RESEARCH CENTRE

Inria Centre at Université Grenoble Alpes

IN PARTNERSHIP WITH:

Université de Grenoble Alpes

2023
ACTIVITY REPORT

Project-Team

CORSE

Compiler Optimization and Run-time SystEms

IN COLLABORATION WITH: Laboratoire d'Informatique de Grenoble (LIG)

DOMAIN

Algorithmics, Programming, Software and Architecture

THEME

Architecture, Languages and Compilation

Contents

Project-Team CORSE

Creation of the Project-Team: 2016 July 01

Keywords

Computer sciences and digital sciences

A1.1.1. – Multicore, Manycore

A1.1.2. – Hardware accelerators (GPGPU, FPGA, etc.)

A1.1.3. – Memory models

A1.1.4. – High performance computing

A1.1.12. – Non-conventional architectures

A1.6. – Green Computing

A2.1.6. – Concurrent programming

A2.1.7. – Distributed programming

A2.1.8. – Aspect-oriented programming

A2.1.10. – Domain-specific languages

A2.2. – Compilation

A2.2.1. – Static analysis

A2.2.2. – Memory models

A2.2.3. – Memory management

A2.2.4. – Parallel architectures

A2.2.5. – Run-time systems

A2.2.6. – GPGPU, FPGA...

A2.2.8. – Code generation

A2.2.9. – Security by compilation

A2.3.2. – Cyber-physical systems

A4.4. – Security of equipment and software

A7.1. – Algorithms

A9.6. – Decision support

Other research topics and application domains

B3.1. – Sustainable development

B4.5. – Energy consumption

B5.3. – Nanotechnology

B6.1.2. – Software evolution, maintenance

B6.6. – Embedded systems

B6.7. – Computer Industry (harware, equipments...)

B9.1. – Education

1 Team members, visitors, external collaborators

Research Scientists

- Fabrice Rastello [Team leader, INRIA, Senior Researcher, HDR]
- Walid Ghandour [INRIA, Starting Research Position, from Mar 2023]
- Guillaume Iooss [INRIA, Researcher]

Faculty Members

- Florent Bouchez [UGA, Associate Professor]
- Ylies Falcone [UGA, Associate Professor]
- Manuel Selva [GRENOBLE INP, Associate Professor]

Post-Doctoral Fellow

- Hugo Pompougnac [INRIA, Post-Doctoral Fellow, from Mar 2023]

PhD Students

- Hamzah Al-Qadasi [Univ. Grenoble Alpes]
- Theo Barollet [INRIA, until Mar 2023]
- Theophile Bastian [UGA]
- Nicolas Derumigny [Foreign university, until Aug 2023, co-supervision with the USA]
- Irman Faqrizal [Univ. Grenoble Alpes, Convecs]
- Florian Gallay [UGA, until Oct 2023, Halted the PhD]
- Chukri Soueidi [INRIA, until Sep 2023]
- Ahang Zuo [Univ. Grenoble Alpes, Convecs]

Technical Staff

- Christophe Guillon [INRIA, Engineer]
- Chukri Soueidi [INRIA, Engineer, from Oct 2023]
- Valentin Trophime-Gilotte [INRIA, Engineer]

Interns and Apprentices

- Thomas Civade [UGA, from Jun 2023 until Aug 2023]
- Etienne Delort [INRIA, Intern, from Apr 2023 until Sep 2023]
- Alexis Detroyat [UGA, Intern, from Jun 2023 until Jul 2023]
- Alban Dutilleul [INRIA, Intern, from Sep 2023]
- Chu Hoang Anh Nguyen [UGA, Intern, from Jun 2023 until Jul 2023]
- Laura Riou [INRIA, Intern, from Apr 2023 until Sep 2023]
- Tristan Rollet [UGA, from Jun 2023 until Jul 2023]
- Anukriti Srivastava [INRIA, from Apr 2023 until Sep 2023]

Administrative Assistant

- Imma Presseguer [INRIA]

2 Overall objectives

Languages, compilers, and run-time systems are some of the most important components to bridge the gap between applications and hardware. With the continuously increasing power of computers, expectations are evolving, with more and more ambitious, *computationally intensive and complex applications*. As desktop PCs are becoming a niche and servers mainstream, three categories of computing impose themselves for the next decade: mobile, cloud, and super-computing. Hence *diversity, heterogeneity* (even on a single chip) and thus also *hardware virtualization* are putting more and more pressure both on compilers and run-time systems. However, because of the energy wall, *architectures* are becoming more and more *complex* and *parallelism ubiquitous* at every level. Unfortunately, the memory-CPU gap continues to increase and energy consumption remains an important issue for future platforms. To address the challenge of *performance and energy consumption* raised by silicon companies, compilers and run-time systems must *evolve* and, in particular, interact, *taking into account the complexity of the target architecture*.

The overall objective of CORSE is to address this challenge by *combining static and dynamic compilation* techniques, with more interactive *embedding of programs and compiler environment in the run-time system*.

3 Research program

3.1 Scientific Foundations

One of the characteristics of CORSE is to base our researches on diverse advanced mathematical tools. Compiler optimization requires the usage of several tools around discrete mathematics: combinatorial optimization, algorithmic, and graph theory. The aim of CORSE is to tackle optimization not only for general purpose but also for domain specific applications. In addition to run-time and compiler techniques for program instrumentation, hybrid analysis and compilation advances will be mainly based on polynomial and linear algebra.

The other specificity of CORSE is to address technical challenges related to compiler technology, run-time systems, and hardware characteristics. This implies mastering the details of each. This is especially important as any optimization is based on a reasonably accurate model. Compiler expertise will be used in modeling applications (e.g. through automatic analysis of memory and computational complexity); Run-time expertise will be used in modeling the concurrent activities and overhead due to contention (including memory management); Hardware expertise will be extensively used in modeling physical resources and hardware mechanisms (including synchronization, pipelines, etc.).

The core foundation of the team is related to the combination of static and dynamic techniques, of compilation, and run-time systems. We believe this to be essential in addressing high-performance and low energy challenges in the context of new important changes shown by current application, software, and architecture trends.

