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

Iterative Optimization

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

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

$\longrightarrow$ 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

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

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

$\Rightarrow$ 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?

Drawbacks

Limitations:

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

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

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

$\Rightarrow$ 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 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

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

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

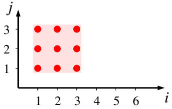
- Compositions of transformations are easily expressed
- Transformation legality is easily checked
- Natural expression of parallelism

A Three-Stage Process

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

1 Analysis: from code to model

$\Downarrow$



2 Transformation in the model

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

$\Downarrow$

3 Code generation: from model to code

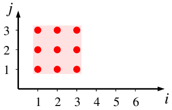
$\Downarrow$

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

A Three-Stage Process

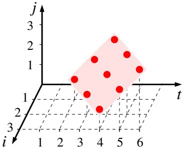
```
do i = 1, 3
|  do j = 1, 3
|  |  A(i+j) = ...
```

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

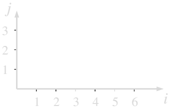
```
do t = 2, 6
|  do i = max(1,t-3), min(t-1,3)
|  |  A(t) = ...
```

A Three-Stage Process

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

1 Analysis: from code to model

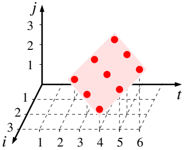
⇓



2 Transformation in the model

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

⇓



3 Code generation: from model to code

⇓

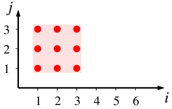
```
do t = 2, 6
|  do i = max(1,t-3), min(t-1,3)
|  |  A(t) = ...
```

A Three-Stage Process

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

1 Analysis: from code to model

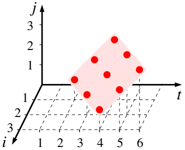
$\Downarrow$



2 Transformation in the model

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

$\Downarrow$



$\Downarrow$

3 Code generation:
from model to code

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

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

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* $\vec{x}_R = (i)$
- 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
```

Iteration domain of R:

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

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
```

Iteration domain of S:

- *iteration vector* $\vec{x}_S = \binom{i}{j}$
- 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):

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

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

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):

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

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

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):

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

- 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)

$\Rightarrow$ A schedule is the result of an **arbitrarily complex composition** of transformation

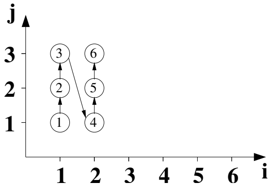
Program Transformations in the Model

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

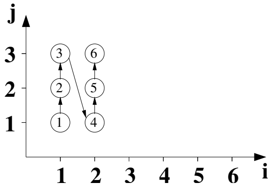
$\Rightarrow$ A schedule is the result of an **arbitrarily complex composition** of transformation

A Scheduling Example

Original Schedule



$\Longrightarrow$



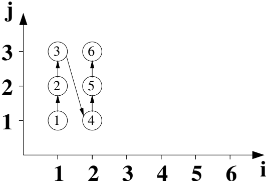
$$\theta_R \begin{pmatrix} \mathbf{i} \\ \mathbf{j} \end{pmatrix} = \begin{pmatrix} \mathbf{i} \\ \mathbf{j} \end{pmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{pmatrix} i \\ j \end{pmatrix}$$

```
do i = 1, 2
|   do j = 1, 3
|   |   a(i,j) = a(i,j) * 0.2
```

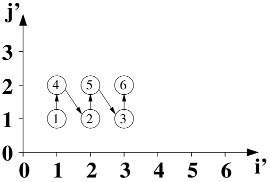
```
do i = 1, 2
|   do j = 1, 3
|   |   a(i,j) = a(i,j) * 0.2
```

A Scheduling Example

Another Schedule



$\Longrightarrow$



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

```
do i = 1, 2
|  do j = 1, 3
|  |  a(i,j) = a(i,j) * 0.2
```

```
do j = 1, 3
|  do i = 1, 2
|  |  a(i,j) = a(i,j) * 0.2
```

Context

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

Transformation	Description
`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`
`peeling`	Extracts one iteration of a given loop
`shifting`	Allows to reorder loops
`fusion`	Fuses two loops, a.k.a. `jamming`
`distribution`	Splits a single loop nest into many, a.k.a. `fission` or `splitting`

