Dependence Analysis

Last Time:

- Brief introduction to interprocedural analysis

Today:

Optimization for parallel machines and memory hierarchies

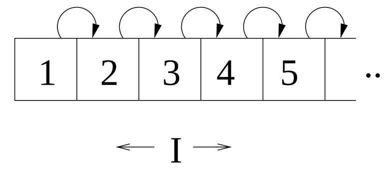
- Dependence analysis
- Loop transformations
- an example - McKinley, Carr, Tseng
  loop transformations to improve cache performance

After that:

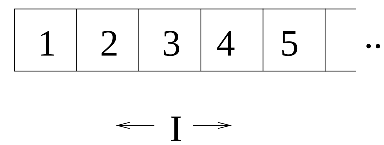
- TRIPS Architecture and Compiler (scheduling)

Dependence Examples

```
do I = 2, 100
     A(I) = A(I-1) + 1
enddo
```




```
do I = 1, 100
     A(I) = A(I) + 1
enddo
```



Can either of these loops be performed in parallel?

A *loop-independent* dependence exists regardless of the loop structure. They do not inhibit parallelization, but they do affect statement ordewhichr with a loop.

A *loop-carried* dependence is induced by the iterations of a loop and prevents safe loop parallelization.

Dependence Classification

$S_1 \ \delta \ S_2$

True (flow) dependence
occurs when S_1 writes a memory location that S_2 later reads.

Anti dependence
occurs when S_1 reads a memory location that S_2 later writes.

Output dependence
occurs when S_1 writes a memory location that S_2 later writes.

Input dependence
occurs when S_1 reads a memory location that S_2 later reads. (Input dependences do not restrict statement order.)

Dependence Analysis Question

Given

$$
\begin{array}{ll}
 & \text{DO } i_1 = L_1, U_1 \\
 & \quad \ldots \\
 & \qquad \text{DO } i_n = L_n, U_n \\
S_1 & \qquad\quad \text{A}(f_1(i_1,\ldots,i_n),\ldots,f_m(i_1,\ldots,i_n)) \ = \ \ldots \\
S_2 & \qquad\quad \ldots \ = \ \text{A}(g_1(i_1,\ldots,i_n),\ldots,g_m(i_1,\ldots,i_n))
\end{array}
$$

A *dependence* between statement S_1 and S_2, denoted $S_1 \delta S_2$, indicates that S_1, the *source*, must be executed before S_2, the *sink* on some iteration of the nest.

Let α & β be a vector of n integers within the ranges of the lower and upper bounds of the n loops.

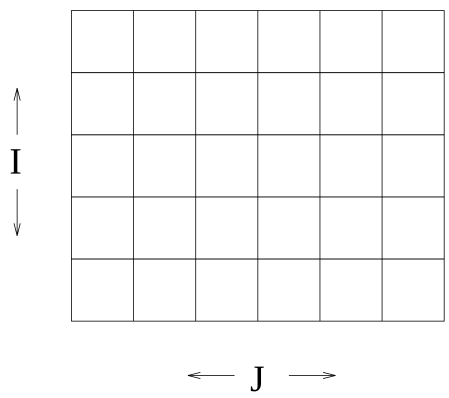
Does $\exists\ \alpha \leq \beta$, s.t.

$$f_k(\alpha) \ = \ g_k(\beta) \qquad \forall k,\ 1 \leq k \leq m \ ?$$

Iteration Space

```
do I = 1, 5
      do J = I, 6
            ...
      enddo
enddo
```

$1 \leq I \leq 5$
$I \leq J \leq 6$



- Lexicographical (sequential) order

  (1,1), (1,2), ..., (1,6)
  (2,1), (2,2), ... (2,6)
  ...
  (5,1), (5,2), ... (5,6)

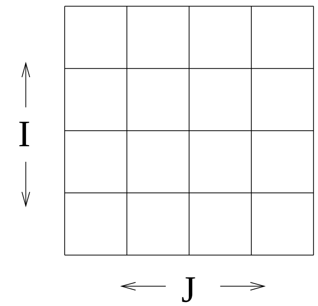
- Given $I = (i_1, \ldots i_n)$ and $I' = (i'_1, \ldots, i'_n)$,

  $I < I'$ *iff*

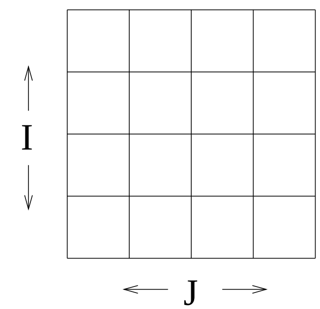
  $$(i_1, i_2, \ldots i_k) = (i'_1, i'_2, \ldots i'_k) \quad \& \quad i_{k+1} < i'_{k+1}$$

