A Quantitative Guide to Navigate Speed/Accuracy Tradeoffs in System Level Design of RISC-V Processor Grids

Lars Luchterhandt*, Vivek Govindasamy†, Yutong Wang†, Christoph Scheytt*, Wolfgang Mueller*, Rainer Dömer†
*Heinz Nixdorf Institute, Paderborn University, Paderborn, Germany
†Center for Embedded and Cyber-physical Systems, University of California, Irvine, CA, USA

***Abstract*—Even when following a well-structured top-down design methodology, system designers regularly face obstacles and pitfalls posed by speed/accuracy tradeoffs in modeling and simulation of complex hardware and software systems. Across the abstraction levels, modeling details grow exponentially, while simulation speed decreases by multiple orders of magnitude. To quantify these effects, we systematically generate, simulate, and evaluate grid-based systems-on-chip in a top-down open-source based tool flow. We map two parallel software applications onto a scalable grid of RISC-V processors and successively refine and validate the models at lower abstraction levels, namely TLM, ISS, RTL, and FPGA. Our comprehensive experimental evaluation over five abstraction levels quantifies the speed-accuracy tradeoffs in simulator build and run times. In addition to its educational value, our work can guide the system designer on an efficient path to a cycle-accurate software simulation on fully constructed hardware.**

***Index Terms*—System-on-chip, Codesign, RISC-V, SystemC TLM-2.0, SystemVerilog.**

I. Introduction and Motivation

Modern design methodologies starting at the Electronic System Level (ESL) are founded on the well-known principle of higher abstraction for lowering complexity. Typically, such model-based design flows start at the system level, where software and hardware are specified together, followed by a top-down codesign flow where the system model is then successively refined (e.g. [1]–[3]) and brought down step by step to the clock-cycle accurate Register Transfer Level (RTL), from where standard hardware digital design processes proceed to FPGA and ASIC implementations. Throughout these codesign flows, system designers regularly encounter tradeoffs between incompatible goals, such as the speed/accuracy tradeoff in model simulation, where the goals of high speed and high accuracy contradict each other. The system designer must then choose between a model with high speed and low accuracy, or vice versa. If both features are needed to make the right design decisions, two or more models must be built and analyzed, significantly increasing the cost and duration of the overall system design process.

While the speed/accuracy tradeoffs in system modeling are generally well understood and a number of specific examples are reported ad hoc in the literature (e.g. [4]–[6]), consistent and reliable figures for the right model selection are few to none, especially when we want to consider implications across multiple abstraction levels.

In this work, we systematically design, model, simulate, and evaluate two non-trivial benchmark applications across five major abstraction levels, namely functional multi-threaded C++, SystemC Transaction Level Modeling (TLM-2.0), Instruction Set Simulation (ISS), Register-Transfer Level (RTL), and a first Field-Programmable Gate Array (FPGA) prototype implementation. At each level, we quantify and report the speed/accuracy tradeoffs in build and run time. Our comprehensive comparison and analysis can provide quantitative insights and guide the system designer in better navigating the paths to an efficient system implementation.

Aiming at medium-to-large SoC designs, we map two highly parallel software applications onto a scalable grid of RISC-V processors. Inspired by processor arrays, such as parallel transputers [7], the recently proposed Grid of Processing Cells (GPC) [8] aims to avoid the well-known memory bottleneck of traditional multiprocessor architectures by using separate on-chip memories that are accessible only by their paired processor and a few immediate neighbors.

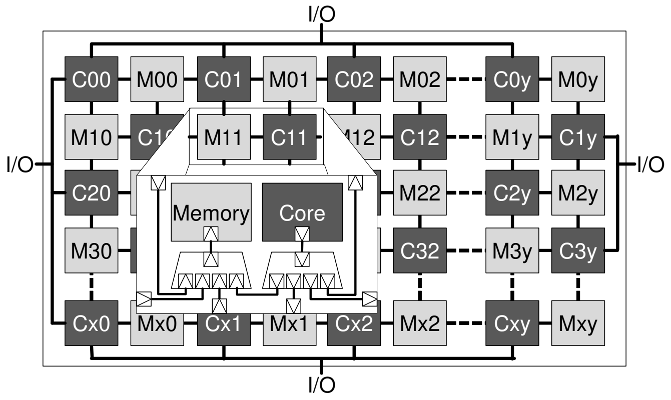

Fig. 1. Checkerboard Grid of Processing Cells (GPC) [8] with processor-memory pairs connected in North, East, South, and West directions.

The GPC platform, as illustrated by its checkerboard variant in Fig. 1, is a scalable 2D array of on-chip processing cells. Each cell consists of a processor Core and a local Memory, as well as local interconnect consisting of address decoders and memory arbiters, shown here with SystemC TLM-2.0 socket connectivity.

Note that individual processor cores can only reach memory addresses within their own cell and the immediate local neighborhood. Specifically, each core has priority access to its own memory and secondary access to the memories of its immediate neighbors, but nothing further. Processing cells are intentionally limited to local interactions only, enabling the true scaling of the grid.

This work provides three main contributions.

1) We propose a seamless top-down design flow for hardware/software codesign with open source tools.
2) We generate scalable grids of RISC-V processors at five abstraction levels, namely functional C++, SystemC TLM-2.0, ISS, RTL, and FPGA. Our experimental evaluation quantifies the speed/accuracy tradeoffs in build and run time, providing systematic and comprehensive educational value across major abstraction levels.
3) Our quantitative evaluation of two parallel benchmark applications can guide the system designer in effectively navigating the large design space, avoiding obstacles and pitfalls, and reaching an efficient SoC implementation.

The remainder of this paper is organized as follows. After a brief review of related work in Section I-A, we describe the five covered abstraction levels and their general purpose in Section II. We then detail our open-source based tool flow with model generation in Section III and evaluate it systematically based on two benchmark applications in Section IV-B and Section IV-C. We finally conclude in Section V with the lessons learned and future work.

A. Related Work

In this section, we briefly review prior work on three topics that are relevant to this article, namely open source EDA software, array-based many-core architectures, and modeling with RISC-V processors.

