Introduction
to
Loop Transformations
Organization of a Modern Compiler

Source Program → Front-end → Middle1 → Middle2 → Back-end → Assembly Code

Front-end: syntax analysis + type-checking + symbol table

Middle1: loop-level transformations

Middle2: conventional optimizations

Back-end: register allocation, instruction selection

High-level Intermediate Representation (loops, array references are preserved)

Low-level Intermediate Representation (array references converted into low level operations, loops converted to control flow)
Key concepts:

Perfectly-nested loop: Loop nest in which all assignment statements occur in body of innermost loop.

for J = 1, N
    for I = 1, N
        Y(I) = Y(I) + A(I,J)*X(J)

Imperfectly-nested loop: Loop nest in which some assignment statements occur within some but not all loops of loop nest

for k = 1, N
    a(k,k) = sqrt( a(k,k) )
    for i = k+1, N
        a(i,k) = a(i,k) / a(k,k)
    for i = k+1, N
        for j = k+1, i
            a(i,j) = a(i,k) * a(j,k)
Our focus for now: perfectly-nested loops
Goal of lecture:

- We have seen two key transformations of perfectly-nested loops for locality enhancement: permutation and tiling.

- There are other loop transformations that we will discuss in class.

- Powerful way of thinking of perfectly-nested loop execution and transformations:
  
  - loop body instances $\leftrightarrow$ iteration space of loop
  - loop transformation $\leftrightarrow$ change of basis for iteration space
Iteration Space of a Perfectly-nested Loop

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

**Iteration space of loop:** all points in \( n \)-dimensional space corresponding to loop iterations

DO \( I = 1, N \)
DO \( J = 1, M \)
S

Execution order = lexicographic order on iteration space:

\[(1, 1) \preceq (1, 2) \preceq \ldots \preceq (1, M) \preceq (2, 1) \preceq (2, 2) \ldots \preceq (N, M)\]
Loop permutation = linear transformation on iteration space

DO I = 1, N
DO J = 1, M
   S

DO K = 1, M
DO L = 1, N
   S

\[
\begin{bmatrix}
0 & 1 \\
1 & 0
\end{bmatrix}
\begin{bmatrix}
I \\
J
\end{bmatrix}
= 
\begin{bmatrix}
K \\
L
\end{bmatrix}
\]
Locality enhancement:

Loop permutation brings iterations that touch the same cache line "closer" together, so probability of cache hits is increased.
Subtle issue 1: loop permutation may be illegal in some loop nests

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

After loop permutation:

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

Transformed loop will produce different values (A[3,1] for example)
=> permutation is illegal for this loop.

Question: How do we determine when loop permutation is legal?
Subtle issue 2: generating code for transformed loop nest may be non-trivial!

Example: triangular loop bounds (triangular solve/Cholesky)

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

Here, inner loop bounds are functions of outer loop indices!
Just exchanging the two loops will not generate correct bounds.
DO  I = 1, N
DO  J = 1, I
  S

DO  K = 1, N
DO  L = I, N
  S

Question: How do we generate loop bounds for transformed loop nest?
General theory of loop transformations should tell us

- which transformations are legal,
- what the best sequence of transformations should be for a given target architecture, and
- what the transformed code should be.

Desirable: quantitative estimates of performance improvement
ILP Formulation
of
Loop Transformations
Goal:

1. formulate correctness of permutation as integer linear programming (ILP) problem
2. formulate code generation problem as ILP
Two problems:

Given a system of linear inequalities $\mathbf{A} \mathbf{x} \leq \mathbf{b}$

where $\mathbf{A}$ is a $m \times n$ matrix of integers,
$b$ is an $m$ vector of integers,
$x$ is an $n$ vector of unknowns,

(i) Are there integer solutions?
(ii) Enumerate all integer solutions.

Most problems regarding correctness of transformations and code generation can be reduced to these problems.
Intuition about systems of linear inequalities:

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

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

Region described by inequality is convex
(if two points are in region, all points in between them are in region)
Intuition about systems of linear inequalities:

Conjunction of inequalities = intersection of half-spaces
=> some convex region

Region described by inequalities is a convex polyhedron
(if two points are in region, all points in between them are in region)
Let us formulate correctness of loop permutation as ILP problem.

