ANR proposal: CoDaS : Complex Data-structure Scheduling

Coordinator: Laure Gonnord
Appel à projet générique 2017, instrument “jeune chercheur”, défi 7

Avril 2017

Abstract

The context of the proposal is the increasing need for methods and tools that are able to deal with the massive parallelism now faced by the programmers. So far, methods and tools based on the polyhedral model (a powerful representation for capturing the flow and the data of a program at the same time) suffer from their lack of expressivity.

The objective of the proposal is to invent a way to reason about more general programs and data structures (e.g., trees). We will explore adapting and extending the existing efficient scheduling techniques that have been developed since the advent of the polyhedral model. The ambition is to develop a new framework with new ideas that may come from other communities such as abstract interpretation, and rewriting systems.

Contents

Partner	Last Name	First Name	Current Position	Involvement (person.month)	Domain of expertise, contribution
ENS Lyon/LIP	Gonnord	Laure	MCF Univ. Lyon1	**Scientific Coordinator**, 33.6 PM (80% of the **research time**)	Coordinator of the whole project, **in charge of Tasks 2.2 and 3**, static analysis for safety and optimization, semantics, (sequential) termination proofs.
Insa Lyon/CITI	Morel	Lionel	MCF Insa	Collaborator, 12.6 PM (30%)	**In charge of Tasks 1 and 2.1** Profiling, semantics, runtime.
Inria RA/LIP	Alias	Christophe	CR Inria	Collaborator, 8.4 PM (20%)	Polyhedral model, scheduling, communication optimization.
Inria Rennes/IRISA	Yuki	Tomofumi	CR Inria	Collaborator, 8.4 PM (20%)	**In charge of Task 4.2** (Scheduling part). Polyhedral model, scheduling, compiler optimization.
Inria RA	Rastello	Fabrice	DR Inria	Collaborator, 6.3 PM (15%)	Code generation, extraction of benchmarks, experimental evaluation.
Birkbeck Univ. of London	Fuhs	Carsten	Lecturer	Collaborator, 6.3 PM (15%)	**In charge of Task 4.1** (Termination part). Rewriting, semantics, termination proofs, static analysis.

Table 1: Involvement of the researchers in the project

1 Context, objective of the proposition, positioning

1.1 Context

The rise of embedded systems and high performance computers generated new problems in high-level code optimization, especially for loops, both for optimizing embedded applications and for transforming programs for high-level synthesis (HLS). Moreover, everything involving data storage is of prime importance as it impacts power consumption, performance, and for hardware, chip area. Thus, there is an increasing need for better scheduling techniques for all kinds of programs, especially those manipulating a huge amount of data.

This general context puts this project at the intersection of the two axes that are "intensive computations for numerical simulation" (axis 6) and "software science and technology" (axis 3) of the 7th challenge "information and communication".

So far, the polyhedral model [FL11], a framework introduced in the late eighties, has been successfully applied to a range of these compilation problems, such as (semi-)automatic parallelization and code generation [Fea92a] or data movement optimization [DI15]. However, this powerful model hits its limit as soon as we are faced with irregular programs (general `while` loops, unpredictable conditions). As a consequence, these powerful techniques have, for the moment, only seen their use in limited (but still important) niches, because of the intrinsic restrictions.

The long term objective is to give a general way to reason about and manipulate programs with general control flow and complex data structures. The means we choose is to start with the current state of the polyhedral model. We are planning to enhance the framework theoretically and algorithmically in order to be able to deal with more general programs. For the period of the current proposal, we are planning to investigate general control flow and focus on specific data structures, such as trees.

The "instrument" (Jeune Chercheur) we choose for funding this project is particularly adapted since this proposal is only the first step toward the long-term objective.

1.2 Summary of the scientific goals of the project

One key strength of the polyhedral model that has led to its success is its intermediate representation, where control structures and data accesses are *succinctly* represented by polyhedra and affine mappings. When it comes to analysis/transformation of irregular (non-affine) programs, several issues arise for the polyhedral model:

- How to "abstract" a general program with the polyhedral representation? i.e., how to handle data, functions, how conservative must the abstraction be?
- How do we schedule (find an execution order for the statements to be executed, and for the data to be read/written) the obtained unified representation?
- How do we generate the program from this new unified representation and the new schedule?

As a specific objective of this project, we will explore how to handle complex control flow arising from various data structures. Programs operating on data structures such as lists or trees often have regular control flow which is not adequately captured by the polyhedral model. We call these programs with "hidden" regularities **covertly regular programs**.

The ambition of the proposal is to develop a new framework based on the the polyhedral model and its extensions for this class of programs. Moreover, we intend to preserve the main advantages of the polyhedral model: a single mathematical object for representing both program and data, endowed with powerful algorithmic techniques for reasoning about it.

We plan to incorporate new ideas coming from other domains of static analysis:

- From the abstract interpretation community: control flow, approximations, and also shape analysis: abstract interpretation is one of the major techniques to approximate the behavior of programs.
- From the termination community: rewriting is one of the major techniques that are able to handle complex data structures and also recursive programs.

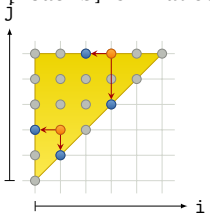
The main objective of this project is to design a general framework for reasoning about covertly regular programs operating on complex data structures, in order to efficiently schedule and parallelize them. For this purpose we shall use emerging ideas from three communities (compilation, static analysis, and rewriting). We expect cross-fertilization of these topics.

1.3 Bibliography: related work

The topics of interest for our project are broad since we want to combine different techniques to enhance both applicability and scalability of the polyhedral model. In this section we make a summary of the abundant literature on polyhedral scheduling, the first attempts to extend its applicability, and also some previous work of interest within the community of termination of rewriting systems. [1]

The polyhedral model: a fundamental tool to reason about regular programs The polyhedral model is a collection of techniques developed around a common intermediate representation of programs: integer polyhedra. Such a mathematical representation of programs inherits nice structural properties. For instance, when loop transformations are represented as affine functions, compositions of transformations are also affine functions due to their closure properties.

The polyhedral representation was linked to loop programs by an analysis proposed by Feautrier [Fea91]. This analysis provides exact dependence analysis information where statement instances (i.e., statements executed at different loop iterations) and array elements are distinguished. The exact dependence information obtained through this analysis and the use of linear programming techniques to explore the space of legal schedules [Fea92b] is what constitutes the base of the polyhedral model for loop transformations.




```
for (i = 0; i < N; i++)
   for (j = i; j < N; j++)
S:    A[j] = f(A[i], A[j]);
```

$$\text{Domain}(S) = \{i, j \mid 0 \le i \le j < N\}$$
$$\text{Read}(S \mapsto \texttt{A}) = (i, j \to i); (i, j \to j)$$
$$\text{Write}(S \mapsto \texttt{A}) = (i, j \to i)$$
$$\text{Dep}(S \mapsto S) = \begin{cases} (i, j \to i, i) & : i < j \\ (i, j \to i-1, j) & : 0 < i \\ \textit{corner case} & : \cdots \end{cases}$$

Figure 1: An example of polyhedral representation. Loop nests that fit the polyhedral model can be viewed as mathematical (constraint-based) objects, which can also be visualized geometrically.

Figure 1 illustrates the polyhedral representation with an example. The statement S is executed approximately $\frac{N^2}{2}$ times during the execution of this loop. The triangular region expressed as a set of constraints, called the *domain* of S, represents this set of dynamic execution instances. Accesses to array A from each of these statement instances can be succinctly captured through affine functions of the loop iterators. The dependences are also expressed as a function between two statement instances. [2] The key insight is that although the specific dependences at a statement instance may differ, they are all captured by a function due to the regularity in the control flow. The figure illustrates dependences for two instances, where one of the dependences $(i, j \to i, i)$ has different lengths depending on the instance.

The "traditional" use of polyhedral techniques in optimizing compilers typically focuses on loop transformations. PLuTo [BHRS08] is a now widely used push-button tool for automatically parallelizing polyhedral loop nests that tries to optimize locality in addition to parallelization. There is also significant work in data layout optimization for polyhedral programs where analyses are performed to minimize the memory requirement [DSV05]. Polyhedral techniques for loop transformations are now adopted by many production level compilers, such as GCC, IBM XL, and LLVM.

[1] Of course, other communities have different approaches to similar objectives of optimizing applications, we only describe here work that is the most closely related to our project. We refer to additional sources of inspiration in Section 2.

[2] There are a number of ways to represent dependences in the polyhedral model. Here, we represent them as a function from consumer instances to producer instances.

Recently, polyhedral techniques are being applied to many different areas besides loop transformations. One natural application of automatic parallelization techniques is in verification of given parallelizations where the tools take parallelized programs as inputs, and use polyhedral analysis to guarantee the absence of parallel bugs [BYR+11, YFRS13]. Another application of scheduling techniques is in synthesis of ranking functions for proving program termination [ADFG10].

Limits of the polyhedral model, extensions Although the polyhedral model provides strong analysis capabilities with many different applications, its main limitation is its applicability. The program must have regular control flow, and in addition, it has to be fully affine. Specifically, loop bounds, array accesses, and `if` conditions must be affine functions of the surrounding loop iterators and runtime constants. As an example, FFT (Fast Fourier Transform) has a regular control flow, but cannot be represented with the polyhedral model because it is not fully affine. [3]

Some attempts exists to extend the expressiveness of the polyhedral model, namely:

- Sparse Polyhedral Framework [VSHS14]. The SPF is an extension to the polyhedral model to deal with sparse computation. Sparse matrices are represented using indirect array accesses, making polyhedral analysis impossible, and an inspector/executor strategy is used to perform optimization at runtime. The SPF represents runtime optimizations (such as data re-ordering) by introducing uninterpreted function symbols to the polyhedral model. Similar approaches exploit the regularities of the data input. While their work also extends the polyhedral framework to complex data structures and control-flow, we seek to extend in another direction with more emphasis on static analysis and scheduling.
- Extensions to `while` loops and data-dependences. Benadberrahmane et al. [BPCB10] and Rajopadhye et al. [RGK11] have developed techniques for extending the polyhedral framework to include `while` loops and data-dependent controls. Their main contributions are (i) extension of the algebraic representation of programs, and (ii) code generation from the extended representation. The schedule being explored remains affine, and our work will complement their work to enrich the framework as a whole.
- Rational transducers. In an early work on this subject, Feautrier [Fea98] suggests automatic parallelization for recursive programs operating on tree-shaped data structures via rational transducers. A limitation of the approach is that it does not allow for object creation at runtime, a limitation that is not present in our approach.
- Hybrid Approaches. Another approach is to mix static and dynamic analysis. Rus et al. [RRH03, RHAR06] have proposed techniques to test for absence of dependences at run-time to parallelize loops. Jimborean et al. [JCD+14] check for program regions at run-time, and optimize regions that were found to be polyhedral using run-time information. Barthou et al. [BCF97] propose a method to compute exactly direct flow dependences for arbitrary programs, the behavior of dynamic control (while loops, non-affine array accesses, pointers, etc.) being hidden into fuzzy parameters whose evaluation is left to execution. This work has set the basis of a formal framework for hybrid analysis: what information is static, what information is dynamic, which predicate to evaluate.
- Language Approaches. In the domain of High-Performance Computing, a large number of domain-specific languages (DSL)/libraries/tools have been developed to cope with the complex communication patterns of highly parallel applications. For regular programs, intensive effort to generate efficient code has been made on some kernels such as FFT (Spiral[4], FFTW[5]), and stencil computations [CSB11, TCK+11, BGK+12]. Then, algorithm recognition approaches [AB03, IAR14] can be helpful to let the compiler use automatically such libraries when the computation kernels are regular. In contrast, pure language approaches let the programmer express the parallelism with an adequate abstraction, and then rely on run-time systems to dynamically schedule tasks [MGR+12, ATNW09, GBP07, BBD+13]. These techniques are designed to deal with programs that are not at all static/regular at compile-time, in contrast to the hidden regularity that we seek to take advantage of. Although polyhedral techniques are applicable to stencil computations, FFT is currently out of the scope. Our ambition is to develop a framework for capturing a wider range of programs that are still statically analyzable, in addition to the typical class of programs that fit the polyhedral model (e.g., stencils, linear algebra, dynamic programming), to algorithms based on trees and lists, which should include FFT as well.

