Automatic Parallelization - Part 2

Y.N. Srikant

Department of Computer Science
Indian Institute of Science
Bangalore 560 012

NPTEL Course on Compiler Design
Automatic Parallelization

- Automatic conversion of sequential programs to parallel programs by a compiler
- Target may be a vector processor (vectorization), a multi-core processor (concurrentization), or a cluster of loosely coupled distributed memory processors (parallelization)

Parallelism extraction process is normally a source-to-source transformation

Requires dependence analysis to determine the dependence between statements

Implementation of available parallelism is also a challenge
  - For example, can all the iterations of a 2-nested loop be run in parallel?
Data Dependence Relations

Flow or true dependence

\[ S1: \; x = \ldots \]
\[ \downarrow \]
\[ S2: \; \ldots = x \]
\[ \delta \]

Anti-dependence

\[ S1: \; \ldots = x \]
\[ \downarrow \]
\[ S2: \; x = \ldots \]
\[ \delta \]

Output dependence

\[ S1: \; x = \ldots \]
\[ \downarrow \]
\[ S2: \; x = \ldots \]
\[ \delta^o \]
Forward or “<” direction means dependence from iteration $i$ to $i + k$ (i.e., computed in iteration $i$ and used in iteration $i + k$)

Backward or “>” direction means dependence from iteration $i$ to $i - k$ (i.e., computed in iteration $i$ and used in iteration $i - k$). This is not possible in single loops and possible in doubly or higher levels of nesting.

Equal or “=” direction means that dependence is in the same iteration (i.e., computed in iteration $i$ and used in iteration $i$)
Individual nodes are statements of the program and edges depict data dependence among the statements.

If the DDG is acyclic, then vectorization of the program is straightforward:
- Vector code generation can be done using a topological sort order on the DDG.

Otherwise, find all the strongly connected components of the DDG, and reduce the DDG to an acyclic graph by treating each SCC as a single node:
- SCCs cannot be fully vectorized; the final code will contain some sequential loops and possibly some vector code.
for $l = 1$ to $100$ do {
  \textbf{S1:} $X(l) = Y(l) + 10$
  for $j = 1$ to $100$ do {
    \textbf{S2:} $B(j) = A(j, N)$
    for $k = 1$ to $100$ do {
      \textbf{S3:} $A(j+1, k) = B(j) + C(j, k)$
    }
  }
  \textbf{S4:} $Y(l+j) = A(j+1, N)$
}
\textbf{S1:} $X(1:100) = Y(1:100) + 10$

\textbf{code for S2, S3, S4 generated at higher levels}
for \( i = 1 \) to \( 100 \) do {
    for \( j = 1 \) to \( 100 \) do {
        code for S2 and S3 generated at higher levels
    }
}  
S4: \( Y(l+1:1+100) = A(2:101, N) \)  

Level 2 DDG for the composite node S2S3S4
Vectorization Example 3.3

```plaintext
for I = 1 to 100 do {
    for J = 1 to 100 do {
        S2: B(J) = A(J,N)
        S3: A(J+1, 1:100) = B(J) + C(J, 1:100)
    }
    S4: Y(I+1:I+100) = A(2:101, N)
}
S1: X(1:100) = Y(1:100) + N
```

Level 3 DDG for the composite node S2S3
Data Dependence Direction Vector

- Data dependence relations are augmented with a direction of data dependence which is expressed as a direction vector.
- There is one direction vector component for each loop in a nest of loops.
- The *data dependence direction vector* (or direction vector) is $\Psi = (\Psi_1, \Psi_2, \ldots, \Psi_d)$, where $\Psi_k \in \{<, =, >, \leq, \geq, \neq, *\}$
- We say $S_v \delta_{\Psi_1, \ldots, \Psi_d} S_w$ (or $S_v \delta_{\Psi} S_w$), when
  1. there exist particular instances of $S_v$ and $S_w$, say, $S_v[i_1, \ldots, i_d]$ and $S_w[j_1, \ldots, j_d]$, such that $S_v[i_1, \ldots, i_d] \delta S_w[j_1, \ldots, j_d]$, and
  2. $\theta(i_k) \Psi_k \theta(j_k)$, for $1 \leq k \leq d$
- $\theta(i_k) < \theta(j_k)$ only when iteration $i_k$ is executed before iteration $j_k$
- $\theta(i_k) = \theta(j_k)$ only when $i_k = j_k$
- $\theta(i_k) > \theta(j_k)$ only when iteration $i_k$ is executed after iteration $j_k$
Data Dependence Direction Vector

- The function $\theta(l_k) = l_k$, when the loop increment is positive and $\theta(l_k) = -l_k$, when the loop increment is negative, satisfies the above requirements.

