Lecture 11
Loop Optimization and Array Analysis

I. Motivation
II. Data dependence analysis

Chapter 11.1-11.1.4, 11.6

Diagram
Software Pipelining

for $i = 1$ to $n$


$\text{LD}_1$

loop $N-1$ times

$\text{LD}_{i+1} \quad \text{ST}_i$

$\text{ST}_n$

• Can I software pipeline the loop?
Software Pipelining

for $i = 1$ to $n$

\[ A[i] = A[i+n] \]

\[ \text{LD}_1 \]

\[ \text{loop N-1 times} \]

\[ \text{LD}_{i+1} \quad \text{ST}_i \]

\[ \text{ST}_n \]

for $i = 1$ to $n$


- Can I software pipeline the loop?
Software Pipelining

for $i = 1$ to $n$

$A[i] = A[i-1]$

LD $A[0]$
loop $N$ times
ST $A[i]$

for $i = 1$ to $n$

$A[i] = A[i+1]$

LD$_1$
loop $N-1$ times
LD$_{i+1}$  ST$_i$

ST$_n$

• Can I software pipeline the loop?
  – We moved LD$_{i+1}$ ahead of store$_i$
    • In case 1, load of $a[i]$ ahead of store $a[i]$
    • In case 2, load of $a[i+2]$ ahead of store $a[i]$

CS243: Loop Optimization and Array Analysis

Dror E. Maydan
Loop Interchange

for i = 1 to n
    for j = 1 to n
        \[ A[j][i] = A[j-1][i-1] \times b[i] \]

- Should I interchange the two loops?
  - Stride-1 accesses are better for caches
  - But one more load in the inner loop
  - But one less register needed to hold the result of the loop
Loop Interchange

for $i = 1$ to $n$

for $j = 1$ to $n$

$A[j][i] = A[j-1][i-1] * b[i]$

Iteration Space

Dependence Vectors
Data Dependences

Do two array references, nested in some loop nest and perhaps some if statements, refer to overlapping memory locations

- As a first step, we check for overlap but don’t care about the nature of dependence

Type of dependences

- True dependence:
  \[ a[i] = a[i-1] \ldots \]

- Anti-dependence:
  \[ a[i] = a[i+1] \]

- Output dependence
  \[ a[i] = \]
  \[ a[i] = \]

Note that we can’t distinguish true from anti by just checking for overlap
Memory Disambiguation

• In general undecidable at compile time

read(n)
for i =
    a[i] = a[n]+ 3

if n > 2
    if a > 0
        if b > 0
            if c > 0
                x[a^n] = x[b^n + c^n]

Can the compiler solve Fermat’s last theorem?
Affine Array Accesses and Bounds

• Common patterns of data accesses: (i, j, k are loop indexes, m is a loop invariant constant)

\[ A[i,j], A[i-1, j+1], A[i+m, j] \]

• Array indexes are affine expressions of surrounding loop indexes and loop invariant constants
  – Loop indexes: \( i_n, i_{n-1}, \ldots, i_1 \)
  – Integer constants: \( c_x, c_{x-1}, \ldots, c_0 \)
  – Loop invariant constants: \( m_y, m_{y-1}, \ldots, m_1 \)
  – Array index: \( c_n i_n + c_{n-1} i_{n-1} + \ldots + c_1 i_1 + c_x m_x + c_{x-1} m_{x-1} + \ldots + c_{n+1} m_1 + c_0 \)
  – Affine expression: linear expression + a constant term \( (c_0) \)

• Loop bounds are affine expressions of surrounding loop indexes and loop invariant constants
  – for \( i \)
    • for \( (j=2*i; j<3*i+7; j+=2) \)
Single Loop, Single Array Dimension

FOR $i := 2$ to $5$ do

- Between read access $A[i]$ and write access $A[i-2]$ there is a dependence if:
  - there exist two iterations $i_r$ and $i_w$ within the loop bounds, s.t.
  - iterations $i_r$ & $i_w$ read & write the same array element, respectively
  
