Iterative Optimization in the Polyhedral Model: One-Dimensional Affine Schedules

Louis-Noël Pouchet, Cédric Bastoul and Albert Cohen

ALCHEMY, LRI - INRIA Futurs

October 17, 2006
1 Introduction
   • Motivation
   • The Polyhedral Model
   • Polyhedral Representation of programs

2 Iterative Optimization in the Polyhedral Model
   • One-Dimensional Schedules
   • Legal Scheduling Space

3 Experimental Results
   • Exhaustive Scan
   • A Transformation Example

4 Conclusion
Iterative Optimization

- Instead of predicting profitability of a transformation, perform it and run the program
- Most of the time, addresses parameters tuning or phase selection

- Alternatively, some works replace the heuristic itself by iterative search

→ We focus on Loop Nest Optimization
Iterative Optimization

- Instead of predicting profitability of a transformation, perform it and run the program
- Most of the time, addresses parameters tuning or phase selection

- Alternatively, some works replace the heuristic itself by iterative search

→ We focus on Loop Nest Optimization
Iterative Optimization

- Instead of predicting profitability of a transformation, perform it and run the program
- Most of the time, addresses parameters tuning or phase selection

- Alternatively, some works replace the heuristic itself by iterative search

→ We focus on Loop Nest Optimization
Iterative Optimization

- Instead of predicting profitability of a transformation, perform it and run the program
- Most of the time, addresses parameters tuning or phase selection

- Alternatively, some works replace the heuristic itself by iterative search

→ We focus on Loop Nest Optimization
Drawbacks

Limitations:
- The set of combinations of transformations is huge!
- Only a subset of them respects the program semantics

⇒ Only a (very small) subset of transformation sequences is actually tested
⇒ The search space is either too restrictive, or too large due to the postponed legality check

⇒ Can we improve the search space construction: model all sequences of transformations, and model only legal ones?
Drawbacks

Limitations:

- The set of combinations of transformations is huge!
- Only a subset of them respects the program semantics

→ Only a (very small) subset of transformation sequences is actually tested

→ The search space is either too restrictive, or too large due to the postponed legality check

⇒ Can we improve the search space construction: model all sequences of transformations, and model only legal ones?
Introduction: Motivation

Drawbacks

Limitations:
- The set of combinations of transformations is huge!
- Only a subset of them respects the program semantics

→ Only a (very small) subset of transformation sequences is actually tested
→ The search space is either too restrictive, or too large due to the postponed legality check

⇒ Can we improve the search space construction: model all sequences of transformations, and model only legal ones?
Drawbacks

Limitations:

- The set of combinations of transformations is huge!
- Only a subset of them respects the program semantics

⇒ Only a (very small) subset of transformation sequences is actually tested
⇒ The search space is either too restrictive, or too large due to the postponed legality check

⇒ 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 Static Control program Parts (SCoP)
- Use a polyhedral abstraction to represent program information
- Use iterative optimization techniques in the constructed search space

→ In the polyhedral model (Feautrier, 92):
  - Compositions of transformations are easily expressed
  - Transformation legality is easily checked
  - Natural expression of parallelism
Iterative Optimization in the Polyhedral Model

- Focus on a Static Control program Parts (SCoP)
- Use a polyhedral abstraction to represent program information
- Use iterative optimization techniques in the constructed search space

→ In the polyhedral model (Feautrier, 92):
  - Compositions of transformations are easily expressed
  - Transformation legality is easily checked
  - Natural expression of parallelism
Iterative Optimization in the Polyhedral Model

- Focus on a Static Control program Parts (SCoP)
- Use a polyhedral abstraction to represent program information
- Use iterative optimization techniques in the constructed search space

→ In the polyhedral model (Feautrier, 92):
  - Compositions of transformations are easily expressed
  - Transformation legality is easily checked
  - Natural expression of parallelism
Introduction: The Polyhedral Model

Iterative Optimization in the Polyhedral Model

- Focus on a Static Control program Parts (SCoP)
- Use a polyhedral abstraction to represent program information
- Use iterative optimization techniques in the constructed search space

→ In the polyhedral model (Feautrier, 92):
  - Compositions of transformations are easily expressed
  - Transformation legality is easily checked
  - Natural expression of parallelism
A Three-Stage Process

1 Analysis: from code to model

\[
\text{do } i = 1, 3 \\
\quad \text{do } j = 1, 3 \\
\quad \quad \text{A}(i+j) = ... \\
\]

2 Transformation in the model

Here: \( \theta(i^j) = t = i + j \)

3 Code generation: from model to code

\[
\text{do } t = 2, 6 \\
\quad \text{do } i = \max(1,t-3), \min(t-1,3) \\
\quad \quad \text{A}(t) = ... \\
\]
A Three-Stage Process

