RESEARCH CENTRE

Inria Nancy - Grand Est Center

IN PARTNERSHIP WITH:

Université de Strasbourg

2022

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 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 STRASBOURG, Professor, HDR]
- Stéphane Genaud [UNIV STRASBOURG, Associate Professor, HDR]
- Alain Ketterlin [UNIV STRASBOURG, Associate Professor]
- Vincent Loechner [UNIV STRASBOURG, Associate Professor]
- Eric Violard [UNIV STRASBOURG, Associate Professor, HDR]

Post-Doctoral Fellow

- Jean Etienne Ndamlabin Mboula [INRIA]

PhD Students

- Guillaume Bertholon [ENS Paris, from Sep 2022]
- Clément Flint [UNIV STRASBOURG]
- Garip Kusoglu [INRIA, until Jun 2022]
- Clément Rossetti [UNIV STRASBOURG, from Oct 2022]
- Anastasios Souris [INRIA, from Jul 2022]
- Hayfa Tayeb [INRIA]
- Arun Thangamani [UNIV STRASBOURG]

Technical Staff

- Begatim Bytyqi [Inria, Engineer, until Aug 2022]
- Raphael Colin [UNIV STRASBOURG, Engineer, from Oct 2022]
- Tiago Trevisan Jost [UNIV STRASBOURG, Engineer]

Interns and Apprentices

- Guillaume Bertholon [ENS Paris, Intern, from Feb 2022 until Jul 2022]
- Maxime Drouhin [INRIA, Intern, from Jun 2022 until Aug 2022]
- Tom Hammer [UNIV STRASBOURG, Apprentice]
- Louis Riboulet [ENS Lyon, Intern, from Jun 2022 until Jul 2022]
- Michel Tching [UNIV STRASBOURG, Intern, from Jun 2022 until Aug 2022]

Administrative Assistant

- Ouiza Herbi [INRIA]

Visiting Scientist

- Rachid Seghir [Université de Batna, from Nov 2022 until Nov 2022]

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 [49]. 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 Multicore Processors

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 the generation of speculative parallel code (issue 3). We can add to our behavior modeling objectives, the behavior monitoring, or profiling, of a program version. Indeed, the complexity

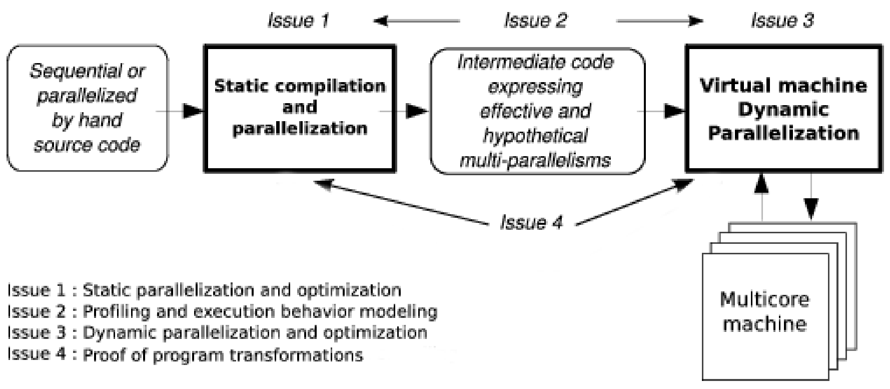
Issue 1
Issue 2
Issue 3
Sequential or parallelized by hand source code
Static compilation and parallelization
Intermediate code expressing effective and hypothetical multi-parallelisms
Virtual machine Dynamic Parallelization
Issue 4
Multicore machine
Issue 1 : Static parallelization and optimization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

Figure 1: Steps for automatic parallelization on multicore architectures.

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 a 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. 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, Arthur Charguéraud, Bérenger Bramas

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 [47]. 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

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, 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, Jens Gustedt

Producing high-performance code is a challenging task. Producing formally verified code is a challenging task. Our goal is to develop a framework for producing code that is both high-performance *and* formally verified. Our approach is to leverage source-to-source transformations, interactively guided by the expert programmer. To begin with, an unoptimized implementation of an algorithm can be verified using interactive proofs, e.g. in Separation Logic. Then, by applying series of transformations, we can turn an unoptimized implementation of an algorithm into a high-performance one. These transformations modify not only the code, but also the logical invariants that accompany it. By doing so, we aim to automatically or semi-automatically derive a formal proof that the optimized code satisfies the same high-level specifications as the unoptimized implementation.

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 Social and environmental responsibility

5.1 Footprint of research activities

We have largely decreased the number of physical meetings, opting for video-conference meetings when possible. We have also favored travel by train rather than by plane in the past year.

5.2 Impact of research results

Regarding the significant impact on energy consumption and related carbon emission of numerical applications, and particularly of high performance computing, every research project of the team will include from now on the important goal of energy efficiency for the generated codes. The optimizing mechanisms that we propose in our research will be evaluated with energy and time performance, both considered at the same priority level.

6 Highlights of the year

Our new team project proposal, which has the same name as the previous team project, i.e. CAMUS for *Compilation pour les Architectures MUlti-processeurs et multi-coeurS*, has been accepted by the project committee of Nancy Grand Est. The creation is expected to be made official in the first months of 2023.

Arthur Charguéraud has been awarded a grant from ANR for the OptiTrust project, which aims to develop a framework for programmer-guided, source-to-source transformations with formal guarantees. This 4-year project started in Oct. 2022.

7 New software and platforms

7.1 New software

7.1.1 TRAHRHE

Name: Trahrhe expressions and applications in loop optimization

Keywords: Polyhedral compilation, Code optimisation, Source-to-source compiler

Functional Description: This software includes a mathematic kernel for computing Trahthe expressions related to iteration domains, as well as extensions implementing source-to-source transformations of loops for applying optimizations based on Trahrhe expressions.

News of the Year: Two extensions have been implemented, that are new alternatives for the runtime computation of trahrhe expressions values, and that may significantly improve the time performance when using the generated C header files: (1) when activated, adds a precision correction mechanism. Useful when the complexity of the trahrhe expressions yields annoying precision issues. Moreover, it makes the use of MPC useless and thus yields significantly better time performance than when using MPC. (2) when activated, avoids the computation of the trahrhe expressions by solving polynomial equations, and uses instead a dichotomic search mechanism to get the current integer values of the trahrhe expression, through invocations to the ranking polynomials. Another improved dichotomic search may also be activated, that uses the history of previous invocations to accelerate even more the search.

