Compiling for Parallelism & Locality

Announcement
– Need to make up November 14th lecture

Last time
– Data dependences and loops

Today
– Finish data dependence analysis for loops

Example

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

Kind of dependence: Flow

Distance vector: (1, -1)
Exercise

Sample code

\[
\begin{align*}
\text{do } & j = 1,5 \\
\text{do } & i = 1,6 \\
& A(i,j) = A(i-1,j+1) + 1 \\
\text{enddo} \\
\text{enddo}
\end{align*}
\]

Kind of dependence: Anti

Distance vector: \((1, -1)\)

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

\[
\begin{align*}
\text{do } & i = 1,5 \\
\text{do } & j = 1,6 \\
& A(j,i) = A(j-1,i-1) + 1 \\
\text{enddo} \\
\text{enddo}
\end{align*}
\]

Direction vector: \((<, <)\)

Distance vector: \((1, 1)\)
**Distance Vectors: Legality**

**Definition**

- A dependence vector, $v$, is **lexicographically nonnegative** when the leftmost 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?**

---

**Loop-Carried Dependences**

**Definition**

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

**Example**

```plaintext
    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:**
- (1,0) for accesses to $A$
- (0,1) for accesses to $B$

**Loop-carried dependences**

- The $i$ loop carries dependence due to $A$
- The $j$ loop carries dependence due to $B
Parallelization

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

Example
\[
\begin{align*}
\text{do } i &= 1, 6 \\
\text{do } j &= 1, 5 \\
A(i, j) &= B(i-1, j-1)+1 \\
B(i, j) &= A(i, j-1)*2
\end{align*}
\]

Parallelize \(i\) loop?

Parallelization

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

Example
\[
\begin{align*}
\text{do } i &= 1, 6 \\
\text{do } j &= 1, 5 \\
A(i, j) &= B(i-1, j-1)+1 \\
B(i, j) &= A(i, j-1)*2
\end{align*}
\]

Parallelize \(j\) loop?
Scalar Expansion: Motivation

Problem

- Loop-carried dependences inhibit parallelism
- Scalar references result in loop-carried dependences

Example

```plaintext
do i = 1, 6
    t = A(i) + B(i)
    C(i) = t + 1/t
enddo
```

Can this loop be parallelized? No.
What kind of dependences are these? Anti dependences.

Convention for these slides: Arrays start with upper case letters, scalars do not

Scalar Expansion

Idea
- Eliminate false dependences by introducing extra storage

Example

```plaintext
do i = 1, 6
    T(i) = A(i) + B(i)
    C(i) = T(i) + 1/T(i)
enddo
```

Can this loop be parallelized? i

Disadvantages?
Scalar Expansion Details

Restrictions
- The loop must be a **countable** loop
  - *i.e.* The loop trip count must be independent of the body of the loop
- There can not be loop-carried flow dependences due to the scalar
- The expanded scalar must have no **upward exposed uses** in the loop

```plaintext
do i = 1,6
    print(t)
    t = A(i) + B(i)
    C(i) = t + 1/t
endo
```
- Nested loops may require much more storage
- When the scalar is live after the loop, we must move the correct array value into the scalar

Example 2: Parallelization (reprise)

Why can’t this loop be parallelized?

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

Why can this loop be parallelized?

```plaintext
do i = 1,100
    A(i) = A(i)+1
endo
```

Distance Vector: (1)

Distance Vector: (0)
**Example 1: Loop Permutation (reprise)**

Sample code

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

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

Why is this legal?
- No loop-carried dependences, so we can arbitrarily change order of iteration execution

**Dependence Testing**

Consider the following code...

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

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

General code

\[
\begin{align*}
&\text{do } i_1 = l_1, h_1 \\
&\quad \ldots \\
&\text{do } i_n = l_n, h_n \\
&\quad A(f(i_1, \ldots, i_n)) = \ldots A(g(i_1, \ldots, i_n)) \\
&\text{enddo} \\
&\quad \ldots \\
&\text{enddo}
\end{align*}
\]

There exists a dependence between iterations \(I=(i_1, \ldots, i_n)\) and \(J=(j_1, \ldots, j_n)\) when

- \(f(I) = g(J)\)
- \((l_1, \ldots, l_n) < I, J < (h_1, \ldots, h_n)\)

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**
- Parametric Integer Programming (Feautrier91)
- Omega test (Pugh92)
Dependence Testing: Simple Case

Sample code

```plaintext
do i = l, h
    A(a*i+c_1) = ... A(a*i+c_2)
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

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

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 = l_i, h_i
        do j = l_j, h_j
            A(a_1*i + a_2*j + a_0) = ... A(b_1*i + b_2*j + b_0) ...
        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 = l_i, h_i
        do j = l_j, h_j
            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?
Concepts

Improve performance by ...
- improving data locality
- parallelizing 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
- Value dependence analysis