Simulating a Million-Core System with Switchboard

Steven Herbst, Noah Moroze, Edgar Iglesias, Andreas Olofsson

***Abstract*—Immense hardware systems are needed to train state-of-the-art machine learning applications such as large language models. The design process for such hardware requires simulation for both architectural exploration and design verification; this can be challenging with traditional simulation approaches due to long build and run times. In this paper, we show how to make simulation tractable for large hardware systems with *Switchboard*, our open-source modular simulation framework. Switchboard facilitates scalable simulation by connecting prebuilt simulators of reasonably-sized hardware blocks through high-performance shared-memory queues. Each queue conveys a latency-insensitive interface, eliminating the need for explicit synchronization, which can otherwise hamper performance. Using Switchboard, we successfully performed an RTL simulation of a million-core system on thousands of cloud compute cores. The total time to build the design and simulate a distributed matrix multiplication was under 15 minutes, demonstrating the agility of our approach.**

I. Introduction

State-of-the-art machine learning applications such as large language models and autonomous driving require vast compute resources to train, motivating the development of various large-scale accelerators. For example, NVIDIA's GB200 NVL72 [1] rack design consists of 72 NVIDIA Blackwell GPUs. Cerebras Wafer Scale Engine 3 [2] consists of 900,000 cores on a wafer, which can be used in clusters of up to 2,048 nodes. Tesla Dojo ExaPOD [3] consists of 1,062,000 cores connected together in a hierarchical fashion.

Unfortunately, detailed hardware simulation at this scale is usually slow in terms of the time required to build and run simulations; this limits opportunities for verification and architectural exploration. Parallel RTL simulation, which seeks to distribute RTL simulations across multiple cores, is a way to address the runtime issue [4], [5]. However, scalability can be limited if cores need to reside in the same host system, and build times can still be long. Runtime may also be improved by compiling designs onto one or more FPGAs [6], but at the cost of even higher build time.

In this paper, we present a way to solve these problems using *Switchboard*, our open-source modular simulation framework. We focus on hardware systems composed of reasonably-sized blocks that communicate with each other through latency-insensitive interfaces, a time-tested design methodology that offers many benefits [7]. With Switchboard, a simulator is built for each unique block, which is fast because the blocks are reasonably-sized. At runtime, a prebuilt simulator is launched for each block instance, with block simulations communicating through shared-memory queues that convey latency-insensitive interfaces.

Our approach addresses both build time and run time. Builds are fast because only a small number of reasonably-sized blocks need to be compiled, and these builds can run in parallel. Runtime performance is good due to the partitioning of blocks across latency-insensitive interfaces, avoiding the need for explicit synchronization between simulators.

To demonstrate the effectiveness of our framework, we used it to simulate a simplified mockup of a large accelerator: a million-core array of PicoRV32 CPUs, programmed to compute a distributed matrix multiplication. The simulation was distributed across thousands of cloud compute cores using Amazon Web Services Elastic Container Service (AWS ECS); total simulation time, including build time, was under 15 minutes.

This paper is a shortened version of a manuscript posted on arXiv [8]. Interested readers may find more details about the Switchboard framework and its applications in the arXiv paper.

II. Switchboard Overview

Switchboard is a modular simulation framework, meaning that it allows simulators for hardware submodules to be built independently and connected at runtime. It is open source [9], released under Apache License 2.0, and is installable from PyPI as `switchboard-hw`.

An overview of the framework is shown in Figure 1. We start by enumerating the unique modular blocks in a design. Each unique block is wrapped by tying off its latency-insensitive channels to bridge modules that connect to single-producer, single-consumer (SPSC) shared-memory queues. A simulator is built for each wrapped block, after which point full system simulations can be constructed by running multiple instances of each block simulator, with transmit/receive pairs of channels "wired together" by configuring the corresponding bridge modules to point to the same shared memory queues at runtime.

