Compilers II

Chapter 3:
Loop transformation using unimodular matrices

Given the dependence structure, how does the compiler efficiently find the optimum loop transformation?

Lecturer: *Paul H J Kelly*

Spring 1997

Department of Computing
Imperial College

Room 423
email: phjk

Introduction

⋆ The dependence analysis approach we have studied so far can be used to determine the *validity* of the loop transformations needed for high performance.

⋆ This does not tell us *which* transformations are optimal in a particular situation.

⋆ As we saw with matrix multiplication, several transformations may be needed to reach the best code.

⋆ One approach is to try all the valid transformations, and evaluate the results using metrics of parallelism and locality.

⋆ This "generate and test" approach is not attractive because of the large number of possibilities — many of them uninteresting.

⋆ In this section of the course we study a powerful theory for representing and evaluating loop transformations. This promises:

- ▷ efficient algorithms which guarantee to extract parallelism if it is present.
- ▷ efficient algorithms for optimising load balance and locality.

⋆ The theory is not all-powerful:

- ◇ it does not handle tiling/blocking entirely elegantly, and
- ◇ It needs to be extended substantially to handle real programs (e.g. imperfectly-nested and triangular loops).

Background reading

The material in these lectures is taken from the following research paper:

> U. Banerjee, "Unimodular transformations of double loops" [?].

Banerjee's paper gives a simplified account of the theory in the context only of perfect doubly-nested loops with well-known dependences.

Monica Lam's group at Stanford have used these ideas to build a public-domain prototype compiler system "SUIF". The following paper describes how SUIF uses unimodular transformations to optimise locality:

> M.E. Wolf and M.S. Lam, "A data locality optimizing algorithm" [1].

The Stanford work overcomes many of the restrictions of Banerjee's paper, but lies beyond the scope of this course.

Motivation

We begin by considering a very simple loop with a very common dependence pattern. This example is also used in Banerjee's paper:

```
for I1 = 0 to 3 do
  for I2 = 0 to 3 do
S :   A[I1,I2] := A[I1 - 1,I2] + A[I1,I2 - 1]
```

Dependence structure?

Consider the dependence equations:

```
for I1 = 0 to 3 do
  for I2 = 0 to 3 do
S :   A[I1,I2] := A[I1 - 1,I2] + A[I1,I2 - 1]
```

There are two potential dependences arising from the three references to A.

Therefore two systems of dependence equations to solve:

1. Between $\mathtt{A}[I_1^1,I_2^1]$ and $\mathtt{A}[I_1^2-1,I_2^2]$:

$$\begin{cases} I_1^1 = I_1^2 - 1 \\ I_2^1 = I_2^2 \end{cases}$$

2. Between $\mathtt{A}[I_1^1,I_2^1]$ and $\mathtt{A}[I_1^2,I_2^2-1]$:

$$\begin{cases} I_1^1 = I_1^2 \\ I_2^1 = I_2^2 - 1 \end{cases}$$

((strictly we should also consider output dependences between $\mathtt{A}[I_1^1,I_2^1]$ and $\mathtt{A}[I_1^2,I_2^2]$, but this is obviously absent)).

Equation 1 has solutions for

$$\begin{aligned}
((I_1^1,I_2^1),(I_1^2,I_2^2)) &= ((0,0),(1,0)),\\
&= ((1,0),(2,0)),\\
&= ((2,0),(3,0)),\\
&= ((0,1),(1,1)),\\
&= ((1,1),(2,1)),\\
&= ((2,1),(3,1)),\\
&\vdots\\
&= ((0,3),(1,3)),\\
&= ((1,3),(2,3)),\\
&= ((2,3),(3,3))
\end{aligned}$$

We can summarise this by saying that the *dependence distance vector* for this dependence is

$$(I_1^2 - I_1^1, I_2^2 - I_2^1)$$

that is, $(1,0)$.

This corresponds to a *dependence direction vector* $(<,.)$: there is a loop-carried data dependence in the outer loop.

