

Data Locality

Why locality?

- memory accesses are expensive
- exploit higher levels of memory hierarchy by reusing registers, cache lines, TLB, etc.
- locality of reference \Leftrightarrow reuse

Locality

- temporal locality *reuse of a specific location*
- spatial locality *reuse of adjacent locations (cache lines, pages)*

Reuse

- self-reuse *caused by same reference*
- group-reuse *caused by multiple references*

What reuse occurs in this loop nest?

```
do i = 1, N
do j = 1, N
  A(i) += B(j) + B(j+2)
```

Refs	Reuse on loop i	Reuse on loop j
A		
B		

Loop Transformations to Improve Reuse

To calculate *temporal* and *spatial* reuse

For each loop l in a nest, consider l innermost

1. partition references with group-reuse
 \Rightarrow reference groups
2. compute the cost in cache lines accessed
 \Rightarrow loop cost
3. rank the loops based on their *loop cost*
 \Rightarrow memory order

Key insight

If loop l promotes more reuse than loop k at the innermost position, then it probably promotes more reuse at any outer position

Selecting a loop permutation

- select *memory order* if legal
- if not, find a nearby legal permutation
- avoids evaluating many permutations

Selecting a Loop Permutation

Cost of reference group for loop k

1. select representative from reference group
2. find cost (in cache lines) with k innermost

invariant	1
unit-stride	$(U_k \Leftrightarrow L_k + 1) / c_l s$
otherwise	$U_k \Leftrightarrow L_k + 1$

3. multiply by trip counts of outer loops

Loop cost = sum of costs for reference groups

Matrix multiplication example

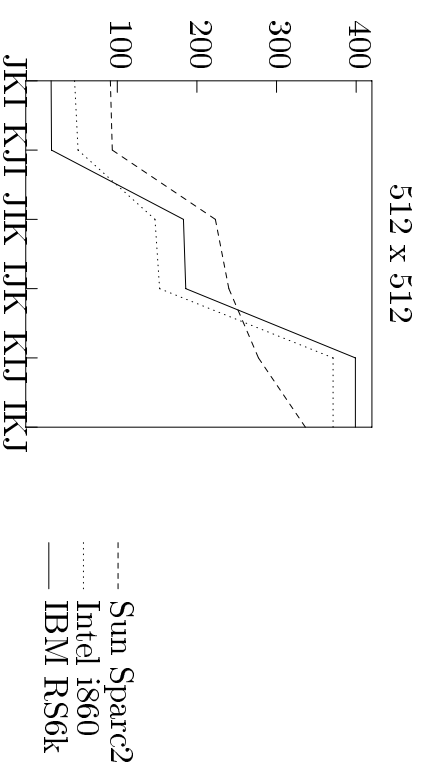
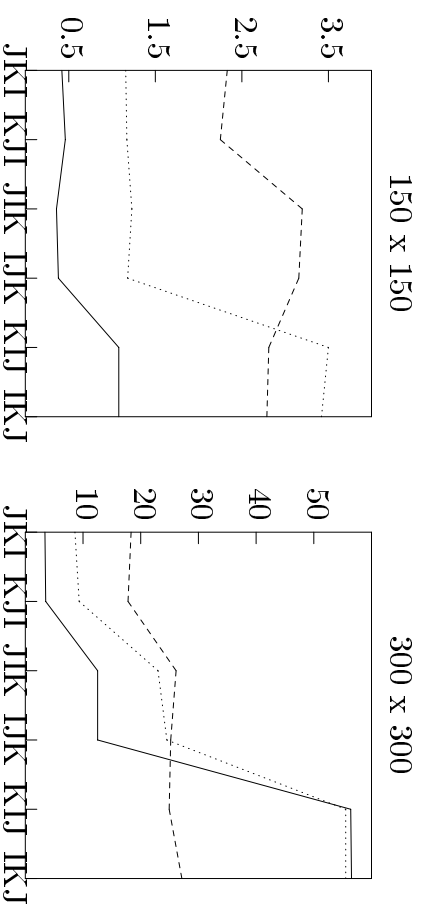
```
do j = 1, N
  do k = 1, N
    do i = 1, N
```

$$C(i, j) = C(i, j) + A(i, k) * B(k, j)$$

RefGroups	J	K	I
C(i,j)	$*n^2$	$*n^2$	$*n^2$
A(i,k)	$*n^2$	$*n^2$	$*n^2$
B(k,j)	$*n^2$	$*n^2$	$*n^2$
<i>total</i>			

LoopCost (with $c_l s = 4$)

Matrix Multiply (exec time in seconds)



Dependence Analysis

Question

Do two references never/maybe/always access the same memory location?

Benefits

- improves alias analysis
- enables loop transformations

Motivation

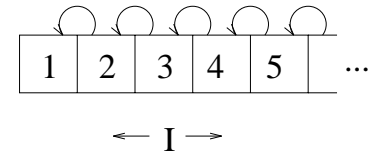
- classic optimizations
- instruction scheduling
- data locality (register/cache reuse)
- vectorization, parallelization

Obstacles

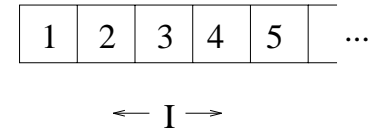
- array references
- pointer references

Dependence Analysis

```
do I = 1, 100
  A(I) =
    = A(I-1)
enddo
```



```
do I = 1, 100
  A(I) =
    = A(I)
enddo
```



A **loop-independent** dependence exists regardless of the loop structure. The source and sink of the dependence occur on the same loop iteration.

