Activity Report 2016

Team CAMUS

Compilation pour les Architectures MUlti-cœurS

Inria teams are typically groups of researchers working on the definition of a common project, and objectives, with the goal to arrive at the creation of a project-team. Such project-teams may include other partners (universities or research institutions).

RESEARCH CENTER
Nancy - Grand Est

THEME
Architecture, Languages and Compilation

Table of contents

Team CAMUS

Creation of the Team: 2009 July 01

Keywords:

Computer Science and Digital Science:

1.1.1. - Multicore
1.1.4. - High performance computing
2.1.1. - Semantics of programming languages
2.1.6. - Concurrent programming
2.2.1. - Static analysis
2.2.3. - Run-time systems
2.2.4. - Parallel architectures
2.2.5. - GPGPU, FPGA, etc.
2.2.6. - Adaptive compilation

Other Research Topics and Application Domains:

4.5.1. - Green computing
6.1.1. - Software engineering
6.6. - Embedded systems

1. Members

Research Scientists

Arthur Charguéraud [Inria, Researcher, since Oct 2016]
Jens Gustedt [Inria, Senior Researcher, HDR]

Faculty Members

Philippe Clauss [Team leader, Univ. Strasbourg, Professor, HDR]
Cédric Bastoul [Univ. Strasbourg, Professor, HDR]
Alain Ketterlin [Univ. Strasbourg, Assistant Professor]
Vincent Loechner [Univ. Strasbourg, Assistant Professor]
Nicolas Magaud [Univ. Strasbourg, Assistant Professor]
Julien Narboux [Univ. Strasbourg, Assistant Professor]
Éric Violard [Univ. Strasbourg, Assistant Professor, HDR]

Engineer

Artiom Baloian [Inria, until Oct 2016]

PhD Students

Yann Barsamian [Univ. Strasbourg]
Luis Esteban Campostrini [Inria, until Jun 2016]
Paul Godard [Caldera, from Sep 2016]
Juan Manuel Martinez Caamaño [Univ. Strasbourg, until Oct 2016]
Harenome Ranaivoarivony-Razanajato [Univ. Strasbourg, from Oct 2016]
Mariem Saied [Inria & Univ. Strasbourg]
Daniel Salas [INSERM]
Maxime Schmitt [Univ. Strasbourg, from Sep 2016]

Post-Doctoral Fellows

Julien Pagès [Univ. Strasbourg, from Oct 2016]

Manuel Selva [Inria, from Sep 2016]

Administrative Assistant

Véronique Constant [Inria]

2. Overall Objectives

2.1. Overall Objectives

The CAMUS team is focusing on developing, adapting and extending automatic parallelizing and optimizing techniques, as well as proof and certification methods, for the efficient use of current and future multicore processors.

The team's research activities are organized into five main issues that are closely related to reach the following objectives: performance, correction and productivity. These issues are: static parallelization and optimization of programs (where all statically detected parallelisms are expressed as well as all "hypothetical" parallelisms which would be eventually taken advantage of at runtime), profiling and execution behavior modeling (where expressive representation models of the program execution behavior will be used as engines for dynamic parallelizing processes), dynamic parallelization and optimization of programs (such transformation processes running inside a virtual machine), and finally program transformations proof (where the correction of many static and dynamic program transformations has to be ensured).

3. Research Program

3.1. Research Directions

The various objectives we are expecting to reach are directly related to the search of adequacy between the sofware and the new multicore processors evolution. They also correspond to the main research directions suggested by Hall, Padua and Pingali in [24]. Performance, correction and productivity must be the users' perceived effects. They will be the consequences of research works dealing with the following issues:

- Issue 1: Static Parallelization and Optimization
- Issue 2: Profiling and Execution Behavior Modeling
- Issue 3: Dynamic Program Parallelization and Optimization, Virtual Machine
- Issue 4: Proof of Program Transformations for Multicores

Efficient and correct applications development for multicore processors needs stepping in every application development phase, from the initial conception to the final run.

Upstream, all potential parallelism of the application has to be exhibited. Here static analysis and transformation approaches (issue 1) must be processed, resulting in a *multi-parallel* intermediate code advising the running virtual machine about all the parallelism that can be taken advantage of. However the compiler does not have much knowledge about the execution environment. It obviously knows the instruction set, it can be aware of the number of available cores, but it does not know the effective available resources at any time during the execution (memory, number of free cores, etc.).

That is the reason why a "virtual machine" mechanism will have to adapt the application to the resources (issue 3). Moreover the compiler will be able to take advantage only of a part of the parallelism induced by the application. Indeed some program information (variables values, accessed memory adresses, etc.) being available only at runtime, another part of the available parallelism will have to be generated on-the-fly during the execution, here also, thanks to a dynamic mechanism.

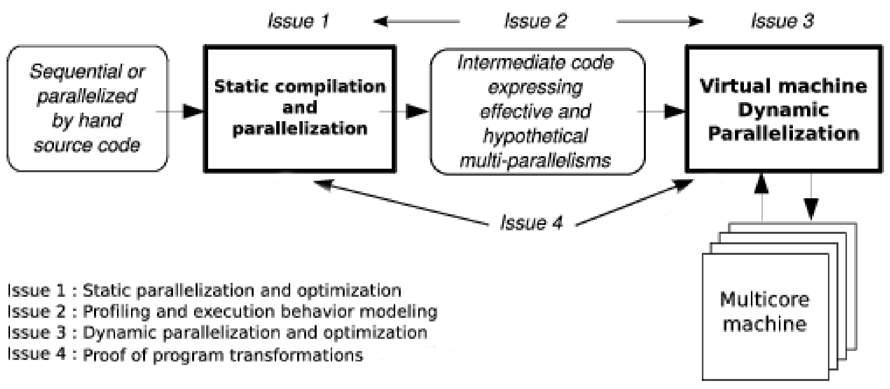
Issue 1
Issue 2
Issue 3
Sequential or parallelized by hand source code
Static compilation and parallelization
Intermediate code expressing effective and hypothetical multi-parallelisms
Virtual machine Dynamic Parallelization
Issue 4
Multicore machine
Issue 1 : Static parallelization and optimization
Issue 2 : Profiling and execution behavior modeling
Issue 3 : Dynamic parallelization and optimization
Issue 4 : Proof of program transformations

Figure 1. Automatic parallelizing steps for multicore architectures

This on-the-fly parallelism extraction will be performed using speculative behavior models (issue 2), such models allowing to generate speculative parallel code (issue 3). Between our behavior modeling objectives, we can add the behavior monitoring, or profiling, of a program version. Indeed current and future architectures complexity avoids assuming an optimal behavior regarding a given program version. A monitoring process will allow to select on-the-fly the best parallelization.

These different parallelizing steps are schematized on figure 1.

Our project lies on the conception of a production chain for efficient execution of an application on a multicore architecture. Each link of this chain has to be formally verified in order to ensure correction as well as efficiency. More precisely, it has to be ensured that the compiler produces a correct intermediate code, and that the virtual machine actually performs the parallel execution semantically equivalent to the source code: every transformation applied to the application, either statically by the compiler or dynamically by the virtual machine, must preserve the initial semantics. They must be proved formally (issue 4).

In the following, those different issues are detailed while forming our global and long term vision of what has to be done.

3.2. Static Parallelization and Optimization

Participants: Vincent Loechner, Philippe Clauss, Éric Violard, Cédric Bastoul, Arthur Charguéraud.

Static optimizations, from source code at compile time, benefit from two decades of research in automatic parallelization: many works address the parallelization of loop nests accessing multi-dimensional arrays, and these works are now mature enough to generate efficient parallel code [23]. Low-level optimizations, in the assembly code generated by the compiler, have also been extensively dealt for single-core and require few adaptations to support multicore architectures. Concerning multicore specific parallelization, we propose to explore two research directions to take full advantage of these architectures: adapting parallelization to multicore architecture and expressing many potential parallelisms.

3.3. Profiling and Execution Behavior Modeling

Participants: Alain Ketterlin, Philippe Clauss, Manuel Selva.

The increasing complexity of programs and hardware architectures makes it ever harder to characterize beforehand a given program's run time behavior. The sophistication of current compilers and the variety of transformations they are able to apply cannot hide their intrinsic limitations. As new abstractions like transactional memories appear, the dynamic behavior of a program strongly conditions its observed performance. All these reasons explain why empirical studies of sequential and parallel program executions have been considered increasingly relevant. Such studies aim at characterizing various facets of one or several program runs, *e.g.*, memory behavior, execution phases, etc. In some cases, such studies characterize more the compiler than the program itself. These works are of tremendous importance to highlight all aspects that escape static analysis, even though their results may have a narrow scope, due to the possible incompleteness of their input data sets.

3.4. Dynamic Parallelization and Optimization, Virtual Machine

Participants: Manuel Selva, Juan Manuel Martinez Caamaño, Luis Esteban Campostrini, Artiom Baloian, Mariem Saied, Daniel Salas, Philippe Clauss, Jens Gustedt, Vincent Loechner, Alain Ketterlin.

This link in the programming chain has become essential with the advent of the new multicore architectures. Still being considered as secondary with mono-core architectures, dynamic analysis and optimization are now one of the keys for controling those new mechanisms complexity. From now on, performed instructions are not only dedicated to the application functionalities, but also to its control and its transformation, and so in its own interest. Behaving like a computer virus, such a process should rather be qualified as a "vitamin". It perfectly knows the current characteristics of the execution environment and owns some qualitative information thanks to a behavior modeling process (issue 2). It appends a significant part of optimizing ability compared to a static compiler, while observing live resources availability evolution.

3.5. Proof of Program Transformations for Multicores

Participants: Éric Violard, Alain Ketterlin, Julien Narboux, Nicolas Magaud, Arthur Charguéraud.

Our main objective consists in certifying the critical modules of our optimization tools (the compiler and the virtual machine). First we will prove the main loop transformation algorithms which constitute the core of our system.

The optimization process can be separated into two stages: the transformations consisting in optimizing the sequential code and in exhibiting parallelism, and those consisting in optimizing the parallel code itself. The first category of optimizations can be proved within a sequential semantics. For the other optimizations, we need to work within a concurrent semantics. We expect the first stage of optimizations to produce data-race free code. For the second stage of optimizations, we will first assume that the input code is data-race free. We will prove those transformations using Appel's concurrent separation logic [25]. Proving transformations involving program which are not data-race free will constitute a longer term research goal.

4. Application Domains

4.1. Application Domains

Performance being our main objective, our developments' target applications are characterized by intensive computation phases. Such applications are numerous in the domains of scientific computations, optimization, data mining and multimedia.

Applications involving intensive computations are necessarily high energy consumers. However this consumption can be significantly reduced thanks to optimization and parallelization. Although this issue is not our prior objective, we can expect some positive effects for the following reasons:

- Program parallelization tries to distribute the workload equally among the cores. Thus an equivalent performance, or even a better performance, to a sequential higher frequency execution on one single core, can be obtained.
- Memory and memory accesses are high energy consumers. Lowering the memory consumption, lowering the number of memory accesses and maximizing the number of accesses in the low levels of the memory hierarchy (registers, cache memories) have a positive consequence on execution speed, but also on energy consumption.

5. Highlights of the Year

5.1. Highlights of the Year

Arthur Charguéraud, Inria Research Scientist, has joined the team in October 2016.

The first release of the speculative polyhedral loop parallelizer *Apollo* [1] has been published under the BSD 3-Clause Open Source License.

5.1.1. Awards

BEST PAPER AWARD:

[13]
J. M. MARTINEZ CAAMAÑO, W. WOLFF, P. CLAUSS. *Code Bones: Fast and Flexible Code Generation for Dynamic and Speculative Polyhedral Optimization*, in "Euro-Par 2016", Grenoble, France, SPRINGER-VERLAG (editor), Proceedings of the 22nd International Conference Euro-Par 2016: Parallel Processing, August 2016, vol. 9833, 12 p. [*DOI :* 10.1007/978-3-319-43659-3_17], https://hal.inria.fr/hal-01377656

6. New Software and Platforms

6.1. Apollo

Automatic speculative POLyhedral Loop Optimizer
KEYWORD: Automatic parallelization
FUNCTIONAL DESCRIPTION

Apollo is dedicated to automatic, dynamic and speculative parallelization of loop nests that cannot be handled efficiently at compile-time. It is composed of a static part consisting of specific passes in the LLVM compiler suite, plus a modified Clang frontend, and a dynamic part consisting of a runtime system. It can apply on-the-fly any kind of polyhedral transformations, including tiling, and can handle nonlinear loops, as while-loops referencing memory through pointers and indirections.

- Participants: Manuel Selva, Juan Manuel Martinez Caamaño, Artiom Baloian, and Philippe Clauss
- Contact: Philippe Clauss
- URL: http://apollo.gforge.inria.fr

6.2. CLooG

Code Generator in the Polyhedral Model
FUNCTIONAL DESCRIPTION

[1] http://apollo.gforge.inria.fr

CLooG is a free software and library to generate code (or an abstract syntax tree of a code) for scanning Z-polyhedra. That is, it finds a code (*e.g.* in C, FORTRAN...) that reaches each integral point of one or more parameterized polyhedra. CLooG has been originally written to solve the code generation problem for optimizing compilers based on the polyhedral model. Nevertheless it is used now in various area *e.g.* to build control automata for high-level synthesis or to find the best polynomial approximation of a function. CLooG may help in any situation where scanning polyhedra matters. While the user has full control on generated code quality, CLooG is designed to avoid control overhead and to produce a very effective code. CLooG is widely used (including by GCC and LLVM compilers), disseminated (it is installed by default by the main Linux distributions) and considered as the state of the art in polyhedral code generation.

- Participant: Cédric Bastoul
- Contact: Cédric Bastoul
- URL: http://www.cloog.org

6.3. Clan

A Polyhedral Representation Extraction Tool for C-Based High Level Languages
FUNCTIONAL DESCRIPTION

Clan is a free software and library which translates some particular parts of high level programs written in C, C++, C# or Java into a polyhedral representation called OpenScop. This representation may be manipulated by other tools to, *e.g.*, achieve complex analyses or program restructurations (for optimization, parallelization or any other kind of manipulation). It has been created to avoid tedious and error-prone input file writing for polyhedral tools (such as CLooG, LeTSeE, Candl etc.). Using Clan, the user has to deal with source codes based on C grammar only (as C, C++, C# or Java). Clan is notably the frontend of the two major high-level compilers Pluto and PoCC.

