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

In next iteration of I₂, 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

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 $N-1$
    $A[i][j] = B[j][i]$;

Hit  Miss

<p>| | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Hit  Miss

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

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

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

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

Impact on Visitation Order in Iteration Space

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

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

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 locality
Cache Blocking (aka “Tiling”)

```plaintext
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
    f(A[i],A[j]);
    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

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

- 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 ⊇ localized iteration space ⊇ locality
Types of Data Reuse/Locality

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

```
A[i][j]

O Hit
O Miss

B[j][0]

B[j+1][0]
```

Spatial Temporal Group (temporal)

Kinds of reuse and the factor

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

```
Kinds of reuse and the factor

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

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

What is different about this and previous?

```python
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: \( \mathbb{Z}^n \to \mathbb{Z}^d \)
  - 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
  \[ f(i) = H + c_f \quad \text{AND} \quad g(i) = H + c_g \]
- \( H \) is a linear transform
- \( c_f \) and \( c_g \) are constant vectors

Eg of Uniformly generated sets

for \( I_1 := 0 \) to 5
for \( I_2 := 0 \) to 6
\[
A[I_2 + 1] = \frac{1}{3} \left( A[I_2] + A[I_2 + 1] + A[I_2 + 2] \right)
\]
\[
A[I_2] = \begin{bmatrix} 0 & 1 \\ I_1 & I_2 \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix}
\]
\[
A[I_2 + 2] = \begin{bmatrix} 0 & 1 \\ I_1 & I_2 \end{bmatrix} + \begin{bmatrix} 0 \\ 2 \end{bmatrix}
\]

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 \( \rightarrow \) mem access
**Self-Temporal**

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

Recall that for $nxm$ matrix $A$, the ker $A = \text{nullspace}(A) = \{x^T | Ax = 0\}$

**Example of self-temporal reuse**

```plaintext
for I1 := 1 to n
  for I2 := 1 to n
    for I3 := 1 to n

Access   H     ker H    reuse? Local?
C[I1,I3]   [1 0 0]  span((0,1,0))  n in I2
           [0 0 1]
A[I1,I2]   [1 0 0]  span((0,0,1))
           [0 1 0]
B[I2,I3]   [0 1 0]  span((1,0,0))
           [0 0 1]
```

**Self-Spatial**

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

Reuse is $s^\text{dim(Rst)}$

$R_{ST}$ intersect $L = \text{locality}$

# of mem refs = 1/above
Self-Spatial

- Occurs when we access in order
  - $A[i,j]$: best gain, $l$
  - $A[i,j*k]$: best gain, $l/k$ if $|k| \leq l$
- 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

$$\begin{align*}
\text{for } I_1 := 1 \text{ to } n \\
\text{for } I_2 := 1 \text{ to } n \\
\text{for } I_3 := 1 \text{ to } n \\
C[I_1,I_3] += A[I_1,I_2] \times B[I_2,I_3]
\end{align*}$$

Access $H_s$ ker $H_s$ reuse? Local?

- $C[I_1,I_3]$ $(1 \ 0 \ 0)$ span{(0,1,0), (0,0,1)} $1/l$
- $A[I_1,I_2]$ $(1 \ 0 \ 0)$ span{(0,0,1), (0,1,0)}
- $B[I_2,I_3]$ $(0 \ 1 \ 0)$ span{(1,0,0), (0,0,1)}

Self-spatial reuse/locality

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

Group Temporal

- Two refs $A[Hi+c]$ and $A[Hi+d]$ can have group temporal reuse in $L$ iff
  - they are from same uniformly generated set
  - There is an $r \in L$ s.t. $Hr = c - d$
- if $c-d = r_p$, then there is group temporal reuse, $R_{GT} = \ker H + \text{span}\{r_p\}$
- 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 $n$

If $L = \text{span}(j)$, since $\ker H = \emptyset$:
- $A[i,j]$ and $A[i,j-1] \rightarrow (0,0)-(0,-1) \in \text{span}((0,1))$ yes
- $A[i,j-1]$ and $A[i+1,j] \rightarrow (0,-1)-(1,0) \notin \text{span}((0,1))$ no

Notice equivalence classes

Evaluating group temporal reuse

- Divide all references from a uniformly generated set into equivalent 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 equivalent 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, $z$

$$g_S + (g_T - g_S)/z$$

$$z e^{\dim(R_{SS} \cap L)}$$

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

Now what?

- We have a way to characterize
  - Reuse (potential for locality)
  - Local iteration space
- Can we transform loop to take advantage of reuse?
- If so, can we?
Loop Transformation Theory

- Iteration Space
- Dependence vectors
- Unimodular transformations

Loop Nests and the Iter space

- General form of tightly nested loop
  
  ```
  for I_i := low_i to high_i by step_i
    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 \( \mathbb{Z}^n \) 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 \( p=(p_1, p_2, ... , p_n) \) and \( q=(q_1, q_2, ... , q_n) \)
  
  \[ p >_k q \] iff for \( 1 \leq k \leq n \),
  
  \[ \forall 1 \leq i < k, \ (p_i = q_i \) and \( p_k > q_k \)

- For MM:
  - \( (1,1,1), (1,1,2), (1,1,3), ... \),
  - \( (1,2,1), (1,2,2), (1,2,3), ... \),
  - \( (2,1,1), (2,1,2), (2,1,3), ... \)
  - \( (1,2,1) >_2 (1,1,2), (2,1,1) >_1 (1,4,2), \) etc.

