Simulating Hybrid Analog + RISC-V Systems for HPC Applications

Cameron Durbin*
University of Oregon
Eugene, Oregon, USA
cfd@uoregon.edu

Jacob Flores
Sandia National Laboratories
Albuquerque, New Mexico, USA
jmflore@sandia.gov

Thomas Weatherly*
Georgia Tech Research Institute
Atlanta, Georgia, USA
Thomas.Weatherly@gtri.gatech.edu

Ben Feinberg
Sandia National Laboratories
Albuquerque, New Mexico, USA
bfeinbe@sandia.gov

Abstract

As digital scaling trends have slowed over the past decade, there has been renewed interest in alternative computing paradigms such as analog. Analog computing has the potential to provide performance and efficiency beyond what is achievable by digital systems, but many challenges remain. One such challenge is enabling complex applications to run on analog components that only support a limited set of computational kernels. We examine a class of hybrid analog–digital systems in which analog accelerators function as tightly integrated coprocessors within each core. The RISC-V ISA simplifies hybrid system design by providing a mature software stack for the digital components, enabling system designers to focus on the analog-specific aspects of the architecture and software. To investigate the viability of hybrid architectures for high-performance computing (HPC), we evaluate two iterative linear solvers on analog–digital RISC-V processors with the Structural Simulation Toolkit.

Keywords

RISC-V, High-Performance Computing, Emerging Memories

ACM Reference Format:
Cameron Durbin, Jacob Flores, Thomas Weatherly, and Ben Feinberg. 2025. Simulating Hybrid Analog + RISC-V Systems for HPC Applications. In *Workshops of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC Workshops '25), November 16–21, 2025, St Louis, MO, USA.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3731599.3767532

1 Introduction

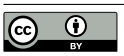
With the end of Dennard Scaling, the trajectory for computer systems is toward higher power consumption for higher performance. This trend, which applies from small embedded processors up to the largest high-performance computing and data centers, implies that increasing problem scale or complexity requires an increasing system power budget. To meet this concern, new technology paradigms beyond conventional digital systems have become a major research interest with several showing potential for orders-of-magnitude improvements in energy efficiency at the operation level. Analog linear algebra acceleration is one such paradigm, taking advantage of analog circuit properties to encode various computations. Although not a new idea, developments in resistive memory technologies over the past decade provide dense, programmable, and CMOS-compatible memories, enabling high programmability and integration scale for these analog accelerators. Using these technologies, recent analog matrix-vector multiplication (MVM) prototypes have demonstrated greater-than-digital energy efficiency in systems with millions of individual resistive memories [1].

A key challenge for the use of analog accelerators is how to effectively integrate them with conventional digital processing. Even if analog components are responsible for the vast majority of the operations, digital systems are still needed for supporting operations such as system configuration, memory and data movement, and other operations which are ill-suited to computation in digital systems. To date, most work on analog MVM accelerators that has considered the role of digital in system design has focused on fixed-function digital pipelines or systems specialized for a narrow range of applications with specialized programming models. Neither of these approaches are suited to more complex algorithms and workflows such as those found in HPC applications.

The RISC-V ISA offers an alternative path. By taking advantage of the open ISA, analog accelerators can be directly integrated with digital logic as tightly-coupled accelerators while also having access to a robust set of digital operations and common software ecosystem.

This paper presents the first analysis of a tightly-coupled analog accelerator with a general purpose digital processor. To analyze these systems, we implement the RISC-V ISA extensions in the cycle-level *Vanadis* RISC-V CPU model of the Structural Simulation Toolkit (SST) [8] and developed a new SST element for simulating analog accelerators. Using this simulation infrastructure, we implement a pair of iterative linear solvers, conjugate gradient (CG) and stabilized biconjugate gradient (BiCG-Stab), and evaluate the scalability of these solvers on analog–digital hybrid systems. As part of this analysis we examine how different architectural choices—different numbers of analog arrays per core—affects the runtime of the iterative solvers and how those architectural choices impact the balance of analog and digital work in the full system. This analysis shows the potential of using RISC-V processors and analog computing hardware for HPC workloads.

