RESEARCH CENTRE

Nancy - Grand Est

IN PARTNERSHIP WITH:

Université de Strasbourg

2020 ACTIVITY REPORT

Project-Team CAMUS

Compiling for Multicore Architectures

IN COLLABORATION WITH: ICube

DOMAIN

Algorithmics, Programming, Software and Architecture

THEME

Architecture, Languages and Compilation

Contents

Project-Team CAMUS

Creation of the Team: 2009 July 01, updated into Project-Team: 2019 March 01

Keywords

Computer sciences and digital sciences

A1.1.1. – Multicore, Manycore

A1.1.4. – High performance computing

A2.1.1. – Semantics of programming languages

A2.1.6. – Concurrent programming

A2.2.1. – Static analysis

A2.2.4. – Parallel architectures

A2.2.5. – Run-time systems

A2.2.6. – GPGPU, FPGA...

A2.2.7. – Adaptive compilation

A2.4. – Formal method for verification, reliability, certification

Other research topics and application domains

B4.5.1. – Green computing

B6.1.1. – Software engineering

B6.6. – Embedded systems

1 Team members, visitors, external collaborators

Research Scientists

- Bérenger Bramas [Inria, Researcher]
- Arthur Charguéraud [Inria, Researcher]
- Jens Gustedt [Inria, Senior Researcher, HDR]

Faculty Members

- Philippe Clauss [Team leader, Univ de Strasbourg, Professor, HDR]
- Cédric Bastoul [Univ de Strasbourg, Professor, until Mar 2020, HDR]
- Alain Ketterlin [Univ de Strasbourg, Associate Professor]
- Vincent Loechner [Univ de Strasbourg, Associate Professor]
- Éric Violard [Univ de Strasbourg, Associate Professor, HDR]

Post-Doctoral Fellow

- Damien Rouhling [Inria, until Aug 2020]

PhD Students

- Clement Flint [Univ de Strasbourg, from Nov 2020]
- Salwa Kobeissi [Univ de Strasbourg, ATER, From Sep 2020]
- Harenome Ranaivoarivony-Razanajato [Univ de Strasbourg, until Sep 2020]

Technical Staff

- Paul Cardosi [Inria, Engineer]

Interns and Apprentices

- Emilien Bauer [Univ de Strasbourg]
- Nicolas Chappe [École Normale Supérieure de Lyon, from Sep 2020]
- Clement Flint [Univ de Strasbourg, until Jul 2020]
- Garip Kusoglu [Univ de Strasbourg]

Administrative Assistant

- Ouiza Herbi [Inria]

2 Overall objectives

The CAMUS team is focusing on developing, adapting and extending automatic parallelization and optimization 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 four main issues that are closely related to reach the following objectives: performance, correctness 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 transformation proofs (where the correctness of many static and dynamic program transformations has to be ensured).

3 Research program

The various objectives we are expecting to reach are directly related to the search of adequacy between the software and the new multicore processors evolution. They also correspond to the main research directions suggested by Hall, Padua and Pingali in [40]. Performance, correctness 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

The development of efficient and correct applications for multicore processors requires 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 performed, resulting in *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 actual 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 (values of the variables, accessed memory addresses, 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, the complexity of current and future architectures avoids assuming an optimal behavior regarding a given program version. A monitoring process will make it possible to select on-the-fly the best parallelization.

These different parallelization steps are schematized in figure 1.

Our project relies 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 correctness 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

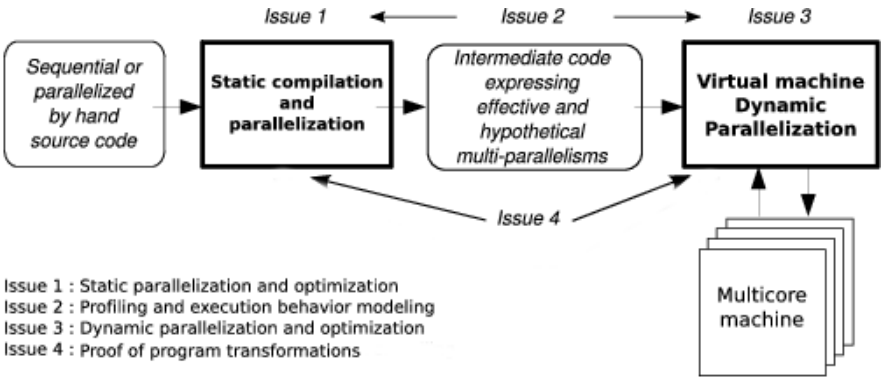

Figure 1: Steps for automatic parallelization on multicore architectures.

dynamically by the virtual machine, must preserve the initial semantics. This must be proved formally (issue 4).

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

3.1 Static Parallelization and Optimization

Participants: Vincent Loechner, Philippe Clauss, Éric Violard, Cédric Bastoul, Arthur Charguéraud, Bérenger Bramas, Harenome Ranaivoarivony-Razanajato

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 [39]. Low-level optimizations, in the assembly code generated by the compiler, have also been extensively dealt with 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 architectures and expressing many potential parallelisms.

3.2 Profiling and Execution Behavior Modeling

Participants: Alain Ketterlin, Philippe Clauss, Salwa Kobeissi

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.

3.3 Dynamic Parallelization and Optimization, Virtual Machine

Participants: Philippe Clauss, Salwa Kobeissi, Jens Gustedt, Alain Ketterlin, Bérenger Bramas

Dynamic parallelization and optimization has become essential with the advent of the new multicore architectures. When using a dynamic scheme, the 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 scheme 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 provides a significant optimization ability compared to a static compiler, while observing the evolution of the availability of live resources.

3.4 Proof of Program Transformations for Multicore Programs

Participants: Arthur Charguéraud, Alain Ketterlin, Éric Violard,

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 optimization to produce data-race free code. For the second stage of optimization we will first assume that the input code is data-race free. We will prove those transformations using Appel's concurrent separation logic [41]. Proving transformations involving programs which are not data-race free will constitute a longer term research goal.

4 Application domains

Computational performance being our main objective, our target applications are characterized by intensive computation phases. Such applications are numerous in the domains of scientific computations, optimization, data mining and multimedia. In particular, several members of the team have contributed to high-performance code for numerical simulation of differential equations.

