Putting Polyhedral Loop Transformations to Work

Unified Model and Compiler Interface

Albert Cohen

with Cédric Bastoul, Sylvain Girbal, Saurabh Sharma, Olivier Temam

A3 Group

INRIA
ROCQUENCOURT

UNIVERSITÉ
PARIS-SUD 11

PRISM

Research Context

- Whole program optimization for peak performance
  - Uniprocessor
  - OpenMP
- Iterative, feedback-directed optimization
  1. Implement the *useful* transformations
  2. Choose the transformation *sequence and parameters*

Research Context

- Whole program optimization for peak performance
  - Uniprocessor
  - OpenMP
- Iterative, feedback-directed optimization
  1. Implement the *useful* transformations
  2. Choose the transformation *sequence and parameters*
- This talk: *promote the polytope model as a viable representation and transformation framework for semi-automatic program optimization and parallelization*

Example: Matrix Multiplication

Alpha EV67, dynamic analysis with Compaq's Alpha simulator

$\mathbf{95\%}$ of peak performance (Parello and Temam, SuperComputing'02)

Transformations	*Speedup*
Original `ikj` loop nest, Compaq f90 -O2 -unroll 1 -nopipeline	**1.00**
Compaq f90 -O5 + KAP	**3.37**
3D blocking for L1 and TLB	2.62
3D + *interchange for store queue* + *unrolling for ILP*	**3.71**
3D + int + unroll + *register blocking*	9.90
3D + int + unroll + reg block + *prefetch*	**10.37**
3D for L1 and L2 + *copy for TLB* + int + unroll + reg block + prefetch	12.75
3D + copy + int + unroll + reg block + prefetch + *low level opt*	**13.56**

Example of Composition of Transformations

```
for (i=0; i<1000; i++)
  for (j=0; j<m; j++)
    B[j] = A[i][j] + …
  for (j=0; j<n; j++)
    … = B[j] + …
```

$\xrightarrow{\textit{fuse}}$

```
for (i=0; i<1000; i++)
  for (j=0; j<max(m,n); j++)
    if (j<m)
      B[j] = A[i][j] + …
    if (j<n)
      … = B[j] + …
```

Example of Composition of Transformations

```
for (i=0; i<1000; i++)
  for (j=0; j<max(m,n); j++)
    if (j<m)
      B[j] = A[i][j] + …
    if (j<n)
      … = B[j] + …
```

$\xrightarrow{\textit{shift}}$

```
for (i=0; i<1000; i++)
  for (j=0; j<max(m,n+1); j++)
    if (j<m)
      B[j] = A[i][j] + …
    if (j>0 && j<=n)
      … = B[j-1] + …
```

Example of Composition of Transformations

```
for (i=0; i<1000; i++)
  for (j=0; j<max(m,n+1); j++)
    if (j<m)
      B[j] = A[i][j] + …
    if (j>0 && j<=n)
      … = B[j-1] + …
```

$\xrightarrow{\textit{strip-mine}}$

```
for (ii=0; ii<1000; ii+=10)
  for (i=ii; i<ii+10; i++)
    for (j=0; j<max(m,n+1); j++)
      if (j<m)
        B[j] = A[i][j] + …
      if (j>0 && j<=n)
        … = B[j-1] + …
```

Example of Composition of Transformations

```
for (ii=0; i<1000; i+=10)
  for (i=ii; i<ii+10; i++)
    for (j=0; j<max(m,n+1); j++)
      if (j<m)
        B[j] = A[i][j] + …
      if (j>0 && j<=n)
        … = B[j-1] + …
```

$\xrightarrow{\textit{prefetch}}$

```
for (ii=0; i<1000; i+=10)
  for (i=ii; i<ii+10; i++)
    for (j=0; j<max(m,n+1); j++)
      if (j<m)
        if (j%4==0)
          prefetch A[i+1][j]
        B[j] = A[i][j] + …
      if (j>0 && j<=n)
        … = B[j-1] + …
```

Some Problems With Syntax-Based Approaches

- Control overhead
  - Regenerate control structures after each transformation
  - Fixed transformation sequence
  - Non-local transformations

```
if (m<n && m%4==0)
  for (ii=0; i<1000; i+=10)
    for (i=ii; i<ii+10; i++)
      for (j=0; j+3<m; j+=4)
        prefetch A[i+1][j]
        B[0] = A[i][j] + …
        … = B[j-1] + …
        B[j] = A[i][j+1] + …
        … = B[j] + …
        B[j] = A[i][j+2] + …
        … = B[j+1] + …
        B[j] = A[i][j+3] + …
        … = B[j+2] + …
      for (j=m; j<n; j++)
        … = B[j-1] + …
else if (m<n && m%4==1)
  …
```

1. POLYHEDRAL REPRESENTATION

Unified Loop Nest Transformation Framework

- Operated by optimization *and* architecture *experts*

Unified Loop Nest Transformation Framework

- Operated by optimization *and* architecture *experts*
- Express any *composition* of analyses and transformations

Unified Loop Nest Transformation Framework

- Operated by optimization *and* architecture *experts*
- Express any *composition* of analyses and transformations
- Domain-specific, representation of *loop nests*

Unified Loop Nest Transformation Framework

- Operated by optimization *and* architecture *experts*
- Express any *composition* of analyses and transformations
- Domain-specific, representation of *loop nests*
- No *intermediate* translation to syntax-tree

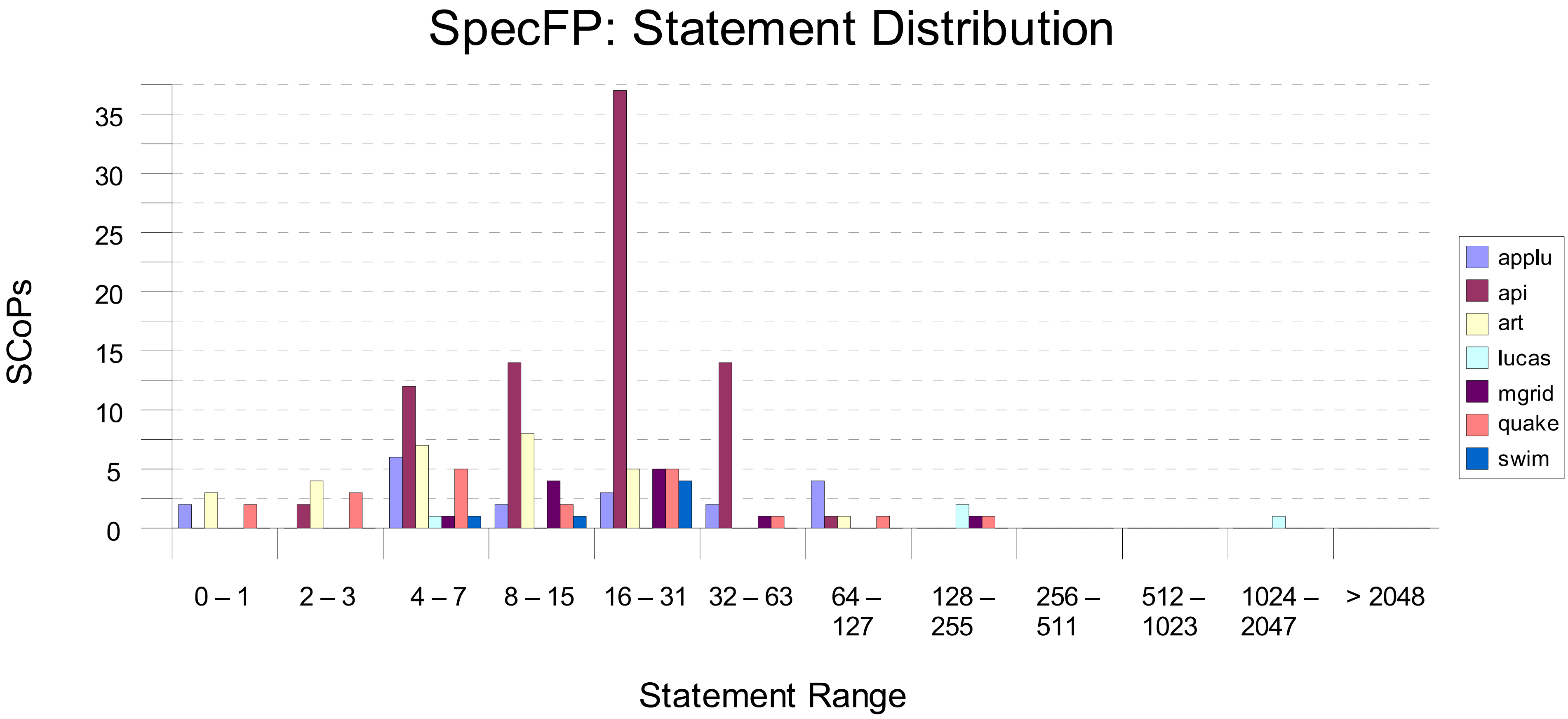
Static Control Parts (SCoPs)

```
     for (i=1; i<3; i++)
 ..........................................................................
 S1  | ...                                  | SCoP 1, one statement
 ..........................................................................
     | while (A[j]!=0)
 ..........................................................................
 S2  | | ...                                | SCoP 2, three statements
     | | for (k=0; k<j; k++)                |   parameters: i,j
     | | | if (j>=2)                        |   iterators: k
 S3  | | | | ...                            |
 S4  | | | ...                              |
 ..........................................................................
     | for (p=0; p<6; p++)                  | SCoP 3, two statements
 S5  | | ...                                |   iterators: p
 S6  | | ...                                |
```

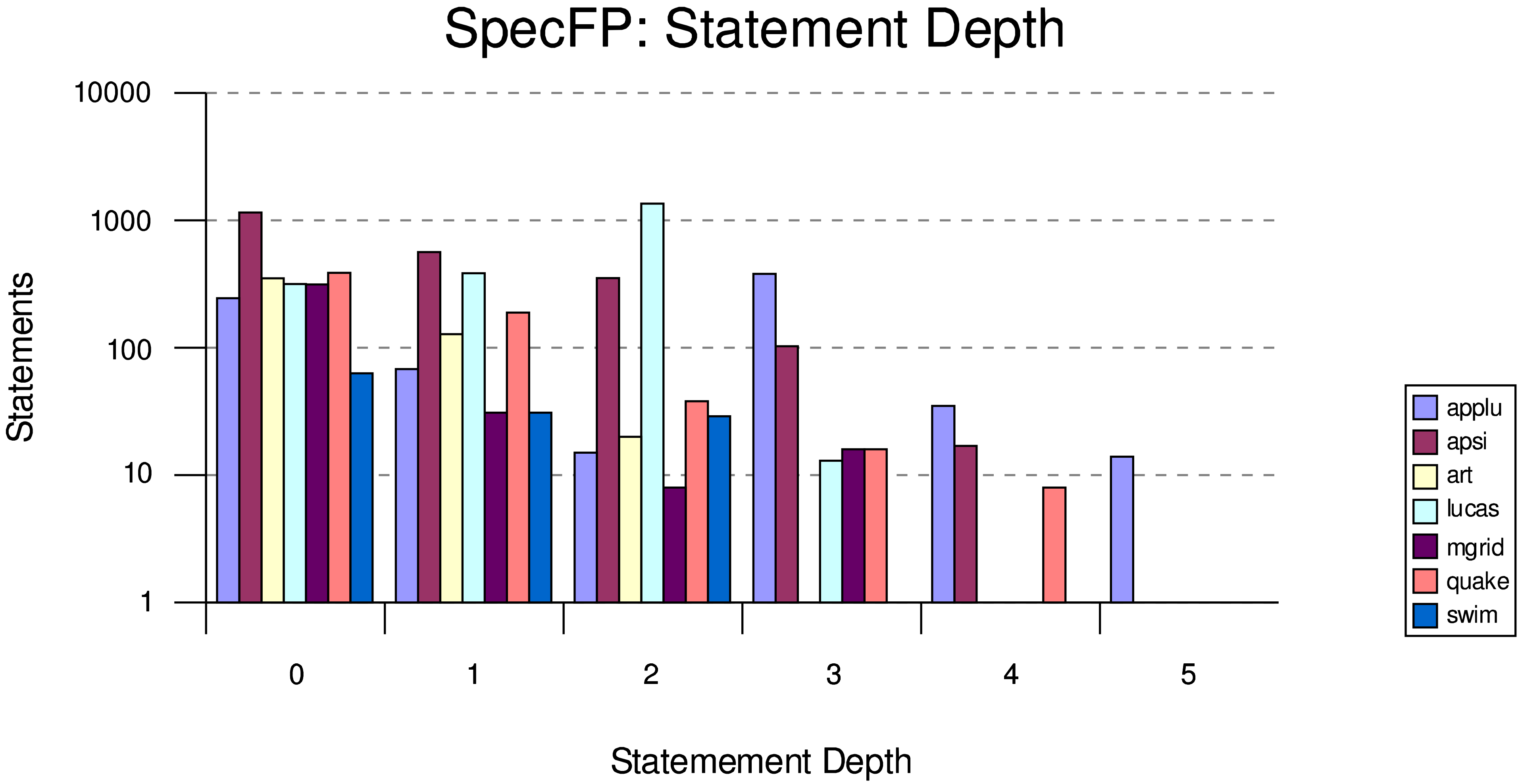
SCoP Coverage (SpecFP)

	SCoPs			Statements		Array References	
	All	Param.	`ifs`	All	in SCoPs	All	Affine
applu	19	15	1	757	84%	1245	100%
apsi	80	80	25	2192	84%	977	78%
art	28	27	4	499	69%	52	100%
lucas	4	4	2	2070	99%	411	40%
mgrid	12	12	2	369	100%	176	99%
quake	20	14	4	639	77%	218	100%
swim	6	6	1	123	100%	192	100%

SCoP Size (SpecFP)



SCoP Depth (SpecFP)



SCoP Polyhedral Representation

- Describes each statement *separately*
- Captures *control* and *array access* semantics

  Through *parameterized* affine (in)equalities

  1. A *domain*

     The bounds of the enclosing loops

  2. A *schedule*

     An affine function assigning logical dates to iterations

  3. A list of *access functions*

     To describe array references

Existing Polyhedral Representations

- A few facts

1. Polytopes are very expressive

   $\boldsymbol{AX \geq 0}$ suffices to characterize all executions of a statement (schedule, domain and memory accesses)

2. Affine schedules emerged in automatic parallelization
3. Affine schedules are not popular for optimization

   (too expensive, too restrictive, too general, non intuitive...)

Existing Polyhedral Representations

- A few facts
  1. Polytopes are very expressive

     $AX \geq 0$ suffices to characterize all executions of a statement (schedule, domain and memory accesses)
  2. Affine schedules emerged in automatic parallelization
  3. Affine schedules are not popular for optimization

     (too expensive, too restrictive, too general, non intuitive...)
- Some common biases
  - ? Schedules are only meant to describe parallelism
  - ? Only one-dimensional schedules are useful
  - ? Schedule and domains can be merged in one matrix

Existing Polyhedral Representations

- A few facts
  1. Polytopes are very expressive
     $AX \geq 0$ suffices to characterize all executions of a statement (schedule, domain and memory accesses)
  2. Affine schedules emerged in automatic parallelization
  3. Affine schedules are not popular for optimization
     (too expensive, too restrictive, too general, non intuitive...)
- Some common biases
  - ? Schedules are only meant to describe parallelism
  - ? Only one-dimensional schedules are useful
  - ? Schedule and domains can be merged in one matrix
- We use *separate* matrices and *full*-dimensional *sequential* schedules

Affine Schedule

- Dense, totally ordered (sequential) schedule
- Unimodular matrix for iteration ordering
- Matrix for parameterization and iteration shifting
- Vector for instruction scattering

