15-745
Optimizing For Data Locality - 2
Seth Copen Goldstein
Seth@cs.cmu.edu
CMU
Based on "A Data Locality Optimizing Algorithm, Wolf & Lam, PLDI '91

Outline

- Loop Transformations
  - dependence vectors
  - Transformations
    - Unimodular transformations
- Tiling
- SRP

Loop Transformation Theory

- Iteration Space
- Dependence vectors
- Unimodular transformations

Loop Nests and the Iter space

- General form of tightly nested loop

  for I_1 := low_1 to high_1 by step_1
  for I_2 := low_2 to high_2 by step_2
  ... for I_i := low_i to high_i by step_i
  ... for I_n := low_n to high_n by step_n
      Stmts

- The iteration space is a convex polyhedron in $Z^n$ bounded by the loop bounds.
- Each iteration is a node in the polyhedron identified by its vector: p=(p_1, p_2, ..., p_n)
**Lexicographical Ordering**

- Iterations are executed in lexicographic order.
- for \( p=(p_1, p_2, \ldots, p_n) \) and \( q=(q_1, q_2, \ldots, q_n) \)
  if \( p \preceq_k q \) iff for \( 1 \leq k \leq n \),
    \[ \forall 1 \leq i < k, (p_i = q_i) \text{ and } p_k > q_k \]
- For MM:
  - \((1,1,1), (1,1,2), (1,1,3), \ldots, (1,2,1), (1,2,2), (1,2,3), \ldots, (2,1,1), (2,1,2), (2,1,3), \ldots\)
  - \((1,2,1) \succ_2 (1,1,2), (2,1,1) \succ_1 (1,4,2), \text{ etc.}\)

**Dependence Vectors**

- Dependence vector in an n-nested loop is denoted as a vector: \( d=(d_1, d_2, \ldots, d_n) \).
- Each \( d_i \) is a possibly infinite range of ints in \( [d_i^{\min}, d_i^{\max}] \), where
  \[ d_i^{\min} \in \mathbb{Z} \cup \{-\infty\}, d_i^{\max} \in \mathbb{Z} \cup \{\infty\} \text{ and } d_i^{\min} \leq d_i^{\max} \]
- So, a single dep vector represents a set of distance vectors.
- A distance vector defines a distance in the iteration space.
- A dependence vector is a distance vector if each \( d_i \) is a singleton.

**Other defs**

- Common ranges in dependence vectors
  - \([1, \infty]\) as + or >
  - \([-\infty, -1]\) as - or <
  - \([-\infty, \infty]\) as ± or *

- A distance vector is the difference between the target and source iterations (for a dependent ref), e.g.,
  \[ d = I_t - I_s \]

**Examples**

\[ \text{for } I_1 := 1 \text{ to } n \]
\[ \text{for } I_2 := 1 \text{ to } n \]
\[ \text{for } I_3 := 1 \text{ to } n \]
\[ C[I_1, I_3] \leftarrow A[I_1, I_2] \ast B[I_2, I_3] \]

\[ \text{for } I_1 := 0 \text{ to } 5 \]
\[ \text{for } I_2 := 0 \text{ to } 6 \]

\[ \text{D}=((0,1),(1,0),(1-1)) \]
Plausible Dependence vectors

- A dependence vector is plausible iff it is lexicographically non-negative.
- All sequential programs have plausible dependence vectors. Why?
  - Plausible: (1,-1)
  - Implausible: (-1,0)

Loop Transforms

- A loop transformation changes the order in which iterations in the iteration space are visited.
- For example, Loop Interchange

Unimodular Transforms

- Interchange: permute nesting order
- Reversal: reverse order of iterations
- Skewing: scale iterations by an outer loop index

Interchange

- Change order of loops
- For some permutation \( p \) of \( 1 \ldots n \)

When is this legal?
Transform and matrix notation

- If dependences are vectors in iteration space, then transforms can be represented as matrix transforms.
- E.g., for a 2-deep loop, interchange is:
  \[
  T = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \quad [0 1]p_1 = [p_2] \\
  \]
- Since, \( T \) is a linear transform, \( Td \) is transformed dependence:
  \[
  \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} d_1 \\ d_2 \end{bmatrix} = \begin{bmatrix} d_2 \\ d_1 \end{bmatrix}
  \]

Reversal

- Reversal of \( i \)th loop reverses its traversal, so it can be represented as:
  \[
  T = \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix} \quad [0 1]p_1 = [-p_1] \\
  \]
  Diagonal matrix with \( i \)th element = -1.

