OPERA: Towards an Open Platform for Extensible Robust Accelerators

Federico Nicolas Peccia
FZI Research for Information Technology
Germany

Anton Paule
FZI Research for Information Technology
Germany

Oliver Bringmann
University of Tübingen
Germany

Abstract

The AI accelerator world faces a reproducibility crisis, as many works do not publish their RTL as open source, making it impractical to replicate the claimed results. Even more, the lack of a unified benchmarking solution for custom accelerators makes it impossible to ensure that two accelerators are compared by executing the exact same model compiled with the exact same compiler toolchain. Motivated by this problem, we envision OPERA, a platform composed of a hardware generator framework and extensions to existing open-source tools, enabling easy sharing and reuse of accelerator components, tapeout configurations, integration tests, benchmarks, and compiler artifacts.

1 Introduction

Comparing AI accelerators requires careful considerations which aren't always taken into account, as several parameters can modify their performance: number of processing elements (PEs), the dataflow of the processing unit (e.g., weight or output stationary), the size and amount of on-chip and off-chip memories, frequency, technology (in the case of an actual physical layout), etc.

Benchmarking is another source of inaccuracy during comparisons. It is common for authors to report "*We used ResNet to evaluate our accelerator.*" but fail to mention the model version or the input size. This information significantly changes the number of operations the accelerator must execute and the amount of memory actually in use. Another missing aspect is normally information about the compiler used to map the tensor operations onto the accelerator. This can significantly affect data reuse and cache access patterns and is important to consider when executing AI models on custom accelerators.

Finally, most AI accelerator papers don't open-source their RTL, making it impossible to replicate their results, so many works tend to compare against the numbers reported in previous papers. A workaround is to replicate previous work using simulators such as Scale-SIM [9] or Timeloop [7], which are good for obtaining approximate metrics but are limited in the configurations they support.

This issue is best illustrated with an example (Table 1). As noted in [11], efficiency by itself is insufficient for a fair comparison between accelerators. None of the design parameters are matched between the two systems. As a result, it is impossible to assert whether the proposed architecture is better than the prior design. A rigorous comparison would instead require all design parameters to be identical (or as closely matched as possible) before drawing architectural conclusions.

Table 1: Data as reported in [6]

	Eyeriss [5]	MPNA [6]
Technology [nm]	65	28
Precision	16-bit	8-bit
PEs	168	128
On-chip memory [kB]	181.5	288
Frequency [MHz]	100-250	280
Area [mm^2]	12.25	2.34
Efficiency [GOP/s/W]	83.1	149.7

2 Proposal

This reproducibility crisis motivates us to propose OPERA, which will facilitate replicable end-to-end results and easy reuse of previously developed accelerator components. Figure 1 shows our vision of OPERA's components, and how they interface with open-source tools.

Composable HW generator[1]: Written in Chisel [2], it will enable us to incorporate advanced language features such as inheritance and traits into the accelerator's design. It will provide a generic architecture that the researcher can configure using a configuration file. OPERA will also implement mechanisms to create entirely new accelerator components by inheriting from predefined abstract classes that enforce the correct HW interfaces.

The core of the design will be the *Execute Modules*, which can implement any matrix-matrix operation (not limited to GEMM). Through the MMIO interface, the *Controller* will receive *execute* instructions for each one of them. Inputs, weights, and outputs will be stored in a set of banked SRAMs. These will be configured by size, datatype, and kind of data to be stored (for example, only inputs, or any). The *SRAM Controller* will then serve as the bridge between the *Execute Modules*, the SRAMs, and the *Load* and *Store* DMAs, connecting the corresponding buses correctly and inserting arbiters where necessary.

It is our vision that future researchers will develop their accelerators using OPERA, ensuring replicability by design: researchers would only need to provide their configuration files for others to replicate their exact RTL.