  $\exists$ integers $i_w, i_r \quad 2 \leq i_w, i_r \leq 5 \quad i_r = i_w - 2$

- Between write access $A[i-2]$ and write access $A[i-2]$ there is a dependence if:

  $\exists$ integers $i_w, i_v \quad 2 \leq i_w, i_v \leq 5 \quad i_w - 2 = i_v - 2$

- To rule out the case when the same instance of the same reference depends on itself:
  - add constraint $i_w \neq i_v$

- For now, just asking if there is overlap, not the nature of the overlap
Multiple dimensions, Nested Loops

• Only use loop bounds and array indexes that are affine functions of loop variables and loop invariant constants

```plaintext
for i = 1 to n
    for j = 2*i to 100
        a[i+2*j+3][4*i+2*j][i*i] = ...
        ...
        = a[1][2*i+1][j]
```

• Assume a data dependence between the read & write operation if there exists:
  – a read instance with indexes \( i_r, j_r \) and
  – a write instance with indexes \( i_w, j_w \)

\[
\exists \text{integers } i_r, j_r, i_w, j_w \\
1 \leq i_w, i_r \leq n \\
2i_w \leq j_w \leq 100 \\
2i_r \leq j_r \leq 100 \\
i_w + 2j_w + 3 = 1 \\
4i_w + 2j_w = 2i_r + 1
\]

• Equate each dimension of array access; ignore non-affine ones
  – No solution \( \rightarrow \) No data dependence
  – Solution \( \rightarrow \) there may be a dependence
General Formulation of Data Dependence Analysis

For every pair of accesses not necessarily distinct one must be a write operation
\[ \exists \text{ integers } i_1, i_2 \quad B_1 i_1 + b_1 \geq 0, \quad B_2 i_2 + b_2 \geq 0 \]
\[ F_1 i_1 + f_1 = F_2 i_2 + f_2 \]

If the accesses are not distinct, then add the constraint \( i_1 \neq i_2 \)

- Equivalent to integer linear programming

\[ \exists \text{ integer } i \quad A_1 i \leq b_1 \quad A_2 i = b_2 \]

- Integer linear programming is NP-hard
  \[ O(n^n) \text{ where } n \text{ is the number of variables} \]
Data Dependence Analysis Algorithm

- Typically solving many tiny, repeated problems
  - Integer linear programming packages optimize for large problems
  - Use memoization to remember the results of simple tests

- Apply a series of relatively simple tests
  - Extended GCD, Single Variable Per Constraint Test, ...

- Backed up by a more expensive algorithm
  - Fourier-Motzkin
GCD Test

- Ignore Bounds

\[ \exists \text{ Integer vector } \begin{bmatrix} i_1 \\ A_1 \end{bmatrix} = \begin{bmatrix} b_1 \end{bmatrix} \]

- Single equation

\[ a_n i_n + a_{n-1} i_{n-1} + \ldots + a_1 i_1 = b \]

There exists an integer solution iff the gcd(\(a_n, a_{n-1}, \ldots\)) divides b

Euclid's Algorithm (around 300 BC)
- Assume a and b are positive integers, and a > b.
- Let c be the remainder of a/b.
  - If c=0, then gcd(a,b) = b.
  - Otherwise, gcd(a,b) = gcd(b,c).
- gcd(\(a_1, a_2, \ldots, a_n\)) = gcd(gcd(\(a_1, a_2\)), a_3 \ldots, a_n)

\[
\begin{align*}
gcd(35,14) &= 7 \\
gcd(14,7) &= 7
\end{align*}
\]
Multiple equations: $A_i = b$

- $x - 2y + z = 0$
- $3x + 2y + z = 5$

$x = 2y - z$; sub into second equation

$6y - 3z + 2y + z = 5$, $8y - 2z = 5$

Replace 5 with 4

$8y - 2z = 4$

Many solutions but simplified system of equations for further analysis
GCD Test

- **Multiple equations**: $Ai = b$
  
  
  
  $$2x - 2y + z = 0$$
  $$3x + 2y + z = 5$$

  Lcm of 2 and 3 is 6
  $$6x - 6y + 3z = 0$$
  $$6x + 4y + 2z = 10$$

  $6x = 6y - 3z$, sub into second equation
  $$6y - 3z + 4y + 2z = 10$$
  $$10y - z = 10$$
After GCD Test