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 Awards

The paper *Provably and Practically Efficient Granularity Control* by Umut A. Acar (CMU and Inria), Vitaly Aksenov (Inria and ITMO University), Arthur Charguéraud (Inria & Université de Strasbourg, CNRS, ICube), and Mike Rainey (Indiana University and Inria), published at the Symposium on Principles and Practice of Parallel Programming (PPoPP) 2019, was selected as a **SIGPLAN Research Highlights Papers** in July 2020. Only a handful of papers are selected each year from the domain of research on programming languages.

6 New software and platforms

6.1 New software

6.1.1 CLooG

Name: Code Generator in the Polyhedral Model

Keywords: Polyhedral compilation, Optimizing compiler, Code generator

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.

Release Contributions: It mostly solves building and offers a better OpenScop support.

URL: http://www.cloog.org

Author: Cédric Bastoul

Contacts: Cédric Bastoul, Albert Cohen

Participant: Cédric Bastoul

6.1.2 OpenScop

Name: A Specification and a Library for Data Exchange in Polyhedral Compilation Tools

Keywords: Polyhedral compilation, Optimizing compiler

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.

URL: http://icps.u-strasbg.fr/people/bastoul/public_html/development/openscop/

Contact: Cédric Bastoul

Participant: Cédric Bastoul

6.1.3 ORWL

Name: Ordered Read-Write Lock

Keywords: Task scheduling, Deadlock detection

Functional Description: ORWL is a reference implementation of the Ordered Read-Write Lock tools. The macro definitions and tools for programming in C99 that have been implemented for ORWL have been separated out into a toolbox called P99.

Publications: inria-00330024, hal-01621936, hal-01325648, hal-01337093

Author: Jens Gustedt

Contacts: Jens Gustedt, Stéphane Vialle, Mariem Saied

Participants: Jens Gustedt, Mariem Saied, Stéphane Vialle

6.1.4 CFML

Name: Interactive program verification using characteristic formulae

Keywords: Coq, Software Verification, Deductive program verification, Separation Logic

Functional Description: The CFML tool supports the verification of OCaml programs through interactive Coq proofs. CFML proofs establish the full functional correctness of the code with respect to a specification. They may also be used to formally establish bounds on the asymptotic complexity of the code. The tool is made of two parts: on the one hand, a characteristic formula generator implemented as an OCaml program that parses OCaml code and produces Coq formulae, and, on the other hand, a Coq library that provides notations and tactics for manipulating characteristic formulae interactively in Coq.

URL: http://www.chargueraud.org/softs/cfml/

Contact: Arthur Charguéraud

Participants: Arthur Charguéraud, Armaël Guéneau, François Pottier

6.1.5 SPETABARU

Name: SPEculative TAsk-BAsed RUntime system

Keywords: HPC, Parallel computing, Task-based algorithm

Functional Description: SPETABARU is a task-based runtime system for multi-core architectures that includes speculative execution models. It is a pure C++11 product without external dependency. It uses advanced meta-programming and allows for an easy customization of the scheduler. It is also capable to generate execution traces in SVG to better understand the behavior of the applications.

URL: https://gitlab.inria.fr/bramas/spetabaru

Contact: Bérenger Bramas

6.1.6 APAC

Keywords: Source-to-source compiler, Automatic parallelization, Parallel programming

Scientific Description: APAC is a compiler for automatic parallelization that transforms C++ source code to make it parallel by inserting tasks. It uses the tasks+dependencies paradigm and relies on OpenMP or SPETABARU as runtime system. Internally, it is based on Clang-LLVM.

Functional Description: Automatic task-based parallelization compiler

URL: https://gitlab.inria.fr/bramas/apac

Contact: Bérenger Bramas

Participants: Bérenger Bramas, Stéphane Genaud, Garip Kusoglu

6.1.7 LetItBench

Name: Lenient to Errors, Transformations, Irregularities and Turbulence Benchmarks

Keywords: Approximate computing, Benchmarking

Functional Description: LetItBench is a benchmark set to help evaluating works on approximate compilation techniques. We propose a set of meaningful applications with an iterative kernel, that is not too complex for automatic analysis and can be analyzed by polyhedral tools. The benchmark set called LetItBench (Lenient to Errors, Transformations, Irregularities and Turbulence Benchmarks) is composed of standalone applications written in C, and a benchmark runner based on CMake. The benchmark set includes fluid simulation, FDTD, heat equations, game of life or K-means clustering. It spans various kind of applications that are resilient to approximation.

URL: https://github.com/Syllo/LetItBench

Contacts: Maxime Schmitt, Cédric Bastoul

6.1.8 ACR

Name: Adaptive Code Refinement

Keywords: Approximate computing, Optimizing compiler

Functional Description: ACR is to approximate programming what OpenMP is to parallel programming. It is an API including a set of language extensions to provide the compiler with pertinent information about how to approximate a code block, a high-level compiler to automatically generate the approximated code, and a runtime library to exploit the approximation information at runtime according to the dataset properties. ACR is designed to provide approximate computing to non experts. The programmer may write a trivial code without approximation, provide approximation information thanks to pragmas, and let the compiler generate an optimized code based on approximation.

URL: https://github.com/Syllo/acr

Contacts: Maxime Schmitt, Cédric Bastoul

6.1.9 APOLLO

Name: 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. Some recent extensions enabling dynamic multi-versioning have been implemented in 2020.

URL: https://webpages.gitlabpages.inria.fr/apollo

Contact: Philippe Clauss

Participants: Aravind Sukumaran-Rajam, Juan Manuel Martinez Caamaño, Manuel Selva, Philippe Clauss

6.1.10 inastemp

Keywords: C++, Vectorization, SIMD, ARM

Functional Description: Inastemp provides a set of C++ classes to make vectorization with intrinsics easier. It aims at developing numerical kernels by separating the algorithm from the hardware target. Inastemp comes with several examples and patterns related to widespread use-cases. The library requires basic C++ knowledge about classes, templates, etc. It is not mandatory to really

know what is vectorization all about, but it certainly helps. For example, the increment of loops is usually tied to the underlying size of the vector, which means that one should understand why and where the increment is not equal to 1. A good way to use Inastemp would be to have a software engineer managing the inclusion and hiding the small complexities of the template process, and to have the scientists work with basic simple functions templatized against a vector type.

URL: https://gitlab.inria.fr/bramas/inastemp

Contact: Bérenger Bramas

6.1.11 Farm-SVE

Keywords: Vectorization, ARM

Functional Description: Naive/scalar implementation of the ARM C language extensions (ACLE) for the ARM Scalable Vector Extension (SVE) in standard C++.