*This work was conducted during an internship at Sandia National Laboratories



SC Workshops '25, St Louis, MO, USA

ACM ISBN 979-8-4007-1871-7/2025/11
https://doi.org/10.1145/3731599.3767532

2 Analog MVM Accelerators

Analog MVM accelerators have been widely explored over the past decade as a potential solution to digital scaling slowdowns. The concept of performing MVM operations in the analog domain predates this recent interest [7]; however, the end of Dennard Scaling and improvements in resistive memory technologies have attracted renewed interest in these accelerators. At their core, analog MVM accelerators take advantage of Ohm's Law and Kirchoff's Current Law to perform a matrix vector multiplication. To perform a computation $\mathbf{Ax} = \mathbf{y}$, the resistive memory elements in a memory array are programmed inversely proportional to the values of $\mathbf{A}$ and a voltage input proportional to $\mathbf{x}$ is applied to all of the wordlines in a memory array. By Ohm's Law, each resistive element performs a multiplication and the resulting currents are summed down the bitlines where the dot product can be read out using an analog to digital converter.

There are two important points about this core analog MVM concept which are important for this paper. First, most resistive memories take substantial time and energy to program necessitating applications where the matrix is substantially fixed for the duration of the application. Second, the analog values used in the computation may only have 4–8 bits of precision and require various techniques to improve the overall precision of the computation. A full discussion of techniques for increasing precision and otherwise optimizing analog MVM accelerators is beyond the scope of this paper, but an interested reader can find a longer discussion in a previous review by Xiao et al. [10].

2.1 Analog MVM Accelerators for Linear Solvers

Applications that tolerant of mixed precision and characterized by largely fixed matrices have driven research on analog accelerators, with a strong emphasis on neural network inference Several prior studies have examined high-precision applications such as iterative linear solvers, and while these solvers demand accuracy, their repeated use of a fixed matrix allows the costly setup to be amortized across iterations. Feinberg et al., proposed a technique using multiple arrays to emulate high-precision floating point using low-precision fixed point analog arrays [5]. Subsequent work by Song et al., improved the convergence and representation efficiency through a modified floating point representation [9]. Another approach by Le Gallo et al., proposed using analog MVMs within an iterative refinement scheme to provide high precision results even with low precision MVMs [6].

2.2 Programming Applications for Analog MVM Accelerators

Programmability has not been a major focus for prior work on analog MVM-based accelerators. Part of this is downstream of the emphasis on neural network inference where a relatively small number of fixed function units can provide the bulk of the functionality [11]. PUMA provides an intermediate approach with a specialized ISA for neural network inference allowing flexibility in layer functionality; however still limited to inference-specific operations [2]. PUMA still requires a specialized software toolchain to support the custom programming model and ISA which creates a major gap when attempting to apply analog accelerators to complex HPC applications. Although the iterative linear solvers evaluated in this paper could be effectively implemented using the PUMA ISA, these applications are merely a starting point for HPC workloads. Moreover, the use of standard RISC-V toolchains simplifies porting existing applications to the new hardware accelerators.

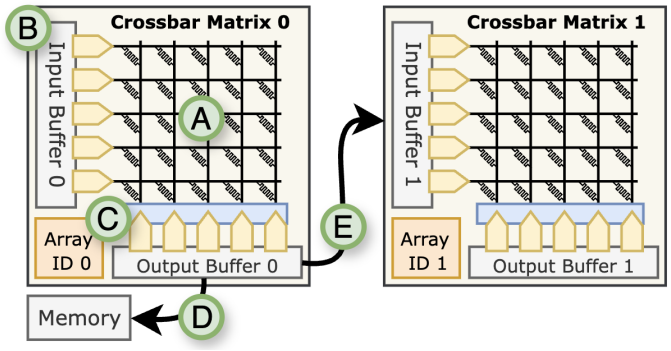


Figure 1: Analog MVM command flow across two crossbar arrays. (A) *mvm.set* configures the compute array. (B) *mvm.l* loads the operand vector into the input buffer. (C) *mvm* performs on-array multiply–accumulate and writes to the output buffer. (D) *mvm.s* stores results to memory. (E) *mvm.mv* forwards results to the next array, enabling multi-stage pipelines without additional memory traffic.

