Loop Transformations, Dependences, and Parallelization

Announcements
– HW1 is due Wednesday February 8th

Today
– Semester long project
– Data dependence recap
  – Parallelism and storage tradeoff
  – Scalar expansion example
– Skewing Smith-Waterman
– Automating transformations like skewing
  – Iteration space representation
  – Transformation representation
  – Applying the transformation to the iteration space
  – Generating code for the new iteration space

Semester Long Project

Posted Online
Main Idea (find a different app and/or different automation tool)
– Evaluate the performance issues of a benchmark or application
– Provide a roofline model for a machine in the department and place your app in that model
– Develop a parallelization and loop transformation strategy for improving the app performance
– Automate some part of that strategy in an existing automation tool
– Compare the automation tool with tools covered in class

Requirements
– Project proposal due next Friday February 17th
– Project intermediate report due Wednesday March 28th
– Final report due Wednesday May 2nd
– Poster presentations sometime before or on Thursday May 10th
Parallelism and Storage Usage Tradeoff

False dependences limit parallelism

Removing false dependences requires more memory/storage

Obtaining performance requires finding an effective tradeoff

Loop-Carried, Storage-Related Dependences

Problem
  – Loop-carried dependences inhibit parallelism
  – Scalar references result in loop-carried dependences

Example

\[
\begin{align*}
\text{do } & i = 1,6 \\
\text{t } &= \text{A}(i) + \text{B}(i) \\
\text{C}(i) &= \text{t} + 1/\text{t} \\
\text{enddo}
\end{align*}
\]

Can this loop be parallelized? No.
What kind of dependences are these? Anti dependences.

Convention for these slides: Arrays start with upper case letters, scalars do not
Removing False Dependences with Scalar Expansion

Idea
– Eliminate false dependences by introducing extra storage

Example

\[
\begin{align*}
&\text{do } i = 1, 6 \\
&T(i) = A(i) + B(i) \\
&C(i) = T(i) + 1/T(i) \\
&\text{enddo} \\
&t = T[6]
\end{align*}
\]

Can this loop be parallelized?

Disadvantages?

Scalar Expansion Details

Restrictions
– The loop must be a countable loop
  \textit{i.e.} The loop trip count must be independent of the body of the loop
– The expanded scalar must have no upward exposed uses in the loop

\[
\begin{align*}
&\text{do } i = 1, 6 \\
&\text{print}(t) \\
&t = A(i) + B(i) \\
&C(i) = t + 1/t \\
&\text{enddo}
\end{align*}
\]

– Nested loops may require much more storage
– When the scalar is live after the loop, we must move the correct array value into the scalar
Protein String Matching Example (smithWaterman.c)

```c
for (i=1; i<=a[0]; i++) {
    for (j=1; j<=b[0]; j++) {
        diag = h[i-1][j-1] + sim[a[i]][b[j]];
        down = h[i-1][j] + DELTA;
        right = h[i][j-1] + DELTA;
        max = MAX3(diag, down, right);
        if (max <= 0) {
            h[i][j] = 0; xTraceback[i][j] = -1; yTraceback[i][j] = -1;
        } else if (max == diag) {
            h[i][j] = diag; xTraceback[i][j] = i-1; yTraceback[i][j] = j-1;
        } else if (max == down) {
            h[i][j] = down; xTraceback[i][j] = i-1; yTraceback[i][j] = j;
        } else {
            h[i][j] = right; xTraceback[i][j] = i; yTraceback[i][j] = j-1;
        }
        if (max > Max) {
            Max = max; xMax = i; yMax = j;
        }
    }  // end for loops
}
```

