Key concepts

Our focus here: perfectly-nested loops

Loop Transformsations

Introduction
loop translation ↔ change of basis for iteration space
loop body instances ↔ iteration space of loop

Powerful way of thinking of partially-nested loop execution and
case
There are other loop translations that we will discuss in
for locality enhancement: permutation and shift
We have seen two key translations of partially-nested loops

Goal of lecture:

Correspondence to loop iterations:

Heretion space of a perfectly-nested loop

Heretion order = lexicographical order on iteration space:

N 1 1
DO I = 1, N

K 1
DO K = 1, M

I 1
DO I = 1, N

M 1 1
DO M = 1, N

Each iteration of a loop nest with n loops can be viewed as an
integer point in an n-dimensional space:

(\mathbf{N}_1, \mathbf{N}_n) \equiv \cdots (z, z) \equiv (1, 1) \equiv \cdots \equiv (z, 1) \equiv (1, 1)

Loop permutation = linear transformation on heretion space

"doesnt" together, so probability of cache hits is increased.

Loop permutation brings iterations that touch the same cache line

Locality enhancement.
DO  J = 1, M
DO  I = 2, N

Assume that array has 1's stored everywhere before loop begins.

After loop permutation:

Transformed loop will produce different values (A[3,1] for example)

Example: Index set loop bounds (transpose order/axes): DO  I = 1, N
DO  J = 1, I
DO  L = I, N
DO  K = 1, N

Question: How do we determine when loop permutation is legal?

If transformed loop produces different values (A[1,1] for example)

DO  I = 1, N
DO  J = 2, N
DO  L = 1, M
DO  K = 1, L

Subtle Issue 1: Loop permutation may be legal in some loop nests.

Question: How do we generate loop bounds for transformed loop nest?

Here, inner loop bounds are functions of outer loop indices.

$S$
FOR J = 1, I-1
FOR I = 1, N

Non-trivial
2. formulate code generation problem as ILP

Problems: (ii) problem

1. formulate correctness of permutation as integer linear

Goal:

Desirable: qualitative estimates of performance improvement

- when the transformed code should be:
- when target architecture, and

- when the best sequence of transformations should be for a
- when transformations are legal.

General theory of loop transformations should tell us

and code generation can be reduced to these problems:

(iii) enumerate all integer solutions

(ii) are there integer solutions?

where \( A \) is a \( m \times n \) matrix of integers,
\( x \) is an \( n \) vector of unknowns,
\( b \) is an \( m \) vector of integers.

Given a system of linear inequalities
\( Ax < b \)

Two problems:
Let us look at dependences.

What does independent mean?

This is stronger than we need, but it is a good starting point.

Determination is certainly hard

Intuition: If all dependences of a loop nest are independent, then

Let us formulate correctness of loop permutation as ILP problem.

Input dependence is not usually important for most applications.

\( L = x \)
\( x = x \)
\( 1 + x = x \)

Dependences:
- control
- data
- flow

(x) \( L = x \) and \( R = x \) from the same location
- control dependence \( R \leq L \)
- data dependence \( R \leq L \)
- flow dependence \( R \leq L \)

(x) \( R \) and \( L \) refer to the same location
- control dependence \( R \leq L \)
- data dependence \( R \leq L \)
- flow dependence \( R \leq L \)

Flow dependence \( R < L \)

(x) \( R \) writes a location that is overwritten later by \( L \)
- control dependence \( R \leq L \)
- data dependence \( R \leq L \)
- flow dependence \( R \leq L \)

Output dependence \( R \leq L \)

Input dependence: \( S_1 \rightarrow S_2 \)
Output dependence: \( S_1 \rightarrow S_2 \)
Anti-dependence: \( S_1 \rightarrow S_2 \)
Flow dependence: \( S_1 \rightarrow S_2 \)

(i) \( S_1 \) executes before \( S_2 \)
(ii) \( S_1 \) and \( S_2 \) both read from the same location
(iii) \( S_1 \) and \( S_2 \) write to the same location
(iv) \( S_1 \) executes before \( S_2 \)
(v) \( S_1 \) reads from a location that is overwritten later by \( S_2 \)
(vi) \( S_1 \) writes into a location that is read by \( S_2 \)
(vii) \( S_1 \) executes before \( S_2 \) in program order

\( x := 2 \)
\( y := x + 1 \)
\( x := 3 \)
\( y := 7 \)

Equality: line (2D), plane (3D), hyperplane (>3D)

