

2025 Activity Report

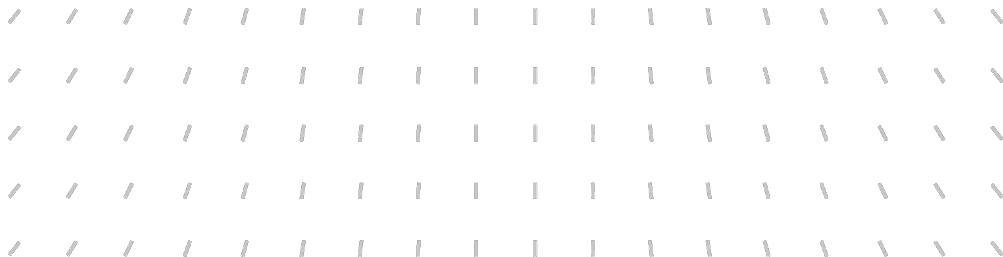
RESEARCH CENTRE: Inria Paris Centre


Project-Team

WHISPER

Well Honed Infrastructure Software for Programming
Environments and Runtimes





Project-Team WHISPER

Creation of the Project-Team: 2023 July 01

Each year, Inria research teams publish an Activity Report presenting their work and results over the reporting period. These reports follow a common structure, with some optional sections depending on the specific team. They typically begin by outlining the overall objectives and research programme, including the main research themes, goals, and methodological approaches. They also describe the application domains targeted by the team, highlighting the scientific or societal contexts in which their work is situated. The reports then present the highlights of the year, covering major scientific achievements, software developments, or teaching contributions. When relevant, they include sections on software, platforms, and open data, detailing the tools developed and how they are shared. A substantial part is dedicated to new results, where scientific contributions are described in detail, often with subsections specifying participants and associated keywords. Finally, the Activity Report addresses funding, contracts, partnerships, and collaborations at various levels, from industrial agreements to international cooperations. It also covers dissemination and teaching activities, such as participation in scientific events, outreach, and supervision. The document concludes with a presentation of scientific production, including major publications and those produced during the year.

Keywords

Computer sciences and digital sciences

- A1. – Architectures, systems and networks
 - A1.1.1. – Multicore, Manycore
 - A1.1.3. – Memory models
 - A1.1.13. – Virtualization
 - A2.1.6. – Concurrent programming
 - A2.1.10. – Domain-specific languages
 - A2.2.1. – Static analysis
 - A2.2.5. – Run-time systems
 - A2.2.8. – Code generation
 - A2.3.1. – Embedded systems
 - A2.5. – Software engineering
 - A2.5.4. – Software Maintenance & Evolution
 - A2.6.1. – Operating systems
 - A2.6.2. – Middleware
 - A2.6.3. – Virtual machines
- A4.5. – Formal method for verification, reliability, certification
 - A4.5.3. – Program proof

Other research topics and application domains

- B5. – Industry of the future
 - B5.2.1. – Road vehicles
 - B5.2.3. – Aviation
 - B5.2.4. – Aerospace
- B6.1. – Software industry
 - B6.1.1. – Software engineering
 - B6.1.2. – Software evolution, maintenance
- B6.5. – Information systems
- B6.6. – Embedded systems

Contents

Project-Team WHISPER	1
1 Team members, visitors, external collaborators	5
2 Overall objectives	6
3 Research program	6
3.1 System performance	6
3.2 System re-engineering	7
4 Application domains	8
4.1 Linux	8
4.2 Device Drivers	8
4.3 Multicore computing	9
5 Social and environmental responsibility	9
5.1 Footprint of research activities	9
6 Latest software developments, platforms, open data	9
6.1 Latest software developments	9
6.1.1 Coccinelle	9
6.1.2 Coccinelle for C++	9
6.1.3 Coccinelle for Rust	10
6.1.4 schedgraph	10
6.2 Open data	10
7 New results	10
7.1 FlexGuard: Fast Mutual Exclusion Independent of Subscription	11
7.2 Scheduler Guided OpenMP Execution in Cloud VMs	11
7.3 Linux as a microkernel: the memory management's case	12
7.4 Tapestry: Revealing Wait-For Dependencies Between Application Threads	12
7.5 The Impact of Kernel Asynchronous APIs on the Performance of a Kernel VPN	12
7.6 Understanding Linux Kernel Code through Formal Verification: A Case Study of the Task-Scheduler Function <code>select_idle_core</code>	12
7.7 Advances in Semantic Patching for HPC-oriented Refactorings with Coccinelle	13
8 Bilateral contracts and grants with industry	13
9 Partnerships and cooperations	14
9.1 National initiatives	14
9.1.1 ANR	14
10 Dissemination	15
10.1 Promoting scientific activities	15
10.1.1 Scientific events: organisation	15
10.1.2 Scientific events: selection	16
10.1.3 Journal	16
10.1.4 Invited talks	16
10.1.5 Leadership within the scientific community	16
10.1.6 Scientific expertise	17
10.2 Teaching - Supervision - Juries - Educational and pedagogical outreach	17
10.2.1 Supervision	17
10.2.2 Juries	17
10.3 Popularization	17

10.3.1 Specific official responsibilities in science outreach structures	17
10.3.2 Other science outreach relevant activities	17
11 Scientific production	18
11.1 Major publications	18
11.2 Publications of the year	19
11.3 Cited publications	19

1 Team members, visitors, external collaborators

Research Scientists

- Julia Lawall [Team leader, INRIA, Senior Researcher, HDR]
- Yu Liang [INRIA, Researcher, from Oct 2025]
- Jean-Pierre Lozi [INRIA, Researcher]

Post-Doctoral Fellows

- Tomas Faltin [INRIA, Post-Doctoral Fellow, until Aug 2025]
- Xutong Ma [INRIA, Post-Doctoral Fellow]

PhD Students

- Maxime Derri [ORANGE]
- Papa Assane Fall [INRIA, until Sep 2025]
- Thomas Fourier [Université Grenoble Alpes, from Oct 2025]
- Victor Laforet [INRIA]
- Keisuke Nishimura [INRIA]
- Himadri Pandya [INRIA, until Nov 2025]

Technical Staff

- Victor Gambier [INRIA, Engineer]

Interns and Apprentices

- Gaspard Deremble [INRIA, Intern, from Jun 2025 until Aug 2025]
- Soulayman Jbari [TELECOM SUDPARIS, from Nov 2025]
- Corinn Tiffany [TELECOM SUDPARIS, Intern, from Sep 2025]

Administrative Assistants

- Martial Le Henaff [INRIA]
- Nelly Maloysel [INRIA]

Visiting Scientists

- Tomas Faltin [UNIV CHARLES - PRAGUE, from Nov 2025]
- Tathagata Roy [VU Amsterdam, from Aug 2025 until Aug 2025]

2 Overall objectives

The focus of Whisper is on how to develop (new) and improve (existing) infrastructure software. Infrastructure software (also called systems software) is the software that underlies all computing. Such software allows applications to access resources and provides essential services such as memory management, synchronization and inter-process interactions. Starting bottom-up from the hardware, examples of infrastructure software include operating systems, virtual machine hypervisors, managed runtime environments, and standard libraries. For such software, efficiency and correctness are fundamental. Any overhead will impact the performance of all supported applications. Any failure will prevent the supported applications from running correctly. Whisper addresses these dual problems of infrastructure software performance and correctness, both in a given instance of the software and as the software evolves.

