Iterative Optimization in the Polyhedral Model: Part II, Multidimensional Time

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

[1] **ALCHEMY group, INRIA Saclay / University of Paris-Sud 11, France**
[2] **Dept. of Computer & Information Sciences, University of Delaware, USA**

June 9, 2008

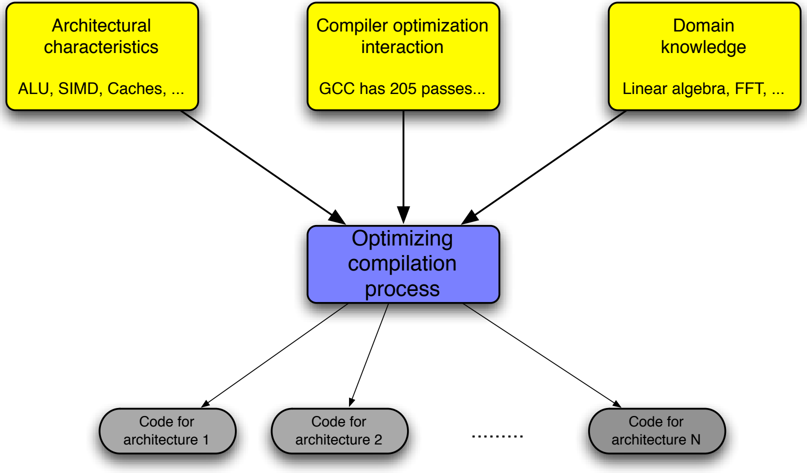
ACM SIGPLAN 2008 Conference on
Programming Languages Design and Implementation
Tucson, Arizona

Motivation

- New architecture → New high-performance libraries needed
- **New architecture → New optimization flow needed**
- Architecture complexity/diversity increases faster than optimization progress
- **Traditional approaches lose performance portability…**

We want a portable optimization process!

The Optimization Problem



Architectural characteristics

ALU, SIMD, Caches, ...

Compiler optimization interaction

GCC has 205 passes...

Domain knowledge

Linear algebra, FFT, ...

Optimizing compilation process

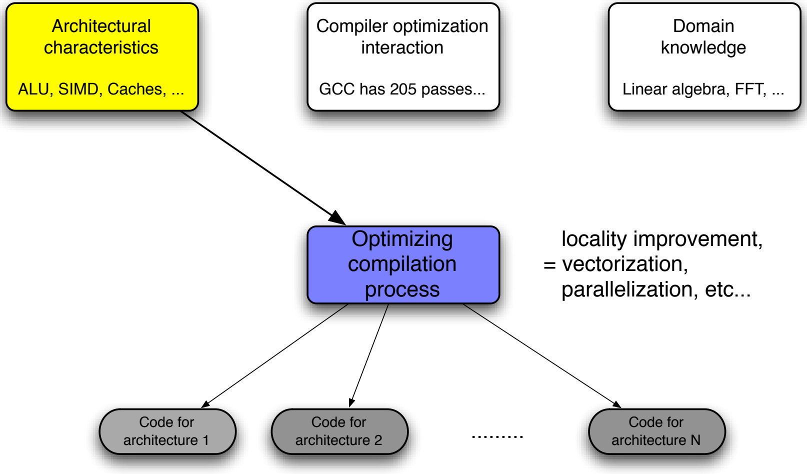
Code for architecture 1

Code for architecture 2

....

Code for architecture N

The Optimization Problem



Architectural characteristics

ALU, SIMD, Caches, ...

Compiler optimization interaction

GCC has 205 passes...

Domain knowledge

Linear algebra, FFT, ...

Optimizing compilation process

= locality improvement, vectorization, parallelization, etc...