While professional chip design environments are dominated by only a few EDA vendors, the development and use of open-source EDA software is fragmented but gaining more traction. Today, several open-source RTL models are available that come with design flow implementations with support for open-source and commercial EDA tools. Some provide partial automation support from RTL to FPGA and chip layout based on open-source Makefiles and scripts. For example, PULP (Parallel Ultra-Low-Power) stands for several RISC-V ecosystems for different RISC-V SoCs, such as OpenPULP, PULPino, and PULPissimo with heterogeneous components [9]. Similarly, Chipyard [10], a framework for chip development of RISC-V based processor systems with configurable cores, offers seamless design automation for RTL simulation (Verilator, FireSim), FPGA prototyping, and VLSI chip implementation (Hammer). Chipyard provides support for commercial and open-source tools (e.g. OpenLane [11]) and several commercial and open-source Process Development Kits (PDKs).

Array-based many-core architectures are based on the principles of multiple instructions and multiple data streams (MIMD) [12]. The concepts of configurable processor grids for massively parallel processing stem from the principles of transputers [7], [13], introduced in the 80s to overcome the classic Von Neumann bottleneck [14]. As such, transputers rely on the principles of a nothing-shared architecture with the advantage of high scalability and configurable identical cells that can be customized for individual applications. A transputer is a single 32-bit RISC microprocessor with local memory and four serial message-passing IOs, establishing communication links to adjacent processors. Transputers served as building blocks of massively parallel supercomputers with up to several hundred nodes [13]. Recent examples of scalable many-core architectures include the Raw Processor [15], and the Tile64 [16] with 64 tiles and a configurable switching network. The recently proposed GPC platform has been explored in early case studies using the Chipyard framework [17], [18].

In system design and modeling with SystemC [2], [19], [20], the use of RISC-V processors is a popular choice due to the free availability of simulators [21] and their seamless integration with TLM-2.0 models [5], [22]. However, models across multiple abstraction levels are sparse and, to the best of our knowledge, no systematic evaluation is available. In contrast, this work presents a full-scale software and hardware implementation with a seamless design flow over five abstraction levels with two non-trivial applications on scaled grids of RISC-V processors.

II. Modeling and Abstraction Levels

Before we go into tool flow details, we review the major abstraction levels and their main concerns in system-level simulation. In this work, we cover five different abstraction levels, namely (1) functional multi-threaded C++, (2) Transaction Level Modeling (TLM-2.0), (3) Instruction Set Simulation (ISS), (4) Register Transfer Level (RTL), and (5) Field Programmable Gate Array (FPGA), as summarized in Table I.

A. Functional Model

At the highest abstraction level, the system designer is generally concerned with the algorithmic functionality of the application. An executable specification is designed to validate the correct functionality, make algorithm and data structure choices, and optimize the efficiency of parallelism, data flow, and communication. Without loss of generality, we assume that the functional model is specified as a multi-threaded C++ program where a number of tasks perform the application's functions and communicate via standard communication and synchronization methods, such as queues and condition variables. The model is fully functional, but there is no notion of timing or structure. These properties are added and refined step by step in the following models using a top-down design approach.

B. TLM-2.0

The first design decisions concern the structural system architecture and include platform allocation and hardware/software partitioning. Specifically, we allocate a GPC of suitable size and map the functional tasks to processors in the cells. The system model is specified using SystemC TLM-2.0 and includes the GPC hardware architecture and the host-compiled software for every cell. Notably, the model contains explicit memory

TABLE I
SYSTEM MODELING AT FIVE ABSTRACTION LEVELS WITH THEIR PRIMARY CONCERNS, LANGUAGES, AND TOOLS

Model	Abstraction Level	Accuracy	Concern	Software		Hardware	
				Language	Tool	Language	Tool
Functional	Untimed	Functions	Algorithm	Multi-threaded C++	g++	-	-
TLM-2.0	Loosely-timed	Transactions	System architecture	C++	g++	SystemC	g++
ISS	Approximately-timed	Instructions	Instruction set	C++, Assembly	g++-rv32	SystemC	g++
RTL	Clock-cycle accurate	Clock cycles	Critical path	C++, Assembly	g++-rv32	SystemVerilog	Verilator
FPGA	Real-time	Physical time	Resources	C++, Assembly	g++-rv32	SystemVerilog	Vivado

components and is address-accurate for load/store transactions of data, so that, for example, contention for data accesses can be accurately observed already at this high abstraction level [5], [23]. On the other hand, the model is only loosely timed. Due to the host-compiled software, execution time is only a rough estimation.

C. ISS

For increased accuracy, we next generate an instruction set simulation model. Here, we cross-compile the software of every cell in the grid for a chosen target processor. The target-executable files are loaded into the instruction memories at the start of simulation and then executed using an interpreted simulator [21]. The main purpose of this model is the selection and configuration of the instruction set architecture (ISA) and corresponding code optimization for every cell.

In this work, we specifically use two RISC-V architectures, namely RV32IMAC_Zicsr (for TokenX, see Section IV-B) and RV32IMAFC_Zicsr (for ParticleSim, see Section IV-C), where the latter offers hardware support for single-precision floating-point arithmetic. Our bare-metal software contains no operating system, only a minimal boot function (assembly code `bootstrap.s`) and a custom queue for communication with neighbors via push and pop methods.

The ISS software is fully accurate in addresses, data, and instructions. This allows for optimizing the ISA and analyzing memory access behavior [22]. While the number of instructions executed is accurate in this model, their delay is still an approximation. In comparison to the previous TLM, timing accuracy improves, but execution speed decreases significantly.

D. RTL

The RTL model is the first model with accurate timing measured in clock cycles. It is also the first fully detailed hardware model that is bit-accurate in pins and signals (observable as waveforms), and serves as the common entry stage into FPGA prototyping and physical chip implementation. The main purpose at RTL is the definition of clock cycles, driven by concerns about minimizing the length of the critical path in logic design optimization.

While SystemC [24] is an option, SystemVerilog [25] is our language of choice to describe and simulate this system model. Note that while the hardware is majorly refined at RTL, the software binary remains unmodified from ISS onward.

