Activity Report 2015

Team CAMUS

Compilation pour les Architectures MUlti-coeurS

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:

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:

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

1. Members

Research Scientist

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, Associate Professor]
Vincent Loechner [Univ. Strasbourg, Associate Professor]
Nicolas Magaud [Univ. Strasbourg I, Associate Professor]
Julien Narboux [Univ. Strasbourg I, Associate Professor]
Eric Violard [Univ. Strasbourg, Associate Professor, HdR]

Engineer

Artiom Baloian [Inria, from Oct 2015]

PhD Students

Yann Barsamian [Univ. Strasbourg]
Luis Esteban Campostrini [Inria]
Jean-Francois Dollinger [Univ. Strasbourg, until Sep 2015]
Imen Fassi [Inria, until Nov 2015]
Juan Manuel Martinez Caamano [Univ. Strasbourg]
Mariem Saied [Inria]
Daniel Salas [INSERM, from Mar 2015]
Aravind Sukumaran Rajam [Inria, until Nov 2015]

Administrative Assistant

Veronique Constant [Inria]

2. Overall Objectives

2.1. Overall Objectives

The CAMUS team is focusing on developping, 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 [31]. 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.

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.

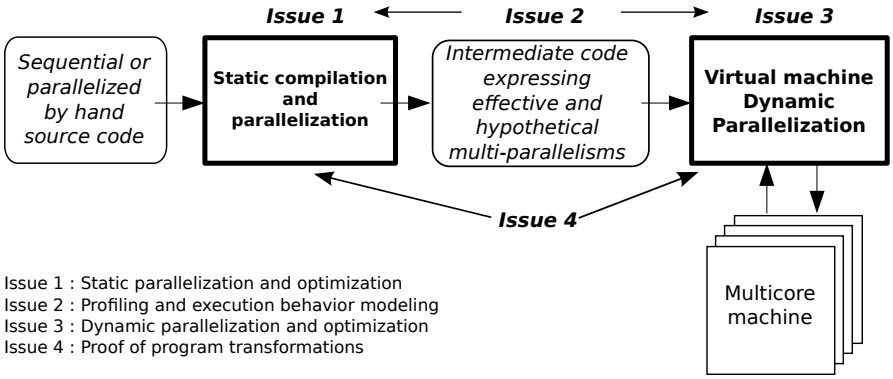

Figure 1. Automatic parallelizing steps for multicore architectures

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, Jean-François Dollinger.

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 [28]. 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, Aravind Sukumaran-Rajam, Luis Esteban Campostrini.

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: Aravind Sukumaran-Rajam, Juan Manuel Martinez Caamaño, Luis Esteban Campostrini, Artiom Baloian, Jean-François Dollinger, 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, Julien Narboux, Nicolas Magaud.

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 [32]. 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

Aravind Sukumaran-Rajam has shown in his PhD work [13] that the polyhedral model, usually exclusively dedicated to advanced static analysis and optimization of linear loops, can also be applied to nonlinear loops. This noteworthy extension of the scope of polyhedral techniques has been made possible thanks to the speculative and dynamic parallelization strategy implemented in the Apollo framework. Significant parallel speed-ups can now be obtained automatically for loops and loop nest that could not be handled before by compilers. Aravind Sukumaran-Rajam and Philippe Clauss have published a paper on this topic in the ACM journal Transactions on Architecture and Code Optimization in 2015 [14].

6. New Software and Platforms

6.1. APOLLO

Automatic speculative POLyhedral Loop Optimizer
FUNCTIONAL DESCRIPTION

We are developing a framework called APOLLO (Automatic speculative POLyhedral Loop Optimizer), 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 has been extended in 2015 to apply on-the-fly any kind of polyhedral transformations, including tiling, and to handle nonlinear loops as while-loops referencing memory through pointers and indirections.

- Participants: Aravind Sukumaran-Rajam, Juan Manuel Martinez Caamaño, Luis Esteban Campostrini, Artiom Baloian, Willy Wolff and Philippe Clauss
- Contact: Juan Manuel Martinez Caamaño

6.2. CLooG

Code Generator in the Polyhedral Model
FUNCTIONAL DESCRIPTION

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 and Imèn Fassi
- 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: Imen Fassi, Philippe Clauss and Cédric Bastoul
- Contact: Philippe Clauss

6.6. XFOR-Wizard

XFOR-Wizard
FUNCTIONAL DESCRIPTION

Xfor-Wizard is a programming environment for XFOR programs, assisting users in writing XFOR codes and applying optimizing transformations. Automatic dependence analysis and comparisons against a referential code (XFOR-loops or classic for-loops) are achieved to order to help the user in ensuring semantic correctness of the written code.

- Participants: Imen Fassi, Philippe Clauss and Cédric Bastoul
- Contact: Philippe Clauss

6.7. XFORGEN

XFOR code generator
FUNCTIONAL DESCRIPTION

XFORGEN is a tool to automatically generate an XFOR code that is equivalent to for-loops that have been automatically transformed using a static polyhedral compiler. The generated XFOR code exhibits the parameters of the transformations that have been applied and thus can be modified for further optimizations.

- Participants: Imen Fassi, Philippe Clauss and Cédric Bastoul
- Contact: Philippe Clauss

6.8. 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/

6.9. ORWL and P99

ORWL is a reference implementation of the Ordered Read-Write Lock tools as described in [5]. 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/.
Software classification: A-3-up, SO-4, SM-3, EM-3, SDL (P99: 4, ORWL: 2-up), DA-4, CD-4, MS-3, TPM-4

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

6.10. stdatomic and musl

We implement the libary side of the C11 atomic interface. It needs compiler support for the individual atomic operations and provides library supports 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 [24]. A short version of it has been accepted for 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/

6.11. 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/

7. New Results

7.1. Formal Proofs for an Ordering Relation in Explicitly Parallel Programs

Participants: Alain Ketterlin, Éric Violard.

This project is a collaborative work with the COMPSYS Inria Team, in Lyon. Participants are: Paul Feautrier, Tomofumi Yuki.

