A Note on the Performance Distribution of Affine Schedules

Louis-Noël Pouchet[1], Cédric Bastoul[1], John Cavazos[2] and Albert Cohen[1]

[1]ALCHEMY, INRIA Futurs / University of Paris-Sud XI, France
[2]Computer and Information Sciences, University of Delaware, USA

January 27, 2008

2nd Workshop on Statistical and Machine learning approaches to ARchitectures and compilaTion
Göteborg, Sweden

Outline

Motivation

- Automatic performance portability: iterative compilation
- Search space expressiveness → **bring the iterative optimization problem into the polyhedral model**

- Tradeoff expressiveness / traversal easiness
  - <u>Improve static characterization of the search space</u>
  - <u>Highlight dynamic properties</u>
  - Validate a dedicated heuristic to traverse the space

The Model

Original Schedule

```
 for (i = 0; i < n; ++i)
  for (j = 0; j < n; ++j){
S1: C[i][j] = 0;
   for (k = 0; k < n; ++k)
S2:  C[i][j] += A[i][k]*
                B[k][j];
  }
```

$$\Theta^{S1}.\vec{x}_{S1} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix} . \begin{pmatrix} \mathbf{i} \\ \mathbf{j} \\ n \\ 1 \end{pmatrix}$$

$$\Theta^{S2}.\vec{x}_{S2} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{pmatrix} . \begin{pmatrix} i \\ j \\ k \\ n \\ 1 \end{pmatrix}$$

```
for (i = 0; i < n; ++i)
 for (j = 0; j < n; ++j){
  C[i][j] = 0;
  for (k = 0; k < n; ++k)
   C[i][j] += A[i][k]*
              B[k][j];
 }
```

- Represent Static Control Parts (control flow and dependences must be statically computable)
- Use code generator (e.g. CLooG) to generate C code from polyhedral representation (provided iteration domains + schedules)

The Model

Original Schedule

```
 for (i = 0; i < n; ++i)
  for (j = 0; j < n; ++j){
S1: C[i][j] = 0;
   for (k = 0; k < n; ++k)
S2:  C[i][j] += A[i][k]*
                B[k][j];

 }
```

$$\Theta^{S1}.\vec{x}_{S1} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix} . \begin{pmatrix} i \\ j \\ n \\ 1 \end{pmatrix}$$

$$\Theta^{S2}.\vec{x}_{S2} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{pmatrix} . \begin{pmatrix} \mathbf{i} \\ \mathbf{j} \\ \mathbf{k} \\ n \\ 1 \end{pmatrix}$$

```
for (i = 0; i < n; ++i)
 for (j = 0; j < n; ++j){
  C[i][j] = 0;
  for (k = 0; k < n; ++k)
   C[i][j] += A[i][k]*
              B[k][j];

 }
```

- Represent Static Control Parts (control flow and dependences must be statically computable)
- Use code generator (e.g. CLooG) to generate C code from polyhedral representation (provided iteration domains + schedules)

The Model

Original Schedule

```
  for (i = 0; i < n; ++i)
   for (j = 0; j < n; ++j){
S1: C[i][j] = 0;
    for (k = 0; k < n; ++k)
S2:  C[i][j] += A[i][k]*
                B[k][j];

  }
```

$$\Theta^{S1}.\vec{x}_{S1} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix} . \begin{pmatrix} i \\ j \\ \mathbf{n} \\ 1 \end{pmatrix}$$

$$\Theta^{S2}.\vec{x}_{S2} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{pmatrix} . \begin{pmatrix} i \\ j \\ k \\ \mathbf{n} \\ 1 \end{pmatrix}$$

```
for (i = 0; i < n; ++i)
 for (j = 0; j < n; ++j){
  C[i][j] = 0;
  for (k = 0; k < n; ++k)
   C[i][j] += A[i][k]*
              B[k][j];

 }
```

- Represent Static Control Parts (control flow and dependences must be statically computable)
- Use code generator (e.g. CLooG) to generate C code from polyhedral representation (provided iteration domains + schedules)

