Module 18: Loop Optimizations

Lecture 36: Cycle Shrinking

The Lecture Contains:

Previous Next

Cycle Shrinking ...

- Dependence cycle with distance > 1
- Transform a serial loop into two nested loops (outer serial and inner parallel)
- Consider the loop

```
for I = 1,n
A[i+k] = B[i] –1
B[i+k] = A[i] + C[i]
endfor
```

```
for i=1, n, k
forall j=1, i+k-1
A[j+k]=B[j]-1
B[j+k]=A[j]+C[j]
endforall
endfor
```

```
For I = 3,n
A[i]=B[i-2]-1
B[i]=A[i-3]*k
Endfor

For j =3, n, 2
forall I = j, j+1
A[i]=B[i-2]-1
B[i]=A[i-3]*k
endforall
Endfor
```

```
A3 = B1 -1
B3 = A0 * k
A4 = B2 -1
B4 = A1 * k
A5 = B3 -1
B5 = A2 * k
A6 = B4 -1
B6 = A3 * k
A7 = B5 -1
B7 = A4 * k
A8 = B6 -1
B8 = A5 * k
```

Cycle Shrinking in Distance Varying Loops

- The distance may not be constant
- Cycle may be reduced by the minimum distance

```
For I = 1,n
X[i]=Y[i]+Z[i]
Y[i+3]=X[i-4]*W[i]
Endfor
```

```
for j=1, n, 3
forall I = j, j+2
X[i]=Y[i]+Z[i]
Y[i+3]=X[i-4]*W[i]
endforall
endfor
```

Previous Next

Loop Un-switching

- Removes loop independent conditionals from a loop
- Reduces frequency of execution of conditional statements
- Makes loop structure more complex

```
For I = 1, n
for j = 2, n
if T[i] > 0 then
A[i,j]=A[i,j-1]*T[i]+B[j]
else
A[I,j] = 0.0
endif
endfor
Endfor
```

```
for I = 1, n
if T[i]>0 then
for j=2,n
A[i,j]=A[i,j-1]*T[i]+B[j]
endfor
else
for j=2,n
A[I,j]=0.0
endfor
endif
endfor
```

Loop Peeling

- Used to handle wrap around variables
- Removes first or the last iteration of the loop into separate code
- Peeling can also be used to remove loop invariant code by executing it only in the first iteration (assuming n = 1)

```
for I = 1,n
A[i]=(x+y)*B[i]
endfor
```

```
A[1] = (t=x+y)*B[1]
for I = 2,n
A[i]=t*B[i]
endfor
```

Index Set Splitting

- Generalization of loop peeling
- Used to remove conditionals from the loops

```
For I = 1, 100
A[i]=B[i]+C[i]
if i>10 then
D[i]=A[i]+A[i-10]
endif
Endfor
```

```
for I = 1,10
A[i]=B[i]+C[i]
endfor
for I = 11 to 100
A[i]=B[i]+C[i]
D[i]=A[i]+A[i-10]
endfor
```

Previous Next

Loop Fusion

- When two adjacent countable loops have the same loop limits they can sometimes be fused
- Reduces cost of test and branch
- Fusing loops which refer to the same data enhances temporal locality
  - It has significant impact on cache and virtual memory performance
- Loop fusion may increase size of the loop which can reduce instruction locality (noticeable with very small cache memories)
- Fusion is legal if all the dependence relations are preserved
- Before fusion all relations must flow from body1 to body2 (unless carried by an outer loop)

```
For I = 1,n
A[i]=B[i]+1
Endfor
For I = 1,n
C[i]=A[i]/2
Endfor
For I = 1,n
D[i]=1/C[i+1]
Endfor
S2→S5
S5→S8
```

```
For I = 1,n
A[i]=B[i]+1
C[i]=A[i]/2
D[i]=1/C[i+1]
Endfor
```

