When Iterative Optimization Meets the Polyhedral Model: One-Dimensional Date

Louis-Noël Pouchet

ALCHEMY, LRI - INRIA Futurs
Under the direction of A. Cohen & C. Bastoul

October 9, 2006

Problematic

- Emerging microprocessors introduce more parallelism / deeper memory hierarchies
- Optimizing compilers are mandatory to take advantage of processor architecture

But:

- Processor mechanism is too complex to be modeled entirely
- Cost models for optimization phases are too restrictive

⇒ How can we override these difficulties ?

Problematic

- Emerging microprocessors introduce more parallelism / deeper memory hierarchies
- Optimizing compilers are mandatory to take advantage of processor architecture

But:

- Processor mechanism is too complex to be modeled entirely
- Cost models for optimization phases are too restrictive

$\Rightarrow$ How can we override these difficulties ?

Problematic

- Emerging microprocessors introduce more parallelism / deeper memory hierarchies
- Optimizing compilers are mandatory to take advantage of processor architecture

But:

- Processor mechanism is too complex to be modeled entirely
- Cost models for optimization phases are too restrictive

$\Rightarrow$ How can we override these difficulties ?

Iterative Optimization

- Program transformations can result in unpredictable performance degradation (Bodin et al., 98)

⇒ Instead of statically decide if a transformation is better, run it on the target architecture

Pros:

- Much more accurate than static optimization
- Provide performance improvements
- Enable machine learning techniques to discover accurate transformation parameters (Stephenson et al., 03)
- Optimization space search can be feedback-directed

Iterative Optimization

- Program transformations can result in unpredictable performance degradation (Bodin et al., 98)

$\Rightarrow$ Instead of statically decide if a transformation is better, run it on the target architecture

Pros:

- Much more accurate than static optimization
- Provide performance improvements
- Enable machine learning techniques to discover accurate transformation parameters (Stephenson et al., 03)
- Optimization space search can be feedback-directed

Iterative Optimization

- Program transformations can result in unpredictable performance degradation (Bodin et al., 98)

$\Rightarrow$ Instead of statically decide if a transformation is better, run it on the target architecture

Pros:

- Much more accurate than static optimization
- Provide performance improvements
- Enable machine learning techniques to discover accurate transformation parameters (Stephenson et al., 03)
- Optimization space search can be feedback-directed

Drawbacks

Limitations:

- The set of combination of transformations is extremely large
- Only a subset of them respects the program semantic

$\rightarrow$ Only a (very small) subset of transformation sequences is actually tested

$\rightarrow$ The search space is too restrictive or too large due to the bottleneck of the legality condition

$\Rightarrow$ Can we improve the search space construction : model all sequences of transformations, and model only legal ones ?

Drawbacks

Limitations:

- The set of combination of transformations is extremely large
- Only a subset of them respects the program semantic

$\longrightarrow$ Only a (very small) subset of transformation sequences is actually tested

$\longrightarrow$ The search space is too restrictive or too large due to the bottleneck of the legality condition

$\Rightarrow$ Can we improve the search space construction : model all sequences of transformations, and model only legal ones ?

Drawbacks

Limitations:

- The set of combination of transformations is extremely large
- Only a subset of them respects the program semantic

$\rightarrow$ Only a (very small) subset of transformation sequences is actually tested

$\rightarrow$ The search space is too restrictive or too large due to the bottleneck of the legality condition

$\Rightarrow$ Can we improve the search space construction : model all sequences of transformations, and model only legal ones ?

Drawbacks

Limitations:

- The set of combination of transformations is extremely large
- Only a subset of them respects the program semantic

$\longrightarrow$ Only a (very small) subset of transformation sequences is actually tested

$\longrightarrow$ The search space is too restrictive or too large due to the bottleneck of the legality condition

$\Rightarrow$ Can we improve the search space construction : model all sequences of transformations, and model only legal ones ?