Equation 2 has solutions for

$$\begin{aligned}
((I_1^1,I_2^1),(I_1^2,I_2^2)) &= ((0,0),(0,1)),\\
&= ((0,1),(0,2)),\\
&= ((0,2),(0,3)),\\
&= ((1,0),(1,1)),\\
&= ((1,1),(1,2)),\\
&= ((1,2),(1,3)),\\
&\ldots\\
&= ((3,0),(3,1)),\\
&= ((3,1),(3,2)),\\
&= ((3,2),(3,3))
\end{aligned}$$

The distance vector is

$$(I_1^2 - I_1^1, I_2^2 - I_2^1) = (0,1)$$

This corresponds to a dependence direction vector $(.,<)$: there is also a loop-carried data dependence in the inner loop.

((Note that it is possible for a dependence to exist without having a consistent dependence distance)).

The same loop:

for I_1 = 0 to 3 do
  for I_2 = 0 to 3 do
S : A[I_1,I_2] := A[$I_1 - 1$,I_2] + A[I_1,$I_2 - 1$]

For humans the easy way to understand this loop nest is to draw the iteration space showing the iteration-to-iteration dependences:

$$\begin{array}{ccccccc}
S^{00} & \xrightarrow[\delta]{} & S^{01} & \xrightarrow[\delta]{} & S^{02} & \xrightarrow[\delta]{} & S^{03} \\
\delta\downarrow & & \delta\downarrow & & \delta\downarrow & & \delta\downarrow \\
S^{10} & \xrightarrow[\delta]{} & S^{11} & \xrightarrow[\delta]{} & S^{12} & \xrightarrow[\delta]{} & S^{13} \\
\delta\downarrow & & \delta\downarrow & & \delta\downarrow & & \delta\downarrow \\
S^{20} & \xrightarrow[\delta]{} & S^{21} & \xrightarrow[\delta]{} & S^{22} & \xrightarrow[\delta]{} & S^{23} \\
\delta\downarrow & & \delta\downarrow & & \delta\downarrow & & \delta\downarrow \\
S^{30} & \xrightarrow[\delta]{} & S^{31} & \xrightarrow[\delta]{} & S^{32} & \xrightarrow[\delta]{} & S^{33}
\end{array}$$

The diagram shows an arrow for each solution of each dependence equation.

Note that:

- This loop is *interchangeable*: the top-to-bottom, left-to-right execution order is also valid since all dependence constraints (as shown by the arrows) are still satisfied.
- The inner loop is not vectorisable since there is a dependence chain linking successive iterations.
- Interchanging the loop does not improve vectorisability.

Consider the following loop:

for k$_1$:= 0 to 3 do
  for k$_2$:= k$_1$ to k$_1$+3 do
S : A[k$_1$,k$_2$-k$_1$] := A[k$_1$-1,k$_2$-k$_1$]+A[k$_1$,k$_2$-k$_1$-1]

- The inner loop's control variable runs from k$_1$ to k$_1$+3.
- The iteration space of this loop has 4^2 iterations just like the original loop.
- If we draw the iteration space with each iteration S^{K_1,K_2} at coordinate position (K_1, K_2), it is skewed to form a lozenge shape:

$$\begin{array}{ccccccc}
S^{00} & S^{01} & S^{02} & S^{03} & & & \\
 & S^{11} & S^{12} & S^{13} & S^{14} & & \\
 & & S^{22} & S^{23} & S^{24} & S^{25} & \\
 & & & S^{33} & S^{34} & S^{35} & S^{36}
\end{array}$$

- This loop performs the same computation as the original.

- To see that this loop performs the same computation, lets work out its dependence structure.
- First label each iteration with the element of A to which it assigns:

S^{00} A$_{00}$	S^{01} A$_{01}$	S^{02} A$_{02}$	S^{03} A$_{03}$			
	S^{11} A$_{10}$	S^{12} A$_{11}$	S^{13} A$_{12}$	S^{14} A$_{13}$		
		S^{22} A$_{20}$	S^{23} A$_{21}$	S^{24} A$_{22}$	S^{25} A$_{23}$	
			S^{33} A$_{30}$	S^{34} A$_{31}$	S^{35} A$_{32}$	S^{36} A$_{33}$

- The loop body is
  A[k$_1$,k$_2$-k$_1$] := A[k$_1$-1,k$_2$-k$_1$]+A[k$_1$,k$_2$-k$_1$-1]
- E.g. iteration S_{23} does:
  A[2,1] := A[1,1]+A[2,0]