In terms of methodology, Whisper is at the interface of the domains of operating systems, software engineering and programming languages. Our approach is to combine the study of problems in the development of real-world infrastructure software with concepts in programming language design and implementation, *e.g.*, of domain-specific languages, and knowledge of low-level system behavior. A focus of our work is on providing support for legacy code, while taking the needs and competences of ordinary system developers into account. We will put an emphasis on achieving measurable improvements in performance and correctness in practice, and on feeding these improvements back to the infrastructure-software developer community.

We focus on two main axes. The first axis, system performance, targets the performance of applications, in both bare-metal and virtual environments, based on the impact of operating-system services. This direction not only involves proposing new algorithms, but also designing tools and methodologies to help developers understand the system behavior. When addressing these areas, we will explore the use of DSLs [3, 6, 8, 9, 10], when appropriate, to enhance the usability, configurability, reliability, and robustness of the proposed approaches. The second axis, system re-engineering, targets the quality of legacy infrastructure software. Infrastructure software tends to evolve faster than many of its users are able to keep up with, leading to massive ecosystem fragmentation. Building on tools such as Coccinelle for automating widespread code changes [1, 4, 13], we will investigate how to facilitate and improve the reliability of the maintenance of such variants, for example in terms of identifying, adapting, and applying relevant bug fixes.

3 Research program

The research program of the Whisper team is designed around two main axes: system performance and system re-engineering.

3.1 System performance

Our work on system performance focuses on scheduling, virtualization, synchronization, and the interactions between them. A task scheduler decides which runnable task should be able to execute on a CPU, and, in a multicore setting, on which core that execution should take place. As the task scheduler determines how much CPU time the tasks of an application receive, as well as whether the chosen CPU has efficient access to needed hardware resources (network, accelerators, caches, etc.), it has a large impact on application performance. In a general-purpose programming setting, the task scheduler is typically located in the operating system kernel, and is expected to serve the needs of all applications. It has no knowledge of future application behavior. Such a setting raises many challenges. We are currently focusing on four main issues: scheduling and virtualization, making locks aware of scheduler preemptions, control over scheduling from the user level, and task scheduling for heterogeneous architectures.

Virtualization decomposes the system support into one or more standard operating systems, known as guest OSes, that in turn run on a minimalistic operating system, known as a hypervisor, which provides isolation between the guest OSes and provides access to the hardware. A guest OS schedules application tasks based on its view of the set of available CPUs, but these CPUs are actually just tasks that are in turn scheduled by the hypervisor on the physical CPUs of the machine. This dual level of scheduling introduces the risk of contradictory scheduling strategies. We are investigating the impact of this dual level of scheduling on the performance of applications in a highly multicore setting.

Efficient lock algorithms are key to ensuring application scalability on multi-core architectures [25, 28], [7]. Two families of lock algorithms exist: (1) with blocking locks, tasks sleep in a wait list until the lock is free, whereas (2) with spin locks, tasks busy-wait on the value of an atomic boolean variable that records whether or not the lock is free. While blocking locks consume less CPU, they are also less reactive, as acquiring the lock requires waking up the acquiring task, which entails a context switch. Spin locks, on the other hand, are very reactive, as the lock is acquired by a busy-waiting task as soon as the lock is released. But spinlocks may perform very poorly in overloaded scenarios, because spinning tasks can preempt the lock-holding task, preventing any progress on the critical path. We would like to turn spin locks into blocking locks in an overloaded scenario, relying on BPF to safely inject code into the kernel to detect when the Linux kernel scheduler preempts the lock holder.

Recently, several frameworks have been developed (notably, ghOSt from Google [30] and sched-ext from Meta [23]) that allow the user level to inject new schedulers into the Linux kernel. We would like to investigate how such a framework can be used to improve performance, using scheduling-decision-making processes that are not feasible in the kernel. For example, machine learning could be beneficial for inferring task properties and guiding scheduling accordingly, but is not practical in the kernel, due to the latency involved, memory constraints, and the lack of kernel support for floating point numbers. Our goal would be to provide a proof-of-concept, as it is not our focus to develop new machine-learning algorithms. We will also consider whether it can be useful to export other kinds of operating system services to user level, such as memory management. Finally, we will consider whether it is possible to design a generic kernel interface for exporting to the user level performance-critical operating-system services. This work is in collaboration with Alain Tchana and Renaud Lachaize of the Erods team at LIG, as part of the Inria Défi OS.

The heuristics for scheduling employed by the Linux kernel have evolved over time, primarily in a homogeneous setting, where all cores have similar properties. Today, heterogeneous computing is becoming more prevalent, combining powerful processors for compute-intensive tasks with slower, more energy-efficient, processors for lighter weight tasks. Such an architecture is found in the big.LITTLE processors from Arm, typically used in mobile environments, and is now moving to desktop computing, with the Alder Lake architecture from Intel. We will assess the degree to which the Linux kernel task scheduler is able to effectively support such architectures and propose improvements, in terms of both meeting application performance requirements and saving energy. This work is in collaboration with David Bromberg and Djob Mvondo of the WIDE team at Inria-Rennes, as part of the Inria Défi OS.

3.2 System re-engineering

In the area of system re-engineering, we are interested in improving the quality of infrastructure software as well as in automating the transformations required to make it possible to use such software in new settings. Our work builds on our previous work on Coccinelle [4], a program transformation system for C code that has been extensively used on the Linux kernel and other C infrastructure software.

We are extending Coccinelle to C++ and to Rust. The extension to C++ reuses the existing Coccinelle implementation, simply adding the C++-specific constructs. The goal of this work, in collaboration with Michele Martone of the Leibniz Supercomputing Center, is to consider how Coccinelle can be used to adapt existing HPC software for use on GPUs, by introducing the use of calls to relevant libraries. The extension to Rust, on the other hand, is a complete re-implementation of Coccinelle, to benefit from tools being developed by the Rust community for parsing and pretty printing (Rust Analyzer and rustfmt) Rust code. The reuse of these tools has made it possible to quickly develop a preliminary implementation for the full Rust language. In the future, we may consider how to generalize this process to other languages, addressing the challenge that generic parsing tools may not present information in a way that is suitable for program transformation, for example in the choice of non-terminals.

We are considering how Coccinelle can be used to facilitate long-term software maintenance. Many software projects are obliged to maintain multiple variants, *e.g.*, to meet the demands of users who rely on the properties of old variants. An essential part of software maintenance is thus to apply new bug fixes to these old variants. Intervening changes in the software, however, may imply that fixes can not be applied to older versions directly. We are looking into how Coccinelle can be used to collect and exploit information about recent code changes, to adapt new bug fixes to older source code.

Today, there are numerous tools for scanning infrastructure software for common types of vulnerabilities, such as NULL pointer dereferences and buffer overflows. Avoiding such issues, however, is not sufficient to

ensure that the code behaves correctly; the code must also implement the intended algorithm. Checking algorithmic properties, however, is much more challenging, because each service implements its own algorithm. Formal verification tools such as Frama-C [24] can make it possible to express and verify such properties, but remain challenging to use, particularly for developers without experience in formal verification. The challenge is compounded by the fact that the code may evolve frequently, making any proofs quickly out of date. We are investigating how to design tools that can facilitate proofs about Linux kernel code, both in how to extract relevant parts of the code base, to reduce the amount of code that has to be considered by the proving process, and in how proofs can be evolved in response to changes in the code base, analogous to the adaptation of bug fixes over multiple versions.

