Iterative Optimization in the Polyhedral Model: Part I, One-Dimensional Time

Louis-Noël Pouchet, Cédric Bastoul, Albert Cohen and Nicolas Vasilache

ALCHEMY, INRIA Futurs / University of Paris-Sud XI

March 12, 2007

Fifth International Symposium on Code Generation and Optimization
San Jose, California

Outline

Context of this study:

- Focus on Loop Nest Optimization for regular loops
- Automatic method for parallelism extraction / loop transformation
- Combine iterative methods with the power of the polyhedral model
- Solution independent of the compiler and the target machine

Our contribution:

- Search space construction
  - 1 point in the space $\Leftrightarrow$ 1 distinct legal program version
  - suitable for various exploration methods
- Performance
  - 99% of the best speedup attained within 20 runs of a dedicated heuristic
  - wall clock optimal transformation discoverable on small kernels

One-Dimensional Scheduling

Original Schedule

```
for (i=0; i<n; ++i) {
. S1(i);
. for (j=0; j<n; ++j)
. . S2(i,j);
}
```

$$\begin{cases} \theta_{S1} = i \\ \theta_{S2} = i \end{cases}$$

```
for (i=0; i<n; ++i) {
. S1(i);
. for (j=0; j<n; ++j)
. . S2(i,j);
}
```

- Specify the outer-most loop only
- **Initial outer-most loop is** i

One-Dimensional Scheduling

Distribute loops

```
for (i=0; i<n; ++i) {
. S1(i);
. for (j=0; j<n; ++j)
. . S2(i,j);
}
```

$$\begin{cases} \theta_{S1} = i \\ \theta_{S2} = i + \mathbf{n} \end{cases}$$

```
for (i=0; i<n; ++i)
. S1(i);
for (i=n; i<2*n; ++i)
. for (j=0; j<n; ++j)
. . S2(i-n,j);
```

- Specify the outer-most loop only
- **All instances of S1 are executed before the first S2 instance**

One-Dimensional Scheduling

Distribute loops + Interchange loops for S2

```
for (i=0; i<n; ++i) {
. S1(i);
. for (j=0; j<n; ++j)
. . S2(i,j);
}
```

$$\begin{cases} \theta_{S1} = i \\ \theta_{S2} = j + n \end{cases}$$

```
for (i=0; i<n; ++i)
. S1(i);
for (j=n; j<2*n; ++j)
. for (i=0; i<n; ++i)
. . S2(i,j-n);
```

- Specify the outer-most loop only
- **The outer-most loop for S2 becomes** j

One-Dimensional Scheduling

Distribute loops + Interchange loops for S2

```
for (i=0; i<n; ++i) {
. S1(i);
. for (j=0; j<n; ++j)
. . S2(i,j);
}
```

$$\begin{cases} \theta_{S1} = i \\ \theta_{S2} = \mathbf{j} + n \end{cases}$$

```
for (i=0; i<n; ++i)
. S1(i);
for (j=n; j<2*n; ++j)
. for (i=0; i<n; ++i)
. . S2(i,j-n);
```

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

One-Dimensional Scheduling

```
for (i=0; i<n; ++i) {
. S1(i);
. for (j=0; j<n; ++j)
. . S2(i,j);
}
```

- **A schedule is an affine function of the iteration vector and the parameters**

$$
\begin{aligned}
\theta_{S1}(\vec{x}_{S1}) &= \mathbf{t_{1_{S1}}}.i_{S1} + \mathbf{t_{2_{S1}}}.n + \mathbf{t_{3_{S1}}}.1 \\
\theta_{S2}(\vec{x}_{S2}) &= \mathbf{t_{1_{S2}}}.i_{S2} + \mathbf{t_{2_{S2}}}.j_{S2} + \mathbf{t_{3_{S2}}}.n + \mathbf{t_{4_{S2}}}.1
\end{aligned}
$$

