Dependence Analysis

Pattern matching and replacement is all that is needed to apply many source-to-source transformations.

However, it is often necessary to gather additional information to determine the correctness of a particular transformation.

Dependence information is widely used for this purpose.

A dependence may be due to data and control
Classes of data dependence

Flow dependence (True dependence)

S1 \( X = A + B \)
S2 \( C = X + 1 \)

Anti-dependence

S1 \( A = X + B \)
S2 \( X = C + D \)

Output dependence

S1 \( X = A + B \)
\[ \ldots \]
S2 \( X = C + D \)
The notion of dependence assumes a sequential program.

The equivalent to the notion of dependence for parallel programs has been studies by Shasha and Snir (1988).

Notice that the order of execution of two statements cannot be changed *without further analysis* if one of them is dependent on the other.

Also, they cannot be executed in parallel. This is a stronger condition. Two statements could be interchangeable but not parallelizable.

Another type of dependence that does not preclude any transformation, but is useful to deal with memory issues is:

Input-dependence

\[ S1 \quad A = X + B \]
\[ S2 \quad Y = X + D \]
Dependences in loops

do I=1 to N
S1 A=B(I)+1
S2 C(I)=A+2
do I = 1 to N

S1  \( X(I+1) = B(I) + 1 \)  

S2  \( A(I) = X(I) \)  

---

do I = 1 to N

S1  \( X(I) = B(I) + 1 \)  

S2  \( A(I) = X(I+1) + 1 \)
Notice that the dependence graph for a loop is a summary of the “unrolled” dependence graph. Therefore some information is lost.

For example, the loop

\[
\begin{align*}
d & \text{do } I=1 \text{ to } N \\
S1 & \quad X(I)=B(I)+1 \\
S1 & \quad A(I)=X(I)
\end{align*}
\]

has the same dependence graph as the first loop on the previous page although their unrolled dependence graphs are different as shown on the next page.
do ...

\[ X(I) = \]
\[ \ldots = X(I) \]

do ...

\[ X(I+1) = \]
\[ \ldots = X(I) \]
Definition of dependence in loops

\[ \text{do } I=1 \text{ to } N \]
\[ \text{S1 } \quad X(F(I)) = B(I)+1 \]
\[ \text{S2 } \quad A(I) = X(G(I))+2 \]

We say that iff \( \exists I_1 \leq I_2 \) and \( I_1, I_2 \in [1,N] \) such that \( F(I_1)=G(I_2) \)

\[ \text{do } I=1 \text{ to } N \]
\[ \text{S1 } \quad A(I) = X(G(I))+1 \]
\[ \text{S2 } \quad X(F(I)) = B(I)+2 \]

We say that iff \( \exists I_1 < I_2 \) and \( I_1, I_2 \in [1,N] \) such that \( F(I_2)=G(I_1) \)
Loop parallelization and vectorization

A loop whose dependence graph is cycle-free can be parallelized or vectorized.

DO I=1,N
    X(I)=B(I)+1
    A(I)=X(I)+1
END DO

X(1:N)=B(1:N)+1
A(1:N)=X(1:N)+1

PARALLEL DO I=1,N
    X(I)=B(I)+1
    A(I)=X(I)+1
END PARALLEL DO

• The reason is that if there are no cycles in the dependence graph, then there will be no races in the parallel loop.
Algorithm Replacement

- Some program patterns occur frequently in programs. They can be replaced with a parallel algorithm. e.g.

```
DO I=1,N
    A(I)=A(I-1)+B(I)
END DO

A(1:N)=REC1N(B(1:N),A(0),N)

X=A(1)
DO I=2,N
    IF(X.GT.A(I)) X=A(I)
END DO

X=MIN(A(1:N))
```
Loop Distribution

- To insulate these patterns, we can decompose loops into several loops, one for each strongly-connected component (π-block) in the dependence graph.

```verbatim
DO I=1,N
S1:  A(I)=B(I)+C(I)
S2:  D(I)=D(I-1)+A(I)
S3:  IF(X.GT.A(I)) THEN
    S4:  X=A(I)
    ENDIF
END DO

DO I=1,N
    A(I)=B(I)+C(I)
END DO
DO I=1,N
    D(I)=D(I-1)+A(I)
END DO
DO I=1,N
    IF(X.GT.A(I)) THEN
        X=A(I)
    END IF
END DO
```
Strongly Connected Components

Let $G=(V,E)$ be a directed graph. We can partition $V$ into equivalence classes $V_i$, $1 \leq i \leq r$, such that vertices $v$ and $w$ are equivalent if and only if there is a path from $v$ to $w$ and a path from $w$ to $v$. Let $E_i \ 1 \leq i \leq r$, be the set of edges connecting the pairs of vertices in $V_i$. The graphs $G_i=(V_i,E_i)$ are called the strongly connected components of $G$. 
Loop Interchanging

- The dependence information determines whether or not the loop headers can be interchanged.
- For example, the following loop headers can be interchanged

\[
\begin{align*}
&\text{do } i=1,n \\
&\quad \text{do } j=1,n \\
&\quad \quad a(i,j) = a(i,j-1) + a(i-1,j) \\
&\quad \text{end do} \\
&\text{end do}
\end{align*}
\]

- However, the headers in the following loop cannot be interchanged
do i=1,n
    do j=1,n
        a(i,j) = a(i,j-1) + a(i-1,j+1)
    end do
end do
Dependence Removal

- Some cycles in the dependence graph can be eliminated by using elementary transformations.