Affine Schedule

- Dense, totally ordered (sequential) schedule
- Unimodular matrix for iteration ordering $(\mathbf{A})$
- Matrix for parameterization and iteration shifting $(\boldsymbol{\Gamma})$
- Vector for instruction scattering $(\boldsymbol{\beta})$

$$\boldsymbol{\theta}(\vec{\imath}, \vec{q}) = \begin{bmatrix} 0 & \cdots & 0 & 0 & \cdots & 0 & \beta_0 \\ A_{1,1} & \cdots & A_{1,d} & \Gamma_{1,1} & \cdots & \Gamma_{1,g} & \Gamma_{1,g+1} \\ 0 & \cdots & 0 & 0 & \cdots & 0 & \beta_1 \\ A_{2,1} & \cdots & A_{2,d} & \Gamma_{2,1} & \cdots & \Gamma_{2,g} & \Gamma_{2,g+1} \\ \vdots & \vdots & \vdots & 0 & \cdots & 0 & \vdots \\ A_{d,1} & \cdots & A_{d,d} & \Gamma_{d,1} & \cdots & \Gamma_{d,g} & \Gamma_{d,g+1} \\ 0 & \cdots & 0 & 0 & \cdots & 0 & \beta_d \end{bmatrix} \begin{bmatrix} i_1 \\ \vdots \\ i_d \\ q_1 \\ \vdots \\ q_g \\ 1 \end{bmatrix}$$

Domain and Access Functions

- Domain matrix

  Exact characterization of the valid iteration vectors

  Parameterized by symbolic constants

Domain and Access Functions

- Domain matrix

  Exact characterization of the valid iteration vectors

  Parameterized by symbolic constants

- Access function

  Iteration Vector $\mapsto$ (Array Name, Vector)

  Parameterized by symbolic constants

Polyhedral Representation Example

```
      for (i=0; i<m; i++)
$S_1$   | ...
        | for (j=5; j<n; j++)
$S_2$   |   | ...
$S_3$   |   | A[2*i][j+1] = ...
```

Access function
for A[2*i][j+1]

$$\left[\begin{array}{ccccc} \mathtt{i} & \mathtt{j} & \mathtt{m} & \mathtt{n} & \mathtt{1} \\ \hline 2 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 \end{array}\right]$$

2-dimensional domain of S_3
(with parameters m and n)

$$\left[\begin{array}{ccccc} \mathtt{i} & \mathtt{j} & \mathtt{m} & \mathtt{n} & \mathtt{1} \\ \hline 1 & 0 & 0 & 0 & 0 \\ -1 & 0 & 1 & 0 & -1 \\ 0 & 1 & 0 & 0 & -5 \\ 0 & -1 & 0 & 1 & -1 \end{array}\right] \geq 0$$

5-dimensional schedule for S_3
$(\mathtt{i},\mathtt{j}) \mapsto (p_0, p_1, p_2, p_3, p_4)$

$$\left[\begin{array}{cccccccccc} p_0 & p_1 & p_2 & p_3 & p_4 & \mathtt{i} & \mathtt{j} & \mathtt{m} & \mathtt{n} & \mathtt{1} \\ \hline 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & -1 \\ 0 & 0 & 0 & 1 & 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & -1 \end{array}\right] = 0$$

2. UNIFIED TRANSFORMATION MODEL

Primitives

Syntax	Prerequisites	Effect
RIGHTU$(\boldsymbol{S}, \mathsf{U})$	$\lvert\det(\mathsf{U})\rvert = 1$	$\mathbf{A}^{S} \leftarrow \mathbf{A}^{S}.\mathsf{U}$
SHIFT$(\boldsymbol{S}, \mathrm{M})$		$\Gamma^{S} \leftarrow \Gamma^{S} + \mathrm{M}$
FUSE$(\boldsymbol{P}, \boldsymbol{o})$		$b = \max\{\beta^{S}_{\dim(P)+1} \mid (P, o) \sqsubseteq \beta^{S}\} + 1;$ Move$((P, o+1), (P, o+1), b);$ Move$(P, (P, o+1), -1)$