One-Dimensional Scheduling

```
for (i=0; i<n; ++i) {
. s[i] = 0;
. for (j=0; j<n; ++j)
. . s[i] = s[i]+a[i][j]*x[j];
}
```

- **A schedule is an affine function of the iteration vector and the parameters**

$$\theta_{S1}(\vec{x}_{S1}) = \mathbf{t_{1_{S1}}}.i_{S1} + \mathbf{t_{2_{S1}}}.n + \mathbf{t_{3_{S1}}}.1$$
$$\theta_{S2}(\vec{x}_{S2}) = \mathbf{t_{1_{S2}}}.i_{S2} + \mathbf{t_{2_{S2}}}.j_{S2} + \mathbf{t_{3_{S2}}}.n + \mathbf{t_{4_{S2}}}.1$$

- For $-1 \leq t \leq 1$, there are $3^7 =$ **2187** possible schedules

One-Dimensional Scheduling

```
for (i=0; i<n; ++i) {
. s[i] = 0;
. for (j=0; j<n; ++j)
. . s[i] = s[i]+a[i][j]*x[j];
}
```

- **A schedule is an affine function of the iteration vector and the parameters**

$$\theta_{S1}(\vec{x}_{S1}) = \mathbf{t}_{\mathbf{1}_{\mathbf{S1}}}.i_{S1} + \mathbf{t}_{\mathbf{2}_{\mathbf{S1}}}.n + \mathbf{t}_{\mathbf{3}_{\mathbf{S1}}}.1$$
$$\theta_{S2}(\vec{x}_{S2}) = \mathbf{t}_{\mathbf{1}_{\mathbf{S2}}}.i_{S2} + \mathbf{t}_{\mathbf{2}_{\mathbf{S2}}}.j_{S2} + \mathbf{t}_{\mathbf{3}_{\mathbf{S2}}}.n + \mathbf{t}_{\mathbf{4}_{\mathbf{S2}}}.1$$

- For $-1 \leq t \leq 1$, there are $3^7 =$ **2187** possible schedules
- But **only 129 legal distinct schedules**

Our Objective

1. Search space construction
   - **Efficiently** construct a space of **all legal, distinct** affine schedules

Our Objective

1. Search space construction
   - **Efficiently** construct a space of **all legal, distinct** affine schedules

	matmult	locality	fir	h264	crout
$\vec{\imath}$-Bounds	$-1,1$	$-1,1$	$0,1$	$-1,1$	$-3,3$
c-Bounds	$-1,1$	$-1,1$	$0,3$	$0,4$	$-3,3$
#Sched.	1.9×10^4	5.9×10^4	1.2×10^7	1.8×10^8	2.6×10^{15}

Our Objective

1. Search space construction
   - **Efficiently** construct a space of **all legal, distinct** affine schedules

	matmult	locality	fir	h264	crout
$\vec{i}$-Bounds	$-1,1$	$-1,1$	$0,1$	$-1,1$	$-3,3$
c-Bounds	$-1,1$	$-1,1$	$0,3$	$0,4$	$-3,3$
#Sched.	1.9×10^4	5.9×10^4	1.2×10^7	1.8×10^8	2.6×10^{15}

$\Downarrow$

#Legal	6561	912	792	360	798

Our Objective

1. Search space construction
   - **Efficiently** construct a space of **all legal, distinct** affine schedules

	matmult	locality	fir	h264	crout
$\vec{\imath}$-Bounds	$-1,1$	$-1,1$	$0,1$	$-1,1$	$-3,3$
c-Bounds	$-1,1$	$-1,1$	$0,3$	$0,4$	$-3,3$
#Sched.	1.9×10^4	5.9×10^4	1.2×10^7	1.8×10^8	2.6×10^{15}

$$\Downarrow$$

#Legal	6561	912	792	360	798

   - Rely on the **polyhedral model** and Integer Linear Programming to **guarantee completeness and correctness** of the space properties