3.2 Main Research Directions

Our project is structured along three main directions. The first direction belongs to the area of **program analysis and optimization**. This direction breaks down into:

- Performance debugging, binary instrumentation, automatic characterization and simulation of architectures
- Loop scheduling, data locality, I/O complexity
- Compiler design, hybrid compilation, domain-specific intermediate representations

The second direction belongs to the area of **runtime monitoring, verification, and enforcement**. This direction breaks into:

- Instrumentation for Java programs for performance and security
- Monitoring of learning-enabled components using geometrical shape abstraction
- Decentralization of the monitoring process for multi-threaded and distributed systems
- Predictive monitoring of business processes

The third direction belongs to the area of **teaching and tutoring of programming**. This direction breaks into:

- Visualisation tools for teaching of programming
- Tools and education of debugging
- Problem based learning, generation, recommandation

4 Application domains

4.1 Transfer

The main industrial sector related to the research activities of CORSE is the one of semi-conductor (programmable architectures spanning from embedded systems to servers). Obviously any computing application which has the objective of exploiting as much as possible the resources (in terms of high-performance but also low energy consumption) of the host architecture is intended to take advantage of advances in compiler and run-time technology. These applications are based on numerical kernels (linear algebra, FFT, convolution, ...) that can be adapted to a wide spectrum of architectures. More specifically, an important activity concerns the optimization of machine learning applications for some high-performance accelerators. Members of CORSE already maintain fruitful and strong collaborations with several companies such as KALRAY, GOOGLE, STMICROELECTRONICS, ARM.

5 Social and environmental responsibility

5.1 Footprint of research activities

As expected, after the COVID pandemia, team members kept travel activities quite low compared to before the pandemia. Whenever long distance meetings (such as conference PC) could be done virtually, travel has been avoided. Also, team members try to better use existing hardware instead of replacing them (buying new ones).

5.2 Impacting research directions for environment

Because of rebound effect, improving efficiency does not necessarily improve environmental impact. It is thus crucial to think how our community can have actual impact on sustainable computing, that is, influence better design ("R" friendly) and better usage (consume less) of our compute resources. To achieve this goal, we arrange panels with the aim of raising awareness within our community about this significant issue. We expect some of our future research projects to address the challenge of sustainable computing without just focusing on energy efficiency but by considering the global systemic impact as much as possible.

5.3 Impacting usage

The main two challenges of sustainable computing are:

1. *Decrease usage*: While the actual environmental impact of our usage is already not that clear to experts like us (need for open data), it is even less clear for users and developers. It is thus our responsibility to expose estimations of resource usage (and associated environmental impact) to the developers. Performance debugging tools should evolve to provide meaningful metrics and make it accessible to none-experts.
2. *Increase the lifetime of hardware* (that is, Reuse, Repair, Re...): The need for supporting the development of simple, open-source, commons, low-impact (not necessarily low-tech) hardware/software solutions is becoming critical but not sufficient. We also need to provide the microscope and the tool-box so that a majority (including sometimes the end-user) can repair or repurpose and device.

Compiler analysis, programming infrastructure, hardware modeling, teaching tools, HIM, etc. are at the heart of those challenges.

6 New software, platforms, open data

6.1 New software

6.1.1 Pipedream

Name: Pipedream

Keywords: Performance analysis, CPU, Reverse engineering

Scientific Description: Pipedream reverse engineers the following performance characteristics: (1) Instruction latency – The number of cycles an instruction requires to execute. (2) Peak micro-op retirement rate – How many fused micro-ops the CPU can retire per cycle. (3) Micro-fusion – The number of fused micro-ops an instruction decomposes into. (4) Micro-op decomposition and micro-op port usage – The list of unfused micro-ops every instruction decomposes into and the list of execution ports every one of these micro-ops can execute on.

The first step of the reverse engineering process consists of generating a number of microbenchmarks. Pipedream then runs these benchmark, measuring their performance using hardware counters. The latency, throughput, and micro-fusion of different instructions can then be read directly from these measurements.

The process of finding port mappings, i.e. micro-op decompositions and micro-op port usage, however, is more involved. For this purpose, we have defined a variation of the maximum flow problem which we call the "instruction flow problem". We have developed a linear program (LP) formulation of the instruction flow problem which can be used to calculate the peak IPC and micro-operations per cycle (MPC) a benchmark kernel can theoretically achieve with a given port mapping. The actual port mapping of the underlying hardware is then determined by finding the mapping for which the throughput predicted by instruction flow best matches the actual measured IPC and MPC.

Functional Description: Pipedream is a tool for measuring specific performance characteristics of CPUs It is used to build the performance model of another tool called Gus (https://gitlab.inria.fr/nderumig/gus). Pipedream finds measured performance characteristics such as the throughput and latency of instructions by running a large set of automatically generated microbenchmarks. The tool can also find port mappings, a model of part of the CPU instruction scheduler, by analysing performance measurements of specially crafted microkernels using a LP solver. We have used it to produce a port mapping for the Intel Skylake CPU architecture. Pipedream is able to find the port mappings for some instructions for which existing approaches fall back to manual analysis.

URL: `https://gitlab.inria.fr/fgruber/pipedream`

Contact: Nicolas Derumigny

6.1.2 IOLB

Keywords: Complexity, Polyhedral compilation, Performance analysis

Functional Description: IOLB computes a symbolic lower bound on the I/O, or data movement, complexity of a computer program, that is the amount of data that needs to be moved between cache and main memory to perform its computation. The input is a C program, and the output is a mathematical formula that depends on program parameters (array sizes...) and cache size.

URL: `https://gitlab.inria.fr/CORSE/iolb`

Publications: hal-02421026, hal-02910961

Contact: Guillaume Iooss

6.1.3 IOOpt

Keywords: I/O, Polyhedral compilation

Functional Description: IOOpt takes as input an abstract representation of a tileable program. The tool generates a tractable set of relevant permutations of the tiling loops, and a symbolic I/O cost expression for each of them. It then uses a non-linear problem optimizer to find the best permutations and corresponding tile sizes for a given value of machine parameters (cache sizes and bandwidths at each level). IOOpt can also be used to find an upper bound on the I/O cost of a program, for a given tiling scheme.

Publication: hal-03200539

Contact: Guillaume Iooss

6.1.4 PALMED

Keywords: CPU, Performance measure, Performance analysis, Reverse engineering

Functional Description: PALMED computes a bipartite graph assembly instructions <-> abstract resources that may be used for performance prediction, targeting static analysis tools and compilers. Internally, PALMED uses PIPEDREAM as a framework for microbenchmarking code generation, and use gurobi to find a first small graph. Then, PALMED deduces from the found resources and the microbenchmarks that saturates them a mapping of every supported instruction.