Shared-memory queues are a natural choice for connecting hardware simulators because they propagate backpressure. Another benefit of these queues is that they are fast, since system calls are only needed for initial setup, not to move data while a simulation is running. Finally, they are straightforward to implement for a variety of hardware model implementations. Switchboard supports mixtures of Verilog simulations, FPGA-based emulators, software models, and even SPICE models.

As a software framework, Switchboard provides a fast shared-memory queue implementation (C++), Verilog bridges for connecting latency-insensitive interfaces to queues (DPI

A. Olofsson is with Zero ASIC Corporation, Lexington, MA 02421 (email: andreas@zeroasic.com). S. Herbst, N. Moroze, and E. Iglesias were with the same company when they contributed to this work.

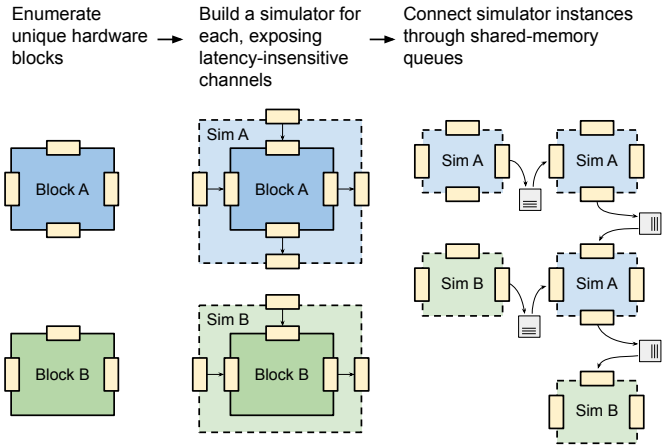

Fig. 1. Our modular simulation approach.

and VPI), Python-based automation for building and running simulations, and a Python API for driving AXI, AXI-Lite, and Universal Memory Interface (UMI) interfaces.

Switchboard enables large simulations to be constructed programmatically, including making decisions about how to partition RTL blocks among one or more simulator processes; it generates Verilog code as necessary to implement these decisions. When multiple RTL blocks are grouped together in a single simulator process, we refer to that process as a "single-netlist simulation."

A. Performance Simulation

Our modular simulation approach yields functionally correct results as-is, but requires modification to estimate performance. This is due to the fact that models run at different rates, and because there is real-world latency in sending data from one model to another.

Figure 2 illustrates this issue. Block A, running in Sim A, sends data to Block B, running in Sim B, which processes the data and sends the result back to Block A. Assuming that Block B takes N clock cycles to process the data it receives, Block A should ideally measure the processing delay (in cycles) to be: $NF_{A,sim}/F_{B,sim}$. However, unless a simulation is running exactly in real-time, the simulation clock rate will be different from the wall time clock rate. Hence, the actual processing time measured by Block A is $NF_{A,wall}/F_{B,wall}$, plus communication latency added by the shared-memory queues and associated bridges.

To correct these issues, we need to slow down simulations A and B so that their wall time clock periods are large compared to the communication latency, while maintaining $F_{A,wall}/F_{B,wall} = F_{A,sim}/F_{B,sim}$. Switchboard provides a mechanism for doing this by sleeping to fill time as needed to achieve a particular simulation rate. Since Switchboard can only slow down simulators, not speed them up, the frequency target set with Switchboard ends up being a maximum frequency bound. As the maximum frequency bound is lowered, the effects of communication latency are attenuated, causing performance measurements to converge to ground truth.

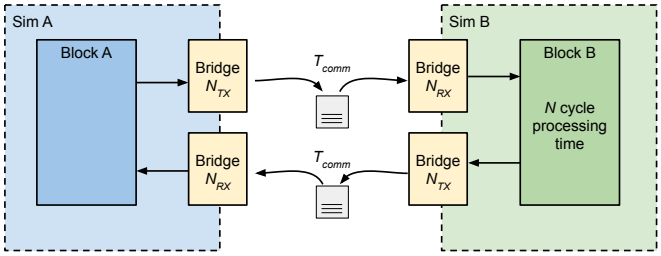

Fig. 2. Nonidealities in performance simulation when conveying latency-insensitive interfaces between simulators.