1 Analysis: from code to model

2 Transformation in the model
Here: $\theta(j) = t = i + j$

3 Code generation: from model to code
A Three-Stage Process

1 Analysis: from code to model

2 Transformation in the model
Here: $\theta(i^j) = t = i + j$

3 Code generation:
from model to code
A Three-Stage Process

1 Analysis: from code to model

2 Transformation in the model
Here: $\theta(j) = t = i + j$

3 Code generation:
from model to code

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

```
do t = 2, 6
    do i = max(1,t-3), min(t-1,3)
        A(t) = ...
    end do
end do
```
A Three-Stage Process

1 Analysis: from code to model
   → Existing prototype tools
   → GCC GRAPHITE branch in development

2 Transformation in the model
   → Build a search space of (legal) transformations

3 Code generation: from model to code
   → Use the CLooG tool for code generation (Bastoul, 04)
   → Produce C compilable code
A Three-Stage Process

1 Analysis: from code to model
   → Existing prototype tools
   → GCC GRAPHITE branch in development

2 Transformation in the model
   → Build a search space of (legal) transformations

3 Code generation: from model to code
   → Use the CLooG tool for code generation (Bastoul, 04)
   → Produce C compilable code
A Three-Stage Process

1 Analysis: from code to model
   → Existing prototype tools
   → GCC GRAPHITE branch in development

2 Transformation in the model
   → Build a search space of (legal) transformations

3 Code generation: from model to code
   → Use the CLooG tool for code generation (Bastoul, 04)
   → Produce C compilable code
Extract the Instance Set

\[ \text{matvect} \]

\[
\begin{align*}
\text{do } & i = 0, n \\
R & \begin{array}{l}
s(i) = 0 \\
\text{do } j = 0, n \\
S & \begin{array}{l}
s(i) = s(i) + a(i, j) \times x(j) \\
\text{end do}
\end{array}
\end{array}
\end{align*}
\]

Iteration domain of \( R \):

- \textbf{iteration vector} \( \vec{x}_R = (i) \)
- Exact set of \textbf{instances} of \( R \) is \( \mathcal{D}_R : \{i | 0 \leq i \leq n\} \)
Extract the Instance Set

\begin{verbatim}
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
\end{verbatim}

Iteration domain of $R$:

- *iteration vector* $\vec{x}_R = (i)$
- Exact set of *instances* of $R$ is $\mathcal{D}_R : \{i \mid 0 \leq i \leq n\}$
**Extract the Instance Set**