Intuition about systems of linear inequalities:

Inequality: half-plane (2D), half-space (>2D)

Region described by inequality is convex
Region described by inequalities is a convex polyhedron

Conjunction of inequalities = intersection of half-spaces

Equality: line (2D), plane (3D), hyperplane (>3D)

Region described by inequality is convex
Region described by inequalities is a convex polyhedron

\( x + y = 12 \)
\( 3x + y = 12 \)
\( x \geq -5 \)
\( 3x - 3y \leq 9 \)
\( y \leq 4 \)

\( x, y \)
Dependences in loops

\[ (x(I))^2 = (x(I))^2 \]
\[ \cdots - (x(I))^2 \]
\[ \text{FOR } 10 \leq I \leq 100 \]

Dependences in nested loops

\[ (x(I))^2 = (x(I))^2 \]
\[ \cdots - (x(I))^2 \]
\[ \text{FOR } 10 \leq I \leq 100 \]

Conservative Approximation:

- Real programs: Imprecise information => need for safe approximation
- How do we compute dependences between iterations of a loop nest?

Dependence between iterations:

Execution of a statement for given loop index values

Franchise Analysis:

- Franchise Analysis: How do we compute dependences between iterations of a loop nest?

In the loop body:

- Dependence on a dynamic instance (I,J) of a statement
- Dependence on a static instance (I,J) of a statement in loop body

Iteration (I,J) is said to depend on iteration (I,J) if

\[ x(I) \neq x(J) \]

in the loop body.
\[ \begin{align*}
\frac{m_i}{1} & \geq 1 + a_i \\
1 + a_i & \geq \frac{m_i}{1} \\
0 & \leq a_i \\
1 - a_i & \leq 0 \\
\frac{m_i}{1} & \geq 1 \\
\end{align*} \]

which can be written as

\[ I + a_i \leq \frac{m_i}{1} \leq 1 \]

Is there a flow dependence between different iterations?

\[ X(2i), \ldots, X(2i+1) = \cdots \]

For \( i = 1, 2, \ldots, 100 \)

\[ \frac{m_i}{1} \]

ILP Formulation

\[ \begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} m_i \\ a_i \\ 1 \end{bmatrix} \begin{bmatrix} 1 & -1 & 0 \\ -1 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \]

can be expressed in the form \( x \leq 0 \) as follows

\[ \begin{align*}
\frac{m_i}{1} & \geq 1 + a_i \\
1 + a_i & \geq \frac{m_i}{1} \\
0 & \leq a_i \\
1 - a_i & \leq 0 \\
\frac{m_i}{1} & \geq 1 \\
\end{align*} \]

Array subscript pairs are affine functions of loop variables

\[ \leq \]

Dependence triples can be formulated as a set of ILP problems

\[ \leq \]

Array subscript pairs are affine functions of loop variables
\[
\begin{align*}
\forall f & = 1 + f \\
\forall I & = 1 - \frac{f}{\text{(exponentiation)}} \\
\text{IF} & \geq \frac{f}{\text{(exponentiation)}} \\
\text{IF} & \geq 1 \\
\forall I & \geq \frac{f}{\text{(exponentiation)}} \\
00 \leq \text{IF} & \leq 1 \\
00 \leq \forall I & \leq 1 \\
\end{align*}
\]

\[
\cdots \text{X(I', I') = X(I-I', I'+1)} \\
\text{FOR I = 1, 100} \\
\text{FOR } I = 1, 100
\]

When above affine loop bounds

- \forall f = 1 + f
- \forall I = 1 - \frac{f}{\text{(exponentiation)}}
- \text{IF} \geq \frac{f}{\text{(exponentiation)}}
- \text{IF} \geq 1
- \forall I \geq \frac{f}{\text{(exponentiation)}}
- 00 \leq \text{IF} \leq 1
- 00 \leq \forall I \leq 1

