Fast Behavioural RTL Simulation of 10B Transistor SoC Designs with Metro-MPI

Guillem López-Paradís*† Brian Li‡ Adrià Armejach*† Stefan Wallentowitz§ Miquel Moretó*† Jonathan Balkind‡
*Barcelona Supercomputing Center †Universitat Politècnica de Catalunya
‡UC Santa Barbara §Munich University of Applied Sciences

***Abstract*—Chips with tens of billions of transistors have become today's norm. These designs are straining our electronic design automation tools throughout the design process, requiring ever more computational resources. In many tools, parallelisation has improved both latency and throughput for the designer's benefit. However, tools largely remain restricted to a single machine and in the case of RTL simulation, we believe that this leaves much potential performance on the table.**

We introduce Metro-MPI to improve RTL simulation for modern 10 billion transistor-scale chips. Metro-MPI exploits the natural boundaries present in chip designs to partition RTL simulations and leverage High Performance Computing (HPC) techniques to extract parallelism. For chip designs that scale in size by exploiting latency-insensitive interfaces like networks-on-chip and AXI, Metro-MPI offers a new paradigm for RTL simulation scalability. Our implementation of Metro-MPI in OpenPiton+Ariane delivers 2.7 MIPS of RTL simulation throughput for the first time on a design with more than 10 billion transistors and 1,024 Linux-capable cores, opening new avenues for distributed RTL simulation of emerging system-on-chip designs. Compared to sequential and multithreaded RTL simulations of smaller designs, Metro-MPI achieves up to 135.98× and 9.29× speedups. Similarly, for a representative regression run, Metro-MPI reduces energy consumption by up to 2.53× and 2.91×.

***Index Terms*—RTL Simulation, MPI, Network-on-Chip, Verilator**

I. Introduction

Designing today's 10 billion transistor-scale chips is only getting more difficult and expensive as their scale grows, with little improvement in EDA tool performance. Such performance stagnation has been seen in Register-Transfer Level (RTL) simulation, which is crucial for accurate modelling. Blocks of reasonable scale (10M-100M transistors) often see RTL simulation throughput of only a few thousands of cycles per second (CPS). This means that simulating a core running at 1 GHz with 1 instruction/cycle for 1 second of execution would require over 10 days of simulation time. With poor scaling as designs grow, RTL simulation of full chips has become too costly and is reserved for the final steps in the design process.

To demonstrate this, we simulate large OpenPiton manycore chips [6] with Verilator [20]. Figure 1 shows that increasing the number of simulated cores causes a throughput degradation. For each chip size, we show the instructions per second or IPS (left, blue), CPS (right, green), and compilation time (black line). The CPS decreases as a larger design requires more simulation work per cycle. As this trend continues with size, it is not viable to simulate very large designs, especially since compilation time increases super-linearly with core count, rapidly becoming a bottleneck. To democratise the design of such large systems-on-chip (SoCs) we need a better solution.

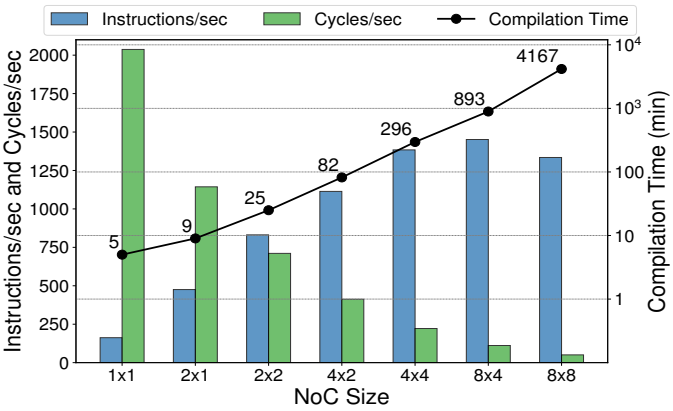


Fig. 1: OpenPiton Verilator Simulation Performance.

Ideally, each stage of our EDA flow would perfectly parallelise independent of chip size, but in reality there are restrictions. Our key insight is to **exploit modern SoCs' natural boundaries (e.g. NoCs) to partition the design and turn RTL simulation into a distributed HPC problem.** We introduce Metro-MPI to enable fast behavioural RTL simulation of emerging-scale chips. In Metro-MPI, each chip is simulated with many independent processes, communicating via the standard Message Passing Interface (MPI) [12] distributed computing runtime. This enables us to scale simulation time and throughput by exploiting more processes and compute nodes as our chips grow. Metro-MPI requires minimal design changes to enable parallel RTL simulation across multiple nodes in an HPC or cloud infrastructure.