`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
  end do
```

Iteration domain of $S$:

- *iteration vector* $\vec{x}_S = (i, j)$
- Exact set of **instances** of $S$ is $\mathcal{D}_S : \{i, j \mid 0 \leq i \leq n, \ 0 \leq j \leq n, \}$
Scheduling a Program

Definition (Schedule)

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

$$\theta_S(\vec{x}_S) = T \left( \begin{array}{c} \vec{x}_S \\ n \\ 1 \end{array} \right)$$

- Two instances having the same date can be run in parallel
- Schedule dimension corresponds to the number of nested sequential loops
Scheduling a Program

Definition (Schedule)

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

$$\theta_S(x_S) = T \left( \begin{array}{c} x_S \\ n \\ 1 \end{array} \right)$$

- Two instances having the same date can be run in parallel
- Schedule dimension corresponds to the number of nested sequential loops
Scheduling a Program

Definition (Schedule)

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

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

- Two instances having the same date can be run in parallel
- Schedule dimension corresponds to the number of nested sequential loops
Program Transformations in the Model

- Every composition of loop transformations can be expressed as affine schedules (Wolf, 92)

⇒ A schedule is the result of an arbitrarily complex composition of transformation
Every composition of loop transformations can be expressed as affine schedules (Wolf, 92)

⇒ A schedule is the result of an arbitrarily complex composition of transformation
A Scheduling Example

Original Schedule

\[ \theta_R \begin{pmatrix} i \\ j \end{pmatrix} = \begin{pmatrix} i \\ j \end{pmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{pmatrix} i \\ j \end{pmatrix} \]

\begin{align*}
\text{do } i = 1, 2 \\
\text{do } j = 1, 3 \\
\quad a(i, j) = a(i, j) \times 0.2
\end{align*}
A Scheduling Example

Another Schedule

\[
\theta_R \begin{pmatrix} i \\ j \end{pmatrix} = \begin{pmatrix} j \\ i \end{pmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{pmatrix} i \\ j \end{pmatrix}
\]

do i = 1, 2
\hspace{1cm} do j = 1, 3
\hspace{1cm} a(i, j) = a(i, j) \times 0.2

do j = 1, 3
\hspace{1cm} do i = 1, 2
\hspace{1cm} a(i, j) = a(i, j) \times 0.2
Context

- Focus on one-dimensional schedules ($T$ is a constant row matrix)
- One-dimensional schedule can represent compositions of:

<table>
<thead>
<tr>
<th>Transformation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>reversal</td>
<td>Changes the direction in which a loop traverses its iteration range</td>
</tr>
<tr>
<td>skewing</td>
<td>Makes the bounds of a given loop depend on an outer loop counter</td>
</tr>
<tr>
<td>interchange</td>
<td>Exchanges two loops in a perfectly nested loop, a.k.a. permutation</td>
</tr>
<tr>
<td>peeling</td>
<td>Extracts one iteration of a given loop</td>
</tr>
<tr>
<td>shifting</td>
<td>Allows to reorder loops</td>
</tr>
<tr>
<td>fusion</td>
<td>Fuses two loops, a.k.a. jamming</td>
</tr>
<tr>
<td>distribution</td>
<td>Splits a single loop nest into many, a.k.a. fission or splitting</td>
</tr>
</tbody>
</table>
Context

- Focus on one-dimensional schedules ($T$ is a constant row matrix)
- One-dimensional schedule can represent compositions of:

<table>
<thead>
<tr>
<th>Transformation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>reversal</td>
<td>Changes the direction in which a loop traverses its iteration range</td>
</tr>
<tr>
<td>skewing</td>
<td>Makes the bounds of a given loop depend on an outer loop counter</td>
</tr>
<tr>
<td>interchange</td>
<td>Exchanges two loops in a perfectly nested loop, a.k.a. permutation</td>
</tr>
<tr>
<td>peeling</td>
<td>Extracts one iteration of a given loop</td>
</tr>
<tr>
<td>shifting</td>
<td>Allows to reorder loops</td>
</tr>
<tr>
<td>fusion</td>
<td>Fuses two loops, a.k.a. jamming</td>
</tr>
<tr>
<td>distribution</td>
<td>Splits a single loop nest into many, a.k.a. fission or splitting</td>
</tr>
</tbody>
</table>
Potential Transformations

\[
\begin{align*}
\text{do } i &= 1, 3 \\
R &
\quad s(i) = 0 \\
\text{do } j &= 1, 3 \\
S &
\quad s(i) = s(i) + a(i)(j) * x(j)
\end{align*}
\]

The two prototype affine schedules for \( R \) and \( S \) are:

\[
\begin{align*}
\theta_R(\vec{x}_R) &= t_{1R} \cdot i_R + t_{2R} \cdot n + t_{3R} \cdot 1 \\
\theta_S(\vec{x}_S) &= t_{1S} \cdot i_S + t_{2S} \cdot j_S + t_{3S} \cdot n + t_{4S} \cdot 1
\end{align*}
\]

⇒ For \(-1 \leq t \leq 1\), there are 59049 values!

<table>
<thead>
<tr>
<th></th>
<th>matvect</th>
<th>locality</th>
<th>matmul</th>
<th>gauss</th>
<th>crout</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bounds</td>
<td>(-1,1)</td>
<td>(-1,1)</td>
<td>(-1,1)</td>
<td>(-1,1)</td>
<td>(-3,3)</td>
</tr>
<tr>
<td>#Sched.</td>
<td>(2.1 \times 10^3)</td>
<td>(5.9 \times 10^4)</td>
<td>(1.9 \times 10^4)</td>
<td>(5.9 \times 10^4)</td>
<td>(2.6 \times 10^{15})</td>
</tr>
</tbody>
</table>
Potential Transformations

\[
\begin{align*}
&\text{do } i = 1, 3 \\
&R \quad s(i) = 0 \\
&\quad \text{do } j = 1, 3 \\
&S \quad s(i) = s(i) + a(i)(j) \cdot x(j)
\end{align*}
\]

The two prototype affine schedules for $R$ and $S$ are:

\[
\begin{align*}
\theta_R(\vec{x}_R) &= t_{1R} \cdot i_R + t_{2R} \cdot n + t_{3R} \cdot 1 \\
\theta_S(\vec{x}_S) &= t_{1S} \cdot i_S + t_{2S} \cdot j_S + t_{3S} \cdot n + t_{4S} \cdot 1
\end{align*}
\]

$\Rightarrow$ For $-1 \leq t \leq 1$, there are 59049 values!

<table>
<thead>
<tr>
<th></th>
<th>matvect</th>
<th>locality</th>
<th>matmul</th>
<th>gauss</th>
<th>crout</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bounds</td>
<td>$-1,1$</td>
<td>$-1,1$</td>
<td>$-1,1$</td>
<td>$-1,1$</td>
<td>$-3,3$</td>
</tr>
<tr>
<td>#Sched.</td>
<td>$2.1 \times 10^3$</td>
<td>$5.9 \times 10^4$</td>
<td>$1.9 \times 10^4$</td>
<td>$5.9 \times 10^4$</td>
<td>$2.6 \times 10^{15}$</td>
</tr>
</tbody>
</table>
Potential Transformations

The two prototype affine schedules for $R$ and $S$ are:

$$\theta_R(\vec{x}_R) = t_{1R} \cdot i_R + t_{2R} \cdot n + t_{3R} \cdot 1$$
$$\theta_S(\vec{x}_S) = t_{1S} \cdot i_S + t_{2S} \cdot j_S + t_{3S} \cdot n + t_{4S} \cdot 1$$

⇒ For $-1 \leq t \leq 1$, there are $59049$ values!

<table>
<thead>
<tr>
<th></th>
<th>matvect</th>
<th>locality</th>
<th>matmul</th>
<th>gauss</th>
<th>crout</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bounds</td>
<td>$-1,1$</td>
<td>$-1,1$</td>
<td>$-1,1$</td>
<td>$-1,1$</td>
<td>$-3,3$</td>
</tr>
<tr>
<td>#Sched.</td>
<td>$2.1 \times 10^3$</td>
<td>$5.9 \times 10^4$</td>
<td>$1.9 \times 10^4$</td>
<td>$5.9 \times 10^4$</td>
<td>$2.6 \times 10^{15}$</td>
</tr>
</tbody>
</table>
Objectives

- Build the set of all *legal* program versions (i.e. which respects all the data dependence of the program)

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

- The resulting space represents all the distinct possible ways to *legally reschedule* the program, using arbitrarily complex sequences of transformations.
Objectives

- Build the set of all *legal* program versions (i.e. which respects all the data dependence of the program)

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

⇒ The resulting space represents all the distinct possible ways to *legally reschedule* the program, using arbitrarily complex sequences of transformations.
Objectives

- Build the set of all *legal* program versions (i.e. which respects all the data dependence of the program)

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

⇒ The resulting space represents all the distinct possible ways to *legally reschedule* the program, using arbitrarily complex sequences 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 (Feautrier, 92)
- Use a subset of the Cartesian product of iteration domains:

\[
\begin{align*}
\text{do } i &= 1, 3 \\
R_{s(i)} &= 0 \\
\text{do } j &= 1, 3 \\
S_{s(i)} &= s(i) + a(i)(j) \times x(j)
\end{align*}
\]
Dependence Expression

- Need to represent the *exact* set of instances in dependence
- Exact computation made possible thanks to the SCoP and Static reference assumptions (Feautrier, 92)
- Use a subset of the Cartesian product of iteration domains:

```
Do i = 1, 3
  Rs(i) = 0
Do j = 1, 3
  Si = Si + a(i)(j) * x(j)
```

**Iteration directions:**

- Iterations of R
  - Iteration direction of R

**Equation:**

\[
\mathcal{D}_{R\delta S} : \begin{bmatrix}
1 & 0 & 0 & 0 & -1 \\
-1 & 0 & 0 & 0 & 3 \\
0 & 1 & 0 & 0 & -1 \\
0 & -1 & 0 & 0 & 3 \\
0 & 0 & 1 & 0 & -1 \\
0 & 0 & -1 & 0 & 3 \\
1 & -1 & 0 & 0 & 0
\end{bmatrix} \begin{bmatrix}
i_R \\
i_S \\
j_S \\
j_n \\
i
\end{bmatrix} \geq \begin{bmatrix}0 \\0\end{bmatrix} = 0
\]
Dependence Expression

- Need to represent the *exact* set of instances in dependence
- Exact computation made possible thanks to the SCoP and Static reference assumptions (Feautrier, 92)
- Use a subset of the Cartesian product of iteration domains:

\[
\begin{align*}
\text{do } i &= 1, 3 \\
R &\quad s(i) = 0 \\
\quad \text{do } j &= 1, 3 \\
S &\quad s(i) = s(i) + a(i)(j) \times x(j)
\end{align*}
\]

\[
\mathcal{D}_{R\delta S} : \begin{bmatrix}
1 & 0 & 0 & 0 & -1 \\
-1 & 0 & 0 & 0 & 3 \\
0 & 1 & 0 & 0 & -1 \\
0 & -1 & 0 & 0 & 3 \\
0 & 0 & 1 & 0 & -1 \\
0 & 0 & -1 & 0 & 3 \\
1 & -1 & 0 & 0 & 0
\end{bmatrix}
\begin{bmatrix}
i_R \\
i_S \\
j_S \\
j_n \\
1
\end{bmatrix} \geq \begin{bmatrix}
\vec{0}
\end{bmatrix}
\]

\[
\begin{bmatrix}
i_R \\
i_S \\
j_S \\
j_n \\
1
\end{bmatrix} = \begin{bmatrix}
\vec{0}
\end{bmatrix}
\]
Dependence Expression

- Need to represent the *exact* set of instances in dependence
- Exact computation made possible thanks to the SCoP and Static reference assumptions (Feautrier, 92)
- Use a subset of the Cartesian product of iteration domains:

\[
\begin{align*}
\text{do } i &= 1, 3 \\
R & \quad s(i) = 0 \\
\text{do } j &= 1, 3 \\
S & \quad s(i) = s(i) + a(i)(j) * x(j)
\end{align*}
\]

\[
\mathcal{D}_{R} \delta S : \begin{bmatrix}
1 & 0 & 0 & 0 & 0 & -1 \\
-1 & 0 & 0 & 0 & 3 \\
0 & 1 & 0 & 0 & -1 \\
0 & -1 & 0 & 0 & 3 \\
0 & 0 & 1 & 0 & -1 \\
0 & 0 & -1 & 0 & 3 \\
1 & -1 & 0 & 0 & 0
\end{bmatrix} \begin{bmatrix}
i_R \\
i_S \\
j_R \\
j_S \\
i
\end{bmatrix} \geq \begin{bmatrix}
\vec{0} \\
\vec{0}
\end{bmatrix} = 0
\]
Formal Definition [1/2]

⇒ 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]

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

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

$\lambda_0$ and $\tilde{\lambda}^T$ are called the Farkas multipliers.

We can express the set of affine, non-negative functions over $\mathcal{D}_{R\delta S}$
Formal Definition [2/2]

→ We can express the legality condition as a set of affine non-negative functions over $D_{R\delta S}$

Lemma (Affine form of Farkas lemma)

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

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

$\lambda_0$ and $\tilde{\lambda}^T$ are called the Farkas multipliers.

⇒ We can express the set of affine, non-negative functions over $D_{R\delta S}$
Formal Definition [2/2]

→ 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 $A\tilde{x} + \tilde{b} \geq \tilde{0}$. Then any affine function $f(\tilde{x})$ is non-negative everywhere in $\mathcal{D}$ iff it is a positive affine combination:

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

$\lambda_0$ and $\tilde{\lambda}^T$ are called the Farkas multipliers.

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

\[
\begin{align*}
\text{do } i &= 1, n \\
R &\quad s(i) = 0 \\
\text{do } j &= 1, n \\
S &\quad s(i) = s(i) + a(i,j) \times x(j)
\end{align*}
\]

The two prototype affine schedules for \( R \) and \( S \) are:

\[
\begin{align*}
\theta_R(\bar{x}_R) &= t_{1R} \cdot i_R + t_{2R} \cdot n + t_{3R} \cdot 1 \\
\theta_S(\bar{x}_S) &= t_{1S} \cdot i_S + t_{2S} \cdot j_S + t_{3S} \cdot n + t_{4S} \cdot 1
\end{align*}
\]

The set of instances of \( R \) and \( S \) in dependence are represented by:

\[
D_{R\delta S} : \begin{bmatrix}
1 & -1 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 \\
-1 & 0 & 0 & 1 & 0 \\
0 & 1 & 0 & 0 & 0 \\
0 & -1 & 0 & 1 & 0 \\
0 & 0 & 1 & 0 & 0 \\
0 & 0 & -1 & 1 & 0
\end{bmatrix} \cdot \begin{bmatrix}
\begin{array}{c}
i_R \\
i_S \\
j_S \\
n \\
1
\end{array}
\end{bmatrix} \geq 0
\]
An Example

The two prototype affine schedules for $R$ and $S$ are:

\[
\theta_R(\vec{x}_R) = t_{1_R} \cdot i_R + t_{2_R} \cdot n + t_{3_R} \cdot 1
\]
\[
\theta_S(\vec{x}_S) = t_{1_S} \cdot i_S + t_{2_S} \cdot j_S + t_{3_S} \cdot n + t_{4_S} \cdot 1
\]

1. Express the set of non-negative functions over $\mathcal{D}_{R\delta S}$
2. Equate the coefficients
3. Solve the system
An Example

\[
\begin{align*}
\text{R} & \quad \text{do } i = 1, n \\
& \quad \quad s(i) = 0 \\
& \quad \quad \text{do } j = 1, n \\
& \quad \quad s(i) = s(i) + a(i,j) \times x(j)
\end{align*}
\]

The two prototype affine schedules for \( R \) and \( S \) are:

\[
\begin{align*}
\theta_R(\bar{x}_R) & = t_{1R} \cdot i_R + t_{2R} \cdot n + t_{3R} \cdot 1 \\
\theta_S(\bar{x}_S) & = t_{1S} \cdot i_S + t_{2S} \cdot n + t_{3S} \cdot 1 + t_{4S} \cdot 1
\end{align*}
\]

We get the following system for \( R \delta S \):

\[
\begin{align*}
D_{R \delta S} & \quad \begin{array}{l}
i_R \\
i_S \\
i_S \\
n \\
1
\end{array} \\
& \quad \begin{array}{l}
-t_{1R} \\
t_{1S} \\
t_{2S} \\
t_3S - t_{2R} \\
t_4S - t_3R - 1
\end{array} \\
& = \begin{array}{l}
\lambda_{D1,1} - \lambda_{D1,2} + \lambda_{D1,7} \\
\lambda_{D1,3} - \lambda_{D1,4} - \lambda_{D1,7} \\
\lambda_{D1,5} - \lambda_{D1,6} \\
\lambda_{D1,2} + \lambda_{D1,4} + \lambda_{D1,6} \\
\lambda_{D1,0}
\end{array}
\end{align*}
\]

\[\Rightarrow\] The constraints on \( t \) gives the set of possible values to respect the legality condition
An Example

The two prototype affine schedules for \( R \) and \( S \) are:

\[
\theta_R(\vec{x}_R) = t_{1R} \cdot i_R + t_{2R} \cdot n + t_{3R} \cdot 1 \\
\theta_S(\vec{x}_S) = t_{1S} \cdot i_S + t_{2S} \cdot j_S + t_{3S} \cdot n + t_{4S} \cdot 1
\]

We get the following system for \( R \delta S \):

\[
\begin{align*}
D_{R \delta S} & \\
\begin{cases}
i_R : & -t_{1R} = \lambda_{D_{1,1}} - \lambda_{D_{1,2}} + \lambda_{D_{1,7}} \\
i_S : & t_{1S} = \lambda_{D_{1,3}} - \lambda_{D_{1,4}} - \lambda_{D_{1,7}} \\
j_S : & t_{2S} = \lambda_{D_{1,5}} - \lambda_{D_{1,6}} \\
n : & t_{3S} - t_{2R} = \lambda_{D_{1,2}} + \lambda_{D_{1,4}} + \lambda_{D_{1,6}} \\
1 : & t_{4S} - t_{3R} - 1 = \lambda_{D_{1,0}}
\end{cases}
\end{align*}
\]

\( \Rightarrow \) The constraints on \( t \) gives the set of possible values to respect the legality condition
Construction Algorithm

- Need to add the constraints obtained for each dependence
  - The set of legal transformations can be infinite
    → Need to bound the space

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

- Need to add the constraints obtained for each dependence
- The set of legal transformations can be infinite
  - Need to bound the space

⇒ To each (integral) point in $D_t$ corresponds a different version of the original program where the semantics is preserved.
Construction Algorithm

- Need to add the constraints obtained for each dependence
- The set of legal transformations can be infinite
  \[\Rightarrow\] Need to bound the space

\[\Rightarrow\] To each (integral) point in \(\mathcal{D}_t\) corresponds a different version of the original program where the semantics is preserved.
Legal Search Space

- Multiple orders of magnitude reduction in the size of the search space compared to state-of-the-art techniques

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Bounds</th>
<th>#Sched</th>