URL: `https://webpages.gitlabpages.inria.fr/trahrhe`

Contact: Philippe Clauss

7.1.2 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.

News of the Year: Addition of lemmas and tactics for proving characteristic formulae sound with respest to the semantics in a foundational way.

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

Contact: Arthur Charguéraud

Participants: Arthur Charguéraud, Armaël Guéneau, François Pottier

7.1.3 openCARP

Name: Cardiac Electrophysiology Simulator

Keyword: Cardiac Electrophysiology

Functional Description: openCARP is an open cardiac electrophysiology simulator for in-silico experiments. Its source code is public and the software is freely available for academic purposes. openCARP is easy to use and offers single cell as well as multiscale simulations from ion channel to organ level. Additionally, openCARP includes a wide variety of functions for pre- and post-processing of data as well as visualization.

URL: https://opencarp.org/

Contact: Vincent Loechner

Participants: Arun Thangamani, Stephane Genaud, Bérenger Bramas, Tiago Trevisan Jost, Raphael Colin

Partner: Karlsruhe Institute of Technology

7.1.4 SPECX

Name: SPEculative eXecution task-based runtime system

Keywords: HPC, Parallelization, Task-based algorithm

Functional Description: Specx (previously 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.

News of the Year: During 2022, Specx has been improved to support heterogeneous and distributed computing nodes. It is now possible to create CPU/GPU tasks and Specx will manage the data transfers between the host and devices. In distributed memory, the communications are now part of the task graph. All of these novelties make Specx a good candidate to develop modern HPC applications in C++.

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

Contact: Bérenger Bramas

8 New results

8.1 Static Parallelization and Optimization

8.1.1 Improvements of the C programming language

Participants: Jens Gustedt.

For the upcoming version of the C standard, C23, we contributed with a large number of proposals that are now integrated in the draft that has been presented by ISO JTC1/SC22 to the national bodies (NB). The final adjustment of the content according to more than 300 NB comments will be made end of Jan, 2023.

This year's contributions that were integrated to C23 are as follows: [19] [22] [23] [24] [25] [26] [30] [41] [32] [33] [34] [35] [36] [37] [42]. Over all, out of the about 50 points that are listed in the change history of C23 (Annex M) as major additions, our contributions include:

- adding new keywords such as `bool`, `static_assert`, `true`, `false`, `thread_local` and others, and allowed implementations to provide macros for the older spelling with a leading underscore followed by a capital letter as well as defining old and new keywords as macros to enable transition of programs easily;
- removing integer width constraints and obsolete sign representations (so-called "1's complement" and "sign-magnitude");
- removing support for function definitions with identifier lists;
- harmonization with ISO/IEC 9945 (POSIX):
  - integration of functions: `gmtime_r`, `localtime_r`;
- adding version test macros to library headers that contained changes to aid in upgrading and portability to be used alongside the `__STDC_VERSION__` macro;
- including the attributes:
  - `[[reproducible]]`, for marking function types for which inputs may always produce predictable output if given the same input (e.g., cached data) but for which the order of such calls still matter;
  - `[[unsequenced]]`, for marking function types which always produce predictable output and have no dependencies upon other data (and other relevant caveats);
- allowing compound literals to may also include storage-class specifiers as part of the type to change their lifetime (and possibly turn it into a constant expression);
- adding the `constexpr` specifier for object definitions and improved what is recognized as a constant expression in conjunction with the `constexpr` storage-class specifier;
- adding a `nullptr` constant and a `nullptr_t` type with a well-defined underlying representation identical to a pointer to `void`;
- allowing Unicode identifiers in syntax following Unicode Standard Annex, UAX #31;
- allowing certain type definitions (i.e., exact-width integer types such as `int128_t`), bit-precise integer types, and extended integer types may exceed the normal boundaries of `intmax_t` and `uintmax_t` for signed and unsigned integer types, respectively;
- mandating support for `call_once`;
- enhancing the `auto` type specifier for single object definitions using type inference;
- adding support for additional time bases, as well as `timespec_getres`, in `<time.h>`;
- adding an `unreachable` feature which has undefined behavior if reached during program execution.

Additionally, the following proposals have been applied that change aspects that are less visible to the end-user of the language:

- Disambiguating the storage class of some compound literals;
- Types and sizes;
- Indeterminate Values and Trap Representations;
- Removing `ATOMIC_VAR_INIT`;
- Properly defining blocks as part of the grammar.

Other proposals have been discused have been seen favorable for integration into a future version of the standard: [29] [27] [28] [40], see also [28] [31] [38] [39].

In addition to the C standard we continued our work for the new technical specification TS 6010 [18] for a sound and verifiable memory model that is based on provenance, that is on the unique attribution of a storage instance to any valid pointer value.

8.1.2 Ionic Models Code Generation for Heterogeneous Architectures

Participants: Arun Thangamani, Tiago Trevisan Jost, Raphaël Colin, Vincent Loechner, Bérenger Bramas, Stéphane Genaud.

We participate in the research and development of a cardiac electrophysiology simulator in the context of the MICROCARD European project. The CAMUS team provides their optimizing compiler expertise to build a bridge from a high-level language convenient for ionic model experts (called EasyML) to a code that will run on future exascale supercomputers. We aim to generate multiple parallel versions of the ionic simulation to exploit the various parallel computing units that are available in the target architecture nodes (SIMD, multicore, GPU, etc.).

The frontend that we contributed to in 2022 is based on the MLIR extensible compiler. We developed a Python script generating MLIR code from an EasyML ionic model description and integrated it in the openCARP project (7.1.3). We have achieved the automatic vectorization of the ionic code and this work is to be published and presented early next year at the CGO '23 conference. The follow-up work concerns the generation of code for GPUs and for multiple heterogeneous architectures, in coordination with the developments carried out by the STORM team (Inria Bordeaux) on the runtime distributed environment (StarPU).

8.1.3 A Polyhedral Programmable Scheduler

Participants: Tom Hammer, Vincent Loechner.

Scheduling is the central operation in the polyhedral compilation chain: finding the best execution order of loop iterations for optimizing the resulting code. Scheduling algorithms rely on the fundamental Farkas lemma to iteratively compute the schedule matrices associated to each statement of the input code.