- Participants: Cédric Bastoul
- Contact: Cédric Bastoul
- URL: http://icps.u-strasbg.fr/people/bastoul/public_html/development/clan/

6.4. Clay

Chunky Loop Alteration wizardrY
FUNCTIONAL DESCRIPTION

Clay is a free software and library devoted to semi-automatic optimization using the polyhedral model. It can input a high-level program or its polyhedral representation and transform it according to a transformation script. Classic loop transformations primitives are provided. Clay is able to check for the legality of the complete sequence of transformation and to suggest corrections to the user if the original semantics is not preserved.

- Participant: Cédric Bastoul
- Contact: Cédric Bastoul
- URL: http://icps.u-strasbg.fr/people/bastoul/public_html/development/clay/

6.5. IBB

Iterate-But-Better
FUNCTIONAL DESCRIPTION

IBB is a source-to-source xfor compiler which automatically translates any C source code containing xfor-loops into an equivalent source code where xfor-loops have been transformed into equivalent for-loops.

- Participants: Philippe Clauss and Cédric Bastoul
- Contact: Philippe Clauss
- URL: http://xfor.gforge.inria.fr

6.6. OpenScop

A Specification and a Library for Data Exchange in Polyhedral Compilation Tools
FUNCTIONAL DESCRIPTION

OpenScop is an open specification that defines a file format and a set of data structures to represent a static control part (SCoP for short), i.e., a program part that can be represented in the polyhedral model. The goal of OpenScop is to provide a common interface to the different polyhedral compilation tools in order to simplify their interaction. To help the tool developers to adopt this specification, OpenScop comes with an example library (under 3-clause BSD license) that provides an implementation of the most important functionalities necessary to work with OpenScop.

- Participant: Cédric Bastoul
- Contact: Cédric Bastoul
- URL: http://icps.u-strasbg.fr/people/bastoul/public_html/development/openscop/

6.7. PolyLib

The Polyhedral Library
FUNCTIONAL DESCRIPTION

PolyLib is a C library of polyhedral functions, that can manipulate unions of rational polyhedra of any dimension. It was the first to provide an implementation of the computation of parametric vertices of a parametric polyhedron, and the computation of an Ehrhart polynomial (expressing the number of integer points contained in a parametric polytope) based on an interpolation method. Vincent Loechner is the maintainer of this software.

- Participant: Vincent Loechner
- Contact: Vincent Loechner
- URL: http://icps.u-strasbg.fr/PolyLib/

6.8. ORWL and P99

ORWL is a reference implementation of the Ordered Read-Write Lock tools as described in [1]. The macro definitions and tools for programming in C99 that have been implemented for ORWL have been separated out into a toolbox called P99. ORWL is intended to become opensource, once it will be in a publishable state. P99 is available under a QPL at http://p99.gforge.inria.fr/.

- Participants: Jens Gustedt, Mariem Saied, Daniel Salas
- Contact: Jens Gustedt
- http://p99.gforge.inria.fr/, http://orwl.gforge.inria.fr/

6.9. Stdatomic and Musl

We implement the library side of the C11 atomic interface. It needs compiler support for the individual atomic operations and provides library support for the cases where no low-level atomic instruction is available and a lock must be taken.

- This implementation builds entirely on the ABIs of the gcc compiler for atomics.
- It provide all function interfaces that the gcc ABIs and the C standard need.
- For compilers that don't offer the direct language support for atomics it provides a syntactically reduced but fully functional approach to atomic operations.
- At the core of the library is a new and very efficient futex-based lock algorithm that is implemented for the Linux operating system.

A description of the new lock algorithm has been given in [2]. A short version of it has been presented at SAC'16.

The primary target of this library is an integration into musl to which we also contribute. It is a re-implementation of the C library as it is described by the C and POSIX standards. It is *lightweight*, *fast*, *simple*, *free*, and strives to be correct in the sense of standards-conformance and safety. Musl is production quality code that is mainly used in the area of embedded device. It gains more market share also in other area, *e.g.* there are now Linux distributions that are based on musl instead of Gnu LibC.

- Participant: Jens Gustedt
- Contact: Jens Gustedt
- http://stdatomic.gforge.inria.fr/, http://www.musl-libc.org/

7. New Results

7.1. Formal Proofs about Happens-before in Explicitly Parallel Polyhedral Programs

Participants: Éric Violard, Alain Ketterlin.

Automatic parallelization has traditionally focused on sequential programs, but the widespread availability of explicitly parallel programming languages (such as OpenMP, Cilk, X10, and others) has led researchers to consider also the optimization and re-parallelization of parallel source programs. Most of these languages have constructions for parallel loops and parallel sections, with the accompanying synchronization primitives. The X10 language is especially interesting in this respect, because it provides simple and powerful constructions. Essentially, parallelism is expressed with the help of the `async` construct, whose body is to be executed in a parallel "activity", and the `finish` construct, which acts as a container for activities (and sub-activities) and waits for their completion. These constructions are complemented with "clocks", which are essentially synchronization barriers. Clocks can be used freely, in an unstructured manner, but are best associated with `finish` constructs, where they provide an intuitive and flexible phasing mechanism. In this case, activities spawned with `async` can either inherit or hide the clock provided by the nearest enclosing `finish`.

We are focusing on polyhedral programs, where all control is based on loops whose bounds are affine combinations of the enclosing loop counters and constant parameters. There is a large body of work on optimizing and parallelizing such programs, but most of them focus on sequential loop nests. Introducing X10's parallel constructions defines the class of *explicitly parallel polyhedral programs*, which is the focus of our work. Many polyhedral analyses and optimization techniques rely on the notion of lexicographic order, which is the order of execution of the statements in the source program. For instance, a data-dependence is defined to be an ordered pair of instruction instances that use or define the same data element, such that the first executes *before* the second. The lexicographic order is a purely syntactic characteristic that can be extracted from the source program. When the source program is explicitly parallel, the execution order becomes partial, because two distinct instruction instances can be part of concurrent activities. In this case the ordering is called the *Happens-before* relation. Paul Feautrier and Tomofumi Yuki have provided the first definition of *Happens-before* for explicitly parallel polyhedral programs, which covers the case of X10 programs using `finish` and `async` but without any clock involved. Being purely syntactic, their definition opens the way to the optimization of parallel X10 `finish`-`async` polyhedral programs. The use of clocks, however, introduces a major difficulty. Since clocks define phases of the program, one would like to use the "phase-number" of each instruction instance as an additional dimension, and include this dimension in further analysis. Phase-numbers have analytic forms (for the class of polyhedral programs), but they belong to the class of Ehrhart's quasi-polynomials, i.e., they are outside the polyhedral (affine) model.

We have formalized the class of programs under consideration, as well as all notions pertaining to the definition of the *Happens-before* relation, in Coq, a proof assistant developed at Inria. The formalization includes minimal structures to represent explicitly parallel polyhedral programs, including `finish` and `async`, loops, and simple statements. The definition of the *Happens-before* relation is that of an inductive predicate, parametrized by the computation of phase-numbers, which is left unspecified. To make the connection between the (static) *Happens-before* relation and the (dynamic) position of instruction instances in program traces, we use a single axiom. To reinforce our confidence in this arbitrary component, we also provide a second set of axioms, which we prove is equivalent to the first. The proof is based on an operational semantics, providing the relation between programs and their executions traces. We then prove that when *Happens-before* holds between two (static) instruction instances, then any trace of the program sees the corresponding dynamic instances ordered. We also prove the converse, which makes the definition of *Happens-before* sound and complete.

