Polyhedral Transformation Framework

Louis-Noël Pouchet

CS & ECE
Colorado State University

February 23, 2020

PPoPP'20 Tutorial

Polyhedral Program Representation

Example: DGEMM

Example (dgemm)

```
   /* C := alpha*A*B + beta*C */
   for (i = 0; i < ni; i++)
    for (j = 0; j < nj; j++) {
S1:      C[i][j] *= beta;
      for (k = 0; k < nk; ++k)
S2:        C[i][j] += alpha * A[i][k] * B[k][j];
      }
```

This code has:

- imperfectly nested loops
- multiple statements
- parametric loop bounds

Granularity of Program Representation

DGEMM has:

- 3 loops
  - For loops in the code, while loops
  - Control-flow graph analysis
- 2 (syntactic) statements
  - Input source code?
  - Basic block?
  - ASM instructions?
- `S1` is executed $\texttt{ni} \times \texttt{nj}$ times
  - dynamic instances of the statement
  - Does not (necessarily) correspond to reality!

Some Observations

Reasoning at the loop/statement level:

- Some loop transformation may be very difficult to implement
  - How to fuse loops with different loop bounds?
  - How to permute triangular loops?
  - How to unroll-and-jam triangular loops?
  - How to apply time-tiling?
  - ...
- Statements may operate on the same array while being independent

Some Motivations for Polyhedral Transformations

- Known problem: scheduling of task graph
- Obvious limitations: task graph is not finite / size depends on problem / effective code generation almost impossible
- Alternative approach: use loop transformations
  - solve all above limitation
  - BUT the problem is to find a sequence that implements the order we want
  - AND also how to apply/compose them
- Desired features:
  - ability to reason at the instance level (as for task graph scheduling)
  - ability to easily apply/compose loop transformations

The Polyhedral Model

Motivating Example [1/2]

Example

```
for (i = 0; i <= 1; ++i)
  for (j = 0; j <= 2; ++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 is a different values for i, j associated to these 6 instances
- There is an order on them (the execution order)

A rough analogy: polyhedral compilation is about (statically) scheduling tasks, where tasks are statement <u>instances</u>, or operations

Polyhedral Representation of Programs

Static Control Parts

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

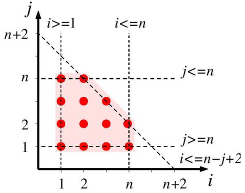
Polyhedral Representation of Programs

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] = ...
```

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



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
- 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];
}
```

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

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

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

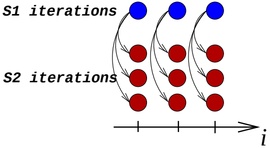
Polyhedral Representation of Programs

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;
}
```

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



Program Transformations

What Can Be Modeled?

Exact vs. approximate representation:

- Exact representation of iteration domains
  - Static control flow
  - Affine loop bounds (includes min/max/integer division)
  - Affine conditionals (conjunction/disjunction)

- Approximate representation of iteration domains
  - Use affine over-approximations, predicate statement executions
  - Full-function support

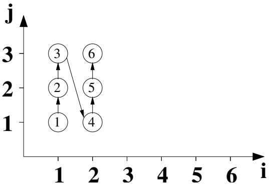
Key Intuition

- Programs are represented with geometric shapes
- Transforming a program is about modifying those shapes
  - rotation, skewing, stretching, ...
- But we need here to assume some order to scan points

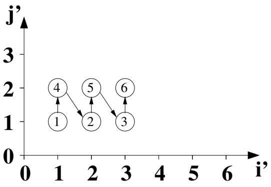
Affine Transformations

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

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

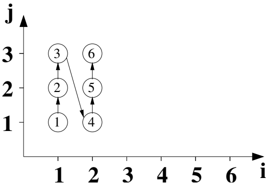
```
do i = 1, 2
  do j = 1, 3
     S(i,j)
```

```
do i' = 1, 3
  do j' = 1, 2
     S(i=j',j=i')
```

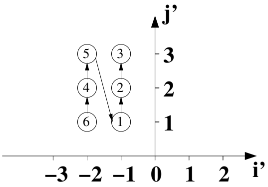
Affine Transformations

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

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

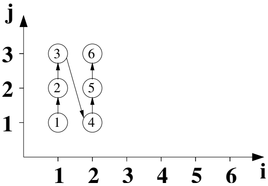
```
do i = 1, 2
  do j = 1, 3
     S(i,j)
```

```
do i' = -1, -2, -1
  do j' = 1, 3
     S(i=3-i',j=j')
```