- For 2 deep loop, reversal of outermost is:
  \[
  T = \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix} \quad [0 1]p_1 = [-p_1] \\
  \]

Skewing

- Skew loop \( I_j \) by a factor \( f \) w.r.t. loop \( I_i \) maps
  \[
  (p_1, ..., p_i, ..., p_j, ...) \rightarrow (p_1, ..., p_i, ..., p_j + fp_i, ...) \\
  \]
- Example for 2D
  \[
  T = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} \quad [1 0]p_1 = [p_1] \\
  \]
  \[
  \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} p_1 \\ p_2 \end{bmatrix} = \begin{bmatrix} p_1 \\ p_2 + p_1 \end{bmatrix}
  \]
Loop Skewing Example

for $I_1 := 0$ to 5
for $I_2 := 0$ to 6

$D = \{(0,1),(1,0),(1,1)\}$

But...is the transform legal?

- Distance/direction vectors give a partial order among points in the iteration space
- A loop transform changes the order in which 'points' are visited
- The new visit order must respect the dependence partial order!

But...is the transform legal?

- Loop reversal ok?
- Loop interchange ok?
But...is the transform legal?

• What other visit order is legal here?

for i = 0 to TS
for j = 0 to N-2
A[j+1] =

But...is the transform legal?

• Skewing...

for i = 0 to TS
for j = 0 to N-2
A[j+1] =

But...is the transform legal?

• Skewing...now we can block

for i = 0 to TS
for j = 0 to N-2
A[j+1] =
But...is the transform legal?

- Skewing...now we can loop interchange

Unimodular transformations

- Express loop transformation as a matrix multiplication
- Check if any dependence is violated by multiplying the distance vector by the matrix - if the resulting vector is still lexicographically positive, then the involved iterations are visited in an order that respects the dependence.

### Transformation Matrices

- **Reversal**: $\begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$
- **Interchange**: $\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$
- **Skew**: $\begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}$

Adjusting loop bounds

- Transformation on iteration space must be reflected in code.
- Since unimodular transforms are all linear, we can easily rewrite code.
- Bounds
- Indices

Goal of SRP

- Use Skewing, Reversal, and permutation to find a fully permutable inner loop nest that minimizes the accesses/iteration.
- Tile the inner loop to turn the reuse into locality.
**Fully Permutable**

- Loops $I_i$ through $I_j$ are fully permutable iff
  - All dependence vectors are lex positive
  - For each dependence vector $d$
    - $(d_1, ..., d_{i-1})$ is lex positive or
    - $i \leq k \leq j, d_k \geq 0$

**SRP**

- Identify loops that carry reuse
- Identify loops that can be in the localized vector space
- From this set, $I$:
  - look at all subsets which can be made fully permutable inner loop nests
  - and remaining loops are legal outermost loops
- Pick subset which minimizes accesses/iter
- Tile inner subset

**Tiling**

- Tiling a perfect fully permutable $L_1 \ldots L_m$
  - Aka blocking
  - Aka strip-mine and interchange
- Foreach $L_k : 1 \leq k \leq m$
  - Assume has form: for $(i=L, i<U; i+=S)$
  - Create controlling loop
    - for $(ii=L; ii<U; ii+=(S*B))$
  - Rewrite original loop as
    - for $(i=ii; i<\text{MIN}(i+B*S-S, U); i+=S)$