Polyhedral Compilation Foundations

Louis-Noël Pouchet

pouchet@cse.ohio-state.edu

Dept. of Computer Science and Engineering, the Ohio State University

Feb 8, 2010

888.11, Class #3

Overview of Today's Lecture

Outline:

- Transformation in the polyhedral representation
  - Affine scheduling
  - Scanning hyperplane
  - Legal transformation
- One-dimensional affine schedules
  - Convex set of legal schedules
  - Objective functions

Mathematical concepts:

- Affine functions
- Affine form of Farkas Lemma

Affine Scheduling

Definition (Affine schedule)

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

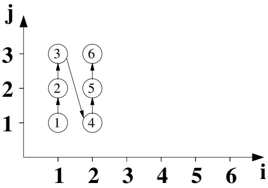
$$\Theta^S(\vec{x}_S) = \mathbf{T}_S \begin{pmatrix} \vec{x}_S \\ \vec{n} \\ 1 \end{pmatrix}, \quad \mathbf{T}_S \in \mathbb{K}^{p \times dim(\vec{x}_S) + dim(\vec{n}) + 1}$$

- **A schedule assigns a timestamp to each executed instance of a statement**
- If T is a vector, then Θ is a one-dimensional schedule
- If T is a matrix, then Θ is a multidimensional schedule

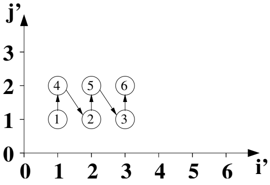
Scheduling Statement Instances

Interchange Transformation

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



$\Longrightarrow$



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

(a) original polyhedron

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

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

(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

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

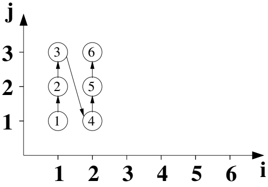
```
do i = 1, 2
  do j = 1, 3
```

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

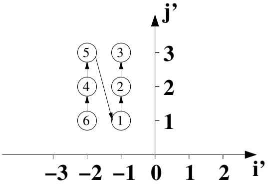
Scheduling Statement Instances

Reversal Transformation

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



$\Longrightarrow$



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

(a) original polyhedron

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

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

(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

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

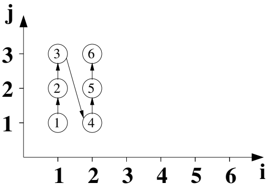
```
do i = 1, 2
  do j = 1, 3
```

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

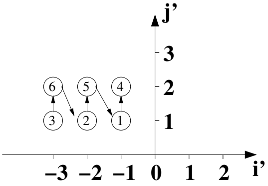
Scheduling Statement Instances

Coumpound Transformation

The transformation matrix is the composition of an interchange and reversal



$\Longrightarrow$



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

(a) original polyhedron

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

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

(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

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

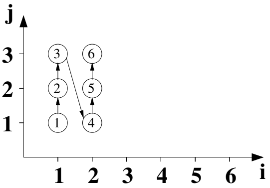
```
do i = 1, 2
  do j = 1, 3
```

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

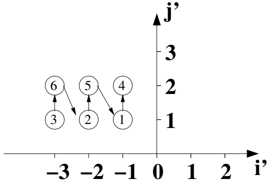
Scheduling Statement Instances

Coumpound Transformation

The transformation matrix is the composition of an interchange and reversal



$\Longrightarrow$



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

(a) original polyhedron

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

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

(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
```

Program Transformations

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];
  }
```

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

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

```
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];
 }
```

- Represent Static Control Parts (control flow and dependences must be statically computable)
- Use code generator (e.g. CLooG) to generate C code from polyhedral representation (provided iteration domains + schedules)

Program Transformations

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];
  }
```

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

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

```
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];
 }
```

- Represent Static Control Parts (control flow and dependences must be statically computable)
- Use code generator (e.g. CLooG) to generate C code from polyhedral representation (provided iteration domains + schedules)

Program Transformations

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];

  }
```

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

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

```
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];

 }
```

- Represent Static Control Parts (control flow and dependences must be statically computable)
- Use code generator (e.g. CLooG) to generate C code from polyhedral representation (provided iteration domains + schedules)

Program Transformations

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];

  }
```

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

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

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

- All instances of S1 are executed before the first S2 instance

Program Transformations

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];
  }
```

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

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

```
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];
```

- The outer-most loop for S2 becomes k

Program Transformations

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];
  }
```

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

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

```
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;
```

- **All instances of S1 are executed <u>after</u> the last S2 instance**

Program Transformations

A legal 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];

  }
```