**Intuition:** If all iterations of a loop nest are independent, then permutation is certainly legal.

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

What does independent mean?

Let us look at dependences.
Input dependence is not usually important for most applications.
**Conservative Approximation:**

- Real programs: imprecise information => need for safe approximation
  
  ‘When you are not sure whether a dependence exists, you must assume it does.’

**Example:**

```
procedure f (X,i,j)
begin
  X(i) = 10;
  X(j) = 5;
end
```

**Question:** Is there an output dependence from the first assignment to the second?

**Answer:** If \((i = j)\), there is a dependence; otherwise, not.

=> Unless we know from interprocedural analysis that the parameters \(i\) and \(j\) are always distinct, we must play it safe and insert the dependence.

**Key notion:** Aliasing: two program names may refer to the same location (like \(X(i)\) and \(X(j)\))

May-dependence vs must-dependence: More precise analysis may eliminate may-dependences
**Loop level Analysis:** granularity is a loop iteration

```plaintext
DO I = 1, 100
  DO J = 1, 100
    S
  ENDDO
ENDDO
```

*Dynamic instance of a statement:*
Execution of a statement for given loop index values

*Dependence between iterations:*

Iteration \((I_1,J_1)\) is said to be dependent on iteration \((I_2,J_2)\) if a dynamic instance \((I_1,J_1)\) of a statement in loop body is dependent on a dynamic instance \((I_2,J_2)\) of a statement in the loop body.

*How do we compute dependences between iterations of a loop nest?*
Dependences in loops

FOR 10 I = 1, N
    X(f(I)) = ...
10      = ...X(g(I))...

- Conditions for flow dependence from iteration $I_w$ to $I_r$:
  - $1 \leq I_w \leq I_r \leq N$ (write before read)
  - $f(I_w) = g(I_r)$ (same array location)

- Conditions for anti-dependence from iteration $I_g$ to $I_o$:
  - $1 \leq I_g < I_o \leq N$ (read before write)
  - $f(I_o) = g(I_g)$ (same array location)

- Conditions for output dependence from iteration $I_{w1}$ to $I_{w2}$:
  - $1 \leq I_{w1} < I_{w2} \leq N$ (write in program order)
  - $f(I_{w1}) = f(I_{w2})$ (same array location)
Dependences in nested loops

FOR 10 I = 1, 100
   FOR 10 J = 1, 200
      X(f(I,J),g(I,J)) = ...
   10         = ...X(h(I,J),k(I,J))..

Conditions for flow dependence from iteration \((I_1, J_1)\) to \((I_2, J_2)\):
Recall: \(\leq\) is the lexicographic order on iterations of nested loops.

\[
\begin{align*}
1 & \leq I_w \leq 100 \\
1 & \leq J_w \leq 200 \\
1 & \leq I_r \leq 100 \\
1 & \leq J_r \leq 200 \\
(I_1, J_1) & \leq (I_2, J_2) \\
f(I_1, J_1) & = h(I_2, J_2) \\
g(I_1, J_1) & = k(I_2, J_2)
\end{align*}
\]
Anti and output dependences can be defined analogously.
Array subscripts are affine functions of loop variables


\Rightarrow

dependence testing can be formulated as a set of ILP problems
**ILP Formulation**

FOR \( I = 1, 100 \)

\[ X(2I) = \ldots X(2I+1) \ldots \]

Is there a flow dependence between different iterations?

\[
\begin{align*}
1 & \leq Iw < Ir \leq 100 \\
2Iw & = 2Ir + 1
\end{align*}
\]

which can be written as

\[
\begin{align*}
1 & \leq Iw \\
Iw & \leq Ir - 1 \\
Ir & \leq 100 \\
2Iw & \leq 2Ir + 1 \\
2Ir + 1 & \leq 2Iw
\end{align*}
\]
The system

\[
\begin{align*}
1 & \leq I_w \\
I_w & \leq Ir - 1 \\
Ir & \leq 100 \\
2I_w & \leq 2Ir + 1 \\
2Ir + 1 & \leq 2I_w
\end{align*}
\]

can be expressed in the form \( Ax \leq b \) as follows:

\[
\begin{pmatrix}
-1 & 0 \\
1 & -1 \\
0 & 1 \\
2 & -2 \\
-2 & 2
\end{pmatrix}
\begin{bmatrix}
I_w \\
Ir
\end{bmatrix}
\leq
\begin{bmatrix}
-1 \\
-1 \\
100 \\
1 \\
-1
\end{bmatrix}
\]
**ILP Formulation for Nested Loops**

\[
\text{FOR } I = 1, 100 \\
\text{FOR } J = 1, 100 \\
\quad X(I,J) = \ldots X(I-1,J+1) \ldots
\]

Is there a flow dependence between different iterations?

\[
1 \leq Iw \leq 100 \\
1 \leq Ir \leq 100 \\
1 \leq Jw \leq 100 \\
1 \leq Jr \leq 100
\]

\[
(Iw, Jw) \prec (Ir, Jr) \text{(lexicographic order)}
\]

\[
Ir - 1 = Iw \\
Jr + 1 = Jw
\]

Convert lexicographic order \( \prec \) into integer equalities/inequalities.
\[(Iw, Jw) \prec (Ir, Jr)\] is equivalent to \[Iw < Ir \text{ OR } ((Iw = Ir) \text{ AND } (Jw < Jr))\]

We end up with two systems of inequalities:

\begin{align*}
1 & \leq Iw \leq 100 \\
1 & \leq Ir \leq 100 \\
1 & \leq Jw \leq 100 \\
1 & \leq Jr \leq 100 \\
Iw & < Ir \\
Ir - 1 & = Iw \\
Jr + 1 & = Jw
\end{align*}

\[\text{OR}\]

\begin{align*}
1 & \leq Iw \leq 100 \\
1 & \leq Ir \leq 100 \\
1 & \leq Jw \leq 100 \\
1 & \leq Jr \leq 100 \\
Iw & = Ir \\
Jw & < Jr \\
Ir - 1 & = Iw \\
Jr + 1 & = Jw
\end{align*}

Dependence exists if either system has a solution.
What about affine loop bounds?

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

\[
1 \leq I_w \leq 100 \\
1 \leq I_r \leq 100 \\
1 \leq J_w \leq I_w \\
1 \leq J_r \leq I_r \\
(I_w, J_w) < (I_r, J_r)\text{(lexicographic order)} \\
I_r - 1 = I_w \\
J_r + 1 = J_w 
\]
We can actually handle fairly complicated bounds involving min’s and max’s.

FOR I = 1, 100
    FOR J = max(F1(I),F2(I)) , min(G1(I),G2(I))
    X(I,J) = ..X(I-1,J+1)...

....

F1(Ir) ≤ Jr
F2(Ir) ≤ Jr
Jr ≤ G1(Ir)
Jr ≤ G2(Ir)
....

Caveat: F1, F2 etc. must be affine functions.
For a given $I$, the $J$ co-ordinate of a point in the iteration space of the loop nest satisfies $\max(L_1(I), L_2(I)) \leq J \leq \min(U_1(I), U_2(I))$.

Min’s and max’s in loop bounds may seem weird, but actually they describe general polyhedral iteration spaces!
More important case in practice: variables in upper/lower bounds

FOR \( I = 1, N \)
  FOR \( J = 1, N-1 \)
    ....

Solution: Treat \( N \) as though it was an unknown in system

\[
1 \leq Iw \leq N \\
1 \leq Jw \leq N - 1 \\
....
\]

This is equivalent to seeing if there is a solution for any value of \( N \).

Note: if we have more information about the range of \( N \), we can easily add it as additional inequalities.
Summary

Problem of determining if a dependence exists between two iterations of a perfectly nested loop can be framed as ILP problem of the form

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

How do we solve this decision problem?
Is there an integer solution to system $Ax \leq b$?

Oldest solution technique: Fourier-Motzkin elimination

Intuition: ”Gaussian elimination for inequalities”

More modern techniques exist, but all known solutions require time exponential in the number of inequalities

=>

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

=>

Equalities should not be converted blindly into inequalities but handled separately.
Presentation sequence:

- one equation, several variables
  \[ 2x + 3y = 5 \]

- several equations, several variables
  \[ 2x + 3y + 5z = 5 \]
  \[ 3x + 4y = 3 \]

- equations & inequalities
  \[ 2x + 3y = 5 \]
  \[ x \leq 5 \]
  \[ y \leq -9 \]

Diophantine equations: use integer Gaussian elimination

Solve equalities first then use Fourier-Motzkin elimination
One equation, many variables:

Thm: The linear Diophantine equation  \( a_1 x_1 + a_2 x_2 + \ldots + a_n x_n = c \) has integer solutions iff \( \gcd(a_1,a_2,\ldots,a_n) \) divides \( c \).

Examples:

(1)  \( 2x = 3 \)  No solutions
(2)  \( 2x = 6 \)  One solution:  \( x = 3 \)
(3)  \( 2x + y = 3 \)
   \( \gcd(2,1) = 1 \) which divides 3.
   Solutions:  \( x = t, y = (3 - 2t) \)
(4)  \( 2x + 3y = 3 \)
   \( \gcd(2,3) = 1 \) which divides 3.
   Let  \( z = x + \text{floor}(3/2) \)  \( y = x + y \)
   Rewrite equation as  \( 2z + y = 3 \)
   Solutions:  \( z = t \)  \( x = (3t - 3) \)
   \( y = (3 - 2t) \)

Intuition: Think of underdetermined systems of eqns over reals.
Caution: Integer constraint \( \Rightarrow \) Diophantine system may have no solns
Thm: The linear Diophantine equation \( a_1 x_1 + a_2 x_2 + \ldots + a_n x_n = c \) has integer solutions iff \( \gcd(a_1, a_2, \ldots, a_n) \) divides \( c \).

Proof: WLOG, assume that all coefficients \( a_1, a_2, \ldots, a_n \) are positive.

We prove only the IF case by induction, the proof in the other direction is trivial.

Induction is on \( \min(\text{smallest coefficient, number of variables}) \).

Base case:

If (\( \# \text{ of variables} = 1 \)) , then equation is \( a_1 x_1 = c \) which has integer solutions if \( a_1 \) divides \( c \).

If (\( \text{smallest coefficient} = 1 \)), then \( \gcd(a_1, a_2, \ldots, a_n) = 1 \) which divides \( c \).

Wlog, assume that \( a_1 = 1 \), and observe that the equation has solutions of the form \((c - a_2 t_2 - a_3 t_3 - \ldots - a_n t_n, t_2, t_3, \ldots, t_n)\).

Inductive case:

Suppose smallest coefficient is \( a_1 \), and let \( t = x_1 + \lfloor \frac{a_2}{a_1} \rfloor x_2 + \ldots + \lfloor \frac{a_n}{a_1} \rfloor x_n \)

In terms of this variable, the equation can be rewritten as

\[
(a_1) t + (a_2 \mod a_1) x_2 + \ldots + (a_n \mod a_1) x_n = c \quad (1)
\]

where we assume that all terms with zero coefficient have been deleted.

Observe that \((1)\) has integer solutions iff original equation does too.

Now \( \gcd(a, b) = \gcd(a \mod b, b) \Rightarrow \gcd(a_1, a_2, \ldots, a_n) = \gcd(a_1, (a_2 \mod a_1), \ldots, (an \mod a_1)) \)

\( \Rightarrow \gcd(a_1, (a_2 \mod a_1), \ldots, (an \mod a_1)) \) divides \( c \).

If \( a_1 \) is the smallest co-efficient in \((1)\), we are left with 1 variable base case.

Otherwise, the size of the smallest co-efficient has decreased, so we have made progress in the induction.
Summary:

Eqn: \[ a_1 x_1 + a_2 x_2 + \ldots + a_n x_n = c \]

- Does this have integer solutions?

= Does \( \gcd(a_1,a_2,\ldots,a_n) \) divide \( c \)?
It is useful to consider solution process in matrix-theoretic terms.

We can write single equation as

\[(3 \ 5 \ 8)(x \ y \ z)^T = 6\]

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

\[(2 \ 0)(a \ b)^T = 8\]

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

looks lower triangular, right?

