Introduction to Loop Transformations
Organization of a Modern Compiler

Front-end

Syntax analysis + type-checking + symbol table

Middle 1

Loop-level transformations

Intermediate representation + conventional optimizations

Middle 2

Intermediate representation

Low-level representation (array references converted into low-level operations, loops converted to control flow)

Back-end

Assembly

Instruction selection

Register allocation

Program source
Key concepts:

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

 Perfectly-nested loop: Loop nest in which all assignment statements occur in body of innermost loop.
Our focus for now: perfectly-nested loops
loop transformation ↔ change of basis for iteration space

- loop body instances ↔ iteration space of loop

- transformations:
  - powerful way of thinking of perfectly-nested loop execution and class.
  - There are other loop transformations that we will discuss in this lecture.

- For locality enhancement: permutation and timing.

- We have seen two key transformations of perfectly-nested loops.

Goal of lecture:
\[(M, N) \supset \cdots \supset (2, 2) \supset (1, 2) \supset (I, 1) \supset \cdots \supset (1, 2) \supset (1, 1)\]

Execution order = lexicographic order on iteration space:

\[
\begin{align*}
\text{DO } & \text{ } I = 1, N \\
\text{DO } & \text{ } I = 1, M \\
S &
\end{align*}
\]

corresponding to loop iterations

Iteration space of loop: all points in n-dimensional space.

Integer point in an n-dimensional space.

Each iteration of a loop nest with n loops can be viewed as an

Iteration Space of a Perfectly-nested Loop
Loop permutation = linear transformation on iteration space

\[
\begin{bmatrix}
I \\
K
\end{bmatrix} =
\begin{bmatrix}
I & 0 \\
I & I
\end{bmatrix}
\begin{bmatrix}
I & 1 \\
0 & I
\end{bmatrix}
\]

\[
(S(K'))
\]

DO I = 1, N
DO K = 1, M

\[
(S(I'))
\]

DO J = 1, M
DO I = 1, N
"closer" together, so probability of cache hits is increased.

Loop permutation brings iterations that touch the same cache line

Locality enhancement:
Question: How do we determine when loop permutation is illegal?

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

\[ A[I',J'] = A[I-I',J+1]+1 \]

\[
\begin{align*}
\text{DO } & I = 2, N \\
\text{DO } & J = 1, M
\end{align*}
\]

After loop permutation:

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

Subtle issue: Loop permutation may be illegal in some loop nests
Just exchanging the two loops will not generate correct bounds.

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

\[
S
\]

\[
\text{FOR } j = 1, i - 1
\]

\[
\text{FOR } i = 1, N
\]

Example: triangular loop bounds (triangular solve/Cholesky)

non-trivial

Subtle issue 2: generating code for transformed loop may be
Question: How do we generate loop bounds for transformed loop nest?

\[ \begin{bmatrix} I \\ K \end{bmatrix} = \begin{bmatrix} I & 0 \\ I & 1 \end{bmatrix} \begin{bmatrix} 0 & 1 \\ I & 0 \end{bmatrix} \]
Desirable: quantitative estimates of performance improvement

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

General theory of loop transformations should tell us
2. Formulate code generation problem as ILP programming (ILP) problem
1. Formulate correctness of permutation as integer linear

Goal:
Two problems:

(i) Are there integer solutions?

(ii) Enumerate all integer solutions.

Given a system of linear inequalities $A \cdot x \geq b$:

- $A$ is an $m \times n$ matrix of integers,
- $b$ is an $m$ vector of integers,
- $x$ is an $n$ vector of unknowns.

Most problems regarding correctness of transformations and code generation can be reduced to these problems.
Region described by inequality is convex (if two points are in region, all points in between them are in region).

Region: $3x + 4y = 12$

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

Intuition about systems of linear inequalities:

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

Inequality: half-plane (2D), half-space (>2D)
Intuition about systems of linear inequalities:

Region described by inequalities is a convex polyhedron (if two points are in region, all points in between them are in region)

Conjunction of inequalities = intersection of half-spaces

=> some convex region
Let us look at dependences.

What does independent mean?

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

Permutation is certainly legal.

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

Let us formulate correctness of loop permutation as ITP problem.
Input dependence is not usually important for most applications.