∃ integer \( i \quad A_1 i \leq b_1 \)

- Where \( i \) is a reduced set of variables subject to the bounds constraints
  - Originally say we had an \( n \)-dimensional array nested \( m \) deep
  - \( N \) equality constraints, \( 2m \) variables
  - Assuming equality constraints are independent, each equality constraint eliminates one variable
  - \( 2m - N \) variables
SVPC Test

for i1 = 1 to 10
  for i2 = 1 to 10
    a[i1][i2] = a[i2 + 10][i1 + 9]

i1 = i2' + 10
i2 = i1' + 9

1 <= i1 <= 10
1 <= i2 <= 10
1 <= i1' <= 10
1 <= i2' <= 10
1 <= i2' + 10 <= 10, -9 <= i2' <= 0
i2' can't be both <= 0 and >= 1 so the system is independent
Fourier-Motzkin Elimination

• Linear Programming
• Solve a matrix linear inequality in the real domain

\[ \exists \text{ integer } i \quad A_i i \leq b_1 \]

If there doesn’t exist a real solution, there doesn’t exist an integer one.
Fourier-Motzkin Elimination

• To eliminate a variable from a set of linear inequalities.
• To eliminate a variable \( x_1 \)
  – Rewrite all expressions in terms of lower or upper bounds of \( Cx_1 \)
  – Create a transitive constraint for each pair of lower and upper bounds.
• Example: Let \( L, U \) be lower bounds and upper bounds resp
  – To eliminate \( x_1 \):

\[
\begin{align*}
L_1(x_2, \ldots, x_n) & \leq Cx_1 \leq U_1(x_2, \ldots, x_n) \\
L_2(x_2, \ldots, x_n) & \leq Cx_1 \leq U_2(x_2, \ldots, x_n)
\end{align*}
\]

If at any point, you get an inconsistent equation; e.g. \( 3 < 2 \), independent

Otherwise there exists a real, but not necessarily integral, solution
Fourier-Motzkin Elimination with Integer Constraints

- One variable and a series of bounds eg $x_1 > 2$ and $x < 3$
  - Does the range contain an integer
    - No, no integral solution
    - Yes, $x_1 > 2$ and $x_1 < 13$
      - pick an integer in the middle of the range, eg 7 and go back to the equation with two variables
- More than one variable and a series of bounds
  - Does the range contain an integer
    - No, might be an integral solution. Branch and bound
      - $x = 3.5$. Add one constraint each to two new problems, $x \leq 3$ and $x \geq 4$. Start over. If neither have a solution, there is no solution
    - Yes, pick an integer in the middle of the range, and go back to the equation with one more variable
Example

\[
\text{FOR } i = 1 \text{ to } 5 \\
\quad \text{FOR } j = i+1 \text{ to } 5 \\
\quad \quad A[i,j] = f(A[i,i], A[i-1,j])
\]

where:

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

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

\[A[i,j] \text{ vrs } A[i', i']\]
\[i' = i, i' = j\]

\[
\begin{align*}
1 \leq i; i \leq 5; i + 1 & \leq j; j \leq 5; \\
1 \leq i'; i' \leq 5; \\
1 \leq i; i \leq 5; & \quad \text{// redundant}
\end{align*}
\]

\[
\begin{align*}
i' + 1 & \leq j'; j' \leq 5; \\
i + 1 & \leq i \quad \text{equivalent to } 1 \leq 0 \quad \text{// inconsistent so independent}
\end{align*}
\]
Example

