Codesign for the Masses

SAND2021-1798R

Cannada Lewis (canlewi@sandia.gov)[*], Simon Hammond, and Jeremiah Wilke

February 16, 2021

1 Topic

(**Architecture & Codesign Methologies**) In this position paper we will address challenges and opportunities relating to the design and codesign of application specific circuits. Given our background as computational scientists, our perspective is from the viewpoint of a highly motivated application developer as opposed to career computer architects.

2 Challenge

It is no secret that major breakthroughs in computational sciences are often reexaminations of previously proposed ideas, in the context of either new algorithmic techniques or new developments in computational capability. In the last 15 years adoption of GPUs and their programming models [9] has given new life to old ideas in a plethora of disciplines from quantum chemistry [4, 13] to—most famously—deep learning [12]. These are examples of approaches that have won *the hardware lottery*[6], where technological developments have favored certain application and algorithmic choices at the expense of others; often irrespective of their theoretical merits. What if we could partially solve the issue of *the hardware lottery*? Because it is not possible to fully decouple modern computational sciences from the specifics of hardware design, what if we could give the application developers better tools to design and understand hardware, instead of forcing them to pick between a few mostly general purpose designs? This is the very essence of codesign.

The current paradigm forces applications to optimize for either a specific architecture or a narrow band of similar architectures, none of which were specifically tailored to the exact problems the application faces. As extreme heterogeneity becomes the norm, between CPUs, GPUs, FPGAs, reconfigurable hardware and combinations of all of them, ideally application developers would have an easy way to design and iterate on their own accelerator designs. This goal presents great challenges though; while modern approaches provide significant quality of life improvements for hardware designers, creating an application specific accelerator is still very much the domain of hardware experts. For example, it is a great achievement that open source CPU cores require only thousands of lines of code[15], albeit in unfamiliar (to application developers) hardware description languages (HDL), such as Verilog, VHDL, pyRTL, or Chisel. But, their design at the HDL level is still out of reach for all but the most dedicated application programmers.

The main alternative to HDL is high level synthesis (HLS) of programming languages like C and C++. HLS of C++ can require less effort than a similar HDL design [11], but it is not a panacea; much like high-performance linear algebra routines, the code required for performant HLS rarely resembles the initial high level implementation and demands the developer have a deep understanding of how the code will be synthesized. All is not lost though, reference [11] suggests that there are quality of life and time to solution improvements available via HLS.

Finally, there is the issue of hardware simulation, the efficient and accurate modelling of general purpose accelerators is a non-trivial task. Many tools exist in this space, but they can be vendor dependent or difficult to use for novices. Work must be done to simplify the design, simulate, and iterate cycle if we want to empower application developers to embrace accelerator design. Other position papers will most likely address this topic in more depth.

The challenges to bring hardware design to the masses (application developers) are great. Computational scientists do not have HDL experience, hardware simulation and the use of FPGAs is unfamiliar, and they may be sceptical of the expected value their applications would ultimately receive. But, there also exists a great opportunity to achieve success that could only come via codesign of algorithms and hardware. Ultimately to truly enable codesign, we require new approaches via software engineering, programing models, and compilers to bridge the gap between software and hardware engineering.

3 Opportunity

The end of Dennard Scaling and the inevitable demise of Moore's law dictate that future performance gains for scientific applications are likely to come from a combination of software engineering, algorithmic improvements, and hardware specialization [8]. In many cases these are multiplicative, which itself

[*]Sandia National Laboratories is a multimission laboratory managed and operated by National Technology and Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

speaks to the value of codesign. The oppportunity for DOE is to lead the creation of a suite of tools that bring these kind of improvements to the broadest possible user base—not just the determined few which has arguably been the situation for some large-scale HPC architectures. The result could be vast increases in scientific delivery and broad families of codesigned accelerators or SoCs for entire computing communities. The piece that is missing to enable this future is the tooling to give developers the ability to explore the space of interactions between algorithms and hardware. One need look no farther than deep learning to see the transformative effects that having access to domain specific hardware can achieve.

By providing the tools required to enable application developers to steer the process of codesign themselves, DOE will be able to initiate a new era of computational science that mirrors the explosion of successes in deep learning. Performance improvements of more than 30X are not out of reach [7], but achieveing this will require deeply coupled codesign of the hardware and software. No one knows better than the domain scientists where the opportunities for codesign lie, but to date, they have not had the technology to make hardware exploitation feasible. As we progress into a post Moore's era this needs to change.

4 Timeliness

We have precedent that application developers are willing to explore the interplay of algorithms and hardware, as evidenced by the rise of parallel programming models such as CUDA [9], SYCL [1] and Kokkos [5]. Adopting these models for parallel programing requires significant investment from applications, but in the end the performance gains are often worth it. Before these libraries and languages existed, it was simply too difficult for the vast majority of application developers to attempt to target GPUs, limiting GPU adoption. Currently, software hardware codesign is in a similar state, but the tools are beginning to become available to allow scientist to explore the hardware landscape. Similarly, to how CUDA helped to bring GPU programming to the masses new compiler tools and domain specific languages for HLS from both academia and industry [2, 3, 14, 10] will help empower application developers to explore domain specific architectures for their problems. But this work is still in its infancy and DOE should invest now, both to help develop the tools and abstractions needed to make high level codesign a reality and also to connect application developers with these tools. In some sense we have the opportunity to codesign the future generation of codesign tools.

References

[1] `https://www.khronos.org/sycl/`.

[2] `https://github.com/google/xls/`.

[3] `https://github.com/llvm/circt`.

[4] