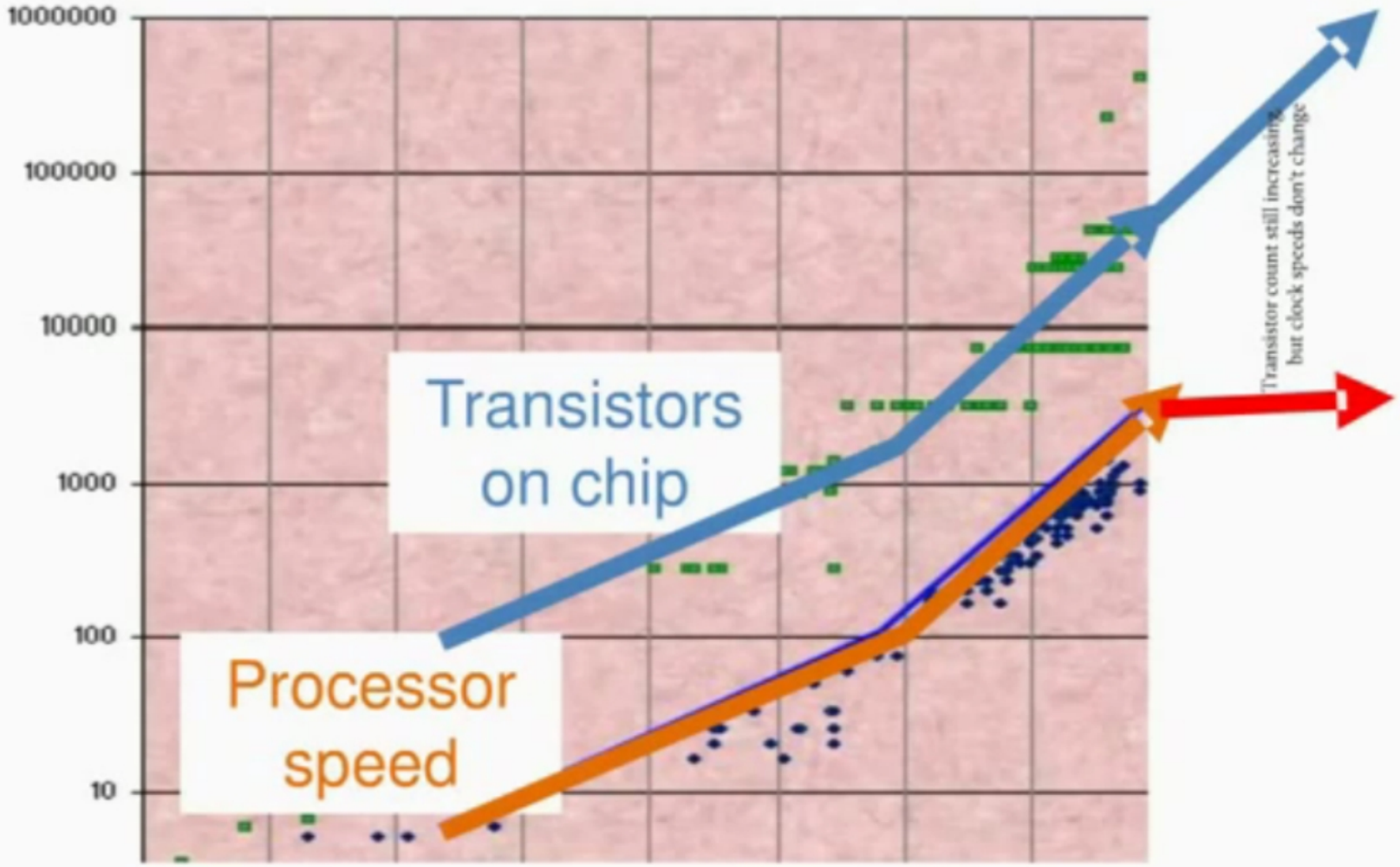
Introduction to Polyhedral Model

Chandan G

Department of Computer Science and Automation
Indian Institute of Science, Bangalore

Moore's Law


1000000
100000
10000
1000
100
10
Transistors on chip
Processor speed
Transistor count still increasing
but clock speeds don't change

Speed up with multicores

Example

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

Speed up with multicores

Example

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

Example

```
#pragma omp parallel
for (i = 0; i < N; ++i)
  for (j = 0; j < N; ++j)
    A[i][j] = i * j;
```

Speed up with multicores

Example

```
for (i = 0; i < N; ++i)
  for (j = 0; j < N; ++j)
    A[i][j] = A[i][j-1]+ 3;
```

Example

```
for (i = 0; i < N; ++i)
  for(j = 0; j < N; ++j)
   A[i][j]=(A[i-1][j-1]+A[i-1][j]+A[i][j-1]) / 3;
```

How to Exploit Parallelism?