3 System Architecture

This work focuses on an HPC accelerator architecture that combines general-purpose RISC-V cores and an analog coprocessor consisting of one or more analog MVM arrays in a single *tile*. The architecture takes full advantage of the open RISC-V ISA to integrate the analog coprocessor as a functional unit for each core invoked through ISA extensions. Treating the analog arrays as discrete functional units allows the system to effectively use a specialized computational kernel accelerator due to the low overhead of data movement into and out of the analog arrays.

Each tile also contains local SRAM, in the evaluated implementation using hardware-managed caches for programming simplicity; however, using a mix of caches and software-managed scratchpad memories is an important potential architectural optimization. Tiles are connected through a high-bandwidth mesh router, and are for programmer simplicity fully cache coherent.

Notably, the proposed tile architecture looks similar to digital RISC-V accelerators such as the Tensix cores in the Tenstorrent Grayskull. [4] In both instances, specialized functional units—matrix and vector FPU for Tensix and analog arrays for the proposed tile—are tightly integrated with scaler compute cores to enable fine-grained offload of more complex operations. The proposed architecture makes several simplifications to reduce design complexity, specifically the use a cache for local storage rather than the specialized circular buffer SRAM, using a single general RISC-V core rather than specialized *baby cores*, and as discussed in the next section, the communication between the analog arrays and general-purpose cores using local scratchpad rather than specialized packer and

unpacker cores. It is likely that many of the Tensix core optimizations would benefit the proposed analog-enabled core; however the increase in design and programming model complexity makes these optimizations beyond the scope of this initial study.

3.1 Analog Coprocessor

The analog coprocessor is organized into one or more compute arrays, each serving as an independent processing unit for analog operations. Every compute array contains three ISA-visible components:

- **Analog array**—contains the programmed analog conductance values. Importantly, in this work we do not optimize the implementation for programming the analog arrays as such work requires substantial device-specific details and for program-once applications is not a major factor in overall system performance.
- **Input buffer**—holds the input operand vector for the MVM operation. These values are converted into analog voltages by the array's digital-to-analog converters (DACs).
- **Output buffer**—holds the result of the analog computation as digital values after conversion by analog-to-digital converters (ADCs).

For the CPU to coprocessor interface we adopt the conventions of the Rocket Custom Coprocessor (RoCC) interface. [3] The RoCC interface is an extension point that enables custom coprocessors integrate directly with the CPU pipeline. Five core instructions form the basis of accelerator and are represented in Figure 1. As noted above, rather than passing individual values in the instructions, we opt to use local SRAM for passing data between the core and coprocessor. By giving each coprocessor its own dedicated memory port—shared by all of the coprocessor's internal arrays—the RISC-V CPU can issue memory accesses seperate from the CPU, eliminating the need to transmit operands serially through the RoCC interface.

This design choice also leads to a uniform structure for the ISA extensions. Each instruction uses *rs1* to specify the array within the coprocessor, and *rs2* to specify the starting address of the memory access for the given operation. Finally, since we are focusing on an HPC accelerator, we use a floating point interface for the coprocessors under the assumption that each ISA-visible array is actually multiple arrays using a scheme for combining outputs as in prior work [5, 9]. In systems tailored for neural network inference operations, these operations would likely need to be performed on the general purpose CPU rather than in dedicated hardware within the coprocessor.

A typical analog MVM computation begins with the processor configuring a compute array using the mnemonic *mvm.set* (Symbol A in Figure 1), followed by loading the operand vector with *mvm.l* (Symbol B). Once both the matrix and vector are staged, *mvm* (Symbol C) initiates computation within the analog array, producing results in the output buffer. These results can be stored back to memory with *mvm.s* (Symbol D) or forwarded directly to another array using *mvm.mv* (Symbol E), enabling multiple-stage processing without additional memory traffic. Cascading MVM operations across arrays within the same tile allows for complex transformations to be executed entirely on-chip, with all intermediate data kept at the source of computation.

