RUTGERS

THE STATE UNIVERSITY OF NEW JERSEY

CS 516 Compilers and Programming Languages II

Parallel Computing-4

Annoucements

Powerstrips will be down today around noon to reboot!

Project 1 deadline extension: Wednesday, April 1 (no joke ☺ !)

Reminder: Homework #2 sample solution has been posted

Homework #1 and #2 grades have been posted

Homework #3 has been posted; will cover vectorization algorithms this week

Loop-carried dependence

Definition:

Statement S_2 has a loop-carried dependence on statement S_1 if and only if S_1 references location M on iteration i, S_2 references M on iteration j and $d(i,j) > 0$ (that is, $D(i,j)$ contains a "<" as leftmost non "=" component).

Example:

```
DO I = 1, N
S1       A(I+1) = F(I)
S2       F(I+1) = A(I)
ENDDO
```

Loop-carried dependence

Level of a loop-carried dependence is the index of the leftmost non-"=" of D(i,j) for the dependence.

For instance:

```
DO I = 1, 10
   DO J = 1, 10
      DO K = 1, 10
S1         A(I, J, K+1) = A(I, J, K)
      ENDDO
   ENDDO
ENDDO
```

- Direction vector for S1 is (=, =, <)
- Level of the dependence is 3
- A level-k dependence between S_1 and S_1 is denoted by $S_1 \delta_k S_1$

Loop-carried Transformations

- **Theorem** Any reordering transformation that does not alter the relative order of any loops in the nest and preserves the iteration order of the level-*k* loop preserves all level-*k* dependences.
- Proof:
  - → D(i, j) has a "<" in the k^{th} position and "=" in positions 1 through k-1
  - ⇒ Source and sink of dependence are in the same iteration of loops 1 through k-1
  - ⇒ Cannot change the sense of the dependence by a reordering of iterations of those loops

- As a result of the theorem, powerful transformations can be applied

Loop-carried Transformations

Example:

```
      DO I = 1, 10
S1    A(I+1) = F(I)
S2    F(I+1) = A(I)
      ENDDO
```

can be transformed to:

```
      DO I = 1, 10
S2       F(I+1) = A(I)
S1       A(I+1) = F(I)
      ENDDO
```

Loop-independent dependences

Definition Statement S_2 has a **loop-independent dependence** on statement S_1 if and only if there exist two iteration vectors i and j such that:

1) Statement S_1 refers to memory location M on iteration i, S_2 refers to M on iteration j, and $i = j$.
2) There is a control flow path from S_1 to S_2 within the iteration.

Example:

```
DO I = 1, 10
S1      A(I) = ...
S2      ... = A(I)
ENDDO
```

Loop-independent dependences

More complicated example:

```
DO I = 1, 9
S1        A(I) = ...
S2        ...  = A(10-I)
ENDDO
```

No common loop is necessary. For instance:

```
DO I = 1, 10
S1       A(I) = ...
ENDDO
DO I = 1, 10
S2        ...  = A(10-I)
ENDDO
```

Loop-independent dependences

More complicated example:

```
DO I = 1, 9
S1        A(I) = ...
S2        ...   = A(10-I)
ENDDO
```

No common loop is necessary. For instance:

```
DO I = 1, 10
S1      A(I) = ...
S2      ...   = A(10-I)
ENDDO
```

Note: Merging these two loops (loop fusion) would change the dependence pattern (true to anti dependencies), and therefore would not be valid.

Loop-independent dependences

Theorem If there is a loop-independent dependence from S_1 to S_2, any reordering transformation that does not move statement instances between iterations and preserves the relative order of S_1 and S_2 in the loop body preserves that dependence.

- S_2 depends on S_1 with a loop independent dependence is denoted by $S_1 \ \delta_\infty \ S_2$
- Note that the direction vector will have entries that are all "=" for loop independent dependences

Loop-carried & Loop-independent Dependencies

- Loop-independent and loop-carried dependence partition all possible data dependences!

- Note that if $S_1 \, \delta \, S_2$, then S_1 executes before S_2. This can happen only if:
  - → The distance vector for the dependence is greater than 0, or
  - → The distance vector equals 0 and S_1 occurs before S_2 textually (note: we assume no control flow within loop)

...precisely the criteria for loop-carried and loop-independent dependences.

Simple Dependence Testing

- **Theorem:** Let α and β be iteration vectors within the iteration space of the following loop nest:

$$
\begin{array}{l}
\texttt{DO } i_1 = L_1,\ U_1 \\
\quad \texttt{DO } i_2 = L_2,\ U_2 \\
\qquad \ldots \\
\qquad\quad \texttt{DO } i_n = L_n,\ U_n \\
S_1 \qquad\qquad\quad A(f_1(i_1,\ldots,i_n),\ldots,f_m(i_1,\ldots,i_n)) = \ldots \\
S_2 \qquad\qquad\quad \ldots = A(g_1(i_1,\ldots,i_n),\ldots,g_m(i_1,\ldots,i_n)) \\
\qquad\quad \texttt{ENDDO} \\
\qquad \ldots \\
\quad \texttt{ENDDO} \\
\texttt{ENDDO}
\end{array}
$$

Simple Dependence Testing

```
DO i1 = L1, U1
     DO i2 = L2, U2
        ...
          DO in = Ln, Un
     S1     A(f1(i1,...,in),...,fm(i1,...,in)) = ...
     S2       ... = A(g1(i1,...,in),...,gm(i1,...,in))
          ENDDO
       ...
     ENDDO
ENDDO
```

A dependence exists from S_1 to S_2 if and only if there exist values of α and β such that

(1) α is lexicographically less than or equal to β ($\alpha \leq \beta$), and

(2) the following **system of dependence equations** is satisfied:

$$f_i(\alpha) = g_i(\beta) \text{ for all } i,\ 1 \leq i \leq m$$

Dependence Testing

Can we solve this problem exactly?
What is conservative in this framework? (false positive vs. false negative)

Typically: restrict the problem to consider index and bound expressions that are linear functions

$\Rightarrow$ solving general system of linear equations
in integers is NP-hard

Solution Methods

Inexact methods

- Greatest Common Divisor (GCD)
- Banerjee's inequalities

Cascade of exact, efficient tests (fall back on inexact methods if needed)

- Rice (see posted PLDI'91 paper)
- Stanford

Polyhedral dependence representation

Exact general tests (integer programming)