URL: `https://gitlab.inria.fr/nderumig/palmed`

Contact: Fabrice Rastello

6.1.5 BISM

Name: BISM: Bytecode-level Instrumentation for Software Monitoring

Keywords: Java, Bytecode, Instrumentation, Control Flow

Functional Description: BISM (Bytecode-level Instrumentation for Software Monitoring) is a lightweight Java bytecode instrumentation tool which features an expressive high-level control-flow-aware instrumentation language. The language follows the aspect-oriented programming paradigm by adopting the joinpoint model, advice inlining, and separate instrumentation mechanisms. BISM provides joinpoints ranging from bytecode instruction to method execution, access to comprehensive context information, and instrumentation methods. BISM runs in two modes: build-time and load-time.

URL: `https://gitlab.inria.fr/bism/bism-public`

Publication: hal-03081265

Contact: Ylies Falcone

Participants: Chukri Soueidi, Ylies Falcone, Ali Kassem

6.1.6 EasyTracker

Keywords: Monitoring, Debug, Visualization, Teaching of programming

Scientific Description: Learning to program involves building a mental representation of how a machine executes instructions and stores information in memory. To help students, teachers often use visual representations to illustrate executions of programs or particular concepts in their lectures. EasyTracker is a library that assists teachers of programming courses in building tools that generate representations tuned to their needs from actual programs. At its core, EasyTracker provides ways of controlling the execution and inspecting the state of programs. The control and inspection are driven and customized through a Python interface. The controlled program itself can be written either in Python or in any GDB supported language like C.

Functional Description: EasyTracker is a Python library for controlling and inspecting program execution. The library allows to pause the program's execution on specific points of interest (modification of a variable, recursive function call return for example) described using a high level interface and allows to inspect the state of a paused program. The controlled program can be written either in Python, C, or assembly languages. While the library has been initially written to ease the creation of visualization tools for teaching programming, it can also be used in other contexts such as debugging or testing.

URL: `https://gitlab.inria.fr/CORSE/easytracker/`

Publication: hal-04368835

Contact: Manuel Selva

Participants: Theo Barollet, Manuel Selva, François Broquedis, Florent Bouchez, Fabrice Rastello, Christophe Guillon

6.1.7 GUS

Keywords: CPU, Microarchitecture simulation, Performance analysis, Dynamic Analysis

Functional Description: GUS' goal is to detect performance bottlenecks at the very low level on monothread applications by the use of sensitivity analysis. It is coded as a QEMU plug-in in order to collect runtime information that are later treated by the generic CPU model.

URL: `https://gitlab.inria.fr/nderumig/gus`

Contact: Nicolas Derumigny

6.1.8 CesASMe

Keywords: CPU, Microarchitecture simulation, Performance analysis

Functional Description: CesASMe fulfills 2 goals: 1- It automatically generates a large amount of microbenchmarks (ie benchmarks whose computation is L1-resident) from a benchmark suite, each implementing a different loop optimisation : tiling, loop fusion, etc. 2- For each compiled microbenchmark, it collects execution time estimates provided by the studied code analyzers (uica, iaca, Gus...), lifts them to common metrics and compare them with each other and with a measure.

Contact: Theophile Bastian

6.1.9 staticdeps

Keywords: CPU, Microarchitecture simulation, Performance analysis

Functional Description: Given an executable, staticdep focuses on the loops within. It computes statically a triplet (source, dest, k) for each dependency, where source is the instruction producing the data carrying the dependency, dest is the instruction consuming the data carrying the dependency, and k is the number of iterations required for the dependency to appear.

URL: `https://gitlab.inria.fr/CORSE/uica-staticdeps`

Contact: Theophile Bastian

6.1.10 Agdbentures

Keywords: Debug, Teaching of programming, Video Game

Functional Description: Agdbentures is a game to teach debugging. It is based on GDB (the Gnu Debugger), uses the Easytracker library, and proposes to students an RPG-like (Role Playing Game) 2D interface, where each level is a program in C-language that contains one or more bugs. To validate a level, one needs to first correct the bugs that blocks the main character in its goals. Difficulty is gradual, and the code for each level is based (and expands) on the preceding level, which allows players to get familiar with the code base.

URL: `https://gitlab.inria.fr/CORSE/agdbentures`

Contact: Florent Bouchez

7 New results

7.1 Performance Debugging and Compiler Optimization

Participants: Fabrice Rastello, Guillaume Iooss, Christophe Guillon, Hugo Pompougnac, Mariana Vargas Vieyra *(EPFL, Switzerland)*, Alban Dutilleul, Nicolas Derumigny, Théophile Bastian, Laura Riou, Étienne Delort, Anukriti Srivastava, Nicolas Tollenaere *(Inria CORSE)*, Albert Cohen *(Google, France)*, P. Sadayappan *(OSU, USA)*.

Our current efforts with regard to code optimization follow three directions.

1. The primary focus is on enhancing compiler optimization techniques by considering pattern-specific applications, such as those within the polyhedral framework or more restrictively, those related to machine learning. This improvement involves exploring the usage of reinforcement learning for optimizing tiling transformations of dense kernels widely used in deep learning, like tensor contraction and convolution. Additionally, new code generation strategies are being explored for optimizing the computing kernel of Sparse Matrix-Matrix (SpMM) operations.

2. The second consists in generating performance debugging tools. In that context we improved our existing tools PALMED (see Section 6.1.4) and GUS (see Section 6.1.2) but also developped an infrastructure called CesASMe (see Section 6.1.8) for evaluating and comparing existing tools.

3. The third consists in developping proof schemes for automatically deriving parametric data movement complexity of programs. In that context, we improved our tool IOLB by incorporating a new proof pattern in it.

We have utilized our compiler expertise to assess a portion of the environmental impact associated with numerical operations. In particular, we advised three interns who investigated the carbon footprint caused by the training and inference phase of deep neural network architectures, and its correlation with the number of instructions and number of memory movements. A report was produced [10].

7.1.1 Performance Modeling, Schedule Optimization and Code Generation for Tensor Computations

Tensor computation such as Sparse Matrix Multi-vector multiplication (SpMM), Sampled Dense Dense Matrix Multiplication, Dense Matrix Multiplication, Tensor Contraction, Convolution are important kernels used in many domains like Fluid Dynamics, Data Analytics, Economic Modelling, and Machine Learning. Developing highly optimized code for such kernels requires the combination of highly tuned register/instruction level micro-kernels and appropriate multi-level tiling.

We focused on the problem of automatic optimization of such operators from multiple directions. A first direction studied statically the structure of the optimization space and the impact of optimization choices, in order to increase the probability of selecting a well-performing implementation using a combination of machine learning and analytical modeling techniques. This works is mostly focused on dense tensor operations on CPU.