The goal of this work is to allow polyhedral compilation experts to produce highly customizable scheduling algorithms in an efficient manner, without having to develop from scratch a new Farkas lemma-based solver and the whole polyhedral compilation toolchain. It will be included in the PeriScop suite (OpenScop, CLooG, PIPLib, etc.). We started to design a specific mini-language (DSL) that allows the user to specify and prioritize its objectives.

This is Tom Hammer's internship main work, in connection with the MICROCARD project that will benefit from the existence of such a scheduling algorithm generator. We wrote a first paper that will be presented to the community during the IMPACT '23 workshop collocated with the HiPEAC '23 conference.

8.1.4 Superloop Scheduling

Participants: Vincent Loechner, Alain Ketterlin, Cedric Bastoul.

This work predates Cédric Bastoul's leave (he is currently detached from the team and working at Qualcomm).

In polyhedral compilation, discovering the best polyhedral schedules remains a grand challenge reinforced by the need to build *affine* functions. Automatic techniques based on solving systems of affine constraints and/or composing affine scheduling primitives are limited, either by the affine modeling or by their primitive set.

We propose a radically new approach to affine scheduling construction called superloop scheduling. The main idea is to leverage basic-block-level instruction reordering compilation techniques, and then to agglomerate statement instances into "super" loops offered by modern trace analysis tools. We will present this work at the IMPACT '23 workshop.

8.1.5 Automatic Task-Based Parallelization using Source to Source Transformations

Participants: Garip Kusoglu, Bérenger Bramas, Stéphane Genaud.

We 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. To do so, we 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. In 2021, we improved our compiler with a new method to automatically build performance models able to predict the execution time of the tasks. Early results [50] have allowed the activation of only those tasks with sufficient granularity. This work has been however put on stand-by early 2022 after the resignation of the first year student who worked on it. A summer internship has explored the opportunity to transition the transformation framework used from Clang to OptiTrust (developed in the team) and a PostDoc will resume the work in 2023.

8.1.6 Parallel kinetic scheme in complex toroidal geometry

Participants: Bérenger Bramas.

Bérenger Bramas collaborated with the TONUS team on a solver for the conservative transport equation with variable coefficients in complex toroidal geometries. He applied several optimizations at algorithm or implementation level to obtain high performance [45].

8.1.7 Automatic vectorization of static computational kernels

Participants: Hayfa Tayeb, Bérenger Bramas.

Compilers can automatically vectorize codes if there are zero to few data transformations to perform. However, when memory accesses are not contiguous, non linear or even chaotic, compilers usually fail. This is why we proposed a new method where we transform the dependency graph into a graph made of vectorial instructions. Our different layers aim at minimizing the number of instructions, especially data transformation instructions, that are needed to get a vectorized code. To do so we have developed the Autovesk compiler, which converts scalar C++ code into a vectorized C++ equivalent. We tested the performance of our approach by vectorizing different types of kernels, and we showed that we outperform the GNU C++ compiler by a speedup factor of 4 on average and up to 11. This work is described in a preprint [44].

8.1.8 Algebraic Loop Tiling

Participants: Clément Rossetti, Philippe Clauss.

We are currently developing a new approach for loop tiling, where tiles are no longer characterized by the sizes of their edges, but by their volumes, i.e., the number of embedded iterations. Algebraic tiling is particularly dedicated to parallel loops, where load balancing among the parallel threads is crucial for reaching high runtime performance.

Rectangular tiles of quasi-equal volumes, but of different shapes, are obtained thanks to mathematical computations based on the inversion of Ehrhart polynomials. The Trahrhe software (see Section 7.1.1) is dedicated to such computations, and to the automatic generation of header files in C, that can be used for the runtime computation of the algebraic tile bounds.

Clément Rossetti has started his related PhD work in October 2022. Integration of algebraic loop tiling into the source-to-source polyhedral compiler Pluto has been initiated by Clément. Any Pluto user will get access to automatic algebraic tiling thanks to the new Pluto flag `-atiling`.

A paper presenting for the first time the algebraic loop tiling technique will be presented at IMPACT 2023 [12].

8.2 Profiling and Execution Behavior Modeling

8.2.1 An efficient particle tracking algorithm for large-scale parallel pseudo-spectral simulations of turbulence

Participants: Bérenger Bramas.

The statistical studies of particles moving in a fluid is a widely used technic to study turbulences. We used this method to perform direct numerical simulations (DNS) of homogeneous isotropic turbulence and investigated the dynamics of different particle shapes at different scales in turbulence using a filtering approach. Bérenger Bramas collaborated on this project to design a particle interaction system able to work efficiently on large supercomputers [10]. The work is implemented in a software called TurTLE.

8.2.2 Multi-GPU parallelization of the Lattice Boltzmann Method

Participants: Clément Flint, Philippe Helluy, Bérenger Bramas, Stéphane Genaud.

In the context of the ITI IRMIA++, we work on the parallelization of the Lattice Boltzmann Methodology (LBM), a general framework for constructing efficient numerical fluid simulations. This method is well-suited to GPU computations. However, a number of GPU implementations are limited in the simulation scale because the data size that has to be handled is constrained by the GPU memory. We have proposed a parallelization based on the division of the lattice structure into multiple subsets that can be executed individually on corresponding data sets. Our implementation enables the distribution of the computations on multiple CPUs and GPUs using StarPU, hence relaxing the constraint on GPU memory. This work has been published in [15].

8.3 Dynamic Parallelization and Optimization

8.3.1 Ordered Read-Write Locks for applications

Participants: Jens Gustedt.

In the context of a project with ICube's Mécaflu team we use our implementation of Ordered Read-Write Locks (ORWL) [4] to integrate parallelism into an already existing Fortran application that computes floods in the geographical region that is subject to the study. That parallelization has been achieved by using ORWL on a process level. It connects several executables that run the legacy code in separate processes and is able to launch such a stiched execution in shared memory and distributed contexts. One of the steps to this goal has been to design a specific decomposition of geological data [9].

8.3.2 Scheduling multiple task-based applications on distributed heterogeneous computing nodes

Participants: Jean Etienne Ndamlabin, Bérenger Bramas.

The size, complexity and cost of supercomputers continue to grow making any waste more critical than in the past. Consequently, we need methods to reduce the waste coming from the users' choices, badly optimized applications or heterogeneous workloads during executions. In this context, we started activities on the scheduling of several task-based applications on given hardware resources. Specifically, we created load balancing heuristics to distribute the task-graph over the processing units. We are creating a new scheduler in StarPU to validate our approach. We will perform performance evaluation in the next stage, first using a simulator (SimdGrid) and then on real hardware.