The Coq source files are kept in an Inria-forge project. Since this is our first effort in formal proofs, it currently amounts to about ten thousands lines of Coq source code. It is not yet clear whether we will publish the proof by itself, or publish an informal version of it as part of our colleagues' work on the use of *Happens-before*. In any case, our short-term plan is to extend the formalization and accompanying theorems and proofs to the case of mixed-programs, where some activities ignore the clock in scope.

7.2. Loop Nests and Integer Polyhedra

Participant: Alain Ketterlin.

The polyhedral model has been found adequate to model a large number of program analyses and transformations. It has now been used for decades in automatic parallelization, locality optimization, high-level code synthesis, and other applications. Thanks to the availability of high-quality software tools, the polyhedral model is now widely used. However, we feel that some of its most fundamental operations need more thorough attention, and possibly new theoretical developments. Even though the translation of loop nests into polyhedra (or unions thereof) obviously use integers only, many algorithms still use an underlying rational (or real) domain. For instance, Fourier-Motzkin variable elimination is defined on rational domains, and its modern incarnation (the Omega test), uses convoluted and costly techniques to compensate for the mishandling of integer variables. When used for projection (for instance during code generation, i.e., turning polyhedra into loop nests), these defects lead to sub-optimal results, with programs including more control than necessary. Overall, we feel that current techniques are inadequate to capture the precise behavior of integer variables.

We have started investigating new representations for inequalities over integer variables, using a notation called "periodic numbers". This notation was invented by Eugène Ehrhart in his classical results on the number of integer points inside integer polyhedra, and rediscovered and generalized by Philippe Clauss in his work on the use of counting for locality optimization and automatic parallelization. Periodic numbers capture all sorts of integer-specific behaviors: for instance, they are especially suitable to represent the seemingly chaotic structure of discrete line intersections, or the modular intersections of parallel hyper-planes. Periodic numbers also have algebraic properties that make them easy to manipulate and combine. We have defined a generalization of affine expressions where the constant term becomes a periodic number: it turns out that this family of expressions has interesting stability properties, that make them especially suitable for variable elimination. We have shown that most problems of Fourier-Motzkin variable elimination are related to the "looseness" of affine inequalities over integer variables, and that periodic numbers can correct this defect. The result is a new representation of inequalities, that makes reasoning with inequalities sound and complete.

An immediate application of our new representation is deciding whether a given integer polyhedron contains an integer point (or: whether a given set of affine constraints on integer variables is feasible). We have developed a straightforward version of Fourier-Motzkin elimination that is always exact. An interesting aspect of this work is that the algorithm is only a slight generalization of the original Fourier-Motzkin elimination, to cover the cases where inequalities have periodic components. We have also extended the basic algorithm to produce arbitrary projections of integer polyhedra. This improves over the Omega elimination strategy in that we are able to produce a provably disjoint union. These interesting properties derive directly from the use of periodic numbers.

Periodic numbers, and periodic linear inequalities, also have applications more directly related to the compilation of affine loop nests. For instance, we have developed a fully-general unswitching transformation. Unswitching a loop containing a conditional amounts to split the loop into one or more new loops such that the conditional has a constant truth value in all loop fragments, and can therefore be removed. The transformation is general in that the resulting program contains only affine loops and periodic linear conditionals. This means that the process can be repeated until obtaining a final version of the loop nest that is completely free of conditionals. We expect this “code generation” strategy, though naive, to remove enough “divergence” to increase existing and enable new applications of vectorization, leading to more efficient code. On the theory side, producing a conditional-free code scanning an arbitrary union of polyhedra has also direct consequences on various polyhedral operations: for instance, computing extrema becomes a trivial task, and linear optimization also falls under this umbrella. We hope to be able to explore these tracks in the near future.

We have developed software making use (and illustrating) our theoretical developments. We expect to share this software with select colleagues very soon, so as to be able to assess the scope of our techniques. Publication of these results is expected in the next year, time permitting. We also expect to extend our current software base to provide a range of integer polyhedra operations (images and pre-images, projection, and linear optimization, mostly). Finally, our middle-term goal is to investigate a formal modeling of the integer polyhedra operations. All algorithms have been kept as simple as possible, favoring elaborate abstractions over complex processing, with the goal of being able to formally specify the fundamental operations.

7.3. Splitting Polyhedra to Generate More Efficient Code

Participants: Harenome Ranaivoarivony-Razanajato, Vincent Loechner, Cédric Bastoul.

Code generation in the polyhedral model takes as input a union of Z-polyhedra and produces a code scanning all of them. Modern code generation tools are heavily relying on polyhedral operations to perform this task. However, these operations are typically provided by general-purpose polyhedral libraries that are not specifically designed to address the code generation problem. In particular, (unions of) polyhedra may be represented in various mathematically equivalent ways which may have different properties with respect to code generation. We investigated this problem and tried to find the best representation of polyhedra to generate an efficient code.

We demonstrated that this problem has been largely under-estimated, showing significant control overhead deviations when using different representations of the same polyhedra. Second, we proposed an improvement to the main algorithm of the state-of-the-art code generation tool CLooG. It generates code with less tests in the inner loops, and aims at reducing control overhead and at simplifying vectorization for the compiler, at the cost of a larger code size. It is based on a smart splitting of the union of polyhedra while recursing on the dimensions.

We implemented our algorithm in CLooG/PolyLib, and compared the performance and size of the generated code to the CLooG/isl version. Our results show that there can be important performance differences between the generated versions. In some cases, our new technique may significantly improve the quality of the generated code, but in some other cases, it may not be adequate compared to the existing solution. Finding other alternatives and chosing the best one remain open problems to be investigated in the future.

7.4. Code-Bones for Fast and Flexible Runtime Code Generation

Participants: Juan Manuel Martinez Caamaño, Artiom Baloian, Philippe Clauss.

We have developed a new runtime code generation technique for speculative loop optimization and parallelization. The main benefit of this technique, compared to previous approaches, is to enable advanced optimizing loop transformations at runtime with an acceptable time overhead. The loop transformations that may be applied are those handled by the polyhedral model. The proposed code generation strategy is based on the generation of *code-bones* at compile-time, which are parametrized code snippets either dedicated to speculation management or to computations of the original target program. These code bones are then instantiated and assembled at runtime to constitute the speculatively-optimized code, as soon as an optimizing polyhedral transformation has been determined. Their granularity threshold is sufficient to apply any polyhedral transformation, while still enabling fast runtime code generation. This approach has been implemented in the speculative loop parallelizing framework Apollo, and published at the conference Euro-Par 2016 where it has been selected as best paper [13]. An extended journal version is currently under review. This is also the main contribution of Juan Manuel Martinez Caamaño's PhD thesis which was defended in September 2016 [8].

7.5. Automatic Collapsing of Non-Rectangular Loops

Participants: Philippe Clauss, Ervin Altıntaş, Matthieu Kuhn.

Loop collapsing is a well-known loop transformation which combines some loops that are perfectly nested into one single loop. It allows to take advantage of the whole amount of parallelism exhibited by the collapsed loops, and provides a perfect load balancing of iterations among the parallel threads.

However, in the current implementations of this loop optimization, as the ones of the OpenMP language, automatic loop collapsing is limited to loops with constant loop bounds that define rectangular iteration spaces, although load imbalance is a particularly crucial issue with non-rectangular loops. The OpenMP language addresses load balance mostly through dynamic runtime scheduling of the parallel threads. Nevertheless, this runtime schedule introduces some unavoidable execution-time overhead, while preventing to exploit the entire parallelism of all the parallel loops.