The growing need to make use of available parallelism has led to new explicitly parallel language constructs. These constructs are usually grouped under the term *Task Parallelism*, because they aim to go beyond "simple" *Data Parallelism* (i.e., loop and array-based parallelism). Prominent examples of languages integrating task parallelism are X10 (http://x10-lang.org) and variants, Cilk (http://supertech.csail.mit.edu/cilk/), and recent versions of OpenMP (http://www.openmp.org). Most of the work on such languages has focused on efficient run-time support for *tasks*, in contrast with *threads*, i.e., for programs generating potentially large numbers of distinct tasks with explicit (but arbitrary) ordering between the tasks. However, little attention has been given to the static analysis and optimization of explicitly parallel programs, probably because their properties are much harder to formalize, compared to their sequential counterpart. Starting with the work of our colleagues Paul Feautrier and Tomofumi Yuki, from the Compsys team in Lyon, we have advanced the formalization and formally proved several properties of some fundamental building blocks for the analysis of certain classes of explicitly parallel programs.

Task parallelism is usually based on a few syntactic constructs to represent tasks and their synchronization. We use X10's terminology (and syntax, with simplifications), but the corresponding constructs of other languages is usually obvious. Across all languages one finds a construct to start (or *spawn*) an asynchronous task, named `async` in X10, and a "container" construct, named `finish` in X10, whose role is to wait for the completion of all task spawned during the execution of its body. Given that these constructs allow the parallel execution of pieces of the program, a first question arises: is there a static (i.e., compile-time) way to decide whether two given statements are ordered, i.e., that the first necessarily executes before the other. Feautrier and Yuki (with colleagues) have defined such a criterion for programs made of `async` and `finish` [33], along with arbitrary statements and for-loops, defining the so-called *polyhedral fragment* of X10. The resulting (partial) relation, called *happens-before*, opens the door to various static analyses, like data-dependence analysis, which are at the heart of a range of optimization techniques. Here is a quick example:

```
finish
for i in ...
    async
    for j in ...
        S(i,j)
```

$\texttt{S}(i,j)$ happens before $\texttt{S}(i',j')$ iff $i = i' \wedge j < j'$

The resulting condition, $i = i' \wedge j < j'$, defines exactly the situation in which two statement executions are ordered, and can be seen as an appropriate extension of the lexicographic order to explicitly parallel programs.

Our work on this basis has been to take the formal definition of happens-before (HB), and implement it in Coq (https://coq.inria.fr). The goal was first to prove various properties of the relation, like transitivity, and second to provide a formal proof of both correctness and completeness of HB itself. The first part has been fairly immediate, due to the high representative power of Coq. The second part took more time, and involved several new contributions. The major part of the work went into defining a formal semantics for the fragment of X10 needed by the definition of HB. Given the semantics, it was possible to obtain the relation between a program and its trace(s), and then to prove that HB is correct (i.e., if HB states that one statement executes before another, then these statements appear in order in all possible traces of the program), and that HB is complete (i.e., that statements that are always ordered in traces are actually recognized as such by HB). The complete proof scripts are available on the Inria forge (gforge.inria.fr), under the `x10-coq` project.

Further work has also started on extending *happens-before* to X10 programs using synchronization primitives called *clocks*, which are basically *barriers*, where distinct tasks can wait for each other. Since an unrestricted use of synchronization barriers can lead to deadlocks, X10 introduces "implicit clocks", which are introduced (and scoped) by a `finish` construct, on which a task can "register", and whose scoping rules ensure that any program point can only use the single "nearest" clock. These restrictions offer termination guarantees, which in turn enables a sound *happens-before* relation between statement instances. The "clock-less" HB relation can then be modified to take into account the additional ordering imposed by clocks. We have started work to update the semantics to the case of implicit clocks, and to formalize this extension in Coq.

7.2. Validity Conditions for Transformations of Non-Affine Programs

Participants: Alain Ketterlin, Philippe Clauss.

This project is a collaborative work with the CORSE Inria Team, in Grenoble. Participant is: Fabrice Rastello.

Representing loop nests with the help of the polyhedral model has been a powerful and fruitful strategy to enable automatic optimization and parallelization. However, this model places strong requirements on the input program, and in many cases these requirements are hard to meet. Because they are based on linear programming, polyhedral techniques require every constraint to be affine in loop counters and parameters. While this is easily verified for loop bounds in a large majority of programs, the same constraint imposed to memory access functions is often too strong. There are several reasons for this. First, programmers often linearize multi-dimensional arrays, turning straightforward accesses like `t[i][j]` into `t1[i*n+j]`, with the unfortunate effect of placing their program outside the scope of the polyhedral model. Second, optimization often happens late in the compilation process (or even during just-in-time compilation at run-time), where multi-dimensional array accesses have been transformed by the compiler itself, for the needs of its earlier passes. Third, complex data storage strategies for certain classes of arrays, e.g., band or triangular matrices, may introduce non-linear access functions, and this non-linearity must be taken into account, e.g., for locality optimization. And fourth, some access functions are almost completely unspecified, like in the case of indirect accesses (`t[s[i]]`) or abstract mappings (`t[f(i)]`).

Our goal is to extend polyhedral analysis techniques to cover at least some of these cases, and see how far we can push the limits of the fundamental algorithms beyond pure linearity. We have started by considering the case of multi-dimensional array linearization, where the code doesn't provide access functions for all (original) dimensions, but rather a single access function, which is linear in loop counters but contains parametric coefficients. Here is an example illustrating our initial target, which is taken from the `gemver` program in the `polybench` suite:

```
  for (i = 0; i < n; i++)
    for (j = 0; j < n; j++)
      // Was: A[i][j] = A[i][j] + u1[i] * v1[j] + ...;
S1:   *(n*i+A+j) = *(n*i+A+j) + *(u1+i) * *(v1+j) + ...;
  for (i = 0; i < n; i++)
    for (j = 0; j < n; j++)
      // Was: x[i] = x[i] + beta * A[j][i] * y[j];
S2:   *(x+i) = *(x+i) + beta * *(n*j+A+i) * *(y+j);
  // ...
```

The original form of the statements appear in comments, but what finally reaches the compiler is much more convoluted: basically, every array access appears as a pointer access whose effective address is a polynomial function mixing counters (`i`, `j`), array base addresses (`A`, `u1`, `v1`, `x`, `y`), and size parameters (`n`). In some other cases, the arrays have been "locally" linearized, i.e., the code still displays different arrays, but their inner dimensions have been linearized. In our example, statement `S1` would appear as:

```
      // Was: A[i][j] = A[i][j] + u1[i] * v1[j] + ...;
S1:   A[n*i+j] = A[n*i+j] + u1[i] + v1[j] + ...;
```

This is an important special case in practice, and its particular structure helps a lot, for example, when data dependence analysis is needed.

Extending current polyhedral techniques to deal with non-affine accesses is a formidable endeavor, requiring the adaptation of the many algorithms developed over decades for analysis, scheduling, and code generation. Rather, we have started by studying a specific task, with immediate practical impact: given a non-affine loop nest *and* a specific desired transformation, what are the conditions under which this condition is valid? It is not unreasonable to expect the transformation to be provided by other means than pure analysis, for instance to be suggested by profiling data. In this case, the problem we are left with is the one of testing whether the given transformation is valid. This in turn requires testing the emptyness of a "problematic system". For any given loop nest, this can be written as:

$$
\begin{array}{lll}
\bigvee_{(A,A')} & \exists (v, v') \text{s.t.} & \\
 & v \in \mathcal{D}_A \wedge v' \in \mathcal{D}_{A'} & \text{(domain)} \\
\wedge & v \prec_{lex} v' & \text{(originalschedule)} \\
\wedge & A(v) = A'(v) & \text{(sameaccesslocation)} \\
\wedge & T_A(v) \neg \prec_{lex} T_{A'}(v') & \text{(transformedschedule)}
\end{array}
$$

where A and A' range over pairs of potentially conflicting accesses, v and v' are iteration vectors, $\mathcal{D}_A$ and $\mathcal{D}_{A'}$ are iteration domains, $A(v)$ and $A'(v')$ are access functions, and T_A and $T'_{A'}$ are schedules. The condition under which the transformation is valid is the projection of this set on parameter dimensions, i.e., the elimination of all variables representing counters. The difficulty of this comes from the non-affine condition expressing the equality of access functions.

Building on previous work, we have devised a projection procedure that eliminates all counters and leaves a (usually complex) condition on parameters. We have also developed several simplification strategies, applied during elimination and also on the final result, that overall produces a test deciding whether the targeted transformation can be applied. For instance, on the fully linearized version of the previous examples, when deciding whether the following transformation is legal:

$$T_{\texttt{S1}}(0,i,j) = (0,i,j) \qquad T_{\texttt{S2}}(1,i,j) = (0,j,i)$$

i.e., interchanging the second loop (around S2) and then applying fusion on both depth-2 loops, our elimination and simplification procedure produces the following run-time test:

```
if ( ((y+n >= x+2) && (x+n >= y+2))
|| ((n >= 2) && (n*n+A >= x+1) && (x >= A+1))
|| ((n >= 2) && (u1+n >= x+1) && (x+n >= u1+2))
|| ((n >= 2) && (n+v1 >= x+1) && (x+n >= v1+1))
|| ((n*n+A >= y+1) && (y >= A+1) && (n >= 2)) || ...)
  // Transformation invalid: run the original version...
else
  // Transformation valid: run the transformed version...
```

The reader may want to verify that this test actually corresponds to verifying that "arrays" do not overlap, but only as far as the given transformation requires it.

A systematic evaluation of our procedure on a benchmark suite has shown that the resulting tests are both accurate and incur very little run-time overhead. The overall mechanism compares favorably with alternative techniques aiming at dealing with non-affine access functions, which consist in statically reconstructing array dimensions [30]. This part of our work is ready for publication. However, to be completely competitive with alternative approaches, we need to find ways to complete the polyhedral compilation chain, with a prior effective scheduling algorithm and a posterior code generation algorithm.

7.3. Automatic Parallelization of Nonlinear Loops

Participants: Aravind Sukumaran-Rajam, Philippe Clauss.

Runtime code optimization and speculative execution are becoming increasingly prominent to leverage performance in the current multi- and many-core era. However, a wider and more efficient use of such techniques is mainly hampered by the prohibitive time overhead induced by centralized data race detection, dynamic code behavior modeling, and code generation. Most of the existing Thread Level Speculation (TLS) systems rely on naively slicing the target loops into chunks and trying to execute the chunks in parallel with the help of a centralized performance-penalizing verification module that takes care of data races. Due to the lack of a data dependence model, these speculative systems are not capable of doing advanced transformations, and, more importantly, the chances of rollback are high. The polyhedral model is a well- known mathematical model to analyze and optimize loop nests. The current state-of-art tools limit the application of the polyhedral model to static control codes. Thus, none of these tools can generally handle codes with while loops, indirect memory accesses, or pointers. Apollo (Automatic POLyhedral Loop Optimizer) is a framework that goes one step beyond and applies the polyhedral model dynamically by using TLS. Apollo can predict, at runtime, whether the codes are behaving linearly or not, and it applies polyhedral transformations on-the-fly.

Apollo has been extended to handle codes whose memory accesses and loop bounds are not necessarily linear [23], [14]. The proposed extension consists of modeling memory addresses that are accessed either as "tubes" obtained through linear regression, or as ranges. More generally, this approach expands the applicability of the polyhedral model at runtime to a wider class of codes. Plugging together both linear and nonlinear accesses to the dependence prediction model enables the application of polyhedral loop optimizing transformations even for nonlinear code kernels while also allowing a low-cost speculation verification.

This work takes part of Aravind Sukumaran-Rajam's PhD thesis that has been defended November the 5th, 2015 [13].

7.4. Dynamic Code Generation for Speculative Polyhedral Optimization

Participants: Juan Manuel Martinez Caamano, Philippe Clauss.

We have developped a new runtime code generation technique for speculative loop optimization and parallelization, that allows to generate on-the-fly codes resulting from any polyhedral optimizing transformation of loop nests, such as tiling, skewing, loop fission, loop fusion or loop interchange, without introducing a penalizing time overhead. The proposed 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 strategy has been implemented in the speculative loop parallelizing framework Apollo.

