Unimodular Transformations and Dependences

Announcements

- Project proposal re-write due this Friday, October 24th.
- PA3 has been posted, due Monday November 3rd.

Today

- PA3 intro
- Unimodular transformation framework rehash with loop reversal
- Dependence problem
  - Lexicographical constraints to compute output, anti, or flow.
  - Computing direction/distance vectors
- Testing transformation legality
- Converting C++ code to iteration space representation

Frameworks for Loop Transformations

Loop Transformations as functions

$$\vec{i'} = f(\vec{i})$$

Unimodular Loop Transformations [Banerjee 90],[Wolf & Lam 91]

- can represent loop permutation, loop reversal, and loop skewing
- unimodular linear mapping (determinant of matrix is + or - 1)

$$\vec{i'} = T\vec{i}$$

  - T is a matrix, i and i' are iteration vectors
  - example

$$\begin{bmatrix} i' \\ j' \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} i \\ j \end{bmatrix}$$

- limitations
  - only perfectly nested loops
  - all statements are transformed the same

Loop Reversal

Idea

– Change the direction of loop iteration
(*i.e.,* From low-to-high indices to high-to-low indices or vice versa)

Benefits

– Could improve cache performance
– Enables other transformations

Example

```
do i = 6,1,-1
  A(i) = B(i) + C(i)
enddo
```

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


Loop Reversal and Distance Vectors

Impact

– Reversal of loop *i* negates the i^{th} entry of all distance vectors associated with the loop
– What about direction vectors?

When is reversal legal?

– When the loop being reversed does not carry a dependence
(*i.e.*, When the transformed distance vectors remain legal)

Example

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

$$\begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} i \\ j \end{bmatrix} = \begin{bmatrix} i \\ -j \end{bmatrix}$$

Dependence: Flow
Distance Vector: (1,1)
Transformed
Distance Vector: (1,-1) legal

Unimodular Framework

Does it have an affine component?

– Actually no. That moves us to the polyhedral model.

Why a unimodular matrix?

– Has an inverse and the inverse is unimodular.

– Preserves the volume of a polytope.

Transformations we can automate

–Loop permutation, skewing, and reversal

–Any combination of the above

Lexicographical Order Constraints in Data Dependencies

Iteration point

– Integer tuple with dimensionality d $(i_0, i_1, ..., i_d)$

Lexicographical Order

– First order the iteration points by i_0, then i_1, … and finally i_d.

$$(i_0, i_1, ..., i_{d-1}) \prec (i_0, i_1, ..., i_{d-1}) \equiv$$
$$(i_0 < j_0) \vee (i_0 = j_0 \wedge i_1 < j_1) \vee ...(i_0 = j_0 \wedge i_1 = j_1 \wedge ...i_{d-1} = j_{d-1})$$

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 at least one of the accesses is a write and

- `f(I) = g(J)`
- `(l1,...ln) < I,J < (h1,...,hn)`
- `I << J or J << I, where << is lexicographically less`

Direction/Dependence Vectors

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

Set up the dependence problem

Subtract the source from the target iteration

Specifying Loop Transformations, Various Frameworks

Unimodular

$$f(\vec{i}) = T\vec{i}$$

Polyhedral/Affine Transformations (also called a schedule or Change of Basis)

$$f(\vec{i}) = T\vec{i} + \vec{j}$$

Kelly and Pugh (Omega, Presburger in general case, also called Polyhedral)

$$\{\vec{i} \rightarrow \vec{x} | \vec{x} = T\vec{i} + \vec{j}\}$$

–As shown is same as polyhedral.

–More general in that can do Presburger arithmetic, which include forall and there exists quantifiers