The Model

Distribute loops

```
 for (i = 0; i < n; ++i)
  for (j = 0; j < n; ++j){
S1: C[i][j] = 0;
   for (k = 0; k < n; ++k)
S2:  C[i][j] += A[i][k]*
                B[k][j];

  }
```

$$\Theta^{S1}.\vec{x}_{S1} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix} . \begin{pmatrix} i \\ j \\ n \\ 1 \end{pmatrix}$$

$$\Theta^{S2}.\vec{x}_{S2} = \begin{pmatrix} 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{pmatrix} . \begin{pmatrix} i \\ j \\ k \\ n \\ 1 \end{pmatrix}$$

```
for (i = 0; i < n; ++i)
 for (j = 0; j < n; ++j)
  C[i][j] = 0;
for (i = n; i < 2*n; ++i)
 for (j = 0; j < n; ++j)
  for (k = 0; k < n; ++k)
   C[i-n][j] += A[i-n][k]*
                B[k][j];
```

- **All instances of S1 are executed before the first S2 instance**

The Model

Distribute loops + Interchange loops for S2

```
 for (i = 0; i < n; ++i)
  for (j = 0; j < n; ++j){
S1: C[i][j] = 0;
   for (k = 0; k < n; ++k)
S2:  C[i][j] += A[i][k]*
                B[k][j];

  }
```

$$\Theta^{S1}.\vec{x}_{S1} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix} . \begin{pmatrix} i \\ j \\ n \\ 1 \end{pmatrix}$$

$$\Theta^{S2}.\vec{x}_{S2} = \begin{pmatrix} 0 & 0 & \mathbf{1} & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ \mathbf{1} & 0 & 0 & 0 & 0 \end{pmatrix} . \begin{pmatrix} i \\ j \\ k \\ n \\ 1 \end{pmatrix}$$

```
for (i = 0; i < n; ++i)
 for (j = 0; j < n; ++j)
  C[i][j] = 0;
for (k = n; k < 2*n; ++k)
 for (j = 0; j < n; ++j)
  for (i = 0; i < n; ++i)
   C[i][j] += A[i][k-n]*
              B[k-n][j];
```

- **The outer-most loop for S2 becomes k**

The Model

Illegal schedule

```
  for (i = 0; i < n; ++i)
   for (j = 0; j < n; ++j){
S1:  C[i][j] = 0;
     for (k = 0; k < n; ++k)
S2:   C[i][j] += A[i][k]*
                  B[k][j];

  }
```

$$\Theta^{S1}.\vec{x}_{S1} = \begin{pmatrix} 1 & 0 & \mathbf{1} & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix} . \begin{pmatrix} i \\ j \\ n \\ 1 \end{pmatrix}$$

$$\Theta^{S2}.\vec{x}_{S2} = \begin{pmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{pmatrix} . \begin{pmatrix} i \\ j \\ k \\ n \\ 1 \end{pmatrix}$$

```
for (k = 0; k < n; ++k)
 for (j = 0; j < n; ++j)
  for (i = 0; i < n; ++i)
   C[i][j] += A[i][k]*
              B[k][j];
for (i = n; i < 2*n; ++i)
 for (j = 0; j < n; ++j)
  C[i-n][j] = 0;
```

- **All instances of S1 are executed after the last S2 instance**

The Model

A legal schedule

```
 for (i = 0; i < n; ++i)
  for (j = 0; j < n; ++j){
S1: C[i][j] = 0;
   for (k = 0; k < n; ++k)
S2:  C[i][j] += A[i][k]*
                B[k][j];

 }
```

$$\Theta^{S1}.\vec{x}_{S1} = \begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix} . \begin{pmatrix} i \\ j \\ n \\ 1 \end{pmatrix}$$

$$\Theta^{S2}.\vec{x}_{S2} = \begin{pmatrix} 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{pmatrix} . \begin{pmatrix} i \\ j \\ k \\ n \\ 1 \end{pmatrix}$$

```
for (i = n; i < 2*n; ++i)
 for (j = 0; j < n; ++j)
  C[i][j] = 0;
for (k= n+1; k<= 2*n; ++k)
 for (j = 0; j < n; ++j)
  for (i = 0; i < n; ++i)
   C[i][j] += A[i][k-n-1]*
              B[k-n-1][j];
```