Input dependence: $S_1$ and $S_2$ both read from the same location
  (i) $S_1$ executes before $S_2$
  (ii) $S_1$ executes before $S_2$

Output dependence: $S_1 \rightarrow S_2$
  (i) $S_1$ and $S_2$ write to the same location
  (ii) $S_1$ executes before $S_2$
  Anti-depence: $S_1 \rightarrow S_2$
  (i) $S_1$ executes before $S_2$
  (ii) $S_1$ reads from a location that is overwritten later by $S_2$
  (i) $S_1$ executes before $S_2$
  (ii) $S_1$ writes into a location that is read by $S_2$
  Flow dependence: $S_1 \rightarrow S_2$
  (i) $S_1$ executes before $S_2$ in program order

$\ell =: \lambda$ 
$\xi =: x$
$I + x =: \lambda$
$\zeta =: x$
Real programs: imprecise information => need for safe approximation

Procedure $f(X,i,j)$

```plaintext
begin
  $X(i) = 10$
  $X(j) = 5$
end
```

Answer: If $i = j$, there is a dependence; otherwise, not.

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

Example:

```
Conservative Approximation:

When you are not sure whether a dependence exists, you must assume it does.

Real programs: imprecise information => need for safe approximation
```

Key notion: Aliasing: two program names may refer to the same location (like $X(1)$ and $X(i)$)

We must play it safe and insert the dependence.

$\Rightarrow$ Unless we know from interprocedural analysis that the parameters $i$ and $j$ are always distinct.

Question: Is there an output dependence from the first assignment to the second?
How do we compute dependences between iterations of a loop nest?

**in the loop body:**

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 loop body.

**Dependence between iterations:**

Execution of a statement for given loop index values is a dynamic instance of a statement.

**Dynamic instance of a statement:**

Granularity is a loop iteration.

```
DO I = 1, 100
  DO J = 1, 100
    S
  ENDDO
ENDDO
```
\[(\text{same array location}) (z^mI)f = (I^nI)f \]
\[N \geq z^mI > I^nI \geq I\]

Conditions for output dependence from iteration \(z^mI\) to \(I^nI\):

\[(\text{same array location}) (\delta I)\theta = (\circ I)f \]
\[N \geq \circ I > \delta I \geq I\]

Conditions for anti-dependence from iteration \(\circ I\) to \(\delta I\):

\[(\text{same array location}) (\delta I)\theta = (mI)f \]
\[N \geq \delta I > mI \geq I\]

Conditions for how dependence from iteration \(\circ I\) to \(mI\):

\[\text{FOR } 10, I = 1, N \]

Dependences in loops
Recall: is the lexicographic order on iterations of nested loops.

Conditions for how dependence from iteration to iteration:

\[(v, v_I) g = (v, v_I) b\]

\[(v, v_I) h = (v, v_I) f\]

\[(v, v_I) \supseteq (v, v_I)\]

\[000 \succ ^a f \succ 1\]

\[000 \succ ^a I \succ 1\]

\[000 \succ ^m f \succ 1\]

\[000 \succ ^m I \succ 1\]

\[\cdots ((v, v_I) f) (v, v_I) \cdots = 10\]

\[\cdots = ((v, v_I) g) (v, v_I) \cdots\]

For \(10 \leq j \leq 1, 200\)

For \(10 \leq i \leq 1, 100\)
Anti and output dependencies can be defined analogously.
dependence testing can be formulated as a set of ILP problems

Array subscripts are affine functions of loop variables
\[ \begin{align*}
2I^r & \geq 2I^r + 1 \\
2I^r + 1 & \geq 2I \\
0 & \geq I \\
I - I^r & \geq mI \\
mI & \geq I
\end{align*} \]

which can be written as

\[ I + I^r = mI \]

\[ 0 \geq I^r > mI \geq I \]

Is there a flaw dependence between different iterations?

\[ \cdots \cdot \cdot \cdot x(2I+1) = x(2I) \]