Metro-MPI can be used through the whole design process, starting with early-stage designs, unlike specialised hardware used to emulate RTL models such as FPGAs and hardware emulators. Such tools usually require a mature, late-stage design and a substantial financial investment in the case of hardware emulators. With its fast compilation times, Metro-MPI can further complement the use of hardware emulation in later stages of the design process.

Our main contributions are:

- A general methodology, exploiting the natural boundaries found in modern chips, to parallelise RTL simulation using MPI, applied to both Verilator and a commercial simulator.
- Speedup compared to sequential and multithreaded RTL simulations of up to 135.98× and 9.29×, respectively.
- Exceptional scaling of RTL simulation to tens of nodes, reaching 2.7 MIPS for a 10B+ transistor, 1,024-core chip.
- Energy reduction of 2.53× for a representative regression.
- Open-source release at https://github.com/metro-mpi.

978-3-9819263-7-8/DATE23/© 2023 EDAA

II. Metro-MPI Framework

Recent manycore chips (e.g. Graviton 3 [1] and ET-SoC-1 [11]) feature 10s-1000s of cores connected via NoCs to enable core count scaling. A tile may contain cores, accelerators, etc., and those heterogeneous tiles are replicated many times. Each tile is connected to adjacent tiles by NoCs that send simple messages. However, even with NoCs to improve scalability, simulating designs is still challenging, requiring an increasing amount of computation per simulated cycle as the design grows. Ideally, we would like to simulate a design with thousands of cores at the simulation speed of a single-core design.

To make simulation manageable, parallelisation is needed. We introduce Metro-MPI, a generic methodology to distribute RTL simulation and unlock SoCs' inherent parallelism. We leverage best practices in parallel programming from HPC to partition well-defined blocks within designs into isolated simulation processes that communicate via MPI message passing. Metro-MPI works particularly well with replicated blocks of comparable size, such as manycores with NoCs. For each cycle, each process simulates in parallel then synchronises with its neighbours. This is enabled by the latency-insensitive interfaces [22] throughout the design in the form of NoCs, AXI, etc.. As a use case, we adopt the OpenPiton+Ariane tiled manycore [5], [6], which connects tiles via NoCs, while peripherals and accelerators use NoCs or AXI. Figure 2 shows an example of Metro-MPI partitioning an OpenPiton SoC's tiles into groups of simulation processes plus individual processes for the chipset, the bootrom, and a MIAOW GPGPU [4].

A. Integration Methodology

Metro-MPI replaces the wires between blocks with MPI messages. We aim to pass only a single round of messages per simulated cycle to reduce overhead. As we carry signal wires through MPI messages (rather than e.g. higher-level NoC messages), simulation processes run in lockstep. To ensure that the selected wires are suitable and to avoid combinational loops, we identify their wire sorts [10]. The `to-sync` and `from-sync` sorts (indicating an input/output port connects to a register or memory) guarantee that only a single message needs to be sent per cycle. Wires with the `to-port` or `from-port` sorts (indicating the input/output port combinationally connects to another input/output port) must be understood by the designer to avoid loops or feedbacks (e.g. valid-ready interfaces where valid depends on ready or vice versa). If there are issues, multiple rounds of messages and circuit evaluation could ensure that the simulation reaches a stable, fixed point each cycle. Such dependencies are uncommon for composable latency-insensitive interfaces. For our NoCs and AXI, only a single round of unidirectional messages is needed. With this generality, Metro-MPI can be applied to arbitrary hardware designs, provided that the interfaces are understood. We focus on NoCs and AXI, but other latency-insensitive interfaces are easy to target, and many common interfaces are usable unmodified.

Metro-MPI uses SystemVerilog Direct Programming Interface (DPI) to make calls like `MPI_Send` and `MPI_Recv` for tile-to-tile communication. MPI requires that the simulation

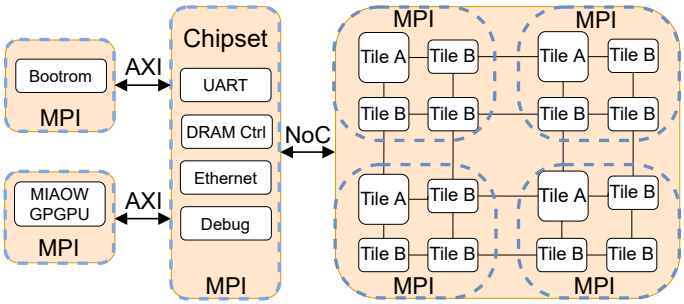


Fig. 2: Metro-MPI in a Heterogeneous Design.

binary is created using the `mpic++` compiler (in addition to an existing compiler like `gcc` or `icc`) and so Metro-MPI supports compiled code SystemVerilog simulators. We demonstrate Metro-MPI using Verilator and a commercial simulator, which are supported differently:

1) Metro-MPI in Verilator: Verilator differs from most SystemVerilog simulators in using C++ testbenches, to which we insert MPI calls at clock generation and evaluation. After calling `eval()` on the rising edge and the design reaching a fixed point, we send all NoC messages, then receive all NoC messages (in the same order), ensuring lockstep synchronisation. We then call `eval()` again to propagate received messages into the simulation. Verilator is optimised to avoid re-evaluation if wires have not changed and empirically we achieve significant speedups.

