Proposal for the Establishment of the OpenCompute Laboratory at MIT

Anthony Donegan
Co-Founder
Massachusetts Institute of Technology
Cambridge, Mass.
ajzd@mit.edu

Ananth Venkatesh
Co-Founder
Massachusetts Institute of Technology
Cambridge, Mass.
ananthv@mit.edu

***Abstract*—The RISC-V architecture, originally designed to satisfy research requirements at the University of California, Berkeley, has enabled the development of fully open-source computing systems. This breakthrough, while critical for the establishment of a fully free computing environment, has not led to the development of many consumer electronics. Doing so would require the development of custom chips, input/output systems, and related firmware and drivers for a fully free and open-source operating system. In this whitepaper, we outline a laboratory with the aim of developing, from start to finish, a prototype of a fully open-source, provably secure, and fully (and possibly purely) functional high-performance computing environment. We also present a sampling of potential related research directions, touching on subjects as diverse as hardware security, homotopy type theory, and compositional design.**

I. Introduction

A. On Freedom

The free and open-source software (FOSS) movement has gained significant traction since its inception, promoting the four essential user freedoms to run, study and change, redistribute, and modify computer programs [1]. The work of the Free Software Foundation and notable contributors like Richard Stallman [2] have resulted in these ideas having a significant impact on modern software development. As a result, FOSS tools are more accessible than ever, supported by a growing and dynamic community of hackers [3].

While these developments are positive and broad in scope, the focus on software has led to a severe lack of free and open-source hardware needed to provide the necessary compute for FOSS applications [4]. This has led in some cases to great disparities and loss of user freedom—take for example the proprietary nature of the CUDA ecosystem as a result of proprietary NVIDIA hardware [5].

The fundamental issues with proprietary software raised by proponents of FOSS alternatives—namely that proprietary applications are inherently malware and often spyware, that they discourage community involvement and impose corporate limits on user freedom, and that they are significantly less secure than open systems [6]—remain (likely with greater importance) in the context of hardware. As the basis of all computational power, proprietary hardware systems infect the software systems that they will eventually be trusted with running [7].

Proprietary hardware systems cannot be trusted to run FOSS software—while proprietary spyware is limited in what data it can collect due to hardware and operating system level constraints, there are no limits to what information a fully proprietary hardware product can collect and transmit without user knowledge [8].

Consumer electronics, the most prolific examples of proprietary hardware systems, are inaccessible to their consumers, who cannot modify or inspect their internals and whose interactions with the device are entirely determined by malicious interests. Proprietary software may strip users of ownership of an information good, but proprietary hardware strikes at the right to repair [9].

Lastly, when proprietary software is insecure, its vulnerabilities are limited by the environment to which it is constrained to run. When this environment is also proprietary—e.g. on Windows systems—the results can be catastrophic [10]. However, vetted underlying open-source systems can contain these vulnerabilities to reasonable levels. However, vulnerabilities in proprietary hardware components open up critical concerns, rendering all software systems deployed on such hardware completely insecure. An issue in a hardware random number generator in such a way that it does not produce cryptographically secure random numbers, for example, could lead to catastrophic security implications making modern encryption impossible. Developing open systems subject to public review can greatly increase transparency and system security [11].

The development of free and open-source hardware (FOSH) eliminates these concerns and allows the development of entirely free, provably secure systems. This goal, despite its immense potential and appeal to almost all consumers of computational power, has yet to be realized in its fullest form —proprietary hardware systems dominate consumer electronics and, even in environments where efficiency, security, and control are highly valued, alternatives are rare [7].

To the best of our knowledge, a fully free and open-source, provably secure laptop does not exist. The first project of the OpenCompute Laboratory will be to challenge the status quo in hardware development by utilizing only FOSH components and fabricating custom parts where necessary. The end product, which will rival existing proprietary laptop designs, should be fully usable by the typical consumer and have feature-parity with standard laptop models. Embarking on this project, however, will not be an exercise purely in hardware development but also in the development of free and open-source software to accompany and extend it, allowing

for the trusted execution of FOSS software on a FOSH platform, unifying these two interconnected regimes.

B. On Progress

Having discussed the security implications and necessity for a fully open-source and provably secure hardware system—i.e. *why* we would want to build such a system in the first place—we now turn our attention to what current progress looks like in the field—i.e. *how* we can build the system and why now is the time to embark on this challenge. This inevitably leads us to the story of the development of the RISC-V instruction set architecture (ISA), which we outline below.

