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 = ... S2: ... = X	δ
Anti-dependence	S1: ... = X S2: X = ...	$\bar{\delta}$
Output dependence	S1: X = ... S2: X = ...	δ^o

Data Dependence Direction Vector

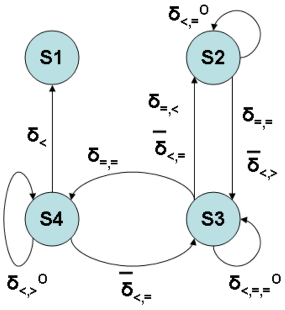
- Forward or “<” direction means dependence from iteration i to $i + k$ (*i.e.,* computed in iteration i and used in iteration $i + k$)
- Backward or “>” direction means dependence from iteration i to $i - k$ (*i.e.,* computed in iteration i and used in iteration $i - k$). This is not possible in single 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)

Data Dependence Graph and Vectorization

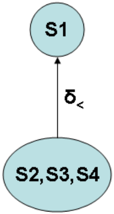
- Individual nodes are statements of the program and edges depict data dependence among the statements
- If the DDG is acyclic, then vectorization of the program is straightforward
  - Vector code generation can be done using a topological sort order on the DDG
- Otherwise, find all the strongly connected 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

Vectorization Example 3.1

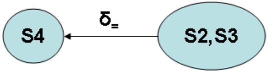
```
    for I = 1 to 100 do {
S1:     X(I) = Y(I) + 10
        for J = 1 to 100 do {
S2:         B(J) = A(J,N)
            for K = 1 to 100 do {
S3:             A(J+1, K) = B(J) + C(J, K)
            }
S4:         Y(I+J) = A(J+1, N)
        }
    }
```

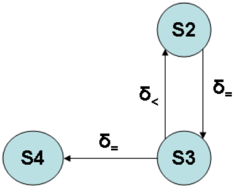



```
    for I = 1 to 100 do {
        code for S2, S3, S4
        generated at higher levels
    }
S1: X(1:100) = Y(1:100) + 10
```



```
        for I = 1 to 100 do {
           for J = 1 to 100 do {
               code for S2 and S3
               generated at
               higher levels
           }
S4:        Y(I+1:I+100) = A(2:101, N)
        }
S1:  X(1:100) = Y(1:100) + N
```

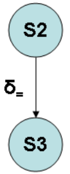




Level 2 DDG for the composite node S2S3S4

Vectorization Example 3.3

```
       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 $\Psi = (\Psi_1, \Psi_2, ..., \Psi_d)$, where $\Psi_k \in \{<, =, >, \leq, \geq, \neq, *\}$
- We say $S_v \delta_{\Psi_1,...,\Psi_d} S_w$ (or $S_v \delta_\Psi S_w$), when
  1. there exist particular instances of S_v and S_w, say, $S_v[i_1, ..., i_d]$ and $S_w[j_1, ..., j_d]$, such that $S_v[i_1, ..., i_d] \delta S_w[j_1, ..., j_d]$, 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 $\theta(I_k) = I_k$, when the loop increment is positive and $\theta(I_k) = -I_k$, 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

Loop	Dependence	Instances

for J = 1 to 100 do {
S: X(J) = X(J) +c
}
	$S\ \bar{\delta}_{=}\ S$	X(1) = X(1) +c
		X(2) = X(2)+c

for J = 1 to 99 do {
S: X(J+1) = X(J) +c
}
	$S\ \delta_{<}\ S$	X(2) = X(1) +c
		X(3) = X(2)+c

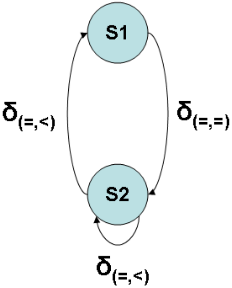
for J = 1 to 99 do {
S: X(J) = X(J+1) +c
}
	$S\ \bar{\delta}_{<}\ S$	X(1) = X(2) +c
		X(2) = X(3)+c

for J = 99 downto 1 do {
S: X(J) = X(J+1) +c
}
	$S\ \delta_{<}\ S$	X(99) = X(100) +c
		X(98) = X(99)+c
		note '-ve' increment

for J = 2 to 101 do {
S: X(J) = X(J-1) +c
}
	$S\ \delta_{<}\ S$	X(2) = X(1) +c
		X(3) = X(2)+c

Direction Vector Example 2

```
      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)
          }
      }
```



Demonstration of direction vector

I=1, J=1: A(1,1)=B(1,1)+C(1,1)
B(1,2)=A(1,1)+B(1,1)
J=2: A(1,2)=B(1,2)+C(1,2)
B(1,3)=A(1,2)+B(1,2)
J=3: A(1,3)=B(1,3)+C(1,3)
B(1,4)=A(1,3)+B(1,3)

S1 $\delta_{(=,=)}$ S2
S2 $\delta_{(=,<)}$ S1
S2 $\delta_{(=,<)}$ S2

Direction Vector Example 3

$S1\ \delta_{(<,>)}\ S2$