_Scalar Expansion:_

\[
\text{DO } I=1,N \\
\text{S1: } A=B(I)+1 \\
\text{S2: } C(I)=A+D(I) \\
\text{END DO}
\]

\[
\text{DO } I=1,N \\
\text{S1: } A_1(I)=B(I)+1 \\
\text{S2: } C(I)=A_1(I)+D(I) \\
\text{END DO} \\
A=A_1(N)
\]
Induction Variable Recognition

DO I=1,N
S1: J=J+2
S2: X(I)=X(I)+J
END DO

DO I=1,N
S1: J1=J+2*I
S2: X(I)=X(I)+J1
END DO
Dependence Testing

Testing the conditions on p.9 could be very expensive in general. However, most subscript expressions are quite simple and can be easily analyzed.

The approach that has traditionally been used is to try to break a dependence, that is to try to prove that the dependence does not exist.

Practical tests are usually conservative. That is, they may not break a dependence that does not exist. Assuming a dependence that exists is conservative but will not lead to incorrect transformations for the cases discussed in this course.
A simple conservative test

The GCD test assumes that

\[ F(I) = A_1 I + A_0 \text{ and } G(I) = B_1 I + B_0 \]

Then, \( F(I_1) = G(I_2) \) iff \( A_1 I_1 - B_1 I_2 = A_0 - B_0 \)

The test breaks the dependence if there is no integer solution to the equation, ignoring the loop limits. This is conservative because the equation could have solutions outside the iteration space only.

There is a solution to the equation \( A_1 I_1 - B_1 I_2 = A_0 - B_0 \) iff the greatest common divisor of \( A_1 \) and \( B_1 \) divides \( A_0 - B_0 \)
A more accurate test

To take into account the loop limits we could apply Banerjee’s test which proceeds by finding an upper bound $U$, and a lower bound $L$ of $A_1 I_1 - B_1 I_2$ under the constrains that $1 \leq I_1 \leq I_2 \leq N$.

If either $L > A_0 - B_0$ or $U < A_0 - B_0$, then the functions do not intersect, and therefore we know there is no flow dependence.

For example, consider the loop

```
    do I=1 to 5
        S1    X(I+5) = B(I)+1
        S2    A(I) = X(I)+2
```

If we apply the GCD test, we will find that there is a solution to the equation $A_1 I_1 - B_1 I_2 = A_0 - B_0$ or $I_1 - I_2 = 5$, the dependence will not be broken because the equation has an integer solutions.

However, the upper limit of $A_1 I_1 - B_1 I_2 = I_1 - I_2$ is 4, which is $< A_0 - B_0$ and therefore the dependence would be broken by the second test.
Direction vectors

One way to increase the accuracy of a dependence graph is with direction vectors. For example, the two unrolled dependence graphs on page 29 could be distinguished by annotating the dependence arc with either $=$ (to indicate that the dependence is within the same iteration) or $<$ (to indicate that the dependence goes across iterations).

In general, given a multiply-nested loop

\[
\begin{align*}
&\text{do } I_1= \\
&\quad\text{do } I_2= \\
&\quad\quad\quad\cdots \\
&\quad\quad\quad\text{do } I_d= \\
&\quad\quad\quad\quad X(F(I_1,I_2,\ldots,I_d))= \ldots \\
&\quad\quad\quad\quad \ldots = X(G(I_1,I_2,\ldots,I_d))
\end{align*}
\]

our second test would check dependences for each possible direction.
For all valid direction vectors \((\Psi_1, \Psi_2, \ldots, \Psi_d)\) with each \(\Psi_i\) is either <, >, or =, the second test tries to show that there is no solution to the equation:

\[
F(I_1, I_2, \ldots, I_d) = G(I_1, I_2, \ldots, I_d)
\]

within the loop limits, with the restrictions:

\[
I_1 \Psi_1 I_1, \quad I_2 \Psi_2 I_2, \quad \ldots, \quad I_d \Psi_d I_d.
\]

These restrictions are taken into account when computing the upper and lower limits (in the complete test, there is a pair of lower and upper limits for each loop index).

If a dependence for a given direction is not broken, an arc annotated with the appropriate direction will be added to the graph.

For more details see (Wolfe and Banerjee 1987).

M. Wolfe and U. Banerjee. Data dependence and its applications to parallel processing. IJPP 16(2). 1987
There are many reasons why Banerjee’s test is conservative:

1. It only checks that there is a *real valued* solution to the equations. The solution does not have to be integer. Therefore, failure to break the dependence does not imply that the dependence exists.

2. For multiple subscript arrays, each subscript equation is tested separately. A dependence will be assumed if there is a solution for each separate equation. This is conservative because the system of equations may not have a solution even though each equation has a solution.

3. It is assumed that the loop limits and the coefficients are known constants. This can be relaxed in that the loop limit can be assumed to be infinity and the test would still work. However, if one of the coefficients is not known a dependence is assumed.