In May of 2010 at the University of California, Berkeley, Professor Krste Asanović and graduate students Yunsup Lee and Andrew Waterman in the Parallel Computing Laboratory (the "Par Lab") were working on improving hardware for parallel processing [12]. They sought to work from the ground up to improve and experiment with novel hardware systems, developing the Chisel hardware construction language in the process [13]. Chisel was then used to design an entirely new processor, based on the RISC-V instruction set, primarily for theoretical research purposes.

The Par Lab open-sourced both Chisel and the RISC-V ISA under the Berkeley Software Distribution (BSD) license, allowing it to be freely used and distributed. The ISA specification was also released under a Creative Commons license, allowing for public review and further research [12].

The impact of the Par Lab's RISC-V ISA cannot be understated—prior to its development, the primary instruction set architectures in use were x86 and ARM-based, both of which are completely proprietary. Other architectures were practically unusable, presenting a significant roadblock to the development a fully open-source CPU, the most significant component of computation [14].

The development of the RISC-V ISA is quite similar to our aspirations for research directions in the OpenCompute Laboratory. The development of RISC-V, one of the most significant innovations in open computing, was the result of research in parallel processing and related hardware accelerations. In much the same way, we aim to develop fundamental breakthroughs in open-source hardware by pursuing the most far-reaching and abstract research directions, in the interest of identifying promising applications.

The core of our laptop will consist of an open-source RISC-V processor. However, as RISC-V is only an instruction set architecture, we will likely need to fabricate a custom physical chip based on an existing RISC-V processor design, as, to the best of our knowledge, no open-source physical chip supporting the RISC-V architecture is available for purchase. However, due to the theoretical nature of RISC-V research, numerous open-source high-performance chip designs exist—they only need to be fabricated for use in a laptop [15].

Once an open-source CPU has been adequately fabricated, the remaining issues are much simpler to solve, the primary ones being the implementation of open-source RAM and GPU, though existing implementations exist, as we discuss later. Most other components for computation are significantly simpler than a CPU core, and numerous open-source implementations exist, facilitating the completion of the fully free laptop design. A similar project based on the RISC-V architecture has already been attempted in [14].

We believe also that the timing of this project is optimal given that the RISC-V processor architecture, having matured for several years, is now supported by most fundamental FOSS components, namely the Linux kernel [14], C compilers [16], and even high-level programming languages like Haskell [17]. This significantly simplifies the porting of standard applications to run on a RISC-V based system, especially when using an operating system with a sufficiently advanced source-based programmatic build tool, like NixOS [18].

II. Related Work

As far as we can tell, the end goal of this project—an entirely open-source, fully free laptop—does not exist. Indeed, we have not even been able to find open-source versions of certain critical hardware components, like memory chips, suggesting that a project of this scale has not been successfully attempted. Nevertheless, much of our work will utilize and extend existing progress in this space—while an entirely open-source laptop does not yet exist, there are many almost free alternatives and there exist liberated versions of most essential hardware components. We detail our findings in the literature, challenges we anticipate in pursuit of our goal, and potential extensions of current research in this section.

A. Similar FOSH systems

The most similar laptop to what we aim to build currently available for purchase is the MNT Reform, which is built from the ground up to ensure compatibility with FOSS and utilizes mostly FOSH components. However, this is notably not a fully free laptop as it is built on a proprietary SoM [19]. It nevertheless remains a good example of what we aim to achieve since its processor architecture is, unlike x86 systems, fully open-source.

Its open nature offers numerous benefits to potential consumers beyond merely theoretical advantages—open components allow for greater modularity, user-friendly design allows for rapid disassembly and reassembly, and the keyboard adheres to the OpenRGB standard for full user control of lighting settings. It achieves this with a relatively powerful processor and 32 GB of RAM, on par with modern proprietary laptops.

However, in addition to the few proprietary components it does contain, the laptop is expensive, heavy, and lacks full integration of hardware and software components [20]. We aim to meet or exceed MNT Reform's performance while improving on price, weight, and integration. These concerns are severe enough to make the laptop unusable for most consumers [21], which is a critical concern if we want to increase uptake of FOSH and FOSS in general.

Other notable advancements in the development of the fully free laptop include work by Purism on the Librem [22]. This laptop is nowhere near our specification of fully free, as it uses a proprietary Intel chip architecture, but their work in implementing and enforcing hardware security is intriguing

and will likely guide our own analysis of hardware security concerns. Some of their key innovations include the introduction of a USB security key, BIOS and boot sequence security enhancements, and tamper-evident sealing. They have also added a Trusted Platform Module (TPM) chip for secure boot verification—our goal will be to create a custom, fully open-source TPM chip for this purpose.