- **Parallel Programming is hard**
- Can a Compiler automatically parallelize the serial code?
- Parallelization of Regular scientific applications
- Most of time is spent in regular for loops
- Arrays as main data structure
- **Polyhedral Model** is a mathematical, high level IR of program, which helps in reasoning about parallelism

Motivating Example [1/2]

Example

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

Program execution:

```
1: A[0][0] = 0 * 0;
2: A[0][1] = 0 * 1;
3: A[0][2] = 0 * 2;
4: A[1][0] = 1 * 0;
5: A[1][1] = 1 * 1;
6: A[1][2] = 1 * 2;
```

Motivating Example [2/2]

A few observations:

- Statement is executed 6 times
- There are different values of i, j associated with these 6 instances
- There is an order on them (the execution order)

Objective:
Construct a representation where these 3 characteristics are captured

Polyhedral Representation

From syntactic code to polyhedral representation

- Modeling iteration domains with polytopes

Underlying mathematical concepts:

- Affine functions, hyperplanes, affine transformations
- Polyhedra (bounded, rational, integer and parametric)

Definition (**Linear**)

A k-dimensional function f is linear iff it can expressed in the following form:

$$linear \quad function \quad f(\vec{v}) = M_f \vec{v} \tag{1}$$

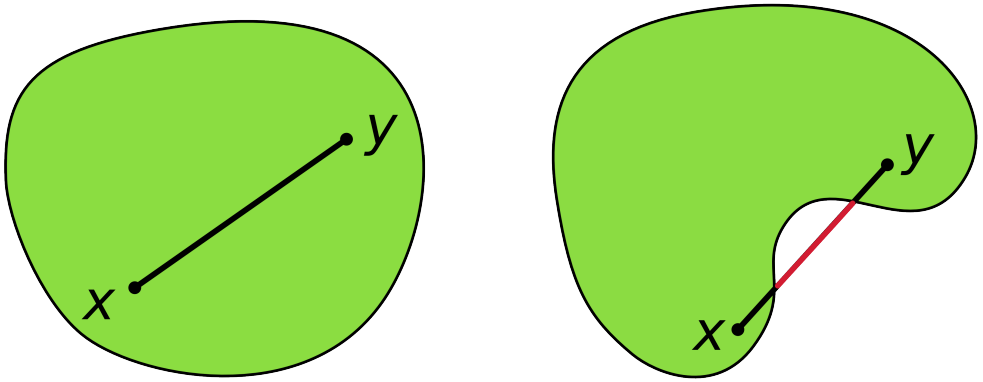
Definition (**Affine**)

A k-dimensional function f is affine iff it can expressed in the following form:

$$affine \quad function \quad f(\vec{v}) = M_f \vec{v} + \vec{f_0} \tag{2}$$

Definition (**Affine sub-space**)

A set of vectors is an affine sub-space of a vector space iff it is closed under affine combination, i.e., if $\vec{x}, \vec{y}$ are in the space, all points lying on the line joining $\vec{x}$ and $\vec{y}$ belong to the space.



Definition (**Affine hyperplane**)

An affine hyperplane is an $n-1$ dimensional affine sub-space of an n dimensional space.

$$\{\vec{v} \quad | \quad \mathbf{h}.\vec{v} = k\}$$

Polyhedron

Definition (Polyhedron)

A set $\mathcal{S} \in \mathbb{K}^m$ is a polyhedron if there exists a system of a finite number of affine inequalities $A\vec{x} \leq \vec{b}$ such that:

$$\mathcal{P} = \{\vec{x} \in \mathbb{K}^m \mid A\vec{x} \leq \vec{b}\}$$

Equivalently, it is the intersection of finitely many half-spaces.

Definition (Polytope)

A polytope is a bounded polyhedron.

Overview of the Polyhedral Representation

- Iteration domain: set of totally ordered n-dimensional vectors
  - **Iteration vector** $\vec{x}_S = (i, j)$
  - Iteration domain: the set of values of $\vec{x}_S$
- Convenient approach: **polytopes model sets** of totally ordered n-dimensional vectors

Returning to the Example

Modeling the iteration domain:

- Polytope dimension: set by the number of surrounding loops
- Constraints: set by the loop bounds

$$\mathcal{D}_R : \begin{bmatrix} 1 & 0 \\ -1 & 0 \\ 0 & 1 \\ 0 & -1 \end{bmatrix} \cdot \begin{pmatrix} i \\ j \end{pmatrix} + \begin{pmatrix} 0 \\ 2 \\ 0 \\ 2 \end{pmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ -1 & 0 & 2 \\ 0 & 1 & 0 \\ 0 & -1 & 2 \end{bmatrix} \cdot \begin{pmatrix} i \\ j \\ 1 \end{pmatrix} \geq \vec{0}$$

$$0 \leq i \leq 2, \quad 0 \leq j \leq 2$$

Increasing the Expressiveness

Problems:

- Parametric loop bounds: use parametric polyhedra!