We propose a technique to automatically collapse any perfectly nested loops defining non-rectangular iteration spaces, whose bounds are linear functions of the loop iterators. Such spaces may be triangular, tetrahedral, trapezoidal, rhomboidal or parallelepiped. Our solution is based on original mathematical results addressing the inversion of a multi-variate polynomial that defines a ranking of the integer points contained in a convex polyhedron.

We show on a set of non-rectangular loop nests that our technique allows to generate parallel OpenMP codes that outperform the original parallel loop nests, parallelized either by using options “static” or “dynamic” of the OpenMP-schedule clause. A conference paper presenting these results, co-authored by Philippe Clauss, Ervin Altıntaş (Master student) and Matthieu Kuhn (Inria Bordeaux Sud-Ouest, team HIEPACS), is currently under review.

7.6. Efficient Data Structures for a PIC Code on SIMD Architectures

Participants: Yann Barsamian, Éric Violard.

In collaboration with Sever Adrian Hirstoaga (mathematician researcher, member of Inria team TONUS), we have developed an efficient particle simulation code. The domain of application is plasma physics, the Particle-In-Cell code simulating 2d2v Vlasov-Poisson equation on Cartesian grid with periodic boundary conditions for Landau damping test-case. We first analyzed different strategies for improving its performance on single core and then we used a standard approach for parallelizing it on many cores using hybrid OpenMP/MPI implementation. The optimization of the sequential code is mainly based on (i) a structure of arrays for the particles, (ii) an efficient data structure for the electric field and the charge density, and (iii) an appropriate code for automatic vectorization of the charge accumulation and of the positions' update. The parallelization of the loops over the particles is performed in a simple way (without domain decomposition) by means of both distributed and share memory paradigms. Satisfactory strong and weak scaling up to 8,192 cores on GENCI's supercomputer Curie are obtained, bounded as expected by the overhead of MPI communications. A conference paper presenting this work is currently under review.

7.7. Interactive Code Restructuring

Participants: Cédric Bastoul, Oleksandr Zinenko, Stéphane Huot.

This work falls within the exploration and development of semi-automatic programs optimization techniques. It consists in designing and evaluating new visualization and interaction techniques for code restructuring, by defining and taking advantage of visual representations of the underlying mathematical model. The main goal is to assist programmers during program optimization tasks in a safe and efficient way, even if they neither have expertise into code restructuring nor knowledge of the underlying theories. This project is an important step for the efficient use and wider acceptance of semi-automatic optimization techniques, which are still tedious to use and incomprehensible for most programmers. More generally, this research is also investigating new presentation and manipulation techniques for code, algorithms and programs, which could lead to many practical applications: collaboration, tracking and verification of changes, visual search in large amount of code, teaching, etc.

This is a new research direction opened by CAMUS which strengthens the team's static parallelization and optimization issue. It is a joint work with two Inria teams specialized in interaction: EX-SITU at Inria Saclay (contact: Oleksandr Zinenko) and MJOLNIR at Inria Lille (contact: Stéphane Huot).

In 2016, we released the first version of our interactive tool, *Clint*, that maps direct manipulation of the visual representation to polyhedral program transformations with real-time semantics preservation feedback (https://ozinenko.com/clint). Oleksandr Zinenko also defended his thesis on the research and development on interactive code restructuring.

7.8. Automatic Generation of Adaptive Simulation Codes

Participants: Cédric Bastoul, Maxime Schmitt.

Compiler automatic optimization and parallelization techniques are well suited for some classes of simulation or signal processing applications, however they usually don't take into account neither domain-specific knowledge nor the possibility to change or to remove some computations to achieve "good enough" results. Quite differently, production simulation and signal processing codes have adaptive capabilities: they are designed to compute precise results only where it matters if the complete problem is not tractable or if the computation time must be short. In this research, we design a new way to provide adaptive capabilities to compute-intensive codes automatically, inspired by Adaptive Mesh Refinement a classical numerical analysis technique to achieve precise computation only in pertinent areas. It relies on domain-specific knowledge provided through special pragmas by the programmer in the input code and on polyhedral compilation techniques, to continuously regenerate at runtime a code that performs heavy computations only where it matters at every moment. A case study on a fluid simulation application shows that our strategy enables dramatic computation savings in the optimized portion of the application while maintaining good precision, with a minimal effort from the programmer.

This research direction started in 2015 and complements our other efforts on dynamic optimization. In 2016, we started a collaboration on this topic with Inria Nancy - Grand Est team TONUS, specialized on applied mathematics (contact: Philippe Helluy), to bring models and techniques from this field to compilers. This collaboration received the support from the excellence laboratory (LabEx) IRMIA through the funding of the thesis of Maxime Schmitt on this topic. A first paper on this new research direction has just been accepted to IMPACT 2017.

7.9. Polyhedral Compiler White-Boxing

Participants: Cédric Bastoul, Lénaïc Bagnères, Oleksandr Zinenko, Stéphane Huot.

While compilers offer a fair trade-off between productivity and executable performance in single-threaded execution, their optimizations remain fragile when addressing compute-intensive code for parallel architectures with deep memory hierarchies. Moreover, these optimizations operate as black boxes, impenetrable for the user, leaving them with no alternative to time-consuming and error-prone manual optimization in cases where an imprecise cost model or a weak analysis resulted in a bad optimization decision. To address this issue, we researched and designed a technique allowing to automatically translate an arbitrary polyhedral optimization, used internally by loop-level optimization frameworks of several modern compilers, into a sequence of comprehensible syntactic transformations as long as this optimization focuses on scheduling loop iterations. With our approach, we open the black box of the polyhedral frameworks enabling users to examine, refine, replay and even design complex optimizations semi-automatically in partnership with the compiler.

This research started in 2014 and we published our first solution in 2016. It has been conducted as a joint work between teams in compiler technologies (CAMUS and Inria Saclay's POSTALE team) and teams in interaction (EX-SITU at Inria Saclay and MJOLNIR at Inria Lille). The first paper on this has been accepted and presented in one of the top conferences on optimization techniques: CGO 2016 [10]. It is also discussed in Lénaïc Bagnèse and Oleksandr Zinenko theses. In 2016 we finally release the tool implementing this research (https://periscop.github.io/chlore).

7.10. Mapping Deviation

Participant: Cédric Bastoul.

Compilers can provide a major help by automating the optimization and parallelization work. However they are very fragile black-boxes. A compiler may take a bad optimization decision because of imprecise heuristics or may turn off an optimization because of imprecise analyses, without providing much control or feedback to the end user. To address this issue, we researched and introduced mapping deviation, a new compiler technique that aims at providing a useful feedback on the semantics of a given program restructuring. Starting from a transformation intuition a user or a compiler wants to apply, our algorithm studies its correctness and can suggest changes or conditions to make it possible rather than being limited to the classical go/no-go answer. This algorithm builds on state-of-the-art polyhedral representation of programs and provides a high flexibility. We present two example applications of this technique: improving semi-automatic optimization tools for programmers and automatically designing runtime tests to check the correctness of a transformation for compilers.

This is a mid-term research on the mathematical ground of polyhedral compilation, started back to 2012. We found a solution and published it in 2016 in one of the main conferences in compilation: Compiler Construction [11]. We plan to release the tool that implements this research during the coming year.

7.11. Combining Locking and Data Management Interfaces

Participants: Jens Gustedt, Mariem Saied, Daniel Salas.