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

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

```
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];
```

- **Delay the S2 instances**
- Constraints must be expressed between Θ^{S1} and Θ^{S2}

Program Transformations

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];
```

- Number of rows of $\Theta \leftrightarrow$ number of outer-most sequential loops

Program Transformations

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];

  }
```

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

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

```
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];
```

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

Program Transformations

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];

  }
```

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

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

```
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];
```

$$\Theta.\vec{x} = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} . \begin{pmatrix} \underbrace{i \;\; j \;\; i \;\; j \;\; k}_{\vec{\imath}} & \underbrace{n \;\; n}_{\vec{p}} & \underbrace{1 \;\; 1}_{\mathbf{c}} \end{pmatrix}^T$$

Program Transformations

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];
  }
```

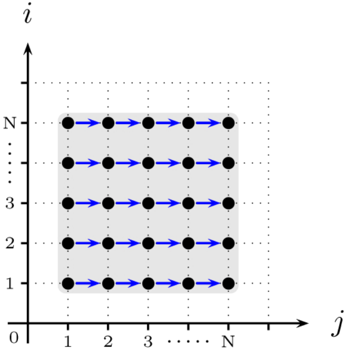
$$\Theta^{S1}.\vec{x}_{S1} = \begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix} . \begin{pmatrix} i \\ j \\ n \\ 1 \end{pmatrix}$$