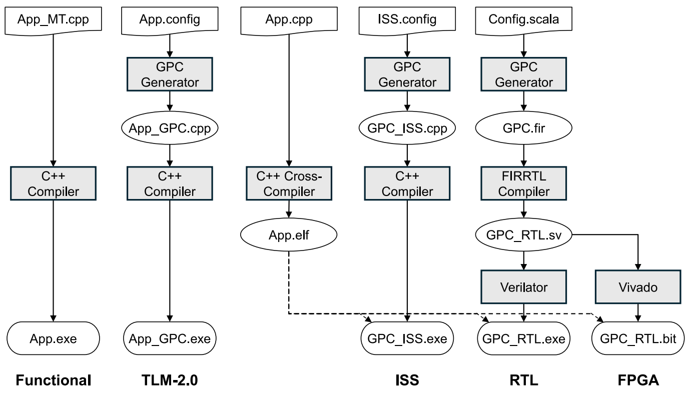

Fig. 2. Conceptual tool flow to generate models at five abstraction levels

Since the entire model is now clock-cycle accurate, precise timing analysis is finally possible, however, at the price of severely reduced simulation speed. For acceptable run times during debugging, typically only a limited number of instructions can be executed. Longer software execution can only be observed in reasonable time at higher abstraction levels, or on the following FPGA platform.

E. FPGA

For this work, we target a prototype implementation on an FPGA. Here, the entire system runs in real-time on the physical FPGA hardware as we will see in Section IV.

The high-speed execution comes at the price of a suitable FPGA board and an increase in build time, since the model needs to be synthesized for the available FPGA resources, i.e. look-up tables (LUTs) and registers. In contrast to all prior abstraction levels, where hardware was represented virtually, the FPGA model uses real hardware resources that need to be allocated, configured, and minimized. This is typically called hardware prototyping, as it can be considered the step toward the final physical chip implementation and layout.

III. TOP-DOWN CODESIGN FLOW

From the functional level down to FPGA, we use model generation to scale and construct each executable model for codesign and co-simulation of software and hardware. Here, we use open-source EDA tools wherever possible [26]. Our tool flow for the five abstraction levels is shown in Fig. 2.

A. Functional Specification

Our design flow starts with the specification of the application in form of multi-threaded C++ source code. Using a regular C++ compiler on a Linux host, it can be executed for fast functional validation, as shown on the left of Fig. 2.

B. Transaction Level Modeling

To define and simulate the overall system architecture, we generate a suitably sized GPC model in SystemC TLM-2.0 using our high-level GPC generator implemented in Python. With the application tasks mapped to its cells, the GPC model can then be compiled and simulated in the same Linux environment as the prior functional model (Fig. 2 column 2).

C. Instruction Set Simulation

For the ISS model, we separate the software compilation and the hardware construction. We export the software code for each cell and compile it using a C++ cross-compiler. The cross-compiler runs on the simulation host, but generates code for the chosen target processor configuration, e.g. RV32IMAC. The cross-compiled binary files are stored in executable and linkable format (ELF) and will be loaded into the on-chip memories of the hardware platform at the start of the simulation (Fig. 2 column 3). Note that, to some extent, the same ELF files can be loaded into the ISS, RTL, and FPGA hardware platforms, as illustrated by the dashed lines at the bottom of Fig. 2.

To simulate the configured target ISA on the host workstation, we use an interpreted open-source ISS [21] integrated in our SystemC model. Configured for the chosen target ISA, our GPC generator replaces the host-compiled tasks of the TLM-2.0 model with a corresponding ISS instance in each cell. The platform model can then be compiled and simulated on a simulation host at ISS level (Fig. 2 column 4).

D. RTL Hardware Generation

To obtain a clock-cycle accurate model, we use our GPC generator for RTL, which is a custom extension of the Rocket chip generator [27] in the Chipyard framework [10]. This replaces the prior SystemC GPC model with a fully constructed hardware model in SystemVerilog.

Our GPC generator for RTL, shown in Fig. 2 column 5, is written in the hardware construction language Chisel [28], embedded in the high-level programming language Scala. It emits FIRRTL [29] code for the desired GPC, properly configured for the chosen grid size, memory size, and RISC-V ISA with or without FPU support. With Chipyard offering multiple design flows, the FIRRTL circuit compiler then generates an RTL model in SystemVerilog, tailored to the selected flow. For RTL simulation, we use the open-source SystemVerilog simulator Verilator [30]. Verilator translates the SystemVerilog model into an intermediate multi-threaded C++ model, which can then be executed after compilation.

E. FPGA Synthesis

To arrive at a first hardware implementation, we synthesize the RTL design for the AMD VCU108 FPGA using AMD

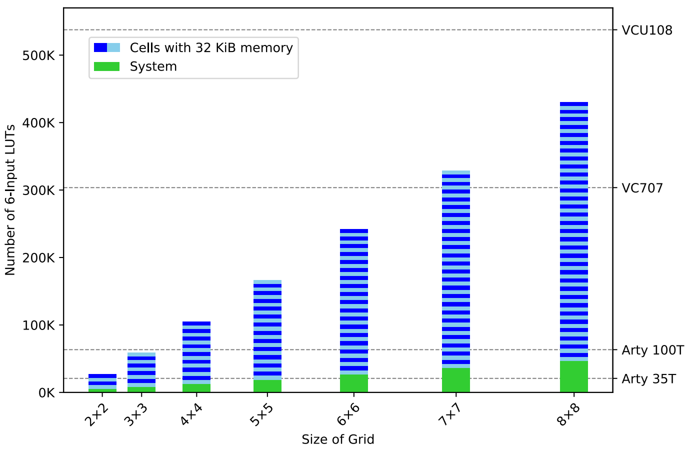

Fig. 3. FPGA resources needed when scaling quadratic GPCs

Vivado[1]. This hardware synthesis step produces a bitstream that, when loaded onto the FPGA board, configures it as the chosen GPC hardware model. For execution on this hardware prototype, the cross-compiled ELF files of the application are loaded into the scratchpad memories via JTAG and then executed by the GPC configured on the FPGA.

IV. Practice and Evaluation

To evaluate our codesign flow, we conduct an extensive set of experiments that build, simulate, and evaluate models for two applications over five abstraction levels. After briefly describing our experimental setup in Section IV-A, we begin in Section IV-B with the introduced top-down design flow using TokenX, our first scalable benchmark designed with constant computational load over its run time. Thereafter, we introduce a second benchmark, ParticleSim, with real-world variable computational load in Section IV-C. Based on this we discuss obstacles and pitfalls that arise when design flows deviate from the designer's initial expectations.