[3] Although being able to automatically optimize FFT is our long term challenging objective, it is out of the scope of the CoDaS project.
[4] `http://www.spiral.net/`
[5] `http://www.fftw.org/`

Extending the polyhedral model for `while` loop programs: a first success During the past five years, Laure Gonnord and Christophe Alias have with other collaborators gained experience in computing schedules from general control flow graphs [ADFG10]. This work is important since, in essence, it is the initial inspiration for the project.

Through the exploration of techniques to transform kernels with `while` loops into kernels with static loop bounds (to make them acceptable for high-level-synthesis tools), the team encountered the problem of counting the number of iterations of `while` loops and thus also their termination.

The main idea of this work was initially to explore the link between scheduling and termination [Dar10, Dar11], through making a parallel between schedules and ranking functions. A schedule can be described as a function that associates (logical) dates with operations that is strictly increasing from some starting date (say, 0). Similarly, a ranking function associates a non-negative expression to each computation of the program that strictly decreases each time the program takes a transition (e.g., change in loop iterators). The main idea was to use the previous work on multidimensional polyhedral loop scheduling [Fea92b], but in the larger scope of general control flow graphs (instead of static `for` loops) and programs operating on numerical variables.

This work was further extended to deal with larger programs [AAG12], and scalability concerns were one of the reasons that developed into a practical study [RAPG14]. More recently, we showed how to drastically decrease the size of the constraint systems we solve while computing ranking functions [GMR15] using ideas from the static analysis community (namely, counter-example guided refinement). Although the programs addressed in this paper have complex control, the complex data structures are abstracted away by a coarse abstraction. For instance, write operations are ignored, and read operations are assumed to return random values.

Other approaches: a quick tour of the termination community As we mentioned previously, the problem of scheduling is closely linked to the well-studied topic of termination. This motivates the investigation of the benefits we may encounter from the rewriting community:

- Automated termination analysis for *term rewrite systems (TRSs)* [BN98] is a well-studied topic and has given rise to several powerful, fully automatic, tools over the last years (e.g., AProVE, Mu-Term, TTT2). TRSs are particularly suitable to represent complex data structures, such as lists or trees.
- Recently, two-stage approaches to termination analysis, harnessing the power of termination tools for TRSs also for programming languages, have been proposed [GRS$^+$11, OBvEG10, GSS$^+$12]. In the first stage, symbolic execution and abstraction on programming language level are used to over-approximate all possible program executions. From this program analysis, one then extracts a TRS whose termination implies the termination of the original program. In the second stage, termination of this TRS is analyzed with existing tools.
  This approach is currently limited mainly by scalability of the program analysis front-ends.
- Recently Fuhs et al. [FGP$^+$09] proposed an extension of term rewriting by *built-in predefined integers*, based on the observation that termination, in particular for imperative programs, often depends on integer variables. This allows existing techniques for proving termination on integers to be combined with techniques for term rewriting. Tools for analysis of term rewriting are now successfully applied for termination analysis (e.g., for Java [OBvEG10, BMOG12] and C programs [SGB$^+$17]).
  This approach also suffers from scalability issues that will prevent its use in actual compilers.

In summary, the rewriting community has a long expertise in analyzing complex data structures, but the approach is still not mature enough to be embedded into compilers. However, the underlying techniques will be our main source of inspiration to reason about such data.

1.4 Challenges, innovation, stumbling blocks

In this project, we propose to investigate how existing techniques from static analysis (and verification) and rewriting can be used in the context of polyhedral analysis to extend it to more complex data structures (we do not want to abstract our programs in TRSs). As far as we know, despite the quite abundant research work in the subject, this is the first time this multidisciplinary solution is explored.

The main difficulty we may encounter is the fact that there is no explicit computation of a scheduling in the state-of-the-art rewriting-based termination methods. Moreover, our application domain (compilation) imposes us some scalability issues that are less under consideration in the static analysis (and termination) communities. We might have to make some compromise between precision and scalability.

These challenges make a significant leap beyond the scope of current polyhedral analysis. Enlarging the data-parallelism expressed in the polyhedral model will lead to many new opportunities, even beyond the scope of this project.

Despite extensive prior research on the general topic of scheduling, there is huge room for improvement. We propose to revisit a more fundamental problem by defining a framework to reason properly about data and control at the same time.
The novelty in this project is in the combination of existing scheduling techniques (from the compilation community) with other techniques coming both from static analysis and from term rewriting, applied to the general problem of efficient compilation and scheduling of programs. As far as we know, this **multidisciplinary project** is the first attempt to combine rewriting techniques and scheduling.

1.5 National and International positioning

High performance computing and Polyhedral model The polyhedral model, as a formalism for loop analyses and transformation, has most of its experts located in France. Among many existing teams and projects, we cite PIPS (analysis, source-to-source), PARKAS (tasks extraction), CAMUS (speculative loop transformations), COMPSYS (analysis, high-level-synthesis), CAIRN (high-level-synthesis), SYNCHRONE (verification) and CORSE (hybrid analysis/transformations). Most of them actively participate in the development of polyhedral libraries (e.g., PolyLib, ISL, PIP, Aspic).

Among them, PARKAS, CAMUS, and CORSE address the problem of handling legacy code by developing techniques to detect regularities not explicitly exposed by the language:

- PARKAS has several contributions that combine run-time systems, data-flow languages, and code transformations. The goal being to:1) exploit the data-flow model to enhance run-time efficiency (for schedule, synchronization and memory management); 2) exploit compiler ability to coarsen (either statically or dynamically) granularity whenever possible. The PARKAS team shares our vision that the future gain of performance via static compilation will be allowed by reasoning on *intermediate representations* or on language expressiveness, like the activity we develop in Section 2.2.2
- CAMUS develops APOLLO, a compiler framework that performs dynamic and speculative loop transformations. They exploit two ideas:1) detect affine memory accesses for codes that do not necessarily have access functions that are affine; 2) split the iteration space into chunks expecting sub-parts to show affine behavior (while the full space does not necessarily). This technique is linked to our work since it can deal with programs with lists; however, it cannot be easily extended to non-regular accesses like in tree traversals, which are under consideration in this project.
- CORSE has two research directions related to CoDaS. The first one aims at developing new hybrid (static/dynamic) analysis and transformation techniques. This direction exploits the following idea: detect affine dependences for codes that do not necessarily have memory accesses that are affine (more general than 1 of CAMUS but do not handle partially affine code as for 2 of CAMUS). The second research direction aims at combining affine with irregular dimensions (that can be enumerated either statically or dynamically in an inspector/executor way). The team is currently considering other iterators than standard numerical ones (such as topological order of tree, ordered list, unordered set, etc.) as Abstract Data Types, for optimization purposes, following the pioneer work of A. Cohen [Coh99]. This last activity is closely linked with the one described in Section 2.2.2.

Streaming languages The research on streaming languages is also related to the CoDaS project. There are two main trends in this area, not necessarily in the polyhedral context:

- The definition of static streaming languages, where there is a focus on the ability to compute schedules during compilation (here the scheduling activity is more coarse-grained than the one addressed by CoDaS, as they schedule *tasks*). Among them we can cite Σ-C (from CEA, [GSLD11]) that implements the Cyclo-Static DataFlow model (CSDF [EL94]). This kind of languages usually suffers from a lack of expressiveness, especially when the programmer aims to define some regularities at the interfaces (I/Os), or during communication of data (for instance, in Σ-C there is no simple way to express that some data will be used more than once). Thus the expressiveness of such languages may benefit from the CoDaS project while still remaining fully static.

- Some other teams/people focus more on the expressiveness of the language itself. The OpenStream language [PC13], introduced by Antoniu Pop, is one the most prominent products of this area of research. Indeed, the language allows the programmer to express and use all kinds of data and communication since scheduling is done dynamically. Here again, such languages may benefit from the results of the CODAS project because being able to do more static analyses at compile time means more efficiency at run time. However, the static analyses we may propose have to be both precise and efficient, and combine well with the dynamic ones.

Termination of term rewrite systems (TRS) and certification Over the last 10–15 years, research in fully automated termination analysis for term rewriting has flourished, and mature tools to automatically find a (dis)proof of termination have been developed at several sites internationally.

- The Research Group Computer Science 2 (LuFG Informatik 2) at RWTH Aachen, Germany is the main development site of the termination analysis tool called AProVE [GAB+17]. AProVE was the most powerful tool for termination of standard and innermost term rewriting at the Termination Competition 2016. A specialty of the group is the application of rewriting to termination analysis of real-life programming languages: AProVE can analyze termination for programs in Java, C, Haskell, and Prolog by a translation to term rewriting (with built-in integers) and then proving termination of the resulting rewrite system.
  In CODAS we intend to use a similar idea, however we want to represent only the data dependences as a rewrite system and not the full program. We expect that this difference leads to significant benefits for efficiency in termination proving for the generated (significantly smaller) rewrite systems.
- The Computational Logic group at the University of Innsbruck, Austria develops both the termination analysis tool TTT2 [KSZM09] and the Isabelle Formalization of Rewriting (and corresponding termination techniques) IsaFoR [TS09]. The latter is used to extract the automated certification tool CeTA for termination proofs from IsaFoR. In this way, correctness of termination proofs for rewriting found by untrusted tools can automatically and efficiently be checked against a formalization in a proof assistant. The combination of this approach with the one of CODAS could be considered in a more long-term project that will have also the objective to certify the result of our method. For the time of CODAS, we consider certification as out of scope.
- Salvador Lucas' group at the Technical University of Valencia (Universidad Politécnica de Valencia), Spain develops the termination analysis tool Mu-Term [AGLN11]. At the Termination Competition 2016, Mu-Term was the most powerful tool for proving termination of context-sensitive rewriting and operational termination of conditional rewriting.

None of these teams address the problem of compiler optimization. However, some of them try to focus on applicability. We expect that we can contribute to this community by providing a new way to reason about programs.

Let us also cite the new ANR project VOCAL, [6] which proposes to develop the first certified data structure library (in OCaml). The expected output will definitely be of interest for our project since there will be a clean implementation of classic data structures and iterators operating on them.

Static analyses for High Performance Computing (HPC) Some individual papers have already explored the idea of using techniques developed for verification purposes in compilers:

- Botinčan et al. [BDJ13] and Hurlin [Hur09] have developed the idea that separation logic proofs can be used to automate the parallelization of sequential code.
- The work by Winterstein et al. [WBC14] parallelizes programs with complex data structures, with HLS as motivation.
  However, the reported run-times indicate that the technique is too costly for our purposes.

The effort to use more powerful static analyses coming from other communities is a current trend in the HPC community, and clearly the CODAS project follows this general idea. However, the idea of combining rewriting techniques with polyhedral scheduling techniques remains a novel idea yet to be explored.

[6] Accepted in 2015, `http://www.agence-nationale-recherche.fr/projet-anr/?tx_lwmsuivibilan_pi2%5BCODE%5D=ANR-15-CE25-0008`

2 Scientific and technical program, project organization

2.1 Scientific objectives of the project

Scheduling covertly regular programs, an objective for optimization Being able to schedule programs with complex data structures has a strong impact on compiler optimizations. In this project, our focus is to study covertly regular programs manipulating complex data structures. The objective of this section is to provide simple examples of optimizations a compiler could apply by reasoning about complex data structures.

Lists As a first example, let us have a look at the piece of a Java program in Figure 2.

```
public class MyList {
    private int val;
    private MyList next;
    // ... constructors and all that ...

    public static void foo(MyList mylist) {
        for (MyList curr = mylist; curr != null; curr = curr.next) {
            curr.val = curr.val + 1;          // S1
        }
        for (MyList curr = mylist; curr != null; curr = curr.next) {
            System.out.println(curr.val);     // S2
}}}
```

Figure 2: Bare-bones list iteration in Java – initial program. We could have written the same using while loops and iterators, we choose here to make the structural manipulations more directly visible.