Another direction focuses on exploring the effect of different optimization and preprocessing techniques for SpMM. In particular, we exploit the structure of the sparse matrix, which we assume is known at compile time, to guide our optimization. We also consider sparse matrices originating from machine learning applications, such as DNN and transformers, which exhibit less sparsity and less structural properties compared to usual sparse matrices.

Cache model and Tiling transformation for tensor operations on CPU A first direction concerns the evaluation of the effectiveness of analytical cache modeling for CPU, following our previous work on IOOpt and Ttile. Due to the complexity of cache behavior, analytical modeling relies on simplified hypotheses, such as full-associativity, or LRU cache policy, to estimate and minimize the number of cache misses. We analyze the trade-off between the reduction of the computational cost and the loss of precision of the model, using several variations of analytical cache modeling algorithms, and the Dinero cache simulator.

Reinforcement learning for tiling transformations Another explored direction for optimizing the performance of tiled tensor operations is the usage of reinforcement learning techniques through iterative compilation on the transformations space.

To this end we are leveraging several components: tiling solutions generation tools and AI compiler backends featuring execution feedback loops; unsupervised surrogate models trained at compilation time; performance metrics such as the ones presented above as additional inputs to the surrogate model; Bayesian Optimization frameworks managing the reinforcement loop and statistical knowledge accumulated over executions.

We explore several dimensions in the choices available for this kind of framework in our particular context: sampling strategy, acquisition function strategy, scalability of Gaussian Processes, categorical v.s. continuous kernels, apriori on performance metrics, management of reduced dimensions when using metrics, variational or closed form processes.

Our objectives are: improve the convergence time for finding good optimization choices compared to the current state of the art; find better solutions to the problem of termination of the iterative search and global optimization given a time budget; provide an easy interface for injecting expert features (estimated performance metrics for instance).

Optimizing sparse matrix multiplication (SpMM) on CPU We focused on improving the performance of the Sparse-Matrix Multiplication kernel, which is a matrix product between a sparse matrix A and a dense matrix B.

A first approach tried to reorder the rows of A to agglomerate the non-zero values into dense blocks. Instead of using a sparse implementation, a dense matrix multiplication microkernel (or accelerator) can be used for these blocks. The size of these blocks is determined by a trade-off between the sparsity of the agglomerated blocks and the performance of the dense microkernels.

A second approach being explored is the usage of recurring patterns of non-zero values in the sparse matrix. These patterns are discovered when scanning the matrice through a small window generally fitting the register file. From these patterns it is possible to have a code generation strategy which among other things combines register tiling, unrolling, pattern outlining or inlining and memory tiling. A large space of optimization choices is available and the challenge is to find a good one-shot heuristic for choosing the parameters controlling these choices. To this end we also explore computed performance estimation metrics for this particular problem. Also, we extensively use tools for performance debugging such as GUS (see Section 6.1.2) for fine grain or the profiling tools based on hardware performance counter.

7.1.2 Automatic Resource Characterization and Performance Feedback

Performance modeling is a critical component for program optimizations, assisting compilers as well as developers in predicting the performance of code variations ahead of time. Performance models can be obtained through different approaches that span from precise and complex simulation of a hardware description (Zesto, GEM5, PTLSim) to application level analytical formulations. An interesting approach for modeling the CPU of modern pipelined, super-scalar, out-of-order processors trades simulation time with accuracy by separately characterizing both latency and throughput of instructions. This approach is suitable both for optimizing compilers, but also for hand-tuning critical kernels written in assembler (see Section 7.1.1). It is used by performance-analysis tools such as CQA, Intel IACA, OSACA, MIAMI or llvm-mca. Cycle-approximate simulators such as ZSim or MCsimA can also take advantage of such an instruction characterization. PALMED (see Section 6.1.4) and GUS (see Section 6.1.2) have been developped in this context. This year, we improved those tools along the following directions. PALMED now supports ARM instruction set. GUS supports AVX512 instruction set and now contains several other extensions that improves its accuracy.

Facing the diversity of existing analyzers, evaluating their strengths and weaknesses is important to guide both their usage and their enhancement. We developped CesASMe (see Section 6.1.8), a fully-tooled solution to evaluate code analyzers on C-level benchmarks composed of a benchmark derivation procedure that feeds an evaluation harness. We conclude that memory-carried data dependencies are a major source of imprecision for these tools. We tackle this issue with staticdeps (see Section 6.1.9), a static analyzer extracting memorycarried data dependencies, including across loop iterations, from an assembly basic block. We integrate its output to uiCA, a state-of-the-art code analyzer, to evaluate staticdeps' impact on a code analyzer's precision through CesASMe.

7.1.3 Automatic derivation of parametric data movement complexity

When studying the properties of an algorithm, we often consider its computational complexity. This quantifies the amount of computation needed, but it does not consider other critical aspects such as the quantity of data movement needed to perfom these computations.

The *I/O complexity* (also called data-movement complexity) is the minimal amount of data movement required by an algorithm, for all valid schedules. In order to estimate this quantity, we derive lower and upper parametric bounds by using two different approaches. A lower bound can be deduced, by proving mathematically that a certain volume of computation must happen. This mathematical proof follows a derivation strategy, such as the K-partitioning method or the wavefront method. An upper bound can be found by exhibiting a schedule that corresponds to this quantity of data movement.

We introduced a new specialised derivation proof strategy to improve the I/O complexity lower bound of the programs exhibiting an *hourglass pattern* in their dependence graph. An hourglass pattern is a pattern of dependencies that combines successive reductions and broadcasts over a parametric number of elements. This pattern strongly constrains the shape of a valid tiling, and its properties can be used to derive tight asymptotic lower bounds. Several important linear algebra algorithms exhibit such a pattern, such as GramSchmidt, QR Householder or Hessenberg matrix factorisation (gehd2). We integrated this proof inside the automatic I/O complexity lower bound derivation tool, IOLB (see Section 6.1.2).

7.2 Runtime Monitoring, Verification, and Enforcement

Participants: Yliès Falcone, Sylvain Hallé, Marius Monnier, Florian Gallay, Irman Faqrizal, Chukri Soueidi, Hamzah Al-Qadasi, Ahang Zuo, Gwen Salaün, Saddek Bensalem, Changshun Wu.

This section overviews our ongoing efforts on the topics of runtime monitoring, verification, and testing. More specifically, our work can be categorized into the following topics:

- Runtime verification and testing, where we define approaches for neural networks [2];
- Runtime verification of multithreaded programs [9, 7];
- Runtime verification of complex systems [8];
- Instrumentation [1, 5, 6], where we define an instrumentation approach to capture conservative models of Java programs for runtime verification.