CHIPS verification framework: Formal verification must play a crucial role without burdening developers. We would go so far as to say that formal equivalence checking, where an `IMP` is checked

[1] An initial implementation can be found at https://github.com/fzi-forschungszentrum-informatik/opera-hw

LATTE '26, March 23, 2026, Pittsburgh, USA

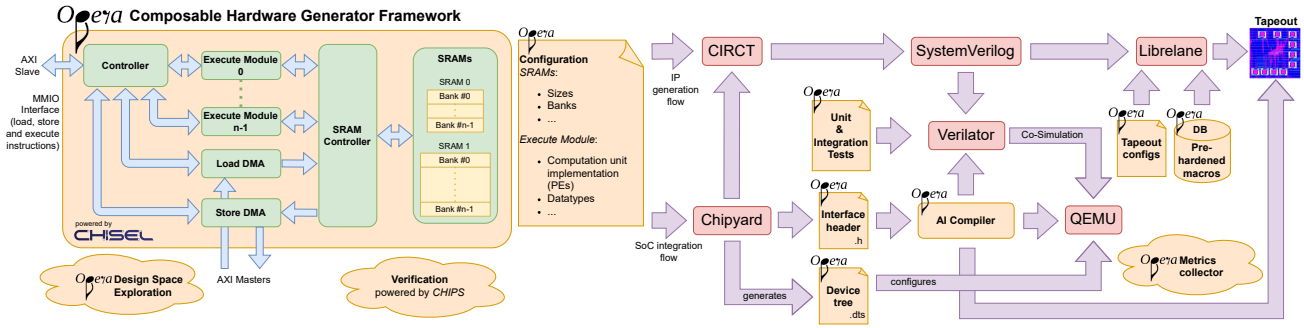

Figure 1: An overview of the features provided/extended by OPERA (orange) and how they interface with open-source tools for EDA (red).

against a SPEC, should be a first-class citizen within the generator itself, enabling continuous, agile feature development. Therefore, OPERA will use a novel approach to integrate formal equivalence checking into the generator, which we will demonstrate with a small example.

```
1  class ExecuteModule(config: Config) extends Module
2  { /* I/0 and Logic ... */ }
3
4  class ExecuteModuleICG(config: Config) extends Module with HasICG {
5    val en = Input(Bool()) // I/0
6    withICGAndReset(clock, reset, en)
7    { /* Clock-gated logic ... */ }
8  }
```

Listing 1: Implementation example.

Listing 1 sketches the implementation of an execution module (`class ExecuteModule`), parametrized by a global configuration. Let's assume the developer wants to implement the same execution module, but this time with an internal clock gating (ICG) mechanism in place. By using the trait `HasICG`, we gain the capability `withICGAndReset` to implement an ICG block. Even though the implementation details are outsourced to the trait itself, the added functionality is not immediately apparent to the developer, especially if code is reused or imported from other projects. New functionalities and features should not break the original implementation.

Listing 2 shows how to implement a parametrized DUV using inheritance. The key novelty lies in the embedded DSL for SEC and the *Chisel Hardware Integration and Verification Services (CHIPS)*. The *CHIPS framework* will be released alongside OPERA and provide basic SEC constructs for defining equivalence checks. *CHIPS* handles the plumbing for configuring formal verification tools, creating miter circuits, and running verification tasks. The developer should only define configuration parameters and instantiate specifications in the derived DUV. Because SEC constructs are now part of the generator itself, we can create DUVs that depend on configuration parameters and demonstrate equivalence across the different specifications simultaneously.

SystemVerilog generation: Will be exported through an IP generation flow (standalone) or integrated into Chipyard [1] as a memory-mapped accelerator.

```
1  class ExecuteModuleICG_DUV(config: Config)      // << IMP
2  extends ExecuteModuleICG(config) {
3    val spec = Module(new ExecuteModule(config)) // << SPEC
4    // == DSL for SEC
5    SEC.addSpec(spec)
6    SEC.addMiter(spec)
7  }
```

Listing 2: Design under verification example.

Simulation: OPERA will integrate with Verilator and provide unit tests for the individual HW modules, and integration tests to evaluate communication with the CPU. This will ensure compatibility of new developed HW modules: if a researcher develops a new `Computation Unit` (the core of each **Execute** module, e.g., a systolic array), it will not be merged into the main branch of OPERA until tests are provided for it. In order to evaluate the integration of the resulting accelerator into a wider context, OPERA will also provide the possibility to run the *verilated* model in a co-simulation manner inside QEMU, which will make system integration testing much easier: an OS could be executed in the resulting QEMU emulation, and the compiled AI model could be executed on it.