- Forward or “<” direction means dependence from iteration $i$ to $i + k$ (i.e., computed in iteration $i$ and used in iteration $i + k$).

- Backward or “>” direction means dependence from iteration $i$ to $i - k$ (i.e., computed in iteration $i$ and used in iteration $i - k$). This is not possible in single loops and possible in doubly or higher levels of nesting.

- Equal or “=” direction means that dependence is in the same iteration (i.e., computed in iteration $i$ and used in iteration $i$).

- “*” is used when the direction is unknown or when all three of $<$, $=$, $>$ apply.
Direction Vector Example 1

for J = 1 to 100 do {
S: \( X(J) = X(J) + c \)
}

for J = 1 to 99 do {
S: \( X(J+1) = X(J) + c \)
}

for J = 1 to 99 do {
S: \( X(J) = X(J+1) + c \)
}

for J = 99 downto 1 do {
S: \( X(J) = X(J+1) + c \)
}

for J = 2 to 101 do {
S: \( X(J) = X(J-1) + c \)
}

\( S \delta_\geq S \)
\[
\begin{align*}
X(1) &= X(1) + c \\
X(2) &= X(2) + c
\end{align*}
\]

\( S \delta_< S \)
\[
\begin{align*}
X(2) &= X(1) + c \\
X(3) &= X(2) + c
\end{align*}
\]

\( S \delta_< S \)
\[
\begin{align*}
X(1) &= X(2) + c \\
X(2) &= X(3) + c
\end{align*}
\]

\( S \delta_< S \)
\[
\begin{align*}
X(99) &= X(100) + c \\
X(98) &= X(99) + c
\end{align*}
\]

\text{note ‘-ve’ increment}

\( S \delta_< S \)
\[
\begin{align*}
X(2) &= X(1) + c \\
X(3) &= X(2) + c
\end{align*}
\]
for \( i = 1 \) to 5 do {
    for \( j = 1 \) to 4 do {
        S1: \( A(i, j) = B(i, j) + C(i, j) \)
        S2: \( B(i, j+1) = A(i, j) + B(i, j) \)
    }
}

\[
\begin{align*}
\text{Demonstration of direction vector} \\
\text{I=1, J=1: } & A(1,1)=B(1,1)+C(1,1) \\
& B(1,2)=A(1,1)+B(1,1) \\
\text{J=2: } & A(1,2)=B(1,2)+C(1,2) \\
& B(1,3)=A(1,2)+B(1,2) \\
\text{J=3: } & A(1,3)=B(1,3)+C(1,3) \\
& B(1,4)=A(1,3)+B(1,3)
\end{align*}
\]
Direction Vector Example 3

\[ S_1 \delta_{(\langle,\rangle)} S_2 \]

```
for I = 1 to N do {
    for J = 1 to N do {
        S1:   A(I+1, J) = ...
        S2:   ... = A(I, J+1)
    }
}
```

```
I = 1, J = 2
S1:   A(2,2) = ...
I = 2, J = 1
S2:   ... = A(2,2)
```

\[ S_2 \delta_{(\langle,\rangle)} S_1 \]

```
for I = 1 to N do {
    for J = 1 to N do {
        S1:   A(I+1, J) = ...
        S2:   ... = A(I, J+1)
    }
}
```

```
I = 1, J = 2
S2:   A(2,2) = ...
I = 2, J = 1
S1:   ... = A(2,2)
```
**Direction Vector Example 4**

```plaintext
for I = 1 to 100 do {
    for J = 1 to 100 do {
        for K = 1 to 100 do {
            S1: X(I, J+1, K) = A(I, J, K) + 10
        }
    }
    for L = 1 to 50 do {
        S2: A(I+1, J, L) = X(I, J, L) + 5
    }
}
```