Iteration Space

Every iteration generates a point in an n-dimensional space, where \( n \) is the depth of the loop nest.

```for (i=0; i<n; i++) {  
  ...  
  }  
for (j=0; j<n; j++) {  
  ...  
  }```
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 \([d_i^{\text{min}}, d_i^{\text{max}}]\), where
  \[ d_i^{\text{min}} \in \mathbb{Z} \cup \{-\infty\}, d_i^{\text{max}} \in \mathbb{Z} \cup \{\infty\} \text{ and } d_i^{\text{min}} \leq d_i^{\text{max}} \]
- 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.

Examples

\[
\begin{align*}
\text{for } I_1 &:= 1 \text{ to } n \\
\text{for } I_2 &:= 1 \text{ to } n \\
\text{for } I_3 &:= 1 \text{ to } n \\
C[I_1, I_3] &\leftarrow A[I_1, I_2] \times B[I_2, I_3]
\end{align*}
\]

\[
\begin{align*}
\text{for } I_1 &:= 0 \text{ to } 5 \\
\text{for } I_2 &:= 0 \text{ to } 6 \\
\end{align*}
\]

\[
D=\{(0,1),(1,0),(1,-1)\}
\]

Other defs

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

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

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
```

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
  ```
  ```
  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 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}
  \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} = \begin{bmatrix} p_1 \\ p_2 \end{bmatrix}
  \begin{bmatrix} p_1 \\ p_2 \end{bmatrix}
  ```
- 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:

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

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

- For 2 deep loop, reversal of outermost is:

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

Skewing

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

$$(p_1, p_2, \ldots, p_1, \ldots, p_j, \ldots) \mapsto (p_1, p_2, \ldots, p_j + fp_1, \ldots)$$

- Example for 2D

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

Loop Skewing Example

- Example for 2D

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

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

for $I_1 := 0$ to 5

for $I_2 := 0$ to 6


for $I_1 := 0$ to 5

for $I_2 := I_1$ to $6 + I_1$

$$A[I_2 - I_1 + 1] := 1/3 \times (A[I_2 - I_1] + A[I_2 - I_1 + 1] + A[I_2 - I_1 + 2])$$

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

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

But...is the transform legal?

- What other visit order is legal here?

```cpp
for i = 0 to TS
    for j = 0 to N-2
```

But...is the transform legal?

```cpp
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

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
---|---|---
\[
\begin{pmatrix}
1 & 0 \\
0 & -1
\end{pmatrix}
\begin{pmatrix}
0 & 1 \\
1 & 0
\end{pmatrix}
\begin{pmatrix}
1 & 1 \\
0 & 1
\end{pmatrix}
\]

“A Data Locality Optimizing Algorithm”, M.E.Wolf and M.Lam

Next Time

- Putting it all together: SRP
- Other loop transformations for locality

Linear Algebra

- Vector Spaces
- Linear Combinations
- dimensions
- Spans
- Kernels

Vector Spaces

- \( \mathbf{n} \) is a point in \( \mathbb{R}^n \)
- \( V = \{ \mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_m \} \) is a finite set of \( n \)-vectors over \( \mathbb{R}^n \).
- Linear combination of vectors of \( V \) is a vector \( \mathbf{x} \) as defined by
  \[
  \mathbf{x} = \alpha_1 \mathbf{v}_1 + \alpha_2 \mathbf{v}_2 + \ldots + \alpha_m \mathbf{v}_m
  \]
  where \( \alpha_i \) are real numbers.
- \( V \) is linearly dependent if a combination results in the \( \mathbf{0} \) vector, otherwise it is linearly independent.
**Dim and Basis**

- dimensionality of V is \( \dim(V) \)
  the number of independent vectors in V
- A basis for an m-dimensional vector space is a set of linearly independent vectors such that every point in V can be expressed as a linear comb of the vectors in the basis.
  - The vectors in the basis are called basis vectors

**Subspaces and span**

- Let V be a set of vectors
- The subspace spanned by V, \( \text{span}(V) \), is a subset of \( \mathbb{R}^n \) such that
  - \( V \subseteq \text{span}(V) \)
  - \( x, y \in \text{span}(V) \Rightarrow x + y \in \text{span}(V) \)
  - \( x \in \text{span}(V) \) and \( \alpha \in \mathbb{R} \Rightarrow \alpha x \in \text{span}(V) \)

**Range, Span, Kernel**

- A matrix A can be viewed as a set of column vectors.
- Range \( A^{n \times m} \) is \( \{Ax | x \in \mathbb{R}^m\} \)
- \( \text{span}(A) = \text{Range } A^{n \times m} \)
- \( \text{nullspace}(A) = \ker(A) = \ker(A^{n \times m}) = \{x^m | Ax \in 0\} \)
- \( \text{rank}(A) = \dim(\text{span}(A)) \)
- \( \text{nullity}(A) = \dim(\ker(A)) \)
- \( \text{rank}(A) + \text{nullity}(A) = n \), for \( A^{n \times m} \)