15-745
Optimizing For Data Locality - 1

Seth Copen Goldstein

Seth@cs.cmu.Edu

CMU

Based on "A Data Locality Optimizing Algorithm, Wolf & Lam, PLDI '91

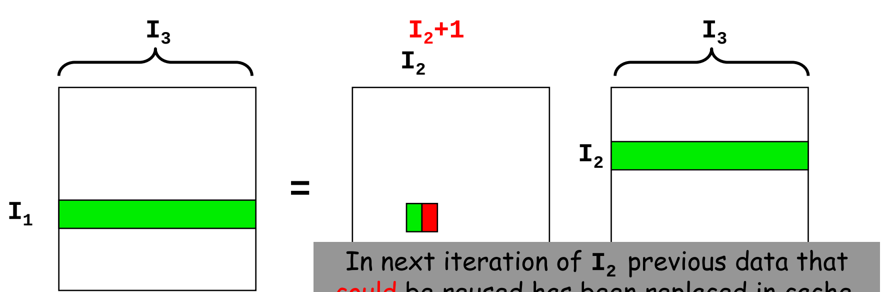
Outline

- The Problem
- Loop Transformations
  - dependence vectors
  - Transformations
  - Unimodular transformations
- Locality Analysis
- SRP

The Issue

- Improve cache reuse in nested loops
- Canonical simple case: Matrix Multiply

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

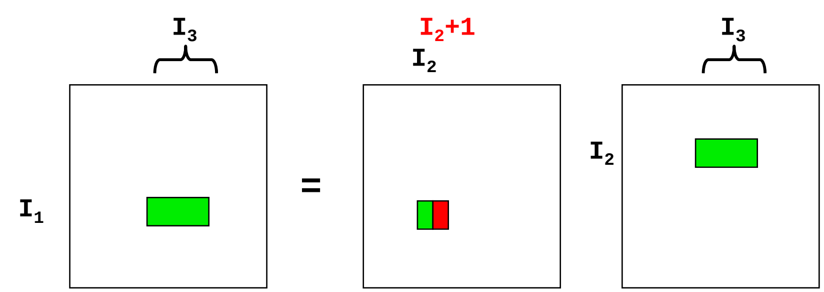


In next iteration of I_2 previous data that could be reused has been replaced in cache.

Tiling solves problem

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

```
for II2 := 1 to n by s
   for II3 := 1 to n by s
      for I1 := 1 to n
         for I2 := II2 to min(II2 + s - 1,n)
            for I3 := II3 to min(II3 + s - 1,n)
               C[I1,I3] += A[I1,I2] * B[I2,I3];
```

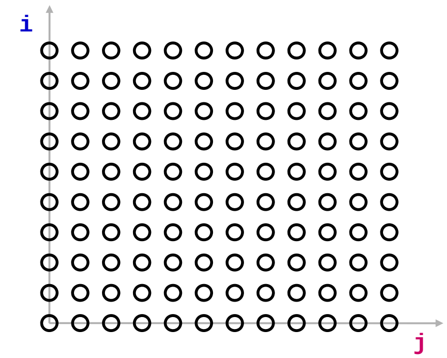


The Problem

- How to increase locality by transforming loop nest
- Matrix Mult is simple as it is both
  - legal to tile
  - advantageous to tile
- Can we determine the benefit?
  (reuse vector space and locality vector space)
- Is it legal (and if so, how) to transform loop?
  (unimodular transformations)

Handy Representation: "Iteration Space"

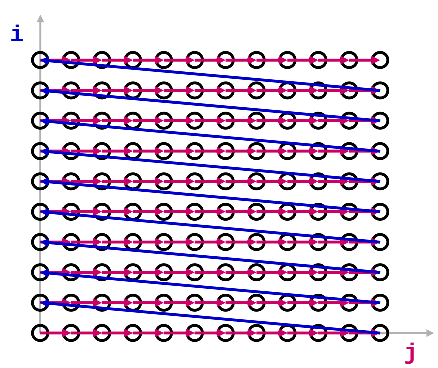
```
for i = 0 to N-1
  for j = 0 to N-1
   A[i][j] = B[j][i];
```