Context

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

Transformation	Description
`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`
`peeling`	Extracts one iteration of a given loop
`shifting`	Allows to reorder loops
`fusion`	Fuses two loops, a.k.a. `jamming`
`distribution`	Splits a single loop nest into many, a.k.a. `fission` or `splitting`

Potential Transformations

```
    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:

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

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

	matvect	locality	matmul	gauss	crout
Bounds	$-1, 1$	$-1, 1$	$-1, 1$	$-1, 1$	$-3, 3$
#Sched.	2.1×10^3	5.9×10^4	1.9×10^4	5.9×10^4	2.6×10^{15}

Potential Transformations

```
   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:

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

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

	matvect	locality	matmul	gauss	crout
Bounds	$-1, 1$	$-1, 1$	$-1, 1$	$-1, 1$	$-3, 3$
#Sched.	2.1×10^3	5.9×10^4	1.9×10^4	5.9×10^4	2.6×10^{15}

Potential Transformations

```
    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:

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

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

	matvect	locality	matmul	gauss	crout
Bounds	$-1, 1$	$-1, 1$	$-1, 1$	$-1, 1$	$-3, 3$
#Sched.	2.1×10^3	5.9×10^4	1.9×10^4	5.9×10^4	2.6×10^{15}

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)

$\longrightarrow$ 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 sequences of transformations.

Objectives

- 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 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:

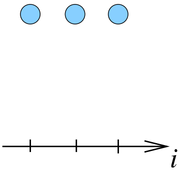
```
     do i = 1, 3
R    |  s(i) = 0
     |  do j = 1, 3
S    |  |  s(i) = s(i) + a(i)(j) * x(j)
```

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

Iterations of R



$$\mathcal{D}_{R\delta S}: \left[\begin{array}{ccccc} 1 & 0 & 0 & 0 & -1 \\ -1 & 0 & 0 & 0 & 3 \\ 0 & 1 & 0 & 0 & -1 \\ 0 & -1 & 0 & 0 & 3 \\ 0 & 0 & 1 & 0 & -1 \\ 0 & 0 & -1 & 0 & 3 \\ \hline 1 & -1 & 0 & 0 & 0 \end{array}\right] \cdot \begin{pmatrix} \mathbf{i_R} \\ i_S \\ j_S \\ \mathbf{n} \\ \mathbf{1} \end{pmatrix} \begin{array}{c} \geq \vec{0} \\ \hline = 0 \end{array}$$

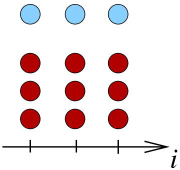
Dependence Expression

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

Iterations of R

Iterations of S



$$\mathcal{D}_{R\delta S}:\left[\begin{array}{ccccc} 1 & 0 & 0 & 0 & -1 \\ -1 & 0 & 0 & 0 & 3 \\ 0 & 1 & 0 & 0 & -1 \\ 0 & -1 & 0 & 0 & 3 \\ 0 & 0 & 1 & 0 & -1 \\ 0 & 0 & -1 & 0 & 3 \\ \hline 1 & -1 & 0 & 0 & 0 \end{array}\right] \cdot \begin{pmatrix} i_R \\ i_S \\ j_S \\ n \\ 1 \end{pmatrix} \begin{array}{c} \geq \vec{0} \\ \hline = 0 \end{array}$$

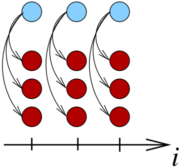
Dependence Expression

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

Iterations of R

Iterations of S



$$\mathcal{D}_{R\delta S} : \left[\begin{array}{ccccc} 1 & 0 & 0 & 0 & -1 \\ -1 & 0 & 0 & 0 & 3 \\ 0 & 1 & 0 & 0 & -1 \\ 0 & -1 & 0 & 0 & 3 \\ 0 & 0 & 1 & 0 & -1 \\ 0 & 0 & -1 & 0 & 3 \\ \hline 1 & -1 & 0 & 0 & 0 \end{array} \right] \cdot \begin{pmatrix} i_R \\ i_S \\ j_S \\ n \\ 1 \end{pmatrix} \begin{array}{c} \geq \vec{0} \\ \hline = 0 \end{array}$$

Formal Definition [1/2]

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:

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

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]

$\longrightarrow$ 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]

$\longrightarrow$ 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 $\mathcal{D}_{R\delta S}$

An Example

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

The two prototype affine schedules for R and S are:

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

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

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

An Example

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

The two prototype affine schedules for R and S are:

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

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

An Example

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

The two prototype affine schedules for R and S are:

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

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

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

$\Rightarrow$ The constraints on t gives the set of possible values to respect the legality condition

An Example

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

The two prototype affine schedules for R and S are:

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

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

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

$\Rightarrow$ 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
  - $\longrightarrow$ 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.

Construction Algorithm

- Need to add the constraints obtained for each dependence
- The set of legal transformations can be infinite
  - $\longrightarrow$ 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.

Construction Algorithm

- Need to add the constraints obtained for each dependence
- The set of legal transformations can be infinite
  - $\longrightarrow$ 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

Benchmark	Bounds	#Sched	#Legal	Time
`matvect`	$-1, 1$	2.1×10^3	129	0.024
`locality`	$-1, 1$	5.9×10^4	6561	0.022
`matmul`	$-1, 1$	1.9×10^4	912	0.029
`gauss`	$-1, 1$	5.9×10^4	506	0.047
`crout`	$-3, 3$	2.6×10^{15}	798	0.046

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.

$\Rightarrow$ Exhaustive scan is achievable on small kernels

Performance Distribution [1/2]

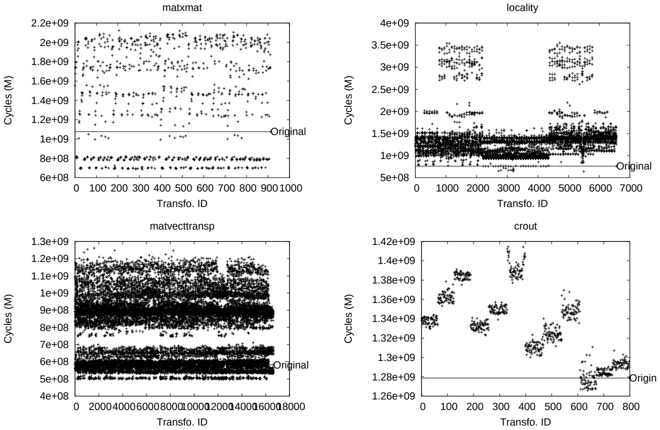
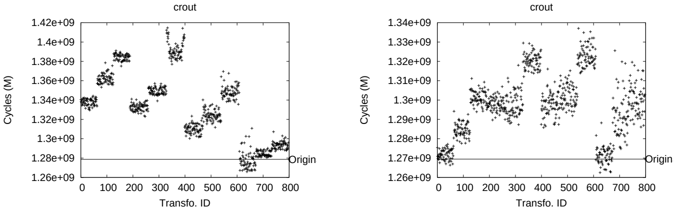


Figure: Performance distribution for `matmul`, `locality`, `mvt` and `crout`

Performance Distribution [2/2]



(a) GCC -O3 (b) ICC -fast

Figure: The effect of the compiler

Some Speedups

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

The `mvt` Kernel