4 Application domains

4.1 Linux

Linux is an open-source operating system that is used in settings ranging from embedded systems to supercomputers. Linux kernel v6.1, released in December 2022, comprises over 23 million lines of code, and supports 23 different families of CPU architectures, around 50 file systems, and thousands of device drivers. Linux is also in a rapid stage of development, with new versions being released roughly every 2.5 months. Recent versions have each incorporated around 13,500 commits, from around 1500 developers. These developers have a wide range of expertise, with some providing hundreds of patches per release, while others have contributed only one. Overall, the Linux kernel is critical software, but software in which the quality of the developed source code is highly variable. These features, combined with the fact that the Linux community is open to contributions and to the use of tools, make the Linux kernel an attractive target for software researchers. Tools that result from research can be directly integrated into the development of real software, where it can have a high, visible impact.

Starting from the work of Engler *et al.* [27], numerous research tools have been applied to the Linux kernel, typically for finding bugs [26, 33, 36, 40] or for computing software metrics [31, 46]. In our work, we have studied generic C bugs in Linux code [12], bugs in function protocol usage [34, 35], issues related to the processing of bug reports [44] and crash dumps [29], and the problem of backporting [41, 45], illustrating the variety of issues that can be explored on this code base. Unique among research groups working in this area, we have furthermore developed numerous contacts in the Linux developer community. These contacts provide insights into the problems actually faced by developers and serve as a means of validating the practical relevance of our work.

4.2 Device Drivers

Device drivers are essential to modern computing, to provide applications with access, via the operating system, to physical devices such as keyboards, disks, networks, and cameras. Development of new computing paradigms, such as the internet of things, is hampered because device driver development is challenging and error-prone, requiring a high level of expertise in both the targeted OS and the specific device. Furthermore, implementing just one driver is often not sufficient; today's computing landscape is characterized by a number of OSes, *e.g.*, Linux, Windows, MacOS, BSD and many real time OSes, and each is found in a wide range of variants and versions. All of these factors make the development, porting, backporting, and maintenance of device drivers a critical problem for device manufacturers, industry that requires specific devices, and even for ordinary users.

The last twenty years have seen a number of approaches directed towards easing device driver development. Réveillère *et al.* propose Devil [39], a domain-specific language for describing the low-level interface of a device. Chipounov *et al.* propose RevNic, [22] a template-based approach for porting device drivers from one OS to another. Ryzhyk *et al.* propose Termite, [42, 43] an approach for synthesizing device driver code from a specification of an OS and a device. Currently, these approaches have been successfully applied to only a small number of toy drivers. Indeed, Kadav and Swift [32] observe that these approaches make assumptions that are not satisfied by many drivers; for example, the assumption that a driver involves little computation other than the direct interaction between the OS and the device. At the same time, a number of tools have been developed for finding bugs in driver code. These tools include SDV [21], Coverity [27],

CP-Miner, [37] PR-Miner [38], and Coccinelle [11]. These approaches, however, focus on analyzing existing code, and do not provide guidelines on structuring drivers.

In summary, there is still a need for a methodology that first helps the developer understand the software architecture of drivers for commonly used operating systems, and then provides tools for the maintenance of existing drivers.

4.3 Multicore computing

Today, multicore computing is fundamental across a wide variety of industries. Applications such as machine learning, scientific computing, image processing, etc., run large numbers of threads on highly multicore processors. Such applications must furthermore contend with a variety of hardware architectures, with different CPU and memory topologies, CPU capacities, access to external resources such as storage, networking and accelerators, etc. Virtualization facilitates hardware sharing, but complicates access to and reasoning about hardware resources. Our work on system performance targets helping multicore applications run more efficiently, and helping developers understand their application performance, to enable them to find appropriate solutions.

5 Social and environmental responsibility

5.1 Footprint of research activities

Most of our resource-intensive research activities are carried out on Grid 5000, which is a shared national infrastructure, saving the cost of building, transporting, and continuously running a variety of dedicated machines.

6 Latest software developments, platforms, open data

6.1 Latest software developments

6.1.1 Coccinelle

Keywords: Code quality, Evolution, Infrastructure software

Functional Description: Coccinelle is a tool for code search and transformation for C programs. It has been extensively used for bug finding and evolutions in Linux kernel code. Extensions to support C++ and Rust are in progress. A prototype has been developed for Java.

URL: <https://coccinelle.gitlabpages.inria.fr/website/>

Contact: Julia Lawall

Participants: Victor Gambier, Julia Lawall

Partner: IRILL

6.1.2 Coccinelle for C++

Keywords: C++, Program transformation

Functional Description: Coccinelle is a tool for program matching and transformation, relying on rules expressed as fragments of source code, amounting to a semantic patch. Coccinelle for C++ extends Coccinelle to handle software written in the C++ programming language. Coccinelle for C++ is being designed to target high-performance computing (HPC) software.

URL: <https://coccinelle.gitlabpages.inria.fr/website/>

Publications: [hal-01890314](#), [hal-03266521](#)

Contact: Julia Lawall

Participants: Michele Martone, Julia Lawall

6.1.3 Coccinelle for Rust

Keywords: Rust, Program transformation

Functional Description: Coccinelle is a tool for program matching and transformation, relying on rules expressed as fragments of source code, amounting to a semantic patch. Coccinelle for Rust brings the power of Coccinelle to software written in the Rust programming language. Just as Coccinelle was designed around the practical needs for software evolution in the Linux kernel, Coccinelle for Rust has benefited from feedback from the Rust for Linux community.

URL: <https://gitlab.inria.fr/coccinelle/coccinelleforrust>, <https://rust-for-linux.com/coccinelle-for-rust>

Contact: Julia Lawall

Participants: Julia Lawall, Tathagata Roy

6.1.4 schedgraph

Keywords: Task scheduling, Linux kernel

Functional Description: Schedgraph is a collection of tools for visualizing Linux kernel scheduling traces.

URL: <https://gitlab.inria.fr/schedgraph/schedgraph>

Publication: [hal-04001993](https://hal.archives-ouvertes.fr/hal-04001993)

Contact: Julia Lawall

6.2 Open data

- The artifact associated with the paper "FlexGuard: Fast Mutual Exclusion Independent of Subscription", published at SOSP 2025 [15], is available at <https://gitlab.inria.fr/flexguard>.
- The artifact associated with the paper "Tapestry: Revealing Wait-For Dependencies Between Application Threads", published at PLOS2025 [14], is available at <https://gitlab.inria.fr/tapestry>.
- The artifact associated with the paper "Understanding Linux Kernel Code through Formal Verification: A Case Study of the Task-Scheduler Function `select_idle_core`", published at OlivierFest 2025 [16], is available at <https://zenodo.org/records/16842821>.

7 New results

The main publications of 2025 were in the area of improving and understanding system performance, covering optimized locks, scheduling in the context of virtualization, understanding the use of user-level spin locks, and improving the performance of the VPN Wireguard. We have also continued our work on verification of Linux kernel code and the improvement of HPC source code using Coccinelle.