Thus the dependence structure of the skewed loop is shown by marking the iteration space with all the dependences:

$$\begin{array}{ccccccccccccc}
S^{00} & \xrightarrow[\delta]{} & S^{01} & \xrightarrow[\delta]{} & S^{02} & \xrightarrow[\delta]{} & S^{03} & & & & & & \\
 & \searrow^{\delta} & & \searrow^{\delta} & & \searrow^{\delta} & & \searrow^{\delta} & & & & & \\
 & & S^{11} & \xrightarrow[\delta]{} & S^{12} & \xrightarrow[\delta]{} & S^{13} & \xrightarrow[\delta]{} & S^{14} & & & & \\
 & & & \searrow^{\delta} & & \searrow^{\delta} & & \searrow^{\delta} & & \searrow^{\delta} & & & \\
 & & & & S^{22} & \xrightarrow[\delta]{} & S^{23} & \xrightarrow[\delta]{} & S^{24} & \xrightarrow[\delta]{} & S^{25} & & \\
 & & & & & \searrow^{\delta} & & \searrow^{\delta} & & \searrow^{\delta} & & \searrow^{\delta} & \\
 & & & & & & S^{33} & \xrightarrow[\delta]{} & S^{34} & \xrightarrow[\delta]{} & S^{35} & \xrightarrow[\delta]{} & S^{36}
\end{array}$$

- Can this loop nest be vectorised?
- Can this loop nest be interchanged?

$$\begin{array}{ccccccc}
S^{00} & S^{01} & S^{02} & S^{03} & . & . & . \\
. & S^{11} & S^{12} & S^{13} & S^{14} & . & . \\
. & . & S^{22} & S^{23} & S^{24} & S^{25} & . \\
. & . & . & S^{33} & S^{34} & S^{35} & S^{36}
\end{array}$$

• The interchanged loop traverses the iteration space top-to-bottom, left-to-right.

```
for k2 := ?  to ?  do
  for k1 := ?  to ?  do
S : A[k1,k2-k1] := A[k1-1,k2-k1]+A[k1,k2-k1-1]
```

• The inner loop is now vectorisable.

• The skewed iteration space has N rows and $2N-1$ columns, but still only N^2 actual statement instances.

• What are the appropriate loop bounds?

For $N_1 = 4$, $N_2 = 5$

$$\begin{array}{cccccccc}
S^{00} & S^{01} & S^{02} & S^{03} & . & . & . & . \\
. & S^{11} & S^{12} & S^{13} & S^{14} & . & . & . \\
. & . & S^{22} & S^{23} & S^{24} & S^{25} & . & . \\
. & . & . & S^{33} & S^{34} & S^{35} & S^{36} & . \\
. & . & . & . & S^{44} & S^{45} & S^{46} & S^{47}
\end{array}$$

Interchanged, skewed loop:

```
for k2 := 0 to 8 do
  for k1 := max(0, K2 - 3) to min(K2, 4) do
S : A[k1,k2-k1] := A[k1-1,k2-k1]+A[k1,k2-k1-1]
```

That is,

```
for k2 := 0 to 2N2 - 2 do
  for k1 := max(0, K2 - N2 + 2) to min(K2, N1) do
S : A[k1,k2-k1] := A[k1-1,k2-k1]+A[k1,k2-k1-1]
```


Summary

- Original loop interchangeable but not vectorisable.
- We skewed inner loop by outer loop by factor 1.
- Still not vectorisable, but interchangeable.
- Interchanged, skewed loop *is* vectorisable.
- Bounds of new loop not simple!
- Is this the best vectorisable version of this loop?
  (Banerjee skews the outer loop by the inner loop, see [?]).

Questions about skewing:

— Is skewing ever invalid?

— Does skewing affect interchangeability?

— Does skewing affect dependence distances?

— Can you predict the usefulness of skewing?