Skewing (smithWaterman.c)

```c
// Let j'=i+j and i'='i.
for (i'=1; i'<=a[0]; i'++) {
    for (j'=i'+1; j'<=i'+b[0]; j'++) {
        diag = h[i'-1][j'-i'-1] + sim[a[i']][b[j'-i']];
        down = h[i'-1][j'-i'] + DELTA;
        right = h[i'][j'-i'-1] + DELTA;
        max = MAX3(diag, down, right);
        if (max <= 0) {
            h[i'][j'-i'] = 0; xTraceback[i'][j'-i'] = -1; yTraceback[i'][j'-i'] = -1;
        } else if (max == diag) {
            h[i'][j'-i'] = diag; xTraceback[i'][j'-i'] = i'-1;
            yTraceback[i'][j'-i'] = j'-i'-1;
        } else if (max == down) {
            h[i'][j'-i'] = down; xTraceback[i'][j'-i'] = i'-1;
            yTraceback[i'][j'-i'] = j'-i';
        } else {
            h[i'][j'-i'] = right; xTraceback[i'][j'-i'] = i';
            yTraceback[i'][j'-i'] = j'-i'-1;
        }
        if (max > Max) {
            Max = max; xMax = i'; yMax = j'-i';
        }
    }  // end for loops
}
```
Automating Loop Transformations with Frameworks

Currently
– Frameworks used in compiler to …
  – abstract loops, memory accesses, and data dependences in loop
  – specify the effect of a sequence of loop transformations on the loop,
    its memory accesses, and its data dependences
  – generate code from the transformed loop
– Loop transformations affect the schedule of the loop

Future
– How can framework technology be exposed in the programming model?

Frameworks we will discuss this semester
– Unimodular
– Polyhedral
– Presburger
– Sparse Polyhedral

Iteration Space Representation

Original code
\[
\text{do } i = 1, 6 \\
\quad \text{do } j = 1, 5 \\
\quad \quad A(i,j) = A(i-1,j+1)+1 \\
\text{enddo} \\
\text{enddo}
\]

Represent the iteration space
– As an intersection of inequalities
– The iteration space is the integer tuples within the intersection

\[
\begin{align*}
1 & \leq i \\
i & \leq 6 \\
1 & \leq j \\
j & \leq 5 \\
\end{align*}
\]

\[
\begin{bmatrix}
-1 & 0 \\
1 & 0 \\
0 & -1 \\
0 & 1
\end{bmatrix}
\begin{bmatrix}
i \\
j
\end{bmatrix}
\leq
\begin{bmatrix}
-1 \\
6 \\
-1 \\
5
\end{bmatrix}
\]
**Lexicographical Order as Schedule**

**Iteration point**
- Integer tuple with dimensionality $d$  
  \[
  \left( i_0, i_1, \ldots, i_d \right)
  \]

**Lexicographical Order**
- First order the iteration points by $i_0$, then $i_1$, … and finally $i_d$.

\[
\left( i_0, i_1, \ldots, i_{d-1} \right) \preceq \left( i_0, i_1, \ldots, i_{d-1} \right) \equiv
\left( i_0 < j_0 \right) \lor \left( i_0 = j_0 \land i_1 < j_1 \right) \lor \ldots \left( i_0 = j_0 \land i_1 = j_1 \land \ldots i_{d-1} = j_{d-1} \right)
\]

---

**Frameworks for Loop Transformations**

**Loop Transformations as functions**

\[
\tilde{i}' = f(\tilde{i})
\]

**Unimodular Loop Transformations [Banerjee 90],[Wolf & Lam 91]**
- can represent loop permutation, loop reversal, and loop skewing
- unimodular linear mapping (determinant of matrix is + or - 1)
  \[
  \tilde{i}' = T \tilde{i}
  \]
  - T is a matrix, i and i’ are iteration vectors
  - example
    \[
    \begin{bmatrix}
    i' \\
    j'
    \end{bmatrix} =
    \begin{bmatrix}
    0 & 1 \\
    1 & 1
    \end{bmatrix}
    \begin{bmatrix}
    i \\
    j
    \end{bmatrix}
    \]
- limitations
  - only perfectly nested loops
  - all statements are transformed the same
**Loop Skewing**

Original code

```
    do i = 1, 6
        do j = 1, 5
            A(i, j) = A(i-1, j+1) + 1
        enddo
    enddo
```

Distance vector: \((1, -1)\)

Skewing:

\[
\begin{bmatrix}
1 & 0 \\
1 & 1 \\
\end{bmatrix}
\begin{bmatrix}
i \\
j \\
\end{bmatrix}
= \begin{bmatrix}
i \\
i + j \\
\end{bmatrix}
\]