7.5. The XFOR Programming Structure

Participants: Imen Fassi, Philippe Clauss, Cédric Bastoul.

We have proposed a new programming control structure called "xfor" or "multifor", providing users a way to schedule explicitly the statements of a loop nest, and take advantage of optimization and parallelization opportunities that are not easily attainable using the standard programming structures, or using automatic optimizing compilers [19]. This is the PhD work of Imen Fassi, who started her work in 2013 and who defended her thesis November the 27th, 2015 [12].

It has been shown that xfor programs often reach better performance than programs optimized by fully automatic polyhedral compilers like Pluto [29]. It has also been shown that different versions of codes may perform very differently, although their memory behaviors are very similar. By analyzing further the origins of such performance differences, we noticed five important gaps in the currently adopted and well-established code optimization strategies [18], [19]: insufficient data locality optimization, excess of conditional branches in the generated code, too verbose code with too many machine instructions, data locality optimization resulting in processor stalls, and finally missed vectorization opportunities.

To ease and extend the usage of the XFOR structure, we have developed:

- Xfor-Wizard, which is a programming environment for XFOR programs, assisting users in writing XFOR codes and applying optimizing transformations. Automatic dependence analysis and comparisons against a referential code (XFOR-loops or classic for-loops) are achieved to order to help the user in ensuring semantic correctness of the written code.
- XFORGEN, which is a tool to automatically generate an XFOR code that is equivalent to for-loops that have been automatically transformed using a static polyhedral compiler. The generated XFOR code exhibits the parameters of the transformations that have been applied and thus can be modified for further optimizations.

7.6. Dynamic Optimization of Binary Code

Participants: Philippe Clauss, Alain Ketterlin.

This project is a collaborative work with the ALF Inria Team, in Rennes. Participants are: Erven Rohou and Nabil Hallou.

Automatic code optimizations have traditionally focused on source-to-source transformation tools and compiler IR-level techniques. Sophisticated techniques have been developed for some classes of programs, and rapid progress is made in the field. However, there is a persistent hiatus between software vendors having to distribute generic programs, and end-users running them on a variety of hardware platforms, with varying levels of optimization opportunities. The next decade may well see an increasing variety of hardware, as it has already started to appear particularly in the embedded systems market. At the same time, one can expect more and more architecture-specific automatic optimization techniques.

Unfortunately, many "old" executables are still being used although they have been originally compiled for now outdated processor chips. Several reasons contribute to this situation:

- commercial software is typically sold without source code (hence no possibility to recompile) and targets slightly old hardware to guarantee a large base of compatible machines;
- though not commercial, the same applies to most Linux distributions [1] – for example Fedora 16 (released Nov 2011) is supported by Pentium III (May 1999) [2];
- with the widespread cloud computing and compute servers, users have no guarantee as to where their code runs, forcing them to target the oldest compatible hardware in the pool of available machines.

All this argues in favor of binary-to-binary optimizing transformations. Such transformations can be applied either statically, i.e., before executing the target code, or dynamically, i.e., while the target code is running.

Dynamic optimization is mostly addressing adaptability to various architectures and execution environments. If practical, dynamic optimization should be preferred because it eliminates several difficulties associated with static optimization. For instance, when deploying an application in the cloud, the executable file may be handled by various processor architectures providing varying levels of optimization opportunities. Providing numerous different adapted binary versions cannot be a general solution. Another point is related to interactions between applications running simultaneously on shared hardware, where adaptation may be required to adjust to the varying availability of the resources. Finally, most code optimizations have a basic cost that has to be recouped by the gain they provide. Depending on the input data processed by the target code, an optimizing transformation may or may not be profitable.

We distinguish two classes of binary transformations:

1. code transformations that can be handled directly by analyzing and modifying the original binary code. We call such transformations *low-level binary transformations*;
2. code transformations that require a higher level of abstraction of the code in order to generate a very different, but semantically equivalent, optimized code. We call such transformations *high-level binary transformations*.

While we target both classes of transformations, the first was addressed by focusing on SSE to AVX transformations of vectorized codes [20].

In this work, we focus on SIMD ISA extensions, and in particular on the x86 SSE and AVX capabilities. Compared to SSE, AVX provides wider registers, new instructions, and new addressing formats. AVX has been first supported in 2011 by the Intel Sandy Bridge and AMD Bulldozer architectures. However, most existing applications take advantage only of SSE and miss significant opportunities. We show that it is possible to automatically convert SSE to AVX whenever profitable. The key characteristics of our approach are:

- we apply the transformation at run-time, i.e. when the hardware is known;
- we only transform hot loops (detected through very lightweight profiling), thus minimizing the overhead;
- we do *not* implement a vectorization algorithm in a dynamic optimizer, instead we recognize already statically vectorized loops, and convert them to a more powerful ISA at low cost.

For high-level binary transformations, we also focus on hot loops and loop nests appearing in executable codes. There is an important literature addressing automatic loop optimization and parallelization techniques. Such optimizations include combinations of loop interchange, loop fusion and fission, loop skewing, loop shifting and loop tiling. However, they are mostly applied at compile-time, either on the source code, or on an intermediate representation form of the code. The most advanced techniques are related to the polyhedral model.

[1] with the exception of Gentoo that recompiles every installed package
[2] http://docs.fedoraproject.org/en-US/Fedora/16/html/Release_Notes/sect-Release_Notes-Welcome_to_Fedora_16.html

Applying such advanced loop optimizing transformations at runtime, on a currently running binary code, without any previous knowledge, is our challenging goal. The same goal has been addressed in [8], but not at runtime. In this work, the binary code is analyzed and transformed without any constraint regarding the related time overhead. Candidate loops are identified regarding their compliance to the polyhedral model: the loop bounds and memory references must be convertible into linear functions of the loop indices. Then, compliant loop nests are translated into an equivalent program in C source code, in order to be used as input for the source-to-source polyhedral compiler Pluto [29]. The resulting optimized code is then compiled and re-injected into the original binary code.