URL: https://gitlab.inria.fr/bramas/farm-sve

Contact: Bérenger Bramas

6.1.12 TBFMM

Keywords: FMM, OpenMP, C++

Functional Description: TBFMM is a Fast Multipole Method (FMM) library parallelized with the task-based method. It is designed to be easy to customize by creating new FMM kernels or new parallelization strategies. It uses the block-tree hierarchical data structure (also known as the group-tree), which is well-designed for the task-based parallelization, and can be easily extended to heterogeneous architectures (not yet supported but WIP). Users can implement new FMM kernels, new types of interacting elements or even new parallelization strategies. As such, it can be used as a simulation toolbox for scientists in physics or applied mathematics. It enables users to perform simulations while delegating the data structure, the algorithm and the parallelization to the library. Besides, TBFMM can also provide an interesting use case for the HPC research community regarding parallelization, optimization and scheduling of applications handling irregular data structures.

URL: https://gitlab.inria.fr/bramas/tbfmm

Contact: Bérenger Bramas

6.1.13 shnell

Name: shnell

Keywords: Programming language, Source-to-source compiler

Scientific Description: Locally replaceable code snippets can be used to easily specify and prototype compiler and language enhancements for the C language that work by local source-to-source transformation. The shnell toolbox implements the feature and provides many directives that can be used for compile time configuration and tuning, code unrolling, compile time expression evaluation and program modularization. The tool is also easily extensible by simple filters that can be programmed with any suitable text processing framework.

Functional Description: Shnell is a tool to easily develop and prototype compiler and language enhancements that can be expressed by source-to-source transformation of C code

URL: https://gustedt.gitlabpages.inria.fr/shnell/

Publication: hal-02998412

Contact: Jens Gustedt

Participant: Jens Gustedt

6.1.14 PolyLib

Name: The Polyhedral Library

Keywords: Rational polyhedra, Library, Polyhedral compilation

Scientific Description: A C library used in polyhedral compilation, as a basic tool used to analyze, transform, optimize polyhedral loop nests. It has been shipped in the polyhedral tools Cloog and Pluto.

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.

Release Contributions: Continuous Integration process has been added to the latest version. The license has been moved from GPL to MIT.

URL: http://icps.u-strasbg.fr/PolyLib/

Contact: Vincent Loechner

Participant: Vincent Loechner

7 New results

7.1 Rec2Poly: Converting Recursions to Polyhedral Optimized Loops Using an Inspector-Executor Strategy

Participants: Salwa Kobeissi, Philippe Clauss

We propose Rec2Poly, a framework which detects automatically if recursive programs may be transformed into affine loops that are compliant with the polyhedral model. If successful, the replacing loops can then take advantage of advanced loop optimizing and parallelizing transformations as tiling or skewing.

Rec2Poly is made of two main phases: an offline profiling phase and an inspector-executor phase. In the profiling phase, the original recursive program, which has been instrumented, is run. Whenever possible, the trace of collected information is used to build equivalent affine loops from the runtime behavior. Then, an inspector-executor program is automatically generated, where the inspector is made of a light version of the original recursive program, whose aim is reduced to the generation and verification of the information which is essential to ensure the correctness of the equivalent affine loop program. The collected information is mainly related to the touched memory addresses and the control flow of the so-called “impacting” basic blocks of instructions. Moreover, in order to exhibit the lowest possible time-overhead, the inspector is implemented as a parallel process where several memory buffers of information are verified simultaneously. Finally, the executor is made of the equivalent affine loops that have been optimized and parallelized.

This work is the topic of Salwa Kobeissi’s PhD. A paper has been published at the international conference SAMOS 2020 [27].

7.2 Uniform Random Sampling in Polyhedra

Participants: Philippe Clauss
External collaborator: Benoît Meister, Reservoir Labs, New York, USA

We propose a method for generating uniform samples among a domain of integer points defined by a polyhedron in a multi-dimensional space. The method extends to domains defined by parametric polyhedra, in which a subset of the variables are symbolic. We motivate this work by a list of applications for the method in computer science. The proposed method relies on polyhedral ranking functions, as well as a recent inversion method for them, named *trahrhe* expressions.

This work has been published at the 10th International Workshop on Polyhedral Compilation Techniques [21].

7.3 Runtime Multi-Versioning

Participants: Philippe Clauss
External collaborators: Raquel Lazcano, Daniel Madroñal, Eduardo Juarez, Center of Software Technologies and Multimedia Systems, Universidad Politécnica de Madrid, Spain

Multi-versioning is a code optimization technique that has been proven to be efficient when the execution context related to each generated version can be accurately predicted. Value-based code specialization generates better optimized code where some parameters have been instantiated as constants. This technique's efficiency also relies on an accurate prediction of the parameter values that may occur at runtime. Memoization is also a well-known optimization technique traditionally dedicated to functions where results of calls are stored and then returned as soon as the same inputs occur again.

We propose a runtime framework that implements code multi-versioning and specialization to optimize and parallelize loop kernels that are invoked many times with varying parameters. These parameters may influence the code structure, the touched memory locations, the workload, and the runtime performance. They may also impact the validity of the parallelizing and optimizing polyhedral transformations that are applied on-the-fly.

For a target loop kernel and its associated parameters, a different optimizing and parallelizing transformation is evaluated at each invocation, among a finite set of transformations (multi-versioning and specialization). The best performing transformed code version is stored and indexed using its associated parameters. When every optimizing transformation has been evaluated, the best performing code version regarding the current parameters, which has been stored, is relaunched at next invocations (memoization).

This work has been published at the 29th International Conference on Compiler Construction [18].