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ -1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & -1 & 1 & 0 \end{bmatrix} \cdot \begin{pmatrix} i \\ j \\ N \\ 1 \end{pmatrix} \geq \vec{0}$$

$$0 \leq i \leq N, \quad 0 \leq j \leq N$$

Increasing the Expressiveness

Problems:

- Non-unit loop strides: normalize the loop!

Example

```
for (i = 0; i < N; i = i+5)
  A[i] = i + 10;
```

Example

```
for (i = 0; i < N/5; i++)
  A[i*5] = (i*5) + 10;
```

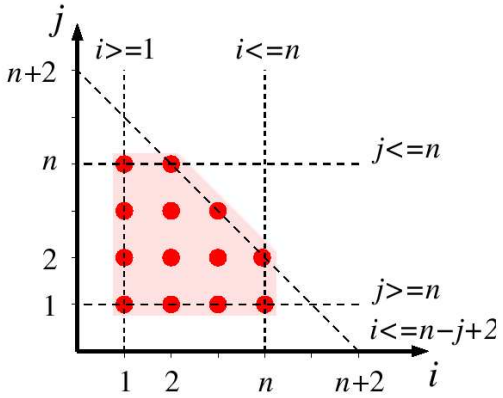
Increasing the Expressiveness

Problems:

- Conditionals:
  - Those which preserve convexity: ok! (add affine constraints)
  - Problem remains for the others...
  - Same can done for while loops which affine exit condition

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

$$\mathcal{D}_{S1} = \begin{bmatrix} 1 & 0 & 0 & -1 \\ -1 & 0 & 1 & 0 \\ 0 & 1 & 0 & -1 \\ -1 & 0 & 1 & 0 \\ -1 & -1 & 1 & 2 \end{bmatrix} \cdot \begin{pmatrix} i \\ j \\ n \\ 1 \end{pmatrix}$$



Iteration domain of S_1

Affine schedule

Definition (Affine schedule)

Given a statement S, a p-dimensional affine schedule $\mathbf{\Theta^S}$ is an affine form on the outer loop iterator $\vec{x}_S$ and the global parameters $\vec{n}$. It is written:

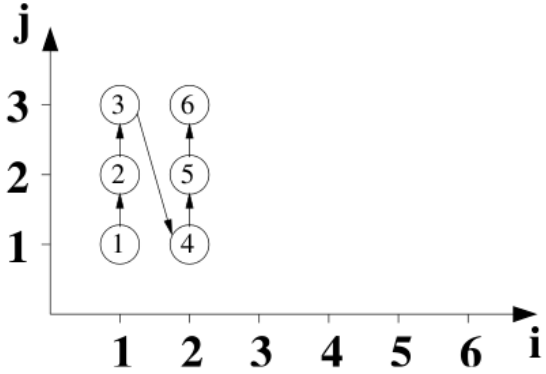
$$\mathbf{\Theta^S}(\vec{x}_S) = T_S \begin{pmatrix} \vec{x}_S \\ \vec{n} \\ 1 \end{pmatrix}, T_S \in \mathbb{K}^{p \times dim(\vec{x}_S) + dim(\vec{n}) + 1} \tag{3}$$

- **A schedule assigns a timestamp to each executed instance of a statement**
- Lexicographic ordering of timestamps

Scheduling Statement Instances Example[1/3]

Interchange Transformation

The transformation matrix is the identity with a permutation of two rows.



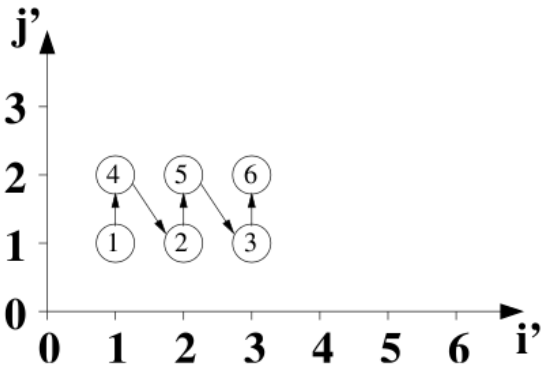
$$\begin{bmatrix} 1 & 0 \\ -1 & 0 \\ 0 & 1 \\ 0 & -1 \end{bmatrix} \begin{pmatrix} i \\ j \end{pmatrix} + \begin{pmatrix} -1 \\ 2 \\ -1 \\ 3 \end{pmatrix} \geq \vec{0}$$