In this program, the content of a list is modified, and then printed, with two successive loops. For each of these loops, let us use i to denote the i-th iteration of the list traversal. If we reason about dependences, then these two loops have the following dependences:

- For each of these two loops, the operation performed on element i of the loop must happen after the operation performed on element $i-1$. Recall that as we have a list, there is a structural dependence between an element and its *next*, which we shall explain in Remark 1.
- The second while loop iteration i depends on the ith iteration of the first loop.

```
public class MyList {
  ...
    public static void foo(MyList mylist) {
        for (MyList curr = mylist; curr != null; curr = curr.next) {
            curr.val = curr.val + 1;          // S1
            System.out.println(curr.val);     // S2
}}}
```

Figure 3: Bare-bones list iteration in Java – optimized program

With the dependence information, we are able to apply *loop fusion*, resulting in another version of the program, depicted in Figure 3. This transformation is commonly used in optimizing compilers. However, finding and computing such dependences for programs involving lists and other data structures is still an open problem. Pattern matching may be sufficient for simple examples, but our objective is to find a general solution to reason about this type of data.

Trees In Figure 4, we depicted another example of a recursive program operating on binary trees [7]

[7] A nicer object-oriented version could have an abstract class `Tree` with two subclasses `InnerNode` (with 2 children) and `Leaf` (with 0 children). The scheduling problem, however, would be analogous.

```
public class Tree {
    private int val;
    private Tree left;
    private Tree right;

    public int treeMax() {
        int leftMax = Integer.MIN_VALUE;       // S1
        int rightMax = Integer.MIN_VALUE;      // S2
        if (this.left != null) {
            leftMax = this.left.treeMax();     // S3
        }
        if (this.right != null) {
            rightMax = this.right.treeMax();   // S4
        }
        return Math.max(this.val, Math.max(leftMax, rightMax));   // S5
    } }
```

Figure 4: Finding the max of a binary tree in Java

This function computes recursively the maximum value of an integer binary tree. Clearly, the computation of the maximum of a tree depends on the computation of each of its children. However, the computation of the max of each child is independent from the other. We can thus transform the code in order to expose the parallelism, obtaining the program of Figure 5. With such explicit parallelism, an optimizing compiler may generate optimized code (here, generate thread-based code, for instance).

```
    public int treeMax() {
        int leftMax = Integer.MIN_VALUE;       // S1
        int rightMax = Integer.MIN_VALUE;      // S2

        if (this.left != null) {
            leftMax = this.left.treeMax();
        } ||                                   // S3 || S4
        if (this.right != null) {
            rightMax = this.right.treeMax();
        }
        return Math.max(this.val, Math.max(leftMax, rightMax));   // S5
    }
```

Figure 5: Finding the max of a binary tree in Java, modified version

Remark 1 *The objective of the proposal is to provide a way to optimize actual codes. As a result, we want to keep the data structures as they are declared and used by the programmer. To clarify this, let us have a look on this simple ML-like program operating on a list:*

```
List.map (fun x-> x+1) mylist
```

Despite the fact that this program could be transformed and optimized by using a parallel `for` *operating on an array, we want to keep the structural dependence "cell → nextcell", because of the complexity of memory layout changes. In the current program, the consequence is that there is no possible parallelization of the computation. We will not make any change to the data, but only on* iterators *that operate on them. However, as the example of Figure 3 shows, depending on the context, there might be another schedule of the program. We are conscious that this might cause some missed opportunities for optimization, thus we will be very careful about this.*

Expressing dependences and schedules with term rewriting In this section we detail the motivation for using term rewriting as inspiration for expressing dependences on data structures and manipulating them. The exploration of this relationship is the base point of the current proposal.

In imperative languages, data structures like `structs`, `records`, and even `classes` can be represented as terms (if there is no *aliasing*). For instance, the Java object `[1,2,3]` of type `MyList` defined in Figure 2 corresponds to the following term t_0:

$$t_0 = \mathsf{MyList}(1, \mathsf{MyList}(2, \mathsf{MyList}(3, \mathsf{null}))) \tag{1}$$

Similar to array entries, we would also like to address particular "entries" of a term, i.e., its *subterms*. Recall that to address entries of an n-dimensional array, we use vectors $(i_1, \ldots, i_n) \in \mathbb{N}^n$ as indices or "positions". Then for an array `A`, we say that `A[`i_1`]`$\ldots$`[`i_n`]` is the "entry" at the array's position $(i_1, \ldots, i_n)$. For terms, we can use a similar notion which are *positions*. For our list t_0, we have $Pos(t_0) = \{\varepsilon, 1, 2, 21, 22, 221, 222\}$, which give an absolute way to access each element or sublist. For instance, 21 denotes the first element of the sublist `[2,3]`, thus the integer 2. We can remark that sublists have positions of the form 2^{k} [8].

Analogous to programs on arrays in the polyhedral framework used e.g. in Figure 1, we can now consider the *positions* of a term that a statement `S` accesses. In analogy to arrays, for now it suffices to consider those positions p that are labeled with MyList (i.e., positions p where we can access an entry at position $p1$, the child of the list node that carries the `int` value at the node).

For the statements S_1 and S_2 from Figure 2, we get the following dependences:

$$\begin{aligned}
\mathrm{Dep}(S_1 \mapsto S_1) &= \mathit{None} \\
\mathrm{Dep}(S_2 \mapsto S_1) &= (2^i \to 2^i) \\
\mathrm{Dep}(S_2 \mapsto S_2) &= (2^i \to 2^j) \ : \ i > j
\end{aligned}$$

From these dependences, we want to obtain a similar schedule we could obtain with manipulating the polyhedral framework, in this case:

$$\begin{aligned}
\Theta(S_1(2^i)) &= (i, 0) \\
\Theta(S_2(2^i)) &= (i, 1)
\end{aligned}$$

and then perform code generation, whose expected output is depicted in Figure 3.

We can represent trees similarly. An object of class `Tree` is represented by $\mathsf{Tree}(\mathit{val}, \mathit{left}, \mathit{right})$ for some terms *val*, *left*, *right* that represent its attributes (so at position 2 we have *left*, and at position 3 we have *right*).

We have the following "interesting" dependences between statements:

$$\begin{aligned}
\mathrm{Dep}(S_5 \mapsto S_3) &= (p \to p2) \\
\mathrm{Dep}(S_5 \mapsto S_4) &= (p \to p3)
\end{aligned}$$

In other words, to return the result for the "iteration" (recursion, actually) at position p, we need to know the result for positions $p2$ and $p3$. These dependences can be rephrased with rewrite rules:

$$\begin{aligned}
\mathsf{dep}(\mathsf{Tree}(\mathit{val}, \mathit{left}, \mathit{right})) &\to \mathsf{dep}(\mathit{left}) \\
\mathsf{dep}(\mathsf{Tree}(\mathit{val}, \mathit{left}, \mathit{right})) &\to \mathsf{dep}(\mathit{right})
\end{aligned}$$

Instead of the positions from the earlier dependence considerations, we now look at the term shapes that we have at these positions. Here *val*, *left*, *right* are variables.

To get a schedule, we need to show that these rewrite rules *terminate*. A widely used technique for proving termination of term rewrite systems is the synthesis of *polynomial interpretations* [Lan79, BCL87, FGM+07], which map function symbols and terms to polynomials. An automatic tool like AProVE could synthesize the following interpretation *Pol*:

$$\begin{aligned}
\mathit{Pol}(\mathsf{dep}) &= x_1 \\
\mathit{Pol}(\mathsf{Tree}) &= x_2 + x_3 + 1
\end{aligned}$$

Then the termination proof consists of checking whether $\mathit{Pol}(l) > \mathit{Pol}(r)$ holds for all rules $l \to r$.[9] From that termination proof, we should be able to extract a valid schedule, and then obtain the desired code.

The preliminary paper [AFG16] describes this relationship between schedules and termination proofs, and derives an estimation on the *parallel complexity* of the given programs. This paper can be viewed as a first step toward a more complete formalization of the problem.

[8] Here a^k stands for repeated *concatenation*: $a^0 = \varepsilon$ and $a^{n+1} = aa^n$. Even though positions are sequences of numbers, we are not doing multiplications with them!

[9] Here it suffices to consider our rewrite rules as *dependency pairs* [AG00], which allows us to use a polynomial interpretation that is independent of some of its arguments, e.g., $\mathit{Pol}(\mathsf{Tree}) = x_2 + x_3 + 1$ does not depend on x_1.

2.2 Executive summary of the project and task organization

This project studies the scheduling of programs in the presence of complex data structures. We propose to use the polyhedral model as a baseline to represent data dependences and formalize the notion of schedule, and to build on existing work of the rewriting termination community to properly reason about complex data structures. The objective is to find a compact solution to represent data and computations at the same time, and then to provide algorithms to schedule programs from this representation. The research will be an experimental loop from benchmarks to theory followed by experimental evaluation.

> The **combination** of precise (yet costly) reasoning about data structures based on term rewriting and more efficient program reasoning based on the polyhedral model and its extensions will enhance the performance and precision of scheduling tools, making them useful on a wider class of systems. The organization of the project is an **experimental loop from applications to theory and back to applications**. The "Young researcher project" is adapted to this first proof-of-concept.

The project is divided into 4 parts, which are summarized below:

- The extraction of challenging benchmarks (Section 2.2.1): middle-sized, "covertly regular" programs with data structures beyond arrays.
- The formal definition of the scope of our study (Section 2.2.2): which data structures are under study. Definition of a mini-language and its semantics, then definition of what is a schedule on this language.
- The study of the relationship between control and data structures, and the extension of the scope of existing rewriting techniques and scheduling techniques. In this task we will properly define analysis and optimization for programs with complex data structures. This task is detailed in Section 2.2.3.
- The implementation of a prototype analyzer/compiler for our mini-language (Section 2.2.4) that implements some of the previous analyses. As a milestone, prove termination of rewrite systems obtained from benchmark programs with complex data structures.

All these tasks are detailed in the next sections. The risk management is done for each of these tasks. Figure 6 depicts the time organization.

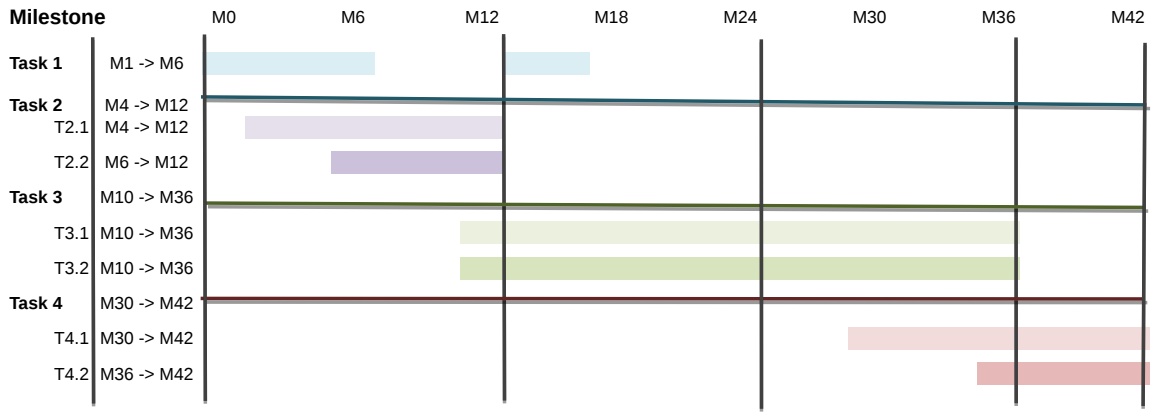


Figure 6: Organization of the different tasks of the CoDaS project

2.2.1 Task 1: Collecting benchmarks: Extraction of covertly regular kernels

The objective of this task is to extract from actual programs some "kernels" for which computing a schedule is challenging. Apart from non-regular programs/kernels (general `while` loops), we want to extract and characterize middle-sized kernels with code operating on:

- Trees (binary trees, general trees): tree traversals, with or without cuts, like in *deep learning* or symbolic computing programs.
- Linked lists, maps. Maps are definitively of interest since they are an essential step toward *sparse matrices*, which are a common object in scientific computing (numerical simulations).