Matrix representation of loop transformations

To skew the inner loop by the outer loop by factor 1 we adjust the loop bounds, and replace I_1 by K_1, and I_2 by $K_2 - K_1$. That is,

$$(K_1, K_2) = (I_1, I_2) \cdot \mathbf{U}$$

where $\mathbf{U}$ is a 2×2 matrix

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

That is,

$$(K_1, K_2) = (I_1, I_2) \cdot \mathbf{U} = (I_1, I_2 + I_1)$$

The inverse gets us back again:

$$(I_1, I_2) = (K_1, K_2) \cdot \mathbf{U}^{-1} = (K_1, K_2 - K_1)$$

Matrix $\mathbf{U}$ maps each statement instance $S^{I_1 I_2}$ to its position in the new iteration space, $S^{K_1 K_2}$:

Original iteration space:

	I_2 :			
I_1	0	1	2	3
0	S^{00}	S^{01}	S^{02}	S^{03}
1	S^{10}	S^{11}	S^{12}	S^{13}
2	S^{20}	S^{21}	S^{22}	S^{23}
3	S^{30}	S^{31}	S^{32}	S^{33}

Transformed iteration space:

	K_2 :						
K_1	0	1	2	3	4	5	6
0	S^{00}	S^{01}	S^{02}	S^{03}			
1		S^{11}	S^{12}	S^{13}	S^{14}		
2			S^{22}	S^{23}	S^{24}	S^{25}	
3				S^{33}	S^{34}	S^{35}	S^{36}

The dependences are subject to the same transformation.

Recall that:

- There is a dependence between two iterations (I_1^1, I_2^1) and (I_1^2, I_2^2) if there is a memory location which is assigned to in iteration (I_1^1, I_2^1), and read in iteration (I_1^2, I_2^2).
  ((unless there is an intervening assignment))
- If (I_1^1, I_2^1) precedes (I_1^2, I_2^2) it is a *data*-dependence.
- If (I_1^2, I_2^2) precedes (I_1^1, I_2^1) it is a *anti*-dependence.
- If the location is assigned to in both iterations, it is an *output*-dependence.
- The dependence distance vector (D_1, D_2) is $(I_1^2 - I_1^1, I_2^2 - I_2^1)$.

- Iterations $(I_1^1, I_2^1) \cdot \mathbf{U}$ and $(I_1^2, I_2^2) \cdot \mathbf{U}$ will also read and write the same location.
- The transformation $\mathbf{U}$ is *valid* iff

  $$(I_1^1, I_2^1) \cdot \mathbf{U} \text{ precedes } (I_1^2, I_2^2) \cdot \mathbf{U}$$

  whenever there is a dependence between (I_1^1, I_2^1) and (I_1^2, I_2^2).
- In the transformed loop the dependence distance vector is also transformed, to

  $$(D_1, D_2) \cdot \mathbf{U}$$

Example: loop given earlier

Before transformation we had two dependences:

1. Distance: $(1, 0)$, direction: $(<, .)$
2. Distance: $(0, 1)$, direction: $(., <)$

After transformation by matrix $\mathbf{U} = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}$

(i.e. skewing of inner loop by outer) we get:

1. Distance: $(1, 1)$, direction: $(<, <)$
2. Distance: $(0, 1)$, direction: $(., <)$

We can also represent loop interchange by a matrix transformation.

After transforming the skewed loop by matrix $\mathbf{V} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$ (i.e. loop interchange) we get:

1. Distance: $(1, 1)$, direction: $(<, <)$
2. Distance: $(1, 0)$, direction: $(<, .)$

The transformed iteration space is the transpose of the skewed iteration space:

$$\begin{array}{cccc} S^{00} & & & \\ S^{10} & S^{11} & & \\ S^{20} & S^{21} & S^{22} & \\ S^{30} & S^{31} & S^{32} & S^{33} \\ & S^{41} & S^{42} & S^{43} \\ & & S^{52} & S^{53} \\ & & & S^{63} \end{array}$$

Summary

- Post-multiplication $(I_1, I_2) \cdot \mathbf{U}$ maps each statement instance (I_1, I_2) to its new position (K_1, K_2) in the transformed loop's execution sequence.
- $(D_1, D_2) \cdot \mathbf{U}$ gives the new dependence distance vector, and allows testing for validity.
- This captures skewing, interchange and reversal.
- We can compose transformations by matrix multiplication $\mathbf{U_1} \cdot \mathbf{U_2}$.
- Working out the resulting loop's bounds is a little tricky.

References

[1] M.E. Wolf and M.S. Lam. A data locality optimizing algorithm. In *Proceedings of the ACM SIGPLAN '91 Conference on Programming Language Design and Implementation*, volume 26, pages 30–44, Toronto, Ontario, Canada, June 1991.