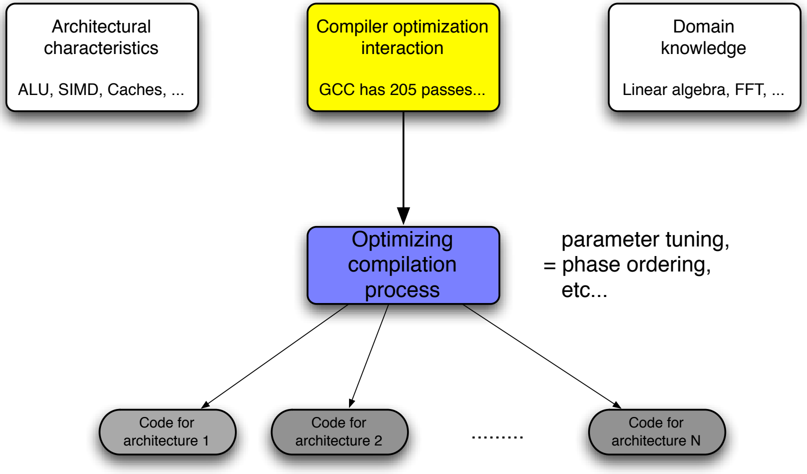
Code for architecture 1

Code for architecture 2

....

Code for architecture N

The Optimization Problem



Architectural characteristics

ALU, SIMD, Caches, ...

Compiler optimization interaction

GCC has 205 passes...

Domain knowledge

Linear algebra, FFT, ...

Optimizing compilation process

= parameter tuning, phase ordering, etc...

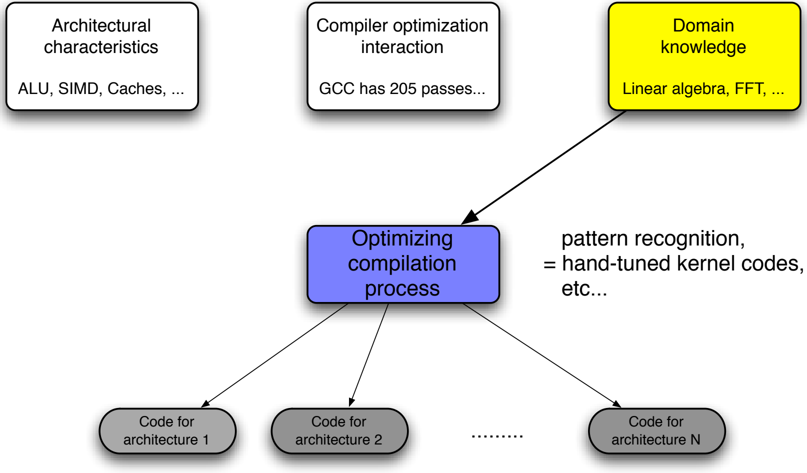
Code for architecture 1

Code for architecture 2

....

Code for architecture N

The Optimization Problem



Architectural characteristics

ALU, SIMD, Caches, ...

Compiler optimization interaction

GCC has 205 passes...

Domain knowledge

Linear algebra, FFT, ...

Optimizing compilation process

pattern recognition,
= hand-tuned kernel codes,
etc...

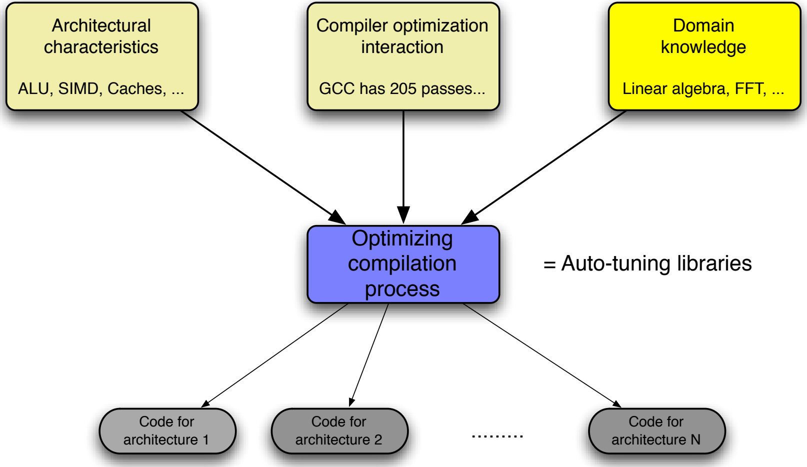
Code for architecture 1

Code for architecture 2

....

