Loop Transformations and Locality
Agenda

- Introduction
- Loop Transformations
- Affine Transform Theory
Memory Hierarchy
Cache Locality

• Suppose array A has column-major layout


```
for i = 1, 100
  for j = 1, 200
  end_for
end_for
```

• Loop nest has **poor** spatial cache locality.
Loop Interchange

• Suppose array A has column-major layout

|--------|--------|-----|--------|--------|-----|--------|-----|

```
for i = 1, 100
  for j = 1, 200
  end_for
end_for
```

```
for j = 1, 200
  for i = 1, 100
  end_for
end_for
```

• New loop nest has **better** spatial cache locality.
Interchange Loops?

for i = 2, 100
    for j = 1, 200
    end_for
end_for

• e.g. dependence from (3,3) to (4,2)
Dependence Vectors

- Distance vector $(1, -1) = (4, 2) - (3, 3)$
- Direction vector $(+, -)$ from the signs of distance vector
- Loop interchange is not legal if there exists dependence $(+, -)$
Agenda

- Introduction
- Loop Transformations
- Affine Transform Theory
Loop Fusion

for i = 1, 1000
    A[i] = B[i] + 3
end_for

for j = 1, 1000
end_for

Better reuse between A[i] and A[i]

for i = 1, 1000
    A[i] = B[i] + 3
end_for

for i = 1, 1000
    C[i] = A[i] + 5
end_for
Loop Distribution

for i = 1, 1000
end_for

for i = 1, 1000
  C[i] = B[i] + 5
end_for

2nd loop is parallel
Register Blocking

for \( j = 1, 2m \)
\[
\text{for } i = 1, 2n
\]
\[
\]
\[
+ A[i-1, j-1]
\]
end_for
end_for

for \( j = 1, 2m, 2 \)
\[
\text{for } i = 1, 2n, 2
\]
\[
\]
\[
\]
\[
\]
\[
\]
end_for
end_for

- Better reuse between \( A[i,j] \) and \( A[i,j] \)
Virtual Register Allocation

for j = 1, 2*M, 2
  for i = 1, 2*N, 2
    r1 = A[i-1,j]
    r2 = r1 + A[i-1,j-1]
    A[i, j] = r2
    r3 = A[i-1,j+1] + r1
    A[i, j+1] = r3
    A[i+1, j] = r2 + A[i, j-1]
    A[i+1, j+1] = r3 + r2
  end_for
end_for

- Memory operations reduced to register load/store
- 8MN loads to 4MN loads
Scalar Replacement

- Eliminate loads and stores for array references

```
for i = 2, N+1
    = A[i-1]+1
A[i] =
end_for
```

```
t1 = A[1]
for i = 2, N+1
    = t1 + 1
    t1 =
    A[i] = t1
end_for
```
Unroll-and-Jam

for \( j = 1, 2*M \)
    for \( i = 1, N \)
    end_for
end_for

for \( j = 1, 2*M, 2 \)
    for \( i = 1, N \)
    end_for
end_for

- Expose more opportunity for scalar replacement
• Suppose arrays A and B have row-major layout

```c
for i = 1, 1000
  for j = 1, 1000
  end_for
end_for
```

- B has poor cache locality.
- Loop interchange will not help.
Loop Blocking

for v = 1, 1000, 20
    for u = 1, 1000, 20
        for j = v, v+19
            for i = u, u+19
            end_for
        end_for
    end_for
end_for

- Access to small blocks of the arrays has good cache locality.
Loop Unrolling for ILP

```
for i = 1, 10
    a[i] = b[i];
    *p = ...
end_for
```

```
for I = 1, 10, 2
    a[i] = b[i];
    *p = ...
    a[i+1] = b[i+1];
    *p = ...
end_for
```

- Large scheduling regions. Fewer dynamic branches
- Increased code size
Agenda

- Introduction
- Loop Transformations
- *Affine Transform Theory*
Objective

- Unify a large class of program transformations.
- Example:

```c
float Z[100];
for i = 0, 9
    Z[i+10] = Z[i];
end_for
```
A d-deep loop nest has d index variables, and is modeled by a d-dimensional space. The space of iterations is bounded by the lower and upper bounds of the loop indices.

Iteration space i = 0, 1, …… 9

```
for i = 0, 9
    Z[i+10] = Z[i];
end_for
```
The iterations in a d-deep loop nest can be represented mathematically as

\[ \{ \vec{i} \in \mathbb{Z}^d \mid B\vec{i} + b \geq 0 \} \]

- \( \mathbb{Z} \) is the set of integers
- \( B \) is a \( d \times d \) integer matrix
- \( b \) is an integer vector of length \( d \), and
- \( 0 \) is a vector of \( d \) 0’s.
Example

for i = 0, 5
for j = i, 7
Z[j,i] = 0;

E.g. the 3rd row –i+j ≥ 0 is from the lower bound j ≥ i for loop j.
Symbolic Constants

for $i = 0, n$

$Z[i] = 0$;

E.g. the 1st row $-i+n \geq 0$ is from the upper bound $i \leq n$. 
Data Space

- An n-dimensional array is modeled by an n-dimensional space. The space is bounded by the array bounds.
- Data space a = 0, 1, ... 99

```c
float Z[100]
for i = 0, 9
    Z[i+10] = Z[i];
end_for
```
Initially assume unbounded number of virtual processors (vp1, vp2, …) organized in a multi-dimensional space.