8.3.3 Multreeprio: a scheduler for task-based applications on heterogeneous hardware

Participants: Hayfa Tayeb, Bérenger Bramas.

In the context of the TEXTAROSSA project, we investigated how the Heteroprio scheduler can be improved. We created a new scheduler called Multreeprio where the tasks can have multiple priorities, instead of having priorities for each type of tasks. We created different heuristics to assign priorities based on the workload, the type of processing units or even the locality. Preliminary results have been presented at COMPAS'22 [17].

8.3.4 Automatic Configuration of the Heteroprio Scheduler

Participants: Clément Flint, Ludovic Paillat, Bérenger Bramas.

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. This requires a significant programming effort and the given configuration might be inefficient 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 results have been published [8].

8.4 Proof of Program Transformations

8.4.1 Survey of Separation Logic for Sequential Programs

Participants: Arthur Charguéraud.

Arthur Charguéraud has written the manuscript for his Habilitation (HDR), titled *A Modern Eye on Separation Logic for Sequential Programs*, to be defended in Febuary 2023. This manuscript provides a survey of Separation Logic for sequential programs, in the line of the tutorial article published at ICFP'20 [48]. It also gives a complete introduction of the working of the CFML tool (7.1.2), as well as its extensions for reasoning about time bounds (PhD thesis of Armaël Guéneau) and reasoning about space bounds (PhD thesis of Alexandre Moine).

8.4.2 Formal Verification of a Transient Data Structure

Participants: Arthur Charguéraud.

Prior to his PhD co-advised by Arthur Charguéraud and François Pottier (Inria Paris, team Cambium), Alexandre Moine worked during his M2 internship on the specification and verification of the functional correctness and time complexity of a *transient* stack, using CFML (7.1.2). A transient data structure is a package of an ephemeral data structure, a persistent data structure, and fast conversions between them. The transient stack studied is a scaled-down version of Sek, a general-purpose sequence data structure implemented in OCaml. Internally, it relies on fixed-capacity arrays, or chunks, which can be shared between several ephemeral and persistent stacks. Dynamic tests are used to determine whether a chunk can be updated in place or must be copied: a chunk can be updated if it is uniquely owned or if the update is monotonic. There are very few examples of verified transient data structures in the literature, let alone examples of transient data structures whose correctness and time complexity are verified. This result has been published at the conference CPP'22 [11] and has received one of the three Distinguished Paper Awards.

8.4.3 Formal Proof of Space Bounds for Garbage-Collected Programs

Participants: Arthur Charguéraud.

Alexandre Moine, co-advised by Arthur Charguéraud and François Pottier, has presented a novel, high-level program logic for establishing space bounds in Separation Logic, for programs that execute with a garbage collector. A key challenge is to design sound, modular, lightweight mechanisms for establishing the unreachability of a block. In the setting of a high-level, ML-style language, a key problem is to identify and reason about the memory locations that the garbage collector considers as roots. We demonstrate the expressiveness and tractability of the proposed program logic via a range of examples, including recursive functions on linked lists, objects implemented using closures and mutable internal state, recursive functions in continuation-passing style, and three stack implementations that exhibit different space bounds. These last three examples illustrate reasoning about the reachability of the items stored in a container as well as amortized reasoning about space. All the results are proved in Coq on top of Iris. This work has been accepted for publication at POPL'23 [43].

8.4.4 Omnisemantics: Operational Semantics for Nondeterministic Languages

Participants: Arthur Charguéraud.

Arthur Charguéraud worked with collegues from MIT (Adam Chlipala and two of his students, Andres Erbsen and Samuel Gruetter) on a paper describing the technique of *omni-semantics*, a style for describing operational semantics particularly well-suited for nondeterministic languages. This technique introduces judgments that relate starting states to sets of outcomes, rather than to individual outcomes. Thus, a single derivation of these semantics for a particular starting state and program describes all possible nondeterministic executions, whereas in traditional small-step and big-step semantics, each derivation only talks about one single execution. We demonstrate how this restructuring allows for straightforward modeling of languages featuring both nondeterminism and undefined behavior. Specifically, omni-semantics inherently assert safety, i.e., they guarantee that none of the execution branches gets stuck, while traditional semantics need either a separate judgment or additional error markers to specify safety in the presence of nondeterminism. Applications presented include proofs of type soundness for lambda calculi, mechanical derivation of reasoning rules for program verification, and a forward proof of compiler correctness for terminating but potentially nondeterministic programs. All results are formalized in

Coq. This work has been accepted for publication in the journal ACM Transactions on Programming Languages and Systems (TOPLAS) [21].

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

Participants: Arthur Charguéraud, Begatim Bytyqi, Guillaume Bertholon.

In 2022, we pursued the development of the OptiTrust prototype framework for producing high-performance code via source-to-source transformations. We completed a major case study: a high-performance parallel implementation of a particle-in-cell algorithm used for plasma simulations. The naive implementation of the simulation consists of about 250 lines of C code. The optimization script consists of about 150 lines of OCaml code. The execution of that script produces a highly optimized code, essentially equivalent to the one that had been implemented by hand a few years ago [46]. A draft paper describing the main features of the framework and the case study is available [20].

The work on OptiTrust from Sept. 2019 to Sept. 2022 has been funded by an Inria "exploratory action". This line of research will continue for 4 years as part of an ANR grant, starting in Oct. 2022, with Arthur Charguéraud as PI. Thomas Koehler has been hired to work as a postdoc on this project, starting in Jan. 2023, to contribute to the implementation of transformations and of static analyses to justify correctness.

The PhD topic of Guillaume Bertholon, starting in Sept 2022, consists of developing a framework for representing Separation Logic derivations and applying proof-preserving, source-to-source transformations onto those derivations. His preliminary work has been presented at the JFLA workshop [14].

8.5 Pedagogical Research: Dissemination of feedback on a teaching experience

Participants: Stéphane Genaud, Vincent Loechner, Basile Sauvage, Arash Habibi.

Stéphane Genaud and Vincent Loechner participated in the AIPU22 (32ème Congrès de l'Association Internationale de Pédagogie Universitaire) pedagogical research conference to present their work on a distributed leadership emergence in the construction of a new class [16].