Most other "open-source" laptops are significantly further away from meeting the criteria that we seek to achieve in our design. Pinebook [23], System76 [24], and Framework [25] have been noted for their open designs, but in reality their products are only "open" in the physical sense that they can be easily taken apart. The underlying hardware components are still mostly proprietary and, in this sense, they are not a major departure from traditional laptop designs. We reject this approach to laptop design in favor of a more transparent one, with a singular focus on maximizing user freedom throughout the process by choosing hardware and software components that facilitate this goal.

B. Anticipated Challenges

Given our preliminary findings, we expect that the greatest obstacles to the completion of our work will lie in the hardware domain, specifically in producing fully open-source components for traditionally proprietary hardware. In particular, fully free RAM chips and GPU architectures may be impossible to find and may require us to design and fabricate a custom chip or some variation of a partially open-source foundation or design.

The closest open-source design of a RAM chip that we could find is the product of work by the OpenDIMM project [26], which works to increase transparency in the ecosystem for memory sticks. Their work has resulted in a preliminary model of a fully open-source RAM chip that we would need to modify and fabricate ourselves for incorporation into a custom-built laptop.

There is also the OpenRAM project [27], which is a Python toolset for creating and testing different RAM designs. This can aid in the verification of custom models and in creating modifications to existing open-source RAM design templates. However, it seems the extent of open-source RAM development is purely theoretical and no such chip has been produced and distributed.

In addition to the challenges of sourcing free RAM, we must also deal with the addition of a graphics processing unit (GPU). While we can opt to perform all graphics on a fully open-source RISC-V based CPU, to realize our goal of a high-performance laptop, a discrete GPU should be developed.

This presents numerous challenges, as GPU architectures are highly proprietary and extremely complex. Currently, the most promising research in this space seems to be the development of the FuryGpu, which was supposedly created by an amateur hardware hacker. However, it is on par with ancient graphics processing and required more than four years to develop. It is also not open-source at the moment and numerous questions remain about its origins and development [28]. We conclude that designing a GPU from scratch would be possible but fall well short of our goal to create a laptop that is high-performance.

However, there are numerous interesting theoretical research directions related to rethinking the architecture of a GPU that we believe might yield promising results, so we will almost certainly pursue this as a side project. However, for practical purposes, laptops we develop will likely lack a discrete GPU or use a proprietary one supporting FOSS drivers if high performance is required. This is an unfortunate compromise that we must make in the present given the state of open-source GPU development, though our aim is to contribute meaningfully to this space to advance our goals in the long-term.

C. Related Research

The primary research directions of the software team are well-studied, though we believe not enough research has been conducted into applications specifically related to implementation and construction of a fully open-source laptop. Here we list some of the research that has inspired the aims of the software team. The specific research directions of this team and its structure are outlined in later sections Section III and Section V.

One interesting area of research previously hinted at is the development of a custom GPU. One of the interesting steps involved in this project would be the development of a custom shader language for it, which we would want to be purely functional to fundamentally restructure the language of graphics. We would also be able to prove certain properties of our implementations and low-level graphics correctness, which is not possible in most existing implementations.

A prototype of what this might look like can be seen in the Funslang shading language, which is a purely functional shading language with Haskell-like syntax and a Haskell compiler. It is transpiled to GLSL, a much more common shading language. Its primary benefits are monadic separation of syntax and semantics, a core tenet of purely functional programming languages, lack of side effects, and an advanced type system, among many others [29]. We will likely seek to emulate these advancements while adding additional optimizations for our custom graphics framework, as well as possibly looking into developing alternative functional high-level graphics interfaces.

We will similarly aim to develop methods for the formal verification of firmware. We will likely borrow the approaches in [30] for testing the integration of firmware and custom hardware, which will be especially important for verifying security properties. Current research in this domain is relatively recent, but we believe there is great potential to expand on current developments and prove more general theorems about the firmware itself as well as its integration into the overall laptop design.

One of our central goals will be to develop a custom operating system or variant of an existing operating system that can seamlessly interface with the complex custom hardware components that make up the laptop. This involves two separate research directions: one focused on developing a custom operating system from scratch and another focused

on extending an existing operating system with specific configuration and patches to create a usable user environment. There is significant prior work in both areas to guide us in completing these projects simultaneously.