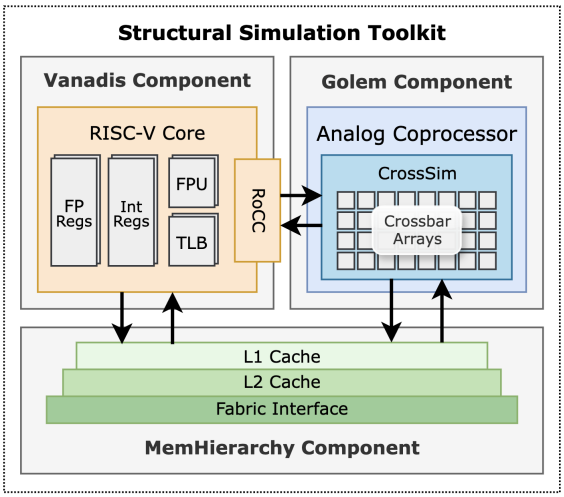


Figure 2: System architecture in SST. Vanadis models a RISC-V core with RoCC; Golem models the analog coprocessor (CrossSim with crossbar arrays); MemHierarchy provides the L1/L2 caches and on-chip fabric. Arrows indicate command and data paths between core, accelerator, and memory.

3.2 System Simulation Model

The architecture in Figure 2, is realized by three specialized SST components: *Vanadis*, *MemHierarchy*, and *Golem* . *Vanadis* models an out-of-order RISC-V core with configurable reorder buffers, pipeline widths, functional units, and load/store queues. *MemHierarchy* implements the private L1 and shared L2 caches, a directory-based MESI protocol, and DRAM controllers connected through a high-radix mesh router. The mesh provides dedicated ports for every core, accelerator, and memory controller, enabling multi-core execution. *Golem* models the analog accelerator, including crossbars and buffers. Together, these components enable full-system execution of RISC-V binaries while exposing pipeline behavior, memory performance, and accelerator utilization in an HPC-class models.

4 Software Stack

We developed an initial software stack to enable complex applications on the proposed system. The stack extends existing RISC-V infrastructure with support for the accelerator. Specifically, we modified LLVM 15.0 to emit the RoCC instructions described in the previous section and implemented a low-level user library to invoke these functions. On top of this library, we built BLAS-like kernels for matrix-vector and matrix-matrix operations, and used OpenMP to distribute tasks across the tiles of the system.

4.1 Analog BLAS

Given the ubiquity of BLAS in HPC applications, we adopt it as the primary user-facing interface to the analog arrays. We implement only kernels that map directly to the proposed analog coprocessor, specifically `gemm` and `gemv`, in both real and complex variants. In addition to exposing a familiar interface, these functions perform

two essential operations for the coprocessor by handling scaling operations and size-agnostic mapping.

Analog arrays typically require scaling for optimal performance. With a fixed-point interface, values must be normalized into the range of [-1,1] or [0,1] to fully utilize the analog domain. With the floating-point interface evaluated here, scaling is less essential but it can reduce the burden of the hardware floating-pointing emulation within the coprocessor.

Unlike digital systems where BLAS functions accept arbitrarily sized matrices, each tile's analog coprocessor supports only up to a fixed matrix size. Moreover, matrices and vectors that are not integer multiples of the underlying array size must be partitioned or zero-padding to map onto the hardware.

To support these operations, the AnalogBLAS functions use a specialized `AnalogMatrix` object to interface with the programmed arrays. Internally, an `AnalogMatrix` allocates arrays from a tile's coprocessor pool, records which array identifiers correspond to each matrix, and maintain the scale factors needed to reverse scaling operations. This design provides prorgammers with a transparent interface to the BLAS functions. In our current implementation, a tile can only allocate arrays within its own coprocessor; problems spanning multiple tiles are handled through OpenMP.

4.2 Multi-tile Operations using OpenMP

To scale problems beyond a single tile, we rely on an unmodified RISC-V OpenMP implementation. In the current implementation, matrices and inputs are manually partitioned so that each tile receives a submatrix no larger than its maximum array capacity. This highlights a key insight for analog acceleration for HPC workloads; systems based on analog arrays naturally favor weak scaling over strong scaling. Although it is possible to allocate smaller portions of the problem to each tile, underutilizing arrays creates significant inefficiencies and should be avoided.