7.1 FlexGuard: Fast Mutual Exclusion Independent of Subscription

Participants: Victor Laforet (*Whisper*), Sanidhya Kashyap (*EPFL*), Călin Iorgulescu (*Oracle Labs*), Julia Lawall (*Whisper*), Jean-Pierre Lozi (*Whisper*).

Performance-oriented applications require efficient locks to harness the computing power of multicore architectures. While fast, spinlock algorithms suffer severe performance degradation when thread counts exceed available hardware capacity, i.e., in oversubscribed scenarios. Existing solutions rely on imprecise heuristics for blocking, leading to suboptimal performance. We present FlexGuard, the first approach that systematically switches from busy-waiting to blocking precisely when a lock-holding thread is preempted. FlexGuard achieves this by communicating with the OS scheduler via eBPF, unlike prior approaches. FlexGuard matches or improves performance in LevelDB, a memory-optimized database index, PARSEC's Dedup, and SPLASH2X's Raytrace and Streamcluster, boosting throughput by 1-6x in nonoversubscribed and up to 5x in oversubscribed scenarios.

This paper was published at SOSP 2025 [15]. It has received badges for "Artifacts Available", "Artifacts Evaluated and Functional", and "Results Reproduced".

7.2 Scheduler Guided OpenMP Execution in Cloud VMs

Participants: Himadri Chhaya-Shailesh (*Whisper*).

OpenMP is a widely used framework for parallelizing applications, enabling thread-level parallelism via simple source-code annotations. It follows the fork-join model and relies heavily on barrier synchronization among worker threads. Running OpenMP-enabled applications in the cloud is increasingly popular due to elasticity, fast startup, and pay-as-you-go pricing.

In cloud-based execution, worker threads run inside a virtual machine (VM) and are subject to dual levels of scheduling: threads are placed by the guest scheduler on guest virtual CPUs (vCPUs), and vCPUs are managed as ordinary tasks by the host scheduler on the host's physical CPUs (pCPUs). Because the guest and host schedulers act independently, a semantic gap emerges that can undermine application performance. Barrier synchronization, whose efficiency depends on timely scheduling decisions, is vulnerable to this semantic gap, and remains underexplored.

Host-level preemptions in which guest vCPUs remain queued on busy pCPUs, stall application progress. We show that performance can be substantially improved by (1) dynamically adapting the degree of parallelism (DoP) at the start of each parallel region and (2) dynamically choosing between spinning versus blocking at barriers on a per-thread, per-barrier basis. We propose paravirtualized, scheduler-informed techniques that accurately guide these decisions and demonstrate their effectiveness in realistic deployments.

The main contributions of this thesis are summarized below:

1. A novel analysis of how the semantic gap between the guest and the host schedulers influences OpenMP applications, facilitated by our in-house host-guest scheduler tracing and visualization tool, `dispel_tracer`.
2. Phantom Tracker, an algorithmic solution that leverages paravirtualized task scheduling to detect and quantify Phantom vCPUs accurately.
3. `pv-barrier-sync`, a dynamic barrier synchronization mechanism driven by the scheduler insights produced by Phantom Tracker.
4. Juunansei, an OpenMP runtime extension that demonstrates the practical utility of Phantom Tracker and `pv-barrier-sync` with additional optimizations.

This thesis was defended on December 17, 2025.[19]

7.3 Linux as a microkernel: the memory management's case

Participants: Papa Assane Fall (*Whisper*).

Main memory is a critical resource in data centers, due to its major impact on application performance and server costs. However, Linux's memory management system, designed to be general-purpose, is not always optimal for diverse workload requirements. This thesis [20] introduces USM, a framework for the rapid development of memory management policies in Linux. USM adopts a microkernel-inspired design, enabling memory-management policies to run in user space. We demonstrate the flexibility of USM by implementing a variety of memory-management policies, and evaluate its performance against state-of-the-art solutions.

This work was carried out as part of the Defi OS. It was supervised by Alain Tchana in Grenoble and Jean-Pierre Lozi in Paris.

7.4 Tapestry: Revealing Wait-For Dependencies Between Application Threads

Participants: Tomáš Faltin (*Whisper*), Himadri Chhaya-Shailesh (*Whisper*), Julia Lawall (*Whisper*), Jean-Pierre Lozi (*Whisper*).

Application slowdowns that accumulate on the critical path significantly impact performance. Identifying them requires visualizing inter-thread wait-for dependencies, a task neglected by existing tools, especially regarding ad-hoc, busywaiting synchronization. We present Tapestry, which efficiently traces these dependencies by dynamically switching between hardware watchpoints and software breakpoints. Using three use cases, we show that Tapestry is able to reveal NUMA effects and explain performance anomalies in oversubscribed and virtualized environments.

This work was published at the 13th Workshop on Programming Languages and Operating Systems (PLOS 2025), held with SOSP 2025 [14].

7.5 The Impact of Kernel Asynchronous APIs on the Performance of a Kernel VPN

Participants: Cesaire Mounah-Honore (*WIDE*), Djob Mvondo (*WIDE*), Julia Lawall (*Whisper*), Yérom-David Bromberg (*WIDE*).

Linux kernel VPNs suffer from severe performance degradation under high load due to execution order inversion (EoI), a phenomenon where packet recombination functions preempt earlier pipeline stages. This leads to severe latency spikes and throughput reductions. We investigate kernel threads and workqueues as alternative kernel asynchronous APIs to address these limitations, achieving up to a 4.7x increase in throughput while reducing tail latency by 65%. These results demonstrate the importance of selecting appropriate kernel asynchronous APIs for kernel-level network applications.

This short paper was published at the 2025 ACM International Systems and Storage Conference (SYSTOR) [18]

7.6 Understanding Linux Kernel Code through Formal Verification: A Case Study of the Task-Scheduler Function `select_idle_core`

Participants: Julia Lawall (*Whisper*), Keisuke Nishimura (*Whisper*), Jean-Pierre Lozi (*Whisper*).

On the one hand, the Linux kernel task scheduler is critical to all application performance. On the other hand, it is widely agreed that its code complexity is spiraling out of control, and only a tiny handful of kernel developers understand it. We are exploring the opportunities and challenges in applying formal verification to Linux kernel task-scheduler code. Building on a previous work focusing on the evolution of the function `should_we_balance`, we here consider a version of the key task-placement function `select_idle_core` and the evolution of the iterators on which it relies.

This work was published at the OlivierFest 2025 workshop, held with SPLASH 2025 [16]. It received an "Artifacts Available" badge.

7.7 Advances in Semantic Patching for HPC-oriented Refactorings with Coccinelle

Participants: Michele Martone (*LRZ*), Julia Lawall (*Whisper*).

Currently, the most energy-efficient hardware platforms for large-scale floating point-intensive calculations (also known as High Performance Computing, or HPC) are graphical processing units (GPUs). However, porting existing scientific codes to GPUs can be far from trivial. This article summarizes our recent advances in enabling machine-assisted, HPC-oriented refactorings with reference to existing APIs and programming idioms available in C and C++. The tool we are extending and using for the purpose is called Coccinelle. An important workflow we aim to support is that of writing and maintaining tersely written application code, while deferring circumstantial, ad-hoc, performance-related changes to specific, separate rules called "semantic patches". GPUs currently offer very limited debugging facilities. The approach we are developing aims at preserving intelligibility, longevity, and relatedly, debuggability of existing code on CPUs, while at the same time enabling HPC-oriented code evolutions such as introducing support for GPUs, in a scriptable and possibly parametric manner. This article sketches a number of self-contained use cases, including further HPC-oriented cases which are independent from GPUs.