7.4 AutoParallel: Automatic Parallelization and Distributed Execution of Affine Loop Nests in Python

Participants: Philippe Clauss
External collaborators: Cristian Ramon-Cortes, Ramon Amela, Jorge Ejarque, Rosa M. Badia, Barcelona Supercomputing Center (BSC), Spain

The last improvements in programming languages and models have focused on simplicity and abstraction, leading Python to the top of the list of the programming languages. However, there is still room for improvement when preventing users from dealing directly with distributed and parallel computing issues. This paper proposes and evaluates AutoParallel, a Python module to automatically find an appropriate task-based parallelisation of affine loop nests and execute them in parallel in a distributed computing infrastructure. It is based on sequential programming and contains one single annotation (in the form of a Python decorator) so that anyone with intermediate-level programming skills can scale up an application to hundreds of cores. The evaluation demonstrates that AutoParallel goes one step further in easing the development of distributed applications. On the one hand, the programmability evaluation highlights the benefits of using a single Python decorator instead of manually annotating each task and its parameters or, even worse, having to develop the parallel code explicitly (e.g., using OpenMP, MPI). On the other hand, the performance evaluation demonstrates that AutoParallel is capable of automatically generating task-based workflows from sequential Python code while achieving the same performances

than manually taskified versions of established state-of-the-art algorithms (i.e., Cholesky, LU, and QR decompositions). Finally, AutoParallel is also capable of automatically building data blocks to increase the tasks' granularity; freeing the user from creating the data chunks, and re-designing the algorithm. For advanced users, we believe that this feature can be useful as a baseline to design blocked algorithms.

This work has been published in the International Journal of High Performance Computing Applications [15].

7.5 OptiTrust: Producing Trustworthy High-Performance Code via Source-to-Source Transformations

Participants: Arthur Charguéraud, Damien Rouhling

Arthur Charguéraud obtained Inria funding for an "exploratory action" (2.5 years of funding, starting from September 2019). The aim of the project is to develop a framework for producing trustworthy high-performance code starting from a high-level description of an algorithm that implements, e.g., a numeric simulation. Damien Rouhling was a postdoc for one year. He designed a framework for performing user-guided source-to-source transformations on C code. An engineer will start in Feb. 2021 to pursue the implementation effort. The goal is to apply the framework to produce, as first case study, a high-performance parallel implementation of a particle-in-cell algorithm used for plasma simulations.

7.6 Separation Logic for Sequential Programs

Participants: Arthur Charguéraud

Separation Logic was introduced 20 years ago. It has since then proved to be a central tool for tackling the formal verification of software components. It has been applied to numerous different programming languages, and to the verification of various kind of programs ranging from low-level operating system kernels to high-level data structures and algorithms. In particular, the approach that consists in embedding Separation Logic in a proof assistant has appeared as the tool of choice for verifying nontrivial pieces of code in a modular way.

To consolidate all the knowledge related to embeddings of Separation Logic in proof assistants, Arthur Charguéraud wrote a course "all in Coq", following the style of the successful Software Foundations series. The course considers sequential programs, leaving out concurrency, and focuses on the construction of a program verification tool, as opposed to its practical use. The latter aspect will be the matter of a future volume.

The course is currently available from the author's webpage, but should soon be available from the Software Foundations website. The first half of the contents of the course has also been published, in traditional LaTeX-style presentation, in the journal *Proceedings of the ACM on Programming Languages* (PACMPL) and was presented at the conference ICFP'20 [13]. An article is in preparation for covering the second half.

7.7 Automatic Task-Based Parallelization using Source to Source Transformations

Participants: Bérenger Bramas, Garip Kusolgu

Bérenger Bramas and Garip Kusolgu worked on a new approach to automatically parallelize any application written in an object-oriented language. The main idea is to parallelize a code as an HPC expert would do it using the task-based method. With this aim, they created a new source-to-source compiler on top of CLang-LLVM called APAC. APAC is able to insert tasks in a source-code by evaluating data access and generating the correct dependencies.

This work has been published at the COMPAS 2020 conference [19].

7.8 Task-Based Parallelization of the Fast-Multipole Method (FMM)

Participants: Bérenger Bramas

Bérenger Bramas implemented TB-FMM, a new FMM library parallelized with OpenMP and SPETABARU, a runtime system developed in the team. TB-FMM is based on original data structures that are used to

study how *C++* classes can be easily moved data movement between memory nodes. It will serve as a basis for future investigation on heterogeneous computing with SPETABARU.

This work has been published at the Journal of Open Source Software [9].

7.9 Automatic Configuration of the Heteroprio Scheduler

Participants: Bérenger Bramas, Clément Flint

Heteroprio is a scheduler designed for heterogeneous machines that was implemented in StarPU, a task-based execution engine on heterogeneous multicore architectures. To use this scheduler, the users must tune it by providing priorities for the different types of tasks that exist in their applications. Consequently, not only it asks for a significant programming effort, but the given configuration might not be efficient because of possible incorrect intuition from the users or because a single application might have different execution scenarios that would execute better with different priorities. Clément Flint and Bérenger Bramas created and evaluated several heuristics to configure Heteroprio automatically. These heuristics are simple and can be evaluated without analyzing the complete graph of tasks. The preliminary results are promising [30] and an article is in preparation.

7.10 Distributed Raster Image Processing

Participants: Paul Godard, Vincent Loechner and Cédric Bastoul

We published a paper in IEEE TC [14] about the efficient out-of-core and out-of-place matrix rotation algorithm developed during Paul Godard's PhD thesis. As a followup to our collaboration with the Caldera company, that algorithm is now used in their latest software (CalderaRIP v14).

This work was also presented in the INRIA national newsboard (`https://www.inria.fr/en/paul-godard-driving-digital-printing-forward`).

We propose an efficient solution to perform in-memory or out-of-core rectangular matrix transposition and rotation by using an out-of-place strategy, reading a matrix from an input file and writing the transformed matrix to another (output) file. It relies on a block-matrix strategy with a parallel, cache-efficient intra-tile processing and an original in-memory file mapping, with an adequate tile scheduling to exploit efficiently the operating system page cache mechanism on a large variety of secondary storage technologies (HDD, SSD, RAID, NVMe). Taking advantage of the efficient random read access offered by flash memory drives while respecting their propensity for efficient sequential write access, our technique is about 10 times faster than a reference implementation on a RAID 0 SSD configuration and more than 5 times faster on a single SSD and NVMe configurations. In many cases, its performance gets close to the baseline performance of a file copy. Compared to other methods, our proposal offers good relative independence to the file system and disk low-level parameters, and good performance portability.

7.11 Optimizing Polyhedral Code

Participants: Harenome Razanajato, Vincent Loechner and Cédric Bastoul

Harenome Razanajato defended his PhD "*Polyhedral Code Generation: Reducing Overhead and Increasing Parallelism*" [22] on Sept. 24, 2020. He left the team to join Huawei Paris as consulting R&D polyhedral compiler engineer.

