LLVM passes and Intro to Loop Transformation Frameworks

Announcements

- This class is recorded and will be in D2L panapto.
- No quiz Monday after spring break.
- Will be doing mid-semester class feedback.

Today

- LLVM passes
- 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

LLVM Passes: Example Skeleton Pass

LLVM documentation

- on writing a pass
- assumes you have control over the llvm source hierarchy

Skeleton example (running it on rose computer)

- `git clone https://github.com/sampsyo/llvm-pass-skeleton.git`
- `cd llvm-pass-skeleton`
- Edit the Skeleton.cpp file
- In llvm-pass-skeleton:
  - `mkdir build; cd build; cmake ..; make VERBOSE=1`
  - Create a test.c file.
  - `clang -S -emit-llvm test.c`
  - `opt -load build/skeleton/libSkeletonPass.so -skeleton < test.ll > /dev/null`

Then what?

LLVM 3.9.1 documentation

- http://releases.llvm.org/3.9.0/docs/
- Programmers Manual

Helpful Hints for Common Operations

- http://releases.llvm.org/3.9.0/docs/ProgrammersManual.html
- Iterate over basic blocks in a function
- Iterate in instructions in a basic block
- Creating and inserting new Instructions
- Deleting instructions
- Etc.

The Core LLVM Class hierarchy

- Instruction class
- Value class

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

Protein String Matching Example (smithWaterman.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
```

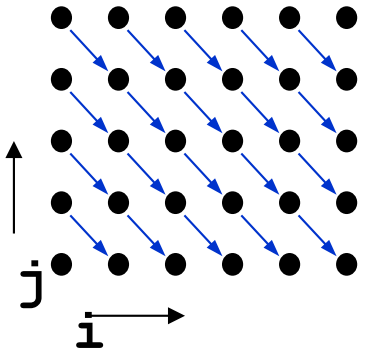
Skewing (smithWaterman.c)