4.3 Implementing Iterative Linear Solvers

Using our Analog BLAS implementation with OpenMP, we implement two iterative solvers, Conjugate Gradient (CG) and BiCG-Stab. The matrix is partitioned with a two-level recursive scheme, first assigning contiguous blocks to tiles and then subdividing those blocks across multiple arrays. CG and BiCG are the same solvers implemented by Feinberg et al., in prior work [5], however by building on the RISC-V software stack we provide more robust and extensible implementation using standard libraries.

5 Evaluation

To evaluate the system architecture we selected two widely-used iterative linear solvers, CG and BiCG-Stab. Although these solvers are typically applied to sparse matrices, we focus the analysis here on synthetic dense matrices. Convergence was verified on smaller test cases such that only ten iterations were simulated for the larger benchmarking effort. We assume that the floating-point emulation from prior work prevents additional iterations due to analog imprecision. A full co-simulation of analog precision and performance is an important topic for future work.

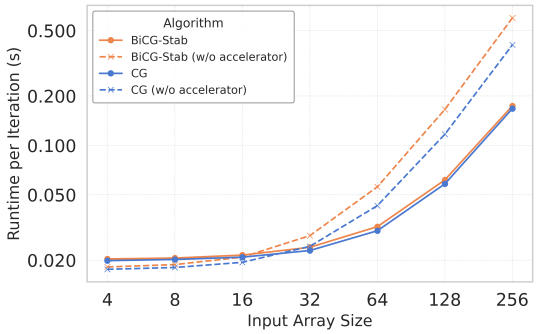


Figure 3: Single-core runtime per iteration of CG and BiCG-Stab versus input size N, with and without a single 256×256 analog accelerator. N varies over powers of two.

5.1 Experimental Setup

Our first experiment evaluates how the runtime of CG and BiCG-Stab scales with increasing problem size N, comparing single-core executions with and without a 256×256 analog accelerator. In this setup each tile includes only a single coprocessor array, so no additional partitioning is required. We vary N over powers of two to observe solver scaling under increasing problem sizes. This configuration establishes the baseline for quantifying the accelerator's contribution to runtime efficiency.

Our second experiment evaluates a fixed-size 1024×1024 dense matrix using 128×128 analog arrays. This workload requires 64 arrays in total. We distribute these arrays across tiles in seven hardware configurations so that #tiles $\times$ #arrays/tile = 64.

Table 1: Processor and system configurations used in experiments.

	Small	Medium	Large
Width (f/d/i/r)	2/2/2/2	6/6/6/6	12/6/12/8
FPUs	2	4	12
IUs	2	4	12
LSQ (LD/ST)	16/8	16/8	192/128
ROB	64	256	512
L1 Cache	32 KB	32 KB	32 KB
L2 Cache	4 MB	4 MB	4 MB
Topology	2D Mesh		
Link Bandwidth	8 GB/s per link		
Router Bandwidth	8 GB/s crossbar		

5.2 Discussion

In our first experiment (Figure 3), both CG and BiCG-Stab show increasing runtime with larger input sizes, but runs without the accelerator rise much more steeply, especially beyond $N = 64$. With a single accelerator attached, both solvers remain consistently faster across all N, and the gap widens at larger sizes. CG is slightly faster than BiCG-Stab in both settings, but the qualitative trend is the same. These results indicate that the accelerator provides a growing benefit as problem size increases.

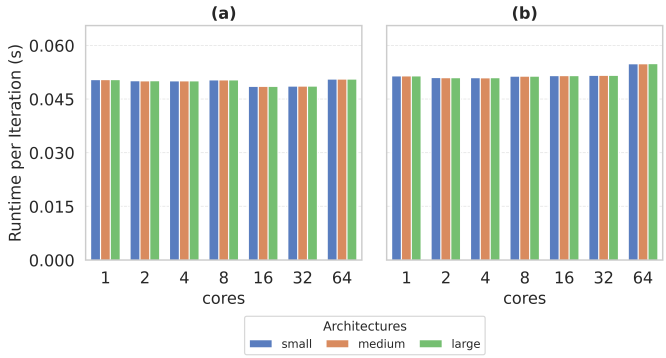


Figure 4: Runtime per iteration for CG and BiCG-Stab on a 1024×1024 dense matrix using 64 arrays of 128×128. (a) CG (b) BiCG-Stab. Seven core/array mappings (1–64 to 64–1); bars denote small/medium/large designs.