In developing a custom operating system from scratch, we draw great inspiration from the House operating system, which is a purely Haskell-based system that lends itself well to formal verification [31]. The work of the House team shows that it is indeed possible to create a fully functioning operating system from scratch, including drivers, window managers, and related software, in a purely functional programming language. This opens up the possibility of formally verifying key components, especially related to security properties, in a way that is as of yet not possible for the Linux kernel.

Moving on to the second project of extending an existing operating system, we would likely be building off the NixOS project, as it allows for purely functional and programmatic builds targeting custom hardware systems [18]. The core of NixOS, the Nix package manager, can be drastically optimized by modifications to the Nix language and core package repository [32], which we may consider if we decide to build a custom package manager to support an extension of NixOS.

The most abstract and eccentric of the software team's proposed research interests, homotopy type theory, is perhaps the most well-studied and interesting of the directions proposed so far. However, as it is extremely broad and powerful in scope, there remains much work to be done, building off the prior work referenced here.

The two primary applications of homotopy type theory to our work that have been well-studied in the literature are in developing systems for formal verification and theorem proving and in analyzing and optimizing chip design and circuit topology. Both of these are intriguing in their own right and merit separate discussions and related literature reviews.

Homotopy type theory is the fundamental basis of most modern formal verification systems, and its use has increased tremendously as mathematicians have developed more advanced theories and explored applications to novel domains [33]. The core idea of homotopy type theory is that it provides an elegant language for designing formal verification systems and writing and proving theorems about phenomena in complex systems with many interacting parts.

Theorem provers like Lean [34] and Rcoq [35] are powerful because of their ability to translate low-level verification tasks into statements involving homotopy type theory. Behind the scenes, they rely on large libraries with domain-specific proof strategies. Writing proofs in a new domain involving different theorems and methods will require creating custom libraries for this purpose, using the power of advanced mathematical machinery to describe the relatively simpler processes we are interested in understanding. The advantage of doing this is that we gain sweeping general insights into the nature of the problems we are solving with exceedingly simple mathematical manipulations. This has led esteemed algebraic topologist Norman Steenrod to quip:

"Perhaps the purpose of categorical algebra is to show that which is trivial is trivially trivial" [36, abstract nonsense]

One interesting application of homotopy type theory would be to develop a similar formal verification system, perhaps based around liquid types in a purely functional programming language. The LiquidHaskell project developed by the Programming Systems Group at the University of California, San Diego, is an interesting proof-of-concept of this idea [37].

The second, more direct application of category theory and homotopy type theory in particular to our work comes from recent research in using it to optimize electronic chip design and study circuit topology. A general foundation for this approach based on category theoretic formulations is given in [38, ch. 6]. A more practical approach to analyzing circuit topology is presented in [39], which we believe will serve as an excellent reference for the formal verification, optimization, and analysis of the hardware team's designs in software. This is still a very active and highly variable area of research, so we anticipate that these references will provide some initial guidance for structuring future research efforts in this space.

Lastly, we examine literature in the space of compositional design, or co-design, a holistic, categorical framework for the design and analysis of complex engineered systems. Co-design breaks down dependencies between independent hardware and software systems and analyzes each constituent subsystem independently as a mapping between a partially ordered set of resources and a partially ordered set of functionalities, where order is defined based on some predefined hierarchy.

This field of study is quite recent, but applications are numerous—the language of category theory seems, so far, to enable the development of general insight into complex phenomena that can help us better design novel systems, like an open-source laptop, in which heterogeneous systems interact to give rise to a set of core functionalities.

The primary resources that we will reference in applying this framework to the problems we face include the seminal textbook on co-design [40] and Professor Zardini's thesis at ETH Zurich [41]. Both provide a general overview of the techniques we will need to formalize the systems we aim to study and provide a plethora of tools to analyze and optimize these systems. As we develop specific designs and other concrete fabrication and software architecture plans, this framework will play a key part in categorically improving performance (functionalities) with minimal resources.

III. Logistics

Broadly speaking, the OpenCompute Laboratory, abbreviated OCL, will be split into a hardware and software team. These teams will be further split into subteams, which will vary depending on the project and phase that the Lab is working on. The details are inherently vague because the terms "hardware" and "software" are poor descriptors for the interdisciplinary abstract research that both teams will be continually engaged in. Below we attempt to give a detailed discussion of how these teams may be organized and what interdisciplinary research might look like.

A. Hardware

The hardware team will be the more practical and applied of the two, though it too will be involved in highly theoretical research in conjunction with the software team, especially in the context of cryptography and hardware security. However, most work will likely focus on the integration and development of custom modules to support the constituent FOSH components of the laptop. This is a highly non-trivial task and will involve extensive low-level systems research, optimization, and, where necessary, experimentation.