- each position represents an iteration

Visitation Order in Iteration Space

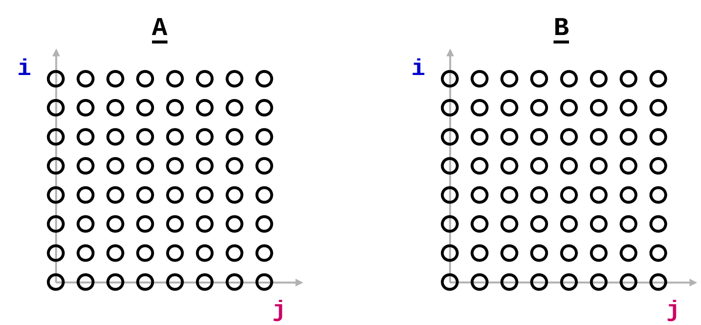
```
for i = 0 to N-1
  for j = 0 to N-1
  A[i][j] = B[j][i];
```



- Note: iteration space is not data space

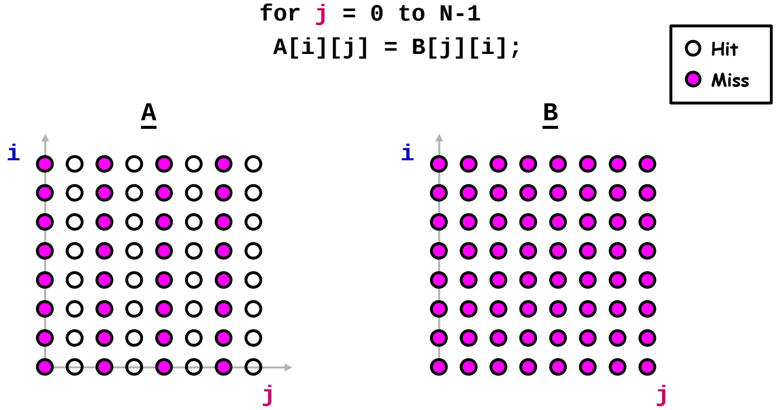
When Do Cache Misses Occur?

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



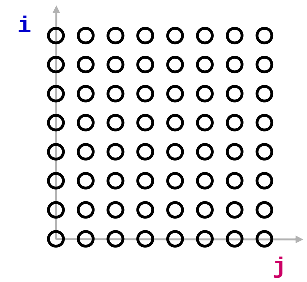
When Do Cache Misses Occur?

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



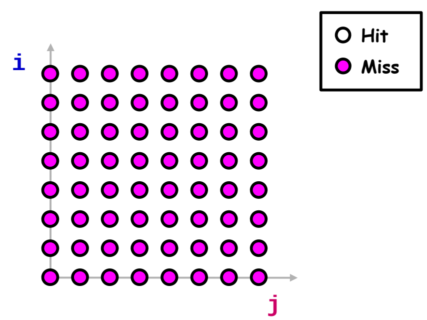
When Do Cache Misses Occur?

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



When Do Cache Misses Occur?

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



Optimizing the Cache Behavior of Array Accesses

- We need to answer the following questions:
  - when do cache misses occur?
    - use "locality analysis"
  - can we change the order of the iterations (or possibly data layout) to produce better behavior?
    - evaluate the cost of various alternatives
  - does the new ordering/layout still produce correct results?
    - use "dependence analysis"

Examples of Loop Transformations

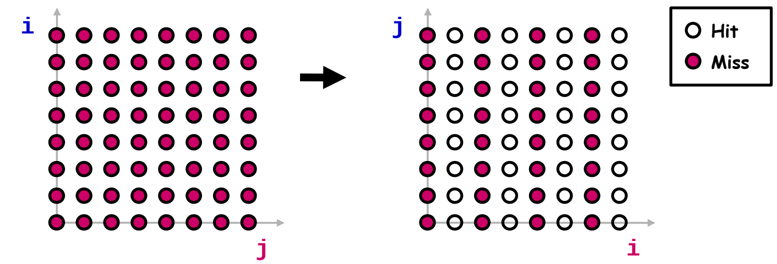
- Loop Interchange
- Cache Blocking
- Skewing
- Loop Reversal
- ...