Chipyard integration: OPERA will extend Chipyard to generate 1) the complete software interface for a programmer to communicate with the accelerator through a memory-mapped interface, and 2) the parameters of the resulting accelerator integrated into the device tree of the SoC.

AI compiler: These two files will be used by OPERA's AI compiler — an extension of e.g., TVM [4], IREE [12], or other — to correctly compile the suite of benchmarks (e.g., MLPerf Tiny [3] or MLPerf Inference [8]). The AI compilers will read the device tree to determine which memories are available in the accelerator, their sizes, the number of *Execute Modules*, which mathematical operations they implement, etc. This will allow the compiler to map the AI model's tensor operations to the correct *Execute Module*. To ensure consistent benchmarking, all metrics will be obtained by using the same version of the AI model benchmarks and a specific compiler version. Researchers focused on HW design will be able to add new features to the HW and reuse the same compiler, and researchers focused on SW development will be able to modify the compiler (e.g., to map tensor operations more efficiently), keeping

the HW as it is. OPERA will leverage its integrations with open-source tools to ensure consistent metric reporting. OPERA will also integrate a *verilated* model of the resulting accelerator directly into the compiler, allowing peeking and poking at the HW's inputs and outputs without interference from the wider system.

Tapeout: OPERA will use open-source tapeout tools, such as Librelane [10], and will provide basic configurations for each available open-source PDK. OPERA's end objective would also be to create a library of pre-hardened macros from the community, so that as much of the physical design as possible can be reused. Librelane also allows integration of proprietary tools at each step (synthesis, place and route, etc.), so implementing accelerators built using OPERA on closed PDKs will still be an option.

Design Space Exploration (DSE): Overarching all of this, OPERA will be the ideal platform for DSE of AI accelerators. The Chisel programming language is suitable for this, as it allows easy sweeping across parameter ranges (including selecting different implementations for the same HW module) and effectively generates hundreds or thousands of designs. OPERA's DSE framework will provide Model Context Protocol (MCP) interfaces for later integration with AI Agents.

3 Future directions

The OPERA concept is still in its early stages, and as such will benefit from the input of the community to identify missing features and capabilities. First, the current OPERA proposal only covers memory-mapped accelerators operating as co-processors. Modeling other architectures, such as tightly coupled or streaming accelerators, is currently not envisioned, but should be considered for easier integration later as the project matures. Second, the platform now envisions using open-source tools like Verilator and QEMU for simulation, but is not limited to them. Integrating other simulation tools is still a possibility and a necessity for commercialization. Finally, the DSE capabilities from OPERA are currently envisioned, but no tool to achieve this has been selected. Currently available open-source tools for DSE still need to be reviewed, and the need for a new OPERA-specific tool needs to be analyzed.

4 Conclusion

We presented OPERA, our vision for a platform that will promote the open-sourcing of AI accelerators, ensuring replicability and consistent benchmarking by design. We showed how the CHIPS framework could be integrated to manage formal verification, how AI compilers would integrate with the HW generation framework, and what information they would need. Finally, we include tapeout tools in the OPERA framework, using open-source PDKs while allowing the use of proprietary tools to target state-of-the-art technologies.

References

[1] Alon Amid, David Biancolin, Abraham Gonzalez, Daniel Grubb, Sagar Karandikar, Harrison Liew, Albert Magyar, Howard Mao, Albert Ou, Nathan Pemberton, Paul Rigge, Colin Schmidt, John Wright, Jerry Zhao, Yakun Sophia Shao, Krste Asanović, and Borivoje Nikolić. 2020. Chipyard: Integrated Design, Simulation, and Implementation Framework for Custom SoCs. *IEEE Micro* 40, 4 (2020), 10–21. https://doi.org/10.1109/MM.2020.2996616

[2] Jonathan Bachrach, Huy Vo, Brian Richards, Yunsup Lee, Andrew Waterman, Rimas Avižienis, John Wawrzynek, and Krste Asanović. 2012. Chisel: constructing hardware in a Scala embedded language. In *Proceedings of the 49th Annual Design Automation Conference* (San Francisco, California) *(DAC '12)*. Association for Computing Machinery, New York, NY, USA, 1216–1225. https://doi.org/10.1145/2228360.2228584

[3] Colby Banbury, Vijay Janapa Reddi, Peter Torelli, Jeremy Holleman, Nat Jeffries, Csaba Kiraly, Pietro Montino, David Kanter, Sebastian Ahmed, Danilo Pau, et al. 2021. MLPerf Tiny Benchmark. *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks* (2021).