In order to pipeline and parallelize the development of these complex and interconnected systems, the hardware team will be split into several subteams. The main categories for development will be in constructing the physical device (chassis, IO, power), PCB design (mainboard, SoM), and chip development (SoC/CPU). These subteams are subject to change based on evolving project needs.

B. Software

The software team will be simultaneously involved in highly theoretical research, investigating practical implementations of those theories to support the hardware team, and in developing software (mostly at a low level) where necessary. As mentioned in Section I.B, we see this abstract research as being crucial to supporting our mission and driving discovery and innovation. In this sense, many of the projects that fall under the domain of the "software" team have little to do with software and even less to do with low-level implementations; a fair number seem at first glance to have no application to OCL's mission, but we believe they will yield crucial insights nevertheless.

At a high level, the software team will be split into various research subteams. The main categories will be in hardware security (jointly with the hardware team), lambda calculus (with a focus on purely functional firmware), systems (with a focus on operating system development and software deployment), homotopy type theory (with a focus on building robust, domain-specific formal verification systems and languages grounded in mathematical theory), and category theory and compositional design (with a focus in rephrasing problems in the nPOV [42] and applying category theory to optimize designs and specifications using the co-design framework [41]). New research directions are sure to emerge, and significant departures from these themes will almost certainly occur, so the structure of subteams should be highly liquid and prone to change, encouraging collaboration and interdisciplinary thinking throughout.

C. Timeline

Weekly general meetings and independently organized subteam meetings, as is typical for build teams, will likely be the primary forum for research, development, and discussion efforts. We will need to request access to relevant fabrication resources when necessary and physical space for design meetings.

IV. Hardware Design

A. Motivation

Proprietary hardware severely limits the modularity, security, and trust of modern consumer electronic devices. To address this, the hardware team's focus is to develop and implement fully open-source, modular, and reproducible consumer-grade electronic devices. The team will embark on a quest to develop a fully open-source laptop platform, the OpenLaptop, as an ideal first and significant step towards this goal of democratizing computation.

To accelerate development, the project is split into four primary stages, each representing a significant and free-standing step towards a fully open hardware ecosystem. These stages can be pursued in parallel or out of order when necessary or helpful. Each stage places a heavy emphasis on documentation and modularity, allowing the devices developed in each phase to interact with one another for continued development.

B. Development

The first stage consists of the development of a fully functional open-source laptop around a commercially available or otherwise pre-existing RISC-V SoM. This allows the team to abstract away the logical components of the platform to focus on the physical device, including a mainboard, power delivery, display, and more. Almost all components for which existing and easily sourced free implementations exist will be open-source. However, the core of this computer—the CPU (but not its instruction set) and RAM (but not the memory controller or relevant firmware)—will be proprietary, which is not ideal.

The second stage is the development or adaption of a fully open-source RISC-V based SoC or CPU which could eventually be integrated with the laptop developed in the first stage. Possible research directions range immensely in difficulty and cost to implement, although there is a lot of existing work to take advantage of. The result of this stage is a computer that has at its core a fully open-source CPU chip. The SoM board for this chip and the RAM remain proprietary.

The third stage is the development of an SoM which will act as an interposer between the SoC/CPU developed in the second stage and the mainboard developed in the first stage. At this stage we will also want to consider open-source RAM chips, so we can present a laptop, without a GPU, built from fully free and open-source hardware components, fulfilling the first pillar of our goal (and the second as well if the software team can complete some initial formal verification).

The last fourth stage will consist of experimental research and development aimed at increasing performance and, crucially, at developing an open-source GPU or working with others to create one that meets our needs. The timeline for this stage is highly variable, but the completed product should fulfill the aim set out at the beginning of this paper—meeting the criteria of being fully free, both in software and hardware, provably secure, and optimized for high-performance computing. Developing this will be a continuing project, and we anticipate annual updates on achieving and extending this goal after the completion of the third stage.

C. Integration

This is a truly full stack project, touching on a broad range of topics even just within the hardware team, including CAD and part fabrication for the exterior, power electronics for the battery and charger, PCB design for the mainboard, HDL architecture for the SoC/CPU design, and more. Completing this project will inevitably involve hands-on exposure to nearly every corner of electrical and computer engineering. The hardware team will also work closely with the software team in creating drivers and low-level code, diving into diverse topics including formal verification and hardware security.