Our Objective

1. Search space construction
   - **Efficiently** construct a space of **all legal, distinct** affine schedules

	matmult	locality	fir	h264	crout
$\vec{i}$-Bounds	$-1,1$	$-1,1$	$0,1$	$-1,1$	$-3,3$
c-Bounds	$-1,1$	$-1,1$	$0,3$	$0,4$	$-3,3$
#Sched.	1.9×10^4	5.9×10^4	1.2×10^7	1.8×10^8	2.6×10^{15}

$\Downarrow$

#Legal	6561	912	792	360	798

   - Rely on the **polyhedral model** and Integer Linear Programming to **guarantee completeness and correctness** of the space properties
   - Search space will emcoumpass **unique, distinct compositions** of reversal, skewing, interchange, fusion, peeling, shifting, distribution

Our Objective

1. Search space construction
   - **Efficiently** construct a space of **all legal, distinct** affine schedules

	matmult	locality	fir	h264	crout
$\vec{\imath}$-Bounds	$-1,1$	$-1,1$	$0,1$	$-1,1$	$-3,3$
c-Bounds	$-1,1$	$-1,1$	$0,3$	$0,4$	$-3,3$
#Sched.	1.9×10^4	5.9×10^4	1.2×10^7	1.8×10^8	2.6×10^{15}

$\Downarrow$

#Legal	6561	912	792	360	798

   - Rely on the **polyhedral model** and Integer Linear Programming to **guarantee completeness and correctness** of the space properties
   - Search space will emcoumpass **unique, distinct compositions** of reversal, skewing, interchange, fusion, peeling, shifting, distribution

2. Search space exploration
   - Perform exhaustive scan to discover wall clock optimal schedule, and evidences of intricacy of the best transformation
   - Build an **efficient heuristic** to accelerate the space traversal

Polyhedral Representation of Programs

Static Control Parts

- Loops have affine control only

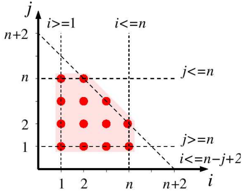
Polyhedral Representation of Programs

Static Control Parts

- Loops have affine control only
- Iteration domain: represented as integer polyhedra

```
for (i=1; i<=n; ++i)
. for (j=1; j<=n; ++j)
. . if (i<=n-j+2)
. . . s[i] = ...
```

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



Iteration domain of S_1

Polyhedral Representation of Programs

Static Control Parts

- Loops have affine control only
- Iteration domain: represented as integer polyhedra
- Memory accesses: static references, represented as affine functions of $\vec{x_S}$ and $\vec{p}$

```
for (i=0; i<n; ++i) {
. s[i] = 0;
. for (j=0; j<n; ++j)
. . s[i] = s[i]+a[i][j]*x[j];
}
```

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

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

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

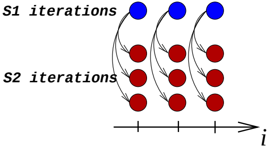
Polyhedral Representation of Programs

Static Control Parts

- Loops have affine control only
- Iteration domain: represented as integer polyhedra
- Memory accesses: static references, represented as affine functions of $\vec{x}_S$ and $\vec{p}$
- Data dependence between S1 and S2: a subset of the Cartesian product of $\mathcal{D}_{S1}$ and $\mathcal{D}_{S2}$ (**exact analysis**)

```
for (i=1; i<=3; ++i) {
. s[i] = 0;
. for (j=1; j<=3; ++j)
. . s[i] = s[i] + 1;
}
```

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



Polyhedral Representation of Programs

Static Control Parts

- Loops have affine control only
- Iteration domain: represented as integer polyhedra
- Memory accesses: static references, represented as affine functions of $\vec{x}_S$ and $\vec{p}$
- Data dependence between S1 and S2: a subset of the Cartesian product of $\mathcal{D}_{S1}$ and $\mathcal{D}_{S2}$ (**exact analysis**)
- Reduced dependence graph labeled by dependence polyhedra

Space Construction

Space Construction

Property (Causality condition for schedules)