A. Host Platforms and FPGA Setup

For our experiments, we compile and simulate the functional, TLM-2.0, and ISS models on a Linux workstation with an Intel Xeon E-2388G CPU with 16 cores at 3.2 GHz. At RTL, we construct and simulate our models on a High Performance Computing (HPC) cluster. Each CPU node in the cluster is equipped with two AMD EPYC 7763 processors, both featuring 64 cores at 2.45 GHz. We also synthesize the FPGA bitstreams on the HPC cluster.

Our FPGA experiments are then performed on an AMD VCU108 FPGA board at 100 MHz. This board has sufficient LUT and register resources for a GPC up to size 8×8 when the RISC-V cells are configured with 32 KiB of memory and without FPUs. Fig. 3 shows the number of 6-input LUTs required for grid configurations from 2×2 up to 8×8. It also compares the capacity with other FPGA boards and provides a breakdown of LUT utilization between the individual processing cells and the surrounding SoC infrastructure. The progressively increasing bars clearly show the linear growth in

[1]Note that Vivado is the only commercial, closed-source tool in our flow.

resource requirements as the GPC scales with the number of cells.

B. Top-Down Reference Design Flow

We first evaluate the presented design flow using a benchmark application with regular structure and behavior. This benchmark is a simple example that exercises messaging on the GPC and carries an evenly distributed workload. We generate five models for 6×6 and 8×8 GPC targets and then analyze the simulation results in detail.

1) TokenX Benchmark: Our TokenX application is a scalable benchmark for communication on a grid with a constant computational load across all cells. TokenX starts with a user-defined number of tokens in each cell and continuously exchanges these tokens with neighbor cells in a pseudo-random but repeatable fashion. It is deterministic and thus suitable for the systematic validation of data transfers on a GPC.

The execution length of TokenX can be adjusted by a user-defined number of *steps*. Each step is one iteration in the TokenX main loop and performs two functions: it computes tokens and then exchanges them with neighbors. By adjusting the number of steps, we can control the execution length of the application to ensure that simulator run times remain acceptable at all abstraction levels. This is particularly important at the ISS level and RTL, where simulators are slow.

2) Experimental Results: To evaluate the speed-accuracy trade-offs of different simulation platforms at different abstraction levels, we run TokenX on grid sizes 6×6 and 8×8. Table II summarizes our results and quantifies the cost for conducting the benchmark on each abstraction level in build time and run time.

In general, the table shows an increase in build time for each refinement down to the FPGA implementation. Lower levels of abstraction result in more detailed models. This increase in complexity causes longer build times. While the functional, TLM-2.0, and ISS models only take up to a few seconds to build, the RTL model takes many minutes. With the FPGA implementations being the most complex ones in the table, their build time is within hours. There is also an increase in build time visible when comparing the 6×6 GPC to the 8×8 GPC. A larger grid size results in more hardware to construct and thus also increases the model's complexity. As expected, the build times of both functional models are identical. Starting at TLM-2.0, there is a slight increase of 21% in build time visible for the 8×8 GPC. With more detailed models, this overhead further increases. The ISS model already shows a 45% increase. At RTL, the build time is increased by a factor of 2.6, and by a factor of 2.4 for FPGA.

As known from textbooks, the cost of simulation increases by orders of magnitude when lowering the abstraction level. To keep run times reasonable at each abstraction level, we adjust the number of steps for TokenX accordingly. Given its linear impact on execution length, and thus on simulator run time, the number of steps provides smooth, fine-grained control.

In the following, we take a look at the run time of TokenX on the 8×8 GPC. Starting at the functional level, we execute a total of 1M steps, resulting in a run time of 22.5 seconds. According to our measurements, TLM-2.0 is only one order of magnitude slower in runtime, caused by the lack of parallelism in SystemC [31], [32] and the explicit modeling of memory transactions. Therefore, we keep the number of steps constant at 1M and measure a run time of nearly three minutes, representing a 7.6x slowdown compared to the functional model. At ISS, we switch from host-compiled to RISC-V target-compiled software with instruction-accurate simulation. Our experiments show an increase in runtime by up to two orders of magnitude from TLM-2.0 to ISS. Thus, we reduce the number of steps by a factor of 100 accordingly (10K steps). As intended, the resulting two-minute run time at the ISS level remains within a similar duration as TLM-2.0. When accounting for the reduced number of steps, we observe a slowdown factor of 68 when comparing TLM-2.0 to ISS. As expected, cycle-accurate RTL simulation is the most expensive simulation in the entire design flow with respect to run time. From ISS to RTL, we observe a run time increase of up to three orders of magnitude. To maintain a reasonable run time, we again reduce the number of steps by a factor of 100 (100 steps). The resulting run time of 34 minutes matches the expectation, as it is only one magnitude slower than the ISS. Considering the reduction in number of steps, there is an increase in run time by a factor of 1.8k from ISS to RTL. In contrast to the other abstraction levels down to RTL, the FPGA operating at 100 MHz achieves performance similar to that observed at the initial functional implementation. Therefore, we run the full amount of 1M steps again. With a run time of less than 8 seconds, the FPGA even surpasses the functional model's performance by a factor of 2.9.

The run time of the 6×6 grid shows similar order-of-magnitude differences across the abstraction levels. Compared to the 8×8 grid, the only distinction is a slight reduction in run time: by a factor of 1.7 at the functional level, 2x for TLM and ISS, and 3.2x at RTL. This indicates that the run-time overhead introduced by increasing the grid size becomes more significant at lower abstraction levels.

The simulation column in Table II presents measured metrics relevant to the given abstraction level. Since the functional model does not involve simulation, no corresponding metrics are reported. Starting at TLM-2.0, we report the measured number of transactions during execution. Beginning with ISS, we include the number of retired instructions, as all subsequent models run target-compiled software. When relating the number of retired instructions to the executed steps, only minor deviations are observed across the abstraction levels. From ISS to RTL, a noticeable deviation of 7–8% appears, while the deviation from RTL to FPGA is negligible at 1–3%. The larger deviation from ISS to RTL/FPGA is expected, as ISS simulates individual instructions, whereas the RTL/FPGA models have detailed processor pipelines and memories. To further compare simulation speed, we report the millions of instructions per second (MIPS), reflecting the previously observed orders of magnitude. At RTL and on FPGA, where execution is driven by a system clock, the number of elapsed clock cycles becomes available. As anticipated, this number is nearly identical between RTL and FPGA, with less than 1% deviation. Finally,