In fact, as shown previously, for the above data structures, it seems quite easy to properly define a notion of *dependence*, in a quite compact way. However, combining data structures and control is challenging since the dependence graphs may quickly become infinite, thus hard to formally manipulate.

As a result, in Task 1 we propose to find more kernels of this kind, from real-world applications, or at least, coming from benchmarks of the literature, namely:

- Open source generalist programs: such programs are widely available and known, and thus amenable to meaningful data structure extraction. Among them, the SPEC benchmarks [10] for intensive computations and the Kraken benchmark from Mozilla for more general applications [11] will be of interest.
- More specialized benchmarks from the polyhedral scheduling literature, for instance Polybench. [12] We will use these benchmarks essentially to measure the performance of our algorithms in the simple case where the classical polyhedral model for scheduling is immediately applicable.
- Signal processing algorithms, like radio signal modulation or channel allocation algorithms within latest-generation radio protocols (e.g., LTE[13]) exhibit irregular programs on large data structures [DMRM14].
- Worklist algorithms, such as abstract interpreters or graph algorithms, also have some hidden regularities that can be exploited [PNK+11].

Such programs are widely available and known, and thus amenable to meaningful comparisons between existing approaches. Furthermore, we strongly believe that the state-of-the-art benchmarks of the polyhedral compilation community will not be sufficient as they only fit the polyhedral model. The very first step of the project will thus be an experimental phase to construct a benchmark of interesting programs to schedule. We might also modify benchmarks from the polyhedral literature and add some computation on lists and/or trees.

What we want to extract in this activity are codes for which the cost of the overall computation exceeds the cost of changing iterators. Typically, improving computations on sparse matrices would mean changing the ordering of the computation steps between different dense blocks, on which classical fine-grain parallelization/optimization could be performed.

Although the main part of the task will be manual (and benefit from our joint experience with HPC benchmarks), we may in some cases use algorithm recognition approaches [AB03, IAR14, ATD04] to automatically detect regular kernels. Of course, they will not suffice to discover non-polyhedral kernels. Other techniques (e.g., *profiling*) could also be useful to discover computation-intensive kernels in general programs. We will typically take inspiration from frameworks to detect potential parallelism at runtime, e.g., APOLLO [CSRB+17].

Means Task 1 is scheduled from month 1 to month 6 of the proposal. We plan to hire a Master student (first year or second year) to help us with this activity.

All the members of the project have experience with experimentally directed research. The four implied people here will be Laure Gonnord, Lionel Morel, Fabrice Rastello, and Tomofumi Yuki.

Fabrice Rastello is an experienced collaborator on this topic. He has a strong background in designing compilers and currently leads a project on dynamic compilation, which strongly relies on proper extraction of information at compile time or at execution time. Lionel Morel has experience in analysis of dataflow programs and design of dataflow runtimes. In particular, he will help widen the application scope of the project towards signal processing applications. Tomofumi Yuki has experience in working with polyhedral benchmarks. He is a contributor to the PolyBench, the most commonly used benchmark suite for polyhedral approaches. He is also in the process of collecting larger scale applications with *almost* polyhedral regions, which may include interesting code regions for this project as well. Laure Gonnord has developed many static analyses inside the LLVM framework. Her experience in *program manipulation and slicing* might be of great interest for this task.

Risks The risks for this task are low: all implied people have experience with use and design of suitable benchmark sets. There are huge polyhedral benchmark sets, and (particularly also open-source) software for scientific computations in all kinds of application domains (weather forecast, data-centric applications, ...).

[10] `https://www.spec.org/benchmarks.html`

[11] `https://wiki.mozilla.org/Kraken`

[12] `https://sourceforge.net/projects/polybench/`

[13] Long-Term Evolution standard for next generation wireless protocols, `http://www.3gpp.org/technologies/keywords-acronyms/98-lte`

Deliverable We plan to extract from these benchmarks a new series of non-polyhedral kernels of interest and to publish these benchmarks in a research report as well as on the project's website.

2.2.2 Task 2: Definition of the scope of our study, and formalization of the problem

The objective of this task is to decouple our theoretical study from the implementation part. We will:

- properly define a language of interest and validate its expressiveness;
- then, we will define the scheduling problem based on our abstract representation.

Task 2.1: Definition Based on the previous benchmarking activity, this task will define the scope of our study. From the benchmarks, we will define which data structures are of interest, and what operations we want to perform on them. Then, we will define a mini-language to avoid implementation details or language specificities. This language will incorporate high-level operators on complex data structures inspired by functional languages (e.g., inductively defined data structures, "à la ML"). The advantages of such a mini-language are:

- The notions of higher level constructs are now implemented in Scala and Java. Map, reduce are already well-used coding patterns for end-users.
- High level operators such as `map`, `iter`, already expose some dependence information (`map` iterators are parallel loops, ...), which might be very helpful to properly define the notion of dependence of computations in our language.
- Rewriting techniques are strongly linked to functional languages. Restricting to ML-style data structures (i.e., trees *without sharing/aliasing*) simplifies the task for the analysis, of course: this kind of complex data can directly be represented as labeled trees, i.e., terms. Sharing/aliasing analysis is an orthogonal problem subject to active ongoing research [BCC+07] since the seminal work [JM82].

This activity will consist of defining a clean "abstract" semantics of our language, in order to avoid specific implementation matters, but also permit the expression of the intrinsic parallelism of both control and data structures. We might find some inspiration of course in ML-based semantics of the literature and also in the previous work of project member Lionel Morel on array iterators and their compilation in the synchronous language Lustre [Mor07] (in fact, the notion of data dependence is also crucial in synchronous languages). Finally, we believe that the previous work of Barthou et al. [BCC00] which proposes (for scalars and arrays, but with non-affine accesses) the concept of *maximal static expansion* to capture the data structures' behavior, might be a good starting point to study the current limits of power of dependence analysis.

We will also have to demonstrate that our language and our (abstract) semantics do not narrow our study too much (for instance, is avoiding dynamic creation or sharing an overly strong restriction?), and that the programs under study could easily be developed in other known languages (such as C). One main difficulty of this task is to define a semantics that will express the characteristics of the data structures under study:

- With enough precision to be able to derive dependences.
- Simple enough to be able to reason easily about it.

Task 2.2: Formalization of the problem Once we have a suitable semantics, we have to express what the notion of a scheduling means for this semantics. Formally, we aim to express the notion of dependence, like the "happens-before" relation of the polyhedral model. This notion of scheduling will enable us to construct a dependence graph of a given program under analysis. As we show in Section 2.1, we will explore how the notion of position could help us to define a compact representation of data dependences and control-flow dependences; then we might be able to define a proper definition of what scheduling means, by extending the notion we already have on the polyhedral model or its extension to `while` loops ([ADFG10]). We might also get some inspiration from the parallelization of functional programs proposed in [RGP16].

Remark 2 About commutativity and associativity *In this proposal, we focus on program transformations preserving the flow dependences, hence the computation (defined as Herbrand equivalence in [AB03]). Even though program transformations modulo equational properties (associativity, commutativity, etc.) open a number of opportunities for code optimization, many challenges still need to be addressed for regular programs before irregular programs can be considered. For example, the PhD work of Guillaume Iooss [IAR14, IRAZ14] points out many open problems for semantic tiling, a transformation modulo associativity/commutativity.*

Means We plan to overlap this activity with Task 1, during M4 to M12.

This activity will be collaborative. In this task, the hired PhD student will be helped by Christophe Alias, Laure Gonnord, Carsten Fuhs, Lionel Morel and Tomofumi Yuki, who have strong experience in defining operational semantics and analysis.

Christophe Alias has experience in dealing with associativity and commutativity, and also in defining polyhedral-based solutions. Laure Gonnord has experience in sequential C-based and synchronous languages semantics. Carsten Fuhs has experience in semantics for Prolog tailored to program analysis [SESK+12]. Lionel Morel has experience in the design of high-level array operators and their compilation to parallel loops. Tomofumi Yuki has experience with parallel programming languages and their semantics, and he has recently worked on semantics for the X10 language being developed by IBM Research.

Risk One of the potential risks is to define a limited scope for the study and a formalization that is purely theoretical and too narrow to be effective on real code. To avoid this pitfall, Task 1 is essential and will be repeated at the end of Task 2 for a less coarse benchmark which will be adapted to the defined scope of analysis.

In case we do not manage to express relationships in a unified way, we could also:

- compute approximations of the transitive closure of dataflow dependences (with the help of abstract interpretation);
- devise *several* specialized frameworks, each tailored to specific classes of scenarios.

In any case, we will make sure to inform our formalization(s) already in Task 2 by suitable examples from practice.

Deliverable The definition of the semantics will be published in a research report. The unified notion of dependence for abstract data structures might be published at a workshop on compilation, like IMPACT (Workshop of Polyhedral Compilation Techniques), or conference, such as CC (Compiler Construction).

2.2.3 Task 3: Combining rewriting techniques and classical scheduling techniques

In this task, we will study the relationship between control and data structures, based on our experiments and also our background in static analyses. The main objective here is to extend the scope of scheduling techniques and rewriting techniques:

- In the domain of polyhedral scheduling techniques, we want to develop a compact representation of program (and data structure) dependences which still permits efficient analyses and the computation of legal schedules.
- In the domain of rewriting, we want to extend the ideas from polyhedral analysis to consider also complex inductively defined data structures. In this setting, data is represented as terms, and the dependences will be represented as a term rewrite system. To cater for programs that operate both on integers/arrays and on complex data structures, we will also consider suitable extensions of standard term rewriting with built-in data structures (like in, e.g., [FGP+09, KN13]) to obtain a unified representation.

For these tasks, we plan to focus initially on one or two data structures, e.g. linked lists and binary trees. The goal is then to lift the gained experience to more general structures (general trees, possibly involving arrays).

Remark 3 Computing transitive closures *In the polyhedral model, dataflow dependences can always be modeled by affine relations source → target. The point is to find* direct *dataflow dependences: the smallest affine relation whose transitive closure is the flow-dependence relation. This is usually referred to as dataflow analysis. Feautrier provides a general solution for regular programs [Fea91] with ILP solving. Later, Barthou provides a generalization to arbitrary programs by introducing fuzzy parameters [BCF97], the point being to eliminate as many fuzzy parameters as possible. From dataflow analysis, many analyses can be applied: array liveness [ABD07], equivalence checking [BFR02, AB03], code generation [BF98], etc. Some of these analyses may require a transitive closure to "walk through" direct dataflow dependences, as for equivalence checking. But we believe that dependence analysis is far less difficult than dataflow analysis and using a transitive closure would be an overkill. Actually, the trade-off is better expressed by the fuzzy parameters of [BCF97]. How to eliminate them? We believe that abstract interpretation (which requires, in a way, a transitive closure) could be helpful.*

Task 3.1: Develop compact representations The objective of this part is to provide a way to reason about data structures and control at the same time, while still being compact and efficient.

We may find a source of inspiration in the current state-of-the-art techniques for proving termination of term rewrite systems. In particular, term rewriting is very suitable for tree-shaped data (i.e., no sharing/"deep aliasing") [OBvEG10]. It is also useful for analyzing termination of DAG-shaped data (also known as trees-with-sharing) [BOG11]. In contrast, it is not as useful when cycles are allowed.

The AProVE termination tool uses a dedicated shape analysis for Java to distinguish these shapes and construct a dedicated term rewriting instance from them. However, despite the fact that the approach used to handle programs with loops by instrumentation with integer variables for the *length* of the cycles has shown its applicability for verification purposes [BMOG12], we advocate for a different approach because our final objective is to enable program transformations for optimization. The main drawback of rewriting techniques based on integer encoding is that it breaks the link between the initial program structure and the system used to prove termination. Therefore it is difficult to directly use the result of the termination proofs for our needs.