after fusion the second dependence is violated

```
For I = 1,n
A[i]=B[i]+1
C[i]=A[i]/2
Endfor
For I = 1,n
D[i]=1/C[i+1]
Endfor
```

```
for I = 1,99
A[i]=B[i]+1
Endfor
for I = 1,98
C[i]=A[i+1]*2
Endfor
```

```
A[1]=B[1]+1
for I = 2,99
A[i]=B[i]+1
Endfor
for I = 1,98
C[i]=A[i+1]* 2
Endfor
```

```
A[1]=B[1]+1
for j = 0,97
A[j+2]=B[j+2]+1
C[j+1]=A[j+2]*2
Endfor
```

Loop Fission

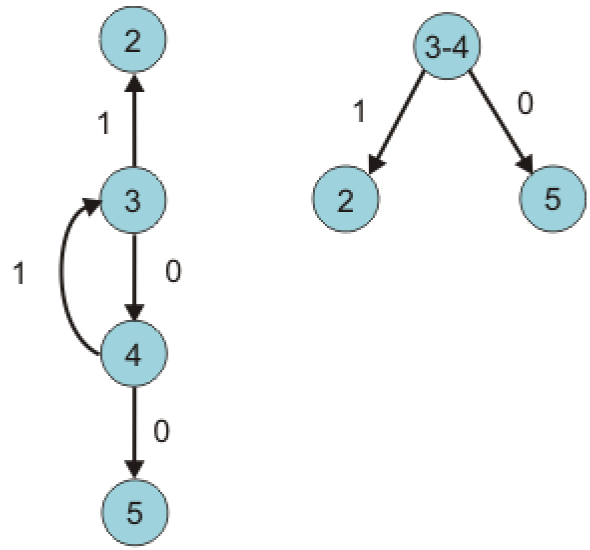
- A single loop may be broken into smaller loops (inverse of loop fusion)
- Used on machines which have very small instruction cache
- Improves memory locality
- Construct a statement level dependence graph of the body of the loop
  - Dependence relations carried by outer loop need not be preserved
  - Inner loops are treated as single nodes
  - If there are no cycles then loop fission can divide the loop into separate loops around each node
  - The loops are ordered in topological order of the dependence graph

Previous Next

Module 18: Loop Optimizations

Lecture 36: Cycle Shrinking

For I = 1,n
A[i] = A[i] + B[i-1]
B[i] = C[i-1]*x + y
C[i] = 1/B[i]
D[i] = sqrt(C[i])
endfor



For ib = 0,n-1
B[ib+1] = C[ib]*x + y
C[ib+1] = 1/B[ib+1]
Endfor
For ib = 0,n-1
A[ib+1] = A[ib+1] + B[ib]
Endfor
For ib = 0,n-1
D[ib+1] = sqrt(C[i])
Endfor
I = n+1

Loop Reversal

- Compiler can decide to run a loop backward
- Always legal for parallel loops
- Illegal for sequential loop if it has loop carried dependence
- Allows loop fusion to proceed where it might otherwise fail

for I = 1,n	for i=n downto 1
A[i]=B[i]+1	A[i]=B[i]+1
C[i]=A[i]/2	C[i]=A[i]/2
endfor	D[i]=1/C[i+1]
for i=1,n	endfor
D[i]=1/C[i+1]	
endfor	

Previous Next

Loop Skewing

- Normalization can change the shape of the iteration space
- It may affect the ability to interchange loops
- Consider following code
  for I = 2, n
  for j = I, n
  A[I,j] = 0.5 *(A[I,j-1]+A[i-1,j])
  endfor
  endfor
- Using un-normalized iteration vector the dependence distances are (0,1) and (1,0)