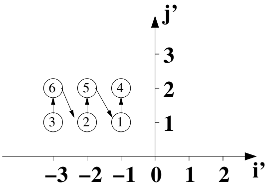
Affine Transformations

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

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

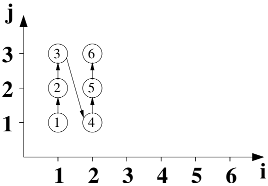
```
do i = 1, 2
  do j = 1, 3
    S(i,j)
```

```
do j' = -1, -3, -1
  do i' = 1, 2
    S(i=4-j',j=i')
```

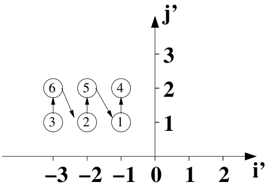
Affine Transformations

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
    S(i,j)
```

```
do j' = -1, -3, -1
  do i' = 1, 2
    S(i=4-j',j=i')
```

So, What is This Matrix?

- We know how to generate code for arbitrary matrices with integer coefficients
  - Arbitrary number of rows (but fixed number of columns)
  - Arbitrary value for the coefficients
- Through code generation, the number of dynamic instances is preserved
- **But this is not true for the transformed polyhedra!**

Some classification:

- The matrix is unimodular
- The matrix is full rank and invertible
- The matrix is full rank
- The matrix has only integral coefficients
- The matrix has rational coefficients

A Reverse History Perspective

1. CLooG: arbitrary matrix
2. Affine Mappings
3. Unimodular framework
4. SARE
5. SURE

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];

  }
```

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

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

```
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];

  }
```

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

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

```
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];

  }
```

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

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

```
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];
```

- **Less (linear) rows than loop depth $\rightarrow$ remaining dimensions are parallel**

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];

  }
```

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

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

```
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];
```

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

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];

  }
```

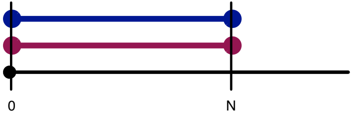
$$\Theta^{S1}.\vec{x}_{S1} = \begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix} \cdot \begin{pmatrix} i \\ j \\ n \\ 1 \end{pmatrix}$$

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

```
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

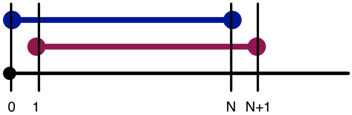
Fusion in the Polyhedral Model




```
for (i = 0; i <= N; ++i) {
  Blue(i);
  Red(i);
}
```

Perfectly aligned fusion

Fusion in the Polyhedral Model

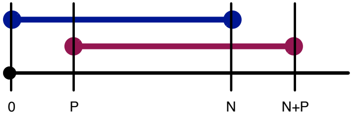



```
Blue(0);
for (i = 1; i <= N; ++i) {
  Blue(i);
  Red(i-1);
}
Red(N);
```

Fusion with shift of 1

Not all instances are fused

Fusion in the Polyhedral Model

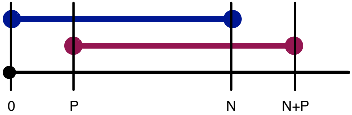



```
for (i = 0; i < P; ++i)
  Blue(i);
for (i = P; i <= N; ++i) {
  Blue(i);
  Red(i-P);
}
for (i = N+1; i <= N+P; ++i)
  Red(i-P);
```

Fusion with parametric shift of P

Automatic generation of prolog/epilog code

Fusion in the Polyhedral Model




```
for (i = 0; i < P; ++i)
  Blue(i);
for (i = P; i <= N; ++i) {
  Blue(i);
  Red(i-P);
}
for (i = N+1; i <= N+P; ++i)
  Red(i-P);
```

Many other transformations may be required to enable fusion: interchange, skewing, etc.

Scheduling Matrix

Definition (Affine multidimensional schedule)

Given a statement S, an affine schedule Θ^S of dimension m is an affine form on the d outer loop iterators $\vec{x}_S$ and the p global parameters $\vec{n}$.
$\Theta^S \in \mathbb{Z}^{m\times(d+p+1)}$ can be written as:

$$\Theta^S(\vec{x}_S) = \begin{pmatrix} \theta_{1,1} & \dots & \theta_{1,d+p+1} \\ \vdots & & \vdots \\ \theta_{m,1} & \dots & \theta_{m,d+p+1} \end{pmatrix} . \begin{pmatrix} \vec{x}_S \\ \vec{n} \\ 1 \end{pmatrix}$$

Θ^S_k denotes the kth row of Θ^S.

Definition (Bounded affine multidimensional schedule)

Θ^S is a bounded schedule if $\theta^S_{i,j} \in [x,y]$ with $x,y \in \mathbb{Z}$

Another Representation

One can separate coefficients of Θ into:

1. The iterator coefficients
2. The parameter coefficients
3. The constant coefficients

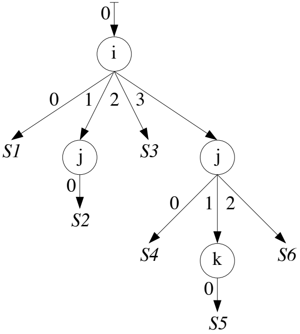
One can also enforce the schedule dimension to be 2d+1.

- A d-dimensional square matrix for the linear part
  - represents composition of interchange/skewing/slowing
- A $d \times n$ matrix for the parametric part
  - represents (parametric) shift
- A $d+1$ vector for the scalar offset
  - represents statement interleaving

- See URUK for instance

Computing the 2d+1 Identity Schedule