Handling data consistency in parallel and distributed settings is a challenging task, in particular if we want to allow for an easy to handle asynchronism between tasks. Our publication [1] shows how to produce deadlock-free iterative programs that implement strong overlapping between communication, IO and computation.

An implementation (ORWL) of our ideas of combining control and data management in C has been undertaken, see 6.8. In previous work it has demonstrated its efficiency for a large variety of platforms. In 2016, work on the ORWL model and library has gained vigor with the thesis of Mariem Saied (Inria & University of Strasbourg) and Daniel Salas (INSERM). We also now collaborate on that subject with the TADAAM project team from Inria Bordeaux, where a postdoc has been hired through Inria funding.

In 2016, a new domain specific language (DSL) has been completed that largely eases the implementation of applications with ORWL. In its first version it provides an interface for stencil codes, but extensions towards other types of applications are on their way. The approach allows to describe stencil codes quickly and efficient and leads to substantial speedups, see [14].

In addition, the framework has successfully been applied to encapsulate imaging applications that use certain pipeline patterns to describe dependencies between computational tasks, see [16]. Generally we have been able to use the knowledge of the communication structure of ORWL programs to map tasks to cores and thereby achieve interesting performance gains on multicore architectures, see [20].

In another work we have successfully applied ORWL to process Large Histopathology Images, see [15].

8. Bilateral Contracts and Grants with Industry

8.1. Caldera

Vincent Loechner and Cédric Bastoul are involved in a collaboration with the French company Caldera (http://www.caldera.com), specialized in software development for wide image processing. The goal of this collaboration is the development of parallel and scalable image processing pipeline for industrial printing. The project started in September 2016 and involves a contract established between the ICube laboratory and the Caldera company. This contract includes the funding of the industrial thesis (CIFRE) of Paul Godard (started in September 2016) on the topic of the collaboration, under the supervision of Vincent Loechner and Cédric Bastoul.

8.2. NANO 2017/PSAIC

The CAMUS team is taking part of the NANO 2017 national research program and its sub-project PSAIC (Performance and Size Auto-tuning thru Iterative Compilation) with the company STMicroelectronics, starting January 2015. Since the release of our automatic speculative parallelization framework Apollo, we have been working on an extension making Apollo usable as a advanced program profiling tool. We are also currently working in extending advanced loop optimization techniques to nonlinear loops using a linear virtual data layout remapping.

9. Partnerships and Cooperations

9.1. National Initiatives

Philippe Clauss, Alain Ketterlin, Cédric Bastoul and Vincent Loechner are involved in the Inria Project Lab entitled "Large scale multicore virtualization for performance scaling and portability" and regrouping several french researchers in compilers, parallel computing and program optimization [2]. The project started officially in January 2013. In this context and since January 2013, Philippe Clauss is co-advising with Erven Rohou of the Inria team PACAP, Nabil Hallou's PhD thesis focusing on dynamic optimization of binary code.

9.2. International Initiatives

9.2.1. Inria International Partners

9.2.1.1. Informal International Partners

The CAMUS team maintains regular contacts with the following entities:

- Reservoir Labs, New York, NY, USA
- University of Batna, Algeria
- Ohio State University, Colombus, USA
- Louisiana State University, Baton Rouge, USA
- Colorado State University, Fort Collins, USA
- Indian Institute of Science (IISc) Bangalore, India

[2]https://team.inria.fr/multicore

9.3. International Research Visitors

9.3.1. Visits of International Scientists

9.3.1.1. Researchers

Rachid Seghir

Date: April 30 - May 14

Institution: University of Batna, Algeria

10. Dissemination

10.1. Promoting Scientific Activities

10.1.1. Scientific Events Selection

10.1.1.1. Member of the Conference Program Committees

Cédric Bastoul has been part of the program committee of IMPACT 2016 (International Workshop on Polyhedral Compilation Techniques), held in conjunction with the international conference HiPEAC 2016.

Philippe Clauss and Cédric Bastoul have been part of the program committee of IMPACT 2017 (International Workshop on Polyhedral Compilation Techniques), held in conjunction with the international conference HiPEAC 2017.

Alain Ketterlin has been part of the program committee of CGO 2016 (International Symposium on Code Generation and Optimization, http://cgo.org/cgo2016).

Cédric Bastoul and Vincent Loechner have been part of the program committee of both HIP3ES 2016 and HIP3ES 2017 (International Workshop on High Performance Energy Efficient Embedded Systems), held in conjunction with the international conference HiPEAC 2016 (resp. HiPEAC 2017).

Cédric Bastoul has been part of the program committee of PARMA+DITAM 2016 and PARMA+DITAM 2017 (Workshop on Parallel Programming and Run-Time Management Techniques for Many-core Architectures + Workshop on Design Tools and Architerctures for Multicore Embedded Computing Platforms), held in conjunction with HiPEAC 2016 (resp. HiPEAC 2017).

Cédric Bastoul has been part of the program committee of the international conference on Compiler Construction 2017 (CC'2017).

10.1.1.2. Reviewer

Philippe Clauss has been reviewer for the following conferences and workshops: IMPACT 2017 (International Workshop on Polyhedral Compilation Techniques), CC 2017 (International Conference on Compiler Construction).

Cédric Bastoul has been reviewer for the following international conferences and workshops: CC 2017 (International Conference on Compiler Construction), PARMA 2016 and 2017 (International Workshop on Parallel Programming and Run-Time Management Techniques for Many-core Architectures), IMPACT 2016 and 2017 (International Workshop on Polyhedral Compilation Techniques), HIP3ES 2016 and 2017 (International Workshop on High Performance Energy Efficient Embedded Systems).

Vincent Loechner has been reviewer for CC 2017 (International Conference on Compiler Construction), HIP3ES 2016 and 2017 (International Workshop on High Performance Energy Efficient Embedded Systems).

10.1.2. Journal

10.1.2.1. Member of the Editorial Boards

Since October 2001, J. Gustedt is Editor-in-Chief of the journal *Discrete Mathematics and Theoretical Computer Science* (DMTCS).

10.1.2.2. Reviewer - Reviewing Activities

Philippe Clauss has been reviewer for the following journals: ACM TACO (Transactions on Architecture and Code Optimization), Parallel Computing.

Cédric Bastoul has been reviewer for the *ACM Transactions on Parallel Computing International Journal* (TOPC).

Jens Gustedt has been reviewer for Theory of Computing Systems.

Vincent Loechner has been reviewer for *Computer Communications* (Elsevier).

10.1.3. Invited Talks

Philippe Clauss has been invited to present the framework Apollo at the Parallel Programming Laboratory of the University of Darmstadt, Germany, October the 28th.

Philippe Clauss has presented the framework Apollo at the COSI research group of the Colorado State University, Fort Collins, USA, July the 1st.

10.1.4. Scientific Expertise

Cédric Bastoul as been an expert for the French research ministry and the French finance ministry for the research tax credit programme.

Jens Gustedt served as expert for project evaluation for the Belgian FNRS, and as evaluator of the FEMTO-ST Lab, Besançon, for the French HCERES.

10.1.5. Standardization

Since Nov. 2014, Jens Gustedt is a member of the ISO working group SC22-WG14 for the standardization of the C programming language. He participates actively in the defect report processing, the planning of future versions of the standard, and publishes an ongoing document to track inconsistencies and improvements of the C threads interface.

