LetSee: the LEgal Transformation SpacE Explorator

Louis-Noël Pouchet,[*1] Cédric Bastoul[*1] and Albert Cohen[*1]

[*] *ALCHEMY group, INRIA Futurs and LRI / University of Paris-Sud XI, France*

ABSTRACT

The increasing complexity of modern architectures and memory models challenges the design of optimizing compilers. It is mandatory to perform several optimizing transformations of the original program to exploit the machine to its best, especially for scientific, computational intensive codes.

We propose a complete framework to address the problem of affine control loop optimization, based on the polyhedral model. Our tool chain takes benefits of iterative compilation, through the design of a search space encompassing only legal and distinct affine transformations of a program.

KEYWORDS: Iterative optimization, polyhedral model, affine scheduling, program transformation

1 Introduction

High level optimizations are mandatory to take advantage of the full capabilities of a given architecture and its associated memory model. But due to the rigidity and the fragility of compilers, discovering good transformation *sequences* is problematic. To overcome this difficulty, we rely on the *polyhedral model* [3], a powerful algebraic representation of programs where arbitrarily complex sequences of transformations are represented as a single function.

But automatically finding the best optimizing transformations is still a hard task, even when focusing on affine control blocks: 1) usual static models fail to capture the complexity of modern architecture and memory; 2) the intricate link between the source code (e.g. how data is allocated), the compiler (e.g. the fragility and conservativeness of optimization heuristics) and the target architecture is far from being amenable to modeling. Iterative compilation (that is, running a program candidate on the target architecture and checking its behavior) is typically designed to address these issues.

We propose a complete framework for *iterative optimization* in the polyhedral model. We use the algebraic properties of the polyhedral model to build and devise properties on a search space encompassing only *legal*, *distinct* program versions, and we use iterative optimization techniques to search for a best program version within this space.

2 Polyhedral Representation of Programs

Only parts of the program, called *Static Control Parts* (SCoP), can be represented algebraically in the polyhedral model. Roughly, a SCoP is a maximal set of consecutive instructions such that: the only allowed surrounding control structures are `for` loops and `if` conditionals, loop bounds, conditionals and array accesses are affine functions of the surrounding loop iterators and the global parameters. In such a program class, semantic information can be represented as polyhedra of integer points [3]. SCoPs are known to capture a large portion of the computation time of scientific and signal processing applications [4].

[1]E-mail: firstname.lastname@inria.fr

The following example illustrates how, with a *schedule* function[2] θ (that is, an order for the execution of all instances of each statement, which is applied to their iteration domain), one can achieve complex transformations within a single expression. The code on the right is produced by a *code generator* [2], which was provided the iteration domains of R and S[3], and the two scheduling functions θ_R and θ_S. The reader may verify that this transformation is optimal for locality.

```
    for (i = 0; i < n; i++)
    |  for (j = 0; j < n; j++) {
R   |  |  b[j] = a[j];
S   |  |  c[j] = a[j + m];
    |  }
```

$$\theta_R(\vec{x}_R) = \begin{pmatrix} j \\ i \end{pmatrix}$$

$$\theta_S(\vec{x}_S) = \begin{pmatrix} j - m \\ i \end{pmatrix}$$

```
    for (j = -m; j < min(0,n-m); j++)
    |  for (i = 0; i < n; i++)
S   |  |  c[j] = a[j + m];
    for (j = 0; j < n-m; j++)
    |  for (i = 0; i < n; i++) {
R   |  |  b[j] = a[j];
S   |  |  c[j] = a[(j+m) + m];
    |  }
    for (j = max(n-m,0); j < n; j++)
    |  for (i = 0; i < n; i++)
R   |  |  b[j] = a[j];
```

3 An Iterative Tool Chain

The `LetSee` software is a subpart of a larger tool chain, depicted in Figure 1. This tool chain relies on several third-party softwares: `Candl`, a dependence analyser for the polyhedral model; `CLooG`[4], a program generator taking as input a polyhedral representation of a program; `PIPLib` and `Polylib`[5], two libraries dedicated to (integer) polyhedra computation; and a compiler (which has to support the `C` language).