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

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

\[(x \ 0 \ 0 \ 0...0)\]
3x + 5y + 8z = 6

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

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

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

\[ 2u + t = 6 \]

Solution:
\( u = p1 \)
\( t = (6 - 2p1) \)

Backsubstitution:
\( y = p2 \)
\( t = (6 - 2p1) \)
\( z = (3p1 - p2 - 6) \)

Backsubstitution:
\( x = (18 - 8p1 + p2) \)
\( y = p2 \)
\( z = (3p1 - p2 - 6) \)

\[
\begin{pmatrix} 3 & 5 & 8 \\ \end{pmatrix}
\]

\[
\begin{pmatrix} 3 & 5 & 8 \\ 1 & -1 & -2 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ \end{pmatrix} \Rightarrow U1
\]

\[
\begin{pmatrix} 3 & 5 & 8 \\ 1 & 0 & 0 \\ -1 & 1 & -1 \\ 0 & 0 & 1 \\ \end{pmatrix} \Rightarrow U2
\]

\[
\begin{pmatrix} 1 & 0 & 0 \\ 1 & -2 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ \end{pmatrix} \Rightarrow U3
\]

Solution: \( (6 \ a \ b) \)

Product of matrices = \[
\begin{pmatrix} 2 & -5 & -1 \\ -1 & 3 & -1 \\ 0 & 0 & 1 \\ \end{pmatrix}
\]

Solution to original system: \[
\begin{pmatrix} 12 - 5a - b \\ -6 + 3a - b \\ b \\ \end{pmatrix}
\]
Systems of Diophantine Equations:

**Key idea:** use integer Gaussian elimination

**Example:**

\[
\begin{align*}
2x + 3y + 4z &= 5 \\
x - y + 2z &= 5
\end{align*}
\]

\[
\begin{bmatrix}
2 & 3 & 4 \\
1 & -1 & 2
\end{bmatrix}
\begin{bmatrix}
x \\
y \\
z
\end{bmatrix} =
\begin{bmatrix}
5 \\
5
\end{bmatrix}
\]

It is not easy to determine if this Diophantine system has solutions.

**Easy special case:** lower triangular matrix

\[
\begin{bmatrix}
1 & 0 & 0 \\
-2 & 5 & 0
\end{bmatrix}
\begin{bmatrix}
x \\
y \\
z
\end{bmatrix} =
\begin{bmatrix}
5 \\
5
\end{bmatrix}
\Rightarrow
\begin{align*}
x &= 5 \\
y &= 3 \\
z &= \text{arbitrary integer}
\end{align*}
\]

**Question:** Can we convert general integer matrix into equivalent lower triangular system?

**INTEGER GAUSSIAN ELIMINATION**
Integer gaussian Elimination

- Use row/column operations to get matrix into triangular form
- For us, column operations are more important because we usually have more unknowns than equations

Overall strategy: Given $Ax = b$

- Find matrices $U_1, U_2, \ldots, U_k$ such that $A*U_1*U_2*\ldots*U_k$ is lower triangular (say $L$)
- Solve $Lx' = b$ (easy)
- Compute $x = (U_1*U_2*\ldots*U_k)x'$

Proof:

$(A*U_1*U_2*\ldots*U_k)x' = b$

$=> A(U_1*U_2*\ldots*U_k)x' = b$

$=> x = (U_1*U_2*\ldots*U_k)x'$
Caution: Not all column operations preserve integer solutions.

\[
\begin{bmatrix}
2 & 3 \\
6 & 7
\end{bmatrix}
\begin{bmatrix}
x \\
y
\end{bmatrix}
= 
\begin{bmatrix}
5 \\
1
\end{bmatrix}
\]

Solution: \( x = -8, y = 7 \)

\[
\begin{bmatrix}
1 & -3 \\
0 & 2
\end{bmatrix}
\]

\[
\begin{bmatrix}
2 & 0 \\
6 & -4
\end{bmatrix}
\begin{bmatrix}
x' \\
y'
\end{bmatrix}
= 
\begin{bmatrix}
5 \\
1
\end{bmatrix}
\]

which has no integer solutions!

Intuition: With some column operations, recovering solution of original system requires solving lower triangular system using rationals.

Question: Can we stay purely in the integer domain?