Moreover, we also believe that we must avoid as far as possible "brutal" encodings of the entire program as rewrite systems since they can be too costly for compilation purposes (the results of the Termination Competition and our recent comparison with Termite [14] show prohibitive timings).

Thus our idea, as we show in Section 2.1, is not to encode the whole program as a TRS, but to focus more on the idea of **dependence**, which is the crucial notion for scheduling. What we expect from this approach is that expressing only data dependences as TRS, or, at least, adapting the existing effective algorithms from TRSs to our scheduling problem, will avoid the non-scalable translation from whole programs to TRS.

Remark 4 (General trees) *As soon as we can handle binary trees, the step to* arbitrary *tree-shaped data objects should be quite straightforward. One small challenge could then still be to combine recursive types and* arrays *(which are after all part of the polyhedral model), as in the Java class to the right.*

```
public class Tree {
   private int value;
   private Tree[] children;
}
```

Task 3.2: Adapt polyhedral scheduling techniques to programs with lists or trees This task is the core of our project. From the compact representations of the data and control dependences obtained in Task 3.1, we want to design algorithms to infer schedules.

The link between scheduling and rewriting in the polyhedral model has been observed through the notion of system of recurrence equations (SRE, [Qui84, KMW67, Raj89]). A polyhedral program can be represented as a set of equations of the form: $A_0[\vec{i}] = f(A_1[u_1(\vec{i})],\ldots,A_n[u_n(\vec{i})]), \vec{i} \in D$, where each A_k is a single assignment array, the u_k express the flow dependences depicted in Figure 1 (red arrows), and D is a convex (integer) polyhedron. The rewrite system $A_0[\vec{i}] \to f(A_1[u_1(\vec{i})],\ldots,A_n[u_n(\vec{i})])$ is convergent and normalizes to the (array of) value(s) computed by the program. Also, it captures exactly the flow dependence, and from that model we are able to schedule computations. Based on an analogous representation (like in Section 2.1) from the output of Task 3.1, we will then derive scheduling techniques that still apply to recursive programs iterating through complex data structures, like the one depicted in Figure 5. For that purpose, we might find inspiration in the previous work of Cohen and Collard on parallelizing recursive programs and programs with trees [CC98] (however, we do not want to use context-free grammars, which might be too expensive).

We may also find some inspiration in the area of separation logic: indeed this community has already conducted some research on using static analysis for parallelization in the presence of complex data structures [BDJ13, Hur09, WBC14]. However, we do not plan to look at the difficult problem of extracting complex information automatically from C-like programs, which would imply complex *aliasing* pre-analysis. This kind of pre-analysis is an orthogonal problem which we will not consider in this project.

Means We plan to overlap this activity with Task 2 and Task 4, from M10 to M36.

This topic is the heart of our project. This will be the main topic for the PhD student we want to hire. The activity will be collaborative. In this task, the hired PhD student will be helped by Laure Gonnord and Tomofumi Yuki, Carsten Fuhs for Task 3.1, and Christophe Alias and Lionel Morel for Task 3.2.

Christophe Alias has a strong background in designing scheduling techniques and optimization for High-Level-Synthesis, and his experience in actual implementation in compilers will be a great help in order to design applicable algorithms. Carsten Fuhs is an expert in term rewriting and its applications in program

[14] `http://termite-analyser.github.io/`

analysis. His background in termination analysis for term rewriting will also be very useful for making the connection between termination proof techniques for rewriting and corresponding program schedules. Laure Gonnord has a strong background in abstract interpretation to compute over-approximated transitive closures. Lionel Morel brings his expertise on identification of data dependencies for high-level array operators.

Risk The risk of this task is low to medium: we already have a fairly clear idea of how to represent dependences for iteration and recursion over lists or tree-shaped data. How to compute a scheduling from them or introduce some approximations due to statically unknown data still remains a challenge; however we could at least find classical *patterns* and provide ad-hoc solutions from them.

Deliverable The combination of polyhedral techniques with rewriting techniques to deal with programs with lists and trees will be published first as a research report, and it will most probably be of interest for conferences like CC (Compiler Construction) or FSCD (Formal Structures in Computation and Deduction), the successor conference of RTA (Rewriting Techniques and Applications).

2.2.4 Task 4: Prototyping

The goal of this task is to implement the first version of analysis and code generation techniques and evaluate on some middle-sized examples. Scheduling is only the first step towards optimization of complex programs.

Task 4.1 Validate on termination As a first milestone, we want to validate that the generated term rewrite systems for the dependences can indeed easily be analyzed and proved terminating by existing termination provers like AProVE, Mu-Term, or TTT2. From these termination proofs, we will synthesize a corresponding schedule. Even in the worst cases, where TRSs are not proven to be terminating, we might find interesting dependences during the proof process. Since we already have considerable experience in empirical evaluations of termination analysis tools, this task is very suitable as a first step.

Task 4.2 Code generation (prototype) Based on the schedules found in Task 4.1, we want to generate corresponding optimized code for our programming language. When scheduling is placed in the context of program optimization, it is also necessary to develop techniques for reflecting the new schedule. One approach is to perform code generation based on the program representation and the schedule such that the execution order of the generated code matches that defined by the schedule. In the domain of optimizing compiler research, it is almost mandatory to provide empirical evaluations for proposed optimizations, which can be facilitated by developing code generators. Although our main objective in this project is scheduling, we would also like to start preliminary exploration of this tightly coupled problem.

We will first explore possible implementations of the schedules by manually editing/writing pieces of programs. The development of a first prototype is planned. The prototype may be limited to some subsets of the programs/schedules that we develop, and the full treatment of the code generator will be left as future work. The benchmarks under study will also be provided as test cases for future implementations.

The challenging part of this task is how to implement a kind of parallelism that is no longer only "loops+do-all" (i.e., the "map" pattern). OpenMP then appears as an appropriate back-end since it provides "do-all" fine-grain parallelism as well as task parallelism. The balance between these two kinds of parallelism will also be an interesting research problem in this task.

Remark 5 About hardware characteristics *The code generator must be able to target different hardware platforms with different characteristics (e.g., caches, number of cores, and so on) to obtain highly efficient code. For low-level optimizations, such as vectorization, we plan to benefit from back-end compilers, e.g., by exposing vectorization-friendly loops. As a higher level design knob, we plan to provide grouping of tasks as a tunable parameter, similar to tile sizes in loop tiling. This allows us to fine-tune the amount of parallelism in the generated code to specific platforms.*

Means We plan to overlap this activity with Task 3, from M30 to M42.

The involved people will be the PhD student together with the other project participants. The activity will be led by Carsten Fuhs (Task 4.1) and Tomofumi Yuki (Task 4.2). Carsten has significant experience in experimental evaluations in the field of termination proving, as also highlighted by his active role in the preparations

of the tool AProVE for the annual Termination Competition. Tomofumi has contributed to the development of the AlphaZ system that uses many techniques from the polyhedral model, including code generators.

Risk The risk of this task is low to medium: in this task as well as the other ones we only target intra-procedural optimization (the case is already difficult enough), and thus the obtained TRSs should be small enough so that performance of the termination analysis tools should not be problematic. For code generation, the risk is higher, but we should at least come to a solution that allows complete code generation for the most simple programs with little modifications to state-of-the-art polyhedral code generators such as CLooG [15].

Deliverable The whole technique with experimental evaluation as a proof of concept might be of interest of larger-scope top-tier conferences like POPL or PLDI. During this period, we will also write the end-project report.

2.3 Partners description and relevance, complementarity

Short Bio of the investigator `http://laure.gonnord.org/pro/`. Laure [16] received her PhD degree in computer science from the University Joseph Fourier (Grenoble), in 2007. She has been an assistant professor at the University of Lille, and currently holds an assistant professor position at University Lyon 1/UCBL. Her main research interests lie in the design of static analyses, with emphasis on the automatic synthesis of numerical invariants and application in compilation (scheduling) and termination proofs. She belongs to the ROMA joint team of the LIP laboratory (ENS Lyon). She has already published in major conferences on static analysis or compilation. She will also benefit from the fruitful discussions with the Plume members of the laboratory, some of them being specialists in programming languages semantics and rewrite systems.

She will be the main advisor [17] of the hired Phd student, who will be co-advised by Lionel Morel.

The publications of the investigator cover many topics from embedded systems design to program verification and code optimization. The major publications of interest for this proposal are detailed in the appendix.

The principal investigator, Laure Gonnord is member of ROMA, a common research project-team of LIP (Laboratoire de l'Informatique du Parallélisme) and Inria. The team is located at Ecole normale supérieure de Lyon (ENS Lyon) in Lyon, France, which will be the unique **partner** of this proposal. Laure Gonnord has experience in Program Analysis for verification and optimization purposes. Her major publications in large-impact conferences (PLDI, OOPSLA) show a growing interest for applicability and scalability of the analyses she proposes.
Moreover, Laure Gonnord has experience in collaborating with **researchers from different communities** (synchronous languages, compilation, static analyses, software engineering). Therefore, she is fully able to coordinate this project and to bridge the research interests of other people involved in the project.

Collaboration This proposition is a "Young researcher" proposition, so only the first partner (ENS Lyon) will benefit from the funding. However, other people will be involved in the project as collaborators coming with their own funding.

- **Christophe Alias**. `http://perso.ens-lyon.fr/christophe.alias`. Christophe received his PhD degree in Computer Science from University of Versailles in 2005. His PhD topic was related to the equivalence of programs expressed as systems of affine recurrence equations (SARE). Then, he worked as a postdoctoral researcher at Ohio State University (USA) on data layout optimization for SIMD optimization. He is now a permanent researcher (CR1) at INRIA. His research interests include polyhedral compilation techniques and high-level-synthesis. Recently, he co-founded the XtremLogic startup company, which aims at spreading his research results to industry. Christophe Alias's expertise both in formal methods and in high-level hardware synthesis of applications will be very useful for us. Christophe's research projects also include the study of complex data structures to optimize their silicon compilation.

[15] `http://www.cloog.org/`
[16] A more detailed CV can be found in the appendix.
[17] Her "Habilitation à Diriger les Recherches" is scheduled for Autumn 2017.

- **Carsten Fuhs**. http://www.dcs.bbk.ac.uk/~carsten/. Carsten defended his PhD on automation of termination proofs for term rewrite systems in 2011 at RWTH Aachen University and has been a lecturer at Birkbeck, University of London since 2015. He is one of the main developers of the automatic termination analysis tool AProVE[18], which is among the leading tools in the annual International Termination Competition[19]. AProVE pioneered the two-stage approach for proving program termination with front-ends to convert from programming languages to term rewrite systems with built-in integers and back-ends to prove termination of these integer rewrite systems [FGP+09]. Carsten Fuhs has been invited to teach a course on *Proving Program Termination via Term Rewriting* in the Advanced Track of the International School on Rewriting 2017. [20] The collaboration with Carsten will benefit from his background in the state-of-the-art of techniques for proving termination of rewrite systems and of programs. He has also worked on static program analysis of worst-case computational complexity [BEF+16]. Carsten's research is motivated by the potential application of the program analyses he develops, he always focuses on techniques that can be incorporated into push-button tools to increase the productivity of software developers. Carsten's own research projects thus have many similarities with CoDaS.
  It is worth noting that although Carsten Fuhs has a position in the UK, there is no real impact of the Brexit on our proposal. This JCJC proposal implies that all the funding will be hosted at ENS de Lyon.
- **Lionel Morel** http://lionel.morel.ouvaton.org. Lionel received his PhD degree in Computer Science from the Grenoble Institute of Technology in 2005. His PhD was related to the extension of the Lustre dataflow language with array iterators, permitting to use regular programming structures together. He proposed compilation and formal verification techniques taking advantage of these language constructs. After his PhD, he worked for two years on designing models of computations for heterogeneous embedded systems, first at Åbo Akademi, in Turku, Finland, then at Inria. Since 2007, he is an Assistant Professor at INSA Lyon. His current research interests include runtimes and performance evaluation techniques for dynamic dataflow languages, as well as design of video decoders and radio protocols with these languages. The collaboration with Lionel will bring us expertise on optimization techniques for dataflow programs as well as his experience with language design and signal processing applications.
- **Fabrice Rastello**. http://perso.ens-lyon.fr/fabrice.rastello/. Fabrice received his PhD degree in Computer Science from ENS Lyon in 2000. He then worked as an engineer for STMicroelectronics for two years before becoming a full researcher at Inria. He is a specialist on dynamic profiling to infer hot-points in code under optimization. He is the head of the Inria CORSE joint team, leading the research in hybrid analyses (both static and dynamic), and hybrid compilation (see Section 1.5). The collaboration with Fabrice Rastello will help us get an expertise in profiling applications to extract intensive computations to optimize.
- **Tomofumi Yuki**. http://perso.ens-lyon.fr/tomofumi.yuki/. Tomofumi received his PhD in Computer Science from Colorado State University in 2013. He has been a member of the CAIRN team at Irisa Rennes as a post-doctorant until 2014. He is now an Inria researcher (CR2) at Inria Rennes. His past research includes static analysis of parallel programs, polyhedral analyses and transformations, and application of polyhedral techniques to High-Level Synthesis. The collaboration with Tomofumi Yuki will bring us expertise in the polyhedral model, and especially the design of polyhedral-based compilers. His current research project includes the use of static analysis techniques, traditionally used for automatic parallelization, to help parallel programming. Modern parallel programming languages involve objects and complex data structures, which are not well treated by traditional techniques. Extending the application of static analysis to such data structures is tightly linked to his objective to provide helpful feedback to parallel programmers.

