not vectorizable even after scalar expansion

for l = 1 to N do {
    S1:   Tx(l) = Tx(l-1) + A(l) + A(l+2)
    S2:   A(l) = Tx(l)
}
Scalar Renaming

The output dependence S1 \( \delta \) S3 cannot be broken by scalar expansion

The output dependence S1 \( \delta \) S3 CAN be broken by scalar renaming

1.

```plaintext
for l = 1 to N do {
    S1:   T = A(l) + B(l)
    S2:   C(l) = T*2
    S3:   T = D(l) * B(l)
    S4:   A(l+2) = T + 5
}
```

2.

```plaintext
for l = 1 to N do {
    S1:   T1 = A(l) + B(l)
    S2:   C(l) = T1*2
    S3:   T2 = D(l) * B(l)
    S4:   A(l+2) = T2 + 5
}
```

3.

```plaintext
S3:   T2(1:100) = D(1:100) * B(1:100)
S4:   A(3:102) = T2(1:100) + 5(1:100)
S1:   T1(1:100) = A(1:100) + B(1:100)
S2:   C(1:100) = T1(1:100) * 2(1:100)
T    = T2(100)
```

5(1:100) and 2(1:100) are vectors of constants
Node Splitting

Node splitting can be used in breaking a cycle consisting of an anti-dependence, but this introduces new temporary arrays.

\[
\text{for } l = 1 \text{ to } 100 \text{ do } \{
\begin{align*}
S1: & \quad A(l) = X(l+1) + X(l) \\
S2: & \quad X(l+1) = B(l)
\end{align*}
\}
\]

\[
\text{for } l = 1 \text{ to } 100 \text{ do } \{
\begin{align*}
S0: & \quad T(l) = X(l+1) \\
S1: & \quad A(l) = T(l) + X(l) \\
S2: & \quad X(l+1) = B(l)
\end{align*}
\}
\]

\[
\text{S0: } \quad T(1:100) = X(2:101) \\
\text{S2: } \quad X(2:101) = B(1:100) \\
\text{S1: } \quad A(1:100) = T(1:100) + X(1:100)
\]
Thresholds

for $l = 1$ to $100$ do {
    $X(l+5) = X(l)$
}

Cannot be vectorized
Threshold value = 5

for $l = 1$ to $20$ do {
    for $J = 1$ to $5$ do {
        $X(l*5+J) = X(l*5+J-5)$
    }
}

Thresholds can be found by modifications of Banjerjee’s test

Cannot be vectorized
Threshold value = 50

for $l = 1$ to $100$ do {
    $A(l) = A(101 - l)$
}

for $l = 1$ to $20$ do {
    $X(5*l+1 : 5*l+5)$
    = $X(5*l-4 : 5*l)$
Automatic Parallelization

If-Conversion

for \( l = 1 \) to 100 do {
  if \( A(l) \leq 0 \) then continue
  \( A(l) = B(l) + 3 \)
}

for \( l = 1 \) to 100 do {
  BR(l) = \( A(l) \leq 0 \)
  if \( \sim BR(l) \) then
    \( A(l) = B(l) + 3 \)
}

\( BR(1:N) = (A(1:N) \leq 0) \)
where \( \sim BR(1:N) \)
\( A(1:N) = B(1:N) + 3 \)

for \( l = 1 \) to \( N \) do {
  S1: \( A(l) = D(l) + 1 \)
  S2: if \( B(l) > 0 \) then
  S3: \( C(l) = C(l) + A(l) \)
  S4: \( D(l+1) = D(l+1) + 1 \)
  end if
}

for \( l = 1 \) to \( N \) do {
  S2: temp(1:N) = B(1:N) > 0
  S4: where temp(1:N)
  \( D(2:N+1) = D(2:N+1) + 1 \)
  S1: \( A(1:N) = D(1:N) + 1 \)
  S3: where temp(1:N)
  \( C(1:N) = C(1:N) + A(1:N) \)
}

Diagram:

- S1
- S2
- S3
- S4

Connections:
- S1 to S2: C
- S2 to S3: C
- S3 to S4: C
- S4 to S1: C
Loop Interchange

- For machines with vector instructions, loops can be interchanged to find vector operations, if the original inner loop cannot be vectorized.
- For multi-core and multi-processor machines, parallel outer loops are preferred and loop interchange may help to make this happen.
- Requirements for simple loop interchange
  1. The loops L1 and L2 must be tightly nested (no statements between loops).
  2. The loop limits of L2 must be invariant in L1.
  3. There are no statements $S_v$ and $S_w$ (not necessarily distinct) in L1 with a dependence $S_v \delta^* (<,>) S_w$. 