TABLE II
EXPERIMENTAL RESULTS FOR TokenX ON 6×6 AND 8×8 GPC TARGETS

Model	Steps	6×6 GPC				8×8 GPC			
		Build Time	Run Time	MIPS	Simulation	Build Time	Run Time	MIPS	Simulation
Functional	1M	0.33s	12.62s	-	-	0.32s	22.46s	-	-
TLM-2.0	1M	3.50s	1:23m	-	240M transactions	4.24s	2:51m	-	448M transactions
ISS	10K	4.14s	55.86s	0.0479	2.68M instructions	6.01s	1:56m	0.0234	2.71M instructions
RTL	100	15:00m	10:34m	0.00004	25.0K instructions 74.6K clock cycles	39:21m	33:59m	0.00001	25.1K instructions 74.7K clock cycles
FPGA	1M	1:04h	7.43s	33.9	252M instructions 743M clock cycles 242K LUTs	2:32h	7.73s	32.4	258M instructions 746M clock cycles 430K LUTs

TABLE III
EXPERIMENTAL RESULTS FOR ParticleSim ON 6×6 AND 8×8 GPC TARGETS

Model	Steps	6×6 GPC				8×8 GPC			
		Build Time	Run Time	MIPS	Simulation	Build Time	Run Time	MIPS	Simulation
Functional	100K	0.54s	28.10s	-	-	0.74s	30.00s	-	-
TLM-2.0	100K	3.48s	3:46m	-	67M transactions	3.95s	4:00m	-	98M transactions
ISS	100	9.79s	2:28h	0.0457	406B instructions	15.81s	2:17h	0.0221	183M instructions
RTL	10	19:08m	2:26h	0.00167	14.6M instructions 52.4M clock cycles	42:12m	3:00h	0.00078	8.41M instructions 32.7M clock cycles
FPGA	100K	2:41h	1:23h	26.9	134B instructions 499B clock cycles 452K LUTs	Design exceeds #LUTs on VCU108 FPGA 762K LUTs required but only 537.6K available			

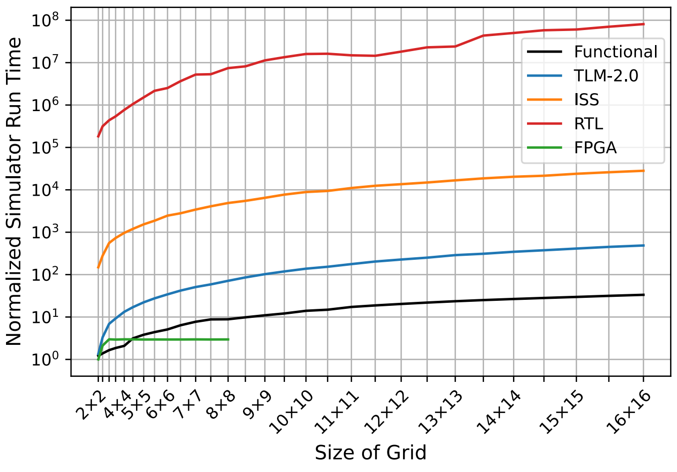

Fig. 4. Run time comparison of TokenX relative to the 2×2 functional model normalized to 100 steps across different grid sizes and abstraction levels

we report the utilization of 6-input look-up tables (LUTs) on the FPGA. As previously discussed and illustrated in Fig. 3, the LUT utilization increases linearly with the number of cores in the grid. This trend is reflected in the observed 77.7% increase in LUTs utilized, closely matching the 77.8% increase in core count when scaling from a 6×6 to an 8×8 grid.

3) Lessons Learned: For comparative performance evaluation of the five abstraction levels, we vary grid sizes from 2×2 to 16×16 as shown in Fig. 4. Limited by the most expensive abstraction level, namely RTL simulation, we restrict measurements to 100 steps. For easier comparison across abstraction levels, each measured run time is expressed as a factor relative to the run time of the 2×2 functional model.

Analyzing the run-time factors between adjacent abstraction levels reveals the following increases: one order of magnitude from the functional level to TLM-2.0, two from TLM-2.0 to ISS, and three from ISS to RTL simulation. This pattern aligns well with textbook expectations regarding simulation cost. Notably, the functional and TLM-2.0 models initially show almost identical run times on the 2×2 grid, but diverge significantly as the grid grows, primarily due to SystemC's restriction to single-threaded execution. All abstraction levels, except the FPGA, scale proportionally with grid size, resulting in nearly equidistant, parallel trends. From a grid size of 3×3 onward, the FPGA run time remains constant, at the cost of increasing LUT utilization. Starting at a grid size of 4×5, the FPGA implementation even outperforms the functional model. This is achieved by a fully parallel, hardware-based prototype of the GPC platform. The FPGA implementation is, however, limited to a maximum grid size of 8×8 by our FPGA. Overall, the design and tool flow demonstrate excellent scalability of both the simulated platform and the TokenX benchmark application.

C. Obstacles and Pitfalls with Real-World Designs

In contrast to the previous reference design flow using the TokenX application with regular behavior, we now study a different benchmark application on the same platform, requiring only minor adjustments to the configuration of each processing cell. ParticleSim is an actual Physics application characterized by an uneven workload that varies over time.

1) ParticleSim Benchmark: Our ParticleSim application simulates the physical movement of a user-defined number of particles in a confined two-dimensional space with configurable velocity, gravity, and attracting or repelling forces. For parallel execution, the space is partitioned into tiles, each assigned to a corresponding cell in the GPC. Each cell first computes the forces on the particles in its tile, taking into account the forces applied by the particles in neighboring tiles. The combined forces affect the direction and velocity of each particle and determine its movement to a new position. When particles cross tile boundaries, they are handed over to the neighboring cell.

ParticleSim carries a high and variable computational load requiring floating-point arithmetic. In our configuration, it starts with 1000 particles, which are evenly distributed over the tiles, resulting in an equal computational load for all cells at the beginning. We also apply an initial velocity in the South-East direction for all particles. As a result, the particles fall down and concentrate in the bottom right tiles, which gradually shifts the computational load to a few cells in the grid, until the particles repel each other and also bounce off the walls, eventually resulting in a somewhat equal distribution over the tiles for the remainder of the simulation.