<table>
<thead>
<tr>
<th>J = 1</th>
<th>I = 1</th>
<th>I = 2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>X(1,2,K) = A(1,1,K)</td>
<td>X(2,2,K) = A(2,1,K)</td>
</tr>
<tr>
<td></td>
<td>A(2,1,L) = X(1,1,L)</td>
<td>A(3,1,L) = X(2,1,L)</td>
</tr>
<tr>
<td>J = 2</td>
<td>X(1,3,K) = A(1,2,K)</td>
<td>X(2,3,K) = A(2,2,K)</td>
</tr>
<tr>
<td></td>
<td>A(2,2,L) = X(1,2,L)</td>
<td>A(3,2,L) = X(2,2,L)</td>
</tr>
<tr>
<td>J = 3</td>
<td>X(1,4,K) = A(1,3,K)</td>
<td>X(2,4,K) = A(2,3,K)</td>
</tr>
<tr>
<td></td>
<td>A(2,3,L) = X(1,3,L)</td>
<td>A(3,3,L) = X(2,3,L)</td>
</tr>
</tbody>
</table>
for \( l = 1 \) to 100 do {
S1: \( X(l) = Y(l) + 10 \)
\hspace{1cm} for \( j = 1 \) to 100 do {
S2: \( B(j) = A(j, N) \)
\hspace{1cm} for \( k = 1 \) to 100 do {
S3: \( A(j+1, k) = B(j) + C(j, k) \)
}\}
S4: \( Y(l+j) = A(j+1, N) \)
\}

\( l=1, j=1 \) \hspace{1cm} B(1) = \ldots \\
\hspace{1.5cm} \text{for } k = \ldots \text{ do } \\
\hspace{2.5cm} \ldots = B(1) \\
\hspace{1cm} j=2 \hspace{1cm} B(2) = \ldots \\
\hspace{1.5cm} \text{for } k = \ldots \text{ do } \\
\hspace{2.5cm} \ldots = B(2) \\
\hspace{1cm} l=2, j=1 \hspace{1cm} B(1) = \ldots \\
\hspace{1.5cm} \text{for } k = \ldots \text{ do } \\
\hspace{2.5cm} \ldots = B(1) \\

\( l=1, j=1 \)  \\
\hspace{1cm} A(1, N) = \ldots \\
\hspace{1.5cm} J=2 \\
\hspace{2.5cm} A(2, N) = \ldots \\
\hspace{1.5cm} A(3, N) = \ldots \\
\hspace{1cm} l=2, j=1  \\
\hspace{1cm} A(1, N) = \ldots \\
\hspace{1.5cm} A(2, N) = \ldots \\
\hspace{2.5cm} A(2, N) = \ldots \\
\hspace{1cm} l=3, j=1  \\
\hspace{1cm} A(2, N) = \ldots \\
\hspace{1.5cm} S3 \overset{\delta_{\leq}}{\rightarrow} S2 \\
\hspace{2.5cm} S2 \overset{\delta_{\neq}}{\rightarrow} S3 \\
\hspace{2.5cm} S3 \overset{\delta_{<}}{\rightarrow} S2 \\
\hspace{2.5cm} S2 \overset{\delta_{<}}{\rightarrow} S3 \\
\hspace{2.5cm} S3 \overset{\delta_{<}}{\rightarrow} S4 \\
\hspace{2.5cm} S4 \overset{\delta_{<}}{\rightarrow} S3

for $l = 1$ to $100$ do {
    S1: $X(l) = Y(l) + 10$
    for $j = 1$ to $100$ do {
        S2: $B(j) = A(j, N)$
        for $k = 1$ to $100$ do {
            S3: $A(j+1, k) = B(j) + C(j, k)$
        }
    }
    S4: $Y(l+j) = A(j+1, N)$
}

$l=1, j=1$ $B(1) = ...$
    for $k = ...$ do
        ...
    $j=2$ $B(2) = ...$
    for $k = ...$ do
        ...
$l=2, j=1$ $B(1) = ...$
    for $k = ...$...
Execution Order Dependence and Direction Vector

- $S_v \Theta S_w$ if $S_v$ can be executed before $S_w$ (in the normal execution of the program)
- $S_v \delta \psi S_w$ only if $S_v \Theta \psi S_w$
- *i.e.*, $\Theta$ may hold but $\delta$ may not hold
- Example:

| $S_1$: $a=b+c$ | $S_1 \Theta S_2$, $S_2 \Theta S_3$, and $S_1 \Theta S_3$ are all true, but $S_1 \delta S_2$ and $S_1 \delta S_3$ are false; only $S_2 \delta S_3$ is true |
| $S_2$: $a=c+d$ |
| $S_3$: $e=a+f$ |

- Hence execution ordering is weaker
- Execution order direction vector is similar to the data dependence direction vector (similar definition)
- Not all direction vectors are possible
- We will now consider legal exec order d.v. by looking at the syntax of constructs
Single Loop Legal Direction Vectors - 1

- $S_1 \Theta_{(\leq)} S_2$, $S_2 \Theta_{(<)} S_1$, $S_1 \Theta_{(<)} S_1$, and $S_2 \Theta_{(<)} S_2$ are all possible.
- Note that $S_2 \Theta_{(=)} S_1$ is not possible because $S_2$ comes after $S_1$ in lexical ordering.