9 Partnerships and cooperations

9.1 European initiatives

9.1.1 H2020 projects

MICROCARD MICROCARD project on cordis.europa.eu

Title: Numerical modeling of cardiac electrophysiology at the cellular scale

Duration: From April 1, 2021 to September 30, 2024

Partners:

- INRIA Nancy - Grand Est, team CAMUS, France
- INRIA Bordeaux, team STORM, France
- MEGWARE Computer Vertrieb und Service GmbH, Germany
- SIMULA Research Laboratory AS, Norway
- Université de Strasbourg (Unistra), France
- Konrad-Zuse-Zentrum für Informationstechnik Berlin (ZIB), Germany

- Universita Della Svizzera Italiana (USI), Switzerland
- Karlsruher Institut Fuer Technologie (KIT), Germany
- Université de Bordeaux (UBx), France
- Universita degli Studi di Pavia (UNIPV), Italy
- Institut Polytechnique de Bordeaux (Bordeaux INP), France
- Numericor GmbH, Austria
- Orobix SRL, Italy

Inria contact: Mark POTSE

Coordinator: Mark POTSE

Summary: Cardiovascular diseases are the most frequent cause of death worldwide and half of these deaths are due to cardiac arrhythmia, a disorder of the heart's electrical synchronization system. Numerical models of this complex system are highly sophisticated and widely used, but to match observations in aging and diseased hearts they need to move from a continuum approach to a representation of individual cells and their interconnections. This implies a different, harder numerical problem and a 10,000-fold increase in problem size. Exascale computers will be needed to run such models.

We propose to develop an exascale application platform for cardiac electrophysiology simulations that is usable for cell-by-cell simulations. The platform will be co-designed by HPC experts, numerical scientists, biomedical engineers, and biomedical scientists, from academia and industry. We will develop, in concert, numerical schemes suitable for exascale parallelism, problem-tailored linear-system solvers and preconditioners, and a compiler to translate high-level model descriptions into optimized, energy-efficient system code for heterogeneous computing systems. The code will be parallelized with a recently developed runtime system that is resilient to hardware failures and will use an energy-aware task placement strategy.

The platform will be applied in real-life use cases with high impact in the biomedical domain and will showcase HPC in this area where it is painfully underused. It will be made accessible for a wide range of users both as code and through a web interface.

We will further employ our HPC and biomedical expertise to accelerate the development of parallel segmentation and (re)meshing software, necessary to create the extremely large and complex meshes needed from available large volumes of microscopy data.

The platform will be adaptable to similar biological systems such as nerves, and components of the platform will be reusable in a wide range of applications.

TEXTAROSSA

Participants: Bérenger Bramas, Hayfa Tayeb.

Title: Towards EXtreme scale Technologies and Accelerators for euROhpc hw/Sw Supercomputing Applications for exascale

Duration: From April 2021 to April 2024

Partners:

- Agenzia Nazionale per l'Energia, le Nuove Tecnologie e lo Sviluppo Economico Sostenibile (ENEA)
- Fraunhofer Gesellschaft Zur Förderung der Angewandten Forschung E.V. (FGH)
- Consorzio Interuniversitario per l'Informatica (CINI)

- Institut National de Recherche en Informatique et en Automatique (Inria)
- Bull SAS
- E4 Computer Engineering SpA
- Barcelona Supercomputing Center - Centro Nacional de Supercomputacion (BSC)
- Instytut Chemii Bioorganicznej Polskiej Akademii Nauk (PSNC)
- Istituto Nazionale di Fisica Nucleare (INFN)
- Consiglio Nazionale delle Ricerche (CNR)
- In Quattro Srl.

Summary: This European project aims at achieving a broad impact on the High Performance Computing (HPC) field both in pre-exascale and exascale scenarios 14 [7]. The TEXTAROSSA consortium will develop new hardware accelerators, innovative two-phase cooling equipment, advanced algorithms, methods and software products for traditional HPC domains as well as for emerging domains in High Performance Artificial Intelligence (HPC-AI) and High Performance Data Analytics (HPDA). We will focus on the scheduling of task-graphs under energy constraints and on porting scientific codes on heterogeneous computing nodes with FPGAs.

9.2 National initiatives

9.2.1 ANR OptiTrust

Participants: Arthur Charguéraud, Guillaume Bertholon, Jens Gustedt, Alain Ketterlin.

Turning a high-level, unoptimized algorithm into a high-performance code can take weeks, if not months, for an expert programmer. The challenge is to take full advantage of vectorized instructions, of all the cores and all the servers available, as well as to optimize the data layout, maximize data locality, and avoid saturating the memory bandwidth. In general, annotating the code with "pragmas" is insufficient, and domain-specific languages are too restrictive. Thus, in most cases, the programmer needs to write, by hand, a low-level code that combines dozens of optimizations. This approach is not only tedious and time-consuming, it also degrades code readibility, harms code maintenance, and can result in the introduction of bugs. A promising approach consists of deriving an HPC code via a series of source-to-source transformations guided by the programmer. This approach has been successfully applied in niche domains, such as image processing and machine learning. We aim to generalize this approach to optimize arbitrary code. Furthermore, the OptiTrust project aims at obtaining formal guarantees on the output code. A number of these transformations are correct only under specific hypotheses. We will formalize these hypotheses, and investigate which of them can be verified by means of static analysis. To handle the more complex hypotheses, we will transform not just code but also formal invariants attached to the code. Doing so will allow exploiting invariants expressed on the original code for justifying transformations performed at the n-th step of the transformation chain.

- Funding: ANR
- Start: October 2022
- End: September 2026
- Coordinator: Arthur Charguéraud (Inria)
- Partners: Inria team Camus (Strasbourg), Inria team TONUS (Strasbourg), Inria team Cambium (Paris), Inria team CASH (Lyon), CEA team LIST

9.2.2 ANR AUTOSPEC

Participants: Bérenger Bramas, Philippe Clauss, Stéphane Genaud, Garip Kusoglu, Michel Tching.

