

Loop Fusion and Fission and Presburger Trans Framework

Last time

- ! Unimodular transformation framework
 - ! Loop permutation, Loop reversal, Loop skewing
- ! Fourier Motzkin

Frameworks

- ! Unimodular
- ! Polyhedral
- ! Presburger
- ! Sparse Polyhedral

Today

- ! Presburger or Kelly & Pugh transformation framework
 - ! Loop fusion
 - ! Loop fission
 - ! Unroll and jam

Loop Fusion

Idea

- ! Combine multiple loop nests into one

Example

```
do i = 1,n
  A(i) = A(i-1)
enddo
do j = 1,n
  B(j) = A(j)/2
enddo
```



```
do i = 1,n
  ! A(i) = A(i-1)
  ! B(i) = A(i)/2
enddo
```

Pros

- May improve data locality
- Reduces loop overhead
- Enables **array contraction** (opposite of scalar expansion)
- May enable better instruction scheduling

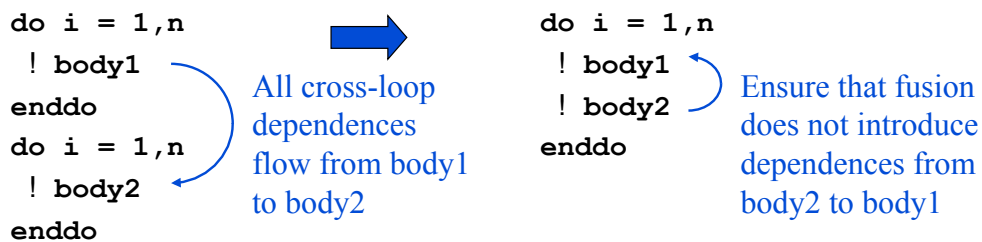
Cons

- May hurt data locality
- May hurt icache performance

Legality of Loop Fusion

Basic Conditions

- ! Both loops must have same structure
 - ! Same loop depth
 - ! Same loop bounds
 - ! Same iteration directions
- } Can we relax any of these restrictions?
- ! Dependences must be preserved
 - e.g.*, Flow dependences must not become anti dependences



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

```

$s_1 \delta^f s_2$
 $s_2 \delta^f s_3$

!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

```

$s_1 \delta^f s_2$
 $s_3 \delta^a s_2$

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

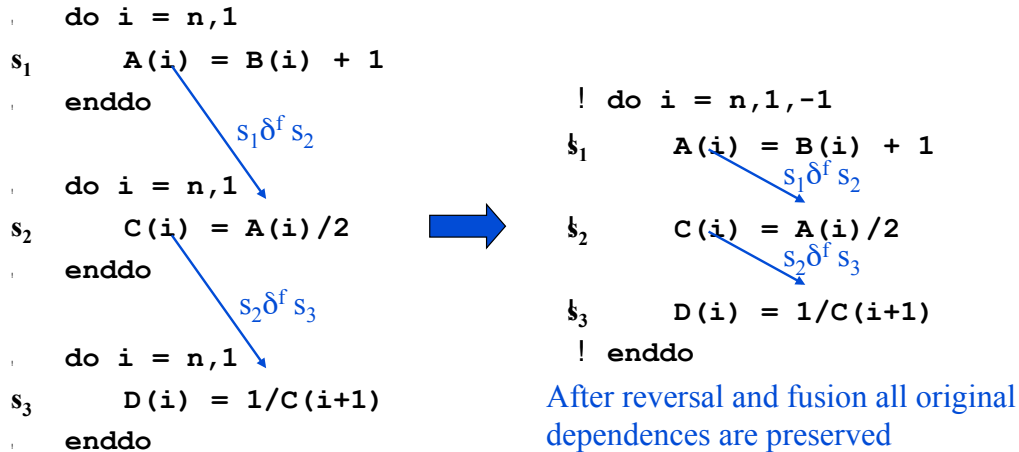
Is there some transformation that will enable fusion of these loops?

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 $s_2 \delta^f s_3$



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 relations between integer tuples (i.e., data dependence relations)

$$\{[i_1, j_1] \rightarrow [i_2, j_2] \mid (i_1 = i_2 - 1) \wedge (j_1 = j_2 - 1) \wedge (1 \leq i_1, j_1, i_2, j_2 \leq n)\}$$

