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 I1 := low1 to high1 by step1
   for I2 := low2 to high2 by step2
      …
       for Ii := lowi to highi by stepi
        …
         for In := lown to highn by stepn
            Stmts
```

- The iteration space is a convex polyhedron in $\mathcal{Z}^n$ bounded by the loop bounds.
- Each iteration is a node in the polyhedron identified by its vector: $\mathbf{p}=(p_1, p_2, \ldots, p_n)$

Lexicographical Ordering

- Iterations are executed in lexicographic order.
- for **p**=(p_1, p_2, …, p_n) and **q**=(q_1, q_2, …, q_n) if **p** $\succ_k$ **q** iff for $1 \le k \le n$,

  $\forall\ 1 \le i < k$, ($p_i = q_i$) and $p_k > q_k$
- For MM:
  - (1,1,1), (1,1,2), (1,1,3), …,
    (1,2,1), (1,2,2), (1,2,3), …,
    …,
    (2,1,1), (2,1,2), (2,1,3), …
  - (1,2,1) $\succ_2$ (1,1,2), (2,1,1) $\succ_1$ (1,4,2), etc.

Dependence Vectors

- Dependence vector in an n-nested loop is denoted as a vector: **d**=(d_1, d_2, …, d_n).
- Each d_i is a possibly infinite range of ints in $\lfloor d_i^{\min}, d_i^{\max} \rfloor$, where

  $$d_i^{\min} \in Z \cup \{-\infty\}, d_i^{\max} \in Z \cup \{\infty\} \text{ and } d_i^{\min} \le d_i^{\max}$$
- 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, ∞] as + or >
  - [- ∞, -1] as – or <
  - [- ∞, ∞] as ± or *
- *A distance vector is the difference between the target and source iterations (for a dependent ref), e.g.,*

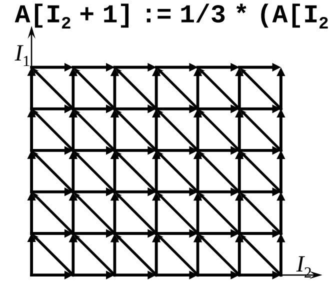
  $$\mathbf{d} = \mathbf{I}_t - \mathbf{I}_s$$

Examples

```
for I1 := 1 to n
   for I2 := 1 to n
      for I3 := 1 to n
         C[I1,I3] += A[I1,I2] * B[I2,I3]
```

(0,1,0)

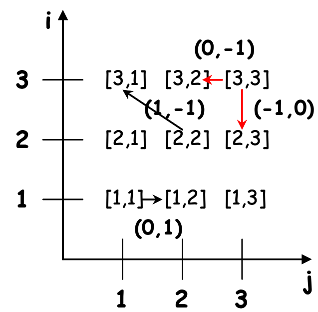
```
for I1 := 0 to 5
  for I2 := 0 to 6
    A[I2 + 1] := 1/3 * (A[I2] + A[I2 + 1] + A[I2 + 2])
```



D={(0,1),(1,0),(1-1)}

Plausible Dependence vectors

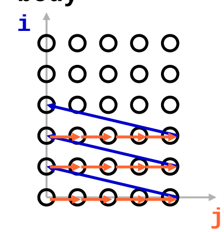
- A dependence vector is plausible iff it is lexicographically non-negative.
- All sequential programs have plausible dependence vectors. Why?
- Plausible: (1,-1)
- implausible (-1,0)



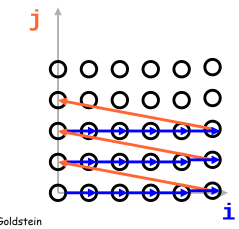
Loop Transforms

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

