Appears in the Proceedings of the Workshop on Reproducible Research Methodologies (REPRODUCE), February 2014

Hardware Generation Languages as a Foundation for Credible, Reproducible, and Productive Research Methodologies

Derek Lockhart and Christopher Batten
School of Electrical and Computer Engineering, Cornell University, Ithaca, NY
{dml257,cbatten}@cornell.edu

Cycle-approximate simulators (CAS) have long been a staple in the experimental toolkit of computer architecture researchers. However, recent technology trends have triggered a growing emphasis on specialization and heterogeneity, bringing into question the viability of a purely CAS-based methodology. In this position paper, we argue computer architects should consider adopting an increased focus on synthesizable hardware description languages (HDL) to improve credibility and reproducibility. Additionally, we discuss the need for hardware generation languages and integrated frameworks to improve researcher productivity and close the gap between current CAS and HDL methodologies.

1. A Tale of Two Research Methodologies

Modern computer architecture research has increasingly relied on the use of *cycle-approximate simulators* (CAS) as the primary tool for evaluating novel architectural mechanisms. We characterize a CAS methodology as the use of a computer architecture simulation framework implemented in C or C++ (e.g., GEM5 [3], SimpleScalar [1]), configured and/or modified to model a particular system architecture and any enhancements. Models built using a CAS methodology are capable of generating fairly accurate estimates of system performance (in terms of cycles executed) provided that the simulated models properly implement a sufficient level of architectural detail [4]. Additional tools like McPAT [8] can augment these frameworks to generate preliminary estimates for other key metrics such as area, energy, and timing. The architecture community has generally considered the benefits (e.g., flexibility, simulation speed) of CAS frameworks to outweigh the drawbacks (e.g., accuracy, reproducibility), making them a popular choice among academic research groups.

An alternative methodology for computer architecture research relies on the construction of synthesizable register-transfer-level implementations using a *hardware description language* (HDL) such as SystemVerilog, Verilog, or VHDL. Paired with a commercial ASIC EDA toolflow, a synthesizable implementation can generate highly accurate estimates of area, energy, cycle time, and cycles executed. Unlike the approximate models of hardware behavior created using a CAS methodology, an HDL methodology constructs a true hardware implementation which provides a number of advantages beyond just improved accuracy. An HDL methodology increases credibility with industry, enforces a model/simulator separation which enhances reproducibility, provides interoperability with a range of industry-grade tools, and enables a path to FPGA or ASIC prototypes. Although widely used by industry and VLSI research groups, the HDL methodology is less popular in computer architecture research due to long HDL development times and slow HDL simulation speeds.

Attempts to leverage both CAS and HDL methodologies can be a challenging prospect due to the considerable gap between the tools and techniques used in each domain. For academic research groups lacking the resources and manpower of an industrial research lab, bridging this gap can be exceedingly difficult.

2. Limitations of Existing Methodologies in an Era of Specialization

When exploring specialization and its impact on system performance, it becomes increasingly difficult to perform credible experiments using a purely CAS methodology. A CAS methodology relies on existing implementations of processors, memories, and networks to act as targets from which models implemented in a CAS methodology are validated. Because specialization creates unique hardware blocks tailored to accelerating workload-specific operations, researchers exploring specialization generally have no hardware targets from which to validate their CAS models. Several studies have shown that without validation, performance and power estimates from CAS models become difficult to trust [5, 6].

An HDL methodology can be leveraged by architects to create register-transfer-level (RTL) implementations of novel specialized blocks, helping address some of the problems encountered when experimenting with architectures incorporating large amounts of specialization. Unfortunately, the limited productivity of an HDL methodology makes it difficult to quickly create hardware instances of many different specialized units. In addition, a purely HDL methodology prevents researchers from mixing blocks at various levels of abstraction so that detailed models of specialized units can be composed with cycle-approximate, analytical, or functional implementations of other components in the system.

Going forward, we believe two major improvements are needed to close the gap between CAS and HDL to enable compelling new research in hardware specialization. The first is better tools for productive RTL implementation, existing HDLs are simply not capable of the extensive parameterization necessary for rapid design space exploration of novel hardware specializations. The second is greater integration between the CAS and HDL methodologies.