<th>#Legal</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>matvect</td>
<td>−1, 1</td>
<td>$2.1 \times 10^3$</td>
<td>129</td>
<td>0.024</td>
</tr>
<tr>
<td>locality</td>
<td>−1, 1</td>
<td>$5.9 \times 10^4$</td>
<td>6561</td>
<td>0.022</td>
</tr>
<tr>
<td>matmul</td>
<td>−1, 1</td>
<td>$1.9 \times 10^4$</td>
<td>912</td>
<td>0.029</td>
</tr>
<tr>
<td>gauss</td>
<td>−1, 1</td>
<td>$5.9 \times 10^4$</td>
<td>506</td>
<td>0.047</td>
</tr>
<tr>
<td>crout</td>
<td>−3, 3</td>
<td>$2.6 \times 10^{15}$</td>
<td>798</td>
<td>0.046</td>
</tr>
</tbody>
</table>
Experimental Protocol

We provide a **source-to-source framework**. Given an input program:

1. **Use** LetSee **to generate a CLooG formatted file per legal transformation.**
2. **Generate the target code with CLooG.**
3. **Compile and launch the whole set of transformed (C) code, and sort the results regarding cycle count.**

⇒ Exhaustive scan is achievable on small kernels.
Experimental Protocol

We provide a **source-to-source framework**. Given an input program:

1. **Use **LetSee** to generate a **CLooG** formatted file per legal transformation.**
2. Generate the target code with **CLooG**.
3. Compile and launch the whole set of transformed (C) code, and sort the results regarding cycle count.

⇒ Exhaustive scan is achievable on small kernels
Experimental Results: Exhaustive Scan

Performance Distribution [1/2]

Figure: Performance distribution for matmul, locality, mvt and crout
Performance Distribution [2/2]

(a) GCC -O3

(b) ICC -fast

Figure: The effect of the compiler
Some Speedups

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Compiler</th>
<th>Options</th>
<th>Parameters</th>
<th>ID best</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>h264</td>
<td>PathCC</td>
<td>-Ofast</td>
<td>N=8</td>
<td>352</td>
<td>36.1%</td>
</tr>
<tr>
<td>h264</td>
<td>GCC</td>
<td>-O2</td>
<td>N=8</td>
<td>234</td>
<td>13.3%</td>
</tr>
<tr>
<td>h264</td>
<td>GCC</td>
<td>-O3</td>
<td>N=8</td>
<td>250</td>
<td>25.0%</td>
</tr>
<tr>
<td>h264</td>
<td>ICC</td>
<td>-O2</td>
<td>N=8</td>
<td>290</td>
<td>12.9%</td>
</tr>
<tr>
<td>h264</td>
<td>ICC</td>
<td>-fast</td>
<td>N=8</td>
<td>N/A</td>
<td>0%</td>
</tr>
<tr>
<td>fir</td>
<td>PathCC</td>
<td>-Ofast</td>
<td>N=150000</td>
<td>72</td>
<td>6.0%</td>
</tr>
<tr>
<td>fir</td>
<td>GCC</td>
<td>-O2</td>
<td>N=150000</td>
<td>192</td>
<td>15.2%</td>
</tr>
<tr>
<td>fir</td>
<td>GCC</td>
<td>-O3</td>
<td>N=150000</td>
<td>289</td>
<td>13.2%</td>
</tr>
<tr>
<td>fir</td>
<td>ICC</td>
<td>-O2</td>
<td>N=150000</td>
<td>242</td>
<td>18.4%</td>
</tr>
<tr>
<td>fir</td>
<td>ICC</td>
<td>-fast</td>
<td>N=150000</td>
<td>392</td>
<td>3.4%</td>
</tr>
<tr>
<td>MVT</td>
<td>PathCC</td>
<td>-Ofast</td>
<td>N=2000</td>
<td>4934</td>
<td>27.4%</td>
</tr>
<tr>
<td>MVT</td>
<td>GCC</td>
<td>-O2</td>
<td>N=2000</td>
<td>13301</td>
<td>18.0%</td>
</tr>
<tr>
<td>MVT</td>
<td>GCC</td>
<td>-O3</td>
<td>N=2000</td>
<td>13320</td>
<td>21.2%</td>
</tr>
<tr>
<td>MVT</td>
<td>ICC</td>
<td>-O2</td>
<td>N=2000</td>
<td>14093</td>
<td>24.0%</td>
</tr>
<tr>
<td>MVT</td>
<td>ICC</td>
<td>-fast</td>
<td>N=2000</td>
<td>4879</td>
<td>29.1%</td>
</tr>
<tr>
<td>matmul</td>
<td>PathCC</td>
<td>-Ofast</td>
<td>N=250</td>
<td>283</td>
<td>308.1%</td>
</tr>
<tr>
<td>matmul</td>
<td>GCC</td>
<td>-O2</td>
<td>N=250</td>
<td>573</td>
<td>243.6%</td>
</tr>
<tr>
<td>matmul</td>
<td>GCC</td>
<td>-O3</td>
<td>N=250</td>
<td>143</td>
<td>248.7%</td>
</tr>
<tr>
<td>matmul</td>
<td>ICC</td>
<td>-O2</td>
<td>N=250</td>
<td>311</td>
<td>356.6%</td>
</tr>
<tr>
<td>matmul</td>
<td>ICC</td>
<td>-fast</td>
<td>N=250</td>
<td>641</td>
<td>645.4%</td>
</tr>
</tbody>
</table>
The $\texttt{mvt}$ Kernel