Composition of Primitives

Syntax	Prerequisites	Effect
RightU(S, U)	$\lvert \det(\mathsf{U}) \rvert = 1$	$\mathbf{A}^S \leftarrow \mathbf{A}^S.\mathsf{U}$
Shift(S, M)		$\Gamma^S \leftarrow \Gamma^S + \mathrm{M}$
Fuse(P, o)		$b = \max\{\beta^S_{\dim(P)+1} \mid (P, o) \sqsubseteq \beta^S\} + 1;$ Move$((P, o + 1), (P, o + 1), b);$ Move$(P, (P, o + 1), -1)$
Tile(S, o, k) *Tiling*	$S \in \mathcal{S}$ $\wedge\, o < d^S$ $\wedge\, k > 0$	$S \leftarrow$ StripMine$(S, o, k);$ $S \leftarrow$ StripMine$(S, o + 2, k);$ $S \leftarrow$ Interchange$(S, o + 1)$

Transformation Language

- Script “generative” language
  - To produce the implementation of primitives
  - To compose primitives

Transformation Language

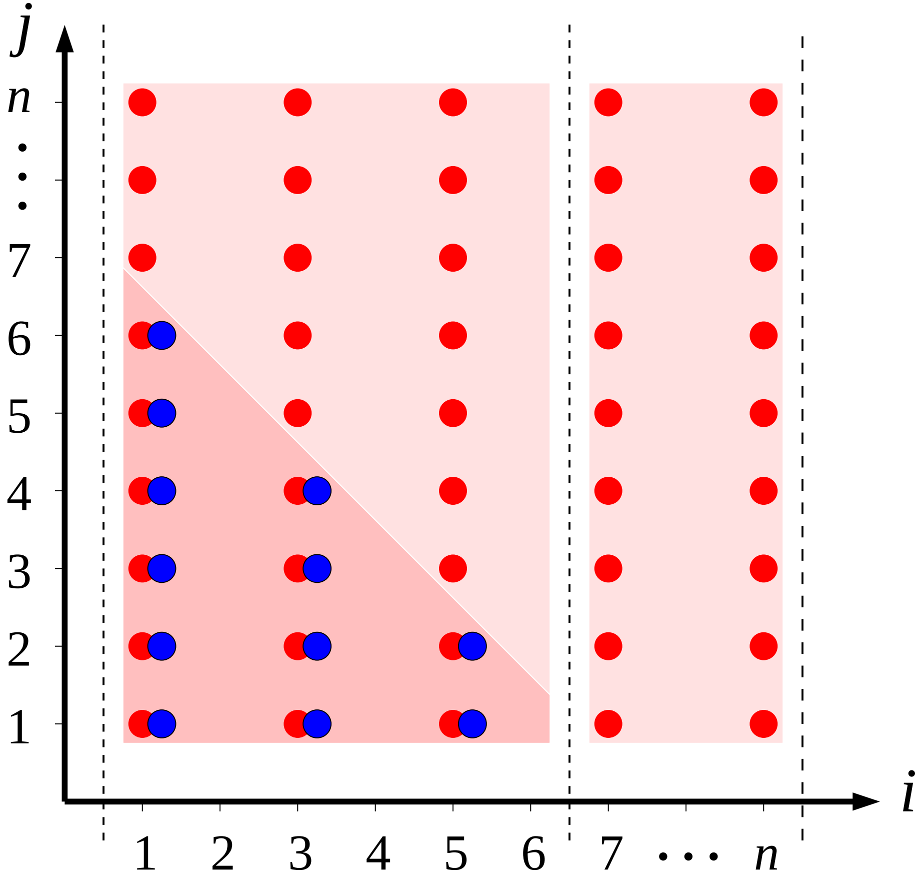
- Script “generative” language
  - To produce the implementation of primitives
  - To compose primitives
- Benefits
  - Regenerate the syntax tree *after the last transformation*
  - Few ordering constraints
  - Complex optimizations, e.g., forward array substitution
  - Combined transformations to reduce search space

    Example, “smart” register tiling: strip-mining + privatization for permutability + interchange + array contraction + register promotion