While a similar approach should be considered to reach the same goal at runtime, it must be handled differently regarding three main issues:

1. At runtime, the time overhead of the employed analysis and optimization techniques must be small. Thus, any translation to source code, that would require costly steps for the de-compilation/re-compilation phases, must be avoided.
2. Static approaches, as the one presented in [8], can only handle loops that are syntactically compliant with the polyhedral model. However, it has been shown, with the Apollo framework, that loops may exhibit a compliant behavior at runtime. Since we target runtime optimizations, we also can take advantage of the information that is only available at runtime, and maybe also use speculative techniques.
3. Binary codes may hide some interesting properties of the embedded loops, and may need very complex analysis techniques for discovering such properties. In short, a whole compiler for binary codes would be required.

To address these issues, we are currently investigating the strategy consisting first of translating, at runtime, any selected loop nest into the LLVM [3] intermediate representation form (LLVM-IR). This representation offers several advantages:

- Analysis and transformation passes of the LLVM compiler can be used on-the-fly, in order to discover and compute relevant information, and to safely transform the code;
- The LLVM just-in-time compiler can be used to compile the optimized code, which is in LLVM-IR, as an executable;
- Existing tools for loop optimization can be used, as Polly [4], for static polyhedral-compliant loops, or Apollo, for dynamic polyhedral-compliant loops.

Hence, this strategy requires a fast binary-to-LLVM-IR translator. For this purpose, we are currently using and extending McSema [5], which is a library for translating the semantics of native code to LLVM-IR. McSema supports translation of x86 machine code, including integer, floating point, and SSE instructions. Control flow recovery is separated from translation, permitting the use of custom control flow recovery front-ends.

For McSema to be able to handle mostly any code, we had to parametrize carefully its translation rules, and also to add some x86 SSE instructions that were not handled. McSema was recently plugged to the Padrone platform. Thus, any hot loop nest is now automatically converted into LLVM-IR, as illustrated in Figure 2.

Instead of taking as input a binary file, McSema takes as input a code extract containing a hot loop nest, thanks to the code address provided by Padrone. Then, McSema builds the control flow graph of the input code and generates a corresponding LLVM-IR. The next step is to plug the polyhedral LLVM compiler Polly (phases *Canonicalication* to *CodeGeneration* in Figure 2), in order to generate automatically an optimized version of the target loop nest, that will be then compiled using the LLVM just-in-time compiler and re-injected in the running code.

7.7. Combining Locking and Data Management Interfaces

Participants: Jens Gustedt, Mariem Saied, Daniel Salas.

[3] http://llvm.org
[4] http://polly.llvm.org
[5] https://github.com/trailofbits/mcsema

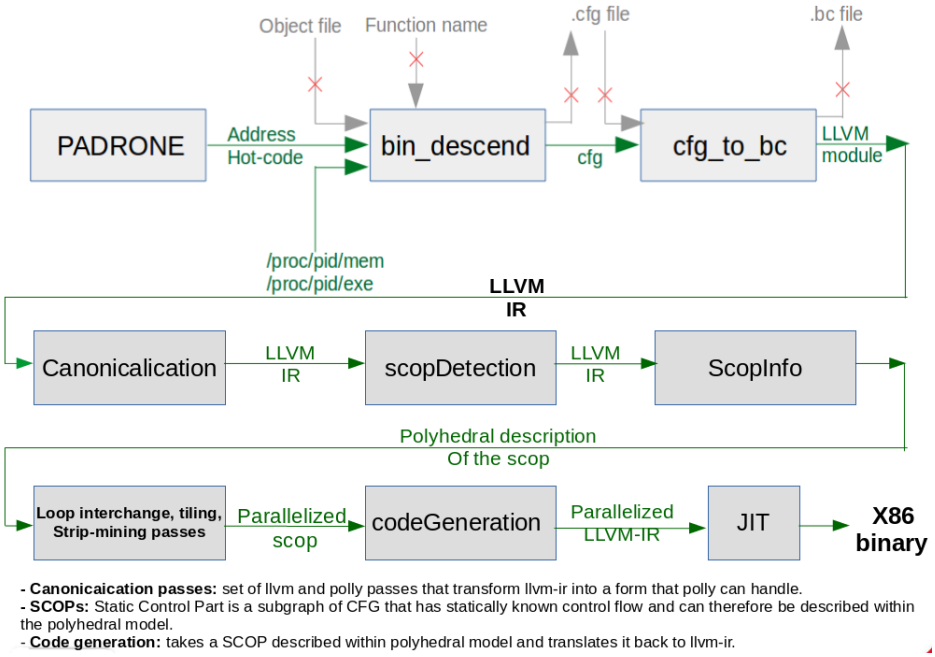

Figure 2. High-level Binary Loop Optimization through LLVM-IR

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 [5] shows how to produce deadlock-free iterative programs that implement strong overlapping between communication, IO and computation. The collaboration with Soumeya Hernane has continued after her thesis defence in 2013. It extends distributed lock mechanisms and combines them with implicit data management, and resulted in a journal submission, see [26].

A new implementation (ORWL) of our ideas of combining control and data management in C has been undertaken, see 6.9. In previous work it has demonstrated its efficiency for a large variety of platforms. In 2015, work on the ORWL model and library has gained vigor with the thesis of Mariem Saied (Inria) 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 2015, a new domain specific language (DSL) has been developed 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. In addition, work has been started to encapsulate imaging applications that use certain pipeline patterns to describe dependencies between computational task.

7.8. Efficient Execution of Polyhedral Codes on GPU and CPU+GPU Systems

Participants: Jean-François Dollinger, Vincent Loechner.

This is the main result of Jean-François Dollinger's PhD, started in 2012 and defended on July the 1st, 2015 [11].

Recent architectures complexity makes it difficult to statically predict the performance of a program. We have developped a reliable and accurate parallel loop nests execution time prediction method on GPUs for

polyhedral codes. It is entirely automatic, and it based on three stages: static code generation, offline profiling on the target architecture, and online prediction.