V. Software Design

A. Motivation

Without repeating the information already discussed in Section II.C, we will give a brief overview of the general research directions, aims, and integration tasks that the software team will tackle. The primary motivation of this team is different from the hardware team, since it is not primarily concerned with the development of components. This allows the software team to pursue significant theoretical research and invest heavily in research and development operations.

The goal of the software team is to make significant progress in theoretical subjects that will yield the greatest applications of interest to the hardware team and help further the overall goals of the OCL. Thus, much of the team's research is partly focused on implementation details to aid in integration tasks, while the remaining research is in significantly more theoretical subjects, but focused on applications of that theory to hardware and software development operations.

B. Development

The primary subject areas of interest to the software team are outlined in Section III.B. We will dive into some specific tasks that each subteam might consider working on in support of OCL's mission.

Hardware security is one of the more applied branches of the software team, and research in this subject will require significant coordination with the hardware team to rapidly test new prototypes implementing theoretical ideas. While there is great potential for novel research in this area, a lot of the initial tests will likely use the significant existing body of research in hardware security and cryptography. Much of our work will be focused on efficiency details and issues involved in the practical implementation of these ideas, concerns which are often overlooked in the literature.

We aim to create efficient hardware implementations of multi-party computation models, garbled circuits, and modern cryptographic primitives as FPGAs. This subteam will also work on hardware authentication keys, algorithms for verifying boot sequence authenticity, and secure information access and transfer. This will require the development of a custom TPM chip or similar, as well as hardware random number generators and other important security modules.

Lambda calculus encompasses a broader range of theoretical directions, including the development of purely functional firmware, domain-specific languages, and drivers for ease of formal verification and mathematical reasoning. This team would also deal with integration of the produced firmware and drivers with existing hardware, which would require the development of memory controllers, TPM firmware, and possibly RAM and GPU firmware if feasible. This subteam will likely work closely with homotopy type theorists on formal verification of developed systems and integration.

Systems deals with the application of high-level theoretical constructs to the development of low-level interfaces. The primary concern of this team will be the development of a purely functional, likely Haskell-based, operating system. Related concerns, like a package manager based on the purely functional software deployment model [18], will likely fall under this domain as well.

More applied systems work will involve porting an existing Linux distribution, likely NixOS, to run on a RISC-V based system and interface correctly with custom hardware. This will require significant upstream contributions to a package repository like `nixpkgs` [32]. To ensure a fully open-source software stack, we will need to create free alternatives to necessary proprietary components and replace them.

Homotopy type theory and formal verification touch on a vast set of subjects and are involved deeply in almost all of the software team's work. Here, we give a sampling of the primary applications of these subjects to the OCL's work.

We aim to apply formal verification wherever possible. At the very least, this will encompass verification of functionality (low-level properties) and security (high-level properties) of implementations, all firmware and drivers to ensure the validity of monadic interfaces and existence of proper error handling routines, and, if using a custom operating system, basic security properties of the OS and validity of its integration with installed drivers.

Work in "pure" homotopy type theory is significantly more theoretical and will involve developing theories for the construction of a custom formal verification system. Our focus will be on automated theorem proving in the context of validating security properties and high-level expectations of functionality at all levels of software development. This will require a study of dependent types and ways these can be used to construct and implement liquid types in an arbitrary purely functional programming language. We will also perform a comparison of different type systems and implement an interactive theorem prover with domain-specific knowledge based on homotopy type theory.

The other primary application of homotopy type theory is in analyzing and optimizing circuit topologies at a very low level. The results of this analysis and its utility will be highly dependent on the methods used by the hardware team, so this research will need to be well coordinated to ensure it can be appropriately integrated.

Lastly, we aim to explore compositional design as a mathematical framework for optimizing all components of our laptop design at any scale. The extent to which we will be able to apply this framework and the utility of the relevant optimizations will again be highly dependent on coordination with the hardware team, as co-design takes a *holistic* view of design involving both the software and hardware compo-

nents. Work in this area will primarily consist of developing the appropriate categorical diagrams to represent potential systems for analysis.

C. Integration

Even more so than the hardware team, work on the software team will require deep familiarity with both highly theoretical concepts and the low-level applications that we seek to develop. This dichotomy is at the heart of great insight and innovative research, and it is our aim to work closely with the hardware team to develop practical implementations of workable ideas. At the core of our guiding research philosophy is the nPOV [42], giving rise to categorical and higher homotopical interpretations of low-level systems.