A **loop-carried** dependence is induced by the iterations of a loop. The source and sink of the dependence occur on different loop iterations.

Loop-carried dependences can inhibit parallelization and loop transformations

Dependence Testing

Given

```

do  $i_1 = L_1, U_1$ 
  ...
  do  $i_n = L_n, U_n$ 
 $S_1$      $\mathbf{A}(f_1(i_1, \dots, i_n), \dots, f_m(i_1, \dots, i_n)) = \dots$ 
 $S_2$      $\dots = \mathbf{A}(g_1(i_1, \dots, i_n), \dots, g_m(i_1, \dots, i_n))$ 
  
```

A *dependence* between statement S_1 and S_2 , denoted $S_1 \delta S_2$, indicates that S_1 , the *source*, must be executed before S_2 , the *sink* on some iteration of the nest.

Let α & β be a vector of n integers within the ranges of the lower and upper bounds of the n loops.

Does $\exists \alpha \leq \beta$, s.t.

$$f_k(\alpha) = g_k(\beta) \quad \forall k, 1 \leq k \leq m ?$$

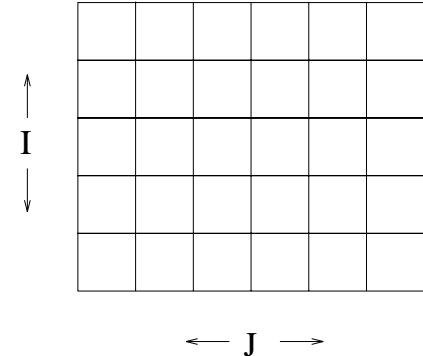
Iteration Space

```

do I = 1, 5
  do J = I, 6
    ...
  enddo
enddo
  
```

$$1 \leq I \leq 5$$

$$I \leq J \leq 6$$



- lexicographical (sequential) order for the above iteration space is

(1,1), (1,2), ..., (1,6)
 (2,2), (2,3), ..., (2,6)
 ...
 (5,5), (5,6)

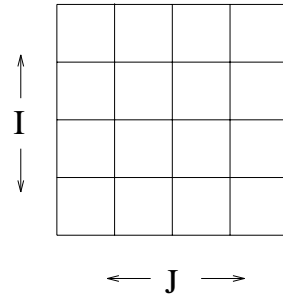
- given $I = (i_1, \dots, i_n)$ and $I' = (i'_1, \dots, i'_n)$,

$I < I'$ iff

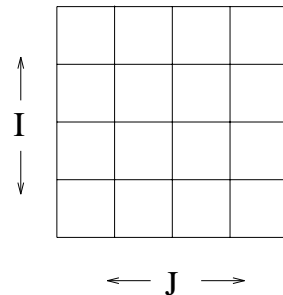
$$(i_1, i_2, \dots, i_k) = (i'_1, i'_2, \dots, i'_k) \quad \& \quad i_{k+1} < i'_{k+1}$$

Distance Vectors

```
do I = 1, N
  do J = 1, N
S1 A(I,J) = A(I,J-1)
  enddo
enddo
```



```
do I = 1, N
  do J = 1, N
S2 A(I,J) = A(I-1,J-1)
S3 B(I,J) = B(I-1,J+1)
  enddo
enddo
```



Distance Vector = number of iterations between accesses to the same location

distance vector

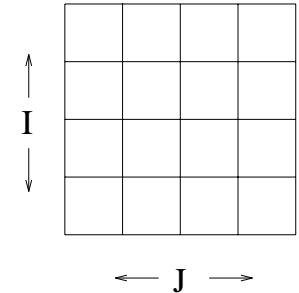
$S_1 \delta S_1$

$S_2 \delta S_2$

$S_3 \delta S_3$

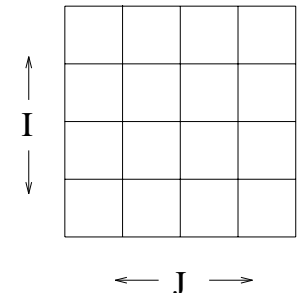
Loop Interchange

```
do I = 1, N
  do J = 1, N
S1 A(I,J) = A(I,J-1)
S2 B(I,J) = B(I-1,J-1)
  enddo
enddo
```



\Rightarrow loop interchange \Rightarrow

```
do J = 1, N
  do I = 1, N
S1 A(I,J) = A(I,J-1)
S2 B(I,J) = B(I-1,J-1)
  enddo
enddo
```



Loop interchange is safe *iff*

- it does not create a lexicographically negative direction vector $(1,-1) \rightarrow (-1,1)$

\Rightarrow Benefits

- may expose parallel loops, incr granularity
- reordering iterations may improve reuse