For \( I = 1 \), 100

ILP Formulation
The system can be expressed in the form \( Ax = q \) as follows:

\[
\begin{pmatrix}
1 & -1 & 1 \\
1 & 1 & 0 \\
0 & 1 & 1 \\
1 & 1 & 0
\end{pmatrix}
\geq
\begin{pmatrix}
2 \\
2 \\
2 \\
2
\end{pmatrix}
\begin{pmatrix}
\mu I \\
\mu I \\
\mu I \\
\mu I
\end{pmatrix}
\begin{pmatrix}
1 \\
0 \\
1 \\
1
\end{pmatrix}
\]
Convert lexicographic order into integer equalities/inequalities.

\[ m_f = 1 + f \]
\[ m_I = 1 - I \]

(lexicographic order)

\[(m_f, m_I) \succ (m_I, m_f) \]

Is there a flow dependence between different iterations?

\[
\begin{array}{c}
\dot{X}(I, J) = \cdot X(I-1, J+1) \\
00 \succ m_f \succ 1 \\
00 \succ m_I \succ 1 \\
00 \succ m_I \succ 1 \\
00 \succ m_f \succ 1
\end{array}
\]

IP Formulation for Nested Loops
Dependence exists if either system has a solution.

\[ m_f = 1 + \mu_f \]
\[ m_I = 1 - \mu_I \]
\[ \mu_f > m_f \]
\[ \mu_I = m_I \]
\[ 001 \gtrless \mu_f \gtrless 1 \]
\[ 001 \gtrless m_f \gtrless 1 \]
\[ 001 \gtrless \mu_I \gtrless 1 \]
\[ 001 \gtrless m_I \gtrless 1 \]

We end up with two systems of inequalities:

\[ ((\mu_f > m_f) \text{ AND } (\mu_I = m_I)) \text{ OR } \mu_I > m_I \]

is equivalent to

\[ (m_f, \mu_f) \cong (m_I, \mu_I) \]
\[
\begin{align*}
\mathcal{M}_f &= I + \mathcal{M}_f \\
\mathcal{M}_I &= I - \mathcal{M}_I \\
(\mathcal{M}_f, \mathcal{M}_I) &\supseteq (\mathcal{M}_f, \mathcal{M}_I) \\
\mathcal{M}_I &\supseteq \mathcal{M}_f & \supset I \\
\mathcal{M}_I &\supseteq \mathcal{M}_f & \supset I \\
00 \supseteq \mathcal{M}_I &\supset I \\
00 \supseteq \mathcal{M}_I &\supset I \\
\end{align*}
\]

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

What about affine loop bounds?
Caveat: \( P_1, P_2 \) etc. must be affine functions.

\[
\begin{align*}
\gamma & \geq \gamma \\
\gamma & \geq \gamma \\
\gamma & \geq (P_2)_I \\
\gamma & \geq (P_1)_I \\
\end{align*}
\]

\[
X(I', j) = X(I-I', j+1) \\
\approx \max(P_1(I), P_2(I), \min(G_1(I), G_2(I)))
\]

For \( j = \max(P_1(I), P_2(I), \min(G_1(I), G_2(I))) \) and \( I = 1, 100 \),

We can actually handle fairly complicated bounds involving mins and maxs.
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!
add it as additional inequalities.

Note: if we have more information about the range of $N$, we can easily

This is equivalent to seeing if there is a solution for any value of $N$.

\[ I - N \geq m_1 \geq I \]
\[ N \geq m_1 \geq I \]

Solution: Treat $N$ as though it was an unknown in system

\[ \text{FOR } I = 1, N-1 \]
\[ \text{FOR } I = 1, N \]

More important case in practice: variables in upper/lower bounds
How do we solve this decision problem?

Is there an integer solution to system $Ax \geq q$?

Problem of determining if a dependence exists between two

Summary
Inequalities should not be converted blindly into inequalities but handled separately.

Anytime 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.

Intuition: "Cayleyan elimination for inequalities"

Older solution technique: Fourier-Motzkin elimination

Is there an integer solution to system $Ax \preceq b$?
Presentation sequence:

- one equation, several variables

\[ 2x + 3y = 5 \]

- several equations, several variables

\[ 2x + 3y + 5z = 5 \]
\[ 3x + 4y = 3 \]
\[ 2x + 3y = 5 \]

\[ x \leq 5 \]
\[ y \leq -9 \]

- equations & inequalities

Diophantine equations:

Solve equalities first then use Fourier-Motzkin elimination.