Distance & Direction Vectors

```
     do I = 1, N
       do J = 1, N
S1       A(I,J) = A(I,J-1) + 1
       enddo
     enddo
```




```
     do I = 1, N
       do J = 1, N
S2       A(I,J) = A(I-1,J-1) + 1
S3       B(I,J) = B(I-1,J+1) + 1
       enddo
     enddo
```



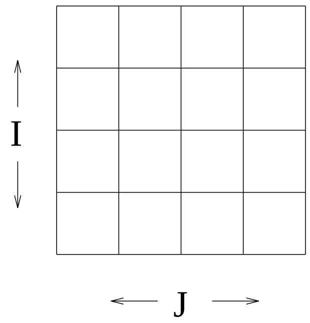
Distance Vector: number of iterations between accesses

Direction Vector: direction in the iteration space

	distance vector	direction vector
$S_1 \ \delta \ S_1$		
$S_2 \ \delta \ S_2$		
$S_3 \ \delta \ S_3$		

Which Loops are Parallel?

```
   do I = 1, N
     do J = 1, N
S1     A(I,J) = A(I,J-1) + 1
   do I = 1, N
     do J = 1, N
S2     A(I,J) = A(I-1,J-1) + 1
   do I = 1, N
     do J = 1, N
S3     B(I,J) = B(I-1,J+1) + 1
```



- A dependence $D = (d_1, \ldots, d_k)$ is *carried* at *level* i, if d_i is the first nonzero element of the distance/direction vector.
- A loop l_i is *parallel*, if $\nexists$ a dependence D_j carried at level i. Either

	distance vector	direction vector
$\forall D_j$	$d_1, \ldots, d_{i-1} \;\; > \;\; 0$	$d_1, \ldots, d_{i-1} \;\; = \;\; \text{“}<\text{”}$
OR	$d_1, \ldots, d_i \;\; = \;\; 0$	$d_1, \ldots, d_i \;\; = \;\; \text{“}=\text{”}$

Approaches to Dependence Testing

- Can we solve this problem exactly?
- What is conservative in this framework?
- Restrict the problem to consider index and bound expressions that are linear functions

$\Longrightarrow$ solve general system of linear equations
NP-complete

Solution Methods

- Cascade of exact, efficient tests (if they fail, use inexact methods)
  - Rice
  - Stanford
- Inexact methods
  - GCD
  - Banerjee's inequalities (Illinois)
  - Fourier-Motzkin (Pugh)

Greatest Common Denominator (GCD) - Inexact test

```
do i = 1, N
   a(2i+1) = a(8i+3) + a(4i)
enddo
```

$$f(I) = 2i+1$$
$$g(I') = 8i'+3$$

$$f(I) = 2i+1$$
$$g(I') = 4i'$$

let
$$\begin{aligned} f(I) &= \alpha_0 + \alpha_1 i_1 + \ldots + \alpha_k i_k \\ g(I') &= \beta_0 + \beta_1 i'_1 + \ldots + \beta_k i'_k \end{aligned}$$

- Test for integer solutions to $f(I) = g(I')$

$$\alpha_1 i_1 - \beta_1 i'_1 + \ldots \alpha_1 i_k - \beta_1 i'_k \;=\; \alpha_0 - \beta_0$$

- $\exists$ a solution *iff*
  $$\gcd(\alpha_1, \ldots, \alpha_k, \beta_1, \ldots, \beta_k) \;=\; |\alpha_0 - \beta_0|$$

- If the gcd = 1, what do we know?

- If the gcd > 1, we test to determine if the index expression ranges over that value, if so $\Longrightarrow \exists$ a dependence.

Banerjee - Inexact test

- Tests for a real solution to the integer equations
- For example, given a single index variable in the subscripts (*e.g.*, 2i and i+3) determines if the lines intersect at a real or integer point.

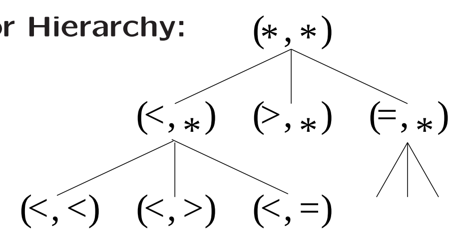
  let $h(I, I') \;=\; \alpha I - \beta I'$,
  $$\begin{aligned} h_i^+(I_k, I'_k) &= \max_{R_k} h(I_k, I'_k) \\ h_i^-(I_k, I'_k) &= \min_{R_k} h(I_k, I'_k) \end{aligned}$$

  $I_k \; D \; I'_k$ is the relation imposed by the direction vector element (either <, >, or =)

Banerjee's inequality

- For a given direction vector D, $\exists$ a real solution to $\alpha I - \beta I'$ *iff*
  $$\sum_{i=1}^{n} H_i^- - D_i \;\leq\; \beta_0 - \alpha_0 \;\leq\; \sum_{i=1}^{n} H_i^+ + D_i$$

Direction Vector Hierarchy:



Exact Test Cascade

- Stanford [Maydan, Hennessy, Lam - PLDI '91]
  - Single variable per constraint: each constraint can be solved directly
  - Acyclic test: variable is constrained by other variables in only one direction, replace variable with lower (upper) bound
  - Loop Residue Test: each constraint is of the form $i - i' \leq \alpha$, cycle with a negative value implies dependence
  - Fourier-Motzkin (inexact)

- Rice [Goff, Kennedy, Tseng - PLDI '91]
  - Index variable classification (complexity & separability)
  - ZIV test, Strong and weak SIV tests
  - Delta test for coupled subscripts: propagate constraints from separable subscripts to determine independence
  - MIV - Banerjee (inexact)

Subscript Classification Rice

Complexity:

```
A(1,   i+1,  j)
A(N,     i,  i)
```

Classification by the number of index variables occurring in subscript

- ZIV → zero index variable (51%)
- SIV → single index variable (46%)
- MIV → multiple index variable (3%)

Separability:

```
A(i+1, j,  j)
A(  i, j,  k)
```

Classification by determining if index variables are shared in subscripts

- Separable (Allen '83)
  Each subscript expression has disjoint index variables

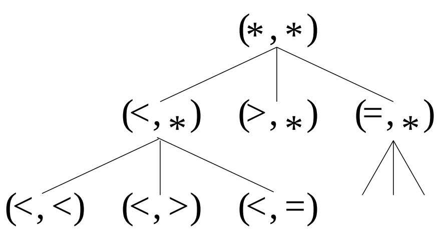
- Coupled (Li, Yew, Zhu '89)
  subscripts expressions share index variables

Taking Advantage of Separability

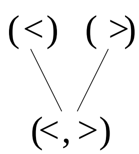
Separable subscripts

- may be tested independently
- merge the resulting dependence information

Direction Vector Hierarchy



Partition Based



Partition-Based Algorithm:

1. Partition into separable & coupled groups
2. Classify as ZIV, SIV, MIV subscripts
3. Apply dependence tests to each group
4. Finished if independent
5. Otherwise merge dependence information

ZIV test

Example: test $A(e_1)$ & $A(e_2)$

Algorithm:

- if $e_1 \neq e_2$ then independent

Symbolic test:

- symbolically compute $e_1 - e_2$

SIV Subscripts

test A(a_1I+c_1) & A(a_2I+c_2)

Strong SIV ($a_1 = a_2$)

Algorithm:

- distance d = ($c_1 - c_2$) / a
- independent if
  1. d is not integer, OR
  2. | d | > U − L

Symbolic test:

- symbolically compute $c_1 - c_2$
- symbolically compare d, U, L

Weak SIV ($a_1 = 0$ or $a_2 = 0$)

Crossing SIV ($a_1 = -a_2$)

Delta Test

- Multiple subscript test
  - Exact for common coupled subscripts
- Constraints for index variable
  - Derived from SIV subscripts
  - Distance, line, point
  - Intersect/propagate → other subscripts

Constraint Intersection

Example: test A(I, I) & A(I+1,I+2)

Constraints must hold simultaneously (intersection)

$c_1 \cap c_2 = (\{d_1 = 1\} \cap \{d_2 = 2\}) = \emptyset$

⇒ no intersection proves independence

Constraint Propagation

Example: test A(I+1, I+J) & A(I,I+J)

Propagate $C_1 = \{d_1 = 1\}$ into second subscript

⇒ A(..., J−1) & A(..., J)
⇒ Generate $C_2 = \{d_2 = -1\}$
⇒ distance vector (1,−1)

Empirical Study

Programs

- Riceps, Perfect, Spec, Eispack, Linpack

Array reference pairs tested

- All reference pairs in loop nest
- After symbolic analysis phase
- Using symbolic expression simplifier

Effectiveness

% of	ZIV	Strong SIV	Weak SIV	MIV	Delta	Sym Used
all subscripts	51	39	7	3		
all successful	31	52	8	3	6	28
all independ.	85	5	2	3	5	10
successful	44	97	90	58	43	
independent	44	3	6	22	13	

Multiple Subscript Tests

- Coupled subscripts
  - 20% of subscripts were coupled
  - 75% of coupled subscripts in Eispack
- Delta test
  - tested 82% with constraint intersection
  - tested 4% with constraint propagation

Summary

- Classifying subscripts is important
  - Complexity → fast exact tests
  - Separability → solve simple systems
- Real programs
  - Have simple subscripts
  - Simple tests are usually exact
- More practical to use quick exact tests
  - Dependence analysis for scalar compilers
  - Save the more powerful but expensive tests

Uses for Dependence Analysis

- parallelization (detection and optimization)
- vectorization
- loop optimizations
- instruction scheduling (pipelined and super scalar)
- cache optimizations

Next Time

Read:

- Improving Data Locality with Loop Transformations, Kathryn S. McKinley, Steve Carr, and Chau-Wen Tseng, *ACM Transactions on Programming Languages and Systems,* 18(4):424-453, July 1996.