This work was published at the 30th International Workshop on High-Level Parallel Programming Models and Supportive Environments (HIPS), held with IPDPS [17].

8 Bilateral contracts and grants with industry

Extension of the Linux kernel with safe user programs (CIFRE)

Participants: Julia Lawall (*Whisper*), Kahina Lazri (*Orange Labs*), Maxime Derri (*Orange Labs*).

The operating system kernel represents a privileged point of observation, which enables collecting data both at the infrastructure level and at the application level. The kernel also provides an observation point for both network and system activities. The language eBPF (extended Berkeley Packet Filter) has revolutionized observation practices by enabling observation programs provided by user-space to be executed at the Linux kernel level. The eBPF technology comprises multiple components, including helpers, maps and verification. Together, these components allow eBPF programs to benefit from a wide access to kernel functions, while at the same time guaranteeing safety.

Several network and system management services have taken advantage of the programmability offered by eBPF to enhance infrastructures with supervision, tracing and security services that run natively at the kernel level. This ensures high performance and a high degree of visibility of application activity. The criticality of the kernel nevertheless restricts the instructions that an eBPF program can execute. The component responsible for verifying program safety is the eBPF verifier. The verifier's objective is to ensure that the programs to be executed by the kernel are safe. This, however, has the effect of significantly reducing the expressiveness offered by eBPF by restricting certain programming mechanisms, such as variable-size loops, or by limiting the number of instructions a given program can execute. These constraints complicate the development of eBPF security applications, such as http traffic processing or managing TLS connections.

The aim of this project is to reconsider the design of program verification as implemented by the eBPF verifier in order to improve the robustness of verification on the one hand, and to increase the expressiveness of programs on the other. The results of this project are expected to compare the verification choices adopted by the Linux community with the state of the art of formal verification on the one hand, and with that of so-called safe programming languages such as Rust, in order to propose an improvement in the usability of the eBPF infrastructure. A subsidiary result of the project would be to reduce the need for helpers (Linux kernel functions).

9 Partnerships and cooperations

9.1 National initiatives

9.1.1 ANR

EMASS

Participants: Matthieu Lemerre (*CEA, EMASS PI*), Sébastien Bardin (*CEA*), Frédéric Recoules (*CEA*), Xavier Rival (*Antique (Inria PI)*), Julia Lawall (*Whisper*), Keisuke Nishimura (*Whisper*), Kosmatov Nikolai (*Thales RT*), Delphine Longuet (*Thales RT*), Romain Soulat (*Thales RT*).

- Awarded in 2023, duration 42 months (2023 - 2026).
- Members: Inria (Antique, Whisper), CEA, Thales RT
- Funding: ANR, 162,720 euros awarded to Inria.
- Objectives: Current systems programs, like OS kernels, hypervisors, or core libraries found in the bottom layer of every application stack, are today mostly written in systems programming languages, such as C, C++, or assembly, that give programmers low-level control over resource management at the expense of safety. This leads to frequent memory corruption errors in systems programs, which is one of the major cybersecurity issues today. Backed by the consortium's strong experience in advanced memory analyses (shape analyses), source and binary-level program analysis for security, software engineering by scalable static analysis on industrial case studies, and OS kernel verification, we want to provide an effective solution to this problem, even for the most challenging systems software, i.e., OS kernels and hypervisors.

The EMASS project targets a solution to this problem relying on a static analysis of memory, using types as a base abstraction. Our goal is to be able to automatically verify the most challenging systems code, by developing a scalable compositional analysis, and by tackling other important security properties such as non-interference between the tasks of the OS, or functional contracts on the OS system calls, with the guidance of a low amount of intuitive type annotations. The project will result in strong scientific results in type-based static analysis, and in the EMASS toolkit, a pragmatic open-source solution for verifying and certifying secure systems software without formal methods expertise, whose practical use will be validated in challenging industrial case studies.

The contribution of the Whisper team will focus on using our tools such as Coccinelle [4] and Prequel [5] to find challenging case studies, and on developing strategies for continuous verification of systems software.

DiVa: DIssaggregated VirtuAlization

Participants: Gaël Thomas (*Inria Benagil (DiVa PI)*), Mathieu Bacou (*Inria Benagil*), Vivien Quema (*Grenoble INP*), Jean-Pierre Lozi (*Whisper*), Julia Lawall (*Whisper*), Alain Tchana (*Grenoble INP*), Daniel Hagi-mont (*INP Toulouse*), Boris Teabe (*INP Toulouse*), Julien Sopena (*Sorbonne University*).

- Awarded in 2023, duration 7 years (2023-2030)
- Members: Inria (Benagil, Whisper), Grenoble INP, INP Toulouse, Sorbonne University
- Funding: ANR-PEPR, 321,839 euros.
- Objectives: Cloud infrastructures are currently undergoing major changes with the advent of disaggregated and edge clouds. The objective of the DiVA (DISaggregated VirtuAlization) project is to prepare the system stack for these next generations of cloud infrastructures. In order to use these infrastructures more efficiently and to avoid wasting hardware resources, we propose to revisit the system mechanisms at the basis of any cloud infrastructure.

In the context of disaggregated infrastructures, it becomes possible, at the scale of a few clusters of machines, to fully mutualize hardware resources. In detail, using the recent CXL protocol, any processor on any machine in the cluster can transparently access any hardware resource (memory, network, disk, accelerator) on any other server. This evolution calls for new hypervisors and operating system designs because virtual machines will become elastic (i.e., it will be possible to add new resources to a virtual machine at runtime) and distributed (i.e., the hardware resources used by a virtual machine are physically distributed). In the DiVA project, we will therefore (i) define new virtualization interfaces in order to allow a virtual machine to dynamically allocate or release hardware resources, (ii) study new scheduling and placement mechanisms in the guest operating system in order to efficiently use these distributed resources, (iii) study how we could use programmable networks to optimize the performance of virtual machines, and (iv) study how we could design transparent replication mechanisms since, in this distributed context, faults are no longer an exception but become the norm.

In the context of edge infrastructures, small clusters of machines are connected by slow networks to powerful data centers. The small clusters perform computations close to the data sources, which avoids expensive data transfers, while having the possibility to use the computing power of a data center. This evolution requires revisiting several system mechanisms to support greater heterogeneity in terms of hardware and software architecture, and to support slow edge/core cloud network communication. In the DiVA project, we will therefore (i) design new virtual machine migration mechanisms in order to migrate virtual machines between heterogeneous machines (instruction set and hypervisor), (ii) transparently optimize the data paths of multi-tier applications distributed between the cloud and the edge in order to minimize the cloud/edge network links, and (iii) define new virtualization interfaces to optimize micro virtual machines with a short lifetime.

In the context of DiVa, Jean-Pierre Lozi is leading a task on rack-scale scheduling, with the goal of making it possible for the scheduler to transparently migrate tasks across machines, while preserving access to open files and network connections.