... \text{X(I', I') = X(I-I', I'+1)}
\text{FOR } I = 1, 100
\text{FOR } I = 1, 100

\[
\begin{align*}
\forall f & = 1 + f \\
\forall I & = 1 - \frac{f}{\text{(exponentiation)}} \\
\text{IF} & \geq \frac{f}{\text{(exponentiation)}} \\
\text{IF} & \geq 1 \\
\forall I & \geq \frac{f}{\text{(exponentiation)}} \\
00 \leq \text{IF} & \leq 1 \\
00 \leq \forall I & \leq 1 \\
\end{align*}
\]

\[
\text{IF} \text{ formulation for Nested Loops}
\]

\[
\begin{align*}
\forall f & = 1 + f \\
\forall I & = 1 - \frac{f}{\text{(exponentiation)}} \\
\text{IF} & \geq \frac{f}{\text{(exponentiation)}} \\
\text{IF} & \geq 1 \\
\forall I & \geq \frac{f}{\text{(exponentiation)}} \\
00 \leq \text{IF} & \leq 1 \\
00 \leq \forall I & \leq 1 \\
\end{align*}
\]

\[
\text{Dependence exists if either system has a solution:}
\]

\[
\begin{align*}
\forall f & = 1 + f \\
\forall I & = 1 - \frac{f}{\text{(exponentiation)}} \\
\text{IF} & \geq \frac{f}{\text{(exponentiation)}} \\
\text{IF} & \geq 1 \\
\forall I & \geq \frac{f}{\text{(exponentiation)}} \\
00 \leq \text{IF} & \leq 1 \\
00 \leq \forall I & \leq 1 \\
\end{align*}
\]

\[\forall f = \frac{f}{\text{(exponentiation)}} \text{ is equivalent to} \]

\[\forall f = \frac{f}{\text{(exponentiation)}} \text{ is equivalent to} \]
How do we solve this decision problem?

Is there an integer solution to system $Ax = b$ of the form

Place of a partially ordered loop can be framed as LTP problem

Problem of determining if a dependence exists between two

Summary

handled separately.

Bounds should not be converted blindly into inequalities but

Anythings you can do to reduce the number of inequalities is good.

exponential in the number of inequalities.

More modern techniques exist, but all known solutions require time

Initialization: Gaussian elimination for inequalities.

Other solution techniques: Fourier-Motzkin elimination

Is there an integer solution to system $Ax \leq b$?

..
Presentation sequence:
- one equation, several variables
- several equations, several variables
- equations & inequalities

Diophantine equations:
use integer Gaussian elimination
Solve equalities first then use Fourier-Motzkin elimination

One equation, many variables:
\[ a_1 x_1 + a_2 x_2 + \cdots + a_n x_n = c \]

Examples:
1. \[ 2x + 3y = 5 \]
2. \[ 2x + 3y + 5z = 5 \]
3. \[ 3x + 4y = 3 \]
4. \[ 2x + 3y = 5 \]

The integer solution will divide \( c \).

Summary:
- Does \( \text{gcd}(a_1, a_2, \ldots, a_n) \) divide \( c \)?
- Does this have integer solutions?

Theorem: The Diophantine equation \( a_1 x_1 + a_2 x_2 + \cdots + a_n x_n = c \) has integer solutions if and only if \( \text{gcd}(a_1, a_2, \ldots, a_n) \) divides \( c \).

Proof:
1. WLOG, assume all coefficients are positive.
2. Induction is on the minimal coefficient number of variables.
3. Base case:
   - If \( a_1 = 1 \), then the equation is \( x = c \) which clearly has integer solutions.
   - If \( \text{gcd}(a_1, a_2, \ldots, a_n) = 1 \), then by the Chinese Remainder Theorem, there exist integers \( b_1, b_2, \ldots, b_n \) such that \( a_1 b_1 \equiv 1 \mod a_2, a_1 b_2 \equiv 1 \mod a_3, \ldots, a_1 b_n \equiv 1 \mod a_1 \).
4. Inductive case:
   - Let \( t = x_1 + \left\lfloor \frac{a_2}{a_1} \right\rfloor x_2 + \cdots + \left\lfloor \frac{a_n}{a_1} \right\rfloor x_n \).
   - Then the equation can be rewritten as \( a_1 t + (a_2 \mod a_1) x_2 + \cdots + (a_n \mod a_1) x_n = c \).
   - By the inductive hypothesis, this equation has integer solutions.
   - Therefore, the original equation also has integer solutions.

Intuition: Think of underdetermined systems of equations over reals.

Caution: Think of underdetermined systems of equations over reals.

\[ \begin{align*}
6y & \Rightarrow y \\
2x + y & \Rightarrow x \\
3z + 5y & \Rightarrow z \text{ or } y \\
2x + 3y + 5z & \Rightarrow 3z + 5y \\
2x + 3y & \Rightarrow 3z + 5y \\
\end{align*} \]

The integer solution will divide \( c \).