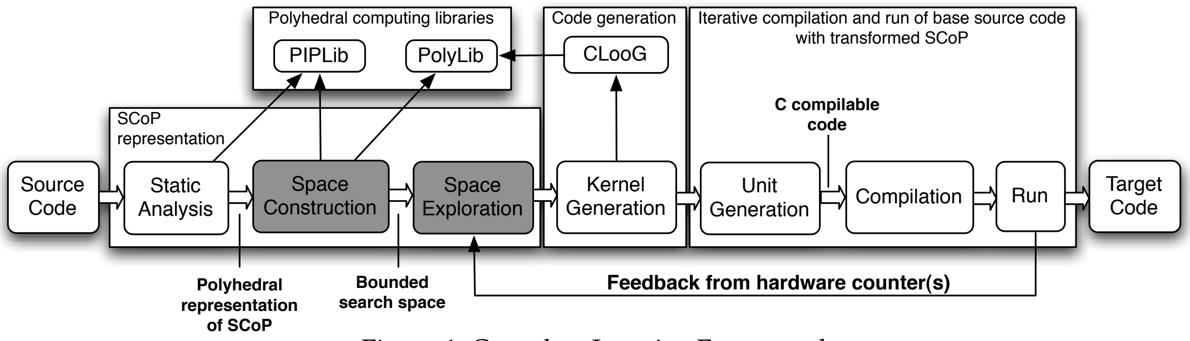


Figure 1: Complete Iterative Framework

The *static analysis* step isolates SCoPs and represent its information as integer polyhedra (iteration domain of each statement, access functions, and instancewise dependence graph). The remainder of the program and the actual statement operations are kept apart.

The `LetSee` core is composed of the two next boxes, depicted in gray: a search space encompassing only legal program versions is computed from the polyhedral representation of the SCoP, and filled to a feedback-directed space exploration algorithm. This algorithm operates only on polyhedra, where each integer point represent a different program version where the semantics is preserved.

From each point, one is able to regenerate the SCoP code (the *kernel generation* step), and then to reinsert all the remainder of the original program plus the mandatory instrumentation[6] for performance feedback (the *unit generation* step).

[2]A schedule of a statement S is $\theta_S(\vec{x}_S) = T.(\vec{x}_S\ \vec{n}\ 1)^t$, with T a constant matrix, $\vec{x}_S$ the statement iterators, $\vec{n}$ the global parameters. θ_S associate a multidimensional timestamp to each executed instance of S.
[3]Iteration domain: the set of instances executed, here it is defined by the $\mathbb{Z}$-polyhedron $\{i, j \mid 0 \leq i, j < n\}$
[4]`http://www.cloog.org`
[5]`http://www.piplib.org`, `http://icps.u-strasbg.fr/polylib`
[6]`LetSee` uses hardware counters to collect the most accurate information on the program behavior

The last step stresses the motivation of this tool chain: eventually, a regular `C` unit file is generated, and can be compiled by any optimizing compiler targeting any architecture. The tight link between the compiler optimizations (including those not amenable with the polyhedral representation) and the source program is though inserted in the iterative process.

The benefits of this approach are threefold: it performs aggressive program transformations guaranteed to be legal, in a high-level algebraic representation, allowing to express many more transformations inaccessible to traditional compilers; the optimization factor is highly tunable: one can optimize w.r.t. the number of cycles, of vectorized instructions, of cache misses, etc. just by mean of feedback from hardware counters; and it is open to various complementary techniques: typically compiler parameters tuning or machine learning algorithms for space exploration [1].

4 Experimental Results

Narrowing the Search Space The upstream characterization of legal *and* distinct schedules in a search space dramatically contributes to reduce its size, and eventually its complexity. To the best of our knowledge, no other iterative compilation approaches rely on these two fundamental properties, and instead face the bottleneck of either identical or inapplicable transformations, which explodes the space size.

Table 2 shows results for a few kernels, mostly extracted from UTDSP. For each kernel, we report the set of distinct schedules of a fixed Dimension (actually the minimal sequential dimension needed). Starting from All of them, inserting the legality criterion narrows the space to Legal. More, we have shown in [5] that many schedule coefficients with a large impact on the space size have in fact a low impact on performance, and the space can be narrowed to Iterators when removing those coefficients[7]. Eventually, the search space is amenable to exhaustive or heuristic traversals, leading to the reported Speedup on AMD Athlon64.

Benchmark	Statements	Dependences	Dimension	All	Legal	Iterators	Speedup
`locality`	1	2	1	5.9×10^4	6561	9	19%
`matmult-250`	2	7	1	1.9×10^4	912	76	243%
`compress-1024`	6	56	2	6.2×10^{24}	6480	9	368%
`edge-2048`	3	30	3	1.7×10^{24}	3.1×10^7	1467	40%
`latnrm-256`	11	75	2	4.1×10^{18}	1.9×10^9	678	32%
`lmsfir-256`	9	112	2	1.2×10^{19}	2.6×10^9	19962	22%

Figure 2: Search Space Size

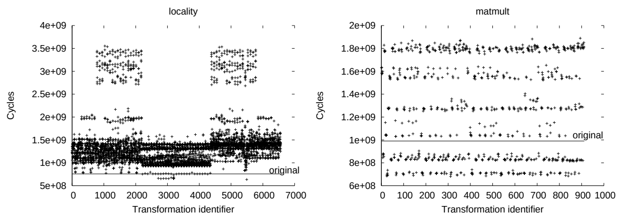
Various Traversal Approaches In some cases (typically those with a one-dimensional schedule), it is possible to exhaustively traverse the space of all legal, distinct schedules. Figure 3(a) shows the performance distribution for two typical cases. Computing these distributions let us make several observations and devise a very efficient Decoupling Heuristic for the one-dimensional case [5] (see Figure 3(b) for a comparison with a Random approach).