10 Dissemination

10.1 Promoting scientific activities

10.1.1 Scientific events: organisation

Member of the organizing committees

- Julia Lawall was an organizer of the OlivierFest 2025 held with SPLASH 2025.
- Julia Lawall was a proceedings chair of FSE 2025.

10.1.2 Scientific events: selection

Chair of conference program committees

- Julia Lawall was a program chair of EuroSys 2025.
- Julia Lawall was a program chair of the ASE 2025 Journal-First track.

Member of the conference program committees

- Julia Lawall was a member of the PC of FSE 2025.
- Julia Lawall was a member of the PC of MSR 2025.
- Julia Lawall was a member of the PC of GPCE 2025.
- Julia Lawall was a member of the PC of SLE 2025.
- Jean-Pierre Lozi was a member of the light PC of USENIX ATC 2025.
- Jean-Pierre Lozi was a member of the PC of EuroSys 2025.

10.1.3 Journal

Member of the editorial boards

- Julia Lawall is a member of the editorial board of the journal "Science of Computer Programming".

Reviewer - reviewing activities

- Julia Lawall was a reviewer for IEEE Transactions on Software Engineering, the ACM Transactions on Storage, the Journal of Systems & Software, and Information and Software Technology.
- Jean-Pierre Lozi was a reviewer for ACM Transactions on Software Engineering and Methodology, IEEE Transactions on Parallel and Distributed Systems, and IEEE Transactions on Computers.
- Yu Liang was a reviewer for the ACM Transactions on Storage and ACM Computing Surveys.

10.1.4 Invited talks

- Julia Lawall gave an keynote talk at OOPSLA 2025 on "Automating maintenance of the Linux kernel: a perspective over 20 years".
- Julia Lawall gave an invited talk at Kernel Recipes 2025 on "Program verification for the Linux kernel".
- Jean-Pierre Lozi gave an invited talk at the Huawei Global Software Technology Summit on "FlexGuard: Fast Mutual Exclusion Independent of Subscription".
- Yu Liang gave invited talks at the December 2025 meeting of the Inria Defi OS and at the 1st SAFARI Workshop ("Co-Designing Operating Systems and Processing-in-Memory for Next-Generation Mobile Devices") at ETH Zurich in December 2025.
- Victor Laforet gave an invited talk on "Flexguard: Fast Mutual Exclusion Independent of Subscription" in the Séminaire Scientifique PEPR Cloud in October 2025.

10.1.5 Leadership within the scientific community

- Julia Lawall is a member of the FSE steering committee.
- Julia Lawall is a member of the EuroSys steering committee.
- Julia Lawall was elected Secretary of EuroSys in 2025, for a four-year term.
- Julia Lawall is secretary of IFIP TC2.

10.1.6 Scientific expertise

- Julia Lawall reviewed a proposal for the ANR.

10.2 Teaching - Supervision - Juries - Educational and pedagogical outreach

10.2.1 Supervision

- Julia Lawall and Jean-Pierre Lozi supervised the PhD of Himadri Chhaya-Shailesh, which was defended on December 17, 2025.
- Julia Lawall and Jean-Pierre Lozi supervise the PhD of Keisuke Nishimura, which started on December 1, 2024.
- Julia Lawall and Jean-Pierre Lozi supervise the PhD of Victor Laforet, which started on October 1, 2023.
- Julia Lawall supervises the PhD of Maxime Derri, which started in 2024, with Kahina Lazri of Orange.
- Jean-Pierre Lozi supervises the PhD of Tara Aggoun, which started in 2025, with Gael Thomas of Inria Saclay.
- Jean-Pierre Lozi supervises the PhD of Maxime Collette, which started in 2025, with Alain Tchana of Grenoble INP.
- Jean-Pierre Lozi supervised the PhD of Papa Assane Fall, which was defended on December 5, 2025, with Alain Tchana of Grenoble INP.
- Yu Liang supervises two PhD students (Lei Li and Riwei Pan) at City University of Hong Kong, who started in 2021, with Prof. Chun Jason Xue.

10.2.2 Juries

- Julia Lawall was a reporter on the PhD of Keita Suzuki at Keio University, Japan.
- Julia Lawall was an opponent on the PhD of Alexandru Dura at the University of Lund.
- Julia Lawall was an examiner on the PhD of Damien Jaime at Sorbonne University.
- Julia Lawall was an examiner on the PhD of Hector Suzanne at Sorbonne University.

10.3 Popularization

- Maxime Derri gave a talk at FOSDEM 2025 on "Performance evaluation of the Linux kernel eBPF verifier".
- Keisuke Nishimura gave a talk at Linux Plumbers on "Applying Program Verification to Linux Kernel Code: Challenges, Practices, and Automation".

10.3.1 Specific official responsibilities in science outreach structures

- Julia Lawall was coordinator for the Linux kernel for Outreachy for the May-August round 2025.

10.3.2 Other science outreach relevant activities

- Julia Lawall was the coordinator and a mentor for Outreachy for the Linux kernel in Summer 2025.
- Julia Lawall was elected to the Linux kernel Technical Advisory Board (TAB) in 2025 (two-year term).