Code for architecture N

The Optimization Problem



Architectural characteristics

ALU, SIMD, Caches, ...

Compiler optimization interaction

GCC has 205 passes...

Domain knowledge

Linear algebra, FFT, ...

Optimizing compilation process

= Auto-tuning libraries

Code for architecture 1

Code for architecture 2

....

Code for architecture N

The Optimization Problem

Architectural characteristics

ALU, SIMD, Caches, ...

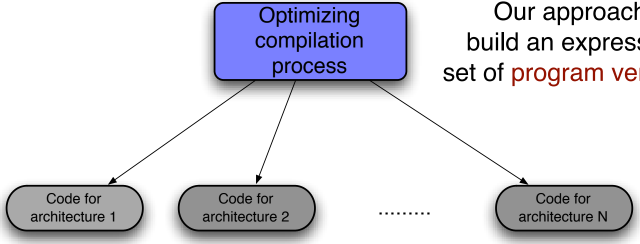
Compiler optimization interaction

GCC has 205 passes...

Domain knowledge

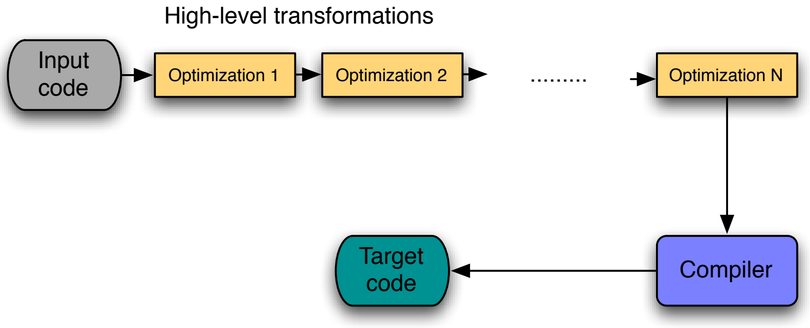
Linear algebra, FFT, ...

In reality, there is a complex interplay between all components



Our approach:
build an expressive
set of program versions

Iterative Optimization Flow



High-level transformations

Input code

Optimization 1

Optimization 2

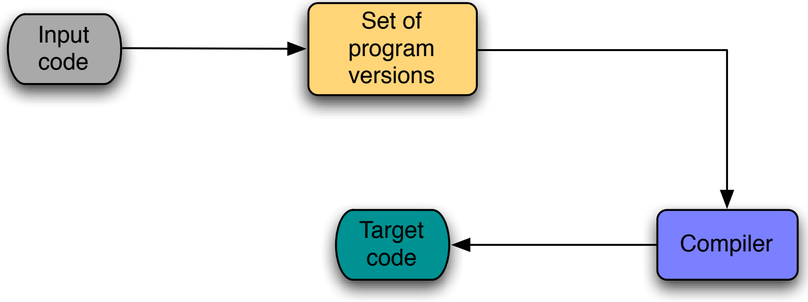
....