For a fixed 1024×1024 matrix, Figure 4 shows CG (a) and BiCG-Stab (b) across seven core/array configurations and three processor designs (small, medium, large). CG is consistently faster than BiCG-Stab. Changing the processor design has negligible effect–the three bars at each core count are nearly coincident—indicating that core width and pipeline resources are not the limiting factors. Runtimes vary only slightly with core count. A modest dip appears around 16–32 cores and is followed by an increase at 64 cores, consistent with higher communication and reduced arrays per core as resources are partitioned. Overall the profiles are flat, suggesting the workload is dominated by data movement and accelerator I/O rather than CPU microarchitecture. Further analysis of interconnect and memory effects is left for future work.

6 Conclusion

We have presented the first full-system simulation study of a hybrid analog–digital architecture that tightly couples analog matrix–vector multiplication accelerators with general-purpose RISC-V cores. Using our SST-based simulator, we have demonstrated the potential benefits of analog hardware for HPC applications, and shown how the openness of the RISC-V ISA speeds the development of new hardware by taking advantage of a mature software stack.

Importantly, this work is only the first step in both the development of hybrid analog-digital systems using general purpose RISC-V processors. There are potential improvements across the entire hardware stack which researchers can explore using the simulation infrastructure presented in this work. On the hardware side, there are significant potential improvements to both the memory system, replacing our caches with scratchpads, and optimizing these scratchpads for the memory access patterns common for analog accelerators. Furthermore, since our general-purpose CPU is specifically supporting the analog hardware, vector support using the RISC-V Vector extension would likely improve system throughput. Additionally, we plan to explore the costs of moving the emulation of floating point operations from a hardened—and by extension inflexible—block within the accelerator to a software implementation running on our general-purpose tile processor. On the software side, we are exploring further extensions to our analog BLAS library and working on an MLIR-based compiler infrastructure to further simplify the deployment of applications for analog accelerators. On the algorithm side, we are exploring methods to improve the realism of our algorithms, incorporating preconditioners and extending our work to GMRES and other widely used iterative linear solvers. Finally, on the simulation side, by coupling these SST simulations with CrossSim, an accuracy simulator for analog arrays, we plan to explore the potential for mixed-precision analog solvers. Potentially increasing the total number of iterations to convergence while improving overall system performance through more efficient analog mappings. By building on our extensible simulation framework for hybrid architectures, researchers from numerous domains can make explore important aspects of these analog + RISC-V architectures for HPC and other applications.

Acknowledgment

This article has been authored by an employee of National Technology & Engineering Solutions of Sandia, LLC under Contract No. DE-NA0003525 with the U.S. Department of Energy (DOE). The employee owns all right, title and interest in and to the article and is solely responsible for its contents. The United States Government retains and the publisher, by accepting the article for publication, acknowledges that the United States Government retains a non-exclusive, paid-up, irrevocable, world-wide license to publish or reproduce the published form of this article or allow others to do so, for United States Government purposes. The DOE will provide public access to these results of federally sponsored research in accordance with the DOE Public Access Plan https://www.energy.gov/downloads/doe-public-access-plan.

References

[1] S. Ambrogio, P. Narayanan, A. Okazaki, A. Fasoli, C. Mackin, K. Hosokawa, A. Nomura, T. Yasuda, A. Chen, A. Friz, M. Ishii, J. Luquin, Y. Kohda, N. Saulnier, K. Brew, S. Choi, I. Ok, T. Philip, V. Chan, C. Silvestre, I. Ahsan, V. Narayanan, H. Tsai, and G. W. Burr. 2023. An analog-AI chip for energy-efficient speech recognition and transcription. *Nature* 620, 7975 (2023), 768–775. doi:10.1038/s41586-023-06337-5