This thesis proposes new extensions to the code generation phase in polyhedral compilers. The main focus of recent work on polyhedral compilation is the optimizations leveraged by polyhedral transformations while state-of-the-art code generation algorithms are considered satisfactory. We show that state-of-the-art polyhedral code generation can still be further improved. We explain how splitting polyhedra can reduce the control overhead introduced by polyhedra scanning in the code generated by a polyhedral compiler. Synchronizations in parallel code can drastically impede a program's performance. We propose a method to detect and lift unnecessary synchronization barriers. Finally, we introduce pipelined multithreading, a transformation that introduces parallelism in a class of programs that was, until now, ignored by polyhedral parallelizers.

8 Partnerships and cooperations

8.1 European initiatives

8.1.1 Collaborations in European programs, except FP7 and H2020

Microcard: Numerical modeling of cardiac electrophysiology at the cellular scale

Participants Vincent Loechner, Cédric Bastoul, Bérenger Bramas.

We participate in the **EuroHPC Joint Undertaking** project Microcard. It aims to simulate cardiac electrophysiology using whole-heart models with sub-cellular resolution on future exascale supercomputers. It includes 10 European industrial and research partners, for a global budget of 5.7M euros. It is lead by Mark Potse (Univ. Bordeaux and INRIA Sud-Ouest) and it was accepted by the end of the year, including its co-funding by ANR. The project will fund a PhD student and a research engineer for 3 years in our team, starting in April 2021.

- Funding: EuroHPC
- Duration: 2021-2025
- Coordinator: University of Bordeaux
- Local coordinator: Vincent Loechner
- Partners: University of Bordeaux, University of Strasbourg, Simula Research Laboratory, Università degli studi di Pavia, Università della Svizzera italiana, Karlsruhe Institute of Technology, Zuse Institute Berlin, MEGWARE, NumeriCor, Orobix
- Website: http://www.microcard.eu/

8.1.2 Informal International Partners

- Benjamin Stamm and Muhammad Hassan: Université d'Aix-la-Chapelle RWTH, MATHCCES (Germany). An integral equation formulation of the N-body dielectricspheres problem.
- Michael Wilczek and Cristian Lalescu: Max Planck Institute for Dynamics and Self-Organization (Germany). Pseudospectral direct numerical simulations (DNS) of the incompressible Navier-Stokes equations.

8.2 National initiatives

8.2.1 ANR Vocal

Participants Arthur Charguéraud.

The goal of the ANR Vocal project is to develop the first formally verified library of efficient general-purpose data structures and algorithms. It targets the OCaml programming language, which allows for fairly efficient code and offers a simple programming model that eases reasoning about programs. The library will be readily available to implementers of safety-critical OCaml programs, such as Coq, Astrée, or Frama-C. It will provide the essential building blocks needed to significantly decrease the cost of developing safe software. The project intends to combine the strengths of three verification tools, namely Coq, Why3, and CFML. It will use Coq to obtain a common mathematical foundation for program specifications, as well as to verify purely functional components. It will use Why3 to verify a broad range of imperative programs with a high degree of proof automation. Finally, it will use CFML for formal reasoning about effectful higher-order functions and data structures making use of pointers and sharing.

- Funding: ANR
- Start: October 2015
- End: March 2021
- Coordinator: Jean-Christophe Filliâtre (LRI)
- Partners: team VALS (Université Paris Sud), team Gallium (Inria Paris), team DCS (Verimag), TrustInSoft, and OCamlPro.
- Website: https://vocal.lri.fr/

8.2.2 SPETABARU-H

Participants Bérenger Bramas, Vincent Loechner, Paul Cardosi.

The SPETABARU task-based runtime system is now being developed in CAMUS. This tool is the first runtime system built on the tasks and dependencies paradigm that supports speculative execution. It is at the same time a robust runtime system that could be used for high-performance applications, and the central component to perform research in parallelization, speculation and scheduling. The SPETABARU-H project aims at improving SPETABARU on several aspects:

- Implement a generic speculative execution model based on the team's research;
- Implement the mechanisms to make SPETABARU supporting GPUs (and heterogeneous computing nodes in general);
- Split the management of the workers and the management of the graph of tasks to allow multiple independent graphs to be used on a single node;
- Use SPETABARU in the Complexes++ application, which is a bio-physic software for protein simulation;
- Maintain and update the code to keep it modern and up to date.

Details

- Funding: Inria ADT
- Start: November 2019
- End: November 2021
- Coordinator: Bérenger Bramas
- Website: https://gitlab.inria.fr/bramas/spetabaru

9 Dissemination

9.1 Promoting scientific activities

9.1.1 Scientific events: organisation

General chair, scientific chair Philippe Clauss organized the Special Session on Compiler Architecture, Design and Optimization (CADO) of the 18th International Conference on High Performance Computing & Simulation (HPCS 2020), that should take place in February 2021 and online.

Philippe Clauss has organized the 10th edition of the International Workshop on Polyhedral Compilation Techniques, held in conjunction with HiPEAC 2020, January 22, 2020, Bologna, Italy.

Member of the conference program committees Arthur Charguéraud was part of the program committee for POPL 2020 (ACM Symposium on Principles of Programming Languages). He was also program comittee member for CADO 2020 (Special Session on Compiler Architecture, Design and Optimization at HPCS 2020).
Philippe Clauss has been part of the program committees of: CC 2020 (ACM SIGPLAN International Conference on Compiler Construction); ICPP 2020 (49th International Conference on Parallel Processing); IPDRM 2020 (Fourth Annual Workshop on Emerging Parallel and Distributed Runtime Systems and Middleware, held in conjunction with the International Conference for High Performance Computing, Networking, Storage and Analysis, SC 20).

9.1.2 Journal

Member of the editorial boards Since October 2001, Jens Gustedt has been the Editor-in-Chief of the journal Discrete Mathematics and Theoretical Computer Science (DMTCS).

Reviewer - reviewing activities Bérenger Bramas was reviewer for the Concurrency and Computation: Practice and Experience (Wiley) and Software: Practice and Experience (Wiley).

Arthur Charguéraud served as reviewer for the Journal of Functional Programming (JFP) and for the journal Proceedings of the ACM on Programming Languages (PACMPL).

Philippe Clauss was reviewer for the Journal of Experimental & Theoretical Artificial Intelligence, IEEE Transactions on Computers and the Journal of Computational Science, in 2020.