(Iteration 1, vp1), (Iteration 2, vp2), …

After parallelization, map to physical processors (p1, p2).

(vp1, p1), (vp2, p2), (vp3, p1), (vp4, p2), …
Affine Array Index Function

- Each array access in the code specifies a mapping from an iteration in the iteration space to an array element in the data space.
- Both i+10 and i are affine.

```c
float Z[100]
for i = 0, 9
    Z[i+10] = Z[i];
end_for
```
Array Affine Access

- The bounds of the loop are expressed as affine expressions of the surrounding loop variables and symbolic constants, and
- The index for each dimension of the array is also an affine expression of surrounding loop variables and symbolic constants
Matrix Formulation

- Array access maps a vector $i$ within the bounds to array element location $F_i + f$.

$$\{ \vec{i} \in \mathbb{Z}^d | B\vec{i} + b \geq 0 \}$$

- E.g. access $X[i-1]$ in loop nest $i,j$

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

- An affine function to assign iterations in an iteration space to processors in the processor space.
- E.g. iteration i to processor 10-i.

```c
float Z[100]
for i = 0, 9
  Z[i+10] = Z[i];
end_for
```
Data Access Region

- An affine function to assign iterations in an iteration space to processors in the processor space.
- Region for $Z[i+10]$ is $\{a \mid 10 \leq a \leq 20\}$.

```plaintext
float Z[100]
for i = 0, 9
    Z[i+10] = Z[i];
end_for
```
Data Dependences

Solution to linear constraints as shown in the last lecture.

- There exist $i_r$, $i_w$, such that
- $0 \leq i_r, i_w \leq 9$, 
- $i_w + 10 = i_r$

```c
float Z[100]
for i = 0, 9 
    Z[i+10] = Z[i];
end_for
```
Affine Transform

\[ \begin{bmatrix} u \\ v \end{bmatrix} = B \begin{bmatrix} i \\ j \end{bmatrix} + b \]
Locality Optimization

for i = 1, 100
  for j = 1, 200
  end_for
end_for

for u = 1, 200
  for v = 1, 100
  end_for
end_for
for i = 1, 100
  for j = 1, 200
  end_for
end_for

\[
\begin{bmatrix}
  u \\
  v
\end{bmatrix} =
\begin{bmatrix}
  0 & 1 \\
  1 & 0
\end{bmatrix}
\begin{bmatrix}
  i \\
  j
\end{bmatrix}
\]

\[
\begin{bmatrix}
  1 & 0 & -1 & 0 \\
  -1 & 0 & 100 & 0 \\
  0 & 1 & -1 & 0 \\
  0 & -1 & 200 & 0
\end{bmatrix}
\begin{bmatrix}
  i \\
  j
\end{bmatrix} \geq \begin{bmatrix}
  0 \\
  0 \\
  0 \\
  0
\end{bmatrix}
\]
New Iteration Space

\[
\begin{bmatrix}
1 & 0 \\
-1 & 0 & 1 \\
0 & 1 & 1 & 0 \\
0 & -1 \\
0 & -1 & u & v \\
0 & 1 & u & v \\
1 & 0 & v \\
-1 & 0 & v \\
\end{bmatrix}
\begin{bmatrix}
-1 \\
100 \\
-1 \\
200 \\
-1 \\
100 \\
-1 \\
200 \\
\end{bmatrix}
\begin{bmatrix}
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
\end{bmatrix}
\geq
\begin{bmatrix}
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
\end{bmatrix}
\]

for \( u = 1, 200 \)
for \( v = 1, 100 \)

\[
\]
end_for
end_for
for $i = 1, 100$
for $j = 1, 200$
end_for
end_for

$A[\begin{bmatrix} 1 & 0 \end{bmatrix}_i, \begin{bmatrix} 0 & 1 \end{bmatrix}_j]$
for $u = 1, 200$
for $v = 1, 100$
end_for
end_for
Interchange Loops?

for i = 2, 1000
for j = 1, 1000
end_for
end_for

- e.g. dependence vector \( d_{old} = (1, -1) \)
A transformation is legal, if the new dependence is lexicographically positive, i.e., the leading non-zero in the dependence vector is positive.
Summary

- Locality Optimizations
- Loop Transformations
- Affine Transform Theory