Similar to TokenX, ParticleSim contains a main loop that repeats the force computation and particle movement for a user-defined number of steps. Again, a shorter simulation can be configured by reducing the number of steps, which shortens the execution length and allows us to keep simulator run times acceptable at lower abstraction levels. However, note that the steps in ParticleSim vary significantly in their computational complexity due to the shifting load of the moving particles, as we will see in the following.

2) Experimental Results: In line with the TokenX evaluation, we generate and run ParticleSim on 6×6 and 8×8 grids at each level, and report the results in Table III.

The build times down to RTL remain consistent with those of TokenX, with only slight increases due to the larger model complexity. At the FPGA level, the build time for the 6×6 grid increases significantly, by a factor of 2.5 compared to TokenX, mainly because of the additional hardware overhead from FPUs and larger memories, leading to an 87% increase in LUT utilization. As a result of the hardware overhead, the 8×8 grid cannot be built, as it exceeds the capacity of our FPGA, requiring 42% more LUTs than available.

Unlike TokenX, ParticleSim shows a run time increase across abstraction levels that deviates from the anticipated orders of magnitude slowdown observed in the previous study. Due to its high and variable computational load, a single step no longer has constant run time, requiring a more detailed analysis. Starting with the functional model, we reduce the number of steps by a factor of 10 (to 100 K) compared to TokenX to account for the increased computational load of ParticleSim. As before, we keep the number of steps constant for the TLM-2.0 model, observing the expected slowdown of up to one order of magnitude compared to the functional model. From TLM-2.0 to ISS, we expect a slowdown of two orders of magnitude and therefore reduce the number of steps by a factor of 100.

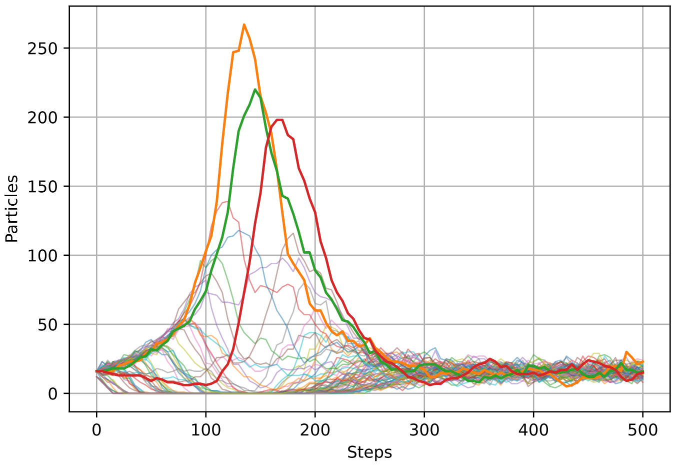

Fig. 5. ParticleSim TLM-2.0 on 8×8 GPC: Number of particles per cell

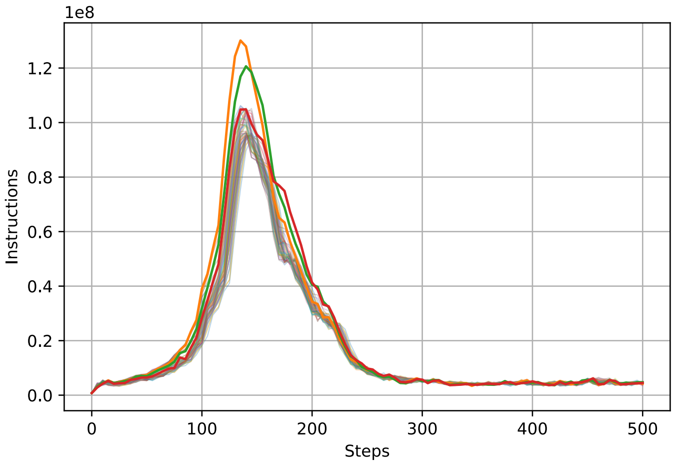

Fig. 6. ParticleSim ISS on 8×8 GPC: Number of instructions retired per cell

However, even with just 1 K steps at ISS level, the simulation does not complete within a day. Therefore, we further reduce the number of steps by a factor of 10, bringing it down to 100 steps. The simulation then completes after 2.5 hours. However, relative to the number of steps, we observe a total slowdown of more than four orders of magnitude from TLM-2.0 to ISS, which contradicts the expected slowdown of two orders that we experienced with the TokenX reference in Fig. 4. We note this mismatch as a first pitfall that requires further analysis.

Based on TokenX, we anticipate the RTL model to show a slowdown of three orders of magnitude compared to the ISS model. Reducing the number of steps accordingly would result in 0.1 steps, which is not feasible in the application. To run at least a few steps, we choose to execute 10 steps on the RTL model. Surprisingly, the RTL simulation completes in under 2.5 hours, less than the ISS model that executes 10 times more steps. Relative to the number of steps, this corresponds to a slowdown of only one order of magnitude from ISS to RTL, much less than anticipated and contrary to the slowdown of three orders observed with our TokenX experience. As this observation appears too good to be true, it becomes the second pitfall we encounter.

3) Analysis: To understand the cause of the pitfalls, we need to carefully analyze the computational load of the ParticleSim application over its execution length. This load varies not only across the cells as the particles move between tiles, but also from one step to the next.

To quickly account for the particles, we instrument the fast TLM-2.0 model on the 8×8 grid to accurately observe the number of particles in each tile at every step. Fig. 5 shows the particle count handled by each cell over 500 steps. The plot confirms our application knowledge: particles move across the grid and concentrate in a few cells at the bottom right before bouncing back again. A clear peak in particle concentration appears around step 140 in three specific cells (orange, green, and red lines). The highest load occurs at step 135, where the cell in row 7, column 5 contains 267 particles, more than a quarter of the total.

As such, the particle count in a cell becomes a critical factor for its computational load. At each step, the ParticleSim algorithm is quadratic in complexity (i.e. $\mathcal{O}(n^2)$ for n particles) as it computes the forces acting on each particle by considering all others within a specified proximity.

We can confirm this understanding using our ISS model, where the computational load of each cell is reflected in the number of instructions retired per step. Fig. 6 plots the executed instruction count for each cell over 500 steps. The peaks for the top three cells (orange, green, and red lines) around step 140 clearly correspond to the particle concentrations observed in Fig. 5. Interestingly, cells with few or no particles still reach a similar number of executed instructions. While this appears counterintuitive, it is easily explained by the fact that all cells must wait for slower neighbors to complete possible particle transfers. Due to our polling-based synchronization, the instructions associated with busy-waiting patterns accumulate accordingly.