```
for i := 0 to n
  for j := 0 to m
    body
```




```
for j := 0 to m
  for i := 0 to n
    body
```



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 … n

for I_1 := …
  for I_2 := …
  …
    for I_n := …
      body

$\Rightarrow$

for $I_{p(1)}$:= …
  for $I_{p(2)}$:= …
  …
    for $I_{p(n)}$:= …
      body

- When is this legal?

Transform and matrix notation

- If dependences are vectors in iter space, then transforms can be represented as matrix transforms
- E.g., for a 2-deep loop, interchange is:

$$T = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \qquad \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} p_1 \\ p_2 \end{bmatrix} = \begin{bmatrix} p_2 \\ p_1 \end{bmatrix}$$

- Since, T is a linear transform, T**d** 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:

Reversal

- Reversal of i^{th} loop reverses its traversal, so it can be represented as: Diagonal matrix with i^{th} element = -1.

- For 2 deep loop, reversal of outermost is:

$$T = \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix} \qquad \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} p_1 \\ p_2 \end{bmatrix} = \begin{bmatrix} -p_1 \\ p2 \end{bmatrix}$$

Skewing

- Skew loop I_j by a factor f w.r.t. loop I_i maps

$$(p_1,\ldots,p_i,\ldots,p_j,\ldots) \qquad (p_1,\ldots,p_i,\ldots,p_j + fp_i,\ldots)$$

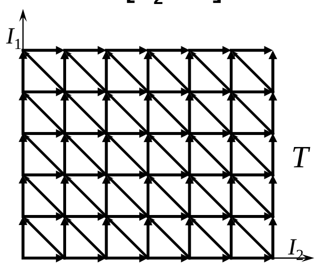
- Example for 2D

$$T = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} \qquad \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} p_1 \\ p_2 \end{bmatrix} = \begin{bmatrix} p_1 \\ p_2 + p_1 \end{bmatrix}$$

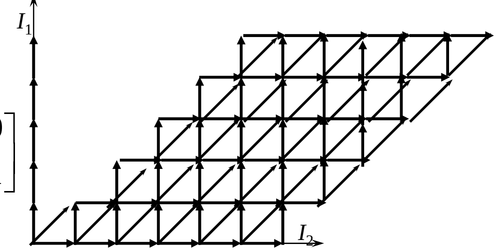
Loop Skewing Example

```
for I1 := 0 to 5
  for I2 := 0 to 6
    A[I2 + 1] := 1/3 * (A[I2] + A[I2 + 1] + A[I2 + 2])
```

D={(0,1),(1,0),(1-1)}



$$T = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}$$




```
for I1 := 0 to 5
  for I2 := I1 to 6+I1
    A[I2-I1+1] := 1/3 * (A[I2-I1] + A[I2-I1+ 1] + A[I2-I1+ 2])
```

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

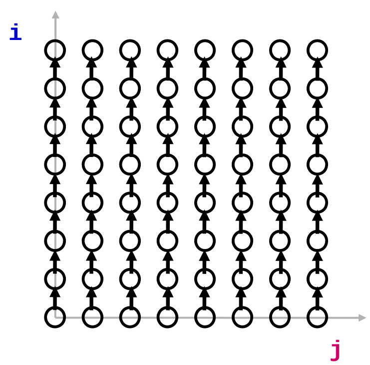
But...is the transform legal?

- 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?

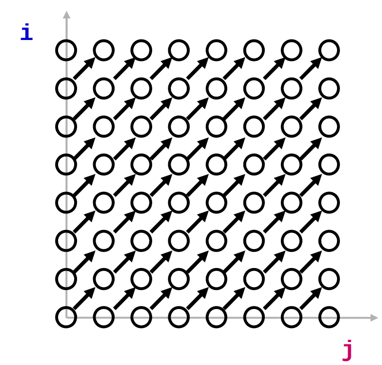
```
for i = 0 to N-1
    for j = 0 to N-1
        A[i+1][j] += A[i][j];
```



But...is the transform legal?

- Loop reversal ok?
- Loop interchange ok?

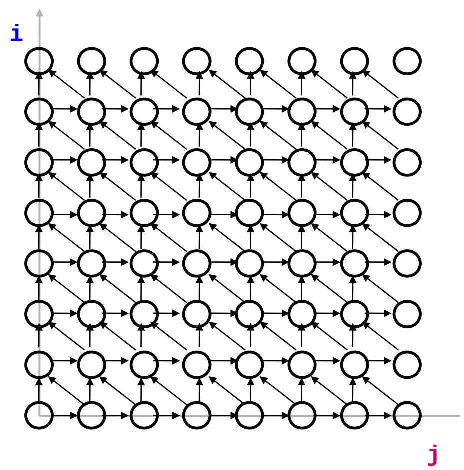
```
for i = 0 to N-1
    for j = 0 to N-1
        A[i+1][j+1] += A[i][j];
```



But…is the transform legal?

- What other visit order is legal here?

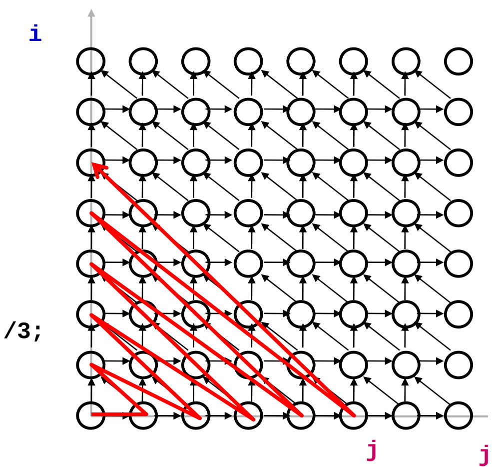
```
for i = 0 to TS
  for j = 0 to N-2
    A[j+1] =
      (A[j] + A[j+1] + A[j+2])/3;
```



But…is the transform legal?

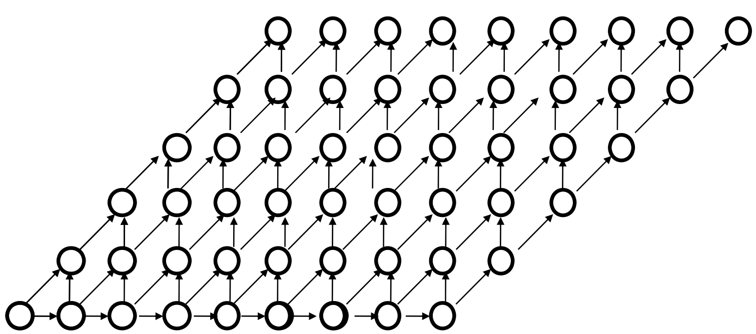
- What other visit order is legal here?