```
      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)
```

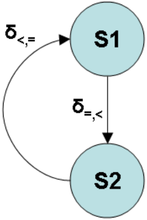
$S2\ \delta_{(<,>)}\ S1$

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

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

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

Direction Vector Example 4



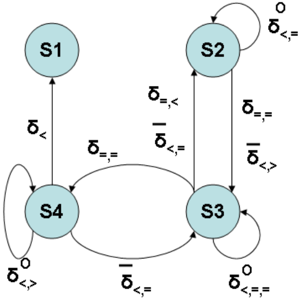

```
        for I = 1 to 100 do {
          for J = 1 to 100  do {
             for K = 1 to 100 do {
S1:              X(I, J+1, K) = A(I, J, K) + 10
             }
             for L = 1 to 50 do {
S2:              A(I+1, J, L) = X(I, J, L) +5
             }
          }
        }
```

	I = 1	I = 2
J = 1 $\delta_{=,<}$	X(1,2,K) = A(1,1,K) A(2,1,L) = X(1,1,L)	X(2,2,K) = A(2,1,K) A(3,1,L) = X(2,1,L)
J = 2	X(1,3,K) = A(1,2,K) A(2,2,L) = X(1,2,L)	X(2,3,K) = A(2,2,K) A(3,2,L) = X(2,2,L)
J = 3	X(1,4,K) = A(1,3,K) A(2,3,L) = X(1,3,L)	X(2,4,K) = A(2,3,K) A(3,3,L) = X(2,3,L)

$\delta_{<,=}$

Direction Vector Example 5.1

```
     for I = 1 to 100 do {
S1:    X(I) = Y(I) + 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(I+J) = A(J+1, N)
       }
     }
```



I=1, J=1 B(1) =...
for k=... do
... = B(1)
J=2 B(2) =...
for k=... do
... = B(2)
I=2, J=1 B(1) =...
for k=...
... = B(1)

S2 $\delta_{(=,=)}$ S3

S3 $\bar{\delta}_{(<,=)}$ S2

I=1, J=1 ... = A(1,N)
A(2,N) = ...
J=2 ... = A(2,N)
A(3,N) = ...
I=2, J=1 ... = A(1,N)
A(2,N) = ...
= A(2,N)
I=3, J=1 A(2,N) = ...

S3 $\delta_{(=,<)}$ S2

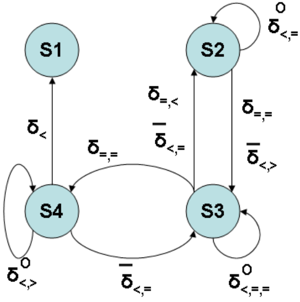
S2 $\bar{\delta}_{(<,>)}$ S3

S3 $\delta_{(=,=)}$ S4

S4 $\bar{\delta}_{(<,=)}$ S3

Direction Vector Example 5.2

```
    for I = 1 to 100 do {
S1:    X(I) = Y(I) + 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(I+J) = A(J+1, N)
       }
    }
```



I=1, J=1 B(1) =...
for k=... do
...
J=2 B(2) =...
for k=... do
...
I=2, J=1 B(1) =...
for k=...
...

S2 $\delta^{o}_{(<,=)}$ S2

I=1,J=4 Y(5) = ...
I=4,J=1 Y(5) = ...

I=5 = Y(5)

S4 $\delta^{o}_{(<,>)}$ S4

S4 $\delta_{(<)}$ S1

I=1,J=1,K=1 A(2,1) = ...
K=2 A(2,2) = ...
I=2,J=1,K=1 A(2,1) = ...
K=2 A(2,2) = ...

S3 $\delta^{o}_{(<,=,=)}$ S3

Execution Order Dependence and Direction Vector

- $S_v \Theta S_w$ if S_v can be exeuted 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.,* Θ may hold but δ may not hold
- Example:

S1: a=b+c S2: a=c+d S3: e=a+f	$S1\ \Theta S2$, $S2\ \Theta S3$, and $S1\ \Theta S3$ are all true, but $S1\ \delta S2$ and $S1\ \delta S3$ are false; only $S2\ \delta S3$ is true

- 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

- $S1\ \Theta_{(\leq)} S2$, $S2\ \Theta_{(<)} S1$, $S1\ \Theta_{(<)} S1$, and $S2\ \Theta_{(<)} S2$ are all possible
- Note that $S2\ \Theta_{(=)} S1$ is not possible because S2 comes after S1 in lexical ordering