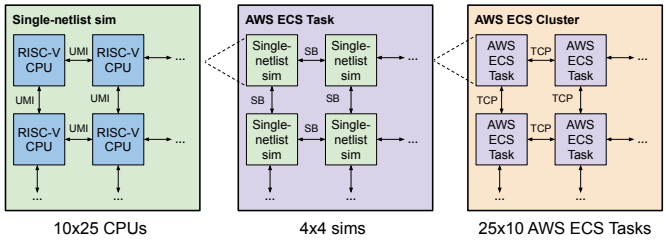

Fig. 3. Architecture of the million-core simulation.

III. Million-Core Simulation

We used Switchboard to simulate an array of one million RISC-V cores, representing a simplified mockup of a large hardware accelerator. This section describes the architecture of the simulation and experimental results.

A. Architecture

The architecture of the million-core simulation is illustrated in Fig. 3. We created a single-netlist simulation of a 10×25 array of RISC-V processors and ran a 4×4 array of these simulations in an AWS ECS task. These tasks were arranged in a 25×10 array in an ECS cluster, resulting in a total of one million RISC-V cores. Fargate was chosen as the ECS capacity provider because it avoided the need to keep cloud resources running when not in use, helping to reduce costs when working with a large number of tasks.

Each ECS task used 16 vCPUs, the maximum allowed with Fargate, and 32 GB RAM. Considering that we used a 25x10 array of ECS tasks, the total number of vCPUs involved in the simulation was 4,000.

The RISC-V processor used in this experiment was PicoRV32 [10], a small, open-source 32-bit implementation. Each processor was wrapped with additional hardware that enabled it to communicate with its neighbors to the north, east, south, and west. For a sense of scale, this unit cell could be implemented with about 20k gates (198k transistors) in the ASAP7 [11] open-source PDK.

We used the RISC-V processor array to implement the matrix multiplication $Y = A \times B$ as illustrated in Figure 4. Each CPU stored an element of B, performing one multiplication

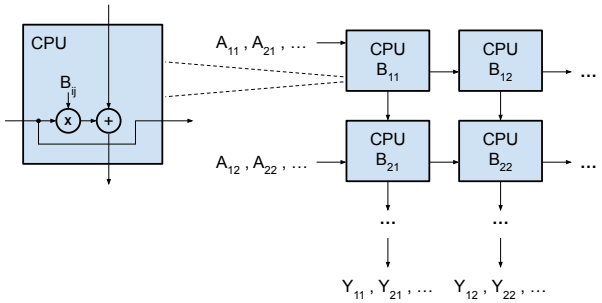

Fig. 4. The RISC-V processor array was used to compute matrix multiplications in a distributed fashion.

TABLE I
TIMING BREAKDOWN FOR THE MILLION-CORE SIMULATION

Name	Time	Percentage
Launch 250 ECS tasks	2m 30s	23%
Wait for ECS tasks to boot	1m 20s	12%
Run simulation	7m 4s	65%
Total	10m 54s	100%

and one addition for each packet it received. The transposed rows of A flowed into the array on its west edge and moved eastward, while partial sums flowed from north to south, with the rows of Y appearing on the south edge.

B. Building and Running the Simulation

Only one RTL simulation needed to be built to run the million-core simulation: the single-netlist simulation of a 10×25 grid of PicoRV32 processors. This modestly-sized simulation took 3m 26s to build on a local machine[1], and the resulting simulation binary was uploaded to AWS Simple Storage Service (S3) where it could be accessed by ECS tasks.

Each ECS task ran 16 copies of this simulation in a 4×4 grid, and the ECS tasks themselves were arranged in a 25×10 grid, resulting in a total of 4,000 copies of the simulation. The simulation took 10m 54s to run a single 1000×1000 by 1000×1000 matrix multiplication, with the timing breakdown shown in Table I. Including build time, it was possible to go from RTL to a behavioral simulation result in under 15m.