2) Metro-MPI in SystemVerilog-compliant Simulators: For other simulators, we add our DPI/MPI calls to the SystemVerilog testbench. We add a delay after the clock edge to ensure the design has reached a fixed point before sending and receiving the MPI messages. The simulator automatically evaluates the design according to the language's simulation timing model without need for manual `eval()` calls.

B. NoC-based Partitioning

For our case study, partitions are connected through NoC routers' input and output signals which we turn into Metro-MPI messages. In OpenPiton, each tile-tile connection has 3 NoCs in each direction, and each NoC has three signals:

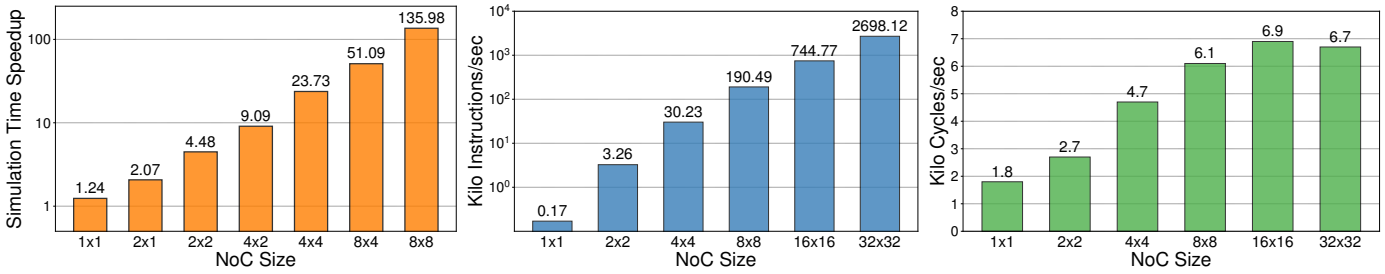
- Valid: Current data is valid on this cycle (1 bit).
- Data: Message itself sent over the NoC (64 bits).
- Yummy: NoC credit return to sender (1 bit).

All signals between two tiles can be grouped and sent using a single MPI message. Empirically, we see roughly a 10% improvement on a single node (regardless of design size) with this grouping, compared to partial or no grouping. We use the best configuration for our evaluation.

Since the SoC is distributed into independent processes, instead of Verilog `$finish`, the chipset notifies the tiles of simulation finish using `MPI_Bcast`. The chipset triggers this on a store to a special memory address (also used by OpenPiton on FPGA). To avoid messaging every cycle, the chipset and tiles communicate at a configurable interval.

C. Multi-Tile Granule

Large designs can have hundreds of tiles so partitioning them using a single tile per MPI process (a "single-tile granule" or



(a) Simulation time speedup over sequential. (b) Actual KIPS scalability. (c) Actual cycles per second scalability.

Fig. 3: Single-tile granule Metro-MPI performance results using Verilator: (a) simulation time speedup normalised to the 1×1 sequential design; (b) simulated KIPS and (c) simulated cycles per second with NoC designs of up to 1,024 tiles (32×32).

STG) requires many HPC nodes (with good connectivity to reduce MPI costs). We created multi-tile granules (MTGs) with multiple tiles in one MPI process, to enable scaling with fewer resources. Figure 2 features a 4×4 chip divided into 2×2 MTGs, which reduces the required number of MPI processes as each process simulates four tiles. As the granule grows, so does the amount of computation for an MPI process per simulated cycle. The advantage is lower communication costs from all intra-granule communication happening locally and the grouping of all signals between pairs of processes into a single MPI message. We evaluate MTGs in Section IV-E.

III. Experimental Setup

We use an HPC system with nodes that contain 2 sockets of Intel Xeon Platinum 8160 CPUs with 24 cores and 32MB LLC each, running at 2.10GHz. The nodes have 96GB DDR4-2667 of main memory (2GB per core) and are connected via a 100Gbit/s Intel Omni-Path HFI Silicon to other nodes.

We evaluate simulations across SoC sizes using a 2D mesh NoC topology, from 1 to 1,024 cores: 1×1, 2×1, 2×2, 4×2, 4×4, 8×4, 8×8, 16×8, 16×16, 32×16, and 32×32. We use an OpenPiton development version dated after release 13 and Ariane v4.2. To support 1,024 cores, we modified OpenPiton's L2 cache to increase the coherence share vector to 1,024 bits. Using 12nm synthesis technology, we conservatively estimate 13 million transistors per tile with a 128 bit share vector (not adapted for 1,024 tiles and thus considerably smaller). Therefore, our 1,024 tile chip would have significantly more than 10 billion transistors. Each Ariane core runs a program passing a token using atomic operations to synchronise and communicate with its adjacent cores (i.e. core+1 and core-1).