[4] Tianqi Chen, Thierry Moreau, Ziheng Jiang, Lianmin Zheng, Eddie Yan, Meghan Cowan, Haichen Shen, Leyuan Wang, Yuwei Hu, Luis Ceze, Carlos Guestrin, and Arvind Krishnamurthy. 2018. TVM: An Automated End-to-End Optimizing Compiler for Deep Learning. arXiv:1802.04799 [cs.LG] https://arxiv.org/abs/1802.04799

[5] Yu-Hsin Chen, Tushar Krishna, Joel S. Emer, and Vivienne Sze. 2017. Eyeriss: An Energy-Efficient Reconfigurable Accelerator for Deep Convolutional Neural Networks. *IEEE Journal of Solid-State Circuits* 52, 1 (2017), 127–138. https://doi.org/10.1109/JSSC.2016.2616357

[6] Muhammad Abdullah Hanif, Rachmad Vidya Wicaksana Putra, Muhammad Tanvir, Rehan Hafiz, Semeen Rehman, and Muhammad Shafique. 2018. MPNA: A Massively-Parallel Neural Array Accelerator with Dataflow Optimization for Convolutional Neural Networks. *CoRR* abs/1810.12910 (2018). arXiv:1810.12910 http://arxiv.org/abs/1810.12910

[7] Angshuman Parashar, Priyanka Raina, Yakun Sophia Shao, Yu-Hsin Chen, Victor A. Ying, Anurag Mukkara, Rangharajan Venkatesan, Brucek Khailany, Stephen W. Keckler, and Joel Emer. 2019. Timeloop: A Systematic Approach to DNN Accelerator Evaluation. In *2019 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. 304–315. https://doi.org/10.1109/ISPASS.2019.00042

[8] Vijay Janapa Reddi, Christine Cheng, David Kanter, Peter Mattson, Guenther Schmuelling, Carole-Jean Wu, Brian Anderson, Maximilien Breughe, Mark Charlebois, William Chou, Ramesh Chukka, Cody Coleman, Sam Davis, Pan Deng, Greg Diamos, Jared Duke, Dave Fick, J. Scott Gardner, Itay Hubara, Sachin Idgunji, Thomas B. Jablin, Jeff Jiao, Tom St. John, Pankaj Kanwar, David Lee, Jeffery Liao, Anton Lokhmotov, Francisco Massa, Peng Meng, Paulius Micikevicius, Colin Osborne, Gennady Pekhimenko, Arun Tejusve Raghunath Rajan, Dilip Sequeira, Ashish Sirasao, Fei Sun, Hanlin Tang, Michael Thomson, Frank Wei, Ephrem Wu, Lingjie Xu, Koichi Yamada, Bing Yu, George Yuan, Aaron Zhong, Peizhao Zhang, and Yuchen Zhou. 2019. MLPerf Inference Benchmark. arXiv:1911.02549 [cs.LG]

[9] Ananda Samajdar, Jan Moritz Joseph, Yuhao Zhu, Paul Whatmough, Matthew Mattina, and Tushar Krishna. 2020. A Systematic Methodology for Characterizing Scalability of DNN Accelerators using SCALE-Sim. In *2020 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. 58–68. https://doi.org/10.1109/ISPASS48437.2020.00016

[10] Mohamed Shalan and Tim Edwards. 2020. Building OpenLANE: A 130nm OpenROAD-based Tapeout- Proven Flow : Invited Paper. In *2020 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*. 1–6.

[11] Vivienne Sze, Yu-Hsin Chen, Tien-Ju Yang, and Joel S. Emer. 2020. How to Evaluate Deep Neural Network Processors: TOPS/W (Alone) Considered Harmful. *IEEE Solid-State Circuits Magazine* 12, 3 (2020), 28–41. https://doi.org/10.1109/MSSC.2020.3002140

[12] The IREE Authors. 2019. *IREE*. https://github.com/iree-org/iree