9.1.3 Scientific expertise

Since Nov. 2014, Jens Gustedt has been a member of the ISO/IEC working group ISO/IEC PL1/SC22/WG14 for the standardization of the C programming language and served as a co-editor of the standards document until March 2020. He participates actively in the clarification report processing, the planning of future versions of the standard and in subgroups that discuss the improvement of the C memory model and an improved compatibility and coordination between C and C++. He was one of the main forces behind the elaboration of C17, the new version of the C standard that has been published by ISO in 2018 and contributes to the future standard C23 in various ways. In particular he proposed the removal of the so-called K&R definitions, the reform of sign representation, maximum width integers, keywords, null pointer constants, timing interfaces, atomicity and synchronization, and function error conventions. Most of these are either integrated in the latest draft or have been adopted subject to reformulations and adaptations. The work has had the following impact in 2020.

- The proposed common C/C++ core [25] has found positive feedback, as well as from WG14 (C) and WG21 (C++). A joint study group between the two committees has been established to improve common language features and for the coordination between the two committees.
- The work on the memory model in general and pointer provenance in particular has been promoted by ballot of the national bodies to an official ISO work item. A first version [26] of a technical specification (ISO TS 6010) has met no opposition and is probably quite close to a final version.
- A new language feature for C, `defer`, has found positive feedback and has chances to find its way into the C standard [17, 24, 37].
- A better interface specification for unsequenced functions via attributes has been proposed to WG14 [23].

9.1.4 Research administration

Arthur Charguéraud was member of the hiring committee for Inria Nancy–Grand-Est CRCN and ISFP positions, in June 2020.

Jens Gustedt is the head of the ICPS team for the ICube lab. He is a member of the executive board of directors of the lab, responsible for the IT and CS policy and for the coordination between the lab and the Inria center. He is also a member of the local recruitment committee for PhD students and postdocs of Inria Center Nancy — Grand Est. In June 2020, he was member of the hiring committee for Inria Grenoble–Rhone-Alpes CRCN and ISFP positions.

Cédric Bastoul, Philippe Clauss and Vincent Loechner are members of the *Comité d'Experts (section 27, informatique)* of the *Université de Strasbourg*, providing their scientific and teaching expertise to the university and to the academy. In particular, this committee is involved in the recruitment of researchers and teachers in computer science. Philippe Clauss has been the Vice President of the committee since April 2019.

Philippe Clauss and Cédric Bastoul are members of the *Collegium Sciences* of the University of Strasbourg, which is a group of representative scientists providing advice regarding the funding of projects.

Philippe Clauss is a member of the *Bureau du Comité des Projets* of the Inria Center Nancy — Grand Est. This group of scientists provides scientific expertise to the Director of the Center.

Since September 2020, Philippe Clauss is in charge of the Master of Computer Science of the University of Strasbourg.

9.2 Teaching - Supervision - Juries

9.2.1 Teaching

- Master: Bérenger Bramas, Compilation and Performance, 24h, M2, Université de Strasbourg, France
- Master: Bérenger Bramas, Compilation, 24h, M1, Université de Strasbourg, France
- Licence: Philippe Clauss, Computer architecture, 18h, L2, Université de Strasbourg, France
- Licence: Philippe Clauss, Bases of computer architecture, 22h, L1, Université de Strasbourg, France
- Master: Philippe Clauss, Compilation, 84h, M1, Université de Strasbourg, France
- Master: Philippe Clauss, Real-time programming and system, 37h, M1, Université de Strasbourg, France
- Master: Philippe Clauss, Code optimization and transformation, 31h, M1, Université de Strasbourg, France
- Licence (Math-Info): Alain Ketterlin, Architecture des systèmes d'exploitation, L3, 38h, Université de Strasbourg, France
- Licence (Math-Info): Alain Ketterlin, Programmation système, L2, 60h, Université de Strasbourg, France
- Master (Informatique): Alain Ketterlin, Preuves assistées par ordinateur, 18h, Université de Strasbourg, France
- Master (Informatique): Alain Ketterlin, Compilation, 84h, Université de Strasbourg, France
- Licence: Vincent Loechner, head of the professional licence degree, Université de Strasbourg, France
- Licence: Vincent Loechner, Algorithmics and programmation, 64h, L1, Université de Strasbourg, France
- Licence: Vincent Loechner, Parallel programming, 32h, L3, Université de Strasbourg, France

- Licence: Vincent Loechner, System administration, 40h, Licence Pro, Université de Strasbourg, France
- Master: Vincent Loechner, Real-time programming and system, 12h, M1, Université de Strasbourg, France
- Eng. School: Vincent Loechner, Parallel Computation, 20h, Telecom Physique Strasbourg - 3rd year, Université de Strasbourg, France

9.2.2 Supervision

- PhD: Harenome Ranaivoarivony-Razanajato, Polyhedral Code Generation: Reducing Overhead and Increasing Parallelism, co-advised by Cédric Bastoul and Vincent Loechner, defended on September 24th, 2020.
- PhD in progress: Salwa Kobeissi, *Dynamic parallelization of recursive functions by transformation into loops*, since Sept. 2017, Philippe Clauss.
- PhD in progress: Clément Flint, *Efficient data compression for high-performance PDE solvers*, since Nov. 2020, Bérenger Bramas.

9.2.3 Juries

- Philippe Clauss was a reviewer for the PhD thesis of Raquel Lazcano, hold on Nov. 13, 2020, at the Universidad Politécnica de Madrid, Spain.
- Vincent Loechner participated as examiner in the PhD jury of Toufik Baroudi, hold on Jul. 9, 2020, at the University of Batna, Algeria.

9.3 Popularization

- Arthur Charguéraud is a co-organizer of the **Concours Castor informatique**. The purpose of the Concours Castor is to introduce pupils, from *CM1* to *Terminale*, to computer sciences. 525,000 teenagers played with the interactive exercises in November and December 2020, despite the partial lockdown.
- Jens Gustedt has given an hour-long interview about Modern C for IEEE's "Software Engineering Radio".
- Vincent Loechner organizes each year a *Google Hash Code* programming contest hub at the University of Strasbourg.

9.3.1 Internal or external Inria responsibilities

- Bérenger Bramas is in charge of the scientific computing group (*axe transverse calcul scientifique*) of the ICube laboratory and initiated the ICube software collection.

9.3.2 Articles and contents

- Jens Gustedt is 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.