The other participants of the project have complementary skills and **expertise in different scientific domains**. Moreover, their own research projects have strong relationships with the CoDaS project. All the necessary expertise will thus be available for this project, and also for a future long-term collaboration. Moreover, this topic is **novel in the lab** (LIP/ENS Lyon), despite its long expertise in the area of parallelism and scheduling. The participants may also benefit from the expertise in topics such as semantics and logics of the Plume team (LIP lab).

[18] http://aprove.informatik.rwth-aachen.de/
[19] http://termination-portal.org/wiki/Termination_Competition
[20] http://www.win.tue.nl/~hzantema/isr.html

Figure 7 and Table 1 (page 2) summarize the involvement of the different researchers participating in the project. As this is a junior proposal, partners from outside ENS Lyon/LIP are external collaborators, with their own fundings. The person months are computed with respect to the total duration of the project (42 months).

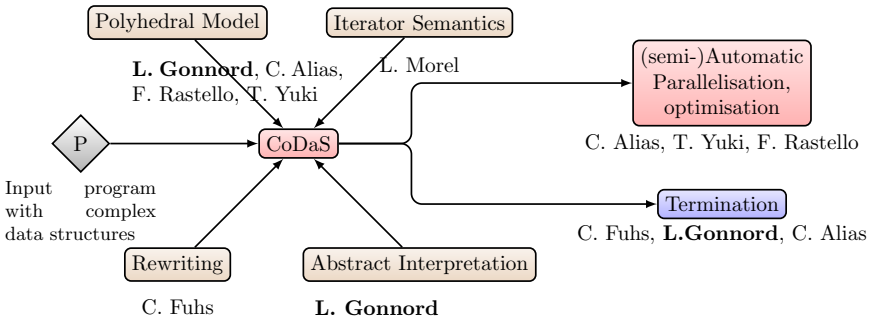


Figure 7: Contributions of the involved persons in the topics of the project

2.4 Scientific justification of requested resources

Financial support is required for the completion of the CoDaS project. Support is divided into manpower, travel, and some small equipment.

Staff

- One Master student (supervised by Lionel Morel and Laure Gonnord in Lyon, in collaboration with Tomofumi Yuki in Rennes). The Master student will work 6 months on both Task 1 and Task 2. Budget: $6 \times 600 = 3600$ €. This budget is included in "other external expenses".
- One PhD student (supervised by Laure Gonnord (HDR in 2017) and Lionel Morel). The thesis work will start at M6 of the first year of the project. The PhD student will start by reading the state of the art, and based on the preliminary results of Tasks 1 and 2, will mostly focus on Tasks 3 and 4. The cost of the PhD student is roughly 100 000 € for three years.

Mission/Travel Frequent meetings (one by trimester) will be organized to evaluate the overall progress of the project: are objectives and milestones far from completion? Are there some unexpected deadlocks? For that purpose, some travel expenses from London (for the work in collaboration with Carsten Fuhs), Grenoble (for the work in collaboration with Fabrice Rastello), Rennes (Tomofumi Yuki) to Lyon are planned.

Participation at conferences in all domains related to the project will be planned to disseminate the obtained results. Three participations per year are considered. As an example of budget, a typical mission to North America for a conference like PLDI is 2000 €.

This budget also contains the cost of missions for the project's principal investigator: kick-off seminar, project review, ANR's colloquiums.

The budget for this item is 25 000 €.

Other expenses The budget also includes some small equipment, mainly devoted to the students: one desktop for the intern and one laptop for the PhD student, and some minor accessories (batteries), in total 3400 €.

Laure Gonnord also applies for teaching compensation for 96 hours per year during the last three years of the project (from M6 to M42). This compensation is also included in the budget ("external facturation", 10 000 € per year, total 30 000 €). The university (University Lyon 1/UCB) will charge ENSL for this budget line, after acceptance by the board of directors. We plan to ask for their official authorization as soon as the CoDaS project is accepted. For 2017–18, Laure Gonnord also requested a "demi-delegation" at Inria and CNRS.

The total cost is thus 162 000 €. The ENS Lyon's charges are 4% of the total amount, and 4% will be devoted to the ROMA team/LIP Lab and thus an additional cost of 6480 € × 2.

The main expenses of the project will be the salary of the PhD student and some additional budget devoted to missions and conferences. The total budget is 174 960 € (including charges).

3 Valorization, Impact of the proposal

3.1 Dissemination

The results of the CoDaS project will be disseminated through publications in scientific conferences or journal papers. The work covered by the proposal is under the scope of many A or A+-ranked conferences or journals, such as ACM Transactions of Programming Languages, ACM Transactions on Architecture and Code Optimization, Science of Computer Programming, or for conferences Principles of Programming Languages, Programming Language Design and Implementation, Static Analysis Symposium, Compiler Construction, and also back to our source of inspiration: Formal Structures in Computation and Deduction as well as the Conference on Automated Deduction. The publications will also be disseminated though open-access devices such as HAL and arXiv.

The software we develop will mainly be prototypes (most probably inside compiler infrastructures such as LLVM or GCC), however, the source code will be publicly available. The participating scientists express a strong preference for licenses such as GNU GPL, or licenses that restrict commercial uses but allow uses for educational and research purposes. If the research is mature enough, we will propose an implementation in the SigmaC compiler within the context of our collaboration with Kalray [21].

A dedicated website will be created at the beginning of the project, and will trace the status and results of the project.

Four participants to the project, namely Christophe Alias, Laure Gonnord, Lionel Morel and Tomofumi Yuki, are involved in Master courses whose topics intersect with the scope of the project (Compilation, Static Analysis, Code Optimization). We plan to integrate some of the results of the project onto our future courses.

3.2 Networking

The results of the project (work in progress, tool demos, research issues) will be communicated to the community (as we are used to doing), in France and abroad, for instance in the French community of compilation[22], and the German and South England programming languages communities[23].

Most participants of the project also belong to the Hipeac European Network on High Performance and Embedded Architecture and Compilation and will use this network to disseminate more mature work in Hipeac events such as the plenary conference, satellite workshops, or tool demos.

3.3 Expected Impact

The long-term expected impact of these work is significantly widened applicability of various tools/compilers related to parallelization. Analysis and representation of schedules are fundamental parts of compiler analysis, especially important when highly parallel architectures are targeted. The extension of static analysis capabilities to complex data structures allows programmers to benefit from sophisticated tools when writing programs dealing with such data structures. The current uses of polyhedral techniques are not limited to automatic parallelization, but are also used for detection of parallel bugs, proving program termination, detecting transient errors, and so on. The proposed work is expected to constitute a major milestone in the area of parallel computing, and especially automatic scheduling, from the theoretical foundations to practical issues.

From a research standpoint, we expect to find a new unified framework to reason about program control and data structures at the same time. We will demonstrate that ideas coming from the two communities of rewriting and compilation are complementary and can be combined for purposes of efficiency and applicability. We expect a mutual enrichment of the foundations of both areas of research.

From a technological standpoint, the results of the CoDaS project will demonstrate that code on complex data structures can also be mechanically optimized. This will send a strong message to both HPC and general-purpose software developers: yes, the technology of automatic parallelization/scheduling is mature enough and practicable.

[21] `http://www.kalrayinc.com/`

[22] `http://compilfr.ens-lyon.fr`

[23] `http://www-ps.informatik.uni-kiel.de/fg214/` and `http://dominic-mulligan.co.uk/?page_id=148`

ANR proposal CoDaS : Complex Data-structure Scheduling - **Appendix**

Coordinator: Laure Gonnord

Appel à projet générique 2017, instrument “jeune chercheur”, défi 7.

Curriculum vitae

CV Laure Gonnord

Postal address:
Laboratoire d’Informatique du Parallélisme
UMR CNRS - ENS Lyon - UCB Lyon 1 - INRIA 5668
46 allée d’Italie
69364 Lyon Cedex 07, France

Université Lyon 1 Claude Bernard

Tel.: 04 72 72 85 69 (LIP)
E-mail:laure.gonnord@ens-lyon.fr
Web: http://laure.gonnord.org/pro/

Tenured Assistant Professor (Maître de Conférences)
Topics: Software Verification, Embedded Systems, High Performance Computing, Static Analysis, Compilation.

Current and former positions

since Sept 2013	**Assistant Professor** *University Lyon 1*. Science Faculty, Computer Science Department. Lab: LIP, Compsys then ROMA team.
2009 – 2013	**Assistant Professor** *University Lille 1*. Polytech Engineer School. Computer Science and Embedded System Departments
2008 – 2009	**Teaching and research assistant** *University Lyon 1* and LIP.
2007 – 2008	**Postdoctoral position**, *INSA Lyon*, Lab CITI, ANR Project REVE.
2003 – 2007	**PhD in Computer Science** Grenoble University, Synchronous Team, Verimag Lab advisor N. Halbwachs and **teaching position**, *Grenoble University*. PhD: “Abstract acceleration to improve precision of Linear Relation Analysis”, defended on October 27th, 2007.

Software

- **Vaphor**: A prototype of a static analysis tool that abstract programs with arrays into array-free Horn clauses (2k OCaml LoCs) (Participation 40%, collaboration with D. Monniaux). The tool has been substantially re-engineered by Julien Braine since June 2016. http://laure.gonnord.org/pro/demopage/vaphor/
- **Termite** (10% participation): termination proof tool, in collaboration with G. Radanne, main developer, and D. Monniaux. https://termite-analyser.github.io/.
- **Aspic**: A static analysis tool that implements accelerated Linear relation Analysis, (20 000 OCaml LoCs) http://laure.gonnord.org/pro/aspic/aspic.html·
- **ReveViewer**: A prototype “proof of concept” for the REVE ANR Project, a remote image viewer and the software architecture around it to deal with resource constraints (5000 C++ LoCs).

Service

Teaching/Supervising

- from 2002, I taught for a total 1600 hours from undergraduate to graduate students, from basic programing courses to advanced courses like Program Analysis and Compilation.
- from 2009, I advised 5 undergraduate internships and 2 Master students.
- from 2012, I was in the PhD Jury of 3 theses, including one where I was one of the two referees.

- from Sept 2014, I have been co-advising (with F. Vivien) the PhD of Maroua Maalej, on designing low cost static analyses.

