Compiling for Parallelism & Locality

Last time

- Dynamic compilation
- End of lectures on low level optimizations

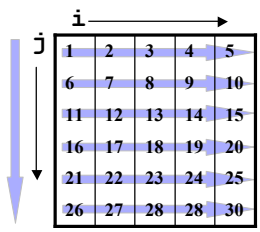
Today

- Data dependences and loops
- Parallelism and locality

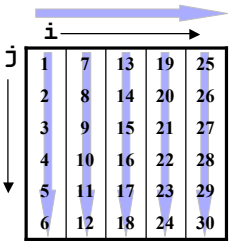
Example 1: Loop Permutation for Improved Locality

Sample code: Assume Fortran's Column Major Order array layout

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



poor cache locality

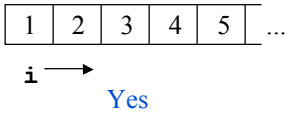


good cache locality

Example 2: Parallelization

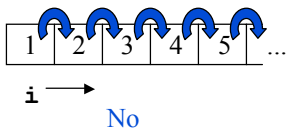
Can we parallelize the following loops?

```
do i = 1,100
   A(i) = A(i)+1
enddo
```



Yes

```
do i = 1,100
   A(i) = A(i-1)+1
enddo
```



No

Data Dependences

Recall

- A data dependence defines ordering relationship two between statements
- In executing statements, data dependences must be respected to preserve correctness

Example

```
S1  a := 5;
S2  b := a + 1;
S3  a := 6;
```

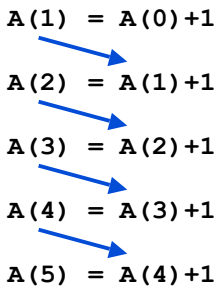
≠ ?

```
S1  a := 5;
S3  a := 6;
S2  b := a + 1;
```

Data Dependences and Loops

How do we identify dependences in loops?

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



Simple view

- Imagine that all loops are fully unrolled
- Examine data dependences as before

Problems

- Impractical
- Lose loop structure

Dependence Analysis for Loops

Big picture

- To improve data locality and parallelism we often focus on loops
- To transform loops, we must understand data dependences in loops
- Since we can't represent all iterations of a loop, we need some abstractions
- The basic question: does a transformation preserve all dependences?

Today and Next Time

- Basic abstractions and machinery

Wednesday

- Its application to loop transformations

Data Dependence Terminology

We say statement s_2 depends on s_1

- **True (flow) dependence**: s_1 writes memory that s_2 later reads
- **Anti-dependence**: s_1 reads memory that s_2 later writes
- **Output dependences**: s_1 writes memory that s_2 later writes
- **Input dependences**: s_1 reads memory that s_2 later reads

Notation: $s_1 \delta s_2$

- s_1 is called the **source** of the dependence
- s_2 is called the **sink** or **target**
- s_1 must be executed before s_2

Dependences and Loops

Loop-independent dependences

```
do i = 1,100
   A(i) = B(i)+1
   C(i) = A(i)*2
enddo
```

Dependences within the same loop iteration

Loop-carried dependences

```
do i = 1,100
   A(i) = B(i)+1
   C(i) = A(i-1)*2
enddo
```

Dependences that cross loop iterations

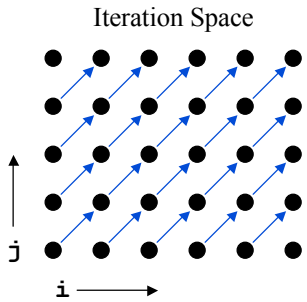
Iteration Spaces

Idea

- Explicitly represent the iterations of a loop nest

Example

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



Iteration Space

- A set of tuples that represents the iterations of a loop
- Can visualize the dependences in an iteration space

Protein String Matching Example

```
q = k_1
r = k_2
score = 0

for i=0 to n1-1
  h[0,-1] = p[0,-1] = 0
  f[0,-1] = -q
  for j=0 to n0-1
    f[i,j] = max(f[i,j-1],h[i,j-1]-q)-r
    EE[i,j] = max(EE[i-1,j],HH[i-1,j],-q)-r
    h[i,j] = p[i,j-1] + pam2[aa1[i],aa0[j]]
    h[i,j] = max( max(0,EE[i,j]), max(f[i,j],h[i,j])
    p[i,j] = HH[i-1,j]
    HH[i,j] = h[i,j]
    score[i,j] = max(score[i,j-1],h[i,j])
  endfor
endfor

return score[n1-1,n0-1]
```

Distance Vectors

Idea

- Concisely describe dependence relationships between iterations of an iteration space
- For each dimension of an iteration space, the distance is the number of iterations between accesses to the same memory location

Definition

- $\mathbf{v} = \mathbf{i}^T - \mathbf{i}^S$

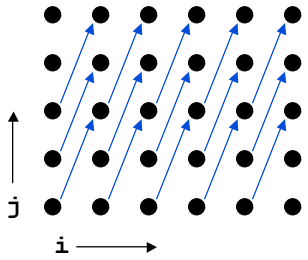
Example

```
do i = 1,6
   do j = 1,5
      A(i,j) = A(i-1,j-2)+1
   enddo
enddo
```

outer loop

inner loop

Distance Vector: (2,1)



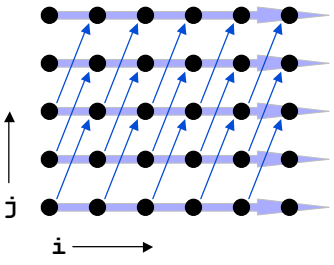
Distance Vectors and Loop Transformations

Idea

- Any transformation we perform on the loop must respect the dependences

Example

```
do i = 1,6
   do j = 1,5
      A(i,j) = A(i-1,j-2)+1
   enddo
enddo
```



Can we permute the `i` and `j` loops?

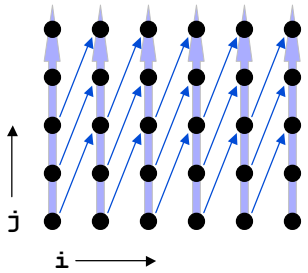
Distance Vectors and Loop Transformations

Idea

– Any transformation we perform on the loop must respect the dependences

Example

```
do j = 1,5
   do i = 1,6
      A(i,j) = A(i-1,j-2)+1
   enddo
enddo
```



Can we permute the `i` and `j` loops?

– Yes