- **Delay the S2 instances**
- Constraints must be expressed between Θ^{S1} and Θ^{S2}

The Model

Implicit fine-grain parallelism

```
 for (i = 0; i < n; ++i)
  for (j = 0; j < n; ++j){
S1: C[i][j] = 0;
   for (k = 0; k < n; ++k)
S2:  C[i][j] += A[i][k]*
                B[k][j];

  }
```

$$\Theta^{S1}.\vec{x}_{S1} = \begin{pmatrix} 1 & 0 & 0 & 0 \end{pmatrix} . \begin{pmatrix} i \\ j \\ n \\ 1 \end{pmatrix}$$

$$\Theta^{S2}.\vec{x}_{S2} = \begin{pmatrix} 0 & 0 & 1 & 1 & 0 \end{pmatrix} . \begin{pmatrix} i \\ j \\ k \\ n \\ 1 \end{pmatrix}$$

```
for (i = 0; i < n; ++i)
 pfor (j = 0; j < n; ++j)
  C[i][j] = 0;
for (k = n; k < 2*n; ++k)
 pfor (j = 0; j < n; ++j)
  pfor (i = 0; i < n; ++i)
   C[i][j] += A[i][k-n]*
              B[k-n][j];
```

- **Number of rows of $\Theta \leftrightarrow$ number of outer-most <u>sequential</u> loops**

The Model

Representing a schedule

```
 for (i = 0; i < n; ++i)
  for (j = 0; j < n; ++j){
S1: C[i][j] = 0;
   for (k = 0; k < n; ++k)
S2:  C[i][j] += A[i][k]*
                B[k][j];

  }
```

$$\Theta^{S1}.\vec{x}_{S1} = \begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix} . \begin{pmatrix} i \\ j \\ n \\ 1 \end{pmatrix}$$

$$\Theta^{S2}.\vec{x}_{S2} = \begin{pmatrix} 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{pmatrix} . \begin{pmatrix} i \\ j \\ k \\ n \\ 1 \end{pmatrix}$$

```
for (i = n; i < 2*n; ++i)
 for (j = 0; j < n; ++j)
  C[i][j] = 0;
for (k= n+1; k<= 2*n; ++k)
 for (j = 0; j < n; ++j)
  for (i = 0; i < n; ++i)
   C[i][j] += A[i][k-n-1]*
              B[k-n-1][j];
```

$$\Theta.\vec{x} = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} . \begin{pmatrix} i & j & i & j & k & n & n & 1 & 1 \end{pmatrix}^T$$

The Model

Representing a schedule

```
 for (i = 0; i < n; ++i)
  for (j = 0; j < n; ++j){
S1: C[i][j] = 0;
   for (k = 0; k < n; ++k)
S2:  C[i][j] += A[i][k]*
                B[k][j];

  }
```

$$\Theta^{S1}.\vec{x}_{S1} = \begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix} . \begin{pmatrix} i \\ j \\ n \\ 1 \end{pmatrix}$$

$$\Theta^{S2}.\vec{x}_{S2} = \begin{pmatrix} 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{pmatrix} . \begin{pmatrix} i \\ j \\ k \\ n \\ 1 \end{pmatrix}$$

```
for (i = n; i < 2*n; ++i)
 for (j = 0; j < n; ++j)
  C[i][j] = 0;
for (k= n+1; k<= 2*n; ++k)
 for (j = 0; j < n; ++j)
  for (i = 0; i < n; ++i)
   C[i][j] += A[i][k-n-1]*
              B[k-n-1][j];
```

$$\Theta.\vec{x} = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} . \big(\underbrace{\mathbf{i}\ \mathbf{j}\ \mathbf{i}\ \mathbf{j}\ \mathbf{k}}_{\vec{\imath}}\ \underbrace{\mathbf{n}\ \mathbf{n}}_{\vec{p}}\ \underbrace{\mathbf{1}\ \mathbf{1}}_{\mathbf{c}} \big)^T$$