Optimization N

Compiler

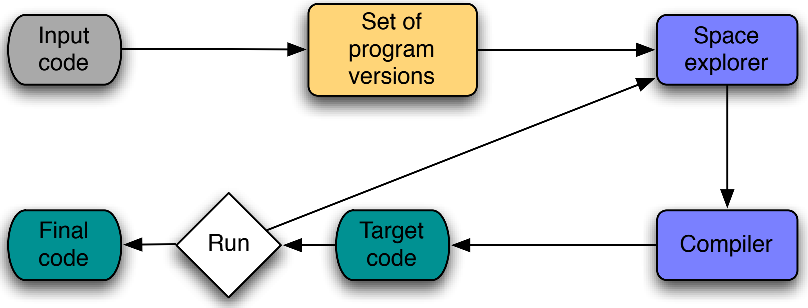
Target code

Iterative Optimization Flow



Program version = result of a sequence of loop transformation

Iterative Optimization Flow



Program version = result of a sequence of loop transformation

Set of Program Versions

What matters is the **result of the application of optimizations**, not the optimization sequence

All-in-one approach:

- **Legality:** semantics is always preserved
- **Uniqueness:** all versions of the set are distinct
- **Expressiveness:** a version is the result of an arbitrarily complex sequence of loop transformation

The Polyhedral Model in a Nutshell

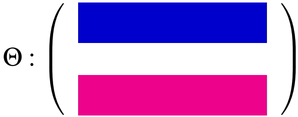
- Arbitrarily complex sequence of loop transformations are modeled in a **single optimization step**: new scheduling matrix
- Granularity: each executed instance of each statement

```
for (i = ...; i < ...; ++i)
  S1(i);
for (i = ...; i < ...; ++i)
  S2(i);
```

- **First row → all outer-most loops**

The Polyhedral Model in a Nutshell

- Arbitrarily complex sequence of loop transformations are modeled in a **single optimization step**: new scheduling matrix
- Granularity: each executed instance of each statement



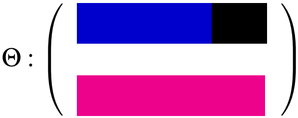
```
for (i = ...; i < ...; ++i)
 for (j = ...; j < ...; ++j)
  S1(i,j);

for (i = ...; i < ...; ++i)
 for (j = ...; j < ...; ++j)
  S2(i,j);
```

- **Second row → all next outer-most loops**

The Polyhedral Model in a Nutshell

- Arbitrarily complex sequence of loop transformations are modeled in a **single optimization step**: new scheduling matrix
- Granularity: each executed instance of each statement



```
for (j = ...; j < ...; ++j)
 S2(...,j);
for (i = ...; i < ...; ++i)
 for (j = ...; j < ...; ++j)
  S1(i,j);
  S2(i,j);
```

- **Minor change → significant impact**

The Polyhedral Model in a Nutshell

- Arbitrarily complex sequence of loop transformations are modeled in a **single optimization step**: new scheduling matrix
- Granularity: each executed instance of each statement