Y.N. Srikant  Automatic Parallelization
Loop Interchange for Vectorizability

for \( l = 1 \) to \( N \) do {
    for \( j = 1 \) to \( N \) do {
        S: \( A(l,J+1) = A(l,J) \cdot B(l,J) + C(l,J) \)
    }
} 

Inner loop is not vectorizable

\( S \bigcirc_{(\leq,\leq)} S \)

for \( j = 1 \) to \( N \) do {
    for \( l = 1 \) to \( N \) do {
        S: \( A(l,J+1) = A(l,J) \cdot B(l,J) + C(l,J) \)
    }
} 

Inner loop is vectorizable

\( S \bigcirc_{(\leq,=)} S \)

for \( j = 1 \) to \( N \) do {
    S: \( A(1:N, J+1) = A(1:N, J) \cdot B(1:N, J) + C(1:N, J) \)
}
Loop Interchange for parallelizability

for \( l = 1 \) to \( N \) do {
    for \( j = 1 \) to \( N \) do {
        S: \( A(l+1,j) = A(l,j) \times B(l,j) + C(l,j) \)
    }
}

Outer loop is not parallelizable, but inner loop is
\( S \delta_{\langle,=\rangle} S \)
Less work per thread

for \( j = 1 \) to \( N \) do {
    for \( l = 1 \) to \( N \) do {
        S: \( A(l+1,j) = A(l,j) \times B(l,j) + C(l,j) \)
    }
}

Outer loop is parallelizable but inner loop is not
\( S \delta_{\langle,=\rangle} S \)
More work per thread

forall \( j = 1 \) to \( N \) do {
    for \( l = 1 \) to \( N \) do {
        S: \( A(l+1,j) = A(l,j) \times B(l,j) + C(l,j) \)
    }
}
Legal Loop Interchange

S11 → S12 → S13
S21 → S22 → S23
S31 → S32 → S33

dependence
loop exec order before interchange
loop exec order after interchange

S \delta(=,\leq) S
Illegal Loop Interchange

dependence

loop exec order
before interchange

S \delta_{(<,>)} S

loop exec order
after interchange
Legal but not beneficial Loop Interchange

S $\delta_{(=,\leq)} S$
and
S $\delta_{(<,=)} S$

dependence  loop exec order  loop exec order
before interchange  after interchange
Loop Fission - Motivation

for \( l = 1 \) to \( N \) do {
S1: \( A(l) = E(l) + 1 \)
S2: \( B(l) = F(l) \times 2 \)
S3: \( C(l+1) = C(l) \times A(l) + D(l) \)
S4: \( D(l+1) = C(l+1) \times B(l) + D(l) \)
}

The above loop cannot be vectorized

L1: for \( l = 1 \) to \( N \) do {
S1: \( A(l) = E(l) + 1 \)
S2: \( B(l) = F(l) \times 2 \)
}

L2: for \( l = 1 \) to \( N \) do {
S3: \( C(l+1) = C(l) \times A(l) + D(l) \)
S4: \( D(l+1) = C(l+1) \times B(l) + D(l) \)
}

L1 can be vectorized, but L2 cannot be
**Lemma**: If a loop $L$ contains statements $S_k$ and $S_j$, where $S_k$ follows $S_j$ in the loop and $S_k \delta^* \prec S_j$, then loop fission may not split the loop at any point between $S_j$ and $S_k$.