```c
for (i = 0; i <= M; i++) {
    x1[i] = 0;
    x2[i] = 0;
    for (j = 0; j <= M; j++) {
        x1[i] += a[i][j] * y1[j];
        x2[i] += a[j][i] * y2[j];
    }
}
```

<table>
<thead>
<tr>
<th>Compiler</th>
<th>Option</th>
<th>Original</th>
<th>Best</th>
<th>Schedule</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>GCC 4.1.1</td>
<td>-O3</td>
<td>6.9</td>
<td>5.1</td>
<td>$\theta_{S1}(\vec{x}_{S1}) = -i - n - 1$</td>
<td>35.3%</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>$\theta_{S2}(\vec{x}_{S2}) = -1$</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>$\theta_{S1}(\vec{x}_{S1}) = j + 1$</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>$\theta_{S2}(\vec{x}_{S2}) = i + j + n + 1$</td>
<td></td>
</tr>
<tr>
<td>ICC 9.0.1</td>
<td>-fast</td>
<td>6.1</td>
<td>4.9</td>
<td>$\theta_{S1}(\vec{x}_{S1}) = n - 1$</td>
<td>24.5%</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>$\theta_{S2}(\vec{x}_{S2}) = -n - 1$</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>$\theta_{S1}(\vec{x}_{S1}) = j + n + 1$</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>$\theta_{S2}(\vec{x}_{S2}) = j - n$</td>
<td></td>
</tr>
<tr>
<td>PathCC 2.5</td>
<td>-Ofast</td>
<td>7.3</td>
<td>5.9</td>
<td>$\theta_{S1}(\vec{x}_{S1}) = -i - n - 1$</td>
<td>23.8%</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>$\theta_{S2}(\vec{x}_{S2}) = -i - n$</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>$\theta_{S1}(\vec{x}_{S1}) = -i + j + n + 1$</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>$\theta_{S2}(\vec{x}_{S2}) = -i + j + 1$</td>
<td></td>
</tr>
</tbody>
</table>
**Generated Code**

**Optimal Transformation for mvt, GCC 4 -O3, P4 Xeon**

- **S1**: \( x_1[i] = 0 \)
- **S2**: \( x_2[i] = 0 \)
- **S3**: \( x_1[i] += a[i][j] \times y_1[j] \)
- **S4**: \( x_2[i] += a[j][i] \times y_2[j] \)

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