(Loop Interchange, Cache Blocking:) Can improve locality

(Skewing, Loop Reversal:) Can enable above

Loop Interchange

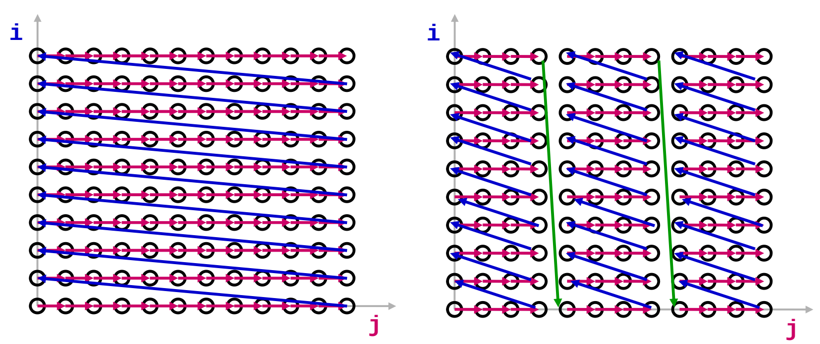
```
for i = 0 to N-1                    for j = 0 to N-1
  for j = 0 to N-1        →           for i = 0 to N-1
    A[j][i] = i*j;                      A[j][i] = i*j;
```



- *(assuming N is large relative to cache size)*

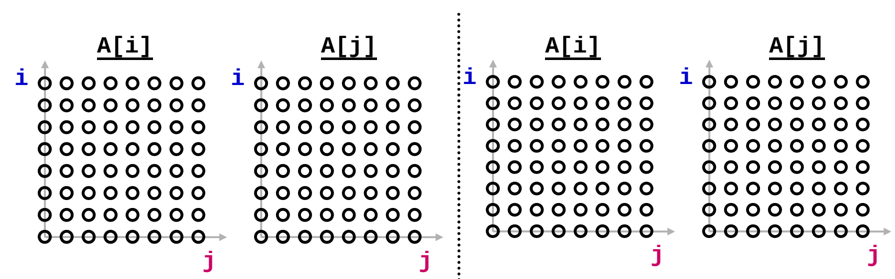
Impact on Visitation Order in Iteration Space

```
for i = 0 to N-1                 for JJ = 0 to N-1 by B
  for j = 0 to N-1      →          for i = 0 to N-1
    f(A[i],A[j]);                    for j = JJ to max(N-1,JJ+B-1)
                                       f(A[i],A[j]);
```



Cache Blocking (aka "Tiling")

```
for i = 0 to N-1                 for JJ = 0 to N-1 by B
  for j = 0 to N-1      →          for i = 0 to N-1
    f(A[i],A[j]);                    for j = JJ to max(N-1,JJ+B-1)
                                       f(A[i],A[j]);
```



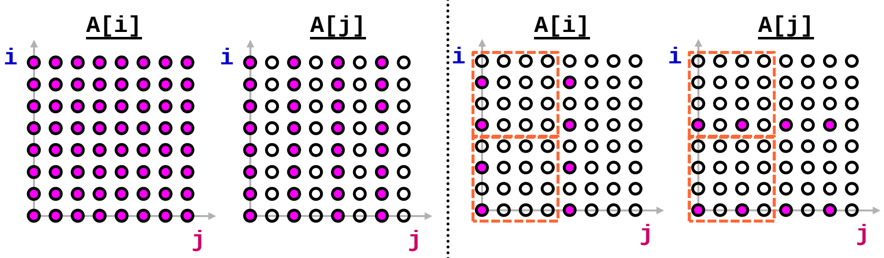
now we can exploit locality

Cache Blocking (aka "Tiling")

```
for i = 0 to N-1
  for j = 0 to N-1
    f(A[i],A[j]);
```

→

```
for JJ = 0 to N-1 by B
  for i = 0 to N-1
    for j = JJ to max(N-1,JJ+B-1)
      f(A[i],A[j]);
```



now we can exploit temporal locality

Cache Blocking in Two Dimensions

```
for i = 0 to N-1
  for j = 0 to N-1
    for k = 0 to N-1
      c[i,k] += a[i,j]*b[j,k];
```

```
for JJ = 0 to N-1 by B
  for KK = 0 to N-1 by B
    for i = 0 to N-1
      for j = JJ to max(N-1,JJ+B-1)
        for k = KK to max(N-1,KK+B-1)
          c[i,k] += a[i,j]*b[j,k];
```