for $l = L$ to $U$ do {
  $S_1$: ...
  $S_2$: ...
}

l = 1
S1
S2

l = 2
S1
S2
Single Loop Legal Direction Vectors - 2

- \( S_1 \Theta(=) S_2 \) and \( S_2 \Theta(=) S_1 \) cannot happen
- \( S_1 \Theta(<) S_2, S_2 \Theta(<) S_1, S_1 \Theta(<) S_1, \) and \( S_2 \Theta(<) S_2 \) are all possible

```
for l = L to U do {
    if (...) then
        S1: ...
        else
            S2: ...
    endif
}
```

<table>
<thead>
<tr>
<th>l</th>
<th>S1</th>
<th>S2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>S1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>S2</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>S2</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>S1</td>
</tr>
</tbody>
</table>

S1 and S2 may be in any order, but both S1 and S2 cannot occur together in any iteration.
Loop 1

- $S_1 \Theta(=,\leq) S_2$, $S_2 \Theta(=,<) S_1$, $S_1 \Theta(<,*), S_2$, $S_2 \Theta(<,*), S_1$, $S_1 \Theta(<,*), S_1$, and $S_2 \Theta(<,*), S_2$ are all possible.
- $S_2 \Theta(=,=) S_1$ and $S_1 \Theta(=,>) S_2$ are not possible.

```
for I = L1 to U1 do {
  for J = LJ to UJ do {
    S1: ...
  }
  S2: ...
}
```
Multi-Loop Legal Direction Vectors - 2

Loop 2

- $S_1 \Theta(=,<) S_2$, $S_1 \Theta(<,\ast) S_2$, $S_2 \Theta(=,<) S_1$, and $S_2 \Theta(<,\ast) S_1$ are all possible
- $S_2 \Theta(=,=) S_1$ and $S_1 \Theta(=,=) S_2$ are not possible

```
for l = Ll to Ul do {
    for j = Lj to Uj do {
        if (...) then
            S1: ...
        else
            S2: ...
        end if
    }
}
```
Given a program segment such as:

\[
\text{for } I_1 = L_1 \text{ to } U_1 \text{ by } N_1 \text{ do } \{ \\
\ldots \\
\text{for } I_d = L_d \text{ to } U_d \text{ by } N_d \text{ do } \{ \\
S_v : \ldots X(\ldots, f(I_1, \ldots, I_d), \ldots) \ldots \\
S_w : \ldots X(\ldots, g(I_1, \ldots, I_d), \ldots) \ldots \\
\} \\
\ldots \\
\} \\
\ldots \\
\]
Data Dependence Equation

Suppose that \( \bar{l} = (l_1, ..., l_d) \), and \( f(\bar{l}) \) and \( g(\bar{l}) \) are given by

\[
\begin{align*}
 f(\bar{l}) & = A_0 + \sum_{k=1}^{d} A_k l_k \\
 g(\bar{l}) & = B_0 + \sum_{k=1}^{d} B_k l_k
\end{align*}
\]

We try to find solutions \( \bar{i} \) and \( \bar{j} \) for \( \bar{l} \) that satisfy the dependence equation

\[
 f(\bar{i}) = g(\bar{j})
\]

such that the DV is also satisfied

\[
\theta(i_k) \psi_k \theta(j_k)
\]
If we use a normalized index $I^n_k$ instead of $I_k$, where

$$I_k = I^n_k N_k + L_k$$

$I^n_k$ satisfies the inequality $0 \leq I^n_k \leq (U_k - L_k)/N_k$ and has increment one.

The dependence equations now become

$$f^n(I^n) = A_0 + \sum_{k=1}^{d} A_k N_k I^n_k + \sum_{k=1}^{d} A_k L_k$$

$$g^n(I^n) = B_0 + \sum_{k=1}^{d} B_k N_k I^n_k + \sum_{k=1}^{d} B_k L_k$$

Finding solutions $\bar{i}^n$ and $\bar{j}^n$ for $\bar{I}^n$ to the normalized equations is equivalent to finding solutions to the original equation.
The GCD Test

The dependence equation

\[ A_1 x_1 + \ldots + A_n x_n - B_1 y_1 - \ldots - B_n y_n = B_0 - A_0 \]

has a solution if and only if

\[ \text{GCD}(A_1, A_2, \ldots, A_d, B_1, B_2, \ldots, B_d) \text{ divides } B_0 - A_0 \]

The GCD test is quick but not very effective in practice.

The GCD test indicates dependence whenever the dependence equation has a solution anywhere, not necessarily within the region imposed by the loop bounds.