CSC2/455 Software Analysis and Improvement
Array/Loop Dependence Analysis

Sreepathi Pai
March 14, 2022
URCS
Outline

Loop Analysis and Transformation

Characterizing loop dependences

Identifying Loop Dependences

Distance and Direction Vectors

Postscript
Outline

Loop Analysis and Transformation

Characterizing loop dependences

Identifying Loop Dependences

Distance and Direction Vectors

Postscript
Why Loop Transformations

- Potentially lots of computation
  - A few operations execute many times
- Potentially lots of memory accesses
- Array-based data structures show up frequently
  - Matrices, vectors, etc.
- Loops are naturally paired with arrays
- FORTRAN
  - FORMula TRANslator
  - World’s first high-level programming language
Important Applications

- Scientific Computing/Computational Science
  - Simulation of Galaxies, Molecules, etc.
  - Drug Discovery
- Audio/Video Processing
  - Signal Processing
  - Compression
- Machine Learning (specifically Deep Learning)
  - Recognizing cats
  - Showing targeted ads
• Multiplying two matrices:
  • A \((m \times n)\)
  • B \((n \times k)\)
  • C \((m \times k)\) [result]

• Here: \(m = n = k\)

```c
for(ii = 0; ii < m; ii++)
  for(jj = 0; jj < n; jj++)
    for(kk = 0; kk < k; kk++)
      C[ii * k + kk] += A[ii * n + jj] * B[jj * k + kk];
```
Matrix Multiply – IKJ

for(ii = 0; ii < m; ii++)
    for(kk = 0; kk < k; kk++)
        for(jj = 0; jj < n; jj++)
            C[ii * k + kk] += A[ii * n + jj] * B[jj * k + kk];
Performance of the two versions?

- on 1024x1024 matrices of ints
- which is faster?
- by how much?
Performance of the two versions

- on 1024x1024 matrices
- Time for IJK: 0.554 s ± 0.003s (95% CI)
- Time for IKJ: 6.618 s ± 0.032s (95% CI)
What caused the nearly 12X slowdown?

- Matrix Multiply has a large number of arithmetic operations
  - But the number of operations did not change
- Matrix Multiply also refers to a large number of array elements
  - Order in which they access elements changed
  - Memory hierarchy/Caches affect cost of each access
Loop Analysis

Which *iterations* of a loop depend on other iterations?

- Once dependences are identified
  - can change order
  - can vectorize
  - can redistribute across processors
Outline

Loop Analysis and Transformation

Characterizing loop dependences

Identifying Loop Dependences

Distance and Direction Vectors

Postscript
• Definition: Two *dynamic* statements have a dependence if:
  • Both access same location (memory or register)
  • And one of the accesses is a write
• *Dynamic* required, since we’re talking about loops (examples later)
Dependence types

$S_1$ occurs “earlier” than $S_2$ (in the dynamic trace)

- True dependence
  - $S_1\delta S_2$
  - $S_1$ writes, $S_2$ reads

- Anti-dependence
  - $S_1\delta^{-1} S_2$
  - $S_1$ reads, $S_2$ writes

- Output dependence
  - $S_1\delta^o S_2$
  - Both $S_1$ and $S_2$ write
What are the dependences in the loop body below?

Can you change the order of the statements in the loop body?

```fortran
DO I = 0, 9
   A(I) = A(I) + B
   C(I) = A(I) + D
ENDDO
```

Can you change the (execution) order of loop iterations?

Note: FORTRAN uses parentheses in array references: e.g., `A(I)`. FORTRAN arrays usually start at 1, but for this lecture, we will assume they start at 0.
Loop-independent dependences visualized

NOTE: Only dependences from first four iterations visualized.
Loop-carried dependences

- What are the dependences in the loop body below?
- Can you change the order of the statements in the loop body?