9.3.3 Education

- Arthur Charguéraud wrote an *all-in-Coq book* on the **Foundations of Separation Logic**. All the material is formalized in Coq, and is meant to be used either in class or for self-teaching. It should be soon released as Volume 6 of the Software Foundations series.

10 Scientific production

10.1 Major publications

[1] U. A. Acar, V. Aksenov, A. Charguéraud and M. Rainey. 'Provably and Practically Efficient Granularity Control'. In: *PPoPP 2019 - Principles and Practice of Parallel Programming*. Washington DC, United States, Feb. 2019. DOI: 10.1145/3293883.3295725. URL: https://hal.inria.fr/hal-01973285.

[2] P. Clauss. 'Counting Solutions to Linear and Nonlinear Constraints Through Ehrhart Polynomials: Applications to Analyze and Transform Scientific Programs'. In: *ICS, International Conference on Supercomputing*. ACM International Conference on Supercomputing 25th Anniversary Volume. Munich, Germany, 2014. DOI: 10.1145/2591635.2667172. URL: https://hal.inria.fr/hal-01100306.

[3] P. Clauss, F. J. Fernández, D. Garbervetsky and 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* 17.8 (Aug. 2009), pp. 983–996. DOI: 10.1109/TVLSI.2008.2002049. URL: https://hal.inria.fr/inria-00504617.

[4] A. Jimborean, P. Clauss, J.-F. Dollinger, V. Loechner and M. Juan Manuel. 'Dynamic and Speculative Polyhedral Parallelization Using Compiler-Generated Skeletons'. In: *International Journal of Parallel Programming* 42.4 (Aug. 2014), pp. 529–545. URL: https://hal.inria.fr/hal-01003744.

[5] A. Ketterlin and 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*. Proceedings of the 6th annual IEEE/ACM international symposium on Code generation and optimization. Boston, United States: ACM, Apr. 2008, pp. 94–103. DOI: 10.1145/1356058.1356071. URL: https://hal.inria.fr/inria-00504597.

[6] A. Ketterlin and P. Clauss. 'Profiling Data-Dependence to Assist Parallelization: Framework, Scope, and Optimization'. In: *MICRO-45, The 45th Annual IEEE/ACM International Symposium on Microarchitecture*. Vancouver, Canada, Dec. 2012. URL: https://hal.inria.fr/hal-00780782.

[7] B. Pradelle, A. Ketterlin and P. Clauss. 'Polyhedral parallelization of binary code'. In: *ACM Transactions on Architecture and Code Optimization*. Special issue on high-performance and embedded architectures and compilers 8.4 (Jan. 2012), 39:1–39:21. DOI: 10.1145/2086696.2086718. URL: https://hal.inria.fr/hal-00664370.

[8] A. Sukumaran-Rajam and P. Clauss. 'The Polyhedral Model of Nonlinear Loops'. In: *ACM Transactions on Architecture and Code Optimization* 12.4 (Jan. 2016). DOI: 10.1145/2838734. URL: https://hal.inria.fr/hal-01244464.

10.2 Publications of the year

International journals

[9] B. Bramas. 'TBFMM: A C++ generic and parallel fast multipole method library'. In: *Journal of Open Source Software* 5.56 (3rd Dec. 2020), p. 2444. DOI: 10.21105/joss.02444. URL: https://hal.inria.fr/hal-02550688.

[10] B. Bramas, M. Hassan and B. Stamm. 'An Integral Equation Formulation of the N-Body Dielectric Spheres Problem. Part II: Complexity Analysis'. In: *ESAIM: Mathematical Modelling and Numerical Analysis* (31st July 2020). DOI: 10.1051/m2an/2020055. URL: https://hal.inria.fr/hal-02913501.

[11] B. Bramas, P. Helluy, L. Mendoza and B. Weber. 'Optimization of a discontinuous Galerkin solver with OpenCL and StarPU'. In: *International Journal on Finite Volumes* 15.1 (29th Jan. 2020), pp. 1–19. URL: https://hal.archives-ouvertes.fr/hal-01942863.

[12] B. Bramas and A. Ketterlin. 'Improving parallel executions by increasing task granularity in task-based runtime systems using acyclic DAG clustering'. In: *PeerJ Computer Science* (13th Jan. 2020). DOI: 10.7717/peerj-cs.247. URL: https://hal.inria.fr/hal-02436826.

[13] A. Charguéraud. 'Separation Logic for Sequential Programs'. In: *Proceedings of the ACM on Programming Languages* 4 (2nd Aug. 2020). DOI: 10.1145/3408998. URL: https://hal.inria.fr/hal-03108936.

[14] P. Godard, V. Loechner and C. Bastoul. 'Efficient Out-of-core and Out-of-place Rectangular Matrix Transposition and Rotation'. In: *IEEE Transactions on Computers* (2020). DOI: 10.1109/TC.2020.3030592. URL: https://hal.inria.fr/hal-02960539.

[15] C. Ramon-Cortes, R. Amela, J. Ejarque, P. Clauss and R. M. Badia. 'AutoParallel: Automatic parallelisation and distributed execution of affine loop nests in Python'. In: *International Journal of High Performance Computing Applications* 34.6 (14th July 2020), pp. 1–14. DOI: 10.1177/1094342020937050. URL: https://hal.inria.fr/hal-02971480.

International peer-reviewed conferences

[16] R. Affeldt, C. Cohen, M. Kerjean, A. Mahboubi, D. Rouhling and K. Sakaguchi. 'Competing inheritance paths in dependent type theory: a case study in functional analysis'. In: IJCAR 2020 - International Joint Conference on Automated Reasoning. Paris, France, 29th June 2020, pp. 1–19. URL: https://hal.inria.fr/hal-02463336.

[17] J. Gustedt and R. C. Seacord. 'C language mechanism for error handling and deferred cleanup'. In: The 36th ACM/SIGAPP Symposium on Applied Computing (SAC'21). virtuelle, South Korea, 22nd Mar. 2021. DOI: 10.1145/3412841.3442116. URL: https://hal.inria.fr/hal-03059076.

[18] R. Lazcano, D. Madroñal, E. Juarez and P. Clauss. 'Runtime Multi-versioning and Specialization inside a Memoized Speculative Loop Optimizer'. In: CC 2020 - 29th International Conference on Compiler Construction. San Diego, United States, 22nd Feb. 2020. DOI: 10.1145/3377555.3377886. URL: https://hal.inria.fr/hal-02457425.