Specify transformations as relations/mappings between integer tuples

$$\{[i, j] \rightarrow [i', j'] \mid (i' = j) \wedge (j' = i)\}$$

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

$$IS_1 = \{[1, i_1, 1] \mid 1 \leq i_1 \leq n\}$$

$$IS_2 = \{[2, i_2, 1] \mid 1 \leq i_2 \leq n\}$$

$$IS_3 = \{[3, i_3, 1] \mid 1 \leq i_3 \leq n\}$$

$$IS = IS_1 \cup IS_2 \cup IS_3$$

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

$$D_{12} = \{[1, i_1, 1] \rightarrow [2, i_2, 1] \mid i_1 = i_2\}$$

$$D_{23} = \{[2, i_2, 1] \rightarrow [3, i_3, 1] \mid i_2 = i_3 + 1\}$$

$$D = D_{12} \cup D_{23}$$

Specify transformations as mappings between integer tuples

$$T_1 = \{[1, i_1, 1] \rightarrow [1, i'_1, 1] \mid i'_1 = i_1\}$$

$$T_2 = \{[2, i_2, 1] \rightarrow [1, i'_2, 2] \mid i'_2 = i_2\}$$

$$T_3 = \{[3, i_3, 1] \rightarrow [1, i'_3, 3] \mid i'_3 = i_3\}$$

$$T = T_1 \cup T_2 \cup T_3$$

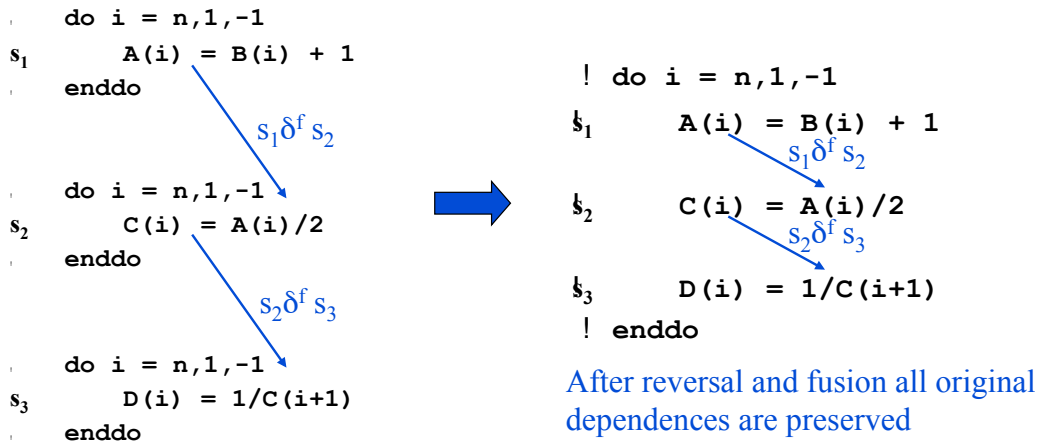
Checking Legality in Kelly & Pugh Framework

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

Loop Fusion Example (cont)

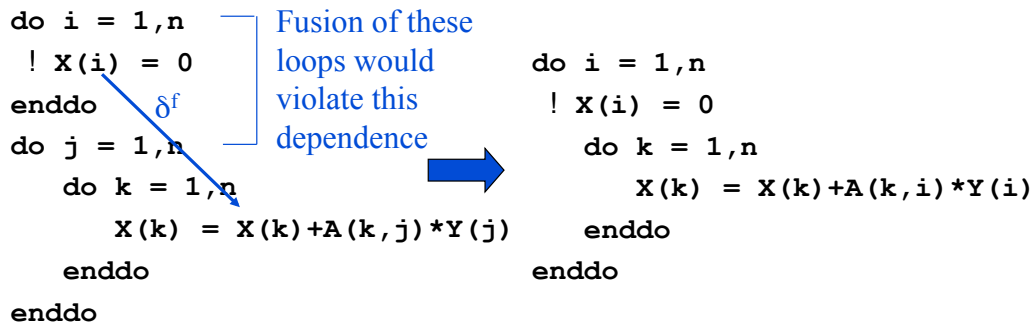
Loop reversal is legal for the original loops

- ! 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 $s_2 \delta^f s_3$



Fusion Example


Can we fuse these loop nests?



Fusion Example (cont)

Use loop interchange to preserve dependences

```
do i = 1,n
  ! X(i) = 0
enddo
do k = 1,n
  do j = 1,n
    X(k) = X(k) + A(k,j) * Y(j)
  enddo
enddo
```



```
do i = 1,n
  ! X(i) = 0
  do j = 1,n
    X(i) = X(i) + A(i,j) * Y(j)
  enddo
enddo
```

The diagram illustrates the transformation of a nested loop. In the original code, the innermost loop is over j (range $1, n$), followed by k (range $1, n$), and then i (range $1, n$). A blue arrow labeled δ^f points from the j loop to the i loop, indicating a fusion operation. The transformed code shows the loops in the order i , j , and then k . A blue arrow labeled δ^f points from the j loop to the i loop in the transformed code, indicating the same fusion operation.