$$\Theta : \begin{pmatrix} \vec{\imath} & \vec{p} & c \\ \vec{\imath} & \vec{p} & c \end{pmatrix}$$

```
for (j = ...; j < ...; ++j)
 S2(...,j);
for (i = ...; i < ...; ++i)
 for (j = ...; j < ...; ++j)
  S1(i,j);
  S2(i,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

Previous Contributions

Previous work (CGO'07, *Part I, One-Dimensional Time*):

- Focus on Static Control Parts (SCoP)
  - SCoP: Consecutive set of statements with affine control flow
- Complete framework for one-dimensional schedules
- Efficient search space construction, efficient traversal

- Drawbacks in applicability
- Drawbacks in expressiveness

We previously solved a simpler problem...

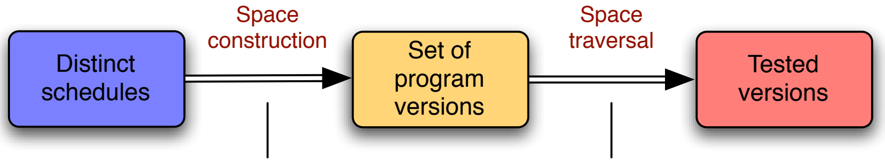
New Contributions

Dealing with multidimensional schedules:

- **Applicability on any Static Control Parts**
- Increased expressiveness
- **Design of scalable traversal methods**
    - Dedicated genetic algorithm
    - Dedicated heuristic

Deeper In The Method

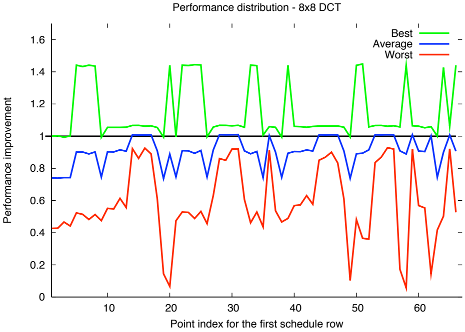
Multidimensional schedules: high expressiveness, complex problem



- **combinatorial** expression of **legality**
- **heuristic needed:** greedy selection of dependences + ordering (see *Some Efficient Solutions to the Affine Scheduling Problem, Part II: Multidimensional Time*, Feautrier, 1992)
- Code generation friendly bounds on the schedule coefficients

- **multiple** polytopes to traverse
- **large** and expressive spaces (up to 10^{50})
- partial enumeration (mandatory): **completion mechanism**+ subspace partitioning
- shape the space: **optimized polytope projection** (required) + constrained dynamic scan

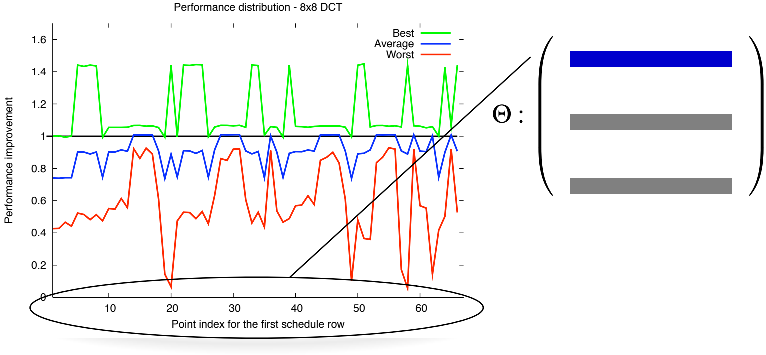
Observations on the Performance Distribution