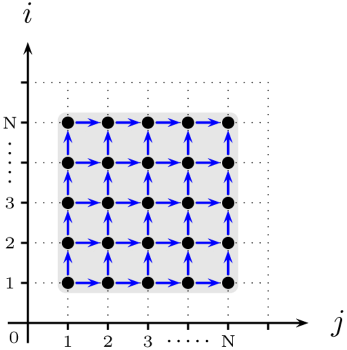
$$\Theta^{S2}.\vec{x}_{S2} = \begin{pmatrix} 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{pmatrix} . \begin{pmatrix} i \\ j \\ k \\ n \\ 1 \end{pmatrix}$$

```
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

Pictured Example





Example of 2 extended dependence graphs

Legal Program Transformation

A few properties:

- A transformation is illegal if a dependence crosses the hyperplane backwards
- A dependence going forward between 2 hyperplanes indicates sequentiality
- No dependence between any point of the hyperplane indicates parallelism

Definition (Precedence condition)

Given Θ^R a schedule for the instances of R, Θ^S a schedule for the instances of S. Θ^R and Θ^S are legal schedules if $\forall \langle \vec{x}_R, \vec{x}_S \rangle \in \mathcal{D}_{R,S}$:

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

Scheduling in the Polyhedral Model

Constraints:

- The schedule must be legal, for all dependences
- Dependence constraints have to be turned into constraints on the solution set

Scheduling:

- Among all possibilities, one has to be picked
- Optimal solution requires to consider all legal possible schedules

One-Dimensional Affine Schedules

For the rest of the lecture, we focus on 1-d schedules

Example

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

- Simple program: 1 loop, 1 polyhedral statement
- 2 dependences:
  - RAW: A[i] $\longrightarrow$ A[i - 1]
  - WAR: A[i + 1] $\longrightarrow$ A[i]

Checking the Legality of a Schedule

Exercise: given the dependence polyhedra, check if a schedule is legal

$$\mathcal{D}_1 : \begin{bmatrix} 1 & 1 & 0 & 0 & -1 \\ 1 & -1 & 0 & 1 & -1 \\ 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & -1 & 1 & -1 \\ 0 & 1 & -1 & 0 & 1 \\ 1 & -1 & 1 & 0 & -1 \end{bmatrix} . \begin{pmatrix} eq \\ i_S \\ i'_S \\ n \\ 1 \end{pmatrix} \qquad \mathcal{D}_2 : \begin{bmatrix} 1 & 1 & 0 & 0 & -1 \\ 1 & -1 & 0 & 1 & -1 \\ 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & -1 & 1 & -1 \\ 0 & 1 & -1 & 0 & 1 \\ 1 & -1 & 1 & 0 & -1 \end{bmatrix} . \begin{pmatrix} eq \\ i_S \\ i'_S \\ n \\ 1 \end{pmatrix}$$

1. $\Theta = i$
2. $\Theta = -i$

Checking the Legality of a Schedule

Exercise: given the dependence polyhedra, check if a schedule is legal

$$\mathcal{D}_1 : \begin{bmatrix} 1 & 1 & 0 & 0 & -1 \\ 1 & -1 & 0 & 1 & -1 \\ 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & -1 & 1 & -1 \\ 0 & 1 & -1 & 0 & 1 \\ 1 & -1 & 1 & 0 & -1 \end{bmatrix} \cdot \begin{pmatrix} eq \\ i_S \\ i'_S \\ n \\ 1 \end{pmatrix} \qquad \mathcal{D}_2 : \begin{bmatrix} 1 & 1 & 0 & 0 & -1 \\ 1 & -1 & 0 & 1 & -1 \\ 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & -1 & 1 & -1 \\ 0 & 1 & -1 & 0 & 1 \\ 1 & -1 & 1 & 0 & -1 \end{bmatrix} \cdot \begin{pmatrix} eq \\ i_S \\ i'_S \\ n \\ 1 \end{pmatrix}$$

1. $\Theta = i$
2. $\Theta = -i$

- Solution: Check for the existence of pairs of instances in dependence in the dependence polyhedron when the timestamp are equals

A (Naive) Scheduling Approach

- Pick a schedule for the program statements
- Check if it respects all dependences

  This is called **filtering**

Limitations:

- How to use this in combination of an objective function?
- The density of legal 1-d affine schedules is low:

	matmult	locality	fir	h264	crout