```
// 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
```

Iteration Space Representation

Original code

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



Represent the iteration space

–As an intersection of inequalities

–The iteration space is the integer tuples within the intersection

Bounds:

$$\begin{array}{lcl} 1 & <= & i \\ i & <= & 6 \\ 1 & <= & j \\ j & <= & 5 \end{array}$$

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

Lexicographical Order as Schedule

Iteration point

- Integer tuple with dimensionality d $(i_0, i_1, \ldots, i_d)$

Lexicographical Order

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

$$(i_0, i_1, ..., i_{d-1}) \prec (j_0, j_1, ..., j_{d-1}) \equiv$$
$$(i_0 < j_0) \vee (i_0 = j_0 \wedge i_1 < j_1) \vee ...(i_0 = j_0 \wedge i_1 = j_1 \wedge ...i_{d-1} < j_{d-1})$$

Frameworks for Loop Transformations

Loop Transformations as functions

$$\vec{i'} = f(\vec{i})$$

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)

$$\vec{i'} = T\vec{i}$$

  - T is a matrix, i and i' are iteration vectors
  - example

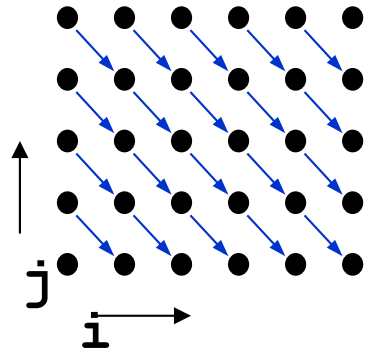
$$\begin{bmatrix} i' \\ j' \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} i \\ j \end{bmatrix}$$

- limitations
  - only perfectly nested loops
  - all statements are transformed the same

Loop Skewing

Original code

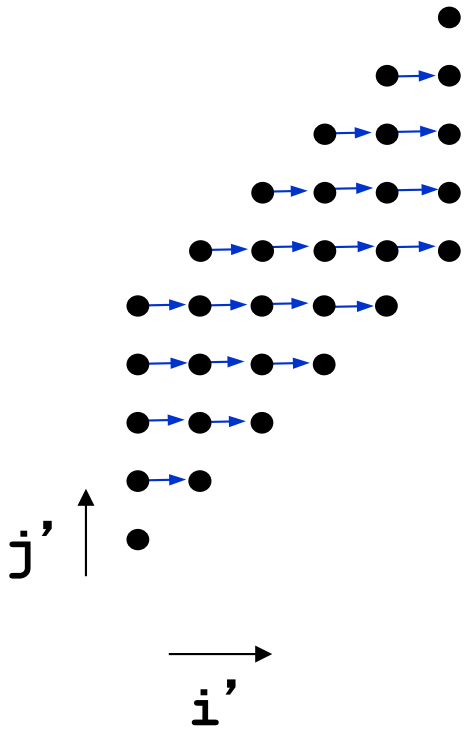
```
do i = 1,6
     do j = 1,5
        A(i,j) = A(i-1,j+1)+1
     enddo
enddo
```



Distance vector: (1, -1)

Skewing:

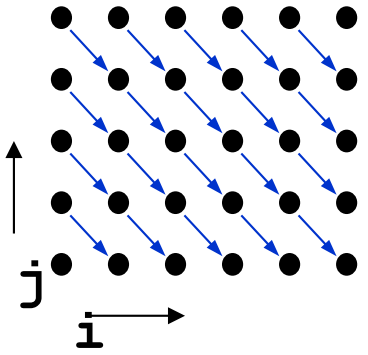
$$\begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} i \\ j \end{bmatrix} = \begin{bmatrix} i \\ i+j \end{bmatrix}$$



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:

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

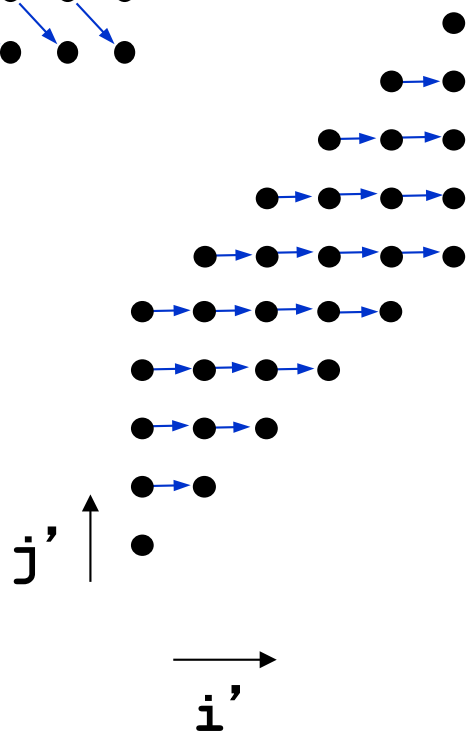
New Array Accesses:

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

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

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

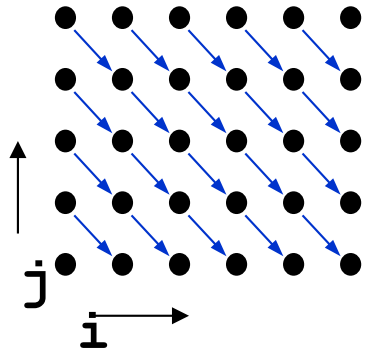
$$A\left(\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ -1 & 1 \end{bmatrix} \begin{bmatrix} i' \\ j' \end{bmatrix} + \begin{bmatrix} -1 \\ 1 \end{bmatrix}\right) = A(i'-1, j'-i'+1)$$



Transforming the Loop Bounds

Original code

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



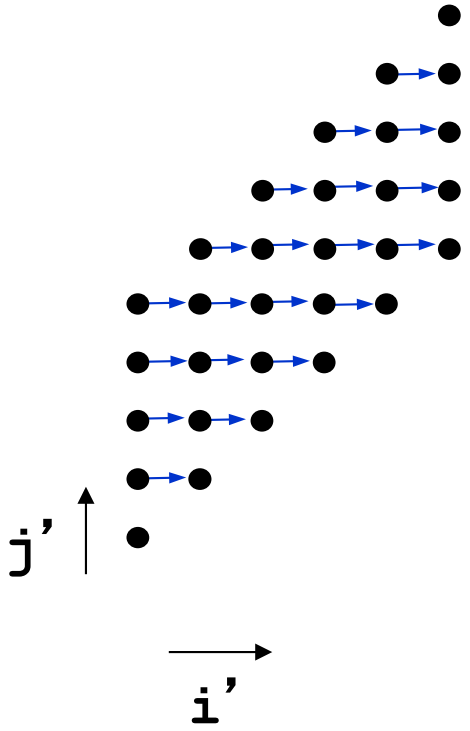
Bounds:

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

$$\begin{bmatrix} -1 & 0 \\ 1 & 0 \\ 0 & -1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ -1 & 1 \end{bmatrix} \begin{bmatrix} i' \\ j' \end{bmatrix} \leq \begin{bmatrix} -1 \\ 6 \\ -1 \\ 5 \end{bmatrix}$$

Transformed code

```
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)

```
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.

```
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;
     …
```

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?

Skewing smithWaterman.c (Is this transformation legal?)

```
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 and i'=i+j.

Thus i=j' and j=i'-j'.

```
for (i'=2;i'<=a[0]+b[0];i'++) {
  for (j'=max(1,i'-b[0]); j'<=min(a[0],i'-1); j'++) {
    diag    = h[j'-1][i'-j'-1] + sim[a[i']][b[i'-j']];
    down    = h[j'-1][i'-j'] + DELTA;
    right   = h[j'][i'-j'-1] + DELTA;
    …
```

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

[Banerjee90] Uptal Banerjee, "Unimodular transformations of double loops," In Advances in Languages and Compilers for Parallel Computing, 1990.

[Wolf & Lam 91] Wolf and Lam, "A Data Locality Optimizing Algorithm," In Programming Languages Design and Implementation, 1991.

Next Time

Reading assignment

- Unifying Framework for Iteration Reordering Transformations by Kelly and Pugh (read all but Section 5)

Lecture

- More loop transformation frameworks