

Polyhedral model

What is Polyhedral model?

The polyhedral model is a [compilation technique](#) which

- treats each loop iteration within nested loops as [lattice points](#) inside mathematical objects called [polyhedra](#)
- performs [affine transformations](#) or more general non-affine transformations such as [tiling](#) on the polytopes
- converts the transformed polytopes into equivalent, but optimized (depending on targeted optimization goal), loop nests through polyhedra scanning.
- Nested loop programs are the typical, but not the only example, and the most common use of the model is for [loop nest optimization](#) in [program optimization](#)

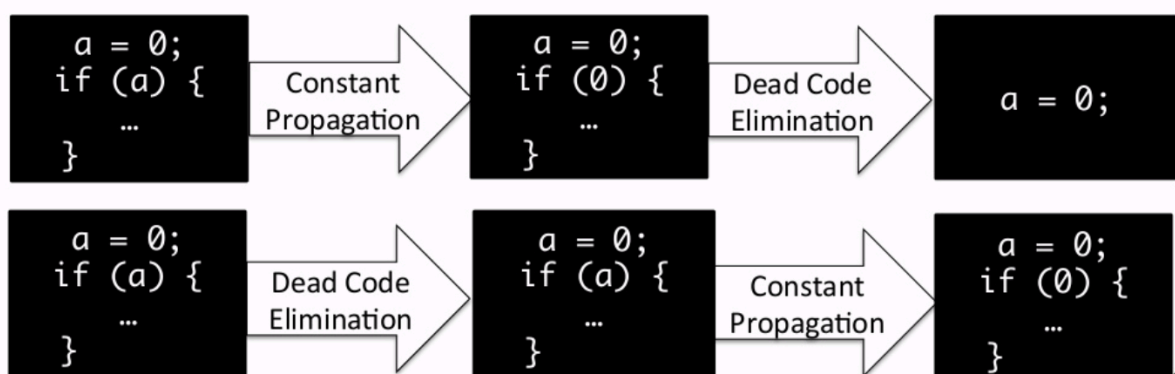
[Introduction to Polyhedral Compilation](#)

Why Polyhedral Model

The Polyhedral Model is a convenient alternative representation which combines analysis power, expressiveness and high flexibility” - OpenScop Specification and Library

- In contrast to Abstract Syntax Tree
- One solution for tackling the phase-ordering problem

❑ Which order is better?



- Good for performing a set of loop transformations
 - Loop permutation (interchange) : stride access or offset access
 - Loop fusion/distribution:

Fused	Distributed
<pre> for (i = 0; i < N; i++) { a[i] = b[i] + c[i]; d[i] = a[i] + e[i]; } </pre>	<pre> for (i = 0; i < N; i++) { a[i] = b[i] + c[i]; } for (i = 0; i < N; i++) { d[i] = a[i] + e[i]; } </pre>

Better temporal locality
on CPUs

Good for Vectorization
on CPUs



Depending on the loop size "N"

13

- Loop tiling

Schedules

A transformation for an iteration vector

Scalar Dimensions

$$T_s(\vec{i}) = \begin{pmatrix} \phi_s^1(\vec{i}) \\ \phi_s^2(\vec{i}) \\ \phi_s^3(\vec{i}) \\ \phi_s^4(\vec{i}) \\ \vdots \\ \phi_s^d(\vec{i}) \end{pmatrix} = \begin{pmatrix} C_{11}^S & C_{12}^S & C_{13}^S & C_{14}^S & \dots & C_{1m_s}^S \\ C_{21}^S & C_{22}^S & C_{23}^S & C_{24}^S & \dots & C_{2m_s}^S \\ C_{31}^S & C_{32}^S & C_{33}^S & C_{34}^S & \dots & C_{3m_s}^S \\ C_{41}^S & C_{42}^S & C_{43}^S & C_{44}^S & \dots & C_{4m_s}^S \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ C_{d1}^S & C_{d2}^S & C_{d3}^S & C_{d4}^S & \dots & C_{dm_s}^S \end{pmatrix} \begin{pmatrix} \vec{i} \end{pmatrix} + \begin{pmatrix} C_{10}^S \\ C_{20}^S \\ C_{30}^S \\ C_{40}^S \\ \vdots \\ C_{d0}^S \end{pmatrix}$$

Schedules
e.g., (0, i, 0, j)

m_s

d

1

$d = 2m_s + 1, m_s = \text{the size of iteration vector}$



32

- s denotes a statement, \vec{i} denotes the iteration vector
- Function T : return the logical state of each statement