Summary:
- One equation, many variables:
It is useful to consider solution process in matrix-theoretic terms.

\[ (3 \ 5 \ 8)(x \ y \ z) = \begin{bmatrix} 6 \\ T \end{bmatrix} \]

\[ T(a \ b) = 8 \]

Solution is \( a = 4, b = t \)

Looks lower triangular, right?

We can write single equation as

It is hard to read off solution from this, but for special matrices, it is easy.

Key concept: column echelon form - "lower triangular form for underdetermined systems"

For a matrix with a single row, column echelon form is

\[ (x \ 0 0 \ldots 0) \]

Substitution: \( t = x + y + 2z \)

New equation:

\[ 3t + 2y + 2z = 6 \]

Substitution: \( u = y + z + t \)

New equation:

\[ 2u + t = 6 \]

Solution:

\[ u = p_1 \]

\[ t = 6 - 2p_1 \]

Backsubstitution:

\[ y = p_2 \]

\[ t = 6 - 2p_1 \]

\[ z = (3p_1 - p_2 - 6) \]

Backsubstitution:

\[ x = (18 - 8p_1 + p_2) \]

Substitution:

\[ (3 \ 2 \ 2) \]

\[ (3 \ 5 \ 8) \]

\[ (1 \ -2 \ 0) \]

\[ (1 \ 0 \ 0) \]

\[ (0 \ 1 \ 0) \]

\[ (0 \ 0 \ 1) \]

Solution:

\[ (6 \ a \ b) \]

Product of matrices =

\[ 2 \ -5 \ -1 \]

\[ -1 \ 3 \ -1 \]

\[ 0 \ 0 \ 1 \]

Solution to original system:

\[ 12 - 5a - b \]

\[ -6 + 3a - b \]

\[ b \]

We can write single equation as

Il est possible to consider solution process in matrix-theoretic terms.

\( \text{INTEGER GAUSSIAN ELIMINATION} \)

Key idea: use integer Gaussian elimination

Systems of Diophantine Equations:

Key idea: use integer Gaussian elimination

Explain:

Almost always more unknowns than equations

For us, column operations are more important because we - the column operations to get matrix into triangular form

\( \text{INTEGER GAUSSIAN ELIMINATION} \)

\( \text{INTEGER GAUSSIAN ELIMINATION} \)

\( \text{INTEGER GAUSSIAN ELIMINATION} \)

\( \text{INTEGER GAUSSIAN ELIMINATION} \)

\( \text{INTEGER GAUSSIAN ELIMINATION} \)

\( \text{INTEGER GAUSSIAN ELIMINATION} \)

\( \text{INTEGER GAUSSIAN ELIMINATION} \)

\( \text{INTEGER GAUSSIAN ELIMINATION} \)

\( \text{INTEGER GAUSSIAN ELIMINATION} \)

\( \text{INTEGER GAUSSIAN ELIMINATION} \)

\( \text{INTEGER GAUSSIAN ELIMINATION} \)

\( \text{INTEGER GAUSSIAN ELIMINATION} \)

\( \text{INTEGER GAUSSIAN ELIMINATION} \)

\( \text{INTEGER GAUSSIAN ELIMINATION} \)

\( \text{INTEGER GAUSSIAN ELIMINATION} \)

\( \text{INTEGER GAUSSIAN ELIMINATION} \)

\( \text{INTEGER GAUSSIAN ELIMINATION} \)

\( \text{INTEGER GAUSSIAN ELIMINATION} \)

\( \text{INTEGER GAUSSIAN ELIMINATION} \)

\( \text{INTEGER GAUSSIAN ELIMINATION} \)

\( \text{INTEGER GAUSSIAN ELIMINATION} \)

\( \text{INTEGER GAUSSIAN ELIMINATION} \)

\( \text{INTEGER GAUSSIAN ELIMINATION} \)

\( \text{INTEGER GAUSSIAN ELIMINATION} \)

\( \text{INTEGER GAUSSIAN ELIMINATION} \)

\( \text{INTEGER GAUSSIAN ELIMINATION} \)

\( \text{INTEGER GAUSSIAN ELIMINATION} \)

\( \text{INTEGER GAUSSIAN ELIMINATION} \)

\( \text{INTEGER GAUSSIAN ELIMINATION} \)

\( \text{INTEGER GAUSSIAN ELIMINATION} \)

\( \text{INTEGER GAUSSIAN ELIMINATION} \)

\( \text{INTEGER GAUSSIAN ELIMINATION} \)