```
for i = 0 to TS
  for j = 0 to N-2
    A[j+1] =
      (A[j] + A[j+1] + A[j+2])/3;
```



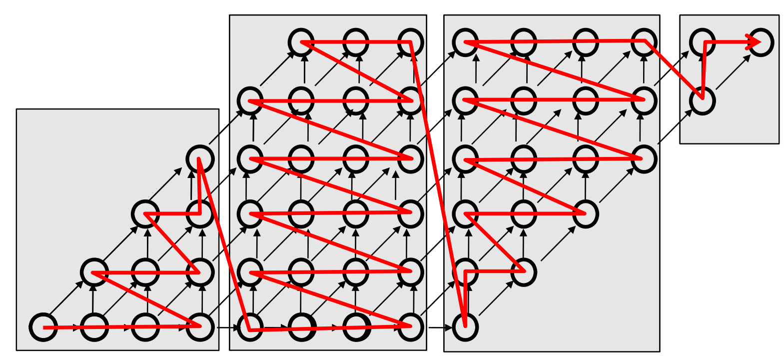
But…is the transform legal?

- Skewing…



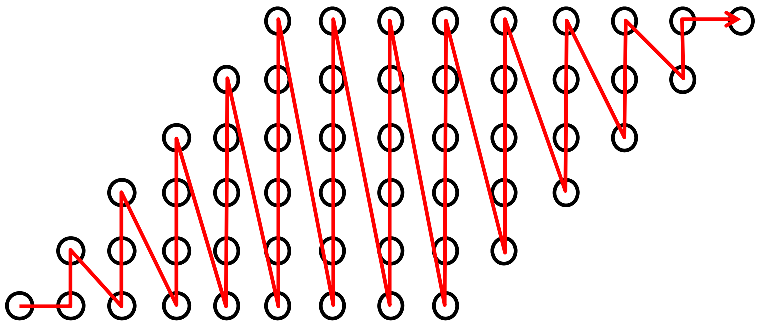
But…is the transform legal?

- Skewing…now we can block



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.

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

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**
    - $(d_1, \ldots, d_{i-1})$ 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<MIN(i+B*S-S, U); i+=S)