We first assign one process per tile (STG Metro-MPI) and one for the chipset (with no GPGPU). We use the minimum number of nodes to accommodate all required MPI processes (i.e. 1 node for NoCs up to 8×4, and up to 22 nodes for 32×32). When using more than 1 node, we balance the number of processes per node to maintain uniform parallel execution and better exploit the available last-level cache. We do not manually place processes and leave such optimisations for future work.

We use Verilator v4.034, GCC v10.1 and Intel MPI v2017.7. Our profiling with `perf` v5.4.133 and Intel VTune v2019.4 (Sec. IV-A) led us to identify instruction cache bottlenecks and thus change our compiler flags. For best performance, we compile simulation models with `"-Os"`. Compilation time is 5 minutes to 69 hours (1 to 64 cores) for the unmodified design, and 3 minutes to 2 hours (1 to 1,024 cores) for Metro-MPI.

TABLE I: Verilator simulation profiling results.

		NoC Size				
		1x1	**2x2**	**4x4**	**8x8**	**8x8**
ITLB MPKI	Sequential	0.03	0.54	1.11	1.06	1.14
	Metro-MPI	0.01	0.01	0.01	0.12	0.39
ICache MPKI	Sequential	11.71	8.69	17.14	19.16	29.30
	Metro-MPI	7.99	9.44	10.56	9.03	9.52
IPC	Sequential	1.05	0.87	0.55	0.53	0.34
	Metro-MPI	1.31	1.31	1.37	1.18	0.96

IV. Evaluation

We evaluate Metro-MPI with STGs (Sec. IV-A and IV-B), compare to Verilator's multithreading (Sec. IV-C), and evaluate a commercial simulator (Sec. IV-D). Sec. IV-E introduces MTGs. Sec. IV-G shows partitioning of heterogeneous SoCs.

A. Simulation Time Speedup

Figure 3a shows the speedup of STG Metro-MPI from 1×1 to 8×8 (1 tile to 64 tiles) normalised to sequential Verilator. We evaluated the sequential design only up to 64 tiles due to its long compilation times (69+ hours for 8×8). We show Metro-MPI speedup normalised to sequential simulation runtime on the y-axis and the chip dimensions on the x-axis. We observe a $1.2\times$ speedup for 1×1, near-linear speedups up to 4×2, and super-linear speedups beyond, reaching $135.9\times$ with 64 tiles.

To understand these super-linear speedups we profiled both settings. Table I shows instruction cache (ICache) and TLB (ITLB) misses per kilo instruction (MPKI) and instructions per cycle (IPC) for sequential and STG Metro-MPI simulations over a subset of our configurations. The sequential design has high ICache and ITLB MPKIs that increase significantly with chip size, reaching 1.14 ITLB MPKI and 29.30 ICache MPKI for 64 tiles, while IPC decreases by $3\times$ from 1.05 to 0.34. This indicates a clear bottleneck in the host's front-end. This can be explained by the code generation, which we find has very long functions and many hard-to-predict branches. The sequential binary starts at 2MB for 1×1 and grows to 89MB for 8×8.

In contrast, Metro-MPI has significantly lower ICache and ITLB MPKIs which do not increase significantly with chip size; ITLB MPKI starts to increase at 8×8 and ICache MPKI remains relatively stable. This is due to the STG partitioning:

TABLE II: Metro-MPI speedup versus Verilator multithreading.

Speedup Comparison	4x4	8x4
Multithreading vs sequential	4.2	5.5
Metro-MPI vs sequential	23.7	51.1
Metro-MPI vs multithreading	5.64	9.29

the design has a single tile in each process. As such, the Metro-MPI tile binary has a constant size of 3MB regardless of chip size. We also see that 1×1 Metro-MPI has better IPC (1.31) than sequential (1.05), in line with the speedup in Figure 3a. IPC drops somewhat with size but only by 26.7% at 64 tiles.

Our findings are confirmed by a recent paper [7] describing Verilator's conversion of core logic into long C++ files with low code reusability. With Metro-MPI we can alleviate the ICache and ITLB problems that come with RTL simulation of large designs, giving super-linear speedups on scalable HPC systems.

B. Throughput Scalability

Figure 3b shows throughput scalability in KIPS for STG Metro-MPI on the y-axis (log scale) and chip dimensions on the x-axis for up to 1,024 tiles (32×32). We obtain throughputs from 0.17 KIPS for 1×1 to 2.7 MIPS for 32×32. This confirms that Metro-MPI scales well, reaching the milestone of MIPS-range RTL simulation speeds, which is within $10\times$ of high-level C++ manycore simulators [13].