[2] Aayush Ankit, Izzat El Hajj, Sai Rahul Chalamalasetti, Geoffrey Ndu, Martin Foltin, R. Stanley Williams, Paolo Faraboschi, Wen-mei W Hwu, John Paul Strachan, Kaushik Roy, and Dejan S. Milojicic. 2019. PUMA: A Programmable Ultra-efficient Memristor-based Accelerator for Machine Learning Inference. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems* (Providence, RI, USA) *(ASPLOS '19)*. 715–731. doi:10.1145/3297858.3304049

[3] Krste Asanović, Rimas Avizienis, Jonathan Bachrach, Scott Beamer, David Biancolin, Christopher Celio, Henry Cook, Daniel Dabbelt, John Hauser, Adam Izraelevitz, Sagar Karandikar, Ben Keller, Donggyu Kim, John Koenig, Yunsup Lee, Eric Love, Martin Maas, Albert Magyar, Howard Mao, Miquel Moreto, Albert Ou, David A. Patterson, Brian Richards, Colin Schmidt, Stephen Twigg, Huy Vo, and Andrew Waterman. 2016. *The Rocket Chip Generator.* Technical Report UCB/EECS-2016-17. http://www2.eecs.berkeley.edu/Pubs/TechRpts/2016/EECS-2016-17.html

[4] Nick Brown and Ryan Barton. 2024. Accelerating stencils on the Tenstorrent Grayskull RISC-V accelerator . In *SC24-W: Workshops of the International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE Computer Society, Los Alamitos, CA, USA, 1690–1700. doi:10.1109/SCW63240.2024.00211

[5] Ben Feinberg, Uday Kumar Reddy Vengalam, Nathan Whitehair, Shibo Wang, and Engin Ipek. 2018. Enabling Scientific Computing on Memristive Accelerators. In *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*. 367–382. doi:10.1109/ISCA.2018.00039

[6] Manuel Le Gallo, Abu Sebastian, Roland Mathis, Matteo Manica, Heiner Giefers, Tomas Tuma, Costas Bekas, Alessandro Curioni, and Evangelos Eleftheriou. 2017. Mixed-precision in-memory computing. *Nature Electronics* 1 (2017), 246 – 253. https://doi.org/10.1038/s41928-018-0054-8

[7] R. Genov and G. Cauwenberghs. 2001. Charge-mode parallel architecture for vector-matrix multiplication. *IEEE Trans. Circuits Syst. II, Analog Digit. Signal*

Process. 48, 10 (2001), 930–936. doi:10.1109/82.974781

[8] A. F. Rodrigues, K. S. Hemmert, B. W. Barrett, C. Kersey, R. Oldfield, M. Weston, R. Risen, J. Cook, P. Rosenfeld, E. Cooper-Balis, and B. Jacob. 2011. The structural simulation toolkit. *SIGMETRICS Perform. Eval. Rev.* 38, 4 (March 2011), 37–42. doi:10.1145/1964218.1964225

[9] Linghao Song, Fan Chen, Hai Li, and Yiran Chen. 2023. ReFloat: Low-Cost Floating-Point Processing in ReRAM for Accelerating Iterative Linear Solvers. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis* (Denver, CO, USA) *(SC '23)*. Association for Computing Machinery, New York, NY, USA, Article 75, 15 pages. doi:10.1145/3581784.3607077

[10] T. Patrick Xiao, Christopher H. Bennett, Ben Feinberg, Sapan Agarwal, and Matthew J. Marinella. 2020. Analog architectures for neural network acceleration based on non-volatile memory. *Appl. Phys. Rev.* 7, 3 (July 2020), 031301. arXiv:https://pubs.aip.org/aip/apr/article-pdf/doi/10.1063/1.5143815/19740151/031301_1_online.pdf doi:10.1063/1.5143815

[11] T. Patrick Xiao, Ben Feinberg, Christopher H. Bennett, Vineet Agrawal, Prashant Saxena, Venkatraman Prabhakar, Krishnaswamy Ramkumar, Harsha Medu, Vijay Raghavan, Ramesh Chettuvetty, Sapan Agarwal, and Matthew J. Marinella. 2022. An Accurate, Error-Tolerant, and Energy-Efficient Neural Network Inference Engine Based on SONOS Analog Memory. *IEEE Transactions on Circuits and Systems I: Regular Papers* 69, 4 (2022), 1480–1493. doi:10.1109/TCSI.2021.3134313