The Problem

Loop Transformations
- dependence vectors
- Transforms
- Unimodular transformations

Locality Analysis

SRP

The Issue

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

Tiling solves problem
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”

- each position represents an iteration

Visitation Order in Iteration Space

- Note: iteration space is not data space

When Do Cache Misses Occur?
When Do Cache Misses Occur?

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

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

Can improve locality

Cache Blocking (aka “Tiling”)

Can improve locality

Skewing

Can enable above

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

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

15-745 13© 2005-9 Seth Copen Goldman

15-745 14© 2005-9 Seth Copen Goldman

15-745 15© 2005-9 Seth Copen Goldman

15-745 16© 2005-9 Seth Copen Goldman
Cache Blocking (aka “Tiling”)

```
for i = 0 to N-1
for j = 0 to N-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
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

for $i = 0$ to 2
for $j = 0$ to 100

<table>
<thead>
<tr>
<th>Hit</th>
<th>Miss</th>
</tr>
</thead>
</table>

Spatial  Temporal  Group (temporal)

Kinds of reuse and the factor

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

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

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?

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
  \[
  f(i) = H_i + c_f \quad \text{AND} \quad g(i) = H_i + c_g
  \]
• \( H \) 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: \( H = [0 \ 1] \)

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

\[
\begin{align*}
A[I_2 + 1] &= \frac{1}{3} \times (A[I_2] + A[I_2 + 1] + A[I_2 + 2]) \\
A[I_2] &= \begin{bmatrix} 0 & 1 \\ I_1 & I_2 \end{bmatrix} \\
A[I_2 + 2] &= \begin{bmatrix} 0 & 1 \\ I_1 & I_2 \end{bmatrix} + \begin{bmatrix} 2 \\ I_2 \end{bmatrix}
\end{align*}
\]

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, $A[H_i+c]$, 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 $A = \text{nullspace}(A) = \{x^m | Ax = 0\}$

Example of self-temporal reuse

```plaintext
for $I_1 := 1$ to $n$
  for $I_2 := 1$ to $n$
    for $I_3 := 1$ to $n$
      $C[I_1,I_3] := A[I_1,I_2] * B[I_2,I_3]$

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 $|k| \leq l$
• How do we get spatial reuse for UG: $H$?

Reuse is $s^{\dim(R_{ST})}$
• $R_{ST}$ intersect $L = \text{locality}$
• # of mem refs = $1/\text{above}$
**Self-Spatial**

- Occurs when we access in order
  - \( A[i,j] \): best gain, 1
  - \( A[i,j*k] \): 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, \( H_s \)

  \[
  \text{self-spatial reuse vector space} = R_{SS} = \ker H_s
  \]

- 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**

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

**Self-spatial reuse/locality**

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

**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 \( r \in L \text{ s.t. } Hr = c - d \)
- 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 = 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} ∩ L = R_{ST} ∩ 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

  \[
  \frac{g_S + (g_T - g_S)/z}{z^e \dim(R_{SS} \cap L)}
  \]

  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
  
  \[
  \begin{align*}
  \text{for } I_1 := \text{low}_1 \text{ to high}_1 \text{ by step}_1 \\
  & \quad \text{for } I_2 := \text{low}_2 \text{ to high}_2 \text{ by step}_2 \\
  & \quad \quad \ldots \quad \text{for } I_i := \text{low}_i \text{ to high}_i \text{ by step}_i \\
  & \quad \quad \quad \ldots \quad \text{for } I_n := \text{low}_n \text{ to high}_n \text{ by step}_n \\
  \text{Stmts}
  \end{align*}
  \]

• 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: \( p=(p_1, p_2, \ldots, p_n) \)

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 \( p \geq_k q \) 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), ...
  - (1,2,1), (1,2,2), (1,2,3), ...
  - (2,1,1), (2,1,2), (2,1,3), ...
  - (1,2,1) \geq_2 (1,1,2), (2,1,1) \geq_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.