FOR $i = 1$ to $5$
  FOR $j = i+1$ to $5$
    $A[i,j] = f(A[i,i], A[i-1,j])$

$A[i,j]$ vrs $A[i'-1, j']$
$i' = i+1, j' = j$

1. $1 \leq i; i \leq 5; // i$ bounds
2. $i+1 \leq j; j \leq 5; // j$ bounds
3. $1 \leq i+1; i+1 \leq 5; // i'$ bounds, equivalent to $0 \leq i; i \leq 4$
4. $i+2 \leq j; j \leq 5; // j'$ bounds

$1 \leq i \leq 4; // combine 1 and 3$
  $i+2 \leq j; j \leq 5; // 4$
i+2 \leq 5, // First step of Fourier-Motzkin on 4, equivalent to $i \leq 3$

Pick $i=2, j=4, i' = 3, j' = 4$  
**Distance Vectors**

for $i = 1$ to $n$

for $j = 1$ to $n$

$$A[j][i] = A[j-1][i-1] \times b[i]$$

$j_w = j_r - 1$, $i_w = i_r - 1$

$j_r - j_w = 1$, $i_r - i_w = 1$

We say there is a dependence from the write to the read of distance (1, 1)

How do we compute distances

**GCD Test:** look for normalized equations during gcd test $i' - i = c$

**More sophisticated:** Add constraint $i' - i = d$ to Fourier-Motzkin and see if it is constrained to a single value
Direction Vectors

for i = 1 to n
   for j = 1 to i

$j_r = i_w$
$j_w <= i_w$
$j_r <= i_r$

$j_r - j_w >= 0$
$i_r - i_w >= 0$

We say there is a dependence from the write to the read of direction ($>=, >=$)
Direction Vectors

Computing Direction Vectors

Start with nothing (*, *, ...)  
Independent: done
Dependent
  check (> , *, *, ...) , (=, *, *, ...) and (<, *, *, ...)
  Add in a constraint i' - i > 0, etc
If, for example, (<, *, *, ...) is dependent
  check (<, >, *, ...), (<, =, *, ...) and (<, <, *, ...)

Exponential blow-up
  Don’t do it for constant distances
Unused variables are automatically *
  for i = 1,m
    for j = 1,n
      a[j] = a[j+1].
Lexicographically Positive Direction Vectors

There is a dependence from the write to the read of direction (>=, >=)
   All this means is that the two references may refer to the same location on all
   iterations when ir >= iw and jr >= jw

If jr - jw = some vector d (or a set of vectors), clearly jw - jr = -d (for all d)
   So we don’t run dependence analysis twice, once for the write versus the read and
   once for the read versus the write

   Instead we run it once and negate the result to get the opposite edges

Dependence vectors are a relation between two iterations of two references. We choose
to draw the dependence from the reference with the earlier iteration to the latter
Why?
   We have a property that a location is written by the source of an edge before the
   sink
   We don’t want to change the order in which we access any memory location
      As long as after transformation, the edges are still positive, the
   transformation is legal
Lexicographically Positive Direction Vectors

- Represent dependences as lexicographically positive
  - First non-equal component must be “>”.
  - All “=” vectors only allowed if source precedes sink in same iteration

- There was a dependence from the write to the read of direction (>=, >=)
  - In a given iteration, the read happens before the write

  - Dependence from write to read of the positive components of (>=, >=)
    - (> , > ), (= , >)
    - Don’t include (=, =) because the read happens before the write

  - Dependence from read to write of positive components of (<=, <=)
    - (= , =)
void foo(float *a, float *b, int n)
{
    for (int i=0; i<n; i++) {
        a[i] = b[i] + 1.0;
    }
}

if (a[n-1] < b[0] || b[n-1] < a[0]) {

If all you care about is software pipelining, you want to move b up, not a, and a common scenario is a==b
care if b[i] == a[i-c] for c> 0 (actually c can't be from 1 to max schedule of SWP)

if (a[0] <= b[0] || b[5] < a[0]) {

Reduction

for (i=0; i<n; i++)
  for (j=0; j<n; j++)
    A[3] += b[i][j];

Dependence from write to itself is (> , *), (= , >)
Loop interchange will change that to (* , >), (> , =) which has negative components

- Is it safe?