In addition, we derived two techniques to fully exploit the computing resources at disposal on a computer. The first technique consists in jointly using all CPU cores and GPUs for executing a code. In order to achieve good performance, it is mandatory to consider load balance, in particular by predicting the execution time of a loop nest distributed on all those processing units. The runtime scheduler uses the profiling results to predict the execution times and adjust the parallel loop bounds to ensure load balance. The second technique puts CPU and GPU in a competition: instances of the considered code are simultaneously executed on CPU and GPU. The winner of the competition notifies its completion to the other instance, implying its termination.

7.9. 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 rather new research direction which strengthens CAMUS'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 2015, we presented our interactive tool, *Clint*, that maps direct manipulation of the visual representation to polyhedral program transformations with real-time semantics preservation feedback. We conducted two user studies showing that Clint's visualization can be accurately understood by both experts and non-expert programmers, and that the parallelism can be extracted better from Clint's representation than from the source code in many cases [21]. We are planing a first release of that tool in the coming year.

7.10. Automatic Generation of Adaptive Simulation Codes

Participants: Cédric Bastoul, César Sabater.

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. We are in the process of a collaboration with Inria Nancy Grand Est team TONUS, specialized on applied mathematics (contact: Philippe Helluy), to bring models and techniques from this field to compilers. First results, investigated during the Inria Internship Program of César Sabater, have been presented to the SimRace international conference dedicated on industrial fluid simulation applications [16].

7.11. 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 found the first solution in 2015. 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 in 2015 to be presented in one of the top conferences on optimization techniques: CGO 2016 [15]. Subsequent work and a first release of the tool implementing the technique is planned during 2016.

8. Bilateral Contracts and Grants with Industry

8.1. Bilateral Contracts with Industry

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. Luis Esteban Campostrini has been recruited as PhD student in this project. His work is focusing in extending advanced loop optimization techniques to nonlinear loops using a linear virtual data layout remapping. Artiom Baloian has been recruited in October 2015 as research engineer, in order to make the Apollo framework applicable to ARM Cortex platforms and to merge all the last extensions inside the framework.

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 [6]. 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 ALF, Nabil Hallou's PhD thesis focusing on dynamic optimization of binary code.

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

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
- Intel, Santa Clara, CA, USA
- UPMARC, University of Uppsala, Sweden
- University of Batna, Algeria
- Ohio State University, Colombus, USA
- Louisiana State University, Baton Rouge, USA
- Indian Institute of Science (IIIS) Bangalore, India
- University of Delaware, DE, USA

9.3. International Research Visitors

9.3.1. Visits of International Scientists

Professor P. Sadayappan from Ohio State University, USA, has been visiting the CAMUS team from November the 4th to November the 7th. He took part of Aravind Sukumaran-Rajam's PhD jury as a reviewer and made several presentations of his research work.

10. Dissemination

10.1. Promoting Scientific Activities

10.1.1. Scientific events organisation

10.1.1.1. Member of the organizing committees

Cédric Bastoul has been co-organizing the HIP3ES 2015 workshop (High Performance Energy Efficient Embedded Systems) held in conjunction with the international conference HiPEAC 2015. He is also currently organizing the next HIP3ES event, to be held in conjunction with the international conference HiPEAC 2016.

10.1.2. Scientific events selection

10.1.2.1. Member of the conference program committees

Philippe Clauss and Vincent Loechner have been part of the program committee of IMPACT 2015 (International Workshop on Polyhedral Compilation Techniques), held in conjunction with the international conferences HiPEAC 2015.

Philippe Clauss has been part of the program committee of the third workshop on Energy Efficient Super Computing (E2SC), held in conjunction with SC15.

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

Cédric Bastoul and Vincent Loechner have been part of the program committee of both HIP3ES 2015 and HIP3ES 2016 (International Workshop on High Performance Energy Efficient Embedded Systems), held in conjunction with the international conferences HiPEAC 2015 (resp. HiPEAC 2016).

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

Cédric Bastoul has been part of the program committee of PARMA+DITAM 2015 and PARMA+DITAM 2016 (Workshop on Parallel Programming and Run-Time Management Techniques for Many-core Architectures + Workshop on Design Tools and Architectures for Multicore Embedded Computing Platforms), held in conjunction with HiPEAC 2015 (resp. HiPEAC 2016).

10.1.2.2. Reviewer

Cédric Bastoul has been reviewer for the following conferences and workshops: PARMA 2015 and 2016 (International Workshop on Parallel Programming and Run-Time Management Techniques for Many-core Architectures), IMPACT 2016 (International Workshop on Polyhedral Compilation Techniques), HIP3ES 2015 and 2016 (International Workshop on High Performance Energy Efficient Embedded Systems).

10.1.3. Journal

10.1.3.1. Editorial board membership

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

In 2014, the episcience platform for open access journals has been created in a joint effort by Inria and CNRS. In 2015, DMTCS, as one of the first journals, has moved to this platform presenting a new server interface which lives on top of the national scientific archive HAL. We have been a driving force in the definition and debugging of the new platform, see [1].

10.1.3.2. Reviewer - Reviewing activities

Jens Gustedt has served as a reviewer for *Theory of Computing Systems* and *IEEE Transactions on Parallel and Distributed Systems*.

Philippe Clauss has served as a reviewer for the following journals: *ACM Transactions on Architecture and Code Optimization, ACM Transactions on Programming Languages and Systems*.

Alain Ketterlin has served as a reviewer for the following journals: *Parallel Computing*, and *International Journal of Parallel Programming*. He has also served as sub-reviewer for the PACT 2015 conference.

Cédric Bastoul has been reviewer for the *Parallel Computing international journal* (ParCo).

10.1.4. Scientific expertise

Cédric Bastoul has been an expert for the European Commission for the call FETHPC of the H2020 programme. He also has been an expert for the French research ministry and the French finance ministry for the research tax credit programme.

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, see [27].

This work on the C programming language also gave rise to the proposal of a language extension, Modular C, see [25].

10.2. Teaching - Supervision - Juries

10.2.1. Teaching

Licence : Alain Ketterlin, Algorithmique et Structures de Données (Licence de Mathématique), 40h, L3, Université de Strasbourg, France

Licence : Alain Ketterlin, Réseaux et Protocoles (Licence d'Informatique), 64h, L3, Université de Strasbourg, France

Master : Alain Ketterlin, Ingénierie de la preuve, 21h, M1, Université de Strasbourg, France.