11 Scientific production

11.1 Major publications

- [1] J. Brunel, D. Doligez, R. R. Hansen, J. L. Lawall and G. Muller. ‘A foundation for flow-based program matching using temporal logic and model checking’. In: *POPL*. Savannah, GA, USA: ACM, Jan. 2009, pp. 114–126 (cit. on p. 6).
- [2] J. Brunel, D. Doligez, R. R. Hansen, J. L. Lawall and G. Muller. ‘A foundation for flow-based program matching: using temporal logic and model checking’. In: *The 36th annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages, POPL’09*. The 36th annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages, POPL’09. Savannah, Georgia, United States: ACM, Jan. 2009, pp. 114–126. doi: [10.1145/1480881.1480897](https://doi.org/10.1145/1480881.1480897). URL: <https://hal.science/hal-01299040>.
- [3] L. Burgy, L. Réveillère, J. L. Lawall and G. Muller. ‘Zebu: A Language-Based Approach for Network Protocol Message Processing’. In: *IEEE Trans. Software Eng.* 37.4 (2011), pp. 575–591 (cit. on p. 6).
- [4] J. Lawall and G. Muller. ‘Coccinelle: 10 Years of Automated Evolution in the Linux Kernel’. In: 2018 USENIX Annual Technical Conference. Boston, MA, United States, 11th July 2018. URL: <https://inria.hal.science/hal-01853271> (cit. on pp. 6, 7, 14).
- [5] J. Lawall, D. Palinski, L. Gnirke and G. Muller. ‘Fast and Precise Retrieval of Forward and Back Porting Information for Linux Device Drivers’. In: 2017 USENIX Annual Technical Conference. Santa Clara, CA, United States, 12th July 2017, p. 12. URL: <https://inria.hal.science/hal-01556589> (cit. on p. 14).
- [6] B. Lepers, R. Gouicem, D. Carver, J.-P. Lozi, N. Palix, M.-V. Aponte, W. Zwaenepoel, J. Sopena, J. Lawall and G. Muller. ‘Provable Multicore Schedulers with Ipanema: Application to Work Conservation’. In: Eurosys 2020 - European Conference on Computer Systems. Heraklion / Virtual, Greece, 27th Apr. 2020. doi: [10.1145/3342195.3387544](https://doi.org/10.1145/3342195.3387544). URL: <https://hal.inria.fr/hal-02554342> (cit. on p. 6).
- [7] J.-P. Lozi, F. David, G. Thomas, J. L. Lawall and G. Muller. ‘Remote Core Locking: Migrating Critical-Section Execution to Improve the Performance of Multithreaded Applications’. In: *Proceedings of the 2012 USENIX Annual Technical Conference (USENIX ATC ’12)*. 2012 USENIX Annual Technical Conference (USENIX ATC ’12). Boston, MA, United States: ACM, 2012. doi: [10.5555/2342821.2342827](https://doi.org/10.5555/2342821.2342827). URL: <https://inria.hal.science/hal-00779908> (cit. on p. 7).
- [8] F. Méryllon, L. Réveillère, C. Consel, R. Marlet and G. Muller. ‘Devil: An IDL for hardware programming’. In: *Proceedings of the Fourth Symposium on Operating Systems Design and Implementation (OSDI)*. San Diego, California: USENIX Association, Oct. 2000, pp. 17–30 (cit. on p. 6).
- [9] G. Muller, C. Consel, R. Marlet, L. P. Barreto, F. Méryllon and L. Réveillère. ‘Towards Robust OSes for Appliances: A New Approach Based on Domain-specific Languages’. In: *Proceedings of the 9th Workshop on ACM SIGOPS European Workshop: Beyond the PC: New Challenges for the Operating System*. Kolding, Denmark, 2000, pp. 19–24 (cit. on p. 6).
- [10] G. Muller, J. L. Lawall and H. Duchesne. ‘A Framework for Simplifying the Development of Kernel Schedulers: Design and Performance Evaluation’. In: *HASE - High Assurance Systems Engineering Conference*. Heidelberg, Germany: IEEE, Oct. 2005, pp. 56–65 (cit. on p. 6).
- [11] Y. Padioleau, J. L. Lawall, R. R. Hansen and G. Muller. ‘Documenting and Automating Collateral Evolutions in Linux Device Drivers’. In: *EuroSys*. Glasgow, Scotland, Mar. 2008, pp. 247–260 (cit. on p. 9).
- [12] N. Palix, G. Thomas, S. Saha, C. Calvès, J. L. Lawall and G. Muller. ‘Faults in Linux 2.6’. In: *ACM Transactions on Computer Systems* 32.2 (June 2014), 4:1–4:40 (cit. on p. 8).
- [13] L. Serrano, V.-A. Nguyen, F. Thung, L. Jiang, D. Lo, J. Lawall and G. Muller. ‘SPINFER: Inferring Semantic Patches for the Linux Kernel’. In: USENIX Annual Technical Conference. Boston / Virtual, United States, 15th July 2020. URL: <https://hal.inria.fr/hal-02906912> (cit. on p. 6).

11.2 Publications of the year

International peer-reviewed conferences

- [14] T. Faltin, H. Chhaya-Shailesh, J. Lawall and J.-P. Lozi. ‘Tapestry: Revealing Wait-For Dependencies Between Application Threads’. In: *Proceedings of the 13th Workshop on Programming Languages and Operating Systems*. PLOS 2025 - 13th Workshop on Programming Languages and Operating Systems. Seoul, South Korea: ACM, 2025, p. 8. DOI: [10.1145/3764860.3768339](https://doi.org/10.1145/3764860.3768339). URL: <https://inria.hal.science/hal-05306363> (cit. on pp. 10, 12).
- [15] V. Laforet, S. Kashyap, C. Iorgulescu, J. Lawall and J.-P. Lozi. ‘FlexGuard: Fast Mutual Exclusion Independent of Subscription’. In: *SOSP 2025 - The 31st Symposium on Operating Systems Principles*. Seoul, South Korea, 2025. DOI: [10.1145/3731569.3764852](https://doi.org/10.1145/3731569.3764852). URL: <https://hal.science/hal-05241781> (cit. on pp. 10, 11).
- [16] J. Lawall, K. Nishimura and J.-P. Lozi. ‘Understanding Linux Kernel Code through Formal Verification: A Case Study of the Task-Scheduler Function `select_idle_core`’. In: *OLIVIERFEST ’25: Proceedings of the Workshop Dedicated to Olivier Danvy on the Occasion of His 64th Birthday*. OLIVIERFEST ’25 - Workshop Dedicated to Olivier Danvy on the Occasion of His 64th Birthday. Singapore, Singapore, 12th Oct. 2025, pp. 94–105. DOI: [10.1145/3759427.3760371](https://doi.org/10.1145/3759427.3760371). URL: <https://inria.hal.science/hal-05337063> (cit. on pp. 10, 13).
- [17] M. Martone and J. Lawall. ‘Advances in Semantic Patching for HPC-oriented Refactorings with Coccinelle’. In: *2025 IEEE International Parallel and Distributed Processing Symposium, IPDPS 2025 - Workshops*. 30th International Workshop on High-Level Parallel Programming Models and Supportive Environments. Milan, Italy, 3rd June 2025. URL: <https://inria.hal.science/hal-05337107> (cit. on p. 13).
- [18] H. C. Mounah, D. Mvondo, J. Lawall and Y.-D. Bromberg. ‘The Impact of Kernel Asynchronous APIs on the Performance of a Kernel VPN’. In: *SYSTOR 2025 - 18th ACM International System and Storage Conference*. Virtual conference, Israel, 2025, pp. 167–173. DOI: [10.1145/3757347.3759133](https://doi.org/10.1145/3757347.3759133). URL: <https://hal.science/hal-05211974> (cit. on p. 12).

Doctoral dissertations and habilitation theses

- [19] H. Chhaya-Shailesh. ‘Scheduler Guided OpenMP Execution in Cloud VMs: Using Para-Virtualized Task Scheduling Insights For Barrier Synchronization’. Inria - Paris, 17th Dec. 2025. URL: <https://inria.hal.science/tel-05438117> (cit. on p. 11).
- [20] P. A. Fall. ‘Linux as a microkernel: the memory management’s case’. UGA (Université Grenoble Alpes), 5th Dec. 2025. URL: <https://inria.hal.science/tel-05549156> (cit. on p. 12).

11.3 Cited publications

- [21] T. Ball, E. Bounimova, B. Cook, V. Levin, J. Lichtenberg, C. McGarvey, B. Ondrusek, S. K. Rajamani and A. Ustuner. ‘Thorough Static Analysis of Device Drivers’. In: *EuroSys*. 2006, pp. 73–85 (cit. on p. 8).
- [22] V. Chipounov and G. Candea. ‘Reverse Engineering of Binary Device Drivers with RevNIC’. In: *EuroSys*. 2010, pp. 167–180 (cit. on p. 8).
- [23] J. Corbet. ‘The extensible scheduler class’. In: *Linux Weekly News* (Feb. 2023). URL: <https://lwn.net/Articles/922405/> (cit. on p. 7).
- [24] P. Cuoq, F. Kirchner, N. Kosmatov, V. Prevosto, J. Signoles and B. Yakobowski. ‘Frama-C: A Software Analysis Perspective’. In: *Software Engineering and Formal Methods - 10th International Conference (SEFM)*. Thessaloniki, Greece, Oct. 2012. URL: http://pathcrawler-online.com/pubs/final_sefm12.pdf (cit. on p. 8).
- [25] T. David, R. Guerraoui and V. Trigonakis. ‘Everything you always wanted to know about synchronization but were afraid to ask’. In: *Symposium on Operating Systems Principles (SOSP)*. 2013, pp. 33–48 (cit. on p. 7).