```
    do i=1, n
S1  | x = a(i,i)
    | do j=1, i-1
S2  | | x = x - a(i,j)**2
S3  | p(i) = 1.0/sqrt(x)
    | do j=i+1, n
S4  | | x = a(i,j)
    | | do k=1, i-1
S5  | | | x = x - a(j,k)*a(i,k)
S6  | | a(j,i) = x*p(i)
```



Transformation Catalogue [1/2]

Syntax	Effect
$\textsc{Unimodular}(P,\mathrm{U})$	$\forall S \in \mathcal{S}_{\mathrm{cop}} \mid P \sqsubseteq \beta^S, \mathrm{A}^S \leftarrow \mathrm{U}.\mathrm{A}^S;\ \Gamma^S \leftarrow \mathrm{U}.\Gamma^S$
$\textsc{Shift}(P,\mathrm{M})$	$\forall S \in \mathcal{S}_{\mathrm{cop}} \mid P \sqsubseteq \beta^S, \Gamma^S \leftarrow \Gamma^S + \mathrm{M}$
$\textsc{CutDom}(P,c)$	$\forall S \in \mathcal{S}_{\mathrm{cop}} \mid P \sqsubseteq \beta^S, \Lambda^S \leftarrow \mathsf{AddRow}\left(\Lambda^S, 0, c/\gcd(c_1,\ldots,c_{d^S+d^S_{\mathrm{lv}}+d_{\mathrm{gp}}+1})\right)$
$\textsc{Extend}(P,\ell,c)$	$\forall S \in \mathcal{S}_{\mathrm{cop}} \mid P \sqsubseteq \beta^S, \begin{cases} d^S \leftarrow d^S+1;\ \Lambda^S \leftarrow \mathsf{AddCol}(\Lambda^S, c, 0); \\ \beta^S \leftarrow \mathsf{AddRow}(\beta^S, \ell, 0); \\ \mathrm{A}^S \leftarrow \mathsf{AddRow}(\mathsf{AddCol}(\mathrm{A}^S, c, 0), \ell, \mathbf{1}_\ell); \\ \Gamma^S \leftarrow \mathsf{AddRow}(\Gamma^S, \ell, 0); \\ \forall(\mathtt{A},\mathrm{F}) \in \mathcal{L}^S_{\mathrm{hs}} \cup \mathcal{R}^S_{\mathrm{hs}}, \mathrm{F} \leftarrow \mathsf{AddRow}(\mathrm{F}, \ell, 0) \end{cases}$
$\textsc{AddLocalVar}(P)$	$\forall S \in \mathcal{S}_{\mathrm{cop}} \mid P \sqsubseteq \beta^S, d^S_{\mathrm{lv}} \leftarrow d^S_{\mathrm{lv}}+1;\ \Lambda^S \leftarrow \mathsf{AddCol}(\Lambda^S, d^S+1, 0);$ $\forall(\mathtt{A},\mathrm{F}) \in \mathcal{L}^S_{\mathrm{hs}} \cup \mathcal{R}^S_{\mathrm{hs}}, \mathrm{F} \leftarrow \mathsf{AddCol}(\mathrm{F}, d^S+1, 0)$
$\textsc{Privatize}(\mathtt{A},\ell)$	$\forall S \in \mathcal{S}_{\mathrm{cop}}, \forall(\mathtt{A},\mathrm{F}) \in \mathcal{L}^S_{\mathrm{hs}} \cup \mathcal{R}^S_{\mathrm{hs}}, \mathrm{F} \leftarrow \mathsf{AddRow}(\mathrm{F}, \ell, \mathbf{1}_\ell)$
$\textsc{Contract}(\mathtt{A},\ell)$	$\forall S \in \mathcal{S}_{\mathrm{cop}}, \forall(\mathtt{A},\mathrm{F}) \in \mathcal{L}^S_{\mathrm{hs}} \cup \mathcal{R}^S_{\mathrm{hs}}, \mathrm{F} \leftarrow \mathsf{RemRow}(\mathrm{F}, \ell)$
$\textsc{Fusion}(P,o)$	$b = \max\{\beta^S_{\dim(P)+1} \mid (P,o) \sqsubseteq \beta^S\} + 1$ $\mathsf{Move}((P,o+1),(P,o+1),b);\ \mathsf{Move}(P,(P,o+1),-1)$