(a) original polyhedron
$A\vec{x} + \vec{a} \geq \vec{0}$

$\Longrightarrow$

$$\begin{pmatrix} i' \\ j' \end{pmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{pmatrix} i \\ j \end{pmatrix}$$

(b) transformation function
$\vec{y} = T\vec{x}$



$$\begin{bmatrix} 0 & 1 \\ 0 & -1 \\ 1 & 0 \\ -1 & 0 \end{bmatrix} \begin{pmatrix} i' \\ j' \end{pmatrix} + \begin{pmatrix} -1 \\ 2 \\ -1 \\ 3 \end{pmatrix} \geq \vec{0}$$

(c) target polyhedron
$(AT^{-1})\vec{y} + \vec{a} \geq \vec{0}$

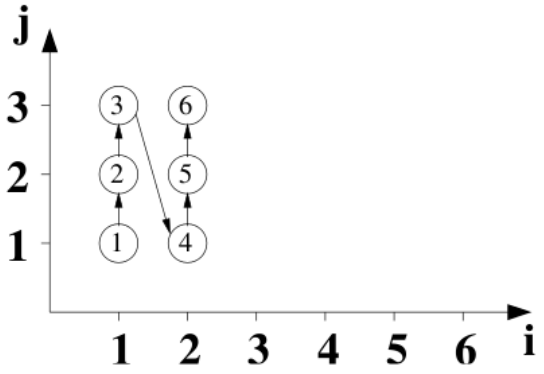
```
do i = 1, 2
  do j = 1, 3
```

```
do j = 1, 3
  do i = 1, 2
```

Scheduling Statement Instances Example[2/3]

Reversal Transformation

The transformation matrix is the identity with one diagonal element replaced by -1.



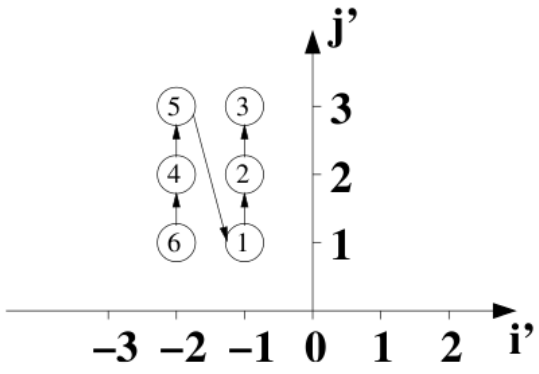
$$\begin{bmatrix} 1 & 0 \\ -1 & 0 \\ 0 & 1 \\ 0 & -1 \end{bmatrix} \begin{pmatrix} i \\ j \end{pmatrix} + \begin{pmatrix} -1 \\ 2 \\ -1 \\ 3 \end{pmatrix} \geq \vec{0}$$

(a) original polyhedron
$A\vec{x} + \vec{a} \geq \vec{0}$

$$\Longrightarrow$$

$$\begin{pmatrix} i' \\ j' \end{pmatrix} = \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix} \begin{pmatrix} i \\ j \end{pmatrix}$$

(b) transformation function
$\vec{y} = T\vec{x}$



$$\begin{bmatrix} -1 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & -1 \end{bmatrix} \begin{pmatrix} i' \\ j' \end{pmatrix} + \begin{pmatrix} -1 \\ 2 \\ -1 \\ 3 \end{pmatrix} \geq \vec{0}$$

(c) target polyhedron
$(AT^{-1})\vec{y} + \vec{a} \geq \vec{0}$

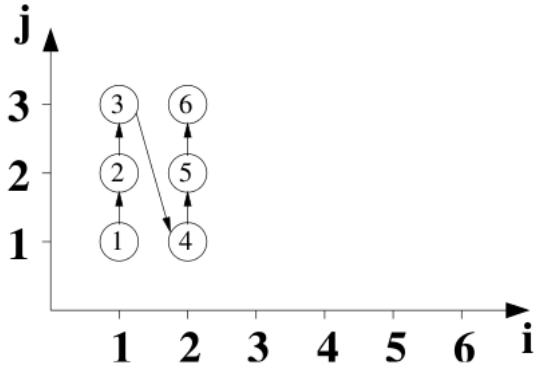
```
do i = 1, 2
  do j = 1, 3
```

```
do i = -1, -2, -1
  do j = 1, 3
```

Scheduling Statement Instances Example[3/3]