```
      for I = L to U do {
S1:     ...
S2:     ...
      }
```

```
I = 1
   S1
   S2
I = 2
   S1
   S2
```

Single Loop Legal Direction Vectors - 2

- $S1\ \Theta_{(=)} S2$ and $S2\ \Theta_{(=)} S1$ cannot happen
- $S1\ \Theta_{(<)} S2$, $S2\ \Theta_{(<)} S1$, $S1\ \Theta_{(<)} S1$, and $S2\ \Theta_{(<)} S2$ are all possible

```
        for I = L to U do {

            if (...) then
S1:         ...
            else
S2:         ...
            endif
        }
```

```
I = 1
    S1
I = 2
    S2
I = 3
    S2
I = 4
    S1
```

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

Multi-Loop Legal Direction Vectors - 1

Loop 1

- $S1\ \Theta_{(=,\leq)} S2$, $S2\ \Theta_{(=,<)} S1$, $S1\ \Theta_{(<,*)} S2$, $S2\ \Theta_{(<,*)} S1$, $S1\ \Theta_{(<,*)} S1$, and $S2\ \Theta_{(<,*)} S2$ are all possible
- $S2\ \Theta_{(=,=)} S1$ and $S1\ \Theta_{(=,>)} S2$ are not possible

Loop 1

```
      for I = LI to UI do {
          for J = LJ to UJ do {
S1:       ...

S2:       ...
          }
      }
```

```
I = 1
    J = 1    S1
             S2
    J = 2    S1
             S2
I = 2
    J = 1    S1
             S2
    J = 2    S1
             S2
```

Loop 2

- $S1\ \Theta_{(=,<)} S2$, $S1\ \Theta_{(<,*)} S2$, $S2\ \Theta_{(=,<)} S1$, and $S2\ \Theta_{(<,*)} S1$ are all possible
- $S2\ \Theta_{(=,=)} S1$ and $S1\ \Theta_{(=,=)} S2$ are not possible

Loop 2

```
      for I = LI to UI do {
         for J = LJ to UJ do {
            if (...) then
S1:         ...
            else
S2:         ...
            end if
         }
      }
```

```
I = 1
    J = 1     S1
    J = 2     S2

I = 2
    J = 1     S2
    J = 2     S1

I = 3
    J = 1     S1
    J = 2     S2
```

Data Dependence Equation

Given a program segment such as:

$$
\begin{array}{ll}
 & \text{for } I_1 = L_1 \text{ to } U_1 \text{ by } N_1 \text{ do } \{ \\
 & \quad \ldots \\
 & \quad\quad \text{for } I_d = L_d \text{ to } U_d \text{ by } N_d \text{ do } \{ \\
S_v: & \quad\quad\quad \ldots\ X(\ldots,\ f(I_1, ..., I_d), ...)\ \ldots \\
S_w: & \quad\quad\quad \ldots\ X(\ldots,\ g(I_1, ..., I_d), ...)\ \ldots \\
 & \quad\quad \} \\
 & \quad \ldots \\
 & \}
\end{array}
$$

Data Dependence Equation

- Suppose that $\bar{I} = (I_1, ..., I_d)$, and$f(\bar{I})$ and $g(\bar{I})$ are given by

$$f(\bar{I}) = A_0 + \sum_{k=1}^{d} A_k I_k$$

$$g(\bar{I}) = B_0 + \sum_{k=1}^{d} B_k I_k$$

- We try to find solutions $\bar{i}$ and $\bar{j}$ for $\bar{I}$ that satisfy the dependence equation

$$f(\bar{i}) = g(\bar{j})$$

such that the DV is also satisfied

$$\theta(i_k) \quad \Psi_k \quad \theta(j_k)$$

- If we use a normalized index I_k^n instead of I_k, where

$$I_k = I_k^n N_k + L_k$$

- I_k^n satisfies the inequality $0 \le I_k^n \le (U_k - L_k)/N_k$ and has increment one
- The dependence equations now become

$$\begin{aligned} f^n(\overline{I^n}) &= A_0 + \sum_{k=1}^{d} A_k N_k I_k^n + \sum_{k=1}^{d} A_k L_k \\ g^n(\overline{I^n}) &= B_0 + \sum_{k=1}^{d} B_k N_k I_k^n + \sum_{k=1}^{d} B_k L_k \end{aligned}$$

- Finding solutions $\overline{i^n}$ and $\overline{j^n}$ for $\overline{I^n}$ to the normalized equations is equivalent to finding solutions to the original equation

The GCD Test - 1

- The dependence equation

$$A_1 x_1 + ... + A_n x_n - B_1 y_1 - ... - B_n y_n \quad = \quad B_0 - A_0$$

  has a solution *if and only if*
  $GCD(A_1,\ A_2,\ ...,\ A_d,\ B_1,\ B_2,\ ...,\ B_d)$ *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