Use integer Gaussian elimination.

Equations & inequalities:

\[ x \rightarrow 5 \]
\[ y \rightarrow -9 \]
\[ 2x + 3y = 5 \]

- several equations, several variables

\[ 3x + 4y + 5z = 5 \]

\[ 2x + 3y = 5 \]

\[ 2x + 3y = 5 \]

- one equation, several variables

Presentation sequence:
One equation, many variables:
$$a_1 x_1 + a_2 x_2 + \ldots + a_n x_n = c$$

Examples:
$$\text{GCD}(2, 1) = 1 \text{ which divides 3.}$$
$$\text{Solutions: } x = t, \quad y = (3 - 2t)$$

Let
$$z = x + \text{floor}(y/3/2) \quad \text{ and } \quad y + x = y$$

Rewrite equation as
$$2z + y = 3$$

Let
$$z = t \quad \text{ which divides 3.}$$

$$\text{Solutions: } z = t, \quad y = (3 - 2t)$$

$$\text{GCD}(2, 3) = 1 \text{ which divides 3.}$$

$$\text{Solutions: } z = t \quad \text{ which divides 3.}$$

$$\text{GCD}(2, 3) = 1 \text{ which divides 3.}$$

$$\text{Solutions: } z = t \quad \text{ which divides 3.}$$

$$\text{GCD}(2, 3) = 1 \text{ which divides 3.}$$

$$\text{Solutions: } z = t \quad \text{ which divides 3.}$$

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.

Intuition: Think of underdetermined systems of eqns over reals.

Caution: Integer constraint => Diophantine system may have no solns.

Examples:

- (1) $2x = 3$  No solutions
- (2) $2x = 6$  One solution: $x = 3$
- (3) $2x + y = 3$
- (4) $2x + 3y = 3$

Intuition: Think of underdetermined systems of eqns over reals.

Caution: Integer constraint => Diophantine system may have no solns.

Examples:

- (1) $2x = 3$  No solutions
- (2) $2x = 6$  One solution: $x = 3$
- (3) $2x + y = 3$
- (4) $2x + 3y = 3$
Theorem: 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:

Without loss of generality, 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 \( \min(\text{smallest coefficient}, \text{number of variables}) \).

Base case:

If the form \( c = a_2 x_2 - a_3 x_3 - \ldots - a_n x_n \), and observe that the equation has solutions if \( \gcd(a_1, a_2, \ldots, a_n) = 1 \), which divides \( c \).

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

Inductive case:

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

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

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

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

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

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

Now \( \gcd(a, b) = \gcd(a \mod b, b) \) \Rightarrow \gcd(a_1, \ldots, a_n) = \gcd(a_1, (a_2 \mod a_1), \ldots, (a_n \mod a_1)) \) divides \( c \).

If \( a_1 \) is the smallest coefficient in (1), we are left with 1 variable base case. Otherwise, the size of the smallest coefficient has decreased, so we have made progress in the induction.
Eqn: \[ a_1 x_1 + a_2 x_2 + \ldots + a_n x_n = c \]

Summary:
- Does \( \gcd(a_1, a_2, \ldots, a_n) \) divide \( c \)?
- Does this have integer solutions?
It is useful to consider solution process in matrix-theoretic terms.

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

It is lower triangular, right?

\[
\begin{pmatrix}
2 & 0 \\
\end{pmatrix}
\begin{pmatrix}
a \\
b \\
\end{pmatrix}
\] = 8

Solution is \(a = 4\), \(b = 1\).

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

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

\[
\begin{pmatrix}
x \\
0 \\
0 \\
\end{pmatrix}
\]

We can write single equation as:

It is easy.