The AUTOSPEC project aims to create methods for automatic task-based parallelization and to improve this paradigm by increasing the degree of parallelism using speculative execution. The project will focus on source-to-source transformations for automatic parallelization, speculative execution models, DAG scheduling, and the activation mechanisms for speculative execution. With this aim, the project will rely on a source-to-source compiler that targets the C++ language, a runtime system with speculative execution capabilities, and an editor (IDE) to enable compiler-guided development. The outcomes from the project will be open-source with the objective of developing a user community. The benefits will be of great interest both for developers who want to use an automatic parallelization method, but also for high-performance programming experts who will benefit from improvements of the task-based programming. The results of this project will be validated in various applications such as a protein complexes simulation software, and widely used open-source software. The aim will be to cover a wide range of applications to demonstrate the potential of the methods derived from this project while trying to establish their limitations to open up new research perspectives.

- Funding: ANR (JCJC)
- Start: October 2021
- End: September 2025
- Coordinator: Bérenger Bramas

10 Dissemination

10.1 Promoting scientific activities

Member of the conference program committees

- Arthur Charguéraud was a PC member for the Symposium on Principles of Programming Languages (POPL 2022)and for the Internation Workshop on Advances in Separation Logic (ASL 2022).
- Philippe Clauss was a PC member for IMPACT 2022 (12th International Workshop on Polyhedral Compilation Techniques) and for IPDRM'2022 (Fifth Annual Workshop on Emerging Parallel and Distributed Runtime Systems and Middleware).

Reviewer

- Bérenger Bramas was a reviewer for Supercomputing (Poster), Compas, and the Conference on Mathematical Foundations of Computer Science (MFCS).
- Vincent Loechner and Philippe Clauss were reviewers for PPoPP '23 (27th ACM SIGPLAN Annual Symposium on Principles and Practice of Parallel Programming).

10.1.1 Journal

Member of the editorial boards

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

Reviewer - reviewing activities

- Bérenger Bramas was a reviewer for The Journal of Supercomputing (JOS), IEEE Transactions on Consumer Electronics (TCE), the Journal of Computer Science and Technology (JCST), Parallel Computing (Parco), and Parallel Processing Letters (PPL).
- Arthur Charguéraud wrote a review for Formal Methods Europe (FME), for the book *Functional Algorithms, Verified!*.
- Philippe Clauss was a reviewer for ACM TACO (Transactions on Architecture and Code Optimization), for Software: Practice and Experience (Wiley), for IET Software and for IEEE Micro.

10.1.2 Scientific expertise

- Jens Gustedt is a member of the ISO/IEC working groups ISO/IEC PL1/SC22/WG14 and WG21 for the standardization of the C and C++ programming languages, respectively.

10.1.3 Research administration

- Jens Gustedt is the head of the ICPS team for the ICube lab.
- Jens Gustedt is a member of the executive board of directors of the ICubelab, responsible for the IT and CS policy and for the coordination between the lab and the Inria center. In that function he also represents ICube in the board of the INRIA Nancy - Grand Est research center.
- Jens Gustedt is member of the steering committee of the interdisciplanary institute IRMIA++ of Strasbourg University.

10.2 Teaching - Supervision - Juries

10.2.1 Teaching

- Licence: Vincent Loechner, Algorithmics and programmation, 82h, L1, Université de Strasbourg, France
- Licence: Vincent Loechner, System administration, 40h, Licence Pro, Université de Strasbourg, France
- Licence: Vincent Loechner, System programming, 20h, L2, 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 systems, 12h, M1, Université de Strasbourg, France
- Eng. School: Vincent Loechner, Parallel programming, 20h, Telecom Physique Strasbourg - 3rd year, Université de Strasbourg, France
- 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: Alain Ketterlin, Architecture des systèmes d'exploitation, L3 Math-Info, 38h, Université de Strasbourg, France
- Licence: Alain Ketterlin, Programmation système, L2 Math-Info, 60h, Université de Strasbourg, France
- Master: Alain Ketterlin, Preuves assistées par ordinateur, 18h, Université de Strasbourg, France
- Master: Alain Ketterlin, Compilation, 84h, Université de Strasbourg, France
- Licence: Stéphane Genaud, Algorithmics and programmation, 82h, L1, Université de Strasbourg, France
- Licence: Stéphane Genaud, Parallel programming, 30h, L3, Université de Strasbourg, France
- Master: Stéphane Genaud, Cloud and Virtualization, 12h, M1, Université de Strasbourg, France
- Master: Stéphane Genaud, Large-Scale Data Processing, 15h, M1, Université de Strasbourg, France
- Master: Stéphane Genaud, Distributed Storage and Processing, 15h, M2, Université de Strasbourg, France
- Eng. School: Stéphane Genaud, Introduction to Operating Systems, 16h, Telecom Physique Strasbourg - 1st year, Université de Strasbourg, France
- Eng. School: Stéphane Genaud, Object-Oriented Programming, 60h, Telecom Physique Strasbourg - 1st year, Université de Strasbourg, France

10.2.2 Supervision

- PhD in progress: Clément Flint, *Efficient data compression for high-performance PDE solvers.*, advised by Philippe Helluy (TONUS), Stéphane Genaud, and Bérenger Bramas, since Nov 2020.
- PhD in progress: Arun Thangamani, *Code generation for heterogeneous architectures*, advised by Stéphane Genaud and Vincent Loechner, since Sept 2021.
- PhD from Oct 2022 to : Garip Kusoglu, *Automatic task-based parallelization by source-to-source transformations*, advised by Stéphane Genaud and Bérenger Bramas, since Oct 2021.
- PhD in progress: Alexandre Moine, *Formal Verification of Space Bounds*, advised by Arthur Charguéraud and François Pottier, at Inria Paris, since Oct 2021.
- PhD in progress: Hayfa Tayeb, *Efficient scheduling of task-based applications under energy constraints*, advised by Abdou Guermouche (HiePACS) and Bérenger Bramas, since Nov 2021.
- PhD in progress: Guillaume Bertholon, *Formal Verification of Source-to-Source Transformations*, is advised by Arthur Charguéraud and Jens Gustedt, since Sept 2022.
- PhD in progress: Anastasios Souris, *Speculative execution in the task-based parallelization*, advised by Philippe Clauss and Bérenger Bramas, since July 2022.
- PhD in progress: Clément Rossetti, *Algebraic loop transformations*, advised by Philippe Clauss, since Oct 2022.