7.2.1 Runtime Verification and Testing for Neural Networks

There has been a growing trend in AI testing toward developing new test prioritization algorithms for deep learning systems. These algorithms aim to reduce the cost and time needed to annotate test datasets by prioritizing instances with a higher chance of exposing faults. Various metrics have been used to evaluate the effectiveness of these algorithms, e.g., APFD, RAUC, and ATRC. However, there is a lack of research to confirm their validity. The results indicate that the existing metrics have severe limitations. For example, some metrics ignore the labeling budget and prioritize the fault detection rate instead of the fault detection ratio. Moreover, others overlook the prioritization difficulty in the evaluation. As a solution, we develop a new metric (WFDR), which solves the deficiencies of previous metrics. We also draw attention to a new research area, known as severity prioritization, which emphasizes the importance of prioritizing misclassified instances according to the severity level, particularly in critical situations. Our experiments reveal that instances with high severity make up more than 20% of all misclassified instances. Thus, these instances should be prioritized when it comes to labeling. Consequently, we proposed a new metric known as (SFDR) that evaluates the effectiveness of algorithms in prioritizing high-severity instances. Our evaluations show that our proposed metrics are more effective than other existing metrics. Besides, our two metrics re-evaluate some recent algorithms and indicate that these algorithms perform poorly.

The DeepAbstraction algorithm that we developped employs a box-abstraction concept, the efficiency of which depends on the tau parameter, the clustering parameter, that influences the size of these boxes. The conclusion of the previous experiments using tau values of 0.4 or 0.05 has failed to produce optimal results among all experiments. This highlights a significant challenge in the DeepAbstraction framework concerning the appropriate selection of the tau parameter. The selection of the tau value is extremely crucial, given its decisive effect on box size and, subsequently, the stability and efficacy of the framework. Addressing this challenge, we propose a methodology called combined parameterized boxes. This approach leverages the collective verdicts of monitors with various tau values to evaluate network predictions. We assign appropriate weights to these verdicts to ensure that no single verdict influences the decision-making process, thereby ensuring balance. Furthermore, we propose multiple strategies for integrating the weighted verdicts of monitors into a conclusive verdict, such as mean, max, product, and mode. The results of our investigation demonstrate that our approach can notably boost the DeepAbstraction framework's performance. Compared to the leading algorithms, DeepAbstraction++ consistently outperforms its competitors, delivering an increase in performance between 2.38% and 7.71%. Additionally, DeepAbstraction++ brings remarkable stability to the process, addressing a significant shortcoming of the earlier version of DeepAbstraction.

7.2.2 Runtime Verification of Multithreaded Programs

Monitoring concurrent programs typically relies on collecting traces of abstract program executions. However, existing approaches targeting general behavioral properties are either not tailored for online monitoring, are no longer maintained, or implement naive instrumentation that often leads to unsound verdicts. We first define the notion of when a trace is representative of a concurrent execution. We then present a non-blocking vector clock algorithm to collect sound concurrent traces on the fly reflecting the partial order between events. Moreover, concurrent events in the representative trace pose a soundness problem for monitors synthesized from total order formalisms. For this, we extract a causal dependence relation from the monitor to check if the trace has the needed orderings and define the conditions to decide at runtime when a collected trace is monitorable. We implement our contributions in a tool, FACTS, which instruments programs compiling to Java bytecode, constructs sound representative traces, and warns the monitor about non-monitorable traces. We evaluate our work and compare it with existing approaches.

Moreover, we introduced a generic approach for monitoring multithreaded programs online leveraging existing runtime verification (RV) techniques. In our setting, monitors are deployed to monitor specific threads and only exchange information upon reaching synchronization regions defined by the program itself. They use the opportunity of a lock in the program, to evaluate information across threads. As such, we refer to this approach as opportunistic monitoring. By using the existing synchronization, our approach reduces additional overhead and interference to synchronize at the cost of adding a delay to determine the verdict. We utilize a textbook example of readers-writers to show how opportunistic monitoring is capable of expressing specifications on concurrent regions. We also presented a preliminary assessment of the overhead of our approach and compare it to classical monitoring showing that it scales particularly well with the concurrency present in the program.

7.2.3 Runtime Verification of Complex Systems

We considered the problem of discovering divergences between the actions of a digital twin and those of its real-world counterpart. It observes the similarities between this problem and an existing field of formal methods called Runtime Verification (RV), and suggests leveraging and adapting RV techniques to this effect. Concretely, three important aspects of the problem are identified and for which both theoretical and practical challenges must be addressed.

We also introduced a flexible and modular approach to dynamic program analysis for JVM-based languages, aiming to address the limitations of existing tools, in particular their limited expressivity and tight coupling between instrumentation and analysis. The proposed solution decouples these two processes using BISM (see Section 6.1.5), a lightweight instrumentation language, and BeepBeep, a complex event processing engine. This novel combination enhances expressiveness, promotes reusability, and integrates seamlessly into JVM-based projects. Various analyses such as monitoring, profiling, coverage measurement, and complex event generation are demonstrated, showcasing the approach's flexibility.

7.2.4 Instrumentation for Monitoring

Instrumentation is crucial in Runtime Verification because it should ensure that monitors are fed with relevant and accurate information about the executing program under monitoring. While expressive instrumentation is desirable to handle any possible monitoring scenario, instrumentation should also efficiently capture the just-needed information and impact the monitoring program as least as possible. Our presented tutorial comprehensively overviews the instrumentation process and considerations for single and multithreaded programs. We discuss often overlooked aspects in instrumenting multithreaded programs. We also cover metrics for evaluating the efficiency and effectiveness of instrumentation. We use four hands-on use cases to apply the introduced concepts and provide practical guidance on choosing and applying instrumentation for runtime verification.

We also introduced a novel instrumentation language for BISM, a lightweight bytecode-level instrumentation tool for JVM languages. The new DSL aims to simplify the instrumentation process, making it more accessible to a wider user base. It employs an intuitive syntax, directly mapping to the key requirements of program instrumentation for runtime verification. It enhances productivity by eliminating boilerplate code and low-level details, while also supporting code generation and collaboration. The DSL balances expressiveness, and abstraction, bridging the gap between domain experts and the complexities of instrumentation specification.

7.3 Teaching of Algorithms, Programming and Debugging

Participants: Théo Barollet, Florent Bouchez Tichadou, Manuel Selva, Fabrice Rastello, Christophe Guillon, Valentin Trophime-Gilotte.