Figure 3c shows scalability in CPS on the y-axis and chip dimensions on the x-axis. We achieve 1750 CPS for 1×1 to close to 7000 CPS for 16×16. For 32×32, we see a slight decrease in throughput, which we attribute to MPI communication cost at 22 nodes. Comparing to sequential (Fig. 1), we see that Metro-MPI scales significantly better. On the 8×8 NoC we see 50.5 CPS for sequential versus 6128 CPS for Metro-MPI. With large designs, Metro-MPI not only increases KIPS throughput, but also maintains CPS throughput.

C. Comparison With Multithreading

We compare to Verilator's automatic multithreading, restricting Metro-MPI to a single node. Table II shows the simulation time speedup with multithreaded Verilator (thread count matching the design's core count) and Metro-MPI over sequential Verilator. We use 4×4 and 8×4 designs, which are the two largest to fit on one HPC node. We achieve moderate speedups with Verilator's multithreading, $4.2\times$ and $5.5\times$ for 4×4 and 8×4, respectively. Metro-MPI is significantly better, even improving over Verilator's multithreading by $5.64\times$ and $9.29\times$ for 4×4 and 8×4, respectively. In addition, Verilator's multithreading cannot scale to multiple nodes, and is thus constrained to the tens of cores typical for a single node.

D. Commercial Simulator Performance

To demonstrate its generality in improving simulation performance, we also evaluated Metro-MPI using a "Big 3" commercial simulator. Due to licensing limitations, we ran on a single node 16 core (32 threads) Intel Xeon Gold 6226R with 96GB of DDR4 DRAM instead of the HPC cluster, which limited scaling. Table III shows the speedup improvement in simulation time, CPS, and IPS that Metro-MPI can bring with the commercial simulator. All three metrics show very good scaling up to the core count of the machine (16 simulated cores + 1 simulated chipset on 16 physical cores) with a dropoff thereafter. These results are very promising, showing that Metro-MPI can provide similar performance improvements across different simulators.

TABLE III: Metro-MPI scaling with a commercial simulator.

Speedups	1x1	2x1	2x2	4x2	4x4	8x4
Simulation time	0.93	1.54	3.20	6.17	8.44	7.81
CPS	0.91	1.26	2.73	5.15	7.08	6.75
IPS	1.41	1.36	2.82	5.36	7.35	6.90

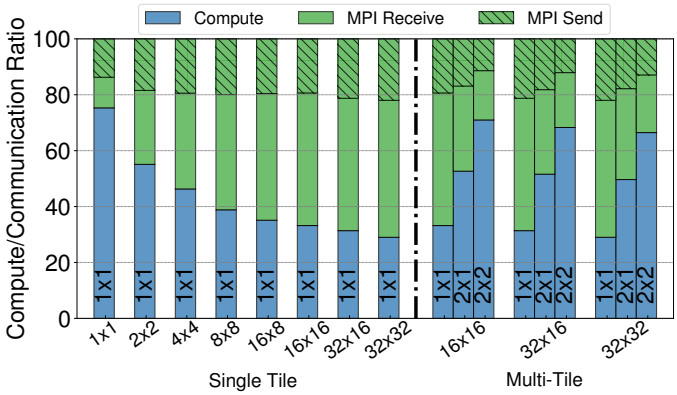


Fig. 4: Communication/Compute Ratios for up to 1,024 tiles (32×32). With STGs (1x1) and with MTGs of 2×1 and 2×2.

E. Metro-MPI with Multi-Tile Granule

For instances where the available computational resources cannot fit the required number of processes, we explore Metro-MPI with MTGs. First, we study STG Metro-MPI to determine the ratio of computation to MPI communication. Then, we evaluate the performance and efficiency of MTGs.

1) MPI Communication Overhead: Moving from a sequential simulation to a parallel one adds the cost of inter-process communication, usually a non-negligible aspect of scaling any application. Figure 4 shows the ratio of simulation time used for computation (in blue) to MPI communication (in green) when scaling STG Metro-MPI from one node to multiple nodes. We also distinguish between the time spent sending MPI messages (striped green) and receiving MPI messages (plain green).

We see the MPI communication overhead become a significant portion of simulation time as the design size grows. When simulating a single tile (1×1), the MPI overhead is about 25%. For 4×4, we observe that it already dominates with respect to the compute portion of the simulation. The overhead plateaus for large configurations, reach 71% for 32×32, which uses 22 compute nodes. As the number of nodes increases, this behaviour is expected with lockstep communication.

When looking at the division of the MPI overhead, *receive* dominates over *send*. This is because sending a message is asynchronous, while waiting for a message is synchronous (to ensure a granule has received all incoming messages for the given cycle). This is a limitation introduced by simulating partitions in lockstep.