3. Hardware Generation Languages (HGLs)

Previous work has proposed replacing the current design practice of constructing specific *chip instances* with an approach focused on construction of *chip generators*, a technique that would allow for templated RTL implementations, rapid design-space exploration, and codification of designer knowledge [10]. Development of modern, highly-parameterizable hardware development languages, which we term *hardware generation languages* (HGLs), are a key component to enabling chip generator construction and productive RTL design in general. A number of projects already exist which could be considered HGLs, including Genesis2 [11], Chisel [2], and Bluespec [9]. We argue that future computer architecture methodologies should have a strong foundation built on HGLs in order to enable credible, reproducible, and productive research.

HGL Credibility – Credibility comes from commercially-vetted processes built around industry-standard HDLs such as SystemVerilog, Verilog, and VHDL. Nearly all modern ASIC flows are built around one or more of these languages and an extensive set of commercial tools exist to simulate, verify, and synthesize hardware specified in these HDLs. Since HGLs generally provide a translation path to an industry-standard HDL they enable access to the high-fidelity analysis generated by commercial EDA tools. This is in contrast to a CAS methodology where there are no industry standards and few attempts at detailed performance, area, energy, and timing validation against real hardware implementations.

HGL Reproducibility – HGLs usually provide standardized, concise hardware specifications with well-defined semantics. The HGL translation path to industry-standard HDLs ensures compatibility with a wide range of of toolflows; researchers can independently verify reported results by feeding HDL source into their own tools. In contrast, a CAS methodology presents numerous challenges to reproducibility due to the large number of independent simulation frameworks, each with their own software architecture and modeling strategy. The vast majority of computer architecture simulators are written in a sequential functional fashion, as opposed to the concurrent structural composition of hardware, making it difficult to reason about the hardware behavior they are modeling [13]. Even when provided with the source code of a CAS model reproducing prior work in a different simulation infrastructure can be quite challenging.

HGL Productivity – In addition to enabling the creation of hardware templates for design space exloration, HGLs can improve designer productivity by introducing higher-level language abstractions. Techniques such as polymorphism, parameterizable types, bitwidth inference, bundles, and programmable structural composition help facilitate succinct and expressive RTL implementations. HGLs can also help address the low performance of RTL simulation by enabling alternative simulation techniques. For example, Chisel has the ability to generate fast C++ simulators which demonstrate 10x speedups over commercial tools for long running simulations. [2]

4. Integrated CAS/HGL Methodologies

While HGLs can greatly improve the productivity of HDL methodologies via construction of chip generators, the detailed nature of an RTL design is unlikely to ever surpass the productivity (or simulation speed) of high-level CAS modeling. We argue that future computer architecture methodologies should provide tight integration of both HGL and CAS methodologies to create a unified framework that is credible, reproducible, and productive. We identify three incremental steps to such tight integration of HGL and CAS methodologies.

Integrating HGLs into Widely Adopted CAS Frameworks – Perhaps the most straight-forward approach is to translate HGL design instances into industry standard HDLs and then to compile these HDL design instances into a widely adopted CAS framework. For example, one could imagine using Chisel to implement a specialized block, generating the corresponding Verilog RTL, using a tool such as Verilator [14] to translate the Verilog RTL into a cycle-accurate C++ model, and then linking this model into the GEM5 simulation framework. While this is a promising first-step, most widely adopted CAS frameworks were not designed with HGL integration in mind. This can potentially limit the granularity of integration. For example, while it might be possibly to integrate blocks designed with an HGL methodology into the memory system of GEM5, it would be more challenging to create new specialized functional units or accelerators that are tightly coupled to the main processor pipeline.

Integrating HGLs into New CAS Frameworks – A more radical approach, would be to develop a new CAS framework from scratch specifically designed to facilitate tight integration with HGLs. Such a CAS framework will likely need to avoid performance optimizations such as split functional/timing models and use some form of concurrent structural modeling [13]. A strong example of such an approach is Cascade, a C++ framework used in the development of the Anton 2 supercomputer and specifically designed to enable rapid design-space exploration using a CAS methodology, yet also enable tight Verilog integration [7]. Cascade includes support for interfacing binding, enabling composition of Verilog and C++ modules without the need for specialized data-marshalling functions.