C. Cost Analysis

The million-core simulation was run in the Northern California AWS region (`us-west-1`), where the cost of Fargate resources was $0.04656/vCPU/hr and $0.00511/GB memory/hr. Hence, the cost of running the million-core simulation was $41.26. Although not trivial, this is likely reasonable within the context of large-scale hardware design project. In normalized terms, the cost of simulation was approximately 0.16 cents per megagate-megacycle, a value that could be used in back-of-the-envelope cost estimations for other systems.

[1]2.8 GHz quad-core Intel Core i7 with 16 GB 1600 MHz DDR3 memory

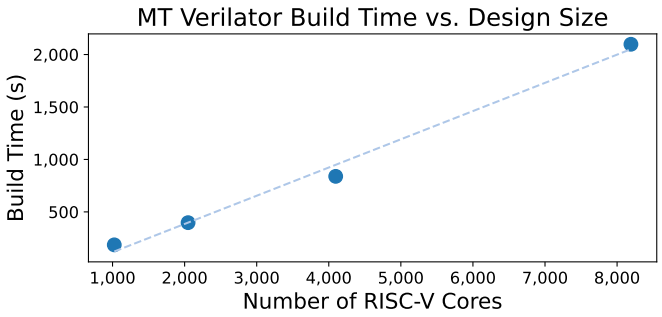

Fig. 5. Build time for different design sizes when using multithreaded Verilator instead of Switchboard.

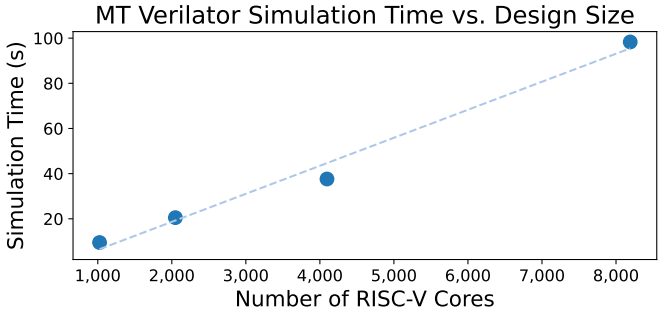

Fig. 6. Time to simulate one matrix multiplication for different design sizes when using multithreaded Verilator instead of Switchboard.

D. Comparison to Multithreaded Verilator

To explore the value of using Switchboard, we consider an alternative: building and running a single multithreaded (MT) Verilator simulator for the entire RISC-V array on a large machine. For this study, the large machine used was an AWS EC2 instance of `c6a.48xlarge`, which provides 192 vCPUs and 384 GB RAM.

To start, we examined build time, scaling up the size of the RISC-V array in factors of two while recording the time needed to build an MT Verilator simulator. The result is shown in Figure 5, demonstrating a fairly linear trend. We stopped at 8,192 cores, where the build time had grown to around half an hour.

Although it is unknown whether a 1M core simulation could be built in any amount of time on this machine, we can arrive at a very rough estimate by extrapolating the trend line. This is clearly problematic because we are extrapolating over two orders of magnitude, however it is still useful to get a sense of how long such a build would take. The extrapolation yields an estimated million-core build time of 3.1 days, with is 1,300x slower than with Switchboard. Given that other work [12] reported a 2.9 day build time for a smaller design (10B transistors), this is likely an underestimate.

