

# Image and Preimage

## Definition (Image)

The image of a polyhedron  $\mathcal{P} \in \mathbb{Z}^n$  by an affine function  $f : \mathbb{Z}^n \rightarrow \mathbb{Z}^m$  is a  $\mathbb{Z}$ -polyhedron  $\mathcal{P}'$ :

$$\mathcal{P}' = \{f(\vec{x}) \in \mathbb{Z}^m \mid \vec{x} \in \mathcal{P}\}$$

## Definition (Preimage)

The preimage of a polyhedron  $\mathcal{P} \in \mathbb{Z}^n$  by an affine function  $f : \mathbb{Z}^n \rightarrow \mathbb{Z}^m$  is a  $\mathbb{Z}$ -polyhedron  $\mathcal{P}'$ :

$$\mathcal{P}' = \{\vec{x} \in \mathbb{Z}^n \mid f(\vec{x}) \in \mathcal{P}\}$$

We have  $Image(f^{-1}, \mathcal{P}) = Preimage(f, \mathcal{P})$  if  $f$  is invertible.

# Returning to the Example

## Exercise: Compute the set of cells of A accessed

### Example

```
for (i = 0; i < N; ++i)
  for (j = i; j < N; ++j)
    A[2i + 3][4j] = i * j;
```

- ▶  $\mathcal{D}_S: \{i, j \mid 0 \leq i < N, i \leq j < N\}$
- ▶ Function:  $f_A: \{2i + 3, 4j \mid i, j \in \mathbb{Z}\}$
- ▶  $\text{Image}(f_A, \mathcal{D}_S)$  is the set of cells of A accessed (a  $\mathbb{Z}$ -polyhedron):
  - ▶ Polyhedron:  $Q: \{i, j \mid 3 \leq i < 2N + 2, 0 \leq j < 4N\}$
  - ▶ Lattice:  $L: \{2i + 3, 4j \mid i, j \in \mathbb{Z}\}$

# Data Dependence

## Definition (Bernstein conditions)

Given two references, there exists a dependence between them if the three following conditions hold:

- ▶ they reference the same array (cell)
- ▶ one of this access is a write
- ▶ the two associated statements are executed

Three categories of dependences:

- ▶ RAW (Read-After-Write, aka flow): first reference writes, second reads
- ▶ WAR (Write-After-Read, aka anti): first reference reads, second writes
- ▶ WAW (Write-After-Write, aka output): both references writes

Another kind: RAR (Read-After-Read dependences), used for locality/reuse computations

# Purpose of Dependence Analysis

- ▶ Not all program transformations preserve the semantics
- ▶ Semantics is preserved if the dependence are preserved
- ▶ In standard frameworks, it means reordering statements
  - ▶ Statements containing dependent references should not be executed in a different order
  - ▶ Granularity: usually a reference to an **array**
- ▶ In the polyhedral framework, it means reordering statement **instances**
  - ▶ Statement instances containing dependent references should not be executed in a different order
  - ▶ Granularity: a reference to an **array cell**

# An Intuitive Dependence Test Algorithm

**Idea: compute the sets associated to the Bernstein conditions**

Given two references  $a$  and  $b$  to the same array:

- ▶ Compute  $\mathcal{W}_a: \text{Image}(f_a, \mathcal{D}_a)$  if  $a$  is a write,  $\emptyset$  otherwise
  - ▶ Compute  $\mathcal{R}_a: \text{Image}(f_a, \mathcal{D}_a)$  if  $a$  is a read,  $\emptyset$  otherwise
  - ▶ Compute  $\mathcal{W}_b: \text{Image}(f_b, \mathcal{D}_b)$  if  $b$  is a write,  $\emptyset$  otherwise
  - ▶ Compute  $\mathcal{R}_b: \text{Image}(f_b, \mathcal{D}_b)$  if  $b$  is a read,  $\emptyset$  otherwise
- 
- ▶ If  $\mathcal{W}_a \cap \mathcal{R}_b \neq \emptyset \vee \mathcal{W}_a \cap \mathcal{W}_b \neq \emptyset \vee \mathcal{R}_a \cap \mathcal{W}_b \neq \emptyset$  then  $a\delta b$

# A (Naive) Dependence Test Algorithm

**Exercise: Write a dependence test algorithm for a program**

# A (Naive) Dependence Test Algorithm

## Exercise: Write a dependence test algorithm for a program

- ▶ Create the Data Dependence Graph, with one node per statement
- ▶ For all pairs  $a, b$  of distinct references, do
  - If  $a$  and  $b$  reference the same array, do
    - (i) Compute  $\mathcal{W}_a, \mathcal{R}_a, \mathcal{W}_b, \mathcal{R}_b$
    - (ii) If  $\mathcal{W}_a \cap \mathcal{R}_b \neq \emptyset \vee \mathcal{W}_a \cap \mathcal{W}_b \neq \emptyset \vee \mathcal{R}_a \cap \mathcal{W}_b \neq \emptyset$  then
      - Add an edge between the statement with the reference  $a$  and the statement with the reference  $b$  in the DDG