VI. Conclusion

In this whitepaper, we have presented the foundations for the establishment of a new research-oriented build team at MIT, the OpenCompute Laboratory, or OCL. The goal of the OCL is to develop a fully open-source, provably secure, and high-performance laptop, a task which has not been successfully attempted to date. The development of this system will mark a major milestone for user freedom, hardware security, and distributed, democratized computing. The core innovations that will power the development of such a system are grounded in rigorous, abstract mathematical theory, which provides the tools to overcome several key hardware and software limitations we anticipate.

Our novel approach to this project, as well as our team's unique structure as a research-oriented organization devoted to promoting user freedom, is optimal for the task that we aim to solve, which we believe is a significant component of the single most important computing movement of the last few decades or so. We end this proposal by reiterating our steadfast determination to the completion of this vision, which will require ingenuity in design, abstraction and philosophy, and likely the endurance of a significant amount of ardor, tumult, and obloquy.

References

[1] [Online]. Available: https://www.gnu.org/philosophy/philosophy.en.html

[2] R. Stallman, "Richard Stallman's Personal Site." [Online]. Available: https://stallman.org/

[3] R. Milewicz, G. Pinto, and P. Rodeghero, "Characterizing the Roles of Contributors in Open-Source Scientific Software Projects," in *2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)*, May 2019, pp. 421–432. doi: 10.1109/MSR.2019.00069.

[4] G. Gupta, T. Nowatzki, V. Gangadhar, and K. Sankaralingam, "Open-source Hardware: Opportunities and Challenges." [Online]. Available: https://arxiv.org/abs/1606.01980

[5] [Online]. Available: https://docs.nvidia.com/cuda/eula/index.html

[6] "Proprietary Software Is Often Malware." [Online]. Available: https://www.gnu.org/proprietary/proprietary.en.html

[7] Z. Li, W. Seering, J. D. Ramos, M. Yang, and D. R. Wallace, *Why Open Source?: Exploring the Motivations of Using an Open Model for Hardware Development*. in International Design Engineering Technical Conferences and Computers and Information in Engineering Conference. 2017, p. V1T02A059. doi: 10.1115/DETC2017-68195.

[8] A. Belous and V. Saladukha, "Hardware Trojans in Electronic Devices," in *Viruses, Hardware and Software Trojans: Attacks and Countermeasures*, Cham: Springer International Publishing, 2020, pp. 209–275. doi: 10.1007/978-3-030-47218-4_3.

[9] "Defend Your Right to Repair!." [Online]. Available: https://www.eff.org/issues/right-to-repair

[10] C. Djajapranata, "Is Global Tech Infrastructure Too Vulnerable? Professor Responds to CrowdStrike, Microsoft Outage." [Online]. Available: https://www.georgetown.edu/news/ask-a-professor-crowdstrike-outage/

[11] R. Clarke, D. Dorwin, and R. Nash, *Is Open Source Software More Secure?*. 2005, p. 34. [Online]. Available: https://courses.cs.washington.edu/courses/csep590/05au/whitepaper_turnin/oss(10).pdf

[12] "About RISC-V International." [Online]. Available: https://riscv.org/about/

[13] J. Bachrach *et al.*, "Chisel: Constructing hardware in a Scala embedded language," in *DAC Design Automation Conference 2012*, 2012, pp. 1212–1221. doi: 10.1145/2228360.2228584.

[14] S. Butler *et al.*, "An Exploration of Openness in Hardware and Software Through Implementation of a RISC-V Based Desktop Computer," in *Proceedings of the 18th International Symposium on Open Collaboration*, in OpenSym '22. Madrid, Spain: Association for Computing Machinery, 2022. doi: 10.1145/3555051.3555065.

[15] U. Laghi, "Efficient Trace for RISC-V: Design, Evaluation, and Integration on a CVA6 multicore design," 2024. [Online]. Available: https://amslaurea.unibo.it/id/eprint/33923/

[16] K. Cheng, P. Dabbelt, A. Waterman, Müllner Christoph, and others, "riscv-collab/riscv-gnu-toolchain." [Online]. Available: https://github.com/riscv-collab/riscv-gnu-toolchain

[17] S. Tennie, "!13105: Add RISCV64 Native Code Generator (NCG) to master · Merge requests · Glasgow Haskell Compiler / GHC · GitLab." [Online]. Available: https://gitlab.haskell.org/ghc/ghc/-/merge_requests/13105

[18] E. Dolstra, M. de Jonge, and E. Visser, "Nix: A Safe and Policy-Free System for Software Deployment," in *Proceedings of the 18th USENIX Conference on System Administration*, in LISA '04. Atlanta, GA: USENIX Association, 2004, pp. 79–92.