Coumpound Transformation

The transformation matrix is the composition of an interchange and reversal



$$\begin{bmatrix} 1 & 0 \\ -1 & 0 \\ 0 & 1 \\ 0 & -1 \end{bmatrix} \begin{pmatrix} i \\ j \end{pmatrix} + \begin{pmatrix} -1 \\ 2 \\ -1 \\ 3 \end{pmatrix} \geq \vec{0}$$

(a) original polyhedron

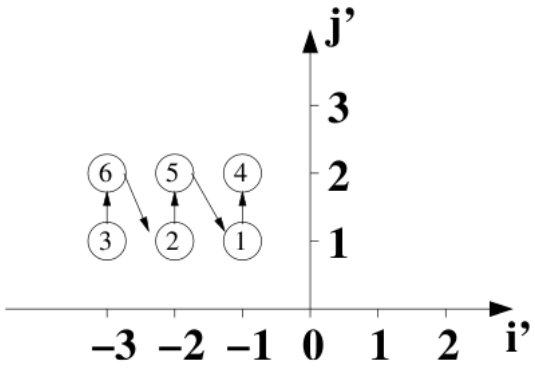
$A\vec{x} + \vec{a} \geq \vec{0}$

$\Longrightarrow$

$$\begin{pmatrix} i' \\ j' \end{pmatrix} = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} \begin{pmatrix} i \\ j \end{pmatrix}$$

(b) transformation function

$\vec{y} = T\vec{x}$



$$\begin{bmatrix} 0 & -1 \\ 0 & 1 \\ 1 & 0 \\ -1 & 0 \end{bmatrix} \begin{pmatrix} i' \\ j' \end{pmatrix} + \begin{pmatrix} -1 \\ 2 \\ -1 \\ 3 \end{pmatrix} \geq \vec{0}$$

(c) target polyhedron

$(AT^{-1})\vec{y} + \vec{a} \geq \vec{0}$

```
do i = 1, 2
  do j = 1, 3
```

```
do j = -1, -3, -1
  do i = 1, 2
```

Are all Transformations Valid?

Example

```
for (i = 1; i < N; ++i)
  A[i] = (A[i-1] + A[i] ) / 2;
```

Is Loop Reversal valid?

Data Dependence

Definition (Conditions)

Given two iterations, there exists a dependence between them if the following conditions are met:

- they reference the same memory location
- one of the accesses is a write

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

Illustrations

Example

```
for (i = 0; i < N; ++i)
  for (j = 0; j < N; ++j)
    A[i][j] = A[i-1][j-1];
```

Example

```
for (i = 0; i < N; ++i)
  for(j = 0; j < N; ++j)
   A[i][j]=(A[i-1][j-1]+A[i-1][j]+A[i-1][j+1]) / 3;
```

Example

```
for (i = 0; i < N; ++i) {
  s[i] = i;
  for (j = 0; j < N; ++j)
    t[i] = s[i] + 1;
 }
```

Some Terminology on Dependence Relations

We categorize the dependence relation in three kinds:

- **Uniform dependences:** the distance between two dependent iterations is a constant
  - ex: $i \rightarrow i + 1$
  - ex: $i, j \rightarrow i + 1, j + 1$
- **Non-uniform dependences:** the distance between two dependent iterations varies along the execution
  - ex: $i \rightarrow i + j$
  - ex: $i \rightarrow 2i$
- **Parametric dependences:** at least a parameter is involved in the dependence relation
  - ex: $i \rightarrow i + N$
  - ex: $i + N \rightarrow j + M$

Data Dependence Analysis

Objective: compute the set of statement instances which are dependent

Some of the several possible approaches:

- **Distance vector:** compute an indicator of the distance between two dependent iteration
  - Problems: approximative for non-uniform dependences
- Best solution: **Dependence polyhedron**, list of sets of dependent instances, with a set of dependence polyhedra for each pair of statements

Dependence Polyhedron

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

Definition (Dependence of statement instances)

A statement R depends on a statement S (written $S \rightarrow R$) 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

Data Dependent Graph

There is an edge between statements if they are depedent. And for each edge we define a dependence polyhedron for $S \rightarrow R$ at a given level l and for a given pair of references f_R, f_S

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
- Can be represented as matrix, since all the constraints that define are affine

Legal Program Transformation

- A Transformation is illegal if a dependence crosses the hyperplane backwards
- A dependence going in forward direction between two hyperplanes indicates sequentiality
- No dependence between any point of the hyperplane indicates parallelism

Definition (Precedence condition)

Given $\mathbf{\Theta^T}$ a schedule for the instance of T, $\mathbf{\Theta^S}$ a schedule for the instance of S. $\mathbf{\Theta^T}$ and $\mathbf{\Theta^S}$ are legal schedules if
$\forall < \vec{x}_S, \vec{x}_T > \in D_{S,T} : \mathbf{\Theta^S} << \mathbf{\Theta^T}$
Equivalently: $\Delta_{T,S} = \mathbf{\Theta^T} - \mathbf{\Theta^S} >= 1$