Licence : Alain Ketterlin, Architecture et Programmation des mécanismes de base d'un système informatique, 68h, L1 (IUT), Université de Strasbourg, France.

Licence : Alain Ketterlin, Modèles de calcul, 14h, L1, 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

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 : Vincent Loechner, Systèmes d'exploitation, 38h, 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 : Eric Violard, Programmation Fonctionnelle, 21h, L2, Université de Strasbourg, France

Licence : Eric Violard, Architecture des Ordinateurs, 16h, L2, Université de Strasbourg, France

Licence : Eric Violard, Logique et Programmation Logique, 26h, L2, Université de Strasbourg, France

Licence : Eric Violard, Algorithmique et Structure de Données, 35h, L3, Université de Strasbourg, France

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

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

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

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

Master : Cédric Bastoul, Advanced Compilation, 23h, M1, Strasbourg University, France

Master : Cédric Bastoul, Parallelism, 16h, M1, Strasbourg University, France

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

10.2.2. Supervision

PhD in progress: Yann Barsamian, *Space-Filling Curves and their Application to the Numerical Resolution of Vlasov Equations*, since Oct 2014, Eric Violard

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

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: Juan Manuel Martinez Caamaño, *Dynamic and flexible generation of parallel loops using a dedicated intermediate representation*, since November 2013, Philippe Clauss and Philippe Helluy (IRMA lab., University of Strasbourg)

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

PhD in progress: Luis Esteban Campostrini, *Virtual linear data layout*, since January 2015, Philippe Clauss

PhD in progress : Lénaïc Bagnères, Automatic parallelization and optimization for manycore architectures, November 2012, Christine Eisenbeis and Cédric Bastoul

PhD in progress : Alexander Zinenko, Interactive program manipulation, September 2013, Stéphane Huot and Cédric Bastoul

PhD: Aravind Sukumaran-Rajam, *Beyond the Realm of the Polyhedral Model: Combining Speculative Program Parallelization with Polyhedral Compilation*, University of Strasbourg, November the 5th, 2015, Philippe Clauss

PhD: Imen Fassi, *XFOR (Multifor): A New Programming Structure to Ease the Formulation of Efficient Loop Optimizations*, University of Strasbourg, November the 27th, 2015, Philippe Clauss

PhD: Jean-François Dollinger, *A framework for efficient execution on GPU and CPU+GPU systems*, University of Strasbourg, July the 1st, 2015, Vincent Loechner and Philippe Clauss

10.2.3. Juries

Jens Gustedt participated to the following PhD jury in 2015:

Date	**Candidate**	**Place**	**Role**
July 8	Florian David	Univ. Pierre et Marie Curie, Paris	Examiner

Philippe Clauss participated to the following PhD juries in 2015:

Date	**Candidate**	**Place**	**Role**
Oct. 9	Hangbing Li	Univ. Rennes 1	Reviewer
Sept. 18	Nicolas Triquenaux	Univ. Versailles Saint-Quentin-en-Yvelines	Reviewer

Alain Ketterlin was a member of the following PhD juries in 2015:

Date	**Candidate**	**Place**	**Role**
Mar. 5	Bharath Narasimha-Swamy (Adv.: André Seznec)	Univ. Rennes 1	Examiner
Nov. 5	Fabien Rozar (Adv.: Jean Roman & Guillaume Latu)	Univ. Bordeaux	Examiner

Cédric Bastoul participated to the following HDR jury in 2015:

Date	**Candidate**	**Place**	**Role**
May 18	Corinne Ancourt	École des Mines de Paris	President
Nov. 2	Erven Rohou	Université de Rennes 1	Examiner

Cédric Bastoul participated to the following PhD jurys in 2015:

Date	**Candidate**	**Place**	**Role**
September 25	Riyadh Badhdadi	Pierre et Marie Curie University	Reviewer

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 stack**overflow** community a technical Q&A site for programming and related subjects. A book about modern C is in preparation.

Cédric Bastoul presented lectures and activities at the *Kids University* event at the University of Strasbourg in November 2015

Cédric Bastoul prepared activities for *Fête de la Science* at University of Paris-Sud in October 2015

11. Bibliography

Major publications by the team in recent years

[1] C. BERTHAUD, L. CAPELLI, J. GUSTEDT, C. KIRCHNER, K. LOISEAU, A. MAGRON, M. MEDVES, A. MONTEIL, G. RIVERIEUX, L. ROMARY. *EPISCIENCES - an overlay publication platform*, in "ELPUB2014 - International Conference on Electronic Publishing", Thessalonique, Greece, D. P. POLYDORATOU (editor), IOS Press, June 2014, pp. 78-87 [*DOI :* 10.3233/978-1-61499-409-1-78], https://hal.inria.fr/hal-01002815

[2] J. C. BEYLER, P. CLAUSS. *Performance driven data cache prefetching in a dynamic software optimization system*, in "ICS '07: Proceedings of the 21st annual international conference on Supercomputing", New York, NY, USA, ACM, 2007, pp. 202–209, http://doi.acm.org/10.1145/1274971.1275000

[3] J. C. BEYLER, M. KLEMM, P. CLAUSS, M. PHILIPPSEN. *A meta-predictor framework for prefetching in object-based DSMs*, in "Concurr. Comput. : Pract. Exper.", September 2009, vol. 21, pp. 1789–1803

[4] P. CLAUSS, F. J. FERNÁNDEZ, D. GARBERVETSKY, S. VERDOOLAEGE. *Symbolic polynomial maximization over convex sets and its application to memory requirement estimation*, in "IEEE Transactions on Very Large Scale Integration (VLSI) Systems", Aug 2009, vol. 17, n° 8, pp. 983-996

[5] P.-N. CLAUSS, J. GUSTEDT. *Iterative Computations with Ordered Read-Write Locks*, in "Journal of Parallel and Distributed Computing", 2010, vol. 70, n° 5, pp. 496-504 [*DOI :* 10.1016/J.JPDC.2009.09.002], https://hal.inria.fr/inria-00330024