Iteration Space of The Loop

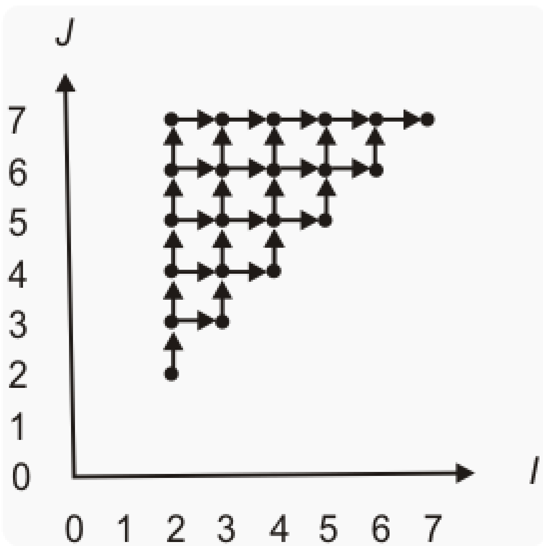


Figure Courtesy: High Performance Compilers
For Parallel Computing by Wolfe

- After interchange the code is
  for j = 2, n
  for i = 2, j
  A[I,,j] = 0.5 *(A[I,,j-1]+A[i-1,,j])
  endfor
  endfor

- Using normalized iteration vectors the shape of the iteration space changes as shown below

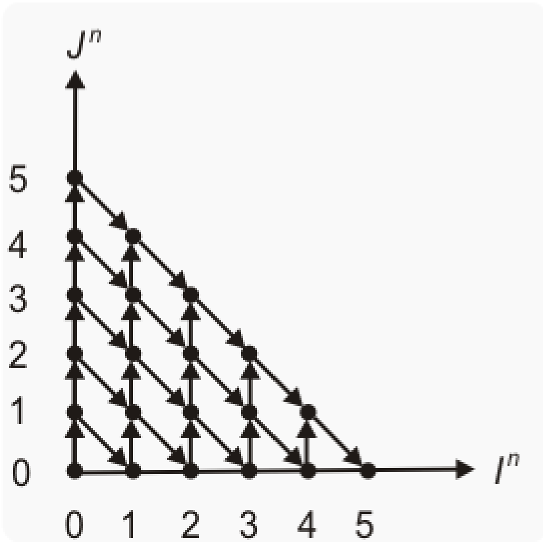


Figure Courtesy: High Performance Compilers For Parallel Computing by Wolfe

The dependence distance are (0,1) and (1,-1)
This dependence prevents loop interchange

- If normalization can prevent interchange then un-normalization can enable loop interchange
- This is called loop skewing
- Skewing changes the iteration vector of each iteration by adding the outer loop index value to the inner loop index
    - (I,j) becomes (I, j+i)
    - A dependence relation from (i1, j1) to (i2, j2) will have distance (i1, j1) –(i2-j2) = (d1, d2)
    - After skewing the distance will change to (i1, j1+i1) –(i2, j2+i2) = (d1, d2+d1)
- In general loops can be skewed by a factor changing iteration label from (I,j) to (I, j+fi)
    - This changes distance from (d1, d2) to (d1, d2+fd1)
    - F can also be negative
- Choosing whether to skew and the factor by which to skew depends upon the goal to enable other transformations

Example

- Interchange following loop using skewing
```
for I = 2, n
for j = 2, m
A[I,j] = 0.5 * (A[i-1, j-1]+A[i-1, j+1])
endfor
Endfor
```
- The two dependence distances are (1,1) and (1,-1)
- The second one prevents the interchange
- Skewing the loop would change the dependence distance to (1,2) and (1,0) allowing the interchange
- The compiler must generate the correct limits using FM method

The Final Code After Skewing is

```
For js = 2, n+m-2
for is = max(0, js-m+2), min(n-2, js)
I = is+2
j = js-is+2
A[I,j] = 0.5 * (A[i-1, j-1] + A[i-1, j+1])
endfor
endfor
```