Conferences

- PC member of VMCAI '17, WST '14, TAPAS '12.
- PC Jury of the Student Competition at CGO '16.
- Reviewer of the conferences CAV, VMCAI, STACS, LCTES...
- Invited seminars: Google and SRI (June 2015), student seminar at ENSL (2014), ...(`http://laure.gonnord.org/pro/research/seminars.html`).

Projects

- Participation in the projects System@tic "APRON" and ANR "REVE".
- Coordination of the Lille university BQR Project "ALIL" (languages and analysis for software engineering) in 2011/2012.
- Coordination of the CNRS INS2I Project "HLS-RT" for 2012 and 2013. (16k€ for 2 years) High level synthesis under real-time constraints.
- Partner of the "PROSPIEL" Inria associate team (Brasil-France) since 2015 (20k€ per year, 2 years).
- Principal investigator of the BQR ENS de Lyon Project "SODAS", October 2015 (12k€ for 2 years) High performance programming of complex data structures.

Other

- Co-coordinator of the French Compilation group `http://compilfr.ens-lyon.fr/` since 2010.
- Member of the following work-groups: GDR GPL Board, Labex MILYON teaching commission, finance committee of the Science Faculty, finance committee of the LIP.
- Elected member of the LIP Lab council.
- Since 2013, I participated in 3 associate professor and one junior researcher (CR) selection committees. I also belong to the Grenoble-Alpes' Inria doctorate studies committee since 2014.

Publications

The full list of my publications (4 journal papers, 12 international conferences and 6 international workshops at the date of March 1st, 2017) can be found on HAL or on the webpage `http://laure.gonnord.org/pro/papers.html`

Among these publications, the publications that are the most relevant to the CoDaS project are the following: the first three being directly linked to scheduling (termination), and the fourth a sequence of static analyses that might be useful to analyze our programs (slicing, memory analyses, ...)

- Our preliminary informal paper [AFG16] presented at the Workshop on Termination in 2016: Three of the investigators involved in CoDaS (Alias, Fuhs, Gonnord) propose the use of (a generalization of) polynomial interpretations to estimate the worst-case runtime of arbitrary *parallelizations* of a given sequential program on complex data structures. This paper opens up new perspectives that we wish to explore in the CoDaS project.

- Rank [ADFG10] (107 citations according to Google Scholar): This paper is the first combination of techniques coming from the polyhedral model community and static analysis community. Combining the information gathered by abstract interpretation on a program with the scheduling techniques that were formally applied on static nests of `for` loops has permitted to design a new algorithm to compute affine schedules (and thus ranking functions) for arbitrary flowcharts programs. This paper had a quite strong impact on the termination community.

- Termite [GMR15]: This paper is an enhancement and extension of the algorithm of [ADFG10] and other similar algorithms in the sequential termination community. The contribution of this paper is a more general method synthesizing lexicographic linear ranking functions (and thus proving termination), supported by inductive invariants, in the case where the transition relation of the program includes disjunctions and existentials (large block encoding of control flow). Our algorithm incrementally refines

a global linear constraint system according to extremal counterexamples: only constraints that exclude spurious solutions are included.

- Greenarrays [NMS+14]: In this paper we use static analysis to prevent out-of-bounds memory accesses in C programs. There exist several techniques to secure C programs; however, these methods tend to slow down these programs substantially, because they populate the binary code with runtime checks. To deal with this problem, we have designed and tested two static analyses – symbolic region and range analysis – which we combine to remove the majority of these guards.

These three last papers have also contributions in terms of software, namely the WTC tool-suite [24], the Termite tool [25] and the Greenarray framework [26]. All the algorithms are tested on large benchmarks and are available for testing for the community.

References

[AAG12] Guillaume Andrieu, Christophe Alias, and Laure Gonnord. SToP: Scalable termination analysis of (C) programs (tool presentation). In *International Workshop on Tools for Automatic Program Analysis*, TAPAS '12, Deauville, France, September 2012.

[AB03] Christophe Alias and Denis Barthou. Algorithm recognition based on demand-driven data-flow analysis. In *Proceedings of the 10th Working Conference on Reverse Engineering*, WCRE '03, pages 296–305, 2003.

[ABD07] Christophe Alias, Fabrice Baray, and Alain Darte. Bee+cl@k: An implementation of lattice-based array contraction in the source-to-source translator rose. In *Proceedings of the ACM SIGPLAN/SIGBED Conference on Languages, Compilers, and Tools for Embedded Systems*, LCTES '07, pages 73–82, 2007.

[ADFG10] Christophe Alias, Alain Darte, Paul Feautrier, and Laure Gonnord. Multi-dimensional rankings, program termination, and complexity bounds of flowchart programs. In *Proceedings of the 17th International Conference on Static Analysis*, SAS '10, pages 117–133, 2010.

[AFG16] Christophe Alias, Carsten Fuhs, and Laure Gonnord. Estimation of Parallel Complexity with Rewriting Techniques. In *Proceedings of the 15th Workshop on Termination*, WST '16, pages 2:1–2:5, Obergurgl, Austria, September 2016.

[AG00] Thomas Arts and Jürgen Giesl. Termination of term rewriting using dependency pairs. *Theoretical Computer Science*, 236:133–178, 2000.

[AGLN11] Beatriz Alarcón, Raúl Gutiérrez, Salvador Lucas, and Rafael Navarro-Marset. Proving termination properties with mu-term. In *Revised Selected Papers of the 13th International Conference Algebraic Methodology and Software Technology*, AMAST '10, pages 201–208, 2011.

[ATD04] Manuel Arenaz, Juan Touriño, and Ramón Doallo. Compiler support for parallel code generation through kernel recognition. In *Proceedings of the 18th International Parallel and Distributed Processing Symposium*, IPDPS '04, pages 79–, April 2004.

[ATNW09] Cédric Augonnet, Samuel Thibault, Raymond Namyst, and Pierre-André Wacrenier. StarPU: A unified platform for task scheduling on heterogeneous multicore architectures. In *Proceedings of the 15th International Euro-Par Conference*, Euro-Par '09, pages 863–874, August 2009.

[BBD+13] George Bosilca, Aurélien Bouteiller, Anthony Danalis, Mathieu Faverge, Thomas Herault, and Jack J. Dongarra. PaRSEC: Exploiting heterogeneity to enhance scalability. *Computing in Science Engineering*, 15(6):36–45, Nov 2013.

[24] `http://compsys-tools.ens-lyon.fr/wtc/` and `http://compsys-tools.ens-lyon.fr/rank/index.php`
[25] `https://termite-analyser.github.io`
[26] `https://code.google.com/p/ecosoc/wiki/GreenArrays`

[BCC00] Denis Barthou, Albert Cohen, and Jean-François Collard. Maximal Static Expansion. *International Journal of Parallel Programming*, 28(3):213–243, June 2000.

[BCC+07] Josh Berdine, Cristiano Calcagno, Byron Cook, Dino Distefano, Peter W. O'Hearn, Thomas Wies, and Hongseok Yang. Shape analysis for composite data structures. In *Proceedings of the 19th International Conference on Computer Aided Verification*, CAV '07, pages 178–192. Springer, 2007.

[BCF97] Denis Barthou, Jean-François Collard, and Paul Feautrier. Fuzzy array dataflow analysis. *Journal of Parallel and Distributed Computing*, 40(2):210–226, 1997.

[BCL87] Ahlem Ben Cherifa and Pierre Lescanne. Termination of rewriting systems by polynomial interpretations and its implementation. *Science of Computer Programming*, 9(2):137–159, 1987.

[BDJ13] Matko Botinčan, Mike Dodds, and Suresh Jagannathan. Proof-directed parallelization synthesis by separation logic. *ACM Transactions on Programming Languages and Systems*, 35(2):8:1–8:60, 2013.

[BEF+16] Marc Brockschmidt, Fabian Emmes, Stephan Falke, Carsten Fuhs, and Jürgen Giesl. Analyzing runtime and size complexity of integer programs. *ACM Transactions on Programming Languages and Systems*, 38(4):13:1–13:50, 2016.

[BF98] Pierre Boulet and Paul Feautrier. Scanning polyhedra without do-loops. In *IEEE International Conference on Parallel Architectures and Compilation Techniques (PACT '98)*, Washington, DC, 1998.

[BFR02] Denis Barthou, Paul Feautrier, and Xavier Redon. On the equivalence of two systems of affine recurrence equations (research note). In *Proceedings of the 8th International Euro-Par Conference on Parallel Processing*, Euro-Par '02, pages 309–313, London, UK, 2002. Springer-Verlag.

[BGK+12] Denis Barthou, Gilbert Grosdidier, Michael Kruse, Olivier Pène, and Claude Tadonki. QIRAL: A high level language for lattice QCD code generation. *CoRR*, abs/1208.4035, 2012.

[BHRS08] Uday Bondhugula, Albert Hartono, Jagannathan Ramanujam, and Ponnuswamy Sadayappan. A practical automatic polyhedral parallelizer and locality optimizer. In *Proceedings of the 29th ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '08, pages 101–113, 2008.

[BMOG12] Marc Brockschmidt, Richard Musiol, Carsten Otto, and Jürgen Giesl. Automated termination proofs for Java programs with cyclic data. In *Proceedings of the 24th International Conference on Computer Aided Verification*, CAV '12, pages 105–122, 2012.

[BN98] Franz Baader and Tobias Nipkow. *Term rewriting and all that.* Cambridge University Press, 1998.

[BOG11] Marc Brockschmidt, Carsten Otto, and Jürgen Giesl. Modular termination proofs of recursive Java Bytecode programs by term rewriting. In *Proceedings of the 22nd International Conference on Rewriting Techniques and Applications*, RTA '11, pages 155–170, 2011.

[BPCB10] Mohamed-Walid Benabderrahmane, Louis-Noël Pouchet, Albert Cohen, and Cédric Bastoul. The polyhedral model is more widely applicable than you think. In *Proceedings of the 19th Joint European Conference on Theory and Practice of Software, International Conference on Compiler Construction*, CC'10/ETAPS'10, pages 283–303, 2010.

[BYR+11] Vamshi Basupalli, Tomofumi Yuki, Sanjay Rajopadhye, Antoine Morvan, Steven Derrien, Patrice Quinton, and Dave Wonnacott. ompVerify: Polyhedral analysis for the OpenMP programmer. In *Proceedings of the 7th International Workshop on OpenMP*, IWOMP '11, pages 37–53, June 2011.

[CC98] Albert Cohen and Jean-François Collard. Instance-wise Reaching Definition Analysis for Recursive Programs using Context-free Transductions. In *Proceedings of the 1998 International Conference on Parallel Architectures and Compilation Techniques*, PACT '98, pages 332–340, Paris, France, 1998. Best student paper award.

[Coh99] Albert Cohen. *Program Analysis and Transformation: from the Polytope Model to Formal Languages.* PhD thesis, Université de Versailles, France, December 1999.

[CSB11] Matthias Christen, Olaf Schenk, and Helmar Burkhart. PATUS: A code generation and autotuning framework for parallel iterative stencil computations on modern microarchitectures. In *2011 IEEE International Parallel Distributed Processing Symposium*, IPDPS '11, pages 676–687, May 2011.

[CSRB+17] Juan Manuel Martinez Caamano, Aravind Sukumaran-Rajam, Artiom Baloian, Manuel Selva, and Philippe Clauss. Apollo: Automatic speculative polyhedral loop optimizer. In *7th International Workshop on Polyhedral Compilation Techniques*, IMPACT '17, Stockholm, Sweden, January 2017.

[Dar10] Alain Darte. Understanding loops: The influence of the decomposition of Karp, Miller, and Winograd. In *8th ACM/IEEE International Conference on Formal Methods and Models for Codesign*, MEMOCODE '10, pages 139–148, Grenoble, France, July 2010. IEEE Computer Society. Invited paper and keynote talk.

[Dar11] Alain Darte. Optimal parallelism detection in nested loops. In David Padua, editor, *Encyclopedia of Parallel Programming*. Springer, 2011.

[DI15] Alain Darte and Alexandre Isoard. Exact and approximated data-reuse optimizations for tiling with parametric sizes. In *Proceedings of the 24th International Conference on Compiler Construction*, CC '15, pages 151–170, April 2015.