In the general case of multidimensional schedules, designing an heuristic is far more complex. There is an intricate link between the different dimensions of the schedule: some transformations may need more than one dimension to be expressed[8] (e.g. interchange, tiling).

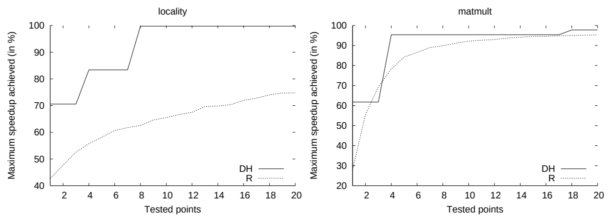
Our preliminary approach consists in traversing the Iterators set, and completing the schedule with legal values as close as possible to 0. This has been proven to be very efficient, and we are currently investigating machine learning techniques to accelerate the traversal.

[7]Each point in this set has different values for the iterator coefficients, and is computed by projecting the $\mathbb{Z}$-polyhedral space representing the set of legal schedules to the subspace containing only the schedule coefficients attached to iterators

[8]The problem arises because the legal scheduling space of a program is by definition represented as distinct sets, one for each schedule dimension. Inter-dimension constraint knowledge is *de facto* mandatory.



(a) Performance distribution



(b) Decoupling Heuristic convergence vs. Random

Figure 3: Search Space Traversal

5 Conclusion

In the iterative compilation approach, rapidly selecting the best transformation involves two orthogonal problems: building a search space containing it (that is, being complete), and efficiently traversing it.

For the case of one-dimensional affine schedules, a complete method exists [5] and is implemented in `LetSee`. For the general case of multidimensional schedules, it is merely impossible in general to build the set of *all* schedules, due to the intrinsic combinatorics of the construction method,[9] thus making extremely difficult to guarantee completeness.

In `LetSee`, the space traversal problem is equivalent to generate all integer points in a polyhedron. Although this problem has been circumvented in the well known context of code generation [2], we face the total lack of applicability of these techniques to the size of our problems: we deal with the traversal of polyhedra of hundreds of dimensions, where any single operation as intersection or redundancy elimination can take minutes or more. To overcome this issue, we developed an efficient algorithm based on the dynamic generation of points[10] which pick a value for each coefficient sequentially in a fixed order.

Iterative and empirical search techniques are one of our last hope to harness the complexity of modern processors and compilers. The `LetSee` framework is a complete approach to address the problem of affine loop nest optimization; and while we have demonstrated the potential of our techniques on several kernels, we are investigating major improvements in terms of traversal speed and scalability to make our platform applicable to the largest possible set of programs.

References

[1] F. Agakov, E. Bonilla, J. Cavazos, B. Franke, G. Fursin, M. F. P. O'Boyle, J. Thomson, M. Toussaint, and C. K. I. Williams. Using machine learning to focus iterative optimization. In *Fourth IEEE/ACM International Symposium on Code Generation and Optimization*, pages 295–305, Washington, DC, USA, 2006.

[2] C. Bastoul. Code generation in the polyhedral model is easier than you think. In *PACT'13 IEEE International Conference on Parallel Architecture and Compilation Techniques*, pages 7–16, Juan-les-Pins, september 2004.

[3] P. Feautrier. Some efficient solutions to the affine scheduling problem. Part II. Multidimensional time. *Int. J. Parallel Program.*, 21(5):389–420, 1992.

[4] S. Girbal, N. Vasilache, C. Bastoul, A. Cohen, D. Parello, M. Sigler, and O. Temam. Semi-automatic composition of loop transformations for deep parallelism and memory hierarchies. *Intl. J. of Parallel Programming*, 34(3), 2006.

[5] L.-N. Pouchet, C. Bastoul, A. Cohen, and N. Vasilache. Iterative optimization in the polyhedral model: Part I, one-dimensional time. In *International Symposium on Code Generation and Optimization*, pages 144–156, San Jose, California, March 2007. IEEE Computer Society.

[9] The legal space depends on the way to select the dependences to solve at a given depth, which is a combinatorial decision problem. Attempts to solve it as an Integer Linear Program fail to be complete [3]

[10] This algorithm makes a massive usage of partial $\mathbb{Q}$-polyhedral projection thanks to an optimized Fourier-Motzkin algorithm, and a constant (i.e. at each step) Gaussian elimination, to compact the manipulated spaces