Loop Blocking or Strip Mining

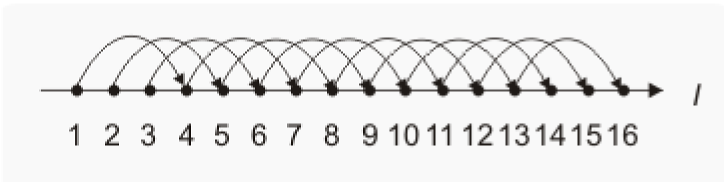
- Creates doubly nested loops out of single loops
- Organizes computation into chunks of approximately equal sizes
- Used to overcome size limitations of caches and local memory

```
for I = 1,n
A[i] = B[i] + C[i]
endfor
```

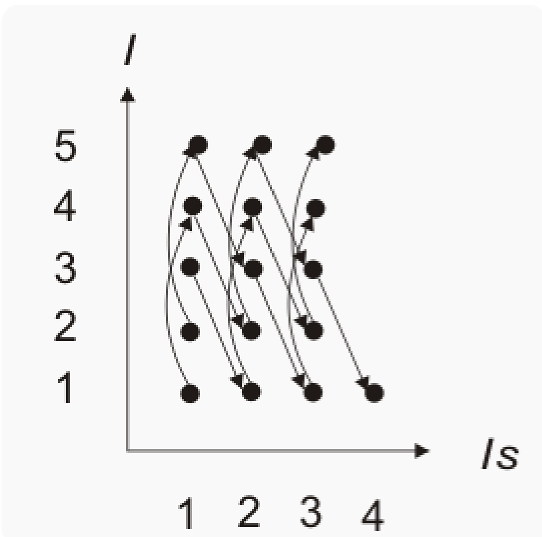
```
for j = 1, n, k
for I = j, min(j+k, n)
A[i]=B[i]+C[i]
endfor
endfor
```

Example

```
for I = 1, 16
A[i+3]=A[i]+B[i]
endfor
```




```
for It = 1, 16, 5
for i=it, min(16, it+4)
A[i+3]=A[i]+B[i]
endfor
endfor
```



Previous Next

Loop Tiling

- Similar to strip mining (Strip mining works for single loops)
- Loop tiling is used for nested loops
- Tiling boundaries are parallel to the iteration space axes and not to iteration space boundaries
- The eventual goal is to interchange tile loops outward and element loops inward
- Tiling is characterized by tile size ts and a tile offset to (0 = to < ts)
- Each tile starts an iteration i such that i mod ts = to
- Each tile iterates from tn-ts+to to (tn+1)-ts+to-1 where tn is tile number
- The compiler must determine the minimum and maximum tile numbers
- The compiler must ensure that element loop does not execute outside its original iteration space
- The general formula for tiling for a loop such as
  for I = lo, hi
  is
  for it = floor((lo-to)/ts)*ts+to, floor((hi-to)/ts)*ts+to, ts
  for I = max(lo,it), min(hi,it+ts-1)

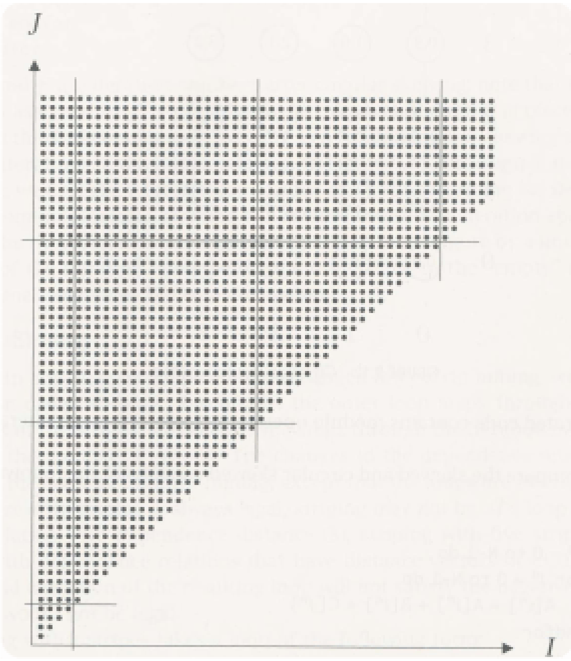
Tile following loops with a tile size of 20 and an offset of 5

```
For I = 1, 50
for j = i, 60
A[I,j] = A[I,j]+1
endfor
endfor
```