3. SOFTWARE TOOLS

Code Generation with CLooG

- Robust version of Quilleré and Rajopadhye’s algorithm
  - Parameterized unions of linearly bounded lattices
  - Depth recursion with direct optimization of conditionals
  - Tradeoff between code expansion and control overhead



```
for (i=1; i<=6; i+=2)
  for (j=1; j<=7-i; j++)
    S1; S2
  for (j=8-i; j<=n; j++)
    S1
for (i=7; i<=n; i+=2)
  for (j=1; j<=n; j++)
    S1
```

Implementation Within Open64/ORC

- *WRaP*: WHIRL Represented as Polyhedra
  - Syntax tree of *static control parts* → tree of polyhedral representations
  - Mapping polytopes to the syntax tree
    - From matrix columns to symbol table entries
    - From abstract arrays to symbol table entries
    - From abstract statements to statement nodes

Implementation Within Open64/ORC

- *WRaP*: WHIRL Represented as Polyhedra
  - Syntax tree of *static control parts* → tree of polyhedral representations
  - Mapping polytopes to the syntax tree
    - From matrix columns to symbol table entries
    - From abstract arrays to symbol table entries
    - From abstract statements to statement nodes

- Enables *whole program* optimization
  - Combined transformations of loops and syntactic expressions may be applied to the whole WRaP
  - Array regions, interprocedural analysis
  - Correctness and compatibility with non-affine sections

WRaP-IT: an Open64/ORC Interface-Tool

1. *Suspend* the WHIRL compilation flow after loop normalization, induction variables, and scalar optimizations

WRaP-IT: an Open64/ORC Interface-Tool

1. *Suspend* the WHIRL compilation flow after loop normalization, induction variables, and scalar optimizations

2. *W2P*: recognition of Static Control Parts (SCoPs)
    - Affine loop bounds, conditionals and array subscripts
    - Build polyhedral domains, sequential schedules and array accesses
    - Graceful degradation when all conditions are not met

WRaP-IT: an Open64/ORC Interface-Tool

1. *Suspend* the WHIRL compilation flow after loop normalization, induction variables, and scalar optimizations
2. *W2P*: recognition of Static Control Parts (SCoPs)
   - Affine loop bounds, conditionals and array subscripts
   - Build polyhedral domains, sequential schedules and array accesses
   - Graceful degradation when all conditions are not met
3. *URUK*: apply WRaP analyses and transformations

WRaP-IT: an Open64/ORC Interface-Tool

1. *Suspend* the WHIRL compilation flow after loop normalization, induction variables, and scalar optimizations
2. *W2P*: recognition of Static Control Parts (SCoPs)
   - Affine loop bounds, conditionals and array subscripts
   - Build polyhedral domains, sequential schedules and array accesses
   - Graceful degradation when all conditions are not met
3. *URUK*: apply WRaP analyses and transformations
4. *WLooG*: code generator (CLooG) with WHIRL output
   - Generate new loops, conditionals and variables
   - Move/duplicate the original statement nodes

WRaP-IT: an Open64/ORC Interface-Tool

1. *Suspend* the WHIRL compilation flow after loop normalization, induction variables, and scalar optimizations
2. *W2P*: recognition of Static Control Parts (SCoPs)
   - Affine loop bounds, conditionals and array subscripts
   - Build polyhedral domains, sequential schedules and array accesses
   - Graceful degradation when all conditions are not met
3. *URUK*: apply WRaP analyses and transformations
4. *WLooG*: code generator (CLooG) with WHIRL output
   - Generate new loops, conditionals and variables
   - Move/duplicate the original statement nodes
5. *Resume* the compilation flow, redoing scalar optimization

Some Related Works

- Codesign and synthesis of specialized coprocessors
  - MMAlpha
  - PICO

- Analysis and transformation frameworks
  - Omega/Petit
  - PIPS, Polaris, SUIF
  - Stratego

- Generative programming
  - ATLAS (BLAS library generator)
  - FFTW (FFT algorithm customization and optimization)
  - SPIRAL (signal-processing language, customization and optimization)

4. THANK YOU

HTTP://WWW-ROCQ.INRIA.FR/A3/WRAP-IT