Connection with Parallelism

- A dependence is loop-carried if 2 iterations of this loop access the same array cell
- If no such dependence exists, then loop is parallel

Example

```
for (i = 0; i < N; ++i)
  for (j = 0; j < N; ++j)
    C[i][j] = 0;
```

Example

```
for (i = 0; i < N; ++i)
  for (j = 0; j < N; ++j)
    A[i][j] = A[i][j -1]*B[i][j];
```

Polyhedral Representation of Programs

Static Control Parts

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

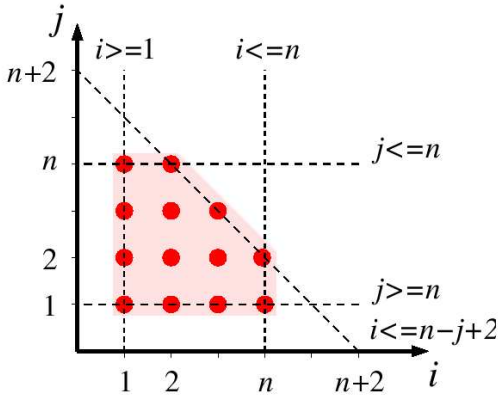
Polyhedral Representation of Programs

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] = ...
```

$$\mathcal{D}_{S1} = \begin{bmatrix} 1 & 0 & 0 & -1 \\ -1 & 0 & 1 & 0 \\ 0 & 1 & 0 & -1 \\ -1 & 0 & 1 & 0 \\ -1 & -1 & 1 & 2 \end{bmatrix} \cdot \begin{pmatrix} i \\ j \\ n \\ 1 \end{pmatrix}$$

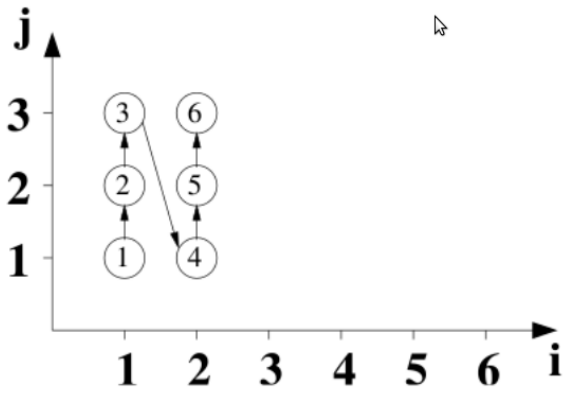


Iteration domain of S_1

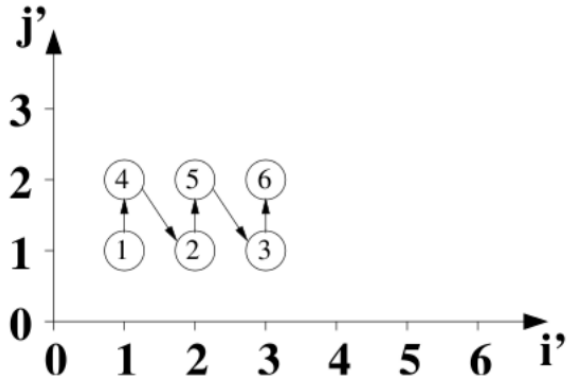
Polyhedral Representation of Programs

Static Control Parts

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



$\Longrightarrow$



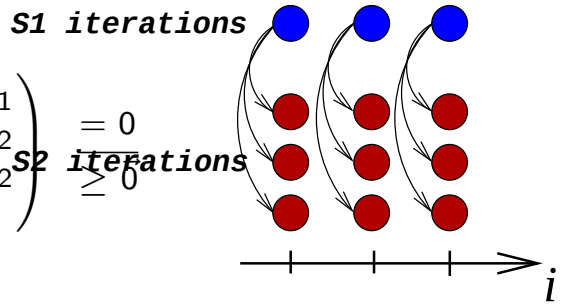
Polyhedral Representation of Programs

Static Control Parts

- Loops have affine control only (over-approximation otherwise)
- Iteration domain: represented as integer polyhedra
- Affine schedule: represented as Affine function
- 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;
}
```

$$\mathcal{D}_{S1\delta S2} : \left[\begin{array}{cccc} 1 & -1 & 0 & 0 \\ \hline 1 & 0 & 0 & -1 \\ -1 & 0 & 0 & 3 \\ 0 & 1 & 0 & -1 \\ 0 & -1 & 0 & 3 \\ 0 & 0 & 1 & -1 \\ 0 & 0 & -1 & 3 \end{array}\right] \cdot \begin{pmatrix} i_{S1} \\ i_{S2} \\ j_{S2} \\ 1 \end{pmatrix} \begin{array}{c} = 0 \\ \hline \geq 0 \end{array}$$