The Model

Representing a schedule

```
 for (i = 0; i < n; ++i)
  for (j = 0; j < n; ++j){
S1: C[i][j] = 0;
   for (k = 0; k < n; ++k)
S2:  C[i][j] += A[i][k]*
               B[k][j];

 }
```

$$\Theta^{S1}.\vec{x}_{S1} = \begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix} . \begin{pmatrix} i \\ j \\ n \\ 1 \end{pmatrix}$$

$$\Theta^{S2}.\vec{x}_{S2} = \begin{pmatrix} 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{pmatrix} . \begin{pmatrix} i \\ j \\ k \\ n \\ 1 \end{pmatrix}$$

```
for (i = n; i < 2*n; ++i)
 for (j = 0; j < n; ++j)
  C[i][j] = 0;
for (k= n+1; k<= 2*n; ++k)
 for (j = 0; j < n; ++j)
  for (i = 0; i < n; ++i)
   C[i][j] += A[i][k-n-1]*
              B[k-n-1][j];
```

	Transformation	Description
$\vec{\imath}$	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
$\vec{p}$	fusion	Fuses two loops, a.k.a. jamming
	distribution	Splits a single loop nest into many, a.k.a. fission or splitting
c	peeling	Extracts one iteration of a given loop
	shifting	Allows to reorder loops

The Search Space

Challenges

- Completeness (combinatorial problem)
- Scalability (large integer polyhedra computation)

Proposed solution

- Philosophically close to Feautrier's maximal fine-grain parallelism
- One point in the space $\Leftrightarrow$ one distinct legal program version
- Bound schedule coefficients in $[-1, 1]$ to limit control overhead
- No completeness, but decent scalability
- Deliver a mechanism to automatically complete / correct schedules

The Hypothesis

Extremely large generated spaces: $> 10^{30}$ points

$\longrightarrow$ we must leverage static characteristics to build traversal mechanisms

Hypothesis:

- **It is possible to statically order the impact on performance of transformation coefficients, that is, decompose the search space in subspaces where the performance variation is maximal or reduced**

- **The more a schedule dimension impacts a performance distribution, the more it is constrained**

`DCT` benchmark

- 32x32 Discrete Cosine Transform, 5 statements, 35 dependences
- 2 imperfectly nested loops
- 3 sequential schedule dimensions outputted

Schedule dimension	$\vec{\imath}$	$\vec{\imath}+\vec{p}$	$\vec{\imath}+\vec{p}+c$
Dimension 1	39	66	471
Dimension 2	729	19683	531441
Dimension 3	60750	1006020	64855485
Total combined	1.7×10^9	1.3×10^{12}	1.6×10^{16}

Figure: Search Space Statistics for `dct`

DCT benchmark

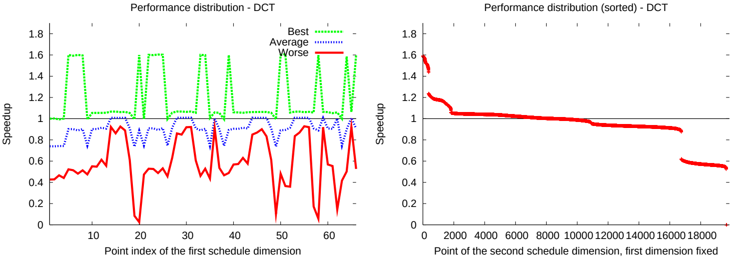
- 32x32 Discrete Cosine Transform, 5 statements, 35 dependences
- 2 imperfectly nested loops
- 3 sequential schedule dimensions outputted

Schedule dimension	$\vec{\imath}$	$\vec{\imath}+\vec{p}$	$\vec{\imath}+\vec{p}+c$
Dimension 1	39	66	471
Dimension 2	729	19683	531441
Dimension 3	60750	1006020	64855485
Total combined	1.7×10^9	1.3×10^{12}	1.6×10^{16}

Figure: Search Space Statistics for dct

- Search space analyzed: $66 \times 19683 = 1.29 \times 10^6$ different legal program versions (arbitrary compositions of skewing, reversal, interchange, fusion, distribution)

Performance Distribution [1/2]

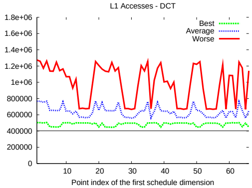


(a) Representatives for each point of Θ_1 (b) Raw performance of each point of Θ_2, for the best value for Θ_1