Iterative Optimization in the Polyhedral Model

- Focus on a subclass of programs: Static Control Parts
- Use a polyhedral abstraction to represent program information
- Use iterative optimization techniques in the constructed space

$\longrightarrow$ In the polyhedral model (Feautrier, 92):

- Composition of transformations are easily expressed
- Transformation legality is easily checked
- Natural expression of parallelism

Iterative Optimization in the Polyhedral Model

- Focus on a subclass of programs: Static Control Parts
- Use a polyhedral abstraction to represent program information
- Use iterative optimization techniques in the constructed space

→ In the polyhedral model (Feautrier, 92):

- Composition of transformations are easily expressed
- Transformation legality is easily checked
- Natural expression of parallelism

Iterative Optimization in the Polyhedral Model

- Focus on a subclass of programs: Static Control Parts
- Use a polyhedral abstraction to represent program information
- Use iterative optimization techniques in the constructed space

→ In the polyhedral model (Feautrier, 92):

- Composition of transformations are easily expressed
- Transformation legality is easily checked
- Natural expression of parallelism

Iterative Optimization in the Polyhedral Model

- Focus on a subclass of programs: Static Control Parts
- Use a polyhedral abstraction to represent program information
- Use iterative optimization techniques in the constructed space

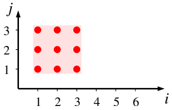
→ In the polyhedral model (Feautrier, 92):

- Composition of transformations are easily expressed
- Transformation legality is easily checked
- Natural expression of parallelism

The Polyhedral Model

```
do i = 1, 3
|  do j = 1, 3
|  |  A(i+j) = ...
```

1 Analysis: from code to model



2 Transformation in the model

Here : $\theta\binom{i}{j} = t = i + j$

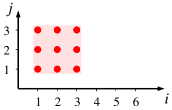
3 Code generation :
from model to code

```
do t = 2, 6
|  do i = max(1,t-3), min(t-1,3)
|  |  A(t) = ...
```

The Polyhedral Model

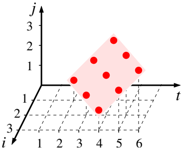
```
do i = 1, 3
| do j = 1, 3
| | A(i+j) = ...
```

1 Analysis: from code to model



2 Transformation in the model

Here : $\theta\binom{i}{j} = t = i + j$



3 Code generation : from model to code

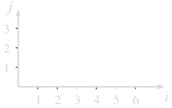
```
do t = 2, 6
| do i = max(1,t-3), min(t-1,3)
| | A(t) = ...
```

The Polyhedral Model

```
do i = 1, 3
|  do j = 1, 3
|  |  A(i+j) = ...
```

1 Analysis: from code to model

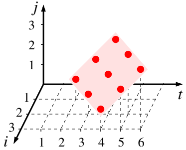
⇓



2 Transformation in the model

Here : $\theta\binom{i}{j} = t = i + j$

⇓



3 Code generation :
from model to code

⇓

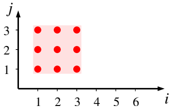
```
do t = 2, 6
|  do i = max(1,t-3), min(t-1,3)
|  |  A(t) = ...
```

The Polyhedral Model

```
do i = 1, 3
|  do j = 1, 3
|  |  A(i+j) = ...
```

1 Analysis: from code to model

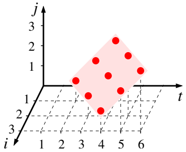
⇓



2 Transformation in the model

Here : $\theta\binom{i}{j} = t = i + j$

⇓



3 Code generation : from model to code

⇓

```
do t = 2, 6
|  do i = max(1,t-3), min(t-1,3)
|  |  A(t) = ...
```

A First Example

matvect

```
    do i = 0, n
R   |   s(i) = 0
    |   do j = 0, n
S   |   |   s(i) = s(i) + a(i,j) * x(j)
    |   end do
    end do
```

Iteration domain of R:

- *iteration vector* $\vec{x}_R = (i)$
- $\mathcal{D}_R : \{i \mid 0 \leq i \leq n\}$
- $\mathcal{D}_R : \begin{bmatrix} 1 \\ -1 \end{bmatrix} . (i) + \begin{pmatrix} 0 \\ n \end{pmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ -1 & 1 & 0 \end{bmatrix} . \begin{pmatrix} i \\ n \\ 1 \end{pmatrix} \geq \vec{0}$

A First Example

matvect

```
    do i = 0, n
R   |   s(i) = 0
    |   do j = 0, n
S   |   |   s(i) = s(i) + a(i,j) * x(j)
    |   end do
    end do
```

Iteration domain of R:

- *iteration vector* $\vec{x}_R = (i)$
- $\mathcal{D}_R : \{i \mid 0 \le i \le n\}$
- $\mathcal{D}_R : \begin{bmatrix} 1 \\ -1 \end{bmatrix} . (i) + \begin{pmatrix} 0 \\ n \end{pmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ -1 & 1 & 0 \end{bmatrix} . \begin{pmatrix} i \\ n \\ 1 \end{pmatrix} \ge \vec{0}$

A First Example

matvect

```
     do i = 0, n
R    |    s(i) = 0
     |    do j = 0, n
S    |    |    s(i) = s(i) + a(i,j) * x(j)
     |    end do
     end do
```

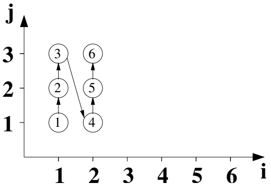
Iteration domain of S:

- *iteration vector* $\vec{x}_S = \binom{i}{j}$
- $\mathcal{D}_S : \{i, j \mid 0 \leq i \leq n,\ 0 \leq j \leq n, \}$
- $\mathcal{D}_S : \begin{bmatrix} 1 & 0 & 0 & 0 \\ -1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & -1 & 1 & 0 \end{bmatrix} \cdot \begin{pmatrix} i \\ j \\ n \\ 1 \end{pmatrix} \geq \vec{0}$

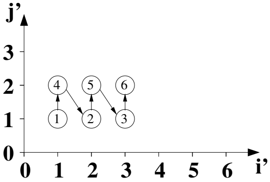
Expressing Transformations

Interchange Transformation

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



$\Longrightarrow$



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

transformation function

$\vec{y} = T_1 \vec{x}$

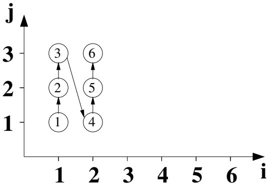
```
do i = 1, 2
  do j = 1, 3
```

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

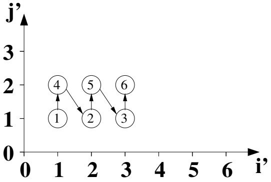
Expressing Transformations

Interchange Transformation

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



$\Longrightarrow$



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

transformation function

$$\vec{y} = T_1 \vec{x}$$

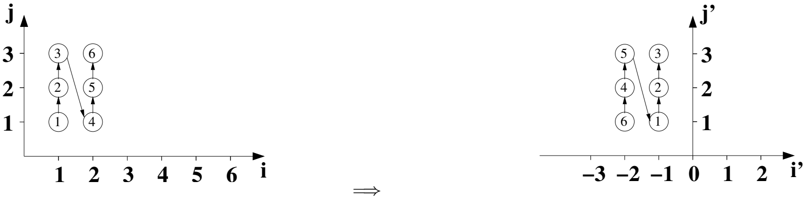
```
do i = 1, 2
  do j = 1, 3
```

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

Expressing Transformations

Reversal Transformation

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



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

transformation function

$\vec{y} = T_2 \vec{x}$

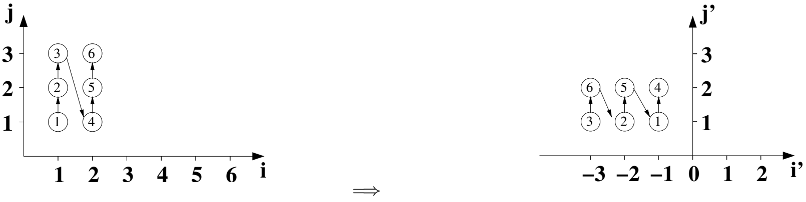
```
do i = 1, 2
  do j = 1, 3
```

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

Expressing Transformations

Compound Transformation

The transformation matrix is the composition of an interchange and reversal



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

transformation function

$\vec{y} = T\vec{x} = T_1 T_2 \vec{x}$

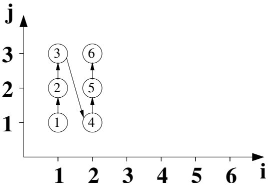
```
do i = 1, 2
  do j = 1, 3
```

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

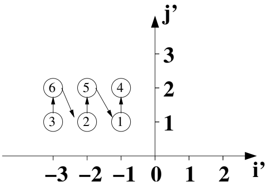
Expressing Transformations

Compound 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
$\vec{y}=T\vec{x}=T_1T_2\vec{x}$