$\vec{i}$-Bounds	$-1,1$	$-1,1$	$0,1$	$-1,1$	$-3,3$
c-Bounds	$-1,1$	$-1,1$	$0,3$	$0,4$	$-3,3$
#Sched.	1.9×10^4	5.9×10^4	1.2×10^7	1.8×10^8	2.6×10^{15}

$$\Downarrow$$

#Legal	6561	912	792	360	798

Objectives for a Good Scheduling Algorithm

- Build a legal schedule!
- Embed some properties in this legal schedule
  - latency: minimize the time between the first and last iteration
  - parallelism (for placement)
  - permutability (for tiling)
  - ...

A 2-step approach:

- Find the solution set of all legal affine schedules
- Find an ILP formulation for the objective function

The Precedence Constraint (Again!)

Precedence constraint adapted to 1-d schedules:

Definition (Causality condition for schedules)

Given $\mathcal{D}_{R,S}$, Θ^R and Θ^S are legal iff for each pair of instances in dependence:

$$\Theta^R(\vec{x_R}) < \Theta^S(\vec{x_S})$$

Equivalently: $\Delta_{R,S} = \Theta^S(\vec{x_S}) - \Theta^R(\vec{x_R}) - 1 \geq 0$

- All functions $\Delta_{R,S}$ which are non-negative over the dependence polyhedron represent legal schedules
- For the instances which are not in dependence, we don't care
- First step: how to get all non-negative functions over a polyhedron?

Affine Form of the Farkas Lemma

Lemma (Affine form of Farkas lemma)

Let $\mathcal{D}$ be a nonempty polyhedron defined by $A\vec{x}+\vec{b} \geq \vec{0}$. Then any affine function $f(\vec{x})$ is non-negative everywhere in $\mathcal{D}$ iff it is a positive affine combination:

$$f(\vec{x}) = \lambda_0 + \vec{\lambda}^T(A\vec{x}+\vec{b}), \textit{ with } \lambda_0 \geq 0 \textit{ and } \vec{\lambda} \geq \vec{0}$$

λ_0 and $\vec{\lambda}^T$ are called the Farkas multipliers.

- Intuition: a positive combination of some positive points is another positive point

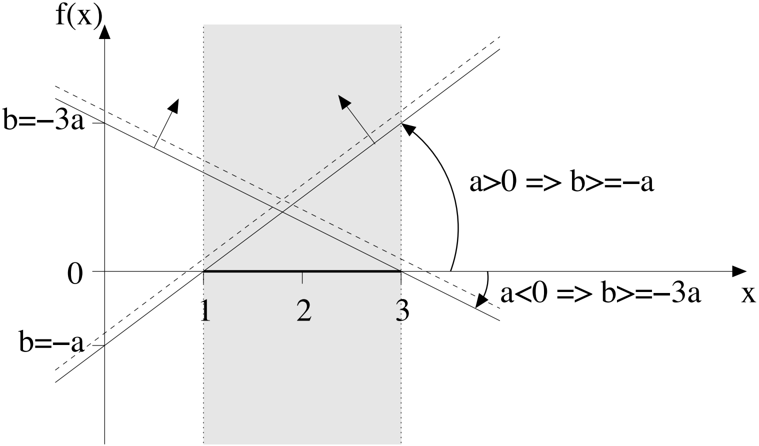
The Farkas Lemma: Example

- Function: $f(x) = ax + b$
- Domain of x: $\{1 \leq x \leq 3\} \rightarrow x - 1 \geq 0, \quad -x + 3 \geq 0$
- Farkas lemma: $f(x) \geq 0 \Leftrightarrow f(x) = \lambda_0 + \lambda_1(x-1) + \lambda_2(-x+3)$

The system to solve:

$$\left\{ \begin{array}{ccccccc} & & \lambda_1 & - & \lambda_2 & = & a \\ \lambda_0 & - & \lambda_1 & + & 3\lambda_2 & = & b \\ \lambda_0 & & & & & \geq & 0 \\ & & \lambda_1 & & & \geq & 0 \\ & & & & \lambda_2 & \geq & 0 \end{array} \right.$$

Pictured Example



(Courtesy of Cedric Bastoul's thesis!)

Example: Semantics Preservation (1-D)

Example: Semantics Preservation (1-D)

Property (Causality condition for schedules)

Given $R\delta S$, Θ^R and Θ^S are legal iff for each pair of instances in dependence:

$$\Theta^R(\vec{x}_R) < \Theta^S(\vec{x}_S)$$

Equivalently: $\Delta_{R,S} = \Theta^S(\vec{x}_S) - \Theta^R(\vec{x}_R) - 1 \geq 0$

Example: Semantics Preservation (1-D)

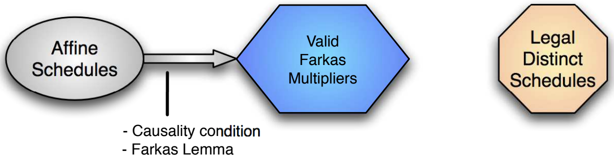
Lemma (Affine form of Farkas lemma)

Let $\mathcal{D}$ be a nonempty polyhedron defined by $A\vec{x}+\vec{b} \geq \vec{0}$. Then any affine function $f(\vec{x})$ is non-negative everywhere in $\mathcal{D}$ iff it is a positive affine combination:

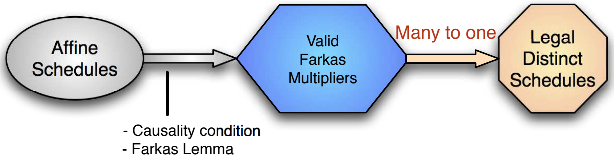
$$f(\vec{x}) = \lambda_0 + \vec{\lambda}^T(A\vec{x}+\vec{b}), \textit{ with } \lambda_0 \geq 0 \textit{ and } \vec{\lambda} \geq \vec{0}.$$

λ_0 and $\vec{\lambda}^T$ are called the Farkas multipliers.