[DMRM14] M. Dardaillon, K. Marquet, T. Risset, and H.-P. Martin, J. Charles. *Software-Defined and Cognitive Radio Technologies for Dynamic Spectrum Access and Management*, chapter Cognitive Radio Programming: Existing Solutions and Open Issues. 2014.

[DSV05] Alain Darte, Robert Schreiber, and Gilles Villard. Lattice-based memory allocation. *IEEE Transactions on Computers*, 54(10):1242–1257, 2005.

[EL94] Marc Engels and Rudy Lauwereins. Cycle-static dataflow: model and implementation. In *Conference Record of the Twenty-Eighth Asilomar Conference on Signals, Systems and Computers, 1994.*, volume 1, pages 503–507, Oct 1994.

[Fea91] Paul Feautrier. Dataflow analysis of array and scalar references. *International Journal of Parallel Programming*, 20(1):23–53, 1991.

[Fea92a] Paul Feautrier. Some efficient solutions to the affine scheduling problem, I, one-dimensional time. *International Journal of Parallel Programming*, 21(5):313–348, October 1992.

[Fea92b] Paul Feautrier. Some efficient solutions to the affine scheduling problem, II, multi-dimensional time. *International Journal of Parallel Programming*, 21(6):389–420, December 1992.

[Fea98] Paul Feautrier. A parallelization framework for recursive tree programs. In *Proceedings of the 4th International Euro-Par Conference on Parallel Processing*, Euro-Par '98, pages 470–479, 1998.

[FGM+07] Carsten Fuhs, Jürgen Giesl, Aart Middeldorp, Peter Schneider-Kamp, René Thiemann, and Harald Zankl. SAT solving for termination analysis with polynomial interpretations. In *Proceedings of the 10th International Conference on Theory and Applications of Satisfiability Testing*, SAT '07, pages 340–354, 2007.

[FGP+09] Carsten Fuhs, Jürgen Giesl, Martin Plücker, Peter Schneider-Kamp, and Stephan Falke. Proving termination of integer term rewriting. In *Proceedings of the 20th International Conference on Rewriting Techniques and Applications*, RTA '09, pages 32–47, 2009.

[FL11] Paul Feautrier and Christian Lengauer. The polyhedron model. In David Padua, editor, *Encyclopedia of Parallel Programming*. Springer, 2011.

[GAB+17] Jürgen Giesl, Cornelius Aschermann, Marc Brockschmidt, Fabian Emmes, Florian Frohn, Carsten Fuhs, Jera Hensel, Carsten Otto, Martin Plücker, Peter Schneider-Kamp, Thomas Ströder, Stephanie Swiderski, and René Thiemann. Analyzing program termination and complexity automatically with AProVE. *Journal of Automated Reasoning*, 58(1):3–31, 2017.

[GBP07] Thierry Gautier, Xavier Besseron, and Laurent Pigeon. KAAPI: A thread scheduling runtime system for data flow computations on cluster of multi-processors. In *Proceedings of the 2007 International Workshop on Parallel Symbolic Computation*, PASCO '07, pages 15–23, 2007.

[GMR15] Laure Gonnord, David Monniaux, and Gabriel Radanne. Synthesis of ranking functions using extremal counterexamples. In *Proceedings of the 36th ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '15. ACM, 2015.

[GRS+11] Jürgen Giesl, Matthias Raffelsieper, Peter Schneider-Kamp, Stephan Swiderski, and René Thiemann. Automated termination proofs for Haskell by term rewriting. *ACM Transactions on Programming Languages and Systems*, 33(2):7:1–7:39, 2011.

[GSLD11] Thierry Goubier, Renaud Sirdey, Stéphane Louise, and Vincent David. σc: A programming model and language for embedded manycores. In *Algorithms and Architectures for Parallel Processing*, pages 385–394. Springer, 2011.

[GSS+12] Jürgen Giesl, Thomas Ströder, Peter Schneider-Kamp, Fabian Emmes, and Carsten Fuhs. Symbolic evaluation graphs and term rewriting: A general methodology for analyzing logic programs. In *Proceedings of the 14th International Symposium on Principles and Practice of Declarative Programming*, PPDP '12, pages 1–12, 2012.

[Hur09] Clément Hurlin. Automatic parallelization and optimization of programs by proof rewriting. In *Proceedings of the 16th International Symposium on Static Analysis*, SAS '09, pages 52–68. 2009.

[IAR14] Guillaume Iooss, Christophe Alias, and Sanjay Rajopadhye. On program equivalence with reductions. In *Proceedings of the 21st International Static Analysis Symposium*, SAS '14, Munich, Germany, September 2014.

[IRAZ14] Guillaume Iooss, Sanjay Rajopadhye, Christophe Alias, and Yun Zou. CART: Constant aspect ratio tiling. In Sanjay Rajopadhye and Sven Verdoolaege, editors, *4th International Workshop on Polyhedral Compilation Techniques*, IMPACT '14, Vienna, Austria, January 2014.

[JCD+14] Alexandra Jimborean, Philippe Clauss, Jean-François Dollinger, Vincent Loechner, and Martinez Juan Manuel. Dynamic and speculative polyhedral parallelization using compiler-generated skeletons. *International Journal of Parallel Programming*, 42(4):529–545, August 2014.

[JM82] Neil D. Jones and Steven S. Muchnick. A flexible approach to interprocedural data flow analysis and programs with recursive data structures. In *Proceedings of the 9th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '82, pages 66–74, New York, NY, USA, 1982. ACM.

[KMW67] Richard M. Karp, Raymond E. Miller, and Shmuel Winograd. The organization of computations for uniform recurrence equations. *Journal of the ACM*, 14(3):563–590, 1967.

[KN13] Cynthia Kop and Naoki Nishida. Term rewriting with logical constraints. In *Proceedings of the 9th International Symposium on Frontiers of Combining Systems*, FroCoS '13, pages 343–358, 2013.

[KSZM09] Martin Korp, Christian Sternagel, Harald Zankl, and Aart Middeldorp. Tyrolean Termination Tool 2. In *Proceedings of the 20th International Conference on Rewriting Techniques and Applications*, RTA '09, pages 295–304, 2009.

[Lan79] Dallas S. Lankford. On proving term rewriting systems are Noetherian. Technical Report MTP-3, Louisiana Technical University, Ruston, LA, USA, 1979.

[MGR+12] G. R. Mudalige, M. B. Giles, I. Reguly, C. Bertolli, and P. H. J. Kelly. OP2: An active library framework for solving unstructured mesh-based applications on multi-core and many-core architectures. In *Proceedings of the 2012 Innovative Parallel Computing*, InPar '12, pages 1–12, May 2012.

[Mor07] Lionel Robert Morel. Array iterators in Lustre: From a language extension to its exploitation in validation. *EURASIP J. Emb. Sys.*, 2007, 2007.

[NMS+14] Henrique Nazaré, Izabela Maffra, Willer Santos, Leonardo Barbosa, Laure Gonnord, and Fernando Magno Quintão Pereira. Validation of memory accesses through symbolic analyses. In *Proceedings of the 2014 ACM International Conference on Object Oriented Programming Systems Languages and Applications*, OOPSLA '14, pages 791–809. ACM, 2014.

[OBvEG10] Carsten Otto, Marc Brockschmidt, Christian von Essen, and Jürgen Giesl. Automated termination analysis of Java Bytecode by term rewriting. In *Proceedings of the 21st International Conference on Rewriting Techniques and Applications*, RTA '10, pages 259–276, 2010.

[PC13] Antoniu Pop and Albert Cohen. Openstream: Expressiveness and data-flow compilation of openmp streaming programs. *TACO*, 9(4):53, 2013.

[PNK+11] Keshav Pingali, Donald Nguyen, Milind Kulkarni, Martin Burtscher, M. Amber Hassaan, Rashid Kaleem, Tsung-Hsien Lee, Andrew Lenharth, Roman Manevich, Mario Méndez-Lojo, Dimitrios Prountzos, and Xin Sui. The tao of parallelism in algorithms. In *Proceedings of the 32nd ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '11, pages 12–25, New York, NY, USA, 2011. ACM.

[Qui84] Patrice Quinton. Automatic synthesis of systolic arrays from uniform recurrent equations. In *Proceedings of the 11th Annual International Symposium on Computer Architecture*, ISCA '84, pages 208–214, 1984.

[Raj89] Sanjay V. Rajopadhye. Synthesizing systolic arrays with control signals from recurrence equations. *Distributed Computing*, 3(2):88–105, 1989.

[RAPG14] Raphael Ernani Rodrigues, Péricles Alves, Fernando Pereira, and Laure Gonnord. Real-world loops are easy to predict: A case study. In *Proceedings of the 14th International Workshop on Termination*, WST '14, Vienna, Austria, July 2014.

[RGK11] Sanjay Rajopadhye, Samik Gupta, and Dae-Gon Kim. Alphabets: An extended polyhedral equational language. *2013 IEEE International Symposium on Parallel & Distributed Processing, Workshops and Phd Forum*, 0:656–664, 2011.

[RGP16] Rodrigo C. O. Rocha, Luís Fabrício Wanderley Góes, and Fernando Magno Quintão Pereira. An algebraic framework for parallelizing recurrence in functional programming. In *Proceedings of the 20th Brazilian Symposium on Programming Languages*, SBLP '16, pages 140–155, 2016.

[RHAR06] Silvius Rus, Guobin He, Christophe Alias, and Lawrence Rauchwerger. Region array SSA. In *Proceedings of the 15th International Conference on Parallel Architectures and Compilation Techniques*, PACT '06, 2006.

[RRH03] Silvius Rus, Lawrence Rauchwerger, and Jay Hoeflinger. Hybrid analysis: Static and dynamic memory reference analysis. *International Journal of Parallel Programming*, 31(4):251–283, 2003.

[SESK+12] Thomas Ströder, Fabian Emmes, Peter Schneider-Kamp, Jürgen Giesl, and Carsten Fuhs. A linear operational semantics for termination and complexity analysis of ISO Prolog. In *Proceedings of the 21st International Conference on Logic-Based Program Synthesis and Transformation*, LOPSTR '11, pages 237–252, 2012.

[SGB+17] Thomas Ströder, Jürgen Giesl, Marc Brockschmidt, Florian Frohn, Carsten Fuhs, Jera Hensel, Peter Schneider-Kamp, and Cornelius Aschermann. Automatically proving termination and memory safety for programs with pointer arithmetic. *Journal of Automated Reasoning*, 58(1):33–65, 2017.

[TCK+11] Yuan Tang, Rezaul Alam Chowdhury, Bradley C. Kuszmaul, Chi-Keung Luk, and Charles E. Leiserson. The pochoir stencil compiler. In *Proceedings of the 23rd Annual ACM Symposium on Parallelism in Algorithms and Architectures*, SPAA '11, pages 117–128, 2011.

[TS09] René Thiemann and Christian Sternagel. Certification of termination proofs using CeTA. In *Proceedings of the 22nd International Conference on Theorem Proving in Higher Order Logics*, TPHOLs '09, pages 452–468, 2009.

[VSHS14] Anand Venkat, Manu Shantharam, Mary Hall, and Michelle Mills Strout. Non-affine extensions to polyhedral code generation. In *Proceedings of Annual IEEE/ACM International Symposium on Code Generation and Optimization*, CGO '14, pages 185:185–185:194, New York, NY, USA, February 2014. ACM.

[WBC14] Felix Winterstein, Samuel Bayliss, and George A. Constantinides. Separation logic-assisted code transformations for efficient high-level synthesis. In *Proceedings of the IEEE 22nd International Symposium on Field-Programmable Custom Computing Machines*, FCCM '14. IEEE, 2014.

[YFRS13] Tomofumi Yuki, Paul Feautrier, Sanjay Rajopadhye, and Vijay Saraswat. Array dataflow analysis for polyhedral X10 programs. In *Proceedings of the 18th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, PPoPP '13, pages 23–34, February 2013.