National peer-reviewed Conferences

[19] G. Kusoglu, B. Bramas and S. Genaud. 'Automatic task-based parallelization of C++ applications by source-to-source transformations'. In: Compas 2020 - Conférence francophone en informatique. Lyon, France, 30th June 2020. URL: https://hal.inria.fr/hal-02867413.

Conferences without proceedings

[20] T. Baroudi, V. Loechner and R. Seghir. 'Static versus Dynamic Memory Allocation: a Comparison for Linear Algebra Kernels'. In: IMPACT 2020, in conjunction with HiPEAC 2020. Bologna, Italy, 22nd Jan. 2020. URL: https://hal.inria.fr/hal-02456533.

[21] B. Meister and P. Clauss. 'Uniform Random Sampling in Polyhedra'. In: IMPACT 2020 - 10th International Workshop on Polyhedral Compilation Techniques. Bologna, Italy, 20th Jan. 2020. URL: https://hal.inria.fr/hal-02425752.

[22] H. Razanajato, C. Bastoul and V. Loechner. 'Pipelined Multithreading Generation in a Polyhedral Compiler'. In: IMPACT 2020, in conjunction with HiPEAC 2020. Bologna, Italy, 22nd Jan. 2020. URL: https://hal.inria.fr/hal-02456521.

Scientific books

[23] É. Alepins and J. Gustedt. *Unsequenced functions*. http://www.open-std.org/jtc1/sc22/wg14/www/docs/n2539.pdf, 12th July 2020, p. 11. URL: https://hal.inria.fr/hal-02952723.

[24] A. Ballman, A. Gilding, J. Gustedt, T. Scogland, R. C. Seacord, M. Uecker and F. Wiedijk. *Defer Mechanism for {C}*. http://www.open-std.org/jtc1/sc22/wg14/www/docs/n2542.pdf, 29th Sept. 2020, p. 57. URL: https://hal.inria.fr/hal-02953399.

[25] J. Gustedt. *A Common C/C++ Core Specification*. http://www.open-std.org/jtc1/sc22/wg14/www/docs/n2522.pdf, 10th May 2020, p. 627. URL: https://hal.inria.fr/hal-02952700.

[26] J. Gustedt, P. Sewell, K. Memarian, V. B. F. Gomes and M. Uecker. *A Provenance-aware Memory Object Model for C*. http://www.open-std.org/jtc1/sc22/wg14/www/docs/n2577.pdf, 4th Oct. 2020, p. 89. URL: https://hal.inria.fr/hal-02957464.

Scientific book chapters

[27] S. Kobeissi, A. Ketterlin and P. Clauss. 'Rec2Poly: Converting Recursions to Polyhedral Optimized Loops Using an Inspector-Executor Strategy'. In: *SAMOS 2020: Embedded Computer Systems: Architectures, Modeling, and Simulation*. 7th Oct. 2020, pp. 96–109. DOI: 10.1007/978-3-030-60939-9_7. URL: https://hal.inria.fr/hal-02971434.

Doctoral dissertations and habilitation theses

[28] H. Razanajato. 'Polyhedral Code Generation: Reducing Overhead and Increasing Parallelism'. Université de Strasbourg (Unistra), FRA., 24th Sept. 2020. URL: https://hal.inria.fr/tel-03125702.

Reports & preprints

[29] B. Bramas and Q. Bramas. *On the improvement of the in-place merge algorithm parallelization*. 26th May 2020. URL: https://hal.inria.fr/hal-02613668.

[30] C. Flint and B. Bramas. *Finding new heuristics for automated task prioritizing in heterogeneous computing*. 6th Nov. 2020. URL: https://hal.inria.fr/hal-02993015.

[31] J. Gustedt. *C source-to-source compiler enhancement from within*. INRIA, 10th Nov. 2020. URL: https://hal.inria.fr/hal-02998412.

[32] J. Gustedt. *Function literals and value closures: proposal for C23*. ISO JCT1/SC22/WG14, 12th Jan. 2021, p. 25. URL: https://hal.inria.fr/hal-03106767.

[33] J. Gustedt. *Improve type generic programming: proposal for C23*. ISO JCT1/SC22/WG14, 12th Jan. 2021, p. 54. URL: https://hal.inria.fr/hal-03106758.

[34] J. Gustedt. *Lvalue closures: proposal for C23*. ISO JCT1/SC22/WG14, 12th Jan. 2021, p. 8. URL: https://hal.inria.fr/hal-03106930.

[35] J. Gustedt. *Type inference for variable definitions and function returns: proposal for C23*. ISO JCT1/SC22/WG14, 12th Jan. 2021. URL: https://hal.inria.fr/hal-03106763.

[36] J. Gustedt. *Type-generic lambdas: proposal for C23*. ISO JCT1/SC22/WG14, 12th Jan. 2021, p. 10. URL: https://hal.inria.fr/hal-03106919.

[37] J. Gustedt and R. C. Seacord. *Deferred cleanup and error handling in C*. Inria Nancy - Grand Est, Dec. 2020, p. 19. URL: https://hal.inria.fr/hal-03090771.

10.3 Other

Softwares

[38] [SW] B. Bramas, *Farm-SVE: A scalar C++ implementation of the ARM® Scalable Vector Extension (SVE)* version 0.1, 24th July 2020. HAL: ⟨hal-02906179⟩, URL: https://hal.inria.fr/hal-02906179, VCS: https://gitlab.inria.fr/bramas/farm-sve, SWHID: ⟨swh:1:dir:1d1509453950e3569b5f55a6dece1da5eeab9572;origin=https://hal.archives-ouvertes.fr/hal-02906179;visit=swh:1:snp:3506821b2a12082a9bcb490b05b446053c9a940d;anchor=swh:1:rev:7f55f2b6c15d89f826730cf89f500d340d4aefa8;path=/⟩.

10.4 Cited publications

[39] 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. URL: https://hal.archives-ouvertes.fr/ccsd-00017260.

[40] M. Hall, D. Padua and K. Pingali. ‘Compiler research: the next 50 years’. In: *Commun. ACM* 52.2 (2009), pp. 60–67. URL: http://doi.acm.org/10.1145/1461928.1461946.

[41] A. Hobor, A. W. Appel and F. Z. Nardelli. ‘Oracle Semantics for Concurrent Separation Logic’. In: *ESOP*. 2008, pp. 353–367.