Our goal here is to combine our expertise in compilation and teaching to help teachers and learners in computer science fields such as programming, algorithms, data structures, automata, debugging, or more generally computing literacy. This axis is developed into three projects:

- EasyTracker: a library for controlling and inspecting the execution of a program. See Section 6.1.6.
- Agdbentures: a game that helps learners to gain skills in debugging, which is based on EasyTracker. See Section 6.1.10.
- Usage of active learning techniques in the context of large programming classes

7.3.1 Easytracker : A generic library for controlling and inspecting program execution and state

Learning to program involves building a mental representation of how a machine executes instructions and stores data in memory. To help students, teachers often use visual representations to illustrate the execution of programs or particular concepts in their lectures. As a famous example, teachers often represent references/pointers with arrows pointing to objects or memory locations. While these visual representations are mostly hand-drawn, there is a tendency to supplement them with tools. However, building such a tool from scratch requires much effort and a high level of debugging technical expertise, while existing tools are difficult to adapt to different contexts.

EasyTracker (See Section 6.1.6) is a Python library targeting teachers who are not debugging experts. By providing ways of controlling the execution and inspecting the state of programs, EasyTracker simplifies the development of tools that generate tuned visual representations from the controlled execution of a program. The controlled program can be written either in Python, C, or assembly languages. To ease the development of visualization tools working for programs in different languages and to allow the building of web-based tools, EasyTracker provides a language-agnostic and serializable representation of the state of a running program.

We pursued our work on the EasyTracker library with a focus on usability. To that end, we worked on the packaging of the library, we improved the application programming interface and we created a complete documentation for the library.

This work has been accepted for publication at ACM/IEEE CGO 2024 [3].

As a side project of EasyTracker, still to help computer science teachers, we started the development of a library dedicated to visualization. The aim of this library is to provide an application programming interface allowing to draw hierarchical nodes with connections and positioning constraints between them.

7.3.2 Agdbentures: A game to learn to debug in autonomy

Debugging is an important task in software development and can be the source of a lot of frustration and time consumption. However, it is not often taught explicitly in computer science curricula even at university level. For these reasons, we developped Agdbentures (see Section 6.1.10), a debug practicing game where “levels” consist of programs containing bugs that the learner needs to debug to advance in the game.

In Agdbentures, the level programs are executed using Easytracker, which allows us to present a live visual representation of the program state during execution in the form of a 2D RPG-like world. For instance, the “player_x” and “player_y” variables in the level code are inspected at runtime and used to place a character representing the player on a graphical 2D map. The interest is three-fold: First, this makes the game appealing as the player/learner is plunged into a “real” game; Second, it showcases the importance of having information on the state of the program being executed in order to be able to do debugging; Third, it separates completely the graphical code, which can be very complex and is hidden from players, from the level code which is given to players: this allows us to simplify the source code so novice programmers won’t be rebuked. The levels share a common codebase that is increasing in size and complexity as the player advances in the game. It initially only controls the main character position, then more features are added such as interactive objects, NPCs (non playable characters), level logic (activating levers, collecting items...). This allows the player to get familiar with the codebase over time so we can present more difficult bugs which could arise in real life development. It also allows us to create “fun” levels where bugs have interesting or amusing effects on the visual representation, and where finding the solution (fixing the bugs) is rewarding.

The first experiments we conducted are very encouraging about the engagement of students at the L2 university level. All were eager to participate and declared they would really like to continue playing Agdbentures on their own with more levels. Work this year has been devoted to increase the number of levels, fix existing problems in the back-story world and the difficulty progression, as well as develop new game mechanisms that can be used in levels.

7.3.3 Active learning method in the context of large programming classes

Manuel Selva has been using for four years an active learning method in the context of large programming classes (called "Cours Magistraux" in France). This method, called scientific debate and initially cretated in the context of teaching mathematics, focuses on teaching concepts called thresholds that:

- are hard to grasp;
- change how students perceive a given discipline;
- allow making relations within a discipline
- students cannot forget them once learned

The scientific debate method involves students by having them defend their position with arguments and scales up with the number of students by leveraging collective intelligence.

From the experience we gathered during the last four years, we wrote a report that presents in detail how we apply scientific debate in a programming class. We also discuss student exchanges during debates and gather feedback after the last debate. Students report they stay more focused and motivated during class with scientific debate compared to traditional transmissive lectures. They also indicate that they understood the goal of this new pedagogical contract.

This work has been accepted for publication at ACM SIGCSE TS 2024 [4].

8 Partnerships and cooperations

8.1 International initiatives

8.1.1 Inria associate team not involved in an IIL or an international program

RV4IoT

Title: Runtime Verification for the Internet of Things

Duration: 2021 – 2024

Coordinator: Sylvain Hallé (shalle@acm.org)

Partners:

- Université du Québec à Chicoutimi Chicoutimi (Canada)

Inria contact: Ylies Falcone

Summary: The goal of the associate team is to develop theories, formal techniques, and tools based on runtime verification for the detection of security issues on connected objects, and the mitigation of potential attacks through runtime enforcement mechanisms.

8.2 National initiatives

ANR SEVERITAS

Title: Secure and Verifiable Test and Assessment System (SEVERITAS)

Duration: May 2021 – April 2025

Coordinator: Ylies Falcone

Partners:
- Laboratoire d'Informatique de Grenoble (LIG)
- Laboratoire d'Informatique, de Modélisation et d'Optimisation des Systèmes (LIMOS)
- University of Luxembourg / Interdisciplianary Center for Security, Reliability and Trust (SnT/UL)
- Laboratoire lorrain de recherche en informatique et ses applications (LORIA)

CORSE contact: Ylies Falcone

Summary: SEVERITAS advances information socio-technical security for Electronic Test and Assessment Systems (e-TAS). These systems measure skills and performances in education and training. They improve management, reduce time-to-assessment, reach larger audiences, but they do not always provide security by design. This project recognizes that the security aspects for e-TAS are still mostly unexplored. We fill these gaps by studying current and other to-be-defined security properties. We develop automated tools to advance the formal verification of security and show how to rigorously validate e-TAS security. We also develop new secure, transparent, verifiable and lawful e-TAS procedures and protocols. We also deploy novel run-time monitoring strategies to reduce frauds and study the user experience about processes to foster e-TAS usable security. And thanks to connections with players in the business of e-TAS, such as OASYS, this project will contribute to the development of secure e-TAS.