Overall, we note that the computational load per step, as shown in Fig. 6, varies highly over the 500-step execution. Starting from the first step, the load increases rapidly, reaches its peak around step 140, and then declines just as quickly before settling into a steady state beyond step 300. At its peak, the instruction count is over 20 times higher than in the steady state. In simple terms, it takes roughly 20 times longer to compute steps at the peak than at the beginning or after step 300 of the particle simulation.

The curve in Fig. 6 explains the pitfalls we encountered. When running the functional and TLM-2.0 models with $100\,\mathrm{K}$ steps, the initial peak is only a very brief anomaly (only 0.2% of all steps), making it unnoticeable and seemingly negligible. However, this peak is what causes the pitfalls encountered with the much shorter execution length of the ISS and RTL models. First, when the ISS model executes $1\,\mathrm{K}$ steps, the computational peak dominates the run time, resulting in an unexpectedly long run time of over a day. Reducing the ISS run to just 100 steps still includes the steep ramp-up phase, explaining the observed slowdown by two additional orders of magnitude beyond anticipation.

Second, when running the RTL model for only the initial 10 steps, it captures a phase where the computational load remains well below the later average. This explains the surprisingly short RTL simulator run time, which initially seemed too good to be true. Essentially, the two-orders-of-magnitude discrepancy arises from a combination of the ISS run being disproportionally slow and the RTL simulation covering only a computationally light segment.

4) Lessons Learned: Our detailed analysis above not only explains the obstacles and pitfalls encountered but also serves as a good example for important insights in system design and simulation across multiple abstraction levels.

First, the system designer must choose a model at the right abstraction level for the specific purpose at hand. Here, examples are the choice of the TLM-2.0 model to quickly and accurately count the particles handled by the cells, and the choice of ISS to get an early yet accurate estimate of computational load per step.

Second, the one-two-three orders-of-magnitude slowdown from functional to TLM-2.0, to ISS, and to RTL, observed in Fig. 4, is a good quantitative rule of thumb. However, it only applies to GPC applications with low and evenly distributed computational load, such as our TokenX benchmark.

Third, to avoid pitfalls from naive expectations, the system designer must be aware of the application's behavior and anticipate anomalies caused by variable computational load. In particular, for simulator run-time estimation, it is crucial to choose the right slice of execution, such as best, average, or worst-case scenarios. An example would be a stress test of ParticleSim over the first 300 steps, which include the maximum load.

Fourth, FPGA prototyping is relatively expensive in platform cost and build time, but essential for running accurate and realistic workloads in real time. In our case, the entire workload of $100\,\mathrm{K}$ steps completes in under 1.5 hours.

V. Conclusion

System designers face obstacles and pitfalls posed by speed/accuracy tradeoffs across abstraction levels. Using a top-down open-source tool flow, we have studied two application benchmarks on a modern grid architecture of RISC-V processors over five abstraction levels, from the functional specification down to FPGA prototype implementation. Unfortunately, achieving real-time performance with complex designs on FPGA still requires commercial synthesis tools and evaluation boards. Across these levels, we have quantified the differences in simulation speed that amount to six orders of magnitude in total. The insights gained from our detailed study can be used by the system designer to better navigate the tradeoffs and efficiently reach a cycle-accurate implementation on fully constructed hardware.

Future work includes the study of additional software applications with other grid configurations. Additionally, we consider the support of FPGA-accelerated RTL simulation (FireSim), partitioning of larger grids over multiple FPGAs (FireAxe), and physical VLSI chip design (Hammer) based on the IHP SG13G2 open-source PDK.

Acknowledgments

This work is partially funded by the German Bundesministerium für Forschung, Technologie und Raumfahrt (BMFTR) through the Scale4Edge project (16ME0133). We gratefully acknowledge the computing time and resources provided on the Noctua 2 HPC Cluster at the NHR Center PC2, and the support from Heinrich Riebler.

References

[1] D. D. Gajski, J. Zhu, R. Dömer, A. Gerstlauer, and S. Zhao, *SpecC: Specification Language and Design Methodology*. Kluwer Academic Publishers, 2000.

[2] T. Grötker, S. Liao, G. Martin, and S. Swan, *System Design with SystemC*. Kluwer Academic Publishers, 2002.

[3] S. Sutherland, S. Davidmann, P. Flake, and P. Moorby, *System Verilog for Design: A Guide to Using System Verilog for Hardware Design and Modeling*. Norwell, MA, USA: Kluwer Academic Publishers, 2004.

[4] G. Schirner and R. Dömer, "Quantitative analysis of the speed/accuracy trade-off in transaction level modeling," *ACM Transactions on Embedded Computing Systems*, vol. 8, no. 1, pp. 4:1–4:29, Jan. 2009. [Online]. Available: http://doi.acm.org/10.1145/1457246.1457250

[5] E. M. Arasteh and R. Dömer, "Fast loosely-timed deep neural network models with accurate memory contention," *ACM Transactions on Embedded Computing Systems*, vol. 23, no. 5, Aug. 2024. [Online]. Available: https://doi.org/10.1145/3607548

[6] K. Fujiwara and H. Casanova, "Speed and accuracy of network simulation in the SimGrid framework," in *Proceedings of the 2nd International Conference on Performance Evaluation Methodologies and Tools*, ser. ValueTools '07. Brussels, BEL: ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2007.

[7] R. Dettmer, "Occam and the transputer," *Electronics and Power*, vol. 31, no. 4, pp. 283–287, 1985.

[8] R. Dömer, "A Grid of Processing Cells (GPC) with Local Memories," Center for Embedded and Cyber-physical Systems, UCI, Tech. Rep. CECS-TR-22-01, Apr. 2022.

[9] P. D. Schiavone, D. Rossi, A. Pullini, A. Di Mauro, F. Conti, and L. Benini, "Quentin: an ultra-low-power pulpissimo soc in 22nm fdx," in *2018 IEEE SOI-3D-Subthreshold Microelectronics Technology Unified Conference (S3S)*, 2018, pp. 1–3.

[10] A. Amid, D. Biancolin, A. Gonzalez, D. Grubb, S. Karandikar, H. Liew, A. Magyar, H. Mao, A. Ou, N. Pemberton, P. Rigge, C. Schmidt, J. Wright, J. Zhao, Y. S. Shao, K. Asanović, and B. Nikolić, "Chipyard: Integrated design, simulation, and implementation framework for custom socs," *IEEE Micro*, vol. 40, no. 4, pp. 10–21, 2020.