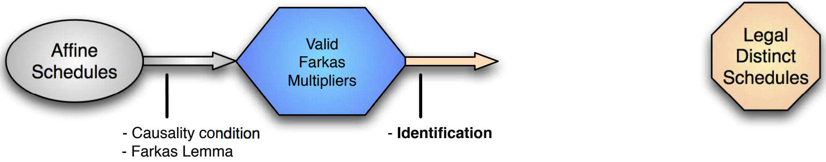
Example: Semantics Preservation (1-D)



Example: Semantics Preservation (1-D)



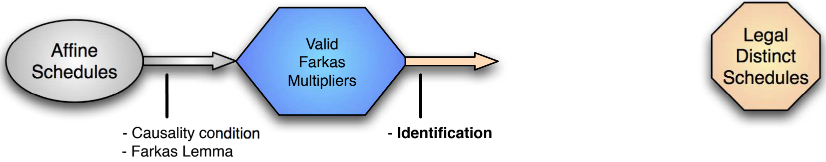
Example: Semantics Preservation (1-D)



$$\Theta^S(\vec{x_S}) - \Theta^R(\vec{x_R}) - 1 = \lambda_0 + \vec{\lambda}^T \left(D_{R,S} \begin{pmatrix} \vec{x_R} \\ \vec{x_S} \end{pmatrix} + \vec{d}_{R,S} \right) \geq 0$$

$$D_{R\delta S} \left\{ \begin{array}{rcl} \mathbf{i_R} & : & \lambda_{D_{1,1}} - \lambda_{D_{1,2}} + \lambda_{D_{1,3}} - \lambda_{D_{1,4}} \\ \mathbf{i_S} & : & -\lambda_{D_{1,1}} + \lambda_{D_{1,2}} + \lambda_{D_{1,5}} - \lambda_{D_{1,6}} \\ \mathbf{j_S} & : & \lambda_{D_{1,7}} - \lambda_{D_{1,8}} \\ \mathbf{n} & : & \lambda_{D_{1,4}} + \lambda_{D_{1,6}} + \lambda_{D_{1,8}} \\ \mathbf{1} & : & \lambda_{D_{1,0}} \end{array} \right.$$

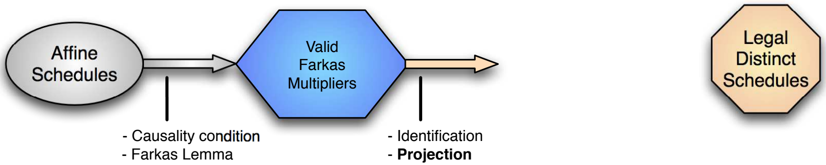
Example: Semantics Preservation (1-D)



$$\Theta^S(\vec{x}_S) - \Theta^R(\vec{x}_R) - 1 = \lambda_0 + \vec{\lambda}^T \left(D_{R,S} \begin{pmatrix} \vec{x}_R \\ \vec{x}_S \end{pmatrix} + \vec{d}_{R,S} \right) \geq 0$$

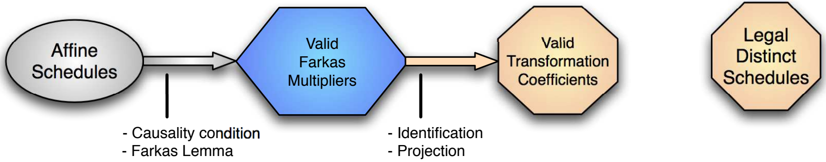
$$D_{R\delta S} \left\{ \begin{array}{lcrcl} \mathbf{i_R} & : & -t_{1_R} & = & \lambda_{D_{1,1}} - \lambda_{D_{1,2}} + \lambda_{D_{1,3}} - \lambda_{D_{1,4}} \\ \mathbf{i_S} & : & t_{1_S} & = & -\lambda_{D_{1,1}} + \lambda_{D_{1,2}} + \lambda_{D_{1,5}} - \lambda_{D_{1,6}} \\ \mathbf{j_S} & : & t_{2_S} & = & \lambda_{D_{1,7}} - \lambda_{D_{1,8}} \\ \mathbf{n} & : & t_{3_S} - t_{2_R} & = & \lambda_{D_{1,4}} + \lambda_{D_{1,6}} + \lambda_{D_{1,8}} \\ \mathbf{1} & : & t_{4_S} - t_{3_R} - 1 & = & \lambda_{D_{1,0}} \end{array} \right.$$

Example: Semantics Preservation (1-D)

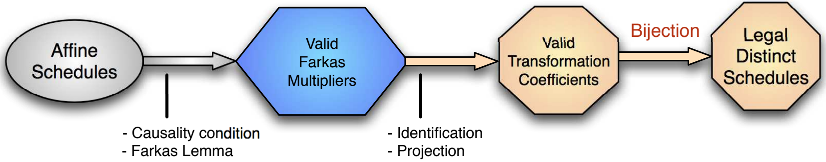


- Solve the constraint system
- Use (purpose-optimized) Fourier-Motzkin projection algorithm
  - Reduce redundancy
  - Detect implicit equalities

Example: Semantics Preservation (1-D)



Example: Semantics Preservation (1-D)



- One point in the space ⇔ one set of legal schedules w.r.t. the dependences
- These conditions for semantics preservation are not new! [Feautrier,92]

Scheduling Algorithm for Multiple Dependences

Algorithm

- Compute the schedule constraints for each dependence
- Intersect all sets of constraints
- Output is a convex solution set of all legal one-dimensional schedules

- Computation is fast, but requires eliminating variables in a system of inequalities: **projection**
- Can be computed as soon as the dependence polyhedra are known

Selecting a Good Schedule

Build a cost function to select a (good) schedule:

- Minimize latency: bound the execution time

  Bound the program execution / find bounded delay [Feautrier]
  Given $L = w_0 + \vec{u}.\vec{w}$, compute $min(\Theta(\vec{x}) - L)$ s.t. Θ is legal

- Exhibit coarse-grain parallelism

  Placement constraints [Lim/Lam]
  $\Theta^R(\vec{x}_R) = \Theta^S(\vec{x}_S)$ for all instances s.t. Θ is legal

- Many more possible...

Limitations of One-dimensional Schedules

- Not all programs have a legal one-dimensional schedule

Example

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

- Not all compositions of transformation are possible
  - Interchange in inner-loops
  - Fusion / distribution of inner-loops

Next week: the general case of multidimensional schedules