BPI OTPaaS

Title: Développement et renforcement de la filière française et européenne du Cloud

Duration: October 2021 – September 2024

Coordinator: P. Betinelli

CORSE contact: Fabrice Rastello

CORSE participants: Fabrice Rastello, Christophe Guillon

Partners: Agileo, Atos, Captronic, Duliprint, IMT, MDM, Prosyst, SE, Soben, Tridimeo, Solem, CEA, Valeo

INRIA Partners: DataMove

Summary: The OTPaaS project targets massive digitization by offering a suitable cloud for scanning that is compatible with Gaia-X and easy to use by companies including SMEs. The consortium brings together national technology providers and users from major groups and SMEs/ETIs, with strong support from major French research institutes. The platform OTPaaS will be validated by 6 demonstrators and followed by ambitious industrialization programs.

BPI DeepGreen

Title: Plateforme independante pour le deep learning embarqué

Duration: April 1st 2023 – 2027

Coordinator: CEA

CORSE contact: Fabrice Rastello

CORSE participants: Fabrice Rastello, Christophe Guillon, Hugo Pompougnac, Valentin Trophine, Guillaume Iooss

Partners: CEA, ADAGOS, PULSE AUDITION, KALRAY, DOLPHIN DESIGN, THALES RESEARCH & TECHNOLOGY FRANCE, ARCYS, MBDA, ARCELORMITTAL, EDF, SYSNAV, HAWAI.TECH, EZAKO

Summary: The DeepGreen project aims to bring together major industrial players and small and medium-sized enterprises (SMEs) in France for the deployment of Artificial Intelligence on constrained hardware targets through a software platform that meets the requirements and expectations of each stakeholder.

HOLIGRAIL – PEPR AI

Title: HOLIistic approaches to GReener model Architectures for Inference and Learning

Duration: Oct 1st 2023 – 2027

Coordinator: Olivier Sentieys

CORSE contact: Fabrice Rastello

CORSE participants: Fabrice Rastello, Christophe Guillon, Hugo Pompougnac

Partners: CEA List, TIMA

INRIA Partners: Taran, Emeraude

Summary: The vision of this action is to create a synergy with the research on the foundations of AI frugality (as proposed in SHARP action) to propose cutting-edge methods that significantly improve the energy efficiency of both inference and training of a model. We will propose (i) more compact and efficient number representations that still maintain a quality of inference or training close to the reference, (ii) hardware-aware training algorithms that enhance certain types of sparsity (e.g., more structured), coding compactness (aggressive quantization, maximum entropy) and tensor transformations. Most state-of-the-art solutions are agnostic of the hardware they run on. By taking advantage of this interplay between the hardware and the algorithms, we can achieve breakthroughs beyond current solutions, in particular by developing (iii) efficient hardware mechanisms, especially optimized to take advantage of sparsity, extreme quantization and ad-hoc number representations, together with (iv) compiler optimizations, to demonstrate the effectiveness of the proposed methods. Our approaches are holistic in the sense that they will jointly optimize the whole computing stack of AI, i.e., at the algorithm, arithmetic, compiler and hardware levels.

9 Dissemination

9.1 Promoting scientific activities

9.1.1 Scientific events: organisation

Member of the Conference Steering Committee

- Fabrice Rastello: Member of the steering Committee of ACM/IEEE CGO.

9.1.2 Scientific events: selection

Chair of conference program committees

- Fabrice Rastello: Program co-chair of track Programming Models, Compilers, and Runtime Systems, IPDPS 2024

Member of the conference program committees

- Guillaume Iooss, IDPDS 2024
- Yliès Falcone, RV 2023

Reviewer

- Manuel Selva, ACM conference on Innovation and Technology in Computer Science Education (ITiCSE 2023)
- Manuel Selva, ACM Technical Symposium on Computer Science Education (SIGCSE TS 2024)

9.1.3 Journal

Reviewer

- Guillaume Iooss, ACM TOCS

9.1.4 Invited talks

- Florent Bouchez Tichadou, Talk and Roundtable participation at "Journée enseignement de la SIF (Société Informatique de France)", May 2023
- Fabrice Rastello, Scalperf 2023: "IOUB: A tool for automatically computing and maximizing the operational intensity of affine programs. Can it be used to optimize neural networks?"

9.1.5 Leadership within the scientific community

- Fabrice Rastello: member of Inria evaluation committee (CE) since Sept 2023
- Fabrice Rastello: deputy scientific director of Inria Grenoble Rhône-Alpes (DSA) since Sept 2022
- Fabrice Rastello: scientific council of Inria Grenoble Rhône-Alpes (CoS)
- Fabrice Rastello: vice-president of the Inria CRCN/IFSP recruiting committee

9.1.6 Research administration

- Guillaume Iooss: RADAR local correspondant

9.2 Teaching - Supervision - Juries

9.2.1 Teaching

- Licence: Florent Bouchez Tichadou, Algorithms languages and programming, 118 hours, L2 UGA.
- Licence: Florent Bouchez Tichadou, Algorithms, 16 hours, L3 UGA.
- Licence: Florent Bouchez Tichadou, Programming project, 12 hours, L3 UGA.
- Master: Florent Bouchez Tichadou, Introduction to Problem-Based Learning, 4 hours, MEEF M1, UGA.
- Formation NSI, Florent Bouchez Tichadou, Teaching high-school teachers. Algorithms. 10 hours, UGA.
- License: Yliès Falcone, Languages and Automata, Univ. Grenoble Alpes, 45 hours
- Master: Yliès Falcone, is responsible for the above course.

- License 3: Manuel Selva, Imperative programming using Python, 80 hours, Grenoble Institute of Technology (Ensimag)
- License 3: Manuel Selva is responsible for the above course.
- License 3: Manuel Selva, Assembly programming, 15 hours, Grenoble Institute of Technology (Ensimag)
- License 3: Manuel Selva, Processor design, 15 hours, Grenoble Institute of Technology (Ensimag)
- License 3: Manuel Selva, C programming, 60 hours, Grenoble Institute of Technology (Ensimag)
- License 2: Guillaume Iooss, Algorithms languages and imperative programming (TD/TP), 3 hours, DLST, UGA UFR IM2AG
- License 3: Valentin Trophime-Gilotte, C programming, 30 hours, Grenoble Institute of Technology (Ensimag)

9.2.2 Supervision

- PhD: Théo Barollet, Learning programming via debugging problems in a game and visualization of the states of programs, advised by Florent Bouchez Tichadou and Fabrice Rastello, defended in November 2023.
- PhD: Nicolas Derumigny, Throughput Optimisation Techniques for Heterogeneous Architectures, advised by Fabrice Rastello and Louis-Noël Pouchet (CSU), defended in December 2023.
- PhD in progress: Théophile Bastian, Performance study: identifying bottlenecks by means of sensitivity analysis, September 2021, advised by Fabrice Rastello.
- PhD in progress: Chukri Soueidi, Instrumentation, Runtime Verification and Enforcement for Multithreaded Programs, October 2020, advised by Yliès Falcone.
- PhD: Florian Gallay, Decentralized Runtime Enforcement, October 2022, advised by Yliès Falcone. Florian halted his PhD in September 2023.
- L2 internships: Tristan Rollet, Alexis Détroyat, Nguyen Chu Hoang Anh, working on the Agdbentures project. June-July 2023.