One solution: Use only unimodular column operations
Unimodular Column Operations:

(a) Interchange two columns

\[
\begin{bmatrix}
2 & 3 \\
6 & 7 \\
\end{bmatrix}
\rightarrow
\begin{bmatrix}
3 & 2 \\
7 & 6 \\
\end{bmatrix}
\]

Let \( x, y \) satisfy first eqn.
Let \( x', y' \) satisfy second eqn.
\( x' = y, \quad y' = x \)

Check

(b) Negate a column

\[
\begin{bmatrix}
2 & 3 \\
6 & 7 \\
\end{bmatrix}
\rightarrow
\begin{bmatrix}
2 & -3 \\
6 & -7 \\
\end{bmatrix}
\]

Check

\( x' = x, \quad y' = -y \)

(c) Add an integer multiple of one column to another

\[
\begin{bmatrix}
2 & 3 \\
6 & 7 \\
\end{bmatrix}
\rightarrow
\begin{bmatrix}
2 & 1 \\
6 & 1 \\
\end{bmatrix}
\]

Check

\( n = -1 \)

\( x = x' + n y' \)
\( y = y' \)
Example:

\[
\begin{bmatrix}
2 & 3 & 4 \\
1 & -1 & 2
\end{bmatrix}
\begin{bmatrix}
x \\
y \\
z
\end{bmatrix}
= 
\begin{bmatrix}
5 \\
5
\end{bmatrix}
\]

\[
\begin{bmatrix}
2 & 3 & 4 \\
1 & -1 & 2
\end{bmatrix}
\Rightarrow
\begin{bmatrix}
2 & 3 & 0 \\
1 & -1 & 0
\end{bmatrix}
\Rightarrow
\begin{bmatrix}
2 & 1 & 0 \\
1 & -2 & 0
\end{bmatrix}
\Rightarrow
\begin{bmatrix}
0 & 1 & 0 \\
5 & -2 & 0
\end{bmatrix}
\Rightarrow
\begin{bmatrix}
1 & 0 & 0 \\
0 & 2 & 0
\end{bmatrix}
\]

\[
\begin{bmatrix}
1 & 0 & -2 \\
0 & 1 & 0 \\
0 & 0 & 1
\end{bmatrix}
\Rightarrow
\begin{bmatrix}
1 & -1 & 0 \\
0 & 1 & 0 \\
0 & 0 & 1
\end{bmatrix}
\Rightarrow
\begin{bmatrix}
1 & 0 & 0 \\
0 & 1 & 0 \\
-2 & 1 & 0 \\
0 & 0 & 1
\end{bmatrix}
\Rightarrow
\begin{bmatrix}
0 & 1 & 0 \\
1 & 0 & 0 \\
0 & 0 & 1
\end{bmatrix}
\]

\[
\begin{bmatrix}
1 & 0 & 0 \\
-2 & 5 & 0
\end{bmatrix}
\begin{bmatrix}
x' \\
y' \\
z'
\end{bmatrix}
= 
\begin{bmatrix}
5 \\
5
\end{bmatrix}
\Rightarrow
x' = 5
\]

\[
\begin{bmatrix}
-1 & 3 & -2 \\
1 & -2 & 0 \\
0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
x \\
y \\
z
\end{bmatrix}
= 
\begin{bmatrix}
-1 \\
5 \\
4-2t
\end{bmatrix}
\Rightarrow
x = 5
\]

\[
\begin{bmatrix}
1 & 0 & 0 \\
0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
z' = t
\end{bmatrix}
\]
Facts:

1. The three unimodular column operations
   - interchanging two columns
   - negating a column
   - adding an integer multiple of one column to another
   on the matrix $A$ of the system $A\mathbf{x} = \mathbf{b}$
   preserve integer solutions, as do sequences of these operations.

2. Unimodular column operations can be used to reduce
   a matrix $A$ into lower triangular form.

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

4. The product of two unimodular matrices is also unimodular.
Algorithm: Given a system of Diophantine equations $Ax = b$

1. Use unimodular column operations to reduce matrix $A$ to lower triangular form $L$.
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. 
   
   $$x = Ux'$$ where $x'$ is the solution of the system $Lx' = b$

Detail: Instead of lower triangular matrix, you should to compute ‘column echelon form’ of matrix.

Column echelon form: Let $r_j$ be the row containing the first non-zero in column $j$.
   (i) $r(j+1) > r_j$ if column $j$ is not entirely zero.
   (ii) column $(j+1)$ is zero if column $j$ is.

```
x  0  0
x  0  0
x  x  x
```

is lower triangular but not column echelon.

Point: writing down the solution for this system requires additional work with the last equation (1 equation, 2 variables). This work is precisely what is required to produce the column echelon form.

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