Given $R\delta S$, θ_R *and* θ_S *are legal iff for each pair of instances in dependence:*

$$\theta_R(\vec{x_R}) < \theta_S(\vec{x_S})$$

Equivalently: $\Delta_{R,S} = \theta_S(\vec{x_S}) - \theta_R(\vec{x_R}) - 1 \geq 0$

Space Construction

- Causality condition
- **Farkas Lemma**

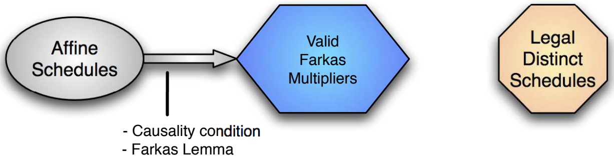
Lemma (Affine form of Farkas lemma)

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

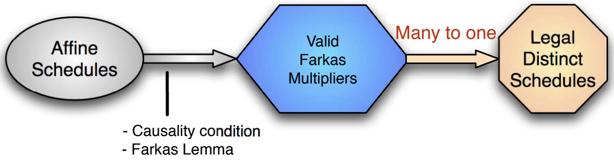
$$f(\vec{x}) = \lambda_0 + \vec{\lambda}^T(A\vec{x}+\vec{b}), \text{ with } \lambda_0 \geq 0 \text{ and } \vec{\lambda} \geq \vec{0}.$$

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

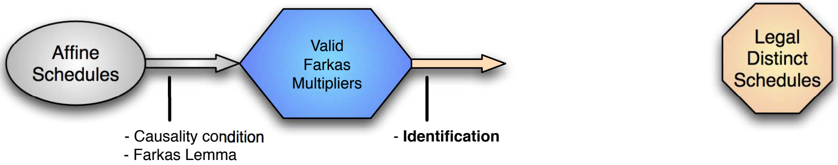
Space Construction



Space Construction



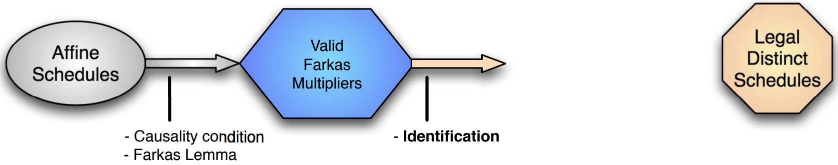
Space Construction



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

$$D_{R\delta S} \left\{ \begin{array}{lcl} \mathbf{i_R} & : & \lambda_{D_{1,1}} - \lambda_{D_{1,2}} + \lambda_{D_{1,3}} - \lambda_{D_{1,4}} \\ \mathbf{i_S} & : & -\lambda_{D_{1,1}} + \lambda_{D_{1,2}} + \lambda_{D_{1,5}} - \lambda_{D_{1,6}} \\ \mathbf{j_S} & : & \lambda_{D_{1,7}} - \lambda_{D_{1,8}} \\ \mathbf{n} & : & \lambda_{D_{1,4}} + \lambda_{D_{1,6}} + \lambda_{D_{1,8}} \\ \mathbf{1} & : & \lambda_{D_{1,0}} \end{array} \right.$$