This work on the C programming language also gave rise to the proposal of a language extension, Modular C. It has been used for the implementation of an efficient toolbox for *higher order automatic differenciation*, `arbogast`, see [18] and [19], which has been presented at the quadrennial conference of the domain, AD2016.

10.2. Teaching - Supervision - Juries

10.2.1. Teaching

Licence : Philippe Clauss, Architecture des ordinateurs, 45h, Université de Strasbourg, France

Licence : Philippe Clauss, Systèmes d'exploitation, 40h, Université de Strasbourg, France

Master : Philippe Clauss, Compilation, 78h, Université de Strasbourg, France

Master : Philippe Clauss, Système et programmation temps-réel, 25h, Université de Strasbourg, France

Master : Philippe Clauss, Compilation avancée, 30h, Université de Strasbourg, France

Licence : Éric Violard, Programmation Fonctionnelle (licence informatique), 64h eq. TD, L2, Université de Strasbourg, France

Licence : Éric Violard, Architecture des Ordinateurs (licence informatique), 54h eq. TD, L2, Université de Strasbourg, France

Licence : Éric Violard, Logique et Programmation Logique (licence informatique), 34h eq. TD, L2, Université de Strasbourg, France

Licence : Éric Violard, Algorithmique et Structures de Données (licence mathématique), 39h eq. TD, L3, Université de Strasbourg, France

Licence : Éric Violard, Modèles de Calcul (licence informatique), 29h eq. TD, L1, Université de Strasbourg, France

Licence : Vincent Loechner, Systèmes d'exploitation, 51h, L2, Université de Strasbourg, France

Master : Vincent Loechner, parallélisme, 14h, M1, Université de Strasbourg, France

Master : Vincent Loechner, calcul parallèle, 32h, M1, Université de Strasbourg, France

Master : Vincent Loechner, langages interprétés, 37h, M1, Université de Strasbourg, France

Master : Vincent Loechner, OS embarqués, 31h, M2, Université de Strasbourg, France

Telecom Physique Strasbourg : Vincent Loechner, calcul parallèle, 20h, M2, Université de Strasbourg, France

Licence : Alain Ketterlin, Systèmes d'exploitation, 20h, Université de Strasbourg, France

Licence : Alain Ketterlin, Systèmes Concurrents, 24h, Université de Strasbourg, France

Licence : Alain Ketterlin, Réseaux et protocoles, 42h, Université de Strasbourg, France

Master : Alain Ketterlin, Compilation, 26h, Université de Strasbourg, France

Licence : Cédric Bastoul, Architecture, 68h, L1 (IUT), Université de Strasbourg, France

Licence : Cédric Bastoul, Operating Systems, 16h, L2, Université de Strasbourg, France

Licence : Cédric Bastoul, Concurrent Systems, 19h, L3, Université de Strasbourg, France

Master : Cédric Bastoul, Compiler Design, 48h, M1, Université de Strasbourg, France

Master : Cédric Bastoul, Parallelism, 16h, M1, Université de Strasbourg, France

Master : Cédric Bastoul, Introduction to Research, 7h, L3+M1, Université de Strasbourg, France

2nd year engineering school: Jens Gustedt, programmation avancée, 20h, ENSIIE Strasbourg, France

Licence : Jens Gustedt, systèmes concurrents, 20h, Université de Strasbourg, France

10.2.2. Supervision

PhD: Tomasz Buchert, Madynes team, *Orchestration of experiments on distributed systems*, since Oct 2011, defended on Jan 6 2016, Jens Gustedt & Lucas Nussbaum.

PhD: Juan Manuel Martinez Caamaño, *Fast and Flexible Compilation Techniques for Effective Speculative Polyhedral Parallelization*, September 29th 2016, Philippe Clauss and Philippe Helluy (IRMA lab., University of Strasbourg)

PhD: Michel Massaro, *Méthodes numériques pour les plasmas sur architectures multicœurs*, December 16th 2016, Philippe Helluy and Vincent Loechner

PhD: Lénaïc Bagnères, Adaptation automatique et semi-automatique des optimisations de programmes, September 30th, Christine Eisenbeis and Cédric Bastoul

PhD: Olexander Zinenko, Interactive Program Restructuring, November 25th 2016, Stéphane Huot and Cédric Bastoul

PhD in progress: Yann Barsamian, *Optimization and parallelization of particle and semi-Lagrangian methods for multi species plasma simulations*, since Oct 2014, Éric Violard

PhD in progress: Mariem Saied, *Ordered Read-Write Locks for Multicores and Accelerators*, since Nov 2013, Jens Gustedt & Gilles Muller.

PhD in progress: Daniel Salas, *Integration of the ORWL model into parallel applications for medical research*, since Mar 2015, Jens Gustedt & Isabelle Perseil.

PhD in progress: Nabil Hallou, *Dynamic binary optimizations*, since January 2013, Erven Rohou (PACAP team) and Philippe Clauss

PhD in progress: Harenome Ranaivoarivony-Razanajato, *Hierarchical Optimization and Parallelization*, September 2016, Vincent Loechner and Philippe Clauss

PhD in progress: Maxime Schmitt, *Automatic Generation of Adaptive Codes*, September 2016, Cédric Bastoul and Philippe Helluy

PhD in progress: Paul Godard, *Parallelization and Scalability of an Image Processing Pipeline for Professional Printing*, September 2016, Vincent Loechner and Cédric Bastoul

10.2.3. Juries

Philippe Clauss participated to the following PhD committees in 2016:

Date	**Candidate**	**Place**	**Role**
Jun. 30	Guillaume Iooss	Colorado State Univ., USA	Reviewer
Oct. 28	Zhen Li	Univ. Darmstadt, Germany	Reviewer
Sept. 29	Juan Manuel Martinez Caamaño	Univ. Strasbourg	Advisor
Dec. 14	Julien Pagès	Univ. Montpellier	Reviewer

Cédric Bastoul participated to the following PhD committees in 2016:

Date	**Candidate**	**Place**	**Role**
June 22	Abdul Memon	Paris-Saclay University	Reviewer
September 30	Lénaïc Bagnères	Paris-Saclay University	Advisor
November 18	Albert Saa	Universitat Autònoma de Barcelona	Reviewer
November 25	Oleksandr Zinenko	Paris-Saclay University	Advisor
November 30	Pierre Guillou	Paris Sciences et Lettres Research University	Reviewer

Vincent Loechner participated to the following PhD committees in 2016:

Date	**Candidate**	**Place**	**Role**
May 10	Arjun Suresh	Univ. Rennes 1	Examiner
December 16	Michel Massaro	Univ. Strasbourg	Co-advisor

Vincent Loechner was the president of the recruiting jury (*comité de sélection*) for an assistant professor position at the Department of Mathematics and Computer Science of the University of Strasbourg, during Spring 2016.

10.3. Popularization

Jens Gustedt is regularly blogging about efficient programming, in particular about the C programming language. He also is an active member of the stackoverflow community a technical Q&A site for programming and related subjects. A first complete online version of his book *Modern C*, to appear in 2017, has been accessed more than 10000 times on a single day.