- [26] I. Dillig, T. Dillig and A. Aiken. ‘Sound, complete and scalable path-sensitive analysis’. In: *PLDI*. June 2008, pp. 270–280 (cit. on p. 8).
- [27] D. R. Engler, B. Chelf, A. Chou and S. Hallem. ‘Checking System Rules Using System-Specific, Programmer-Written Compiler Extensions’. In: *OSDI*. 2000, pp. 1–16 (cit. on p. 8).
- [28] H. Guiroux, R. Lachaize and V. Quéma. ‘Multicore Locks: The Case Is Not Closed Yet’. In: *2016 USENIX Annual Technical Conference, USENIX ATC 2016, Denver, CO, USA, June 22-24, 2016*. Ed. by A. Gulati and H. Weatherspoon. USENIX Association, 2016, pp. 649–662 (cit. on p. 7).
- [29] L. Guo, J. Lawall and G. Muller. ‘Oops! Where Did That Code Snippet Come From?’ In: *MSR 2014 - 11th Working Conference on Mining Software Repositories*. Hyderabad, India: ACM, May 2014, pp. 52–61. DOI: [10.1145/2597073.2597094](https://doi.org/10.1145/2597073.2597094). URL: <https://inria.hal.science/hal-01080397> (cit. on p. 8).
- [30] J. T. Humphries, N. Natu, A. Chaugule, O. Weisse, B. Rhoden, J. Don, L. Rizzo, O. Rombakh, P. Turner and C. Kozyrakis. ‘GhOSt: Fast & Flexible User-Space Delegation of Linux Scheduling’. In: *Symposium on Operating Systems Principles*. Association for Computing Machinery, 2021, pp. 588–604 (cit. on p. 7).
- [31] A. Israeli and D. G. Feitelson. ‘The Linux kernel as a case study in software evolution’. In: *Journal of Systems and Software* 83.3 (2010), pp. 485–501 (cit. on p. 8).
- [32] A. Kadav and M. M. Swift. ‘Understanding modern device drivers’. In: *ASPLOS*. 2012, pp. 87–98 (cit. on p. 8).
- [33] J. Koschel, P. Borrello, D. C. D’Elia, H. Bos and C. Giuffrida. ‘Uncontained: Uncovering Container Confusion in the Linux Kernel’. In: *32nd USENIX Security Symposium (USENIX Security 23)*. Anaheim, CA: USENIX Association, Aug. 2023, pp. 5055–5072. URL: <https://www.usenix.org/conference/usenixsecurity23/presentation/koschel> (cit. on p. 8).
- [34] J. Lawall, J. Brunel, N. Palix, R. R. Hansen, H. Stuart and G. Muller. ‘WYSIWIB: exploiting fine-grained program structure in a scriptable API-usage protocol-finding process’. In: *Software: Practice and Experience* 43.1 (Jan. 2013), pp. 67–92. DOI: [10.1002/spe.2102](https://doi.org/10.1002/spe.2102). URL: <https://hal.science/hal-00940320> (cit. on p. 8).
- [35] J. Lawall, B. Laurie, R. R. Hansen, N. Palix and G. Muller. ‘Finding Error Handling Bugs in OpenSSL Using Coccinelle’. In: *European Dependable Computing Conference*. Valencia, Spain, Apr. 2010, pp. 191–196. DOI: [10.1109/EDCC.2010.31](https://doi.org/10.1109/EDCC.2010.31). URL: <https://hal.science/hal-00940375> (cit. on p. 8).
- [36] T. Li, J. Bai, Y. Sui and S. Hu. ‘Path-sensitive and alias-aware tpestate analysis for detecting OS bugs’. In: *ASPLOS*. ACM, 2022, pp. 859–872 (cit. on p. 8).
- [37] Z. Li, S. Lu, S. Myagmar and Y. Zhou. ‘CP-Miner: A Tool for Finding Copy-paste and Related Bugs in Operating System Code’. In: *OSDI*. 2004, pp. 289–302 (cit. on p. 9).
- [38] Z. Li and Y. Zhou. ‘PR-Miner: automatically extracting implicit programming rules and detecting violations in large software code’. In: *Proceedings of the 10th European Software Engineering Conference*. 2005, pp. 306–315 (cit. on p. 9).
- [39] F. Méryllon, L. Réveillère, C. Consel, R. Marlet and G. Muller. ‘Devil: An IDL for Hardware Programming’. In: *4th Symposium on Operating System Design and Implementation (OSDI)*. 2000, pp. 17–30 (cit. on p. 8).
- [40] J. Oh, N. F. Yildiran, J. Braha and P. Gazzillo. ‘Finding broken Linux configuration specifications by statically analyzing the Kconfig language’. In: *ESEC/FSE*. Ed. by D. Spinellis, G. Gousios, M. Chechik and M. D. Penta. ACM, 2021, pp. 893–905 (cit. on p. 8).
- [41] L. R. Rodriguez and J. L. Lawall. ‘Increasing Automation in the Backporting of Linux Drivers Using Coccinelle’. In: *11th European Dependable Computing Conference - Dependability in Practice*. 11th European Dependable Computing Conference - Dependability in Practice. Paris, France, Nov. 2015. URL: <https://hal.inria.fr/hal-01213912> (cit. on p. 8).
- [42] L. Ryzhyk, P. Chubb, I. Kuz, E. Le Sueur and G. Heiser. ‘Automatic device driver synthesis with Termite’. In: *SOSP*. 2009, pp. 73–86 (cit. on p. 8).

- [43] L. Ryzhyk, A. Walker, J. Keys, A. Legg, A. Raghunath, M. Stumm and M. Vij. ‘User-Guided Device Driver Synthesis’. In: *OSDI*. 2014, pp. 661–676 (cit. on p. 8).
- [44] R. K. Saha, J. L. Lawall, S. Khurshid and D. E. Perry. ‘On the Effectiveness of Information Retrieval Based Bug Localization for C Programs’. In: *ICSME 2014 - 30th International Conference on Software Maintenance and Evolution*. IEEE. Victoria, Canada, Sept. 2014, pp. 161–170. DOI: [10.1109/ICSME.2014.38](https://doi.org/10.1109/ICSME.2014.38). URL: <https://inria.hal.science/hal-01086082> (cit. on p. 8).
- [45] F. Thung, D. X. B. Le, D. Lo and J. L. Lawall. ‘Recommending Code Changes for Automatic Backporting of Linux Device Drivers’. In: *32nd IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE. Raleigh, North Carolina, United States, Oct. 2016. URL: <https://hal.inria.fr/hal-01355859> (cit. on p. 8).
- [46] W. Wang and M. Godfrey. ‘A Study of Cloning in the Linux SCSI Drivers’. In: *Source Code Analysis and Manipulation (SCAM)*. IEEE, 2011 (cit. on p. 8).