# Connection with Statement Instances

Objective: get the set of **instances** which are in dependence, not only statements

**Exercise:** Compute this set, from  $\mathcal{W}_a$  and  $\mathcal{R}_b$  (RAW dependence)



# Connection with Statement Instances

Objective: get the set of **instances** which are in dependence, not only statements

**Exercise: Compute this set, from  $\mathcal{W}_a$  and  $\mathcal{R}_b$  (RAW dependence)**

- ▶ Idea:  $Preimage(f_a, \mathcal{W}_a \cap \mathcal{R}_b)$  gives the set of instances
- ▶ Must generalize to multiple references, we lose convexity (unions)

# Dependence Polyhedron [1/3]

Principle: model all **pairs** of instances in dependence

## Definition (Dependence of statement instances)

A statement  $S$  depends on a statement  $R$  (written  $R \rightarrow S$ ) if there exists an operation  $S(\vec{x}_S)$  and  $R(\vec{x}_R)$  and a memory location  $m$  such that:

- ①  $S(\vec{x}_S)$  and  $R(\vec{x}_R)$  refer to the same memory location  $m$ , and at least one of them writes to that location,
- ②  $x_S$  and  $x_R$  belongs to the iteration domain of  $R$  and  $S$ ,
- ③ in the original sequential order,  $S(\vec{x}_S)$  is executed before  $R(\vec{x}_R)$ .

## Dependence Polyhedron [2/3]

- 1 *Same memory location*: equality of the subscript functions of a pair of references to the same array:  $F_S \vec{x}_S + a_S = F_R \vec{x}_R + a_R$ .
- 2 *Iteration domains*: both  $S$  and  $R$  iteration domains can be described using affine inequalities, respectively  $A_S \vec{x}_S + c_S \geq 0$  and  $A_R \vec{x}_R + c_R \geq 0$ .
- 3 *Precedence order*: each case corresponds to a common loop depth, and is called a *dependence level*.

For each dependence level  $l$ , the precedence constraints are the equality of the loop index variables at depth lesser to  $l$ :  $x_{R,i} = x_{S,i}$  for  $i < l$  and  $x_{R,l} > x_{S,l}$  if  $l$  is less than the common nesting loop level. Otherwise, there is no additional constraint and the dependence only exists if  $S$  is textually before  $R$ .

Such constraints can be written using linear inequalities:

$$P_{l,S} \vec{x}_S - P_{l,R} \vec{x}_R + b \geq 0.$$

## Dependence Polyhedron [3/3]

The dependence polyhedron for  $R \rightarrow S$  at a given level  $l$  and for a given pair of references  $f_R, f_S$  is described as [Feautrier/Bastoul]:

$$\mathcal{D}_{R,S,f_R,f_S,l} : D \begin{pmatrix} \vec{x}_S \\ \vec{x}_R \end{pmatrix} + d = \begin{bmatrix} F_S & -F_R \\ A_S & 0 \\ 0 & A_R \\ PS & -P_R \end{bmatrix} \begin{pmatrix} \vec{x}_S \\ \vec{x}_R \end{pmatrix} + \begin{pmatrix} a_S - a_R \\ c_S \\ c_R \\ b \end{pmatrix} \begin{matrix} = 0 \\ \geq 0 \end{matrix}$$

A few properties:

- ▶ We can always build the dep polyhedron for a given pair of affine array accesses (it is convex)
- ▶ It is exact, if the iteration domain and the access functions are also exact
- ▶ it is over-approximated if the iteration domain or the access function is an approximation

# Polyhedral Representation of Programs

## Static Control Parts

- ▶ Loops have affine control only (over-approximation otherwise)

# Polyhedral Representation of Programs

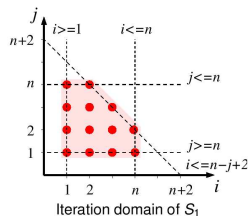
## Static Control Parts

- ▶ Loops have affine control only (over-approximation otherwise)
- ▶ Iteration domain: represented as integer polyhedra

```

for (i=1; i<=n; ++i)
. for (j=1; j<=n; ++j)
. . if (i<=n-j+2)
. . . s[i] = ...
  