TABLE IV: Multi-tile Metro-MPI performance results using Verilator: (i) simulation time slowdown and (ii) simulated KIPS per host core used, both normalised to the 1×1 STG configuration; and (iii) actual simulated KIPS.

Metrics	NoC Configs								
	16x16			32x16			32x32		
	1x1	2x1	2x2	1x1	2x1	2x2	1x1	2x1	2x2
Slowdown	1.0	1.79	3.39	1.0	1.69	3.19	1.0	1.77	2.55
KIPS/core	1.0	1.11	1.16	1.0	1.18	1.25	1.0	1.13	1.56
KIPS	745	414	220	1318	780	413	2698	1528	1057

2) Multi-Tile MPI Overhead: Figure 4 shows the same MPI overhead study with three MTGs (1×1, 2×1 and 2×2) for the three largest chips (16×16, 32×16, and 32×32). As the MTG grows, the computation ratio increases, since each MPI process is simulating more tiles per cycle. The computation part (blue) increases and reaches around 50% for the 2×1 granule, and up to 70% for the 2×2 granule. The communication component (green) is also affected by the MTG, decreasing as the number of tiles in the granule grows. The communication ratio spent on MPI Receive (plain green) decreases substantially. For 16×16: from almost 50% in STG 1×1, to around 30% in MTG 2×1, and finally below 20% for the 2×2 MTG. The time spent on MPI Send also decreases from 20% to 11%. Other chip sizes follow similar trends.

We can conclude that MTGs alleviate MPI overhead for large chips. Depending on the HPC infrastructure's network, these overheads can represent a large portion of the simulation time. MTGs thus enable large chip simulations with fewer nodes.

3) Multi-Tile Efficiency Evaluation: To determine the efficiency of MTGs in Metro-MPI we evaluate simulation time (seconds), KIPS, and efficiency, across three chip sizes: 16×16, 32×16, and 32×32, and use three granules of size 1×1, 2×1, and 2×2. Table IV presents simulation time slowdowns normalised to the 1×1 MTG for each chip size. We observe that the simulation time slowdown for the 2×1 MTG is similar for all chip sizes: between $1.69\times$ and $1.79\times$. However, we can see that for the 2×2 MTG there is less slowdown as the chip grows in size: from $3.39\times$ for 16×16 to $2.55\times$ for 32×32. As expected, employing MTGs has a negative impact on simulation time, since more work is done per MPI process. However, the slowdown is not linear with respect to the granule size, which means simulations are more efficient with respect to the computational resources they employ.

Table IV also presents an efficiency metric: KIPS (simulation speed) divided by host cores (MPI processes, computational resources) used. Results are normalised to the STG for each chip size. When using a MTG of 2×1 we obtain efficiency improvements of $1.11\times$, $1.18\times$, and $1.13\times$ for chip sizes 16×16, 32×16, and 32×32, respectively. Using the largest MTG of 2×2 yields even higher efficiencies, with up to $1.56\times$ for the 32×32 NoC. These improvements in simulation efficiency stem from the MPI communication overhead reductions shown in Figure 4, and prove that MTGs are useful both to lift the constraints imposed by a given HPC infrastructure and to run multiple simulations in parallel with better overall throughput.

Finally, Table IV shows simulated KIPS. These results follow the same trends already discussed for simulation time slowdown, validating previous observations.

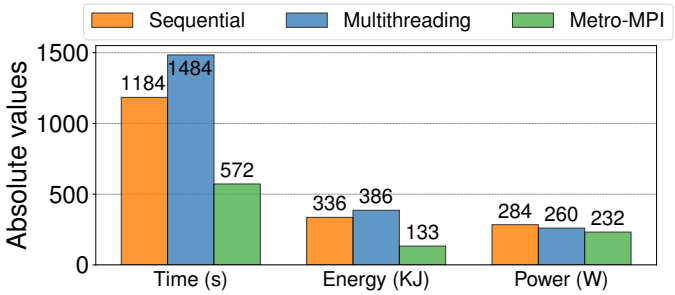


Fig. 5: Metro-MPI Energy Comparison vs Multithreaded and Sequential simulation for a fixed amount of work and resources.

F. Energy and Power Analysis

Figure 5 shows execution time (seconds), total energy (Kilo-Joules) and average power (Watts) for *sequential*, *automatic multithreading by Verilator*, and *Metro-MPI* configurations when executing a fixed amount of work, specifically 32 simulations of the 8×4 NoC. We have also fixed the amount of computational resources to one node of our HPC infrastructure described in Section III. In this manner, this experiment resembles the problem of running a regression which usually has a fixed amount of work with a fixed amount of resources. *Sequential* executes 32 individual sequential simulations in parallel on different cores; for *multithreading* we have evaluated different thread counts, fitting as many simulations as possible, and show the configuration that yields the best results; finally, *Metro-MPI* parallelises each individual simulation using STG processes, running one simulation at a time.