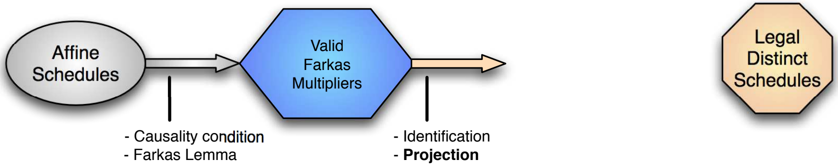
Space Construction



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

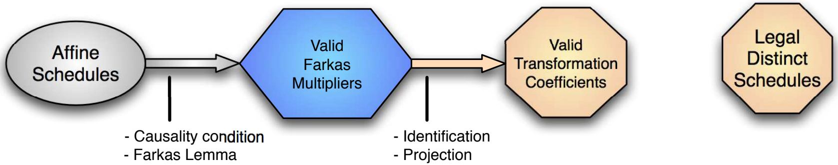
$$\left\{ \begin{array}{lllrcl} D_{R\delta S} & \mathbf{i_R} & : & -t_{1_R} & = & \lambda_{D_{1,1}} - \lambda_{D_{1,2}} + \lambda_{D_{1,3}} - \lambda_{D_{1,4}} \\ & \mathbf{i_S} & : & t_{1_S} & = & -\lambda_{D_{1,1}} + \lambda_{D_{1,2}} + \lambda_{D_{1,5}} - \lambda_{D_{1,6}} \\ & \mathbf{j_S} & : & t_{2_S} & = & \lambda_{D_{1,7}} - \lambda_{D_{1,8}} \\ & \mathbf{n} & : & t_{3_S} - t_{2_R} & = & \lambda_{D_{1,4}} + \lambda_{D_{1,6}} + \lambda_{D_{1,8}} \\ & \mathbf{1} & : & t_{4_S} - t_{3_R} - 1 & = & \lambda_{D_{1,0}} \end{array} \right.$$

Space Construction

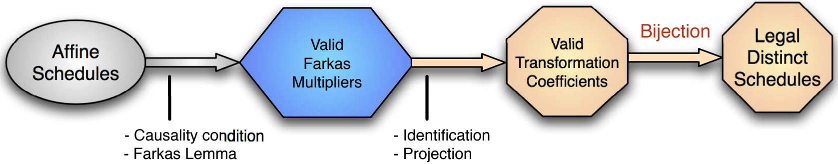


- Solve the constraint system
- Use (optimized) Fourier-Motzkin projection algorithm
  - Reduce redundancy
  - Detect implicit equalities

Space Construction



Space Construction



- One point in the space $\Leftrightarrow$ one set of legal schedules w.r.t. the dependence

Overview

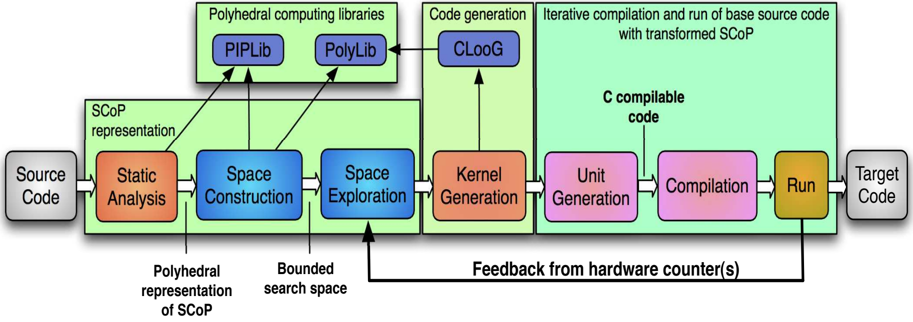
Algorithm

- Add constraints obtained for each dependence
- Bound the space
- Search space: set of linear constraints on the schedule coefficients (i.e. $\mathbb{Z}$-polytope)

- **To each integral point in the space corresponds a distinct program version where the semantics is preserved**

Benchmark	$\vec{i}$-Bounds	#Sched	#Legal	Time
`matmult`	$-1,1$	1.9×10^4	912	0.029
`locality`	$-1,1$	5.9×10^4	6561	0.022
`fir`	$0,1$	1.2×10^7	792	0.047
`h264`	$-1,1$	1.8×10^8	360	0.024
`crout`	$-3,3$	2.6×10^{15}	798	0.046

Workflow



- **CLooG**: `http://www.cloog.org`
- **PiPLib**: `http://www.piplib.org`
- **PolyLib**: `http://icps.u-strasbg.fr/polylib`

Performance Distribution [1/2]

