

Review

Distance vectors

- Concisely represent dependences in loops (*i.e.*, in iteration spaces)
- Dictate what transformations are legal
 - *e.g.*, Permutation and parallelization

Legality

- A dependence vector is **legal** when it is lexicographically nonnegative

Loop-carried dependence

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

Loop Fusion Example

What are the dependences?

```
do i = 1, n
s1   A(i) = B(i) + 1
enddo
do i = 1, n
s2   C(i) = A(i) / 2
enddo
do i = 1, n
s3   D(i) = 1/C(i+1)
enddo
```



What are the dependences?

```
do i = 1, n
s1   A(i) = B(i) + 1
s2   C(i) = A(i) / 2
s3   D(i) = 1/C(i+1)
enddo
```

Fusion changes the dependence between s_2 and s_3 , so fusion is illegal

Kelly and Pugh Transformation Framework

Specify iteration space as a set of integer tuples

$$\{ [i, j] \mid 1 \leq i, j \leq n \}$$

Specify data dependences as mappings between integer tuples (i.e., data dependence relations)

$$\{ [i, j] \rightarrow [i+1, j+1] \mid 1 \leq i, j \leq n-1 \}$$

Specify transformations as mappings between integer tuples

$$\{ [i, j] \rightarrow [j, i] \}$$

Execute iterations in transformed iteration space in lexicographic order

Specifying Loop Fusion in Kelly and Pugh Framework

Specify iteration space as a set of integer tuples

symbolic n;
ISG1 := { [1,i,1] : 1 ≤ i ≤ n };
ISG2 := { [2,i,1] : 1 ≤ i ≤ n };
ISG3 := { [3,i,1] : 1 ≤ i ≤ n };

Specify data dependences as mappings between integer tuples (i.e., data dependence relations)

D12 := { [1,i,1] → [2,i,1] };
D23 := { [2,i,1] → [3,i-1,1] };

Specify transformations as mappings between integer tuples

T1 := { [1,i,1] → [1,j,1] };
T2 := { [2,i,1] → [1,j,2] };
T3 := { [3,i,1] → [1,j,3] };

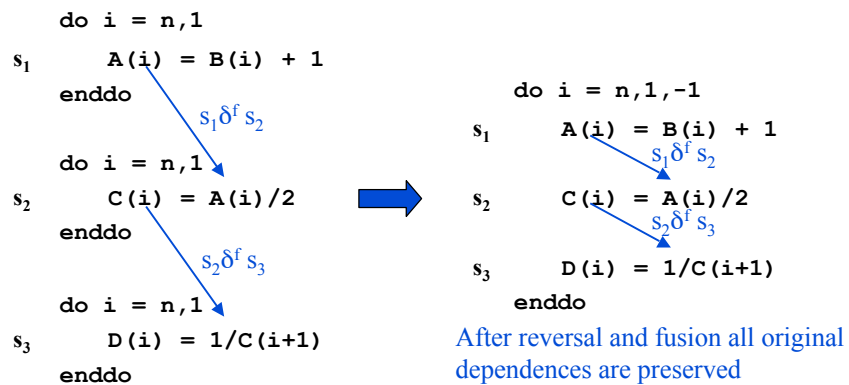
Checking Legality in Kelly & Pugh Framework

For each dependence, $[I] \rightarrow [J]$ the transformed I iteration must be executed after the transformed J iteration.

Loop Fusion Example (cont)

Loop reversal is legal for the original loops

- Does not change the direction of any dep in the original code
- Will reverse the direction in the fused loop: $s_3 \delta^a s_2$ will become $s_2 \delta^f s_3$



Loop Fission (Loop Distribution)

Idea

- Split a loop nest into multiple loop nests (the inverse of fusion)

Example

```
do i = 1, n
  A(i) = B(i) + 1
  C(i) = A(i) / 2
enddo
```



```
do i = 1, n
  A(i) = B(i) + 1
enddo
do i = 1, n
  C(i) = A(i) / 2
enddo
```

Motivation?

- Produces multiple (potentially) less constrained loops
- May improve locality
- Enable other transformations, such as interchange

Legality?

CS553 Lecture

Loop Transformations

8

Loop Fission (cont)

Legality

- Fission is legal when the loop body contains no cycles in the dependence graph

```
do i = 1, n
  body1
  body2
enddo
```



```
do i = 1, n
  body1
enddo
do i = 1, n
  body2
enddo
```

Cycles cannot be preserved because after fission all cross-loop dependences flow from body1 to body2

CS553 Lecture

Loop Transformations

9

Loop Fission Example

Recall our fusion example

```

do i = 1,n
s1   A(i) = B(i) + 1
enddo
do i = 1,n
s2   C(i) = A(i)/2
enddo
do i = 1,n
s3   D(i) = 1/C(i+1)
enddo
    
```

Can we perform fission on this loop?

```

do i = 1,n
s1   A(i) = B(i) + 1
s2   C(i) = A(i)/2
s3   D(i) = 1/C(i+1)
enddo
    
```

Loop Fission Example (cont)

If there are no cycles, we can reorder the loops with a topological sort

```

do i = 1,n
s1   A(i) = B(i) + 1
enddo
do i = 1,n
s3   D(i) = 1/C(i+1)
enddo
do i = 1,n
s2   C(i) = A(i)/2
enddo
    
```

Can we perform fission on this loop?

```

do i = 1,n
s1   A(i) = B(i) + 1
s2   C(i) = A(i)/2
s3   D(i) = 1/C(i+1)
enddo
    
```

Scheduling SAREs and Loop Transformations

Loop Transformations

- parse iterative code and build an abstraction for iteration space
- analyze dependences within iteration space
- specify a transformation on the iteration space that preserves the dependences
- execute the transformed iteration space in lexicographic order

SAREs: Systems of Affine Recurrence Equations

- example: `for (i = 1..n, j = 1..n), X[i,j] = X[i-1,j] + X[i,j-1]`
- specify computation in a high-level functional language, main difference is that no order is specified for iterating over i and j
- dependences are easy to derive because everything is single assignment
- determine a parallel schedule for the computation

Scheduling a SARE

```
for (i = 1..n, j = 1..n), X[i,j] = X[i-1,j] + X[i,j-1]
```

Linear schedule is of the form

$$t(i,j) = a*i + b*j + c$$

Finding constraints on the schedule is similar to checking transformation legality in omega

$$t(i,j) = a*i + b*j + c$$

Concepts

Loop transformation

- Loop fusion
- Loop fission

Kelly & Pugh Transformation Framework

- iteration spaces as constrained sets of integer tuples
- data dependences as mappings between integer tuples
- transformations as mappings between integer tuples

Scheduling for ALPHA programs involves determining what constraints will satisfy the data dependences

Next Time

Lecture

- Tiling