\( \text{INTEGER GAUSSIAN ELIMINATION} \)

\( \text{INTEGER GAUSSIAN ELIMINATION} \)

\( \text{INTEGER GAUSSIAN ELIMINATION} \)

\( \text{INTEGER GAUSSIAN ELIMINATION} \)

\( \text{INTEGER GAUSSIAN ELIMINATION} \)

\( \text{INTEGER GAUSSIAN ELIMINATION} \)

\( \text{INTEGER GAUSSIAN ELIMINATION} \)

\( \text{INTEGER GAUSSIAN ELIMINATION} \)

\( \text{INTEGER GAUSSIAN ELIMINATION} \)

\( \text{INTEGER GAUSSIAN ELIMINATION} \)

\( \text{INTEGER GAUSSIAN ELIMINATION} \)

\( \text{INTEGER GAUSSIAN ELIMINATION} \)

\( \text{INTEGER GAUSSIAN ELIMINATION} \)

\( \text{INTEGER GAUSSIAN ELIMINATION} \)

\( \text{INTEGER GAUSSIAN ELIMINATION} \)

\( \text{INTEGER GAUSSIAN ELIMINATION} \)

\( \text{INTEGER GAUSSIAN ELIMINATION} \)

\( \text{INTEGER GAUSSIAN ELIMINATION} \)

\( \text{INTEGER GAUSSIAN ELIMINATION} \)

\( \text{INTEGER GAUSSIAN ELIMINATION} \)

\( \text{INTEGER GAUSSIAN ELIMINATION} \)

\( \text{INTEGER GAUSSIAN ELIMINATION} \)

\( \text{INTEGER GAUSSIAN ELIMINATION} \)

\( \text{INTEGER GAUSSIAN ELIMINATION} \)

\( \text{INTEGER GAUSSIAN ELIMINATION} \)

\( \text{INTEGER GAUSSIAN ELIMINATION} \)

\( \text{INTEGER GAUSSIAN ELIMINATION} \)

\( \text{INTEGER GAUSSIAN ELIMINATION} \)

\( \text{INTEGER GAUSSIAN ELIMINATION} \)
The product of two unimodular matrices is also unimodular.

A unimodular matrix has integer entries and a determinant of +1 or -1.

The three unimodular column operations preserve integer solutions, as do sequences of these operations:
- adding an integer multiple of one column to another
- negating a column
- interchanging two columns

Facts:
1. The three unimodular column operations
2. Unimodular column operations can be used to reduce the matrix A of a system A x = b into lower triangular form.
3. A unimodular matrix of integer entries has a determinant of +1 or -1.
4. The product of two unimodular matrices is also unimodular.

Exercise:
1. Use only unimodular column operations to solve the system:

\[
\begin{align*}
\begin{bmatrix}
1 & 0 & 1 \\
0 & 1 & 2 \\
1 & 0 & 1
\end{bmatrix} x &=
\begin{bmatrix}
2 \\
3 \\
4
\end{bmatrix}
\end{align*}
\]

Using matrices, construct the reduced matrix A' and find the solution x.

Solution:
\[
\begin{align*}
x &= \begin{bmatrix}
2 \\
3 \\
4
\end{bmatrix}
\end{align*}
\]
Algorithm:
1. Use unimodular column operations to reduce matrix
   \[ A \]
   to column echelon form.

   Let \( r_j \) be the row containing the first non-zero
   entry in column \( j \).

   (i) If \( r_{j+1} > r_j \) and column \( j+1 \) is zero if column \( j \) is not entirely zero.

   (ii) If \( r_{j+1} < r_j \) if column \( j \) is not entirely zero.

   Column echelon form:
   
   Let \( L \) be the lower triangular matrix, so that
   
   \[
   L = \begin{bmatrix}
   x & x & x \\
   0 & x & 0 \\
   0 & 0 & x
   \end{bmatrix}
   
   \]

   is lower triangular but not column echelon.

   Detail: Instead of lower triangular matrix, you should
   work with the column \( l \) equation \( 2 \) variables. The work is
   precisely what is required to produce the column echelon form.

2. If \( Lx' = b \) has integer solutions, so does the original system.

3. If explicit form of solutions is desired, let \( U \) be the product
   of unimodular matrices corresponding to the column operations.

   \[
   x = Ux'
   \]

   is lower triangular but not column echelon.

   Note: Even in regular Gaussian elimination, we want column echelon form rather than
   lower triangular form when we have under-determined systems.