```c
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);
}
```

```c
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);
    }
```
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

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>#Schedules</th>
<th>Heuristic.</th>
<th>#Runs</th>
<th>%Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>locality</td>
<td>6561</td>
<td>Rand DH</td>
<td>125</td>
<td>96.1%</td>
</tr>
<tr>
<td>matmul</td>
<td>912</td>
<td>Rand DH</td>
<td>170</td>
<td>99.9%</td>
</tr>
<tr>
<td>mvt</td>
<td>16641</td>
<td>Rand DH</td>
<td>30</td>
<td>93.3%</td>
</tr>
</tbody>
</table>
Conclusion:

Iterative Compilation Framework independent of the compiler and the architecture
Optimizing and / or Enabling transformation process
Leads to encouraging speedups
On small kernels, exhaustive scan is achievable

Future work:
Develop new exploration heuristics
Deal with multidimensional schedules
Integrate in GCC GRAPHITE branch
Conclusion

→ Iterative Compilation Framework independent of the compiler and the architecture
→ Optimizing and / or Enabling transformation process
→ Leads to encouraging speedups
→ On small kernels, exhaustive scan is achievable

Future work:
→ Develop new exploration heuristics
→ Deal with multidimensional schedules
→ Integrate in GCC GRAPHITE branch
Conclusion

→ Iterative Compilation Framework independent of the compiler and the architecture

→ Optimizing and / or Enabling transformation process

→ Leads to encouraging speedups

→ On small kernels, exhaustive scan is achievable

Future work:

→ Develop new exploration heuristics

→ Deal with multidimensional schedules

→ Integrate in GCC GRAPHITE branch
Conclusion

→ Iterative Compilation Framework independent of the compiler and the architecture
→ Optimizing and / or Enabling transformation process
→ Leads to encouraging speedups
→ On small kernels, exhaustive scan is achievable

Future work:

→ Develop new exploration heuristics
→ Deal with multidimensional schedules
→ Integrate in GCC GRAPHITE branch
Conclusion

→ Iterative Compilation Framework independent of the compiler and the architecture
→ Optimizing and / or Enabling transformation process
→ Leads to encouraging speedups
→ On small kernels, **exhaustive scan** is achievable

Future work:
→ Develop new exploration heuristics
→ Deal with multidimensional schedules
→ Integrate in GCC GRAPHITE branch
Conclusion

→ Iterative Compilation Framework independent of the compiler and the architecture
→ Optimizing and / or Enabling transformation process
→ Leads to encouraging speedups
→ On small kernels, exhaustive scan is achievable

Future work:
→ Develop new exploration heuristics
→ Deal with multidimensional schedules
→ Integrate in GCC GRAPHITE branch
Conclusion

→ Iterative Compilation Framework independent of the compiler and the architecture
→ Optimizing and / or Enabling transformation process
→ Leads to encouraging speedups
→ On small kernels, exhaustive scan is achievable

Future work:
→ Develop new exploration heuristics
→ Deal with multidimensional schedules
→ Integrate in GCC GRAPHITE branch