10.2.3 Juries

- Arthur Charguéraud was an examiner for:
  - the PhD thesis of Adam Khayam, defended on Nov. 30, 2022, at the University of Rennes 1.
- Philippe Clauss was a reviewer for:
  - the PhD thesis of Paul Ianetta, defended on May 2, 2022, at ENS Lyon.
  - the PhD thesis of Camille Le Bon, defended on July 5, 2022, at the University of Rennes 1.
- Philippe Clauss was an examiner for:
  - the PhD thesis of Jean-Romain Luttringer, defended on Nov. 28, 2022, at the University of Strasbourg.
- Vincent Loechner was a reviewer for:
  - the PhD thesis of Van Man NGUYEN, defended on Dec. 16, 2022, at the University of Bordeaux.

10.3 Popularization

10.3.1 Castor Informatique Contest

Participants: Arthur Charguéraud.

Arthur Charguéraud is a co-organizer of the Concours Castor informatique. The purpose of the Concours Castor in to introduce pupils, from CM1 to Terminale, to computer sciences. 700,000 teenagers played with the interactive exercises in November and December 2022.

10.3.2 Modern C

Participants: Jens Gustedt.

Jens Gustedt has a blog about efficient programming, in particular about the C programming language.

10.3.3 Interventions

Arthur Charguéraud visited two classes (5ème, age 12) in December, to comment on the Concours Castor Informatique, and to give a brief overview on the development of computer sciences since the introduction of the Internet, as well as on the path to becoming a programmer.

11 Scientific production

11.1 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] P.-N. Clauss and J. Gustedt. ‘Iterative Computations with Ordered Read-Write Locks’. In: *Journal of Parallel and Distributed Computing* 70.5 (2010), 496–504. DOI: 10.1016/j.jpdc.2009.09.002. URL: https://hal.inria.fr/inria-00330024.

[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. 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.

11.2 Publications of the year

International journals

[7] G. Agosta, M. Aldinucci, C. Alvarez, R. Ammendola, Y. Arfat, O. Beaumont, M. Bernaschi, A. Biagioni, T. Boccali, B. Bramas et al. ‘Towards EXtreme scale technologies and accelerators for euROhpc hw/Sw supercomputing applications for exascale: The TEXTAROSSA approach’. In: *Microprocessors and Microsystems: Embedded Hardware Design* 95 (Nov. 2022), p. 104679. DOI: 10.1016/j.micpro.2022.104679. URL: https://hal.inria.fr/hal-03936864.

[8] C. Flint, B. Bramas and L. Paillat. ‘Automated prioritizing heuristics for parallel task graph scheduling in heterogeneous computing’. In: *PeerJ Computer Science* (16th Sept. 2022). URL: https://hal.inria.fr/hal-02993015.

[9] S. Hariri, S. Weill, J. Gustedt and I. Charpentier. ‘A balanced watershed decomposition method for rain-on-grid simulations in HEC-RAS’. In: *Journal of Hydroinformatics* 24.2 (22nd Jan. 2022), pp. 315–332. URL: https://hal.science/hal-03250815.

[10] C. Lalescu, B. Bramas, M. Rampp and M. Wilczek. ‘An efficient particle tracking algorithm for large-scale parallel pseudo-spectral simulations of turbulence’. In: *Computer Physics Communications* 278 (Sept. 2022), p. 108406. DOI: 10.1016/j.cpc.2022.108406. URL: https://hal.inria.fr/hal-03682460.

International peer-reviewed conferences

[11] A. Moine, A. Charguéraud and F. Pottier. ‘Specification and Verification of a Transient Stack’. In: CPP 2022 - 11th ACM SIGPLAN International Conference on Certified Programs and Proofs. Philadelphia, United States, 17th Jan. 2022. DOI: 10.1145/3497775.3503677. URL: https://hal.inria.fr/hal-03472028.

[12] C. Rossetti and P. Clauss. ‘Algebraic Tiling’. In: *IMPACT 2023, 13th International Workshop on Polyhedral Compilation Techniques.* IMPACT 2023, 13th International Workshop on Polyhedral Compilation Techniques. Toulouse, France, 16th Jan. 2023. URL: https://hal.inria.fr/hal-03944790.

[13] A. Thangamani, T. Trevisan, V. Loechner, S. Genaud and B. Bramas. ‘Lifting Code Generation of Cardiac Physiology Simulation to Novel Compiler Technology’. In: *Proceedings of the 21st ACM/IEEE International Symposium on Code Generation and Optimization (CGO ’23).* 21st ACM/IEEE International Symposium on Code Generation and Optimization (CGO ’23). Montréal Québec, Canada: ACM, 27th Feb. 2023, p. 13. DOI: 10.1145/3579990.3580008. URL: https://hal.inria.fr/hal-03977688.

National peer-reviewed Conferences

[14] G. Bertholon and A. Charguéraud. 'An AST for Representing Programs with Invariants and Proofs'. In: *Journées Francophones des Langages Applicatifs*. JFLA 2023 - 34èmes Journées Francophones des Langages Applicatifs. Praz-sur-Arly, France, 16th Jan. 2023, pp. 43–58. URL: https://hal.inria.fr/hal-03936618.

[15] C. Flint, B. Bramas, S. Genaud and P. Helluy. 'Parallelization of the Lattice-Boltzmann schemes using the task-based method'. In: COMPAS 2022 - Conférence francophone d'informatique en Parallélisme, Architecture et Système. Amiens, France, 5th July 2022. URL: https://hal.inria.fr/hal-03763577.

[16] B. Sauvage, S. Genaud, A. Habibi, V. Loechner and P.-O. Simonard. 'Émergence d'un leadership distribué pour la construction d'un enseignement'. In: AIPU22 - 32ème Congrès de l'Association Internationale de Pédagogie Universitaire. Rennes, France, 30th May 2022. URL: https://hal.inria.fr/hal-03653969.

[17] H. Tayeb, B. Bramas, A. Guermouche and M. Faverge. 'MulTreePrio: Scheduling task-based applications for heterogeneous computing systems'. In: COMPAS 2022 - Conférence francophone d'informatique en Parallélisme, Architecture et Système. Amiens, France, 5th July 2022. URL: https://hal.inria.fr/hal-03763824.

Scientific books

[18] J. Gustedt, P. Sewell, K. Memarian, V. B. F. Gomes and M. Uecker. *A Provenance-aware Memory Object Model for C*. ISO/IEC TC1/SC22/WG14, 16th June 2022, p. 131. URL: https://hal.inria.fr/hal-02957464.

Reports & preprints

[19] É. Alepins and J. Gustedt. *Unsequenced functions*. Inria Nancy - Grand Est, 8th Apr. 2022, p. 11. URL: https://hal.inria.fr/hal-02952723.