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

(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 a Program

Definition (Schedule)

A schedule of a program is a function which associates a timestamp to each instance of each instruction. It can be written, for a statement S (T is a constant matrix):

$$\theta_S(\vec{x_S}) = T \begin{pmatrix} \vec{x_S} \\ \vec{n} \\ 1 \end{pmatrix}$$

Example:

$\theta_R(\vec{x}_R) = \left[\begin{array}{c} 1 \end{array}\right] . (i)$

$\theta_S(\vec{x}_S) = \left[\begin{array}{cc} 1 & 0 \\ 0 & 1 \end{array}\right] . \binom{i}{j}$

Is the original lexicographic order for R and S.

Objectives

- Focus on one-dimensional schedules (T is a constant row matrix)
- Build the set of all *legal* program versions (i.e. which respects all the data dependence of the program)

$\longrightarrow$ Perform an exact dependence analysis
$\longrightarrow$ Build the set of all possible values of T

$\Rightarrow$ The resulting space represents all the distinct possible ways to **legally reschedule** the program, using arbitrarily complex sequence of transformations.

Objectives

- Focus on one-dimensional schedules (T is a constant row matrix)
- Build the set of all *legal* program versions (i.e. which respects all the data dependence of the program)

$\longrightarrow$ Perform an exact dependence analysis
$\longrightarrow$ Build the set of all possible values of T

$\Rightarrow$ The resulting space represents all the distinct possible ways to **legally reschedule** the program, using arbitrarily complex sequence of transformations.

Objectives

- Focus on one-dimensional schedules (T is a constant row matrix)
- Build the set of all *legal* program versions (i.e. which respects all the data dependence of the program)

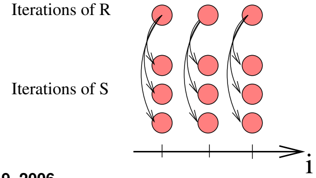
$\longrightarrow$ Perform an exact dependence analysis
$\longrightarrow$ Build the set of all possible values of T

$\Rightarrow$ The resulting space represents all the distinct possible ways to **legally reschedule** the program, using arbitrarily complex sequence of transformations.

Dependence Expression

- Need to represent the *exact* set of instances in dependence
- Exact computation made possible thanks to the SCoP and Static reference assumptions (Bastoul, 04)
- Use a subset of the Cartesian product of iteration domains:

```
    do i = 1, 3
R   |  s(i) = 0
    |  do j = 1, 3
S   |  |  s(i) = s(i) + a(i,j) * x(j)
```



$$\mathcal{D}_{R\delta S}:\ \left[\begin{array}{ccccc} 1 & -1 & 0 & 0 & 0 \\ \hline 1 & 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 1 & 0 \end{array}\right] \cdot \begin{pmatrix} i_R \\ i_S \\ j_S \\ n \\ 1 \end{pmatrix} \begin{array}{c} = 0 \\ \hline \geq \vec{0} \end{array}$$

Formal Definition [1/2]

Assuming $R\delta S$, $\mathcal{D}_{R\delta S}$ is the exact set of instances of R and S where the dependence exists.

A schedule is **legal** iff, $\forall \vec{x_R} \times \vec{x_S} \in \mathcal{D}_{R\delta S}$, $\theta_R(\vec{x_R}) < \theta_S(\vec{x_S})$.

Legal Schedule

$\Rightarrow$ Assuming $R\delta S$, $\theta_R(\vec{x_R})$ and $\theta_S(\vec{x_S})$ are legal iff:

$$\Delta_{R,S} = \theta_S(\vec{x_S}) - \theta_R(\vec{x_R}) - 1$$

Is non-negative for each point in $\mathcal{D}_{R\delta S}$.