$\textsc{Fission}(P,o,b)$	$\mathsf{Move}(P,(P,o,b),1);\ \mathsf{Move}((P,o+1),(P,o+1),-b)$
$\textsc{Motion}(P,T)$	$\text{if } \dim(P)+1 = \dim(T) \text{ then } b = \max\{\beta^S_{\dim(P)} \mid P \sqsubseteq \beta^S\} + 1 \text{ else } b = 1$ $\mathsf{Move}(\mathrm{pfx}(T,\dim(T)-1),T,b)$ $\forall S \in \mathcal{S}_{\mathrm{cop}} \mid P \sqsubseteq \beta^S, \beta^S \leftarrow \beta^S + T - \mathrm{pfx}(P,\dim(T))$ $\mathsf{Move}(P,P,-1)$

Transformation Catalogue [2/2]

Syntax	Effect	Comments
$\textsc{Interchange}(P,o)$	$\forall S \in \mathcal{S}_{\text{cop}} \mid P \sqsubseteq \beta^S,$ $\begin{cases} \mathrm{U} = \mathrm{I}_{d^S} - \mathbf{1}_{o,o} - \mathbf{1}_{o+1,o+1} + \mathbf{1}_{o,o+1} + \mathbf{1}_{o+1,o}; \\ \textsc{Unimodular}(\beta^S, \mathrm{U}) \end{cases}$	swap rows o and $o+1$
$\textsc{Skew}(P,\ell,c,s)$	$\forall S \in \mathcal{S}_{\text{cop}} \mid P \sqsubseteq \beta^S,$ $\begin{cases} \mathrm{U} = \mathrm{I}_{d^S} + s \cdot \mathbf{1}_{\ell,c}; \\ \textsc{Unimodular}(\beta^S, \mathrm{U}) \end{cases}$	add the skew factor
$\textsc{Reverse}(P,o)$	$\forall S \in \mathcal{S}_{\text{cop}} \mid P \sqsubseteq \beta^S,$ $\begin{cases} \mathrm{U} = \mathrm{I}_{d^S} - 2 \cdot \mathbf{1}_{o,o}; \\ \textsc{Unimodular}(\beta^S, \mathrm{U}) \end{cases}$	put a -1 in (o,o)
$\textsc{StripMine}(P,k)$	$\forall S \in \mathcal{S}_{\text{cop}} \mid P \sqsubseteq \beta^S,$ $\begin{cases} c = \dim(P); \\ \textsc{Extend}(\beta^S, c, c); \\ u = d^S + d^S_{\text{lv}} + d_{\text{gp}} + 1; \\ \textsc{CutDom}(\beta^S, -k \cdot \mathbf{1}_c + (\mathrm{A}^S_{c+1}, \Gamma^S_{c+1})); \\ \textsc{CutDom}(\beta^S, k \cdot \mathbf{1}_c - (\mathrm{A}^S_{c+1}, \Gamma^S_{c+1}) + (k-1)\mathbf{1}_u) \end{cases}$	insert intermediate loop constant column $k \cdot \mathbf{i}_c \leq \mathbf{i}_{c+1}$ $\mathbf{i}_{c+1} \leq k \cdot \mathbf{i}_c + k - 1$
$\textsc{Tile}(P,o,k_1,k_2)$	$\forall S \in \mathcal{S}_{\text{cop}} \mid (P,o) \sqsubseteq \beta^S,$ $\begin{cases} \textsc{StripMine}((P,o), k_2); \\ \textsc{StripMine}(P, k_1); \\ \textsc{Interchange}((P,0), \dim(P)) \end{cases}$	strip outer loop strip inner loop interchange

Some Final Observations

Some classical pitfalls

- The number of rows of Θ does not correspond to actual parallel levels
- Scalar rows vs. linear rows
- Linear independence
- Parametric shift for domains without parametric bound
- ...