[6] 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

[7] 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

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

[9] 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° 2, pp. 8.1-8.27 [*DOI :* 10.1145/2207222.2207224], http://hal.inria.fr/inria-00582388

[10] S. VERDOOLAEGE, R. SEGHIR, K. BEYLS, V. LOECHNER, M. BRUYNOOGHE. *Counting Integer Points in Parametric Polytopes Using Barvinok's Rational Functions*, in "Algorithmica", 2007, vol. 48, n° 1, pp. 37–66, http://dx.doi.org/10.1007/s00453-006-1231-0

Publications of the year

Doctoral Dissertations and Habilitation Theses

[11] J.-F. DOLLINGER. *A framework for efficient execution on GPU and CPU+GPU systems*, Université de Strasbourg, July 2015, https://hal.inria.fr/tel-01251719

[12] I. FASSI. *XFOR (Multifor): A New Programming Structure to Ease the Formulation of Efficient Loop Optimizations*, Université de Strasbourg, November 2015, https://hal.inria.fr/tel-01251721

[13] A. SUKUMARAN-RAJAM. *Beyond the Realm of the Polyhedral Model: Combining Speculative Program Parallelization with Polyhedral Compilation*, Université de Strasbourg, November 2015, https://hal.inria.fr/tel-01251748

Articles in International Peer-Reviewed Journals

[14] A. SUKUMARAN-RAJAM, P. CLAUSS. *The Polyhedral Model of Nonlinear Loops*, in "ACM Transactions on Architecture and Code Optimization", December 2015, vol. 12, n^{o} 4 [*DOI :* 10.1145/2838734], https://hal.inria.fr/hal-01244464

International Conferences with Proceedings

[15] 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

[16] C. BASTOUL, C. SABATER. *Automatic Generation of Adaptive Simulation Codes*, in "SimRace, Conference on Numerical Methods and High Performance Computing for Industrial Fluid Flows", Rueil-Malmaison, France, IFPEN, December 2015, https://hal.inria.fr/hal-01245558

[17] T. BUCHERT, L. NUSSBAUM, J. GUSTEDT. *Towards Complete Tracking of Provenance in Experimental Distributed Systems Research*, in "REPPAR - Second International Workshop on Reproducibility in Parallel Computing – held together with Euro-Par", Vienna, Austria, August 2015, https://hal.inria.fr/hal-01191855

[18] P. CLAUSS. *Mind The Gap! A study of some pitfalls preventing peak performance in polyhedral compilation using a polyhedral antidote*, in "IMPACT 2015, Fifth International Workshop on Polyhedral Compilation Techniques, In conjunction with HiPEAC 2015", Amsterdam, Netherlands, January 2015, https://hal.inria.fr/hal-01099583

[19] I. FASSI, P. CLAUSS. *XFOR: Filling the Gap between Automatic Loop Optimization and Peak Performance*, in "14th International Symposium on Parallel and Distributed Computing", Limassol, Cyprus, IEEE (editor), June 2015, https://hal.inria.fr/hal-01155144

[20] N. HALLOU, E. ROHOU, P. CLAUSS, A. KETTERLIN. *Dynamic Re-Vectorization of Binary Code*, in "International Conference on Embedded Computer Systems: Architectures, Modeling and Simulation - SAMOS XV", Agios Konstantinos, Greece, July 2015, https://hal.inria.fr/hal-01155207

[21] O. ZINENKO, C. BASTOUL, S. HUOT. *Manipulating Visualization, Not Codes*, in "International Workshop on Polyhedral Compilation Techniques (IMPACT)", Amsterdam, Netherlands, January 2015, 8 p. , https://hal.inria.fr/hal-01100974

Conferences without Proceedings

[22] J.-F. DOLLINGER, V. LOECHNER. *CPU+GPU Load Balance Guided by Execution Time Prediction*, in "Fifth International Workshop on Polyhedral Compilation Techniques (IMPACT 2015)", Amsterdam, Netherlands, January 2015, https://hal.inria.fr/hal-01095890

[23] A. SUKUMARAN-RAJAM, L. E. CAMPOSTRINI, M. JUAN MANUEL, P. CLAUSS. *Speculative Runtime Parallelization of Loop Nests: Towards Greater Scope and Efficiency*, in "20th International Workshop on High-level Parallel Programming Models and Supportive Environments, held in conjunction with 29th IEEE International Parallel & Distributed Processing Symposium", Hyderabad, India, May 2015, https://hal.inria.fr/hal-01155172

Research Reports

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

[25] J. GUSTEDT. *Modular C*, Inria, June 2015, n^{o} RR-8751, https://hal.inria.fr/hal-01169491

[26] S. L. HERNANE, J. GUSTEDT, M. BENYETTOU. *Data handover on a peer-to-peer system*, Inria Nancy - Grand Est (Villers-lès-Nancy, France) ; Inria, February 2015, n^{o} RR-8690, 37 p. , https://hal.inria.fr/hal-01120837

[27] D. KEATON, J. GUSTEDT. *Underspecified Aspects of Threads in C*, Inria, 2015, n^{o} RT-0470, https://hal.inria.fr/hal-01230011

References in notes

[28] 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

[29] U. BONDHUGULA, A. HARTONO, J. RAMANUJAM, P. SADAYAPPAN. *A practical automatic polyhedral parallelizer and locality optimizer*, in "PLDI '08", ACM, 2008, pp. 101–113

[30] T. GROSSER, J. RAMANUJAM, L.-N. POUCHET, P. SADAYAPPAN, S. POP. *Optimistic Delinearization of Parametrically Sized Arrays*, in "Proceedings of the 29th ACM on International Conference on Supercomputing", New York, NY, USA, ICS '15, ACM, 2015, pp. 351–360, http://doi.acm.org/10.1145/2751205.2751248

[31] 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

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

[33] T. YUKI, P. FEAUTRIER, S. RAJOPADHYE, V. SARASWAT. *Array Dataflow Analysis for Polyhedral X10 Programs*, in "Proceedings of the 18th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming", New York, NY, USA, PPoPP '13, ACM, 2013, pp. 23–34, http://doi.acm.org/10.1145/2442516.2442520