\[
\begin{align*}
\text{for } (i=0; i<n; i++) \{ \\
& \quad \text{***} \\
\text{for } (j=0; j<n; j++) \{ \\
& \quad \quad \text{***}
\end{align*}
\]
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 \(d_i^{\text{min}} \in \mathbb{Z} \cup \{-\infty\}\) and \(d_i^{\text{max}} \in \mathbb{Z} \cup \{\infty\}\) 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

for \(I_1 := 1\) to \(n\)
for \(I_2 := 1\) to \(n\)
for \(I_3 := 1\) to \(n\)
\(C[I_1, I_3] := A[I_1, I_2] \times B[I_2, I_3]\)

for \(I_1 := 0\) to \(5\)
for \(I_2 := 0\) to \(6\)

\(D = (0, 1, 0)\)

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., \(d = I_t - I_s\)

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

```plaintext
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

```plaintext
for I_1 := ... 
  for I_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:

```plaintext
T = \[
  0 & 1 \\
  1 & 0
\]

\[
0 1 \begin{bmatrix} p_1 \\ p_2 \end{bmatrix} = \begin{bmatrix} p_2 \\ p_1 \end{bmatrix}
\]
```

- Since, T is a linear transform, Td is transformed dependence:

```plaintext
\[
\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} d_1 \\ d_2 \end{bmatrix} = \begin{bmatrix} d_2 \\ d_1 \end{bmatrix}
\]
Reversal

- Reversal of $i^{th}$ loop reverses its traversal, so it can be represented as:

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

For 2 deep loop, reversal of outermost is:

\[
T = \begin{bmatrix}
-1 & 0 \\
0 & 1
\end{bmatrix}
\]

\[
\begin{bmatrix}
-1 \\
0
\end{bmatrix} p_1 = \begin{bmatrix}
-p_1 \\
p_2
\end{bmatrix}
\]

Skewing

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

\[
(p_1, p_2, \ldots) \rightarrow (p_1, p_2, \ldots, p_j + fp_i, \ldots)
\]

- Example for 2D

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

Loop Skewing Example

\[
\begin{align*}
\text{for } I_1 &: \text{ 0 to 5} \\
\text{for } I_2 &: \text{ 0 to 6} \\
A[I_2 + 1] &= \frac{1}{3} \times (A[I_2] + A[I_2 + 1] + A[I_2 + 2])
\end{align*}
\]

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

\[
\begin{align*}
\text{for } I_1 &: \text{ 0 to 5} \\
\text{for } I_2 &= I_1 \text{ to } 6 + I_1 \\
A[I_2-I_1+1] &= \frac{1}{3} \times (A[I_2-I_1] + A[I_2-I_1+1] + A[I_2-I_1+2])
\end{align*}
\]

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

15-745 57© 2005-9 Seth Copen Goldstein

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?

• What other visit order is legal here?

for \( i = 0 \) to \( TS \)
for \( j = 0 \) to \( N-2 \)
But...is the transform legal?

• What other visit order is legal here?

\[
\text{for } i = 0 \text{ to } TS \\
\text{for } j = 0 \text{ to } N-2 \\
A[j+1] = \\
\]

But...is the transform legal?

• Skewing...

• 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

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

“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 \( \mathbb{R}^n \)
- \( \mathcal{V} = \{ \mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_m \} \) is a finite set of \( n \)-vectors over \( \mathbb{R}^n \).
- Linear combination of vectors of \( \mathcal{V} \) 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.
- \( \mathcal{V} \) is linearly dependent if a combination results in the \( \mathbf{0} \) vector, otherwise it is linearly independent.
Dim and Basis

- The 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 combination 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.
- $\text{Range } A^{n \times m} = \{Ax | x \in \mathbb{R}^m\}$
- $\text{span}(A) = \text{Range } A^{n \times m}$
- $\text{nullspace}(A) = \ker(A) = \ker(A^{n \times m}) = \{x^m | Ax \in \mathbb{0}\}$
- $\text{rank}(A) = \dim(\text{span}(A))$
- $\text{nullity}(A) = \dim(\ker(A))$
- $\text{rank}(A) + \text{nullity}(A) = n$, for $A^{n \times m}$