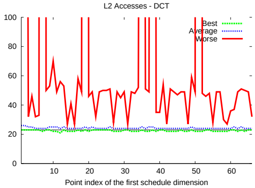
Figure: Performance Distribution for DCT

- Only 0.14% of analyzed points achieve at least 80% of the speedup
- **Θ_1 is a good discriminant for performance**
- **Variance analysis shows $\vec{\imath} > \vec{p} > \vec{c}$**

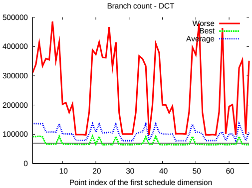
Performance Distribution [2/2]



(a) L1 Accesses



(b) L2 Accesses



(c) Branch Count

Figure: Hardware Counters Distribution for DCT

- L1 Accesses captures the performance distribution shape
- Branch count shows control overhead introduced
- **Origin of performance improvement is opaque most of the time**
  - Interaction with the compiler (trigger optimizations)
  - Better use of processor features

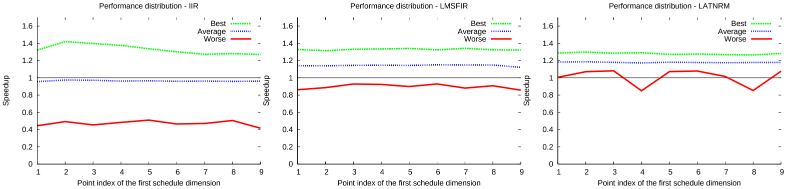
Search Space Statistics

Benchmark	# St.	# Deps.	# Dim.	$\vec{\imath}$	$\vec{\imath}+\vec{p}$	$\vec{\imath}+\vec{p}+c$
latnrm	11	75	3	1	9	27
fir	4	36	2	1	9	18
lmsfir	9	112	2	1	9	27
iir	8	66	3	1	9	18

Figure: Search Space Statistics

- Only one sequence of interchange + skewing + reversal possible for the outer-most loop
- Highly constrained benchmark: side effect of the search space construction algorithm
- Search space must be computed to detect the pattern

Performance Distribution



(a) `iir` (b) `lmsfir` (c) `latnrm`

Figure: Performance Distribution for 3 UTDSP benchmarks

- Significant speedup to discover
- **Performance distribution is almost flat**
- **Final variance analysis confirm the base hypothesis**

Results of the Decoupling Heuristic

- Capitalize on the performance distribution ordering: propose a decoupling heuristic mechanism
- Principle: Iterate first on the most performance impacting coefficients, use a completion algorithm for the non-explored coefficients

	dct	matmult	lpc	edge-c2d	iir	fir	lmsfir	latnrm
#Inst.	5	2	12	3	8	4	9	11
#Loops	6	3	7	4	2	2	3	3
i	39	76	243	1	1	1	1	1
Space	1.6×10^{16}	912	$> 10^{25}$	5.6×10^{15}	$> 10^{19}$	9.5×10^{7}	2.8×10^{8}	$> 10^{22}$
Id Best	46	16	489	11	34	33	51	6
Speedup	57.1%	42.87%	31.15%	5.58%	37.50%	40.24%	30.98%	15.11%

Figure: Heuristic Performance for AMD Athlon

- **Near space optimal speedup discovered in at most 51 runs for SCoPs of less than 10 statements**

Conclusion

Properties of the search space

- "Classical" transformations usually associated to specific schedule coefficients
- Classes of schedule coefficients $(\vec{\imath}, \vec{p}, c)$ map into subspaces ordered w.r.t performance variation
- Schedule rows map into subspaces ordered w.r.t. performance
- Very low density of the best transformations (0.xx%)

Application

- **Partition the optimization space to narrow the search**
- Motivate a heuristic traversal leveraging these characteristics
- Validated on Intel x86_32, AMD x86_64, embedded MIPS32 (Au1500), embedded VLIW (ST231)

Ongoing Work

- **Scalability** Use genetic algorithm traversal for the larger SCoPs
  - Legality preserving operators
- **Expressiveness** Integrate tiling by means of permutability constraints
  - New (static/dynamic) properties of the search space
- **Parallelism** Express coarse-grain parallelism thanks to tiling
  - New search algorithm