Creating a Unified CAS/HGL Framework – Perhaps the most extreme approach would be a completely unified framework that enables both CAS and HGL methodologies in a single high-level language. SystemC was originally envisioned to be such a framework, although it is mostly used in practice for cycle-approximate and even more abstract transaction-level modeling to create virtual system platforms for early software development [12]. In our own research group, we are developing a unified, Python-based framework to facilitate a tightly integrated CAS/HGL methodology. Leveraging a concurrent-structural modeling approach, it naturally supports incremental refinement from high-level algorithms to cycle-approximate model to cycle-accurate implementation. An embedded DSL allows Python to be used as a hardware generation language for construction of highly-parameterized hardware designs. These parameterized designs can then be translated into synthesizable Verilog instances for use with commercial toolflows. Due to the unified nature of the framework, high-level cycle-approximate models can naturally be simulated alongside detailed RTL implementations, allowing users to finely control speed and accuracy tradeoffs on a module by module basis. Existing Verilog IP can be incorporated for cosimulation using a Verilator-based translation toolchain, enabling those with significant Verilog experience to leverage Python as a productive verification language.

Acknowledgements

This work was supported in part by NSF Awards CCF-1149464 and CNS-1059233, DARPA YFA N66001-12-1-4239, and donations from Intel Corporation, NVIDIA Corporation, and Synopsys, Inc.

References

[1] T. Austin et al. SimpleScalar: An Infrastructure for Computer System Modeling. *IEEE Computer*, 35(2):59–67, Feb 2002.

[2] J. Bachrach et al. Chisel: Constructing Hardware in a Scala Embedded Language. *Design Automation Conf. (DAC)*, Jun 2012.

[3] N. Binkert et al. The GEM5 Simulator. *SIGARCH Computer Architecture News*, 39(2):1–7, Aug 2011.

[4] A. Butko et al. Accuracy Evaluation of GEM5 Simulator System. *Workshop on Reconfigurable Communication-Centric Systems-on-Chip (ReCoSoC)*, Jul 2012.

[5] J. Gibson et al. FLASH vs. (Simulated) FLASH: Closing the Simulation Loop. *Int'l Conf. on Arch Support for Prog Languages and Operating Systems (ASPLOS)*, Dec 2000.

[6] M. Govindan et al. End-to-End Validation of Architectural Power Models. *Int'l Symp. on Low-Power Electronics and Design (ISLPED)*, Aug 2009.

[7] J. P. Grossman et al. The Role of Cascade, a Cycle-Based Simulation Infrastructure, in Designing the Anton Special-Purpose Supercomputers. *Design Automation Conf. (DAC)*, Jun 2013.

[8] S. Li et al. The McPAT Framework for Multicore and Manycore Architectures: Simultaneously Modeling Power, Area, and Timing. *ACM Trans. on Arch and Code Opt (TACO)*, 10(1), Apr 2013.

[9] N. Nikhil. Bluespec System Verilog: Efficient, Correct RTL from High-Level Specifications. *Int'l Conf. on Formal Methods and Models for Co-Design (MEMOCODE)*, Jun 2004.

[10] O. Shacham et al. Rethinking Digital Design: Why Design Must Change. *IEEE Micro*, Nov/Dec 2010.

[11] O. Shacham et al. Avoiding Game Over: Bringing Design to the Next Level. *Design Automation Conf. (DAC)*, Jun 2012.

[12] SystemC TLM (Transaction-level Modeling). Online Webpage, 2013 (accessed July, 2013). `http://www.accellera.org/downloads/standards/systemc/tlm`.

[13] M. Vachharajani et al. The Liberty Simulation Environment: A Deliberate Approach to High-Level System Modeling. *ACM Trans. on Computer Systems (TOCS)*, 24(3):211–249, Aug 2006.

[14] Verilator. Online Webpage, 2013 (accessed Aug 7, 2013). `http://www.veripool.org/wiki/verilator`.