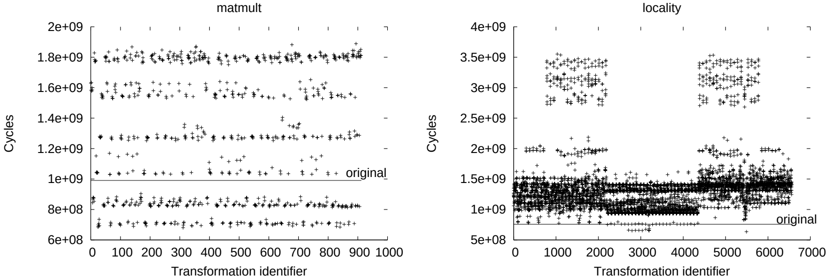
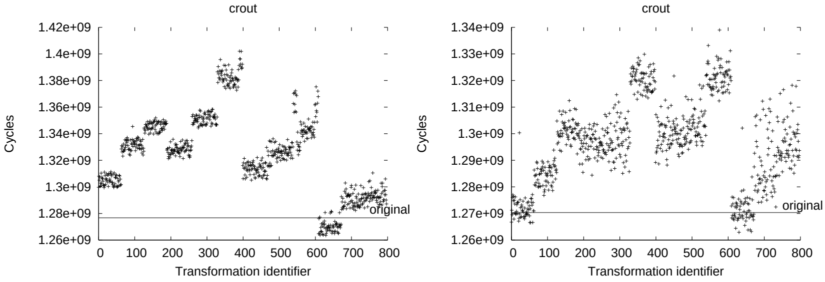


Figure: Performance distribution for `matmult` and `locality`

Performance Distribution [2/2]



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

Figure: The effect of the compiler

Performance Comparison

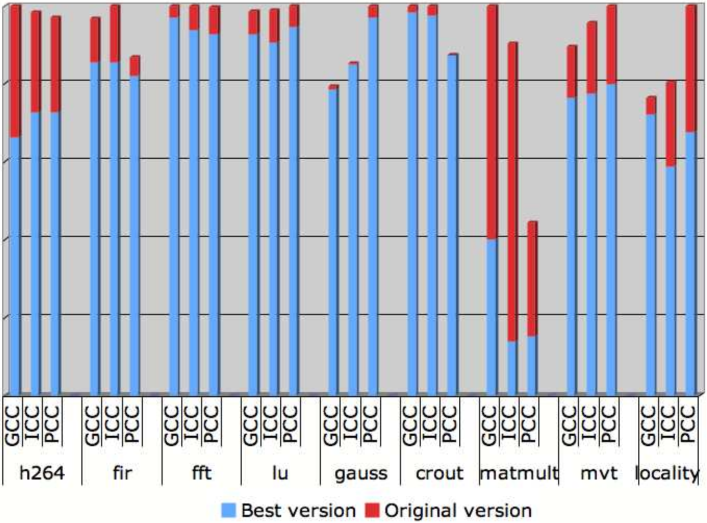


Figure: **Best Version vs Original**

Heuristic Scan

Propose a decoupling heuristic:

- The general "form" of the schedule is embedded in the iterator coefficients
- Decouple the schedule: $\theta_S(\vec{x}_S) = (\vec{\imath}\;\vec{p}\;c)\begin{pmatrix}\vec{x}_S\\ \vec{n}\\ 1\end{pmatrix}$

Heuristic Scan

Propose a decoupling heuristic:

- The general "form" of the schedule is embedded in the iterator coefficients
- Decouple the schedule: $\theta_S(\vec{x}_S) = (\vec{\imath}\;\vec{p}\;c)\begin{pmatrix}\vec{x}_S\\ \vec{n}\\ 1\end{pmatrix}$
- Parameters and constant coefficients can be seen as a refinement

Heuristic Scan

Propose a decoupling heuristic:

- The general "form" of the schedule is embedded in the iterator coefficients
- Decouple the schedule: $\theta_S(\vec{x}_S) = (\vec{\imath}\;\vec{p}\;c)\begin{pmatrix}\vec{x}_S\\ \vec{n}\\ 1\end{pmatrix}$
- Parameters and constant coefficients can be seen as a refinement

Adressing scalability to larger SCoPs:

1. impose a static or dynamic limit to the number of runs (limit to the $\vec{\imath}$ part)
2. replace an exhaustive enumeration of the $\vec{\imath}$ combinations by a limited set of random draws in the $\vec{\imath}$ space.