For each design size in the experiment, we measured how long it took to simulate a single matrix multiplication, as shown in Figure 6. The trend is linear, though we suspect it would become at least quadratic at larger design sizes. The

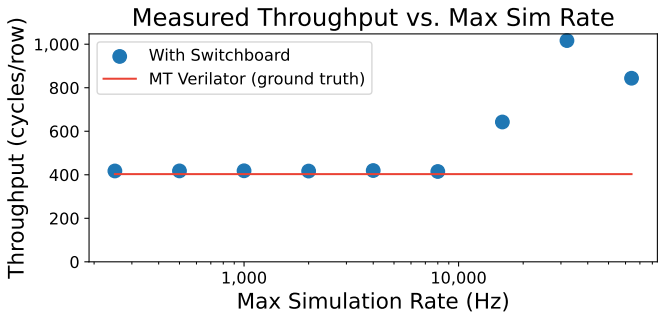

Fig. 7. Effect of varying the Switchboard maximum simulation rate on measured throughput (accuracy indicator).

reason is that the number of threads for each design size was selected for best performance, with the 1,024-core design using 32 threads and the 8,192-core design using 128. Since the machine itself only has 192 vCPUs, the thread count could not be productively increased much more for larger designs. Once all vCPUs are active running simulation threads, the simulation rate would decrease at least linearly with increasing design size, while the number of cycles to simulate would increase linearly, leading to a quadratic trend.

As a result, we believe that extrapolating the linear trend line of simulation time should yield a conservative estimate, in the sense that it would be biased against Switchboard. Performing that extrapolation, we predict that the MT Verilator simulation time for the million-core array would be at least 3h 27m, which is 19x slower than with Switchboard. This seems plausible, given that the Switchboard-based simulation was run on a cluster whose vCPU count was greater than that of the single-machine simulation by a similar factor (21x). Since the cluster used far more vCPUs than are available in a single EC2 instance, Switchboard was the only way to parallelize the simulation to this degree.

Our final comparison entails using MT Verilator to produce a cycle-accurate ground truth against which we measure the accuracy of Switchboard simulations[2]. The performance measurement used in this experiment was the number of clock cycles taken to produce a single row of the matrix multiplication output. As explained in Section II-A, the main control knob for accuracy in Switchboard is the maximum frequency bound. We varied this bound over a wide range while taking performance measurements, with the result shown in Figure 7. As predicted, the performance measurement converged to a value close to ground truth as the max simulation rate was lowered, and for simulation rates of 8 kHz and lower, the accuracy with respect to ground truth was better than 5%.

IV. Related Work

We are aware of only one other research publication about a million-core simulation [13], which used a framework called MuchiSim. MuchiSim employed fast functional processor models in conjunction with cycle count performance models. In contrast, the experiment described here used RTL directly.

Metro-MPI [12] and SimBricks [14] are similar to Switchboard in that they support RTL-based modular simulation. In other words, they partition a large RTL design along interfaces and build simulators for the resulting subdesigns, connecting the simulators together at runtime. However, these frameworks have been evaluated on a smaller scale than Switchboard: 1,024 cores (10B transistors) for Metro-MPI, running on a 48-core HPC system, and 1,000 simulated hosts for SimBricks, running on 26 AWS instances.

FireSim [6] and SMAPPIC [15] both offer cloud FPGA-based emulation, yielding impressive runtime performance in the 1-100 MHz range. However, they have multi-hour build times, since building FPGA images is generally slower than building simulation binaries. In addition, both were evaluated on smaller systems than the system discussed here.

RTL can also be mapped to purpose-built emulation hardware. Commercial products in this space typically have a maximum capacity in the tens of billions of gates [16]–[18], which is approximately the scale of the million-core experiment described here. However, these products are not accessible for open-source work, and often have long build times.

Surveying related work, we believe the research described here is the first RTL-based million-core simulation using open-source tools. To our knowledge, this is also the first use of AWS ECS for distributed RTL simulation.

V. Conclusion

In this paper, we presented an overview of the Switchboard modular simulation framework and demonstrated its application to a mockup of a large hardware accelerator: an array of one million PicoRV32 CPUs. This system was simulated by distributing copies of a small unit cell simulator across AWS ECS Tasks and connecting these simulators together with Switchboard. Including build time, a full-system simulation of a distributed matrix multiplication took less than 15 min. We encourage the open source community to leverage Switchboard's agility and scalability to accelerate large hardware systems research.