Polyhedral Representation of Programs

Static Control Parts

- Loops have affine control only (over-approximation otherwise)
- Iteration domain: represented as integer polyhedra
- Affine schedule: represented as Affine function
- Data dependence between S1 and S2: a subset of the Cartesian product of $\mathcal{D}_{S1}$ and $\mathcal{D}_{S2}$ (**exact analysis**)
- 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];
}
```

$$f_{\mathtt{s}}(\vec{x_{S2}}) = \begin{bmatrix} 1 & 0 & 0 & 0 \end{bmatrix} . \begin{pmatrix} \vec{x_{S2}} \\ n \\ 1 \end{pmatrix}$$

$$f_{\mathtt{a}}(\vec{x_{S2}}) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} . \begin{pmatrix} \vec{x_{S2}} \\ n \\ 1 \end{pmatrix}$$

$$f_{\mathtt{x}}(\vec{x_{S2}}) = \begin{bmatrix} 0 & 1 & 0 & 0 \end{bmatrix} . \begin{pmatrix} \vec{x_{S2}} \\ n \\ 1 \end{pmatrix}$$

Program Transformations Example

Representing a schedule

```
 for (i = 0; i < n; ++i)
  for (j = 0; j < n; ++j){
S1: C[i][j] = 0;
   for (k = 0; k < n; ++k)
S2:  C[i][j] += A[i][k]*
                B[k][j];
  }
```

$$\Theta^{S1}.\vec{x}_{S1} = \begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix} \cdot \begin{pmatrix} i \\ j \\ n \\ 1 \end{pmatrix}$$

$$\Theta^{S2}.\vec{x}_{S2} = \begin{pmatrix} 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{pmatrix} \cdot \begin{pmatrix} i \\ j \\ k \\ n \\ 1 \end{pmatrix}$$

```
for (i = n; i < 2*n; ++i)
 for (j = 0; j < n; ++j)
  C[i][j] = 0;
for (k= n+1; k<= 2*n; ++k)
 for (j = 0; j < n; ++j)
  for (i = 0; i < n; ++i)
   C[i][j] += A[i][k-n-1]*
              B[k-n-1][j];
```

	Transformation	Description
$\vec{\imath}$	`reversal`	Changes the direction in which a loop traverses its iteration range
	`skewing`	Makes the bounds of a given loop depend on an outer loop counter
	`interchange`	Exchanges two loops in a perfectly nested loop, a.k.a. `permutation`
$\vec{p}$	`fusion`	Fuses two loops, a.k.a. `jamming`
	`distribution`	Splits a single loop nest into many, a.k.a. `fission` or `splitting`
c	`peeling`	Extracts one iteration of a given loop
	`shifting`	Allows to reorder loops

Program Transformations Example

Original Schedule

```
 for (i = 0; i < n; ++i)
  for (j = 0; j < n; ++j){
S1: C[i][j] = 0;
   for (k = 0; k < n; ++k)
S2:  C[i][j] += A[i][k]*
                B[k][j];

  }
```

$$\Theta^{S1}.\vec{x}_{S1} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix} \cdot \begin{pmatrix} \mathbf{i} \\ \mathbf{j} \\ \mathbf{n} \\ \mathbf{1} \end{pmatrix}$$

$$\Theta^{S2}.\vec{x}_{S2} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{pmatrix} \cdot \begin{pmatrix} \mathbf{i} \\ \mathbf{j} \\ \mathbf{k} \\ \mathbf{n} \\ \mathbf{1} \end{pmatrix}$$

```
for (i = 0; i < n; ++i)
 for (j = 0; j < n; ++j){
  C[i][j] = 0;
  for (k = 0; k < n; ++k)
   C[i][j] += A[i][k]*
              B[k][j];

 }
```

Program Transformations Example

Distribute loops

```
 for (i = 0; i < n; ++i)
  for (j = 0; j < n; ++j){
S1: C[i][j] = 0;
   for (k = 0; k < n; ++k)
S2:  C[i][j] += A[i][k]*
                B[k][j];
  }
```

$$\Theta^{S1}.\vec{x}_{S1} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix} \cdot \begin{pmatrix} \mathbf{i} \\ \mathbf{j} \\ \mathbf{n} \\ \mathbf{1} \end{pmatrix}$$

$$\Theta^{S2}.\vec{x}_{S2} = \begin{pmatrix} 1 & 0 & 0 & \mathbf{1} & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{pmatrix} \cdot \begin{pmatrix} \mathbf{i} \\ \mathbf{j} \\ \mathbf{k} \\ \mathbf{n} \\ \mathbf{1} \end{pmatrix}$$