```

$$\mathcal{D}_{S_1} = \begin{bmatrix} 1 & 0 & 0 & -1 \\ -1 & 0 & 1 & 0 \\ 0 & 1 & 0 & -1 \\ -1 & 0 & 1 & 0 \\ -1 & -1 & 1 & 2 \end{bmatrix} \cdot \begin{pmatrix} i \\ j \\ n \\ 1 \end{pmatrix} \geq \vec{0}$$



# Polyhedral Representation of Programs

## Static Control Parts

- ▶ Loops have affine control only (over-approximation otherwise)
- ▶ Iteration domain: represented as integer polyhedra
- ▶ Memory accesses: static references, represented as affine functions of  $\vec{x}_S$  and  $\vec{p}$

```
for (i=0; i<n; ++i) {
  . s[i] = 0;
  . for (j=0; j<n; ++j)
  . . s[i] = s[i]+a[i][j]*x[j];
}
```

$$f_{\mathbf{s}}(\vec{x}_{S2}) = \begin{bmatrix} \textcolor{red}{1} & 0 & 0 & 0 \end{bmatrix} \cdot \begin{pmatrix} \vec{x}_{S2} \\ n \\ 1 \end{pmatrix}$$

$$f_{\mathbf{a}}(\vec{x}_{S2}) = \begin{bmatrix} \textcolor{red}{1} & 0 & 0 & 0 \\ 0 & \textcolor{red}{1} & 0 & 0 \end{bmatrix} \cdot \begin{pmatrix} \vec{x}_{S2} \\ n \\ 1 \end{pmatrix}$$

$$f_{\mathbf{x}}(\vec{x}_{S2}) = \begin{bmatrix} 0 & \textcolor{red}{1} & 0 & 0 \end{bmatrix} \cdot \begin{pmatrix} \vec{x}_{S2} \\ n \\ 1 \end{pmatrix}$$

# Polyhedral Representation of Programs

## Static Control Parts

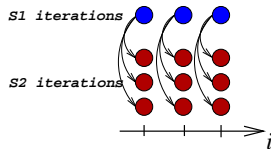
- ▶ Loops have affine control only (over-approximation otherwise)
- ▶ Iteration domain: represented as integer polyhedra
- ▶ Memory accesses: static references, represented as affine functions of  $\vec{x}_S$  and  $\vec{p}$
- ▶ Data dependence between S1 and S2: a subset of the Cartesian product of  $\mathcal{D}_{S1}$  and  $\mathcal{D}_{S2}$  (**exact analysis**)

```

for (i=1; i<=3; ++i) {
  . s[i] = 0;
  . for (j=1; j<=3; ++j)
    . . s[i] = s[i] + 1;
}

```

$$\mathcal{D}_{S1 \& S2} : \begin{bmatrix} 1 & -1 & 0 & 0 \\ 1 & 0 & 0 & -1 \\ -1 & 0 & 0 & 3 \\ 0 & 1 & 0 & -1 \\ 0 & -1 & 0 & 3 \\ 0 & 0 & 1 & -1 \\ 0 & 0 & -1 & 3 \end{bmatrix} \cdot \begin{pmatrix} i_{S1} \\ i_{S2} \\ j_{S2} \\ 1 \end{pmatrix} \begin{matrix} \equiv 0 \\ \geq 0 \end{matrix}$$





# A Dependence Polyhedra Construction Algorithm

- 1 Initialize reduced dependence graph with one node per program statement
- 2 For each pairs of statements  $R, S$  do
- 3   For each pairs of distinct references  $f_R, f_S$  to the same array, do
- 4     If  $R, S$  does not share any loop,  $min\_depth = 0$  else  $min\_depth = 1$
- 5     For each level  $l$  from  $min\_depth$  to  $nb\_common\_loops$ , do
- 6       Build the dependence polyhedron  $\mathcal{D}_{R,S,f_R,f_S,l}$
- 7       If  $\mathcal{D}_{R,S,f_R,f_S,l} \neq \emptyset$  then
- 8          If  $f_R$  is a write and  $f_S$  is a read,  $type = RAW$
- 9          If  $f_R$  is a write and  $f_S$  is a write,  $type = WAW$
- 10        If  $f_R$  is a read and  $f_S$  is a write,  $type = WAR$
- 11        If  $f_R$  is a read and  $f_S$  is a read,  $type = RAR$
- 12         $add\_edge(R, S, \{l, \mathcal{D}_{R,S,f_R,f_S,l}, type\})$

# A First Intuition About Scheduling

**Intuition: the source iteration must be executed before the target iteration**

## Definition (Precedence condition)

Given  $\Theta_R$  a schedule for the instances of  $R$ ,  $\Theta_S$  a schedule for the instances of  $S$ . Then,  $\forall \langle \vec{x}_R, \vec{x}_S \rangle \in \mathcal{D}_{R,S}$ :

$$\Theta_R(\vec{x}_R) \prec \Theta_S(\vec{x}_S)$$

Next week: scheduling and semantics preservation (Farkas method, convex space of legal schedules, tiling hyperplane method)