Loop Transformations, Dependencies, and Parallelization

Announcements
– Quiz 1 is on RamCT and is due Friday night
– HW1 is due Wednesday February 8th

Today
– Loop permutation
– Data dependences
– Loop parallelization
– Loop skewing

Loop Permutation for Improved Locality

Sample code: Assume Fortran’s Column Major Order array layout

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

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

i
j
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
16 17 18 19 20 21 22 23 24 25
26 27 28 29 30

poor cache locality

i
j
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
16 17 18 19 20 21 22 23 24 25
26 27 28 29 30

good cache locality
Loop Permutation Another Example

Idea
- Swap the order of two loops to increase parallelism, to improve spatial locality, or to enable other transformations
- Also known as loop interchange

Example

```
do i = 1,n
do j = 1,n
  x = A(2,j)
enddo
```

This access strides through a row of A

```
do j = 1,n
do i = 1,n
  x = A(2,j)
enddo
```

This code is invariant with respect to the inner loop, yielding better locality

Loop Permutation Legality

Sample code

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

Why is this legal?
- No loop-carried dependences, so we can arbitrarily change order of iteration execution
- Does the loop always have to have NO inter-iteration dependences for loop permutation to be legal?
Data Dependences

Recall
- A data dependence defines ordering relationship two between statements
- In executing statements, data dependences must be respected to preserve correctness

Example

\[
\begin{align*}
{s_1} & \quad a := 5; & \quad ? \\
{s_2} & \quad b := a + 1; & \quad s_3 \quad a := 6; \\
{s_3} & \quad a := 6; & \quad s_2 \quad b := a + 1;
\end{align*}
\]

Dependences and Loops

Loop-independent dependences

\[
\begin{align*}
do \quad i = 1,100 \\
A(i) & = B(i)+1 \\
C(i) & = A(i) \times 2
enddo
\]

Dependences within the same loop iteration

Loop-carried dependences

\[
\begin{align*}
do \quad i = 1,100 \\
A(i) & = B(i)+1 \\
C(i) & = A(i-1) \times 2
enddo
\]

Dependences that cross loop iterations
Data Dependence Terminology

We say statement $s_2$ depends on $s_1$

- **True (flow) dependence**: $s_1$ writes memory that $s_2$ later reads
- **Anti-dependence**: $s_1$ reads memory that $s_2$ later writes
- **Output dependences**: $s_1$ writes memory that $s_2$ later writes
- **Input dependences**: $s_1$ reads memory that $s_2$ later reads

**Notation:** $s_1 \delta s_2$

- $s_1$ is called the *source* of the dependence
- $s_2$ is called the *sink* or *target*
- $s_1$ must be executed before $s_2$

Yet Another Loop Permutation Example

Consider another example

\[
\begin{align*}
\text{Before} & \quad (1,1) \quad C(1,1) = C(2,0) \\
& \quad (1,2) \quad C(1,2) = C(2,1) \\
& \quad \ldots \\
& \quad (2,1) \quad C(2,1) = C(3,0)
\end{align*}
\]

\[
\begin{align*}
\text{After} & \quad (1,1) \quad C(1,1) = C(2,0) \\
& \quad (2,1) \quad C(2,1) = C(3,0)
\end{align*}
\]
Data Dependences and Loops

How do we identify dependences in loops?

```verbatim
do i = 1, 5
    A(i) = A(i-1) + 1
endo
```

Simple view
– Imagine that all loops are fully unrolled
– Examine data dependences as before

Problems
– Impractical and often impossible
– Lose loop structure

Iteration Spaces

Idea
– Explicitly represent the iterations of a loop nest

Example
```verbatim
do i = 1, 6
    do j = 1, 5
        A(i, j) = A(i-1, j-1) + 1
    enddo
endo
doneto
```

Iteration Space
– A set of tuples that represents the iterations of a loop
– Can visualize the dependences in an iteration space
**Distance Vectors**

**Idea**
- Concisely describe dependence relationships between iterations of an iteration space.
- For each dimension of an iteration space, the distance is the number of iterations between accesses to the same memory location.

**Definition**
- $v = i^T - i^S$

**Example**

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

**Distance Vector:** (1,2)

**Distance Vectors and Loop Transformations**

**Idea**
- Any transformation we perform on the loop must respect the dependences.

**Example**

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

**Can we permute the i and j loops?**
Distance Vectors and Loop Transformations

Idea
– Any transformation we perform on the loop must respect the dependences

Example

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

Can we permute the \(i\) and \(j\) loops?
– Yes

Distance Vectors: Legality

Definition
– A dependence vector, \(v\), is lexicographically nonnegative when the left-most 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?
**Example where permutation is not legal**

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

Kind of dependence: Anti

Distance vector: (1, -1)
Loop-Carried Dependences

Definition
– A dependence $D = (d_1, ..., d_n)$ is carried at loop level $i$ if $d_i$ is the first nonzero element of $D$

Example
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:
- (0,1) for accesses to $A$
- (1,0) for accesses to $B$

Loop-carried dependences
- The $j$ loop carries dependence due to $A$
- The $i$ loop carries dependence due to $B$

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

Direction vector: $(<, <)$
Distance vector: $(1, 1)$
**Legality of Loop Permutation**

**Case analysis of the direction vectors**

\( (\leq,\leq) \)

The dependence is loop independent, so it is unaffected by permutation.

