Compiling for Parallelism & Locality

Assignments
- Deadline for project 4 extended to Dec 1

Last time
- Data dependences and loops

Today
- Finish data dependence analysis for loops

Dependence Testing in General

General code

```
do i1 = l1,h1
    ...
      do in = ln,hn
         A(f(i1,...,in)) = ... A(g(i1,...,in))
      enddo
    ...
    enddo
```

There exists a dependence between iterations $I=(i_1, ..., i_n)$ and $J=(j_1, ..., j_n)$ when
- `f(I) = g(J)`
- $(l_1,...l_n) < I,J < (h_1,...,h_n)$

Algorithms for Solving the Dependence Problem

Heuristics
- GCD test (Banerjee76,Towle76): determines whether integer solution is possible, no bounds checking
- Banerjee test (Banerjee 79): checks real bounds
- I-Test (Kong et al. 90): integer solution in real bounds
- Lambda test (Li et al. 90): all dimensions simultaneously
- Delta test (Goff et al. 91): pattern matches for efficiency
- Power test (Wolfe et al. 92): extended GCD and Fourier Motzkin combination

Use some form of Fourier-Motzkin elimination for integers, exponential worst-case
- Parametric Integer Programming (Feautrier91)
- Omega test (Pugh92)

Dependence Testing

Consider the following code…

```
do i = 1,5
      A(3*i+2) = A(2*i+1)+1
enddo
```

Question
- How do we determine whether one array reference depends on another across iterations of an iteration space?

Dependence Testing: Simple Case

Sample code

```
do i = l,h
   A(a*i+c1) = ... A(a*i+c2)
enddo
```

Dependence?

- $a*i_1+c_1 = a*i_2+c_2$, or
- $a*i_1 - a*i_2 = c_2-c_1$
- Solution exists if a divides c_2-c_1

Example

Code

```
do i = l,h
   A(2*i+2) = A(2*i-2)+1
enddo
```

i_1 (points to A(2*i+2))

i_2 (points to A(2*i-2))

Dependence?

$2*i_1 - 2*i_2 = -2 - 2 = -4$

(yes, 2 divides -4)

Kind of dependence?

- Anti? $i_2 + d = i_1 \Rightarrow d = -2$
- Flow? $i_1 + d = i_2 \Rightarrow d = 2$

GCD Test

Idea

- Generalize test to linear functions of iterators

Code

```
do i = li,hi
   do j = lj,hj
      A(a1*i + a2*j + a0) = ... A(b1*i + b2*j + b0) ...
      enddo
   enddo
```

Again

- $a_1*i_1 - b_1*i_2 + a_2*j_1 - b_2*j_2 = b_0 - a_0$
- Solution exists if $gcd(a_1,a_2,b_1,b_2)$ divides $b_0 - a_0$

Example

Code

```
do i = li,hi
   do j = lj,hj
      A(4*i + 2*j + 1) = ... A(6*i + 2*j + 4) ...
   enddo
enddo
```

gcd(4,-6,2,-2) = 2

Does 2 divide 4-1?

Banerjee Test

```
for (i=L; i<=U; i++) {
  x[a_0 + a_1*i] = ...
  ... = x[b_0 + b_1*i]
}
```

Does `a_0 + a_1*i = b_0 + b_1*i'` for some integer `i` and `i'`?
If so then `(a_1*i - b_1*i') = (b_0 - a_0)`

Determine upper and lower bounds on `(a_1*i - b_1*i')`

```
for (i=1; i<=5; i++) {
  x[i+5] = x[i];
}
```

upper bound = a_1*max(i) - b_1 * min(i') = 4
lower bound = a_1*min(i) - b_1*max(i') = -4
b_0 - a_0 =

Distance Vectors: Legality

Definition

- A dependence vector, *v*, is **lexicographically nonnegative** when the left-most entry in *v* is positive or all elements of *v* are zero
  Yes: (0,0,0), (0,1), (0,2,-2)
  No: (-1), (0,-2), (0,-1,1)
- A dependence vector is **legal** when it is lexicographically nonnegative (assuming that indices increase as we iterate)

Why are lexicographically negative distance vectors illegal?

What are legal direction vectors?

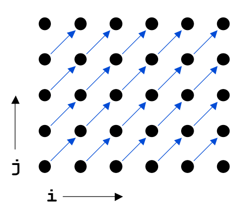
Direction Vector

Definition

- A direction vector serves the same purpose as a distance vector when less precision is required or available
- Element *i* of a direction vector is <, >, or = based on whether the source of the dependence precedes, follows or is in the same iteration as the target in loop *i*

Example

```
do i = 1,6
   do j = 1,5
      A(i,j) = A(i-1,j-1)+1
   enddo
enddo
```



Direction vector: (<,<)
Distance vector: (1,1)

Loop-Carried Dependences

Definition

- A dependence $D=(d_1,...d_n)$ is **carried** at loop level *i* if d_i is the first nonzero element of D

Example

```
do i = 1,6
   do j = 1,6
      A(i,j) = B(i-1,j)+1
      B(i,j) = A(i,j-1)*2
   enddo
enddo
```

Distance vectors: (0,1) for accesses to **A**
(1,0) for accesses to **B**

Loop-carried dependences

- The j loop carries dependence due to **A**
- The i loop carries dependence due to **B**

Parallelization

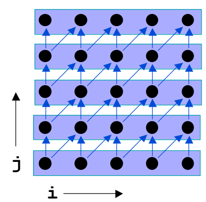
Idea

– Each iteration of a loop may be executed in parallel if it carries no dependences

Example

```
do i = 1,6
   do j = 1,5
      A(i,j) = B(i-1,j-1)+1
      B(i,j) = A(i,j-1)*2
   enddo
enddo
```

Iteration Space



Parallelize `i` loop?

Distance Vectors:

(1,0) for **A** (flow)
(1,1) for **B** (flow)

Parallelization

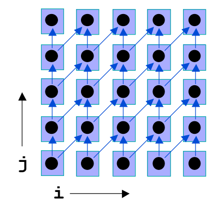
Idea

– Each iteration of a loop may be executed in parallel if it carries no dependences

Example

```
do i = 1,6
   do j = 1,5
      A(i,j) = B(i-1,j-1)+1
      B(i,j) = A(i,j-1)*2
   enddo
enddo
```

Iteration Space



Parallelize `j` loop?

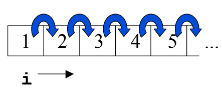
Distance Vectors:

(1,0) for **A** (flow)
(1,1) for **B** (flow)

Example 2: Parallelization (reprise)

Why can't this loop be parallelized?

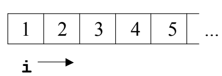
```
do i = 1,100
   A(i) = A(i-1)+1
enddo
```



Distance Vector: (1)

Why can this loop be parallelized?

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



Distance Vector: (0)

Example 1: Loop Permutation (reprise)

Sample code

```
do j = 1,6
   do i = 1,5
      A(j,i) = A(j,i)+1
   enddo
enddo
```

```
do i = 1,5
   do j = 1,6
      A(j,i) = A(j,i)+1
   enddo
enddo
```

Why is this legal?

– No loop-carried dependences, so we can arbitrarily change order of iteration execution

Concepts

Improve performance by ...
- improving data locality
- parallizing the computation

Data Dependences
- iteration space
- distance vectors and direction vectors
- loop carried

Transformation legality
- must respect data dependences
- scalar expansion as a technique to remove anti and output dependences

Data Dependence Testing
- general formulation of the problem
- GCD test

Next Time

Lecture
- Loop transformations for parallelism and locality