In terms of execution time *Metro-MPI* runs faster than *sequential* and *multithreading* by $2.07\times$ and $2.59\times$ respectively. In the case of total energy consumption, *Metro-MPI* consumes $2.53\times$ and $2.90\times$ less energy against *sequential* and *multithreading*. Finally, comparing average power, *Metro-MPI* requires $1.22\times$ and $1.12\times$ less power than *sequential* and *multithreading*. We can conclude Metro-MPI is a better choice for running regressions, saving time and energy.

G. Metro-MPI Heterogeneity

Our methodology is general enough for heterogeneous designs. Figure 2 shows a prototype we have built where Metro-MPI is used with different protocols and the tiles could be simulated using either STGs or MTGs. This OpenPiton/BYOC design has heterogeneous tiles containing different cores or accelerators connected via the Transaction-Response Interface [5] and protocols like AXI, one of the most common communication protocols. In our simulation, the chip employs homogeneous tiles with STGs and the chipset has devices in several independent processes: a main process with UART, DRAM controller, and debug units; and two additional processes independently connected through AXI over Metro-MPI, one for a MIAOW GPGPU [4] and one for the bootrom.

Heterogeneity of Simulators: The generic connection through MPI offers the ability to have a parallel simulation with different hardware modules compiled in different simulators. Such heterogeneous simulations remove difficulties that proprietary simulator licenses can create.

V. Related Work

State-of-the-art approaches to reduce simulation time are mostly focused on high-level simulators and make a trade-off between speed and accuracy. Such solutions rely either on sampling [14], [19], [23] or parallelisation [3], [13], [18] techniques to reduce the number of instructions to simulate. They are far from the cycle accuracy of RTL, making them suitable for other types of experiments: e.g. fast application performance estimations in computer architecture.

In the case of RTL simulations, there are proposals that use MPI, OpenMP or a custom simulator to accelerate simulation. Tariq et al. [2] make use of domain partitioning and OpenMP, obtaining up to a $3.3\times$ simulation time speedup. Essent [8] proposes a new simulator that uses an intermediate language for hardware: FIRRTL [15], which accelerates simulations with practical techniques to reuse and avoid doing extra work. Verilator can automatically partition a design with pthreads, enabling multithreaded simulations out-of-the-box. These proposals are mostly single-node and thus largely orthogonal, meaning they could be adopted alongside Metro-MPI.

PVSim [17] uses partitioning and an optimistic asynchronous simulation algorithm for fine-grained parallel HDL simulation, delivering up to $4.64\times$ using 8 processes. Though both use MPI, Metro-MPI relies on a manual partitioning, exploiting today's chips' structure (which are more than $100\times$ larger than when PVSim was designed) to identify boundaries. The design is then simulated in lockstep, with large enough partitions that communication time does not dominate the simulation performance. To the best of our knowledge, PVSim is not publicly available, making direct comparison impossible.

Regarding improving simulations using FPGAs: FAST [9] is a hybrid approach (CPU+FPGA), RAMP Gold [21] speeds up manycore simulation up to 64 cores, and Firesim [16] is a simulation platform that uses Amazon EC2 F1 instances to provide usability and avoid big investments in hardware equipment. These are very useful for late-stage hardware design but are more tedious to use and modify, often requiring long FPGA build times, and have limitations with the design's size.

VI. Conclusions

We have demonstrated the value of HPC techniques for RTL simulation. For the first time, we obtain 2.7 MIPS on a 10B+ transistor (1,024 core) scale RTL simulation, something only available before on high-level software simulators or FPGA emulators. Making use of MPI and the natural partitioning of hardware blocks in a design, we obtain a remarkable speedup over sequential Verilator simulation of up to $135.98\times$ on an 8×8 chip. We also compared Metro-MPI to Verilator's automatic multithreaded partitioning with a $9.29\times$ speedup on an 8×4 chip. Since chips with such large core counts are already commercially available, Metro-MPI is a solution for high-speed, large-scale RTL simulation in practice. This work is open-source, found at: https://github.com/metro-mpi.

Acknowledgment

This work has been partially supported by the Spanish Ministry of Economy and Competitiveness (contract PID2019-107255GB-C21), by the Generalitat de Catalunya (contract 2017-SGR-1328), by the European Union within the framework of the ERDF of Catalonia 2014-2020 under the DRAC project [001-P-001723], and by the Arm-BSC Center of Excellence. G. López-Paradís has been supported by the Generalitat de Catalunya through a FI fellowship 2021FI-B00994 and GSoC 2021, and M. Moretó by a Ramon y Cajal fellowship no. RYC-2016-21104. A. Armejach is a Serra Hunter Fellow.