Loop fission may not be used to break a cycle of dependence into separate loops.
Loop Fission: Legal and Illegal

for \( l = 1 \) to \( N \) do {
S1: \( A(l) = D(l) \times T \)
S2: \( B(l) = (C(l) + E(l))/2 \)
S3: \( C(l+1) = A(l) + 1 \)
}

In the above loop, S3 \( \delta_\prec \) S2, and S3 follows S2. Therefore, cutting the loop between S2 and S3 is illegal. However, cutting the loop between S1 and S2 is legal.

for \( l = 1 \) to \( N \) do {
S1: \( A(l+1) = B(l) + D(l) \)
S2: \( B(l) = (A(l) + B(l))/2 \)
S3: \( C(l) = B(l) + 1 \)
}

The above loop can be cut between S1 and S2, and also between S2 and S3.
Conditions for Loop Fusion

- Same index sets
- Loops must be adjacent
- No conditional branch (that exits) in either loop (unless the conditions are identical)
- I/O in both loops makes fusion illegal, but I/O in one of the loops is permitted
- Data dependence requirement (later)
Same Index sets for loop fusion

**LOOP 1**

```plaintext
for l = 1 to N do {
S1: A(l) = B(l) + C(l)
}
```

**LOOP 2**

```plaintext
for l = 2 to N do {
S2: D(l) = E(l) * 2
}
```

The above loops are not fusible

**Option for LOOP 1**

```plaintext
A(1) = B(1) + C(1)
for l = 2 to N do {
S1: A(l) = B(l) + C(l)
}
```

**Options for LOOP 2**

**Option A**

```plaintext
for l = 1 to N do {
S2: D(l+1) = E(l+1) * 2
}
```

**Option B**

```plaintext
for l = 1 to N do {
S2: if (l >= 2)
    D(l) = E(l) * 2
}
```
Illegal loop fusion

\[
\text{for } l = 1 \text{ to } N \text{ do } \{
\begin{align*}
S1: & \quad A(l) = B(l) + C(l) \\
S2: & \quad B(l+1) = D(l) \times 2
\end{align*}
\}
\]

If the two loops are fused, then the dependences change!

\[
\text{for } l = 1 \text{ to } N \text{ do } \{
\begin{align*}
S1: & \quad A(l) = B(l) + C(l) \\
S2: & \quad B(l+1) = D(l) \times 2
\end{align*}
\}
\]
Augmented Direction Vector

Let $S_1$ be a statement enclosed in a loop $L_1$ with index set $i$, and let $S_2$ be a statement enclosed in a loop $L_2$ with index set $j$, and let the two index sets be identical. Let $X$ be one of $\{\delta, \bar{\delta}, \delta^0\}$ and let $S_1 X S_2$.

We define the augmented DV to be $(?)$ where, $? \in \{<, =, >\}$ and we say $S_1 X (?) S_2$ when

1. there exist particular iterations of $S_1$ and $S_2$, say, $S_1(i')$ and $S_2(j')$ with $S_1(i') X S_2(j')$ and
2. $i' \neq j'$

Definition 1 above allows a DV to have positions for loops that do not contain both $S_1$ and $S_2$

Lemma: Let $L_1$ and $L_2$ be loops as above. If there are any statements $S_j$ in $L_1$ and $S_k$ in $L_2$ with $S_j \delta^*(>) S_k$, then fusing the loops is illegal
Augmented DV example

\[
\begin{align*}
\text{for } l = 2 \text{ to } N \text{ do } & \left\{ \\
S1: & A(l) = D(l) \times 2 \\
S2: & B(l) = A(l) + 1
\right. \\
\end{align*}
\]

\[ S1 \overset{\delta(=)}{\rightarrow} S2 \]

\[
\begin{align*}
\text{for } l = 2 \text{ to } N \text{ do } & \left\{ \\
S1: & A(l) = D(l) \times 2 \\
S2: & B(l) = A(l-1) + 1
\right. \\
\end{align*}
\]

\[ S1 \overset{\delta(<)}{\rightarrow} S2 \]

\[
\begin{align*}
\text{for } l = 2 \text{ to } N \text{ do } & \left\{ \\
S1: & A(l) = D(l) \times 2 \\
S2: & B(l) = A(l+1) + 1
\right. \\
\end{align*}
\]

\[ S1 \overset{\delta(>)}{\rightarrow} S2 \]