11. Bibliography

Major publications by the team in recent years

[1] P.-N. Clauss, J. Gustedt. *Iterative Computations with Ordered Read-Write Locks*, in "Journal of Parallel and Distributed Computing", 2010, vol. 70, n^{o} 5, pp. 496-504 [*DOI :* 10.1016/j.jpdc.2009.09.002], https://hal.inria.fr/inria-00330024

[2] J. Gustedt. *Futex based locks for C11's generic atomics*, Inria Nancy, December 2015, n^{o} RR-8818, https://hal.inria.fr/hal-01236734

[3] A. JIMBOREAN, P. CLAUSS, J.-F. DOLLINGER, V. LOECHNER, M. JUAN MANUEL. *Dynamic and Speculative Polyhedral Parallelization Using Compiler-Generated Skeletons*, in "International Journal of Parallel Programming", August 2014, vol. 42, n^o 4, pp. 529-545, https://hal.inria.fr/hal-01003744

[4] A. KETTERLIN, P. CLAUSS. *Prediction and trace compression of data access addresses through nested loop recognition*, in "6th annual IEEE/ACM international symposium on Code generation and optimization", Boston, USA, ACM, April 2008, pp. 94-103, http://dx.doi.org/10.1145/1356058.1356071

[5] A. KETTERLIN, P. CLAUSS. *Profiling Data-Dependence to Assist Parallelization: Framework, Scope, and Optimization*, in "MICRO-45 – Proceedings of the 2012 IEEE/ACM 45th International Symposium on Microarchitecture", Vancouver, Canada, December 2012

[6] B. PRADELLE, A. KETTERLIN, P. CLAUSS. *Polyhedral parallelization of binary code*, in "ACM Transactions on Architecture and Code Optimization", January 2012, vol. 8, n^o 4, pp. 39:1–39:21 [*DOI :* 10.1145/2086696.2086718], http://hal.inria.fr/hal-00664370

[7] R. SEGHIR, V. LOECHNER, B. MEISTER. *Integer Affine Transformations of Parametric Z-polytopes and Applications to Loop Nest Optimization*, in "ACM Transactions on Architecture and Code Optimization", June 2012, vol. 9, n^o 2, pp. 8.1-8.27 [*DOI :* 10.1145/2207222.2207224], http://hal.inria.fr/inria-00582388

Publications of the year

Doctoral Dissertations and Habilitation Theses

[8] J. M. MARTINEZ CAAMAÑO. *Fast and Flexible Compilation Techniques for Effective Speculative Polyhedral Parallelization*, Université de Strasbourg, September 2016, https://hal.inria.fr/tel-01377758

Articles in International Peer-Reviewed Journals

[9] A. SUKUMARAN-RAJAM, P. CLAUSS. *The Polyhedral Model of Nonlinear Loops*, in "ACM Transactions on Architecture and Code Optimization", January 2016, vol. 12, n^o 4 [*DOI :* 10.1145/2838734], https://hal.inria.fr/hal-01244464

International Conferences with Proceedings

[10] L. BAGNÈRES, O. ZINENKO, S. HUOT, C. BASTOUL. *Opening Polyhedral Compiler's Black Box*, in "CGO 2016 - 14th Annual IEEE/ACM International Symposium on Code Generation and Optimization", Barcelona, Spain, March 2016, https://hal.inria.fr/hal-01253322

[11] C. BASTOUL. *Mapping Deviation: A Technique to Adapt or to Guard Loop Transformation Intuitions for Legality*, in "CC'2016 25th International Conference on Compiler Construction", Barcelone, Spain, March 2016, https://hal.inria.fr/hal-01271998

[12] J. GUSTEDT. *Futex based locks for C11's generic atomics (extended abstract)*, in "The 31st Annual ACM Symposium on Applied Computing", Pisa, Italy, April 2016 [*DOI :* 10.1145/2851613.2851956], https://hal.inria.fr/hal-01304108

[13] *Best Paper*
J. M. MARTINEZ CAAMAÑO, W. WOLFF, P. CLAUSS. *Code Bones: Fast and Flexible Code Generation for Dynamic and Speculative Polyhedral Optimization*, in "Euro-Par 2016", Grenoble, France, SPRINGER-VERLAG (editor), Proceedings of the 22nd International Conference Euro-Par 2016: Parallel Processing, August 2016, vol. 9833, 12 p. [*DOI :* 10.1007/978-3-319-43659-3_17], https://hal.inria.fr/hal-01377656.

[14] M. SAIED, J. GUSTEDT, G. MULLER. *Automatic Code Generation for Iterative Multi-dimensional Stencil Computations*, in "High Performance Computing, Data, and Analitics", Hydarabat, India, A. BENOÎT (editor), IEEE, December 2016, https://hal.inria.fr/hal-01337093

[15] D. SALAS, J. GUSTEDT, D. RACOCEANU, I. PERSEIL. *Resource-Centered Distributed Processing of Large Histopathology Images*, in "19th IEEE International Conference on Computational Science and Engineering", Paris, France, August 2016, https://hal.inria.fr/hal-01325648

National Conferences with Proceedings

[16] F. MANSOURI, J. GUSTEDT. *Le modèle de programmation ORWL pour la parallélisation d'une application de suivi vidéo HD sur architecture multi-coeurs*, in "Conférence d'informatique en Parallélisme, Architecture et Système (COMPAS)", Lorient, France, July 2016, accepted for publication in Compas'16, https://hal.inria.fr/hal-01325850

Scientific Books (or Scientific Book chapters)

[17] S. LEILA HERNANE, J. GUSTEDT. *Transparent distributed data management in large scale distributed systems*, in "Pervasive Computing", Academic Press, 2016, pp. 153-194, https://hal.inria.fr/hal-01308989

Research Reports

[18] I. CHARPENTIER, J.-P. FRIEDELMEYER, J. GUSTEDT. *Arbogast – Origine d'un outil de dérivation automatique*, Inria, May 2016, n^{o} RR-8911, https://hal.inria.fr/hal-01313355

[19] I. CHARPENTIER, J. GUSTEDT. *Arbogast Higher order AD for special functions with Modular C*, Inria Nancy - Grand Est (Villers-lès-Nancy, France), April 2016, n^{o} 8907, https://hal.inria.fr/hal-01307750

[20] J. GUSTEDT, E. JEANNOT, F. MANSOURI. *Fully-abstracted affinity optimization for task-based models*, Inria Nancy, December 2016, n^{o} RR-8993, https://hal.inria.fr/hal-01409101

Other Publications

[21] J. GUSTEDT, E. JEANNOT, F. MANSOURI. *Optimizing Locality by Topology-aware Placement for a Task Based Programming Model*, September 2016, pp. 164 - 165, IEEE Cluster 2016 Conference, Poster [*DOI :* 10.1109/CLUSTER.2016.87], https://hal.archives-ouvertes.fr/hal-01416284

[22] O. ZINENKO, S. HUOT, C. BASTOUL. *Interactive Program Restructuring*, November 2016, working paper or preprint, https://hal.inria.fr/hal-01406294

References in notes

[23] C. BASTOUL. *Code Generation in the Polyhedral Model Is Easier Than You Think*, in "PACT'13 IEEE International Conference on Parallel Architecture and Compilation Techniques", Juan-les-Pins, France, 2004, pp. 7–16, https://hal.archives-ouvertes.fr/ccsd-00017260

[24] M. HALL, D. PADUA, K. PINGALI. *Compiler research: the next 50 years*, in "Commun. ACM", 2009, vol. 52, n^{o} 2, pp. 60–67, http://doi.acm.org/10.1145/1461928.1461946

[25] A. HOBOR, A. W. APPEL, F. Z. NARDELLI. *Oracle Semantics for Concurrent Separation Logic*, in "ESOP", 2008, pp. 353-367