References

[1] AWS Graviton Processor. https://aws.amazon.com/ec2/graviton/.
[2] T. B. Ahmad and M. Ciesielski. Parallel multi-core verilog hdl simulation using domain partitioning. In ISVLSI, 2014.
[3] E. Argollo, A. Falcón, P. Faraboschi, M. Monchiero, and D. Ortega. Cotson: Infrastructure for full system simulation. SIGOPS Oper. Syst. Rev., 2009.
[4] R. Balasubramanian, V. Gangadhar, Z. Guo, C.-H. Ho, C. Joseph, J. Menon, M. P. Drumond, Paul, and et al. Enabling gpgpu low-level hardware explorations with miaow: An open-source rtl implementation of a gpgpu. ACM Trans. Archit. Code Optim., 12(2), jun 2015.
[5] J. Balkind, K. Lim, M. Schaffner, F. Gao, G. Chirkov, A. Li, A. Lavrov, T. M. Nguyen, Y. Fu, F. Zaruba, K. Gulati, L. Benini, and D. Wentzlaff. BYOC: A "Bring Your Own Core" framework for heterogeneous-ISA research. In ASPLOS, 2020.
[6] J. Balkind, M. McKeown, Y. Fu, T. Nguyen, Y. Zhou, A. Lavrov, M. Shahrad, A. Fuchs, S. Payne, X. Liang, M. Matl, and D. Wentzlaff. OpenPiton: An open source manycore research framework. In ASPLOS. ACM, 2016.
[7] S. Beamer. A case for accelerating software RTL simulation. IEEE Micro, 2020.
[8] S. Beamer and D. Donofrio. Efficiently exploiting low activity factors to accelerate RTL simulation. In DAC, 2020.
[9] D. Chiou, D. Sunwoo, J. Kim, N. A. Patil, W. Reinhart, D. E. Johnson, J. Keefe, and H. Angepat. FPGA-accelerated simulation technologies (FAST): Fast, full-system, cycle-accurate simulators. In MICRO, 2007.
[10] M. Christensen, T. Sherwood, J. Balkind, and B. Hardekopf. Wire sorts: A language abstraction for safe hardware composition. In PLDI, 2021.
[11] D. Ditzel. Accelerating ML recommendation with over a thousand RISC-V/tensor processors on Esperanto's ET-SoC-1 chip. In Hot Chips Symposium, 2021.
[12] M. P. Forum. MPI: A message-passing interface standard. Technical report, 1994.
[13] Y. Fu and D. Wentzlaff. PriME: A parallel and distributed simulator for thousand-core chips. In ISPASS, pages 116–125, 2014.
[14] T. Grass, C. Allande, A. Armejach, A. Rico, E. Ayguadé, J. Labarta, M. Valero, M. Casas, and M. Moreto. MUSA: A multi-level simulation approach for next-generation HPC machines. In SC, 2016.
[15] A. Izraelevitz, J. Koenig, P. Li, R. Lin, A. Wang, A. Magyar, D. Kim, and et al. Reusability is FIRRTL ground: Hardware construction languages, compiler frameworks, and transformations. In ICCAD, 2017.
[16] S. Karandikar, H. Mao, D. Kim, D. Biancolin, A. Amid, D. Lee, N. Pemberton, E. Amaro, C. Schmidt, A. Chopra, Q. Huang, K. Kovacs, B. Nikolic, R. Katz, J. Bachrach, and K. Asanović. Firesim: FPGA-accelerated cycle-exact scale-out system simulation in the public cloud. In ISCA, 2018.
[17] T. Li, Y. Guo, and S.-K. Li. Design and implementation of a parallel verilog simulator: PVSim. In VLSI Design, 2004.
[18] D. Sanchez and C. Kozyrakis. ZSim: Fast and accurate microarchitectural simulation of thousand-core systems. In ISCA, 2013.
[19] T. Sherwood, E. Perelman, G. Hamerly, and B. Calder. Automatically characterizing large scale program behavior. In ASPLOS, 2002.
[20] W. Snyder. Verilator. https://www.veripool.org/wiki/verilator.
[21] Z. Tan, A. Waterman, R. Avizienis, Y. Lee, H. Cook, D. Patterson, and K. Asanović. RAMP Gold: An FPGA-based architecture simulator for multiprocessors. In DAC, 2010.
[22] M. B. Taylor. Basejump stl: Systemverilog needs a standard template library for hardware design. In DAC, 2018.
[23] R. E. Wunderlich, T. F. Wenisch, B. Falsafi, and J. C. Hoe. SMARTS: Accelerating microarchitecture simulation via rigorous statistical sampling. In ISCA, 2003.