**Transforming the Dependences and Array Accesses**

Original code

```
    do i = 1, 6
        do j = 1, 5
            A(i, j) = A(i-1, j+1) + 1
        enddo
    enddo
```

Dependence vector:

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

New Array Accesses:

\[
A\left(\begin{bmatrix}
1 & 0 \\
0 & 1 \\
\end{bmatrix} + \begin{bmatrix}
0 & 0 \\
0 & 0 \\
\end{bmatrix}\right) = A(i, j)
\]

\[
A\left(\begin{bmatrix}
1 & 0 \\
0 & 1 \\
\end{bmatrix} \begin{bmatrix}
1 & 0 \\
-1 & 1 \\
\end{bmatrix} + \begin{bmatrix}
0 & 0 \\
0 & 0 \\
\end{bmatrix}\right) = A(i', j' - i')
\]

\[
A\left(\begin{bmatrix}
1 & 0 \\
0 & 1 \\
\end{bmatrix} + \begin{bmatrix}
0 & -1 \\
1 & 1 \\
\end{bmatrix}\right) = A(i - 1, j + 1)
\]

\[
A\left(\begin{bmatrix}
1 & 0 \\
0 & 1 \\
\end{bmatrix} \begin{bmatrix}
1 & 0 \\
-1 & 1 \\
\end{bmatrix} + \begin{bmatrix}
0 & -1 \\
1 & 1 \\
\end{bmatrix}\right) = A(i' - 1, j' - i' + 1)
\]
Transforming the Loop Bounds

Original code

```plaintext
do i = 1, 6
do j = 1, 5
   A(i,j) = A(i-1,j+1)+1
enddo
enddo
```

Bounds:

\[
\begin{bmatrix}
-1 & 0 & 0 \\
1 & 0 & 1 \\
0 & -1 & 0 \\
\end{bmatrix}
\begin{bmatrix}
i \\
j \\
\end{bmatrix}
\leq
\begin{bmatrix}
-1 & 6 & 1 \\
-1 & 5 & -1 \\
\end{bmatrix}
\]

Transformed code

```plaintext
do i' = 1, 6
do j' = 1+i', 5+i'
   A(i',j'-i') = A(i'-1,j'-i'+1)+1
enddo
enddo
```

Revisiting (smithWaterman.c)

```plaintext
for (i=1;i<=a[0];i++) {
   for (j=1;j<=b[0];j++) {
      diag = h[i-1][j-1] + sim[a[i]][b[j]];
      down = h[i-1][j] + DELTA;
      right = h[i][j-1] + DELTA;
      ...
   }
}

Let j'=i+j and i'=i.
```

```plaintext
for (i'=1;i'<=a[0];i'++) {
   for (j'=1+i';j'<=i'+b[0];j'+) {
      diag = h[i'-1][j'-i'-1] + sim[a[i]][b[j'-i']];
      down = h[i'-1][j'-i'] + DELTA;
      right = h[i'][j'-i'-1] + DELTA;
      ...
   }
}
```
Transformation Legality

Recall …
– A dependence vector is legal if it is lexicographically non-negative.
– Applying the transformation function to each dependence vector produces a dependence vector for the new iteration space.

When is a transformation legal assuming a lexicographical schedule?

What about parallelism?

Converting C loops to iteration space representation

Analyses needed
– Loop analysis
  – Loop bounds from AST or control-flow graph
  – Induction variable detection
– Pointer analysis
  – Do pointers point at same or overlapping memory?
  – Note that in C can cast a pointer to an integer and back and can do pointer arithmetic.
  – In general requires whole program analysis.
– Dependence analysis

Is this even possible?
– Current tools make the optimistic pointer assumption
– We need programming models that simplify or remove the need for such analyses
Concepts

Parallelism and Memory Usage tradeoff

Transformation Frameworks
- Representing the iteration space
- Representing transformations
- Applying transformations to the iteration space, dependences, and array accesses
- Testing the legality of a transformation

Compiler analyses needed in C to obtain an iteration space representation

References


Next Time

Reading
- Read: Automatic Parallelization in the Polytope Model by Feautrier

Homework
- HW1 is due Wednesday 2/8/12

Lecture
- Using the unimodular framework to represent other loop transformations