DO I = 0, 9
   A(I + 1) = A(I) + B
   C(I) = A(I) + D
ENDDO

- Can you change the (execution) order of loop iterations?
Loop-carried dependences visualized

NOTE: Only dependences from first three iterations visualized.
DO I = 0, 9
  DO J = 0, 1
    A(I + 1, J) = A(I, J) + 1
  ENDDO
ENDDO

- Can you change the order of inner loop?
- Can you change the order of the outer loop?
A(0 + 1, 0) = A(0, 0) + 1
A(1 + 1, 0) = A(1, 0) + 1
A(2 + 1, 0) = A(2, 0) + 1
A(0 + 1, 1) = A(0, 1) + 1
A(1 + 1, 1) = A(1, 1) + 1
A(2 + 1, 1) = A(2, 1) + 1

NOTE: Only dependences from first three iterations visualized.
Loop Dependences

- Loop-independent dependence
  - In same iteration, independent of loops
- Loop-carried dependence
  - Across different iterations of at least one loop
- Dependence Level of a Loop-carried Dependence
  - The nesting level $k$ of loop that carries the dependence
  - $S_1 \delta_k S_2$
Iteration Spaces

DO I = 1, 2
    DO J = 1, 2
        S
    ENDDO
ENDDO

- $S$ has four instances $(I, J)$: $(1, 1), (1, 2), (2, 1), (2, 2)$
- Each of these values represents an *iteration vector*
  - Particular values of loop indices
  - Ordered from outermost loop to innermost loop
Iterate $J = 1, 10$

$\text{DO } I = 1, 10$

$\quad A(I+1, J) = A(I, J) + X$

$\text{ENDDO}$

$\text{ENDDO}$

Assuming $A$ starts from 1. FORTRAN allows you to change the "origin" of arrays.
For two vectors $i = (i_1, i_2, ..., i_n)$ and $j = (j_1, j_2, ..., j_n)$, each containing $n$ elements, $i < j$ if there exists $m \in [0, n)$, such that:

- $i_x = j_x$ for $x < m$
- $i_m < j_m$
For two vectors \( i \) and \( j \), each containing \( n \) elements, \( i < j \) is defined as:

```python
def lessthan(i, j, n):
    if n == 1:
        return i[0] < j[0]

    # test prefix for elementwise-equality
    if i[0:n-1] == j[0:n-1]:
        return i[n-1] < j[n-1]
    else:
        return lessthan(i, j, n-1)
```

Can similarly define other order relations.
Loop dependence

Dependence from Statement S1 (source) to statement S2 (sink) if:

- There exist iteration vectors $i$ and $j$ such that $i < j$ or $i = j$
- There is a path from S1 to S2 in the loop
- S1 accesses memory location M in iteration $i$
- S2 accesses memory location M in iteration $j$
- and one of the accesses is a write
Outline

Loop Analysis and Transformation

Characterizing loop dependences

Identifying Loop Dependences

Distance and Direction Vectors

Postscript
Generalizing Loop Indices

DO \text{I}_1 = \ldots \\
DO \text{I}_2 = \ldots \\
\ldots \\
DO \text{I}_N = \ldots \\
A(f_1, f_2, f_3, \ldots, f_M) = \ldots \\
\ldots = A(g_1, g_2, g_3, \ldots, g_M) \\
ENDDO \\
ENDDO \\
ENDDO

where $A$ is $M$-dimensional array, and $f_X$ and $g_X$ are \textit{index functions} of the form

- $f_X(l_1, l_2, \ldots, l_N)$
- $g_X(l_1, l_2, \ldots, l_N)$
- $1 \leq X \leq M$
Dependence using Iteration Vectors

Let $\alpha$ and $\beta$ be iteration vectors:

- $\alpha = (i_1, i_2, i_3, ..., i_N)$
- $\beta = (i'_1, i'_2, i'_3, ..., i'_N)$

Then a dependence exists if:

- (vectors) $\alpha < \beta$
- $f_X(\alpha) = g_X(\beta)$, for $1 \leq X \leq M$
Example