9.2.3 Juries

Florent Bouchez

- PhD, Théo Barollet–Grenoble, Jury, Nov. 2023. Learning programming via debugging problems in a game and visualization of the states of programs.

Fabrice Rastello

- PhD, Luc Forget–Université de Lyon, Chair, June 2023. Description and compilation of ad-hoc arithmetic operators in the context of High-Level Synthesis.
- PhD, Théo Barollet–Grenoble, Jury, Nov. 2023. Learning programming via debugging problems in a game and visualization of the states of programs.
- PhD, Nicolas Derumigny–Grenoble, Jury, Dec. 2023. Throughput Optimisation Techniques for Heterogeneous Architectures.

9.3 Popularization

9.3.1 Internal or external Inria responsibilities

- Fabrice Rastello: scientific council of CEA-EDF-Inria summer schools

9.3.2 Education

- Manuel Selva, participation to the INRIA program called "1 Scientifique - 1 Classe, Chiche !" : one intervention in Lycée les Eaux Claires
- Manuel Selva, presentation of the research side of my work of enseignant-chercheur at Grenoble Institute of Technology (Ensimag)
- Manuel Selva, co-animation of a 2-days formation about an active learning technic in the context of "Parcours Enseigner dans le Supérieur (ES)" at Université Grenoble Alpes

9.3.3 Interventions

- Valentin Trophime, presentation to the seminar of the workgroups CLAP, HiFi and LVP (GDR GPL), March 2023.
- Valentin Trophime, poster to the ACACES summer school, July 2023.
- Guillaume Iooss, presentation of the IO Complexity work to the L3 students of ENS Saclay, November 2023.
- Manuel Selva, presentation of EasyTracker at the CITI laboratory from INSA de Lyon

10 Scientific production

10.1 Publications of the year

International journals

[1] C. Soueidi, M. Monnier and Y. Falcone. 'Efficient and expressive bytecode-level instrumentation for Java programs'. In: *International Journal on Software Tools for Technology Transfer* 25.4 (29th June 2023), pp. 453–479. DOI: 10.1007/s10009-023-00708-z. URL: https://inria.hal.science/hal-04381736.

Invited conferences

[2] H. Al-Qadasi, Y. Falcone and S. Bensalem. 'DeepAbstraction++: Enhancing Test Prioritization Performance via Combined Parameterized Boxes ⋆'. In: AISOLA - Bridging the Gap Between AI and Reality. Crete, Greece, 23rd Oct. 2023. URL: https://inria.hal.science/hal-04380870.

International peer-reviewed conferences

[3] T. Barollet, C. Guillon, M. Selva, F. Broquedis, F. Bouchez-Tichadou and F. Rastello. 'EasyTracker: A Python Library for Controlling and Inspecting Program Execution'. In: *International Symposium on Code Generation and Optimization (CGO)*. International Symposium on Code Generation and Optimization (CGO). Edinburgh, United Kingdom, 2nd Mar. 2024. URL: https://inria.hal.science/hal-04368835.

[4] M. Selva and F. Broquedis. 'Mining Jewels Together: Debating about Programming Threshold Concepts in Large Classes'. In: *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 1 (SIGCSE 2024)*. Proceedings of the 55th ACM Technical Symposium on Computer Science Education (SIGCSE 2024). Portland (OR), United States, 20th Mar. 2024. DOI: 10.1145/3626252.3630893. URL: https://inria.hal.science/hal-04383009.

[5] C. Soueidi and Y. Falcone. 'Bridging the Gap: A Focused DSL for RV-Oriented Instrumentation with BISM'. In: International Conference on Runtime Verification. Vol. 14245. Lecture Notes in Computer Science. Thessalokini, Greece: Springer Nature Switzerland, 1st Oct. 2023, pp. 327–338. DOI: 10.1007/978-3-031-44267-4_17. URL: https://inria.hal.science/hal-04381683.

[6] C. Soueidi and Y. Falcone. 'Instrumentation for RV: From Basic Monitoring to Advanced Use Cases'. In: International Conference on Runtime Verification. Vol. 14245. Lecture Notes in Computer Science. Thessaloniki, Greece: Springer Nature Switzerland, 1st Oct. 2023, pp. 403–427. DOI: 10.1007/978-3-031-44267-4_23. URL: https://inria.hal.science/hal-04381696.

[7] C. Soueidi and Y. Falcone. 'Sound Concurrent Traces for Online Monitoring'. In: International Symposium on Model Checking Software. Vol. 13872. Lecture Notes in Computer Science. Paris, France: Springer Nature Switzerland, 2nd May 2023, pp. 59–80. DOI: 10.1007/978-3-031-32157-3_4. URL: https://inria.hal.science/hal-04381666.

[8] C. Soueidi, Y. Falcone and S. Hallé. 'Dynamic Program Analysis with Flexible Instrumentation and Complex Event Processing'. In: 2023 IEEE 34th International Symposium on Software Reliability Engineering (ISSRE). Florence, Italy: IEEE, 9th Oct. 2023, pp. 742–751. DOI: 10.1109/ISSRE59848.2023.00048. URL: https://inria.hal.science/hal-04381709.

[9] C. Soueidi, A. El-Hokayem and Y. Falcone. 'Opportunistic Monitoring of Multithreaded Programs'. In: Fundamental Approaches to Software Engineering. Vol. 13991. Lecture Notes in Computer Science. Paris, France: Springer Nature Switzerland, 20th Apr. 2023, pp. 173–194. DOI: 10.1007/978-3-031-30826-0_10. URL: https://inria.hal.science/hal-04381611.

Reports & preprints

[10] E. Delort, L. Riou and A. Srivastava. *Environmental Impact of Artificial Intelligence: Bibliographic Report - Artificial Intelligence and Eco-responsibility Internship*. INRIA; CEA Leti, Sept. 2023, pp. 1–33. URL: https://inria.hal.science/hal-04283245.

[11] D. Potop-Butucaru, A. Cohen, G. Plotkin and H. Pompougnac. *Bidirectional Reactive Programming for Machine Learning*. 2023. DOI: 10.48550/arXiv.2311.16977. URL: https://inria.hal.science/hal-04354071.