Loop Fusion using Kelly and Pugh (Omega) Notation

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

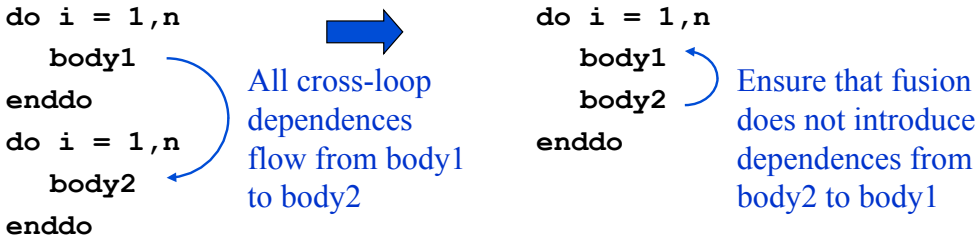
Cons

- May hurt data locality
- May hurt icache performance

Legality of Loop Fusion

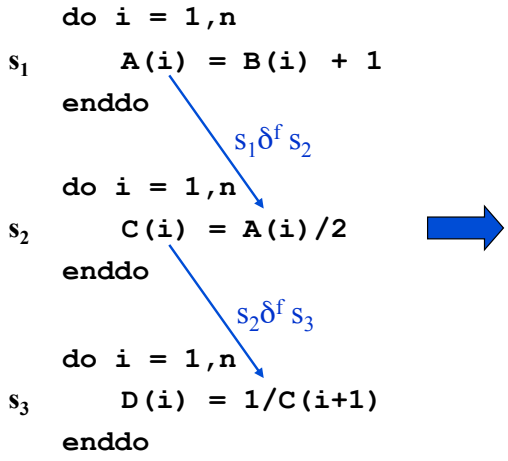
Basic Conditions

- Both loops must have same structure
  - Same loop depth
  - Same loop bounds
  - Same iteration directions
- Dependences must be preserved
  e.g., Flow dependences must not become anti dependences

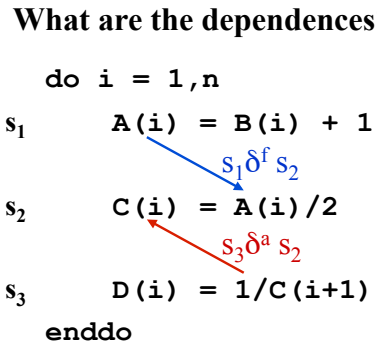


Loop Fusion Example

What are the dependences?



What are the dependences?



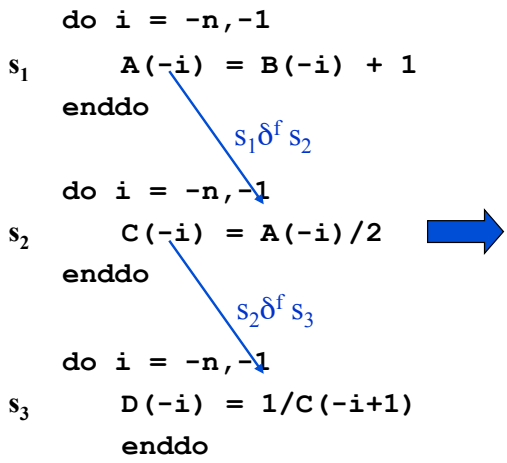
Fusion changes the dependence between s_2 and s_3, so fusion is illegal

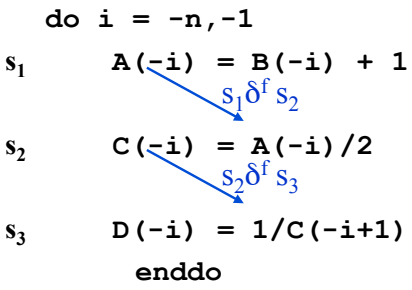
Is there some transformation that will enable fusion of these loops?

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 $s_2\delta^f s_3$





After reversal and fusion all original dependences are preserved

Kelly and Pugh Transformation Framework

Specify iteration space as a set of integer tuples

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

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

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

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

Loop Fusion Example

What are the dependences?

```
    do i = 1,n
s1      A(i) = B(i) + 1
    enddo
                 s1δf s2
    do i = 1,n
s2      C(i) = A(i)/2
    enddo
                 s2δf s3
    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
              s1δf s2
s2      C(i) = A(i)/2
              s3δa s2
s3      D(i) = 1/C(i+1)
    enddo
```

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?

Specifying Loop Fusion in Kelly and Pugh Framework

Specify iteration space as a set of integer tuples

$$IS_1 = \{[1, i_1, 1] \mid 1 \le i_1 \le n\}$$
$$IS_2 = \{[2, i_2, 1] \mid 1 \le i_2 \le n\}$$
$$IS_3 = \{[3, i_3, 1] \mid 1 \le i_3 \le n\}$$
$$IS = IS_1 \cup IS_2 \cup IS_3$$

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

$$D_{12} = \{[1, i_1, 1] \rightarrow [2, i_2, 1] \mid i_1 = i_2\}$$
$$D_{23} = \{[2, i_2, 1] \rightarrow [3, i_3, 1] \mid i_2 = i_3 + 1\}$$
$$D = D_{12} \cup D_{23}$$

Specify transformations as mappings between integer tuples

$$T_1 = \{[1, i_1, 1] \rightarrow [1, i_1', 1] \mid i_1' = i_1\}$$
$$T_2 = \{[2, i_2, 1] \rightarrow [1, i_2', 2] \mid i_2' = i_2\}$$
$$T_3 = \{[3, i_3, 1] \rightarrow [1, i_3', 3] \mid i_3' = i_3\}$$
$$T = T_1 \cup T_2 \cup T_3$$