just applying the formula produces following loop

```
For It = -15, 45, 20
for i = max(1,it), min(50,it+19)
for jt=floor((i-5)/20)*20+5, 45, 20
for j=max(I,jt), min(60, jt+19)
A[I,j] = A[I,j]+1
endfor
endfor
endfor
endfor
```

The Tiled Iteration Space



- Interchange jt loop with I loop
- Compiler finds the new lower limits for jt

```
For It = -15, 45, 20
for jt = max(-15,it), 45, 20
for i = max(I,it), min(50,it+19)
for j=max(I,jt), min(60, jt+19)
A[I,j] = A[I,j]+1
endfor
endfor
endfor
endfor
```

Circular Loop skewing

- A variation of loop skewing
- Skew the inner loop iterations such that they wrap around a cylinder
- The shape of the iteration space does not change but the relative positions change
- Backward dependencies with large distances make tiling unprofitable
- Circular loop skewing shortens backward dependencies

```
For I = 0, n-1
for j = 0, n-1
A[i] = A[i] + B[i] * C[i]
endfor
endfor
```

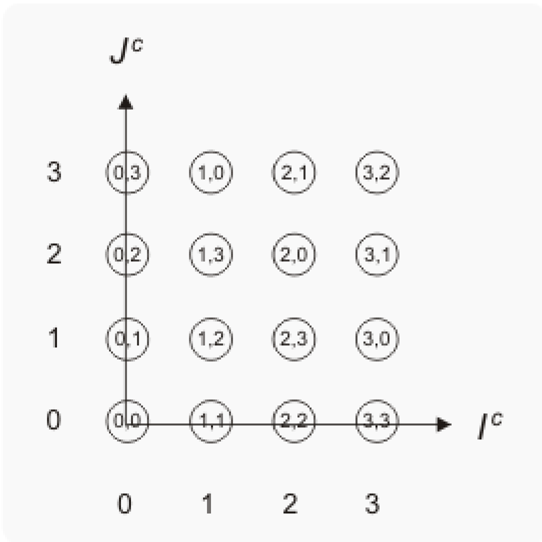


Figure Courtesy: High Performance Compilers For Parallel Computing by Wolfe

The circular loop skewing does not change the shape of the iteration space. It changes the iterations computed at each point.

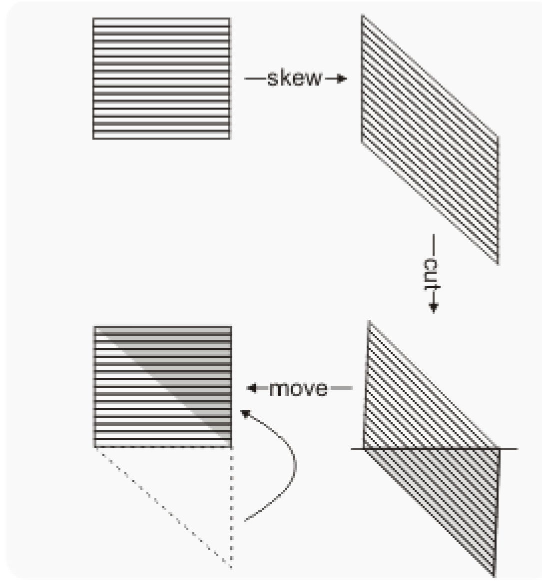


Figure Courtesy: High Performance Compilers
For Parallel Computing by Wolfe