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

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

- Dependencies must be preserved
  *e.g.*, Flow dependences must not become anti dependences

Can we relax any of these restrictions?

Loop Fusion Example

What are the dependences?

\[\begin{align*}
&\text{do } i = 1, n \\
&s_1 \quad A(i) = B(i) + 1 \\
&\text{endo} \\
&s_2 \quad C(i) = A(i)/2 \\
&\text{endo} \\
&s_3 \quad D(i) = 1/C(i+1) \\
&\text{endo}
\end{align*}\]

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: \( s_3 \delta^f s_2 \) will become \( s_2 \delta^f s_3 \)

```
    do i = n,1
    s_1 A(i) = B(i) + 1
    enddo  \( s_1 \delta^f s_2 \)

    do i = n,1
    s_2 C(i) = A(i)/2
    enddo  \( s_2 \delta^f s_3 \)

    do i = n,1
    s_3 D(i) = 1/C(i+1)
    enddo
```

Do i = n,1,-1

```
    do i = n,1
    s_1 A(i) = B(i) + 1
    \( s_1 \delta^f s_2 \)

    do i = n,1
    s_2 C(i) = A(i)/2
    \( s_2 \delta^f s_3 \)

    \( \delta^f \)

    do i = n,1
    s_3 D(i) = 1/C(i+1)
    enddo
```

After reversal and fusion all original dependences are preserved

## Kelly and Pugh Transformation Framework

Specify iteration space as a set of integer tuples

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

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

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

Specify transformations as relations/mappings between integer tuples

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

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: $s_1 \delta s_2$ will become $s_2 \delta s_3$

```
! do i = n,1,-1
  s_1 A(i) = B(i) + 1
  ! do i = n,1,-1
    s_1 A(i) = B(i) + 1
    s_1 \delta s_2
  ! enddo
  s_1,\delta s_2

! do i = n,1,-1
  s_2 C(i) = A(i)/2
  ! do i = n,1,-1
    s_2 C(i) = A(i)/2
    s_2 \delta s_3
  ! enddo
  s_2,\delta s_3

! do i = n,1,-1
  s_3 D(i) = 1/C(i+1)
  ! enddo
  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
  do j = 1,n
    do k = 1,n
      X(k) = X(k) + A(k,i) * Y(i)
    enddo
  enddo
endo
```

Fusion of these loops would violate this dependence
Fusion Example (cont)

Use loop interchange to preserve dependences

\[
\begin{align*}
&\text{do } i = 1, n \\
&\quad ! \ X(i) = 0 \\
&\text{enddo} \\
&\text{do } k = 1, n \\
&\quad \text{do } j = 1, n \\
&\quad \quad X(k) = X(k) + A(k, j) \cdot Y(j) \\
&\quad \text{enddo} \\
&\text{enddo}
\end{align*}
\]

Loop Fission (Loop Distribution)

Idea

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

Example

\[
\begin{align*}
&\text{do } i = 1, n \\
&\quad A(i) = B(i) + 1 \\
&\quad C(i) = A(i)/2 \\
&\text{enddo}
\end{align*}
\]

Motivation?

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

Legality?

\[
\begin{align*}
&\text{do } i = 1, n \\
&\quad A(i) = B(i) + 1 \\
&\text{enddo}
\end{align*}
\]

\[
\begin{align*}
&\text{do } i = 1, n \\
&\quad C(i) = A(i)/2 \\
&\text{enddo}
\end{align*}
\]
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

Can we perform fission on this loop?

Cycles cannot be preserved because after fission all cross-loop dependences flow from body1 to body2
**Loop Fission Example (cont)**

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

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

Can we perform fission on this loop?

```plaintext
! do i = 1,n
  k1 A(i) = B(i) + 1
  k2 C(i) = A(i)/2
  k3 D(i) = 1/C(i+1)
! enddo
```

---

**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

\[
\begin{align*}
\text{do } i &= 1, n \\
A(i) &= B(i) + C(i) \\
\text{enddo}
\end{align*}
\]

\[
\begin{align*}
\text{do } i &= 1, n \text{ by 2} \\
A(i) &= B(i) + C(i) \\
A(i+1) &= B(i+1) + C(i+1) \\
\text{enddo}
\end{align*}
\]

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

\[
\begin{align*}
\text{!do } j &= 1, 2*n \\
\! \text{do } i &= 1, m \\
\! A(j) &= A(j) + B(i) \\
\! \text{enddo}
\end{align*}
\]

- !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
```

Unroll and Jam Example (cont)

```
Jam the inner loops
! 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
```

- 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
**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