An Example: Loop permutation

Loop transformations with schedules: Loop Permutation

Original schedule

$T_{S1}(i, j) = (i, j);$

$$T_{S1}(i, j) = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} i \\ j \end{pmatrix} = \begin{pmatrix} j \\ i \end{pmatrix}$$

Transformation Iteration Vector New Schedule

New schedule

$T_{S1}(i, j) = (j, i);$

Original

```
for (i = 0; i < 2; i++) {
  for (j = 0; j < 3; j++) {
    b[i][j] = ...; // S1
  }
}
```

New

```
for (j = 0; j < 3; j++) {
  for (i = 0; i < 2; i++) {
    b[i][j] = ...; // S1
  }
}
```

3 important things

- Domain: a set of instances for a statement
- Scattering (Scheduling): an instance \rightarrow time stamp (function T')
- Access: an instance \rightarrow array elements

Limitation

- Only applicable for Static Control Part (SCoP) in general \rightarrow Loop bounds and conditions are affine function of the surroundings the loop iterators.

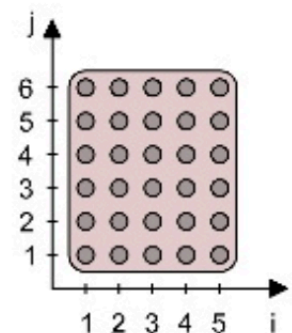
Iteration Domain

Iteration Domain

```
for (i=1; i <= 5; i++)
  for (j=1; j <= 6; j++)
    S1;
```

$$D^{S1} = \begin{pmatrix} 1 & 0 & -1 \\ -1 & 0 & 5 \\ 0 & 1 & -1 \\ 0 & -1 & 6 \end{pmatrix} \begin{pmatrix} i \\ j \\ 1 \end{pmatrix} \geq 0$$

$1 \leq i \leq 5, 1 \leq j \leq 6;$



- A set of constraints to represent instances of a statement
 - Using iteration vectors $(i, j);$
 - If those constraints are affine \rightarrow Polyhedron

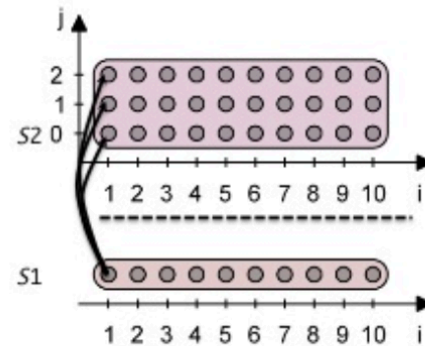
- If those constraints are affine -> Polyhedron (limitation)

Legality

Dependence polyhedron

```
for (i = 1; i <= 10; i++)
  s[i] = ...; // S1
  for (j = 0; j < 3; j++)
    a[i][j] = s[i]; // S2
```

$$\begin{array}{l}
 i_{s1} = i_{s2} \\
 1 \leq i_{s1} \leq 10, \\
 1 \leq i_{s2} \leq 10, \\
 0 \leq j_{s2} < 3;
 \end{array}
 \begin{array}{l}
 D^{s1} \\
 \Rightarrow \\
 D^{s2}
 \end{array}
 \begin{pmatrix}
 1 & -1 & 0 & 0 \\
 1 & 0 & 0 & -1 \\
 -1 & 0 & 0 & 10 \\
 0 & 0 & 1 & 0 \\
 0 & 0 & -1 & 2
 \end{pmatrix}
 \begin{pmatrix}
 i_{s1} \\
 i_{s2} \\
 j_{s2} \\
 1
 \end{pmatrix}
 \begin{array}{l}
 = 0 \\
 \geq
 \end{array}$$



□ Dependence polyhedron : a set of inequalities ($i_{s1} = i_{s2} \Rightarrow i_{s1} - i_{s2} \geq 0 \wedge i_{s2} - i_{s1} \geq 0$)

□ A general and accurate representation of instance-wise dependences

□ Dependence polyhedron: P_e

□ Legality:

- $\forall \langle s, t \rangle \in P_e, (s \in D^{s_i}, t \in D^{s_j}), T_{s_i}(s) < T_{s_j}(t)$
- If “source” instance must happen before “target” instance in the original program, the transformed program must preserve this property (must satisfy the dependence)