[2]Since the largest MT Verilator simulation was of an 8,192-core array, that was the array size chosen for this comparison.

References

[1] NVIDIA. (2024) NVIDIA Blackwell Platform Arrives to Power a New Era of Computing. [Online]. Available: https://nvidianews.nvidia.com/news/nvidia-blackwell-platform-arrives-to-power-a-new-era-of-computing?ncid=no-ncid

[2] Cerebras. (2024) The future of AI is Wafer-Scale. [Online]. Available: https://cerebras.ai/product-chip/

[3] Wikipedia. (2024) Tesla Dojo. [Online]. Available: https://en.wikipedia.org/wiki/Tesla_Dojo

[4] H. Wang and S. Beamer, "RepCut: Superlinear Parallel RTL Simulation with Replication-Aided Partitioning," in *Proc. 28th ASPLOS, Volume 3*, ser. ASPLOS 2023. New York, NY, USA: Association for Computing Machinery, 2023, p. 572–585.

[5] M. Emami *et al.*, "Parendi: Thousand-Way Parallel RTL Simulation," *CoRR*, vol. abs/2403.04714, 2024. [Online]. Available: https://doi.org/10.48550/arXiv.2403.04714

[6] S. Karandikar *et al.*, "FireSim: FPGA-Accelerated Cycle-Exact Scale-Out System Simulation in the Public Cloud," in *45th ISCA*, 2018, pp. 29–42.
[7] L. Carloni *et al.*, "A Methodology for Correct-by-Construction Latency Insensitive Design," in *1999 ICCAD Digest*, 1999, pp. 309–315.
[8] S. Herbst *et al.*, "Switchboard: An Open-Source Framework for Modular Simulation of Large Hardware Systems," 2024. [Online]. Available: https://arxiv.org/abs/2407.20537
[9] ——. (2023) switchboard. [Online]. Available: https://github.com/zeroasiccorp/switchboard/commit/a28374ad751f96dcce0b381459f139ef09680442
[10] C. Wolf *et al.* (2024) PicoRV32. [Online]. Available: https://github.com/YosysHQ/picorv32/commit/87c89acc18994c8cf9a2311e871818e87d304568
[11] L. T. Clark *et al.*, "ASAP7: A 7-nm finFET predictive process design kit," *Microelectronics Journal*, vol. 53, pp. 105–115, 2016.
[12] G. López-Paradís *et al.*, "Fast Behavioural RTL Simulation of 10B Transistor SoC Designs with Metro-Mpi," in *2023 DATE Conf.*, 2023, pp. 1–6.
[13] M. Orenes-Vera *et al.*, "MuchiSim: A Simulation Framework for Design Exploration of Multi-Chip Manycore Systems," in *2024 ISPASS*, 2024.
[14] H. Li *et al.*, "SimBricks: End-to-End Network System Evaluation with Modular Simulation," in *Proc. SIGCOMM 2022*, ser. SIGCOMM '22. New York, NY, USA: ACM, 2022, pp. 380–396.
[15] G. Chirkov and D. Wentzlaff, "SMAPPIC: Scalable Multi-FPGA Architecture Prototype Platform in the Cloud," in *Proc. 28th ASPLOS, Vol. 2*, ser. ASPLOS 2023. New York, NY, USA: ACM, 2023, p. 733–746.
[16] Mentor Graphics. (2024) Veloce Strato CS. [Online]. Available: https://eda.sw.siemens.com/en-US/ic/veloce/strato-hardware/
[17] Synopsys. (2023) Synopsys ZeBu Server 5. [Online]. Available: https://www.synopsys.com/content/dam/synopsys/verification/technical-papers/zebu-server5-spec-mar2023.pdf
[18] Cadence. (2024) Cadence Palladium. [Online]. Available: https://www.cadence.com/en_US/home/tools/system-design-and-verification/emulation-and-prototyping/palladium.html