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
```

The diagram illustrates the transformation of a single loop nest into two separate loop nests. The original code has a single loop over i (range $1, n$) containing two statements: $A(i) = B(i) + 1$ and $C(i) = A(i) / 2$. A blue arrow points to the transformed code, which consists of two separate loop nests, each over i (range $1, n$). The first loop nest contains the statement $A(i) = B(i) + 1$, and the second loop nest contains the statement $C(i) = A(i) / 2$.

Motivation?

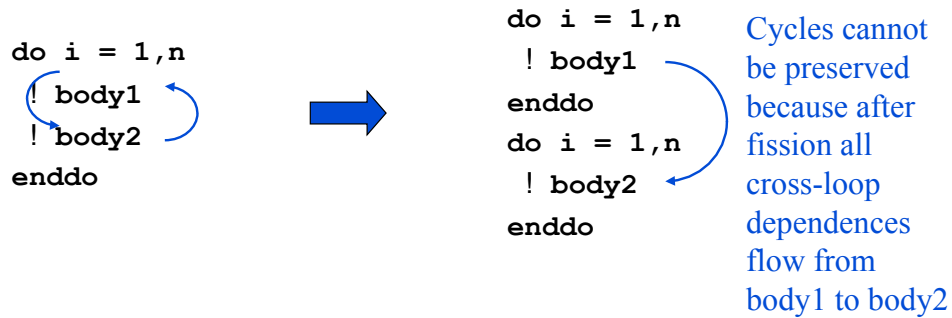
- ! Produces multiple (potentially) less constrained loops
- ! May improve locality
- ! Enable other transformations, such as interchange

Legality?

Loop Fission (cont)

Legality

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



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?

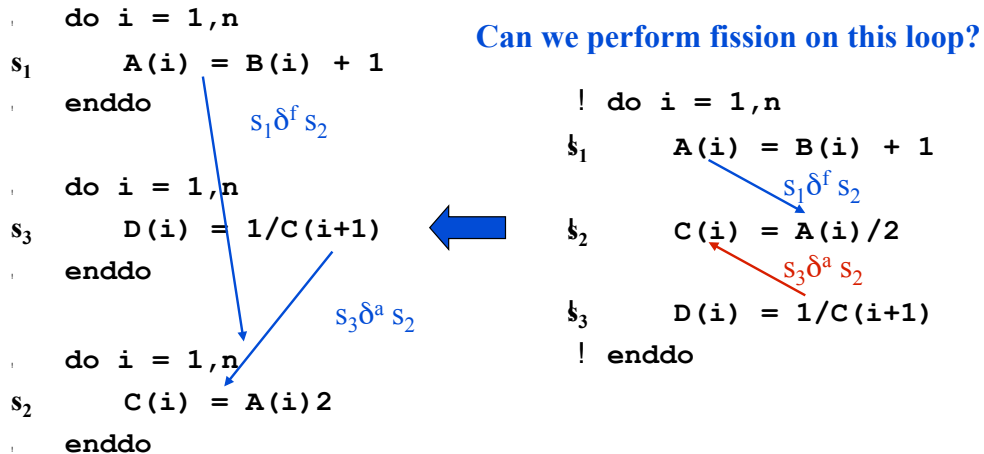
```

! 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



Loop Unrolling

Motivation

- ! Reduces loop overhead
- ! Improves effectiveness of other transformations
 - ! Code scheduling
 - ! CSE


The Transformation

- ! Make n copies of the loop: n is the **unrolling factor**
- ! Adjust loop bounds accordingly

Loop Unrolling (cont)

Example

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



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

Details

- !When is loop unrolling legal?
- !Handle end cases with a cloned copy of the loop
 - !Enter this special case if the remaining number of iteration is less than the unrolling factor

Loop Balance

Problem

- !We'd like to produce loops with the right balance of memory operations and floating point operations
- !The ideal balance is machine-dependent
 - !e.g. How many load-store units are connected to the L1 cache?
 - !e.g. How many functional units are provided?

Example

```
! do j = 1, 2*n
!   do i = 1, m
!     A(j) = A(j) + B(i)
!   enddo
! enddo
```

- !The inner loop has 1 memory operation per iteration and 1 floating point operation per iteration
- !If our target machine can only support 1 memory operation for every two floating point operations, this loop will be memory bound

What can we do?

Unroll and Jam

Idea

–! Restructure loops so that loaded values are used many times per iteration

Unroll and Jam

–! Unroll the outer loop some number of times

–! Fuse (Jam) the resulting inner loops

Example

```
! do j = 1, 2*n
  do i = 1, m
    !   A(j) = A(j) + B(i)
    !   enddo
  ! enddo
```



Unroll the Outer Loop

```
do j = 1, 2*n by 2
  do i = 1, m
    A(j) = A(j) + B(i)
  enddo
  do i = 1, m
    A(j+1) = A(j+1) + B(i)
  enddo
enddo
```

Unroll and Jam Example (cont)

Unroll the Outer Loop

```
do j = 1, 2*n by 2
  do i = 1, m
    A(j) = A(j) + B(i)
  enddo
  do i = 1, m
    A(j+1) = A(j+1) + B(i)
  enddo
enddo
```



Jam the inner loops

```
–! The inner loop has 1 load per iteration and 2 floating point operations per iteration
–! We reuse the loaded value of B(i)
–! The Loop Balance matches the machine balance

! do j = 1, 2*n by 2
!   do i = 1, m
!       A(j) = A(j) + B(i)
!       A(j+1) = A(j+1) + B(i)
!   enddo
! enddo
```

Unroll and Jam (cont)

Legality

- ! When is Unroll and Jam legal?

Disadvantages

- ! What limits the degree of unrolling?

Concepts

Loop transformation

- ! Loop fusion
- ! Loop fission
- ! Unroll and jam

Kelly & Pugh Transformation Framework

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

Next Time

Lecture

–! Automatic Parallelization

Reading

–! Automatic Parallelization