If is hard to read off solution from this, but for special matrices, it is useful to consider solution process in matrix-theoretic terms.
Solution to original system:

\[ (9-6\xi-2d-1) = z \]
\[ (1d-2\xi-6) = x \]
\[ \xi = \delta \]

Backsubstitution:

\[ (6-1p\xi-2b-6) = (9-6\xi-2d-1) \]
\[ (1d-2\xi-6) = (1d-2\xi-6) \]
\[ \xi = \delta \]

Solution:

\[ 2u = 6 \]

New equation:

\[ 9 = 2z + \lambda + x \]

Substitution: \( z + \lambda + x = 1 \)

\[ 9 = 2z + \lambda + x = 1 \]

Solution:

\[ 6 = 2z + \lambda + x \]

\[ 6 = 2z + \lambda + x \]

Product of matrices:

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

Solution:

\[ (6, a, b) \]

New equation:

\[ z + \lambda + x = 1 \]

Substitution: \( 6 + \lambda + x = 1 \)

\[ 6 + \lambda + x = 1 \]

Backsubstitution:

\[ \lambda = \delta \]

Solution:

\[ x = (9-6\xi-2d-1) \]
\[ (1d-2\xi-6) = (9-6\xi-2d-1) \]

Solution:

\[ (6, a, b) \]

New equation:

\[ 9 = 2z + \lambda + x \]

Substitution: \( z + \lambda + x = 1 \)

\[ 9 = 2z + \lambda + x = 1 \]

Solution:

\[ (6, a, b) \]

New equation:

\[ 9 = 2z + \lambda + x \]

Substitution: \( z + \lambda + x = 1 \)

\[ 9 = 2z + \lambda + x = 1 \]
**Key Idea:** Use Integer Gaussian Elimination

**Systems of Diophantine Equations:**

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

**Example:**

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

**Example:**

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

**Example:**

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

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

**Example:**

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

**Answer:**

- \( x = 5 \)
- \( y = 3 \)
- \( z = \) arbitrary integer

**Example:**

\[ \begin{align*}
3 &= y \\
5 &= x \\
z &= \text{arbitrary integer}
\end{align*} \]

**Answer:**

\[ \begin{align*}
x &= 5 \\
y &= 3 \\
z &= \text{arbitrary integer}
\end{align*} \]
**Integer Gaussian Elimination**

Find matrices $U_1, U_2, \ldots, U_k$ such that $A \cdot U_1 \cdot U_2 \cdot \ldots \cdot U_k$ is lower triangular (say $L$)

Solve $Lx' = b$ (easy)

Compute $x = (U_1 \cdot U_2 \cdot \ldots \cdot U_k) x'$

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

**Proof:**

Overall strategy: Given $Ax = b$

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

Find matrices $U_1, U_2, \ldots, U_k$ such that

Given $Ax = b$
Caution: Not all column operations preserve integer solutions.

Question: Can we stay purely in the integer domain?

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

Solution: Use only unimodular column operations.

Which has no integer solution!

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

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

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

Intuition: With some column operations, recovering solutions of original system requires solving lower triangular system using rationals.
Unimodular Column Operations:

(a) Interchange two columns

(b) Negate a column

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

Check

Check

Check

Let \( x', y' \) satisfy second equation.

Let \( x', y' \) satisfy first equation.

Unimodular Column Operations:
Facts:

1. The product of two unimodular matrices is also unimodular.

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 three unimodular column operations are:
   - negating a column
   - adding an integer multiple of one column to another
   - interchanging two columns

On the matrix $A$ of the system $A \mathbf{x} = \mathbf{b}$, integer solutions are preserved by sequences of these operations.
Algorithm:

1. Use unimodular column operations to reduce matrix $A$ to lower triangular form when we have under-determined systems.

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. Then $x = Ux'$ where $x'$ is the solution of the system $Lx' = b$.

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

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

Column echelon form: Let $j$ be the row containing the first non-zero column. Then

- $(i)$ $r_{j+1} < r_j$ if column $j$ is not entirely zero.
- $(ii)$ Column $(j+1)$ is zero if column $j$ is.

To compute column echelon form of matrix, you should write down the solution for this system in column $(j)$.

Detail: Instead of lower triangular but not column echelon form.

$x = Lx'$, where $x'$ is the solution of the system $Lx' = b$.

- $x = Lx'$, where $x'$ is the solution of the system $Lx' = b$.
- If unimodular matrices corresponding to the column operations.
- If explicit form of solutions is desired, let $U$ be the product of unimodular matrices.
- $Lx' = b$ has integer solutions, so does the original system.
- To lower triangular form $L$.

Given a system of Diophantine equations $Ax = b$.