\( (\leq,\lt) \)

The dependence is carried by the \( j \) loop. After permutation the dependence will be \((\lt,\leq)\), so the dependence will still be carried by the \( j \) loop, so the dependence relations do not change.

\( (\lt,\leq) \)

The dependence is carried by the \( i \) loop. After permutation the dependence will be \((\leq,\lt)\), so the dependence will still be carried by the \( i \) loop, so the dependence relations do not change.

**Legality of Loop Permutation (cont)**

**Case analysis of the direction vectors (cont.)**

\( (\lt,\lt) \)

The dependence distance is positive in both dimensions. After permutation it will still be positive in both dimensions, so the dependence relations do not change.

\( (\lt,\gt) \)

The dependence is carried by the outer loop. After interchange the dependence will be \((\gt,\lt)\), which changes the dependences and results in an illegal direction vector, so interchange is illegal.

\( (\gt,\ast) \) \( (\leq,\gt) \)

Such direction vectors are not possible for the original loop.
**Loop Permutation Example**

Consider the $<,>$ case

\[
\begin{align*}
do & \ i = 1, n \\
do & \ j = 1, n \\
C(i, j) & = C(i+1, j-1) \\
\end{align*}
\]

Before

\[
\begin{array}{c}
(1,1) \quad C(1,1) = C(2,0) \\
(1,2) \quad C(1,2) = C(2,1) \\
\vdots \\
(2,1) \quad C(2,1) = C(3,0) \\
\end{array}
\]

After

\[
\begin{array}{c}
(1,1) \quad C(1,1) = C(2,0) \\
(2,1) \quad C(2,1) = C(3,0) \\
\vdots \\
(1,2) \quad C(1,2) = C(2,1) \\
\end{array}
\]

**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**

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

**Distance vectors:**

\[
\begin{array}{c}
(0,1) \text{ for accesses to } A \\
(1,0) \text{ for accesses to } B \\
\end{array}
\]

**Loop-carried dependences**

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

**Idea**
- Each iteration of a loop may be executed in parallel if that loop carries no dependences

**Example (different from last slide)**
```plaintext
do j = 1, 5
do i = 1, 6
   A(i, j) = B(i-1, j-1)+1
   B(i, j) = A(i, j-1)*2
enddo
dendo
```

**Parallelize i loop?**
- Distance Vectors:
  - (1,0) for A (flow)
  - (1,1) for B (flow)

**Loop Skewing**

**Original code**
```plaintext
do i = 1, 6
do j = 1, 5
   A(i, j) = A(i-1, j+1)+1
enddo
dendo
```

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

**Can we permute or parallelize the original loop?**

**What about after skewing?**
```plaintext
do i' = 1, 6
do j' = i', i'+5-1
   A(i', j'-i'+1)=A(i'-1, j'-i'+2)+1
enddo
dendo
```
Protein String Matching Example (smithWaterman.c)

```c
for (i=1;i<=a[0];i++) {
    for (j=1;j<=b[0];j++) {
        diag    = h[i-1][j-1] + sim[a[i]][b[j]];
        down    = h[i-1][j]   + DELTA;
        right   = h[i][j-1]   + DELTA;
        max=MAX3(diag,down,right);
        if (max <= 0)  {
            h[i][j]=0; xTraceback[i][j]=-1; yTraceback[i][j]=-1;
        } else if (max == diag) {
            h[i][j]=diag; xTraceback[i][j]=i-1; yTraceback[i][j]=j-1;
        } else if (max == down) {
            h[i][j]=down; xTraceback[i][j]=i-1; yTraceback[i][j]=j;
        } else  {
            h[i][j]=right; xTraceback[i][j]=i; yTraceback[i][j]=j-1;
        }
        if (max > Max){
            Max=max; xMax=i; yMax=j;
        }
    }
}  // end for loops
```

Loop-Carried, Storage-Related Dependences

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

**Example**

```c
do i = 1,6
   t = A(i) + B(i)
   C(i) = t + 1/t
endo
```

**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
Removing False Dependences with Scalar Expansion

Idea

– Eliminate false dependences by introducing extra storage

Example

\[
\begin{align*}
d\ i = 1, 6 \\
T(i) &= A(i) + B(i) \\
C(i) &= T(i) + 1/T(i)
\end{align*}
\]

\[
t = T[6]
\]

Can \textit{this} loop be parallelized?

Disadvantages?

Scalar Expansion Details

Restrictions

– The loop must be a \textbf{countable} loop
  \textit{i.e.} The loop trip count must be independent of the body of the loop
– The expanded scalar must have no \textbf{upward exposed uses} in the loop

\[
\begin{align*}
d\ i = 1, 6 \\
\text{print}(t) \\
t &= A(i) + B(i) \\
C(i) &= t + 1/t
\end{align*}
\]

– 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
**Concepts**

**Loop Transformations**
- Loop permutation and when it is applicable
- Data dependences including distance vectors, loop carried dependences, and direction vectors
- Loop parallelization
  - Loop carried dependences
  - When loop parallelization is directly applicable
  - When it is not directly applicable
  - Scalar expansion

**Next Time**

**Reading**
- Read: *Automatic Parallelization in the Polytope Model* by Feautrier

**Homework**
- Quiz 1 is due tomorrow night
- HW1 is due Wednesday 2/8/12

**Lecture**
- Automating Parallelization and Performance Optimization of Applications