```
for (i = 0; i < M; i++)
  for (j = 0; j < M; j++) {
    tmp[i][j] = 0.0;
    for (k = 0; k < M; k++)
      tmp[i][j] += block[i][k] *
                    cos1[j][k];
 }
for (i = 0; i < M; i++)
  for (j = 0; j < M; j++) {
    sum2 = 0.0;
    for (k = 0; k < M; k++)
      sum2 += cos1[i][k] * tmp[k][j];
      block[i][j] = ROUND(sum2);
 }
```

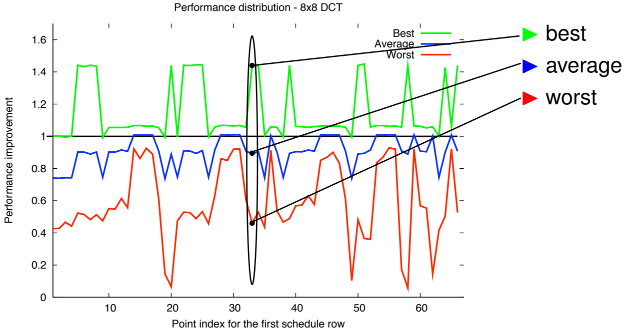
- Extensive study of 8x8 Discrete Cosine Transform (UTDSP)
- Search space analyzed: $66 \times 19683 = 1.29 \times 10^6$ different legal program versions

Observations on the Performance Distribution



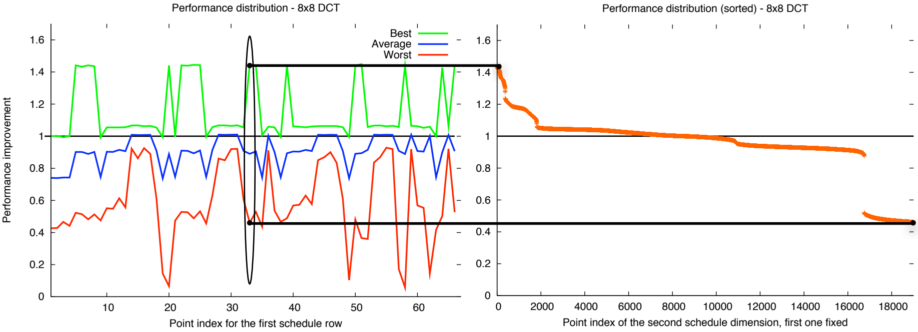
- Extensive study of 8x8 Discrete Cosine Transform (UTDSP)
- Search space analyzed: $66 \times 19683 = 1.29 \times 10^6$ different legal program versions

Observations on the Performance Distribution



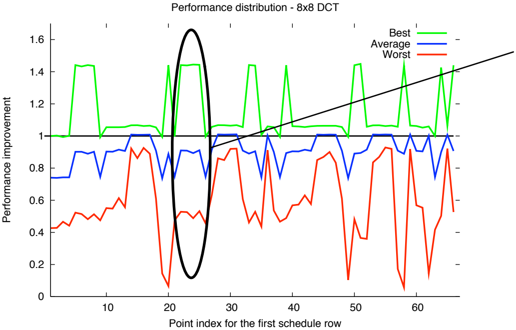
- Take one specific value for the first row
- Try the 19863 possible values for the second row

Observations on the Performance Distribution



- Take one specific value for the first row
- Try the 19863 possible values for the second row
- Very low proportion of best points: $< 0.02\%$

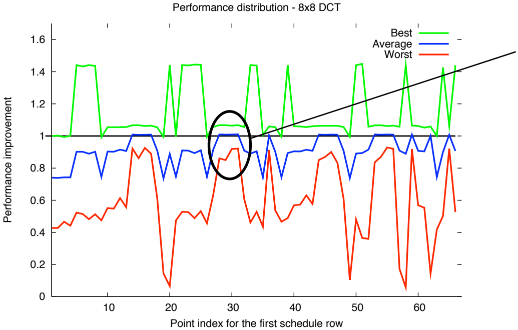
Observations on the Performance Distribution



Large **performance variation**

- Performance variation is large for good values of the first row

Observations on the Performance Distribution



Small **performance variation**

- Performance variation is large for good values of the first row
- It is usually reduced for bad values of the first row

Scanning The Space of Program Versions

The search space:

- Performance **variation** indicates to partition the space
- **Non-uniform distribution of performance**
- No clear analytical property of the optimization function

→ Build dedicated **heuristic** and **genetic operators** aware of these **static and dynamic characteristics**

Dedicated Heuristic

- **Multidimensional version** of the heuristic presented in Part I
- Discover 80%+ of the performance improvement in less than 50 runs for small kernels