[20] A. Charguéraud, B. Bytyqi, D. Rouhling and Y. A. Barsamian. *OptiTrust: an Interactive Framework for Source-to-Source Transformations*. 9th Sept. 2022. URL: https://hal.inria.fr/hal-03773485.

[21] A. Charguéraud, A. Chlipala, A. Erbsen and S. Gruetter. *Omnisemantics: Smooth Handling of Nondeterminism*. 28th Sept. 2022. URL: https://hal.inria.fr/hal-03255472.

[22] A. Gilding and J. Gustedt. *Introduce storage-class specifiers for compound literals*. N3038. ISO JCT1/SC22/WG14, 21st July 2022. URL: https://hal.inria.fr/hal-03799906.

[23] A. Gilding and J. Gustedt. *The constexpr specifier for object definitions*. N3018. ISO JCT1/SC22/WG14, 6th June 2022. URL: https://hal.inria.fr/hal-03799921.

[24] A. Gilding and J. Gustedt. *Type inference for object definitions*. N3007. ISO JCT1/SC22/WG14, 10th June 2022. URL: https://hal.inria.fr/hal-03799959.

[25] A. Gilding and J. Gustedt. *Underspecified object declarations*. N3006. ISO JCT1/SC22/WG14, 10th June 2022. URL: https://hal.inria.fr/hal-03799968.

[26] J. Gustedt. *Add new optional time bases: Proposal for C23*. N2957. ISO JCT1/SC22/WG14, 10th Apr. 2022. URL: https://hal.inria.fr/hal-02378645.

[27] J. Gustedt. *Basic lambdas for C: proposal for C23*. N2892. ISO JCT1/SC22/WG14, 3rd Jan. 2022, p. 52. URL: https://hal.inria.fr/hal-03860638.

[28] J. Gustedt. *Improve type generic programming (slides)*. 2nd Feb. 2022. URL: https://hal.inria.fr/hal-03165732.

[29] J. Gustedt. *Improve type generic programming: proposal for C23*. N2890. ISO JCT1/SC22/WG14, 3rd Jan. 2022, p. 82. URL: https://hal.inria.fr/hal-03106758.

[30] J. Gustedt. *Make false and true first-class language features: proposal for C2x*. N2935. ISO JTC1/SC22/WG14, 15th Feb. 2022. URL: https://hal.inria.fr/hal-02167916.

[31] J. Gustedt. *Options for lambdas: (slides)*. 2nd Feb. 2022. URL: https://hal.inria.fr/hal-03553612.

[32] J. Gustedt. *Pointers and integer types*. N2889. ISO JCT1/SC22/WG14, 3rd Jan. 2022. URL: https://hal.inria.fr/hal-03363711.

[33] J. Gustedt. *Primary expressions and constant expressions, clarification request*. ISO JTC1/SC22/WG14, 14th June 2022. URL: https://hal.inria.fr/hal-03799948.

[34] J. Gustedt. *Remove ATOMIC VAR INIT*. N2886. ISO JTC1/SC22/WG14, 3rd Jan. 2022. URL: https://hal.inria.fr/hal-02167838.

[35] J. Gustedt. *Require exact-width integer type interfaces*. N2888. ISO JTC1/SC22/WG14, 3rd Jan. 2022. URL: https://hal.inria.fr/hal-03363699.

[36] J. Gustedt. *Revise spelling of keywords: proposal for C23*. N2934. ISO JTC1/SC22/WG14, 15th Feb. 2022. URL: https://hal.inria.fr/hal-02167870.

[37] J. Gustedt. *Type inference for variable definitions and function returns: proposal for C23*. N2923. ISO JCT1/SC22/WG14, 30th Jan. 2022, p. 22. URL: https://hal.inria.fr/hal-03106763.

[38] J. Gustedt. *Type inference for variables and functions (slides)*. 2nd Feb. 2022. URL: https://hal.inria.fr/hal-03165731.

[39] J. Gustedt. *Type-generic lambdas: (slides)*. 3rd Feb. 2022. URL: https://hal.inria.fr/hal-03259337.

[40] J. Gustedt. *Type-generic lambdas: proposal for C23*. N2924. ISO JCT1/SC22/WG14, 31st Jan. 2022, p. 14. URL: https://hal.inria.fr/hal-03106919.

[41] J. Gustedt and J. Meneide. *Introduce the nullptr constant*. N3042. ISO JTC1/SC22/WG14, 22nd July 2022. URL: https://hal.inria.fr/hal-02167929.

[42] J. Gustedt and M. Uecker. *Properly define blocks as part of the grammar: proposal for C23*. N2937. ISO JCT1/SC22/WG14, 17th Feb. 2022, p. 3. URL: https://hal.inria.fr/hal-03363674.

[43] A. Moine, A. Charguéraud and F. Pottier. *A High-Level Separation Logic for Heap Space under Garbage Collection (Extended Version)*. Inria, 2022. URL: https://hal.inria.fr/hal-03823056.

[44] H. Tayeb, B. Bramas and L. Paillat. *Autovesk: Automatic vectorization of unstructured static kernels by graph transformations*. 28th Dec. 2022. URL: https://hal.inria.fr/hal-03914178.

Other scientific publications

[45] M. Boileau, B. Bramas, E. Franck, R. Hélie, P. Helluy and L. Navoret. 'Parallel kinetic scheme in complex toroidal geometry'. In: *SMAI Journal of Computational Mathematics* (16th Dec. 2022), pp. 249–271. DOI: 10.5802/smai-jcm.86. URL: https://hal.science/hal-02404082.

11.3 Cited publications

[46] Y. A. Barsamian, A. Charguéraud, S. A. Hirstoaga and M. Mehrenberger. 'Efficient Strict-Binning Particle-in-Cell Algorithm for Multi-Core SIMD Processors'. In: *Euro-Par 2018 - 24th International European Conference on Parallel and Distributed Computing*. Turin, Italy, Aug. 2018. DOI: 10.1007/978-3-319-96983-1_53. URL: https://hal.archives-ouvertes.fr/hal-01890318.

[47] 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.

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

[49] 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.

[50] 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*. Compte tenu de la pandémie de coronavirus, la conférence système française Compas 2020 qui devait avoir lieu du 29 juin 2020 au 3 juillet à Lyon a dû être annulée. Lyon, France, June 2020. URL: https://hal.inria.fr/hal-02867413.