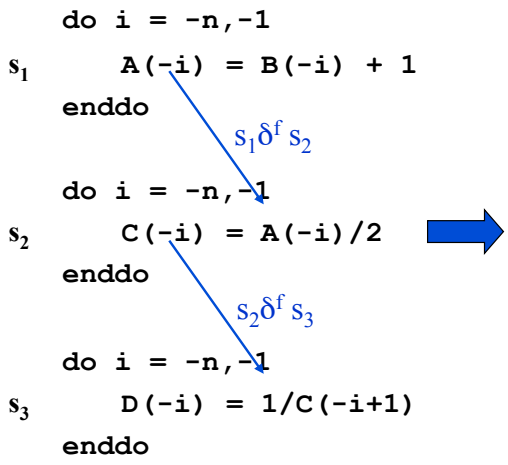
Checking Legality in Kelly & Pugh Framework

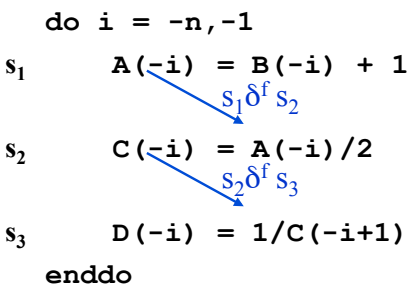
For each dependence, `[I] -> [J]` the transformed `I` iteration must be executed after the transformed `J` iteration.

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 $s_2 \delta^f s_3$





After reversal and fusion all original dependences are preserved

Fusion Example

Can we fuse these loop nests?

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

δ^f

Fusion of these loops would violate this dependence

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


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
```

δ^f

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

δ^f

Two different ways to deal with imperfect loop nesting

Add dimensionality to the schedule for statement order

- Approach taken by Kelly and Pugh

Embed all statements in same iteration space

- Use initial statement order as the tie breaker

Difference

- Has an effect on the scheduling algorithms
- Has an effect on the code generation algorithms

Next Time

Reading

- Advanced Compiler Optimizations for Supercomputers by Padua and Wolfe

Homework

- PA3 is due Monday November 2nd

Lecture

- Finish K&P framework and iscc tool

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?

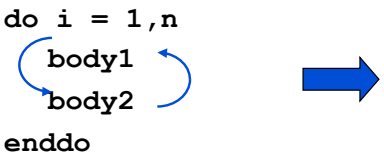
– Produces multiple (potentially) less constrained loops
– May improve locality
– Enable other transformations, such as interchange

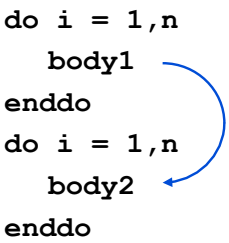
Legality?

Loop Fission (cont)

Legality

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

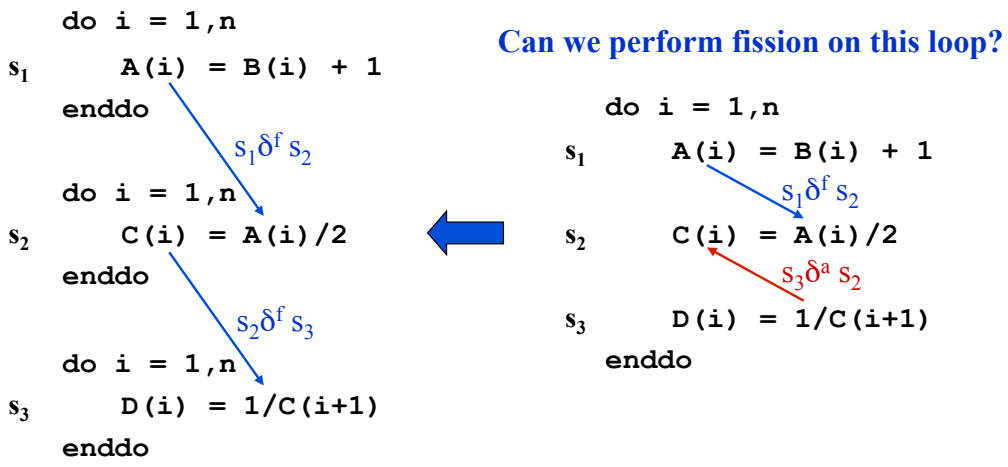




Cycles cannot be preserved because after fission all cross-loop dependences flow from body1 to body2

Loop Fission Example

Recall our fusion example



Loop Fission Example (cont)

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