- **Feedback directed**, yet deterministic
- Leverages our knowledge about performance distribution
- **Relies on the completion algorithm** to instantiate the full version

- But unsatisfactory results for larger programs...

Dedicated GA Operators

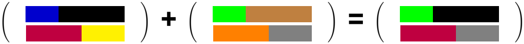
Mutation

- Performance distribution is not uniform
- Tailored to focus on the most promising subspaces
- Preserves legality (closed under affine constraints)

Cross-over

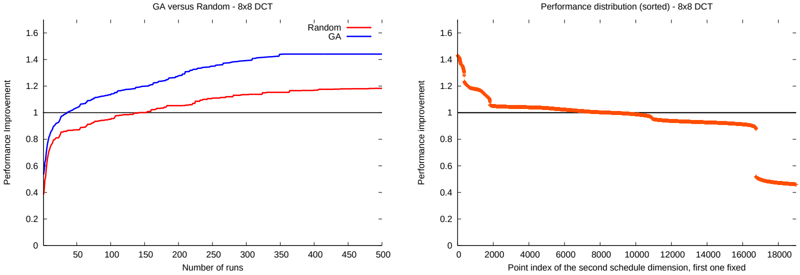
- Row cross-over

- Column cross-over



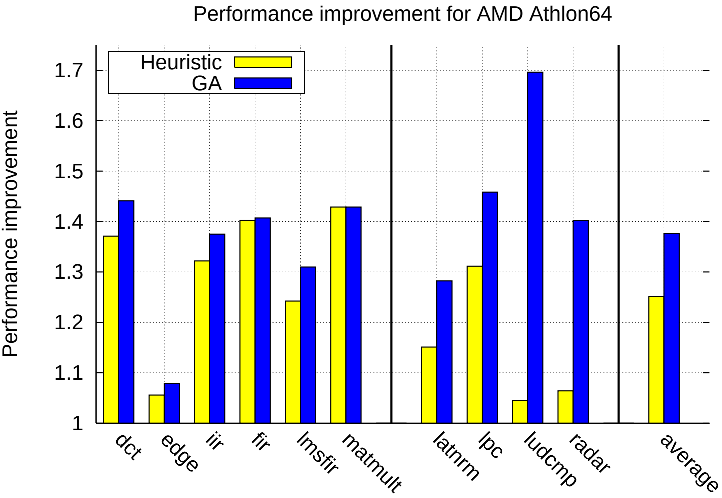
- Both preserve legality

Dedicated GA Results



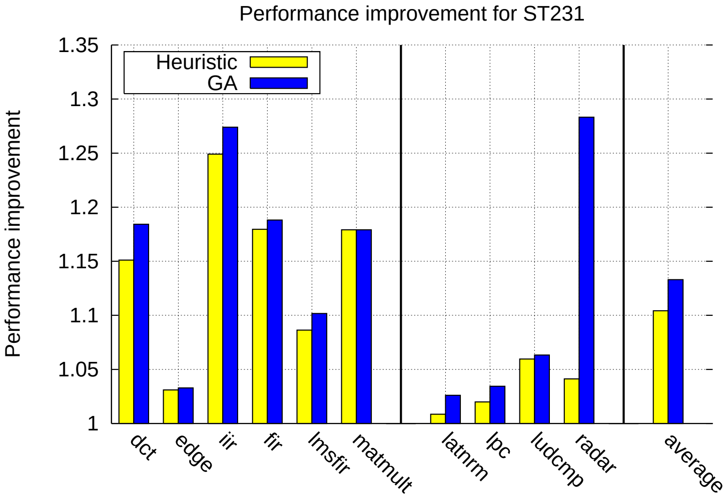
- GA converges towards the maximal space speedup

Experimental Results [1/3]



baseline: gcc -O3 -ftree-vectorize -msse2

Experimental Results [2/3]



baseline: st200cc -O3 -OPT:alias=restrict -mauto-prefetch

Experimental Results [3/3]

Looking into details (hardware counters+compilation trace):

- **Better activity** of the processing units
- Best version may **vary significantly for different architectures**
- Different source code may **trigger different compiler optimizations**

→ **Our method is a portable optimization process**

Conclusion

- Scalable algorithms (GA and heuristic) to traverse the space, with dedicated pruning and search strategies

- Part I + Part II: applicability observed on various compilers (GCC, ICC, Open64) and architectures (x86_32, x86_64, MIPS32, ST231 VLIW)

- **Tunable framework:** open to other search space construction strategies

- Take-home message:
  - All-in-one: **legality, uniqueness, expressiveness**
  - **Generic and portable approach** for high-level transformation selection

Tunuing: Distribute and Tile

- Focus on fuse/distribute legality affine constraints (presented algorithm with additional constraints)
- Use PLuTo as a tiling / parallel backend
- **Driven by program versions**
- Excellent performance gains (research report coming soon...)