[11] M. Shalan and T. Edwards, "Building OpenLANE: A 130nm OpenROAD-based tapeout- proven flow : Invited paper," in *2020 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, 2020, pp. 1–6.

[12] M. J. Flynn, "Some computer organizations and their effectiveness," *IEEE Transactions on Computers*, vol. C-21, no. 9, pp. 948–960, 1972.

[13] A. J. G. Hey, "Supercomputing with transputers—past, present and future," in *Proceedings of the 4th International Conference on Supercomputing*, ser. ICS '90. New York, NY, USA: Association for Computing Machinery, 1990, p. 479–489.

[14] J. Backus, "Can programming be liberated from the von Neumann style? a functional style and its algebra of programs," *Commun. ACM*, vol. 21, no. 8, p. 613–641, Aug. 1978. [Online]. Available: https://doi.org/10.1145/359576.359579

[15] M. B. Taylor, J. S. Kim, J. E. Miller, F. Ghodrat, B. Greenwald, P. R. Johnson, W. Lee, A. Ma, N. R. Shnidman, D. Wentzlaff, M. I. Frank, S. P. Amarasinghe, and A. Agarwal, "The raw processor: A composeable 32-bit fabric for embedded and general purpose computing," 2001.

[16] T. Corporation, "The tile processor™ architecture: Embedded multicore for networking and digital multimedia," in *2007 IEEE Hot Chips 19 Symposium (HCS)*, 2007, pp. 1–12.

[17] L. Luchterhandt, T. Nellius, R. Beck, R. Dömer, P. Kneuper, W. Mueller, and B. Sadiye, "Towards a rocket chip based implementation of the RISC-V GPC architecture," in *MBMV 2023; 26th Workshop*, 2023, pp. 1–7.

[18] L. Luchterhandt, T. Nellius, R. Beck, R. Dömer, P. Kneuper, W. Mueller, and B. Sadiye, "Implementation of different communication structures for a rocket chip based RISC-V grid of processing cells," in *MBMV 2024; 27. Workshop*, 2024, pp. 79–89.

[19] N. Bombieri, F. Fummi, V. Guarnieri, F. Stefanni, and S. Vinco, "Efficient implementation and abstraction of systemc data types for fast simulation," in *Forum on Specification & Design Languages, FDL 2011*, 2011, pp. 1–7.

[20] M. Lora, S. Vinco, and F. Fummi, "Translation, abstraction and integration for effective smart system design," *IEEE Transactions on Computers*, vol. 68, no. 10, pp. 1525–1538, 2019.

[21] V. Herdt, D. Große, and R. Drechsler, "Fast and accurate performance evaluation for risc-v using virtual prototypes," in *2020 Design, Automation and Test in Europe Conference (DATE)*, 2020, pp. 618–621.

[22] V. Govindasamy and R. Dömer, "Mixed-level modeling and evaluation of a cache-less grid of processing cells," *ACM Transactions on Embedded Computing Systems*, Dec. 2024. [Online]. Available: https://doi.org/10.1145/3708988

[23] E. M. Arasteh, V. Govindasamy, and R. Dömer, "BusyMap, an efficient data structure to observe interconnect contention in SystemC TLM-2.0," in *Proceedings of the Design, Automation and Test in Europe (DATE) Conference*, 2024, pp. 1–6.

[24] "IEEE Standard for Standard SystemC Language Reference Manual," *IEEE Std 1666-2011 (Revision of IEEE Std 1666-2005)*, pp. 1–638, 2012.

[25] "IEEE Standard for SystemVerilog–Unified Hardware Design, Specification, and Verification Language," *IEEE Std 1800-2023 (Revision of IEEE Std 1800-2017)*, pp. 1–1354, 2024.

[26] L. Luchterhandt, V. Govindasamy, Y. Wang, R. Dömer, W. Mueller, and C. Scheytt, "Case study on combining open-source tool flows for grids of processing cells," in *Open Source Solutions for Massively Parallel Integrated Circuits (OSSMPIC) Workshop at the Design, Automation and Test in Europe (DATE) Conference*, Lyon, France, Apr. 2025, pp. 1–4.

[27] K. Asanović, R. Avizienis, J. Bachrach, S. Beamer, D. Biancolin, C. Celio, H. Cook, D. Dabbelt, J. Hauser, A. Izraelevitz, S. Karandikar, B. Keller, D. Kim, J. Koenig, Y. Lee, E. Love, M. Maas, A. Magyar, H. Mao, M. Moreto, A. Ou, D. A. Patterson, B. Richards, C. Schmidt, S. Twigg, H. Vo, and A. Waterman, "The rocket chip generator," EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2016-17, Apr 2016. [Online]. Available: http://www2.eecs.berkeley.edu/Pubs/TechRpts/2016/EECS-2016-17.html

[28] J. Bachrach, H. Vo, B. Richards, Y. Lee, A. Waterman, R. Avižienis, J. Wawrzynek, and K. Asanović, "Chisel: Constructing hardware in a scala embedded language," in *DAC Design Automation Conference 2012*, 2012, pp. 1212–1221.

[29] A. Izraelevitz, J. Koenig, P. Li, R. Lin, A. Wang, A. Magyar, D. Kim, C. Schmidt, C. Markley, J. Lawson, and J. Bachrach, "Reusability is FIRRTL ground: Hardware construction languages, compiler frameworks, and transformations," in *2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, Nov 2017, pp. 209–216.

[30] W. Snyder, P. Wasson, D. Galbi *et al.*, *Verilator*. [Online]. Available: https://verilator.org

[31] R. Dömer, "Seven obstacles in the way of standard-compliant parallel SystemC simulation," *IEEE Embedded System Letters*, vol. 8, no. 4, pp. 81–84, Dec. 2016.

[32] S. Vinco, D. Chatterjee, V. Bertacco, and F. Fummi, "Saga: SystemC acceleration on GPU architectures," in *Proceedings of the 49th Annual Design Automation Conference*, ser. DAC '12. New York, NY, USA: Association for Computing Machinery, 2012, p. 115–120. [Online]. Available: https://doi.org/10.1145/2228360.2228382