Results

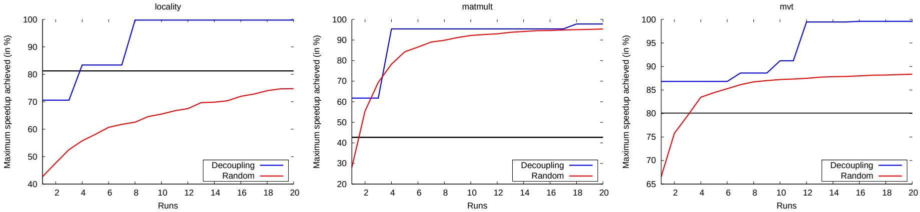
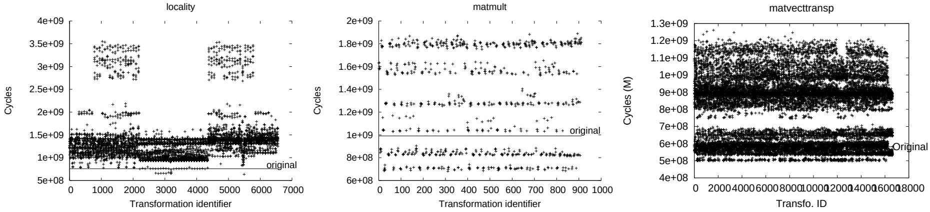


Figure: Comparison between random and decoupling heuristics



Conclusion

- **Optimizing and / or Enabling transformation framework on top of the compiler**
- Encouraging speedups, fast heuristic convergence
- On small kernels, **optimal transformation** can be discovered

Conclusion

- **Optimizing and / or Enabling transformation framework on top of the compiler**
- Encouraging speedups, fast heuristic convergence
- On small kernels, **optimal transformation** can be discovered

Conclusion

- **Optimizing and / or Enabling transformation framework on top of the compiler**
- Encouraging speedups, fast heuristic convergence
- On small kernels, **optimal transformation** can be discovered

Conclusion

- **Optimizing and / or Enabling transformation framework on top of the compiler**
- Encouraging speedups, fast heuristic convergence
- On small kernels, **optimal transformation** can be discovered

Ongoing and future work:

- Couple with state-of-the-art feedback-directed iterative methods
- Part II: multidimensional schedules
- Integrate into GCC GRAPHITE branch

Conclusion

- **Optimizing and / or Enabling transformation framework on top of the compiler**
- Encouraging speedups, fast heuristic convergence
- On small kernels, **optimal transformation** can be discovered

Ongoing and future work:

- Couple with state-of-the-art feedback-directed iterative methods
- Part II: multidimensional schedules
- Integrate into GCC GRAPHITE branch

Conclusion

- **Optimizing and / or Enabling transformation framework on top of the compiler**
- Encouraging speedups, fast heuristic convergence
- On small kernels, **optimal transformation** can be discovered

Ongoing and future work:

- Couple with state-of-the-art feedback-directed iterative methods
- Part II: multidimensional schedules
- Integrate into GCC GRAPHITE branch

Intricacy of the Transformed Code

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

```
S1: B[j] = A[j]
S2: C[j] = A[j + N]

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

```
for (c1=-N;c1<=min(-2,M-N);c1++)
  for (j=0;j<=M;j++)
    S1(c1+N,j);
for (c1=-1;c1<=M-N;c1++) {
  for (j=0;j<=M;j++)
    S2(c1+1,j);
  for (j=0;j<=M;j++)
    S1(c1+N,j);
}
for (c1=max(M-N+1,-1);c1<=M-1;c1++)
  for (j=0;j<=M;j++)
    S2(c1+1,j);
```

→ 19.4% speedup, without vectorization