```
     for (i = 0; i <= M; i++) {
S1      x1[i] = 0;
S2      x2[i] = 0;
        for (j = 0; j <= M; j++)  {
S3         x1[i] += a[i][j] * y1[j];
S4         x2[i] += a[j][i] * y2[j];
        }
     }
```

Compiler	Option	Original	Best	Schedule	Speedup
GCC 4.1.1	-O3	6.9	5.1	$\theta_{S1}(\vec{x}_{S1}) = -i-n-1$ $\theta_{S2}(\vec{x}_{S2}) = -1$ $\theta_{S1}(\vec{x}_{S1}) = j+1$ $\theta_{S2}(\vec{x}_{S2}) = i+j+n+1$	35.3%
ICC 9.0.1	-fast	6.1	4.9	$\theta_{S1}(\vec{x}_{S1}) = n-1$ $\theta_{S2}(\vec{x}_{S2}) = -n-1$ $\theta_{S1}(\vec{x}_{S1}) = j+n+1$ $\theta_{S2}(\vec{x}_{S2}) = j-n$	24.5%
PathCC 2.5	-Ofast	7.3	5.9	$\theta_{S1}(\vec{x}_{S1}) = -i-n-1$ $\theta_{S2}(\vec{x}_{S2}) = -i-n$ $\theta_{S1}(\vec{x}_{S1}) = -i+j+n+1$ $\theta_{S2}(\vec{x}_{S2}) = -i+j+1$	23.8%

Generated Code

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

```
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);
  }
```

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%

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