DO J = 0, 9
    DO I = 0, 9
        A(I+1, J) = A(I, J) + X
    ENDDO
ENDDO

• $f_1(J, I) = I + 1$, $f_2(J, I) = J$

• $g_1(J, I) = I$, $g_2(J, I) = J$

• For $\alpha = (0, 0)$ (i.e. $J = 0, I = 0$) and $\beta = (0, 1)$ (i.e. $J = 0, I = 1$):
  • $f_1(\alpha) = g_1(\beta)$, i.e. $1 = 1$
  • $f_2(\alpha) = g_2(\beta)$, i.e. $0 = 0$
  • Many other values of $\alpha$ and $\beta$ also satisfy these equations.
Dependence Testing

Do iteration vectors $\alpha$ and $\beta$ exist such that:

- (vectors) $\alpha < \beta$
- $f_X(\alpha) = g_X(\beta)$, for $1 \leq X \leq M$

How can we find $\alpha$ and $\beta$ if they exist?
Restrictions on Index functions

- $f_X$ and $g_X$ must be decidable (i.e. computable)
- $f_X$ and $g_X$ must be "analyzable"
  - to avoid brute force search
Affine Index Functions

- Let $f_X$ and $g_X$ must be affine functions of loop indices:
  - i.e. for $f_X(i_1, i_2, i_3, ..., i_n)$
  - $f_X = a_1 i_1 + a_2 i_2 + ... + a_n i_n + e$
  - $e$ is optional loop invariant calculation (i.e. constant for the loop)
Given that $f_X$ and $g_X$ are affine functions of loop indices.

Do iteration vectors $\alpha$ and $\beta$ exist such that:

- (vectors) $\alpha < \beta$
- $f_X(\alpha) = g_X(\beta)$, for $1 \leq X \leq M$

How can we find $\alpha$ and $\beta$ if they exist?

What is this problem better known as? Hint: an affine function is a linear function plus constant.
Dependence Testing

- Integer Linear Programming is NP-complete
- Lots of heuristics invented
  - Profitable to know if no solution exists since it implies no dependence!
  - See Chapter 3 of AK
  - Or Chapter 11 of the Dragon Book
  - We will cover this in a later class
Outline

Loop Analysis and Transformation

Characterizing loop dependences

Identifying Loop Dependences

Distance and Direction Vectors

Postscript
Representing Dependences

Do we need to track all the iterations that have a dependence explicitly (e.g. in a list)?
Distance Vectors

\[ d(i, j)_k = j_k - i_k \]

- Where \( i, j, d(i, j) \) are \( n \)-element vectors
- \( i_k \) indicates \( k \)-th element of \( i \)

Example distance vector: (0, 1)
Direction Vectors

\[ D(i, j)_k = \]

- "\(<\)" , if \(d(i, j)_k > 0\)
- "\(=\)" , if \(d(i, j)_k = 0\)
- "\(>\)" , if \(d(i, j)_k < 0\)

Example direction vector for \((0, 1)\): \((=, <)\)
Information we need to track

For every pair of memory references:

- Iteration Vectors $i$ and $j$ which have a dependence, or
- Unique Distance Vectors $d(i, j)$, or
- Unique Direction Vectors $D(i, j)$
Which of these indicates a loop-independent dependence?

- \((=, =)\)
- \((=, <)\)

Of the loop-carried dependence in example above, what level is the loop-carried dependence?
Theorems

WARNING: Informal language

- **Direction Vector Transform (Theorem 2.3 in AK)**
  - If a transformation reorders loop iterations, and preserves the leftmost non-"=" component as "<", all dependences are preserved.

- **Theorem 2.4 in AK**
  - If a level-\(k\) dependence exists, and a transformation reorders loop iterations while not reordering the level-\(k\) loop
  - And does not move loops inside \(k\) outside the loop and vice versa
  - It preserves all level-\(k\) dependences.

- **Iteration Reordering (Theorem 2.6 in AK)**
  - Iterations of a level \(k\) loop can be reordered if there is no level \(k\) dependence.
Outline

Loop Analysis and Transformation

Characterizing loop dependences

Identifying Loop Dependences

Distance and Direction Vectors

Postscript
• Much of this lecture is based on Allen and Kennedy, Optimizing Compilers for Modern Architectures, Chapter 2.
• Chapter 11 of the Dragon Book also presents this information, but differently.