Formal Definition [2/2]

$\longrightarrow$ We can express the legality condition as a set of affine non-negative functions over $\mathcal{D}_{R\delta S}$

Lemma (Affine form of Farkas lemma)

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

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

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

Formal Definition [2/2]

$\longrightarrow$ We can express the legality condition as a set of affine non-negative functions over $\mathcal{D}_{R\delta S}$

Lemma (Affine form of Farkas lemma)

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

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

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

Formal Definition [2/2]

$\longrightarrow$ We can express the legality condition as a set of affine non-negative functions over $\mathcal{D}_{R\delta S}$

Lemma (Affine form of Farkas lemma)

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

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

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

$\Rightarrow$ We can express the set of affine, non-negative functions over $\mathcal{D}_{R\delta S}$

Formal Definition [2/2]

Lemma (Affine form of Farkas lemma)

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

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

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

$\Rightarrow$ We just need to equate the coefficients:

$$\theta_S(\vec{x_S}) - \theta_R(\vec{x_R}) - 1 = \lambda_0 + \vec{\lambda}^T \left(\mathcal{D}_{R\delta S} \begin{pmatrix} \vec{x_R} \\ \vec{x_S} \end{pmatrix} + \vec{d}_{R\delta S} \right) \geq 0$$

An example

```
        do i = 1, 3
R     |   s(i) = 0
      |   do j = 1, 3
S     |   |   s(i) = s(i) + a(i,j) * x(j)
```

The two prototype affine schedules for R and S are:

$$\begin{aligned} \theta_R(\vec{x}_R) &= t_{1_R}.i_R + t_{2_R}.n + t_{3_R}.1 \\ \theta_S(\vec{x}_S) &= t_{1_S}.i_S + t_{2_S}.j_S + t_{3_S}.n + t_{4_S}.1 \end{aligned}$$

We get the following system for $R\delta S$:

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

$\longrightarrow$ We need to solve this system, to get $\mathcal{D}_t^{R\delta S}$.

An example

```
     do i = 1, 3
R  |   s(i) = 0
   |   do j = 1, 3
S  |   |  s(i) = s(i) + a(i,j) * x(j)
```

The two prototype affine schedules for R and S are:

$$\begin{array}{rcl} \theta_R(\vec{x}_R) & = & t_{1_R}.i_R + t_{2_R}.n + t_{3_R}.1 \\ \theta_S(\vec{x}_S) & = & t_{1_S}.i_S + t_{2_S}.j_S + t_{3_S}.n + t_{4_S}.1 \end{array}$$

We get the following system for $R\delta S$:

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

$\longrightarrow$ We need to solve this system, to get $\mathcal{D}_t^{R\delta S}$.

Construction Algorithm

- Need to build the intersection of all constraints obtained for each dependence, so for *k* dependences:

$$\mathcal{D}_t = \bigcap_k \mathcal{D}_t^k$$

- Need to bound the space, since the set of possible transformations can be infinite

$\Rightarrow$ To each (integral) point in $\mathcal{D}_t$ corresponds a different version of the original program where the semantic is preserved.

Construction Algorithm

- Need to build the intersection of all constraints obtained for each dependence, so for k dependences:

$$\mathcal{D}_t = \bigcap_k \mathcal{D}_t^k$$

- Need to bound the space, since the set of possible transformations can be infinite

$\Rightarrow$ To each (integral) point in $\mathcal{D}_t$ corresponds a different version of the original program where the semantic is preserved.

Construction Algorithm

- Need to build the intersection of all constraints obtained for each dependence, so for k dependences:

$$\mathcal{D}_t = \bigcap_k \mathcal{D}_t^k$$

- Need to bound the space, since the set of possible transformations can be infinite

$\Rightarrow$ To each (integral) point in $\mathcal{D}_t$ corresponds a different version of the original program where the semantic is preserved.

Discussions

- Expression of the set of **all legal, arbitrarily long sequences of transformation** (reversal, skewing, interchange, peeling, shifting, fusion, distribution)
- Multiple orders of magnitude reduction in the size of the search space compared to state-of-the-art techniques
- On small kernels, the search space is small enough to be exhaustively computed, yielding a method to find **The best transformation** within the model

Benchmark	#Dep	#St	Bounds	#Sched	#Legal	Time
matvect	5	2	$-1, 1$	3^7	129	0.024
locality	2	2	$-1, 1$	3^{10}	6561	0.022
matmul	7	2	$-1, 1$	3^9	912	0.029
gauss	18	2	$-1, 1$	3^{10}	506	0.047
crout	26	4	$-3, 3$	7^{17}	798	0.046