```
for (i = 0; i < n; ++i)
 for (j = 0; j < n; ++j)
  C[i][j] = 0;
for (i = n; i < 2*n; ++i)
 for (j = 0; j < n; ++j)
  for (k = 0; k < n; ++k)
   C[i-n][j] += A[i-n][k]*
                B[k][j];
```

Program Transformations Example

Distribute loops + Interchange loops for S2

```
 for (i = 0; i < n; ++i)
  for (j = 0; j < n; ++j){
S1: C[i][j] = 0;
   for (k = 0; k < n; ++k)
S2:  C[i][j] += A[i][k]*
                B[k][j];

  }
```

$$\Theta^{S1}.\vec{x}_{S1} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix} . \begin{pmatrix} \mathbf{i} \\ \mathbf{j} \\ \mathbf{n} \\ \mathbf{1} \end{pmatrix}$$

$$\Theta^{S2}.\vec{x}_{S2} = \begin{pmatrix} 0 & 0 & \mathbf{1} & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ \mathbf{1} & 0 & 0 & 0 & 0 \end{pmatrix} . \begin{pmatrix} \mathbf{i} \\ \mathbf{j} \\ \mathbf{k} \\ \mathbf{n} \\ \mathbf{1} \end{pmatrix}$$

```
for (i = 0; i < n; ++i)
 for (j = 0; j < n; ++j)
  C[i][j] = 0;
for (k = n; k < 2*n; ++k)
 for (j = 0; j < n; ++j)
  for (i = 0; i < n; ++i)
   C[i][j] += A[i][k-n]*
              B[k-n][j];
```

Program Transformations Example

Illegal schedule

```
 for (i = 0; i < n; ++i)
  for (j = 0; j < n; ++j){
S1: C[i][j] = 0;
   for (k = 0; k < n; ++k)
S2:  C[i][j] += A[i][k]*
                B[k][j];

  }
```

$$\Theta^{S1}.\vec{x}_{S1} = \begin{pmatrix} 1 & 0 & \mathbf{1} & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix} \cdot \begin{pmatrix} \mathbf{i} \\ \mathbf{j} \\ \mathbf{n} \\ \mathbf{1} \end{pmatrix}$$

$$\Theta^{S2}.\vec{x}_{S2} = \begin{pmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{pmatrix} \cdot \begin{pmatrix} \mathbf{i} \\ \mathbf{j} \\ \mathbf{k} \\ \mathbf{n} \\ \mathbf{1} \end{pmatrix}$$

```
for (k = 0; k < n; ++k)
 for (j = 0; j < n; ++j)
  for (i = 0; i < n; ++i)
   C[i][j] += A[i][k]*
              B[k][j];
for (i = n; i < 2*n; ++i)
 for (j = 0; j < n; ++j)
  C[i-n][j] = 0;
```

Program Transformations Example

Implicit fine-grain parallelism

```
 for (i = 0; i < n; ++i)
  for (j = 0; j < n; ++j){
S1: C[i][j] = 0;
   for (k = 0; k < n; ++k)
S2:  C[i][j] += A[i][k]*
                B[k][j];
  }
```

$$\Theta^{S1}.\vec{x}_{S1} = \begin{pmatrix} 1 & 0 & 0 & 0 \end{pmatrix} . \begin{pmatrix} \mathbf{i} \\ \mathbf{j} \\ \mathbf{n} \\ \mathbf{1} \end{pmatrix}$$

$$\Theta^{S2}.\vec{x}_{S2} = \begin{pmatrix} 0 & 0 & 1 & 1 & 0 \end{pmatrix} . \begin{pmatrix} \mathbf{i} \\ \mathbf{j} \\ \mathbf{k} \\ \mathbf{n} \\ \mathbf{1} \end{pmatrix}$$

```
for (i = 0; i < n; ++i)
 pfor (j = 0; j < n; ++j)
  C[i][j] = 0;
for (k = n; k < 2*n; ++k)
 pfor (j = 0; j < n; ++j)
  pfor (i = 0; i < n; ++i)
   C[i][j] += A[i][k-n]*
              B[k-n][j];
```

References

- **http://polly.llvm.org/**
- Chapter 11: Optimizing for Parallelism and Locality from **Compiler Principles and Techniques and Tools** book
- http://pluto-compiler.sourceforge.net/
- http://www.cse.ohio-state.edu/ pouchet/software/pocc/

Thank you
Any Questions?
chandan.g@csa.iisc.ernet.in