[19] "MNT Reform." [Online]. Available: https://shop.mntre.com/products/mnt-reform

[20] J. Geerling, "MNT Reform - a hackable laptop, not for everyone." [Online]. Available: https://www.jeffgeerling.com/blog/2024/mnt-reform-hackable-laptop-not-everyone

[21] A. Cunningham, "Review: MNT Reform laptop has fully open hardware and software—for better or worse." [Online]. Available: https://arstechnica.com/gadgets/2022/01/review-mnt-reform-laptop-has-fully-open-hardware-and-software-for-better-or-worse/

[22] "Librem 14 Features BIOS and EC Write Protection." [Online]. Available: https://puri.sm/posts/librem-14-features-bios-and-ec-write-protection/

[23] "Pinebook Pro." [Online]. Available: https://pine64.org/devices/pinebook_pro/

[24] "Laptops." [Online]. Available: https://system76.com/laptops

[25] [Online]. Available: https://frame.work/

[26] V. Metodiev, "The OpenDIMM Project." [Online]. Available: https://opendimm.nicepage.io/

[27] M. Guthaus *et al.*, "OpenRAM." [Online]. Available: https://openram.org/

[28] D. Barrie, "FuryGpu." [Online]. Available: https://www.furygpu.com/blog/hello

[29] B. Challenor, "bchallenor/funslang." [Online]. Available: https://github.com/bchallenor/funslang

[30] S. Ray, N. Ghosh, R. J. Masti, A. Kanuparthi, and J. M. Fung, "INVITED: Formal Verification of Security Critical Hardware-Firmware Interactions in Commercial SoCs," in *2019 56th ACM/IEEE Design Automation Conference (DAC)*, 2019, pp. 1–4.

[31] T. Hallgren, M. P. Jones, R. Leslie, and A. Tolmach, "A principled approach to operating system construction in Haskell," *SIGPLAN Not.*, vol. 40, no. 9, pp. 116–128, Sep. 2005, doi: 10.1145/1090189.1086380.

[32] nixpkgs contributors, "NixOS/nixpkgs." [Online]. Available: https://github.com/NixOS/nixpkgs

[33] J. Avigad and F. van Doorn, "The Lean Theorem Prover and Homotopy Type Theory." [Online]. Available: https://florisvandoorn.com/talks/Toronto2016lean.pdf

[34] L. de Moura, S. Kong, J. Avigad, F. van Doorn, and J. von Raumer, "The Lean Theorem Prover (System Description)," in *Automated Deduction*

- *CADE-25*, A. P. Felty and A. Middeldorp, Eds., Cham: Springer International Publishing, 2015, pp. 378–388.
[35] Y. Bertot, "A Short Presentation of Coq," in *Theorem Proving in Higher Order Logics*, O. A. Mohamed, C. Muñoz, and S. Tahar, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 12–16.
[36] nLab authors, "category theory." Apr. 2025.
[37] N. Vazou, E. L. Seidel, and R. Jhala, "LiquidHaskell: experience with refinement types in the real world," in *Proceedings of the 2014 ACM SIGPLAN Symposium on Haskell*, in Haskell '14. Gothenburg, Sweden: Association for Computing Machinery, 2014, pp. 39–51. doi: 10.1145/2633357.2633366.
[38] D. Spivak and B. Fong, *Electric circuits: Hypergraph categories and operads*. in IAP 2019. Cambridge, Mass.: MIT OCW, 2018. [Online]. Available: https://ocw.mit.edu/courses/18-s097-applied-category-theory-january-iap-2019/resources/18-s097iap19textbook/
[39] D. Youvan, "Exploring Homotopy Type Theory (HoTT) for Optimization in Electronic Chip Design: A Theoretical Framework." p. , 2024. doi: 10.13140/RG.2.2.25772.07042.
[40] A. Censi, G. Zardini, and J. Lorand, *Applied Compositional Thinking for Engineers*. applied-compositional-thinking.engineering, 2021. [Online]. Available: https://applied-compositional-thinking.engineering/wp-content/uploads/2021/12/ACT4E-public-slow.pdf
[41] G. Zardini, A. Censi, and E. Frazzoli, "Co-Design of Autonomous Systems: From Hardware Selection to Control Synthesis," in *2021 European Control Conference (ECC)*, IEEE, Jun. 2021, pp. 682–689. doi: 10.23919/ecc54610.2021.9654960.
[42] nLab authors, "nPOV." Apr. 2025.