Discussions

- Expression of the set of **all legal, arbitrarily long sequences of transformation** (reversal, skewing, interchange, peeling, shifting, fusion, distribution)
- Multiple orders of magnitude reduction in the size of the search space compared to state-of-the-art techniques
- On small kernels, the search space is small enough to be exhaustively computed, yielding a method to find **The best transformation** within the model

Benchmark	#Dep	#St	Bounds	#Sched	#Legal	Time
`matvect`	5	2	$-1, 1$	3^7	129	0.024
`locality`	2	2	$-1, 1$	3^{10}	6561	0.022
`matmul`	7	2	$-1, 1$	3^9	912	0.029
`gauss`	18	2	$-1, 1$	3^{10}	506	0.047
`crout`	26	4	$-3, 3$	7^{17}	798	0.046

Discussions

- Expression of the set of **all legal, arbitrarily long sequences of transformation** (reversal, skewing, interchange, peeling, shifting, fusion, distribution)
- Multiple orders of magnitude reduction in the size of the search space compared to state-of-the-art techniques
- On small kernels, the search space is small enough to be exhaustively computed, yielding a method to find **The best transformation** within the model

Benchmark	#Dep	#St	Bounds	#Sched	#Legal	Time
matvect	5	2	$-1, 1$	3^7	129	0.024
locality	2	2	$-1, 1$	3^{10}	6561	0.022
matmul	7	2	$-1, 1$	3^9	912	0.029
gauss	18	2	$-1, 1$	3^{10}	506	0.047
crout	26	4	$-3, 3$	7^{17}	798	0.046

Performance Distribution [1/2]

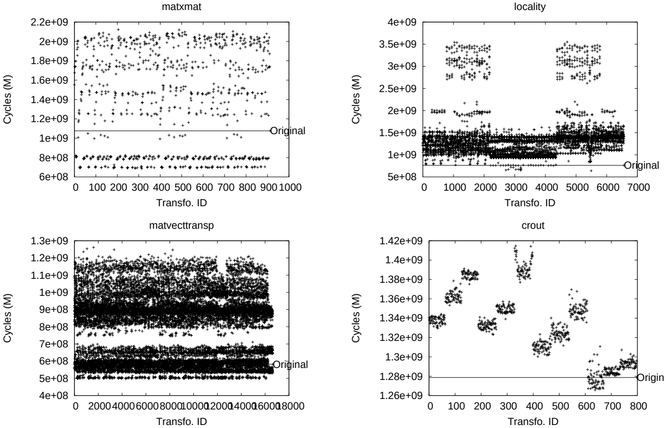


Figure: Performance distribution for `matmul`, `locality`, `mvt` and `crout`

Performance Distribution [2/2]

- Regularities are observable
- Exhaustive scan may achievable on (very) small kernels
- High peak performance discovered thanks to optimization enabling
- The best transformation depends on the compiler, the target architecture, and even the compiler options

Benchmark	Compiler	Options	Parameters	#Improved	ID best	Speedup
h264	PathCC	-Ofast	none	11	352	36.1%
h264	GCC	-O2	none	19	234	13.3%
h264	GCC	-O3	none	26	250	25.0%
h264	ICC	-O2	none	27	290	12.9%
h264	ICC	-fast	none	0	N/A	0%
MVT	PathCC	-Ofast	N=2000	5652	4934	27.4%
MVT	GCC	-O2	N=2000	3526	13301	18.0%
MVT	GCC	-O3	N=2000	3601	13320	21.2%
MVT	ICC	-O2	N=2000	5826	14093	24.0%
MVT	ICC	-fast	N=2000	5966	4879	29.1%
matmul	PathCC	-Ofast	N=250	402	283	308.1%
matmul	GCC	-O2	N=250	318	284	38.6%
matmul	GCC	-O3	N=250	345	270	49.0%
matmul	ICC	-O2	N=250	390	311	56.6%
matmul	ICC	-fast	N=250	318	641	645.4%

Exhaustive vs Heuristic Scan

Propose a decoupling heuristic:

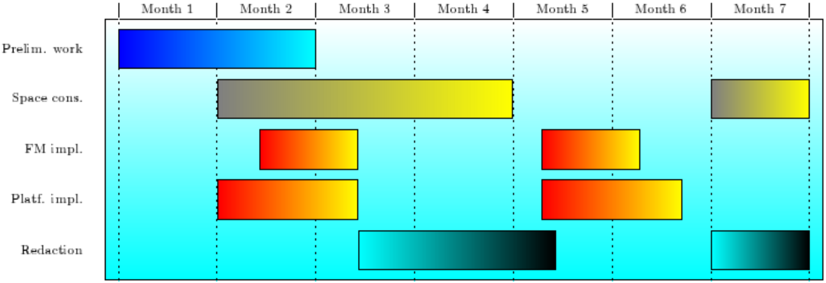
- The general “form” of the schedule is embedded in the iterator coefficients
- Parameters and constant coefficients can be seen as a refinement

→ On some distributions a random heuristic may converge faster

Figure: Heuristic convergence

Benchmark	#Schedules	Heuristic.	#Runs	%Speedup
`locality`	6561	Rand DH	125 123	96.1% 98.3%
`matmul`	912	Rand DH	170 170	99.9% 99.8%
`mvt`	16641	Rand DH	30 31	93.3% 99.0%

What, When, with Who ?



- Constant talks with Nicolas Vasilache (PhD student)
- Advised and oriented by Cedric Bastoul
- Theoretical fruitful discussions with Albert Cohen

Scientific Contribution

- New approach of the search space for iterative optimization
- Mathematically well founded algorithm for the construction of the *legal* transformation space in the polyhedral model
- Better formulation of the Fourier-Motzkin algorithm

- First exhaustive exploration of the performance space in the polyhedral model, for one-dimensional schedules
- Usual mathematical models sub-optimality brought to light
- Many observations on the performance space distribution

Scientific Contribution

- New approach of the search space for iterative optimization
- Mathematically well founded algorithm for the construction of the *legal* transformation space in the polyhedral model
- Better formulation of the Fourier-Motzkin algorithm

- First exhaustive exploration of the performance space in the polyhedral model, for one-dimensional schedules
- Usual mathematical models sub-optimality brought to light
- Many observations on the performance space distribution

Ongoing and Future Work

Ongoing research:

- Expression of equivalence between parts of the search space
- Simulation of multidimensional schedules with correction / completion
- New exploration heuristics
- Feedback directed exploration

PhD objectives:

- Extend the method to multidimensional schedules
- Develop exploration methods for the search space (statistic, machine learning, . . .)

Conclusion

- Very exciting and fruitful internship
- Many applications and collaborative works will be issued
- Novel iterative compilation method

$\Rightarrow$ The polyhedral model contributes to accelerate the convergence of iterative methods and to discover significant opportunities for performance improvements.

Conclusion

- Very exciting and fruitful internship
- Many applications and collaborative works will be issued
- Novel iterative compilation method

$\Rightarrow$ The polyhedral model contributes to accelerate the convergence of iterative methods and to discover significant opportunities for performance improvements.

Conclusion

- Very exciting and fruitful internship
- Many applications and collaborative works will be issued
- Novel iterative compilation method

$\Rightarrow$ The polyhedral model contributes to accelerate the convergence of iterative methods and to discover significant opportunities for performance improvements.

Conclusion

- Very exciting and fruitful internship
- Many applications and collaborative works will be issued
- Novel iterative compilation method

$\Rightarrow$ The polyhedral model contributes to accelerate the convergence of iterative methods and to discover significant opportunities for performance improvements.

Questions:

A Transformation Example

Optimal Transformation for `mvt`, GCC 4 -O2

```
S1: x1[i] = 0
S2: x2[i] = 0
S3: x1[i] += a[i][j] * y1[j]
S4: x2[i] += a[j][i] * y2[j]

for (i = 0; i <= M; i++) {
  S1(i);
  S2(i);
  for (j = 0; j <= M; j++) {
    S3(i,j);
    S4(i,j);
  }
}
```

```
for (i = 0; i <= M; i++)
  S2(i);

for (c1 = 1; c1 <= M-1; c1++)
  for (i = 0; i <= M; i++) {
    S4(i,c1-1);
  }

for (i = 0; i <= M; i++) {
  S1(i);
  S4(i,M-1);
}

S3(0,0);
S4(0,M);
for (i = 1 ; i <= M; i++)
  S4(i,M);

for (c1 = M+2; c1 <= 3*M+1; c1++)
  for (i = max(c1-2*M-1,0); i <= min(M,c1-M-1); i++) {
    S3(i,c1-i-M-1);
  }
```