- brings square sub-blocks of matrix "**b**" into the cache
- completely uses them up before moving on

Predicting Cache Behavior through "Locality Analysis"

- Definitions:
  - Reuse:
    accessing a location that has been accessed in the past
  - Locality:
    accessing a location that is now found in the cache
- Key Insights
  - Locality only occurs when there is reuse!
  - BUT, reuse does not necessarily result in locality.
  - Why not?

Steps in Locality Analysis

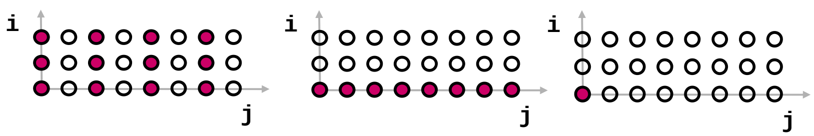
1. Find data reuse
   - if caches were infinitely large, we would be finished
2. Determine "localized iteration space"
   - set of inner loops where the data accessed by an iteration is expected to fit within the cache
3. Find data locality:
   - reuse $\supseteq$ localized iteration space $\supseteq$ locality

Types of Data Reuse/Locality

```
for i = 0 to 2
   for j = 0 to 100
       A[i][j] = B[j][0] + B[j+1][0];
```

Hit / Miss

A[i][j] B[j][0] B[j+1][0]



Spatial | Temporal | Group (spatial)

Kinds of reuse and the factor

```
for i = 0 to N-1
  for j = 0 to N-1
    f(A[i],A[j]);
```

What kinds of reuse are there?
A[i]?

A[j]?

Kinds of reuse and the factor

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


Kinds of reuse and the factor

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

self-temporal in 1, self-spatial in 2
Also, group spatial in 2

What is different about this and previous?

```
for i = 0 to N-1
  for j = 0 to N-1
    f(A[i],A[j]);
```

Uniformly Generated references

- f and g are indexing functions: $Z^n \rightarrow Z^d$
  - n is depth of loop nest
  - d is dimensions of array, A
- Two references A[f(**i**)] and A[g(**i**)] are uniformly generated if

$$f(\mathbf{i}) = H\mathbf{i} + c_f \text{ AND } g(\mathbf{i}) = H\mathbf{i} + c_g$$

- H is a linear transform
- c_f and c_g are constant vectors

Eg of Uniformly generated sets

These references all belong to the same uniformly generated set: H = [0 1]

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

A[I_2 + 1] $\begin{bmatrix} 0 & 1 \end{bmatrix} \begin{pmatrix} I_1 \\ I_2 \end{pmatrix} + \begin{bmatrix} 1 \end{bmatrix}$

A[I_2] $\begin{bmatrix} 0 & 1 \end{bmatrix} \begin{pmatrix} I_1 \\ I_2 \end{pmatrix} + \begin{bmatrix} 0 \end{bmatrix}$

A[I_2 + 2] $\begin{bmatrix} 0 & 1 \end{bmatrix} \begin{pmatrix} I_1 \\ I_2 \end{pmatrix} + \begin{bmatrix} 2 \end{bmatrix}$

Quantifying Reuse

- Why should we quantify reuse?
- How do we quantify locality?

Quantifying Reuse

- Why should we quantify reuse?
- How do we quantify locality?

- Use vector spaces to identify loops with reuse
- We convert that reuse into locality by making the "best" loop the inner loop
- Metric: memory accesses/iter of innermost loop. No locality → mem access

Self-Temporal

- For a reference, A[H**i**+**c**], there is self-temporal reuse between **m** and **n** when H**m**+**c**=H**n**+**c**, i.e., H(**r**)=**0**, where **r**=**m**-**n**.
- The direction of reuse is **r**.
- The self-temporal reuse vector space is: R_{ST} = Ker H
- There is locality if R_{ST} is in the localized vector space.

Recall that for nxm matrix A, the ker A = nullspace(A) = {$\mathbf{x}^m$ | A**x** = **0**}

Example of self-temporal reuse

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

Access	H	ker H	reuse?	Local?
$C[I_1,I_3]$	$\begin{pmatrix}1&0&0\\0&0&1\end{pmatrix}$	span{(0,1,0)}	n in I_2	
$A[I_1,I_2]$	$\begin{pmatrix}1&0&0\\0&1&0\end{pmatrix}$	span{(0,0,1)}		
$B[I_2,I_3]$	$\begin{pmatrix}0&1&0\\0&0&1\end{pmatrix}$	span{(1,0,0)}		

Self-Spatial

- Occurs when we access in order
  - A[i,j]: best gain, L
  - A[i,j*k]: best gain, L/k if |k| <= L
- How do we get spatial reuse for UG: H?

Self-Spatial

- Occurs when we access in order
  - A[i,j]: best gain, L
  - A[i,j*k]: best gain, L/k if |k| <= L
- How do we get spatial reuse for UG: H?
- Since all but row must be identical, set last row in H to 0, H_s
  self-spatial reuse vector space = R_{SS}
  R_{SS} = ker H_S
- Notice, ker H $\subseteq$ ker H_s
- If, $R_{ss} \cap L$ = $R_{ST} \cap L$, then no additional benefit to SS

Example of self-spatial reuse

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

Access	H_s	ker H_s	reuse?	Local?
$C[I_1,I_3]$	$\begin{pmatrix}1&0&0\\0&0&0\end{pmatrix}$	span{(0,1,0), (0,0,1)}		1/l
$A[I_1,I_2]$	$\begin{pmatrix}1&0&0\\0&0&0\end{pmatrix}$	span{(0,0,1), (0,1,0)}		
$B[I_2,I_3]$	$\begin{pmatrix}0&1&0\\0&0&0\end{pmatrix}$	span{(1,0,0), (0,0,1)}		

Self-spatial reuse/locality

- Dim(R_{SS}) is dimensionality of reuse vector space.
- If R_{SS}=0 → no reuse
- If $R_{SS}=R_{ST}$ no extra reuse from spatial
- Reuse of each element is $k/Ls^{dim(R\_SS)}$ where, s is number of iters per dim.
- $R_{SS} \cap L$ is amount of reuse exploited, therefore number of memory references generated is:

$$k/LS^{dim(R\_ST \cap L)}$$

Group Temporal

- Two refs A[H**i**+**c**] and A[H**j**+**d**] can have group temporal reuse in L iff
  - they are from same uniformly generated set
  - There is an **r** ∈ L s.t. H**r** = **c** – **d**
- if **c**-**d** = $\mathbf{r_p}$, then there is group temporal reuse, R_{GT} = ker H+span{$\mathbf{r_p}$}
- However, there is no extra benefit if $R_{GT} \cap L = R_{ST} \cap L$

Example:

```
For i = 1 to n
  for j=I to n
    A[i,j] = 0.2*(A[i,j]+A[i+1,j]+
                  A[i-1,j]+A[i,j+1]+A[i,j-1])
```

If L = span{j}, since ker H = ∅:

A[I,j] and A[I,j-1] → (0,0)-(0,-1) ∈span{(0,1)} yes

A[I,j-1] and A[i+1,j] → (0,-1)-(1,0) ∉span{(0,1)} no

Notice equivalence classes

Evaluating group temporal reuse

- Divide all references from a uniformly generated set into equiv classes that satisfy the R_{GT}
- For a particular L and g references
  - Don't count any group reuse when $R_{GT} \cap L = R_{ST} \cap L$
  - number of equiv classes is g_T.
  - Number of mem references is g_T instead of g

Total memory accesses

- For each uniformly generated set localized space, L line size, z

$$\frac{g_S + (g_T - g_S)/z}{z^e s^{\dim(R\_SS \cap L)}}$$

$$\text{where } e = \begin{cases} 0 \text{ if } R_{ST} \cap L = R_{SS} \cap L \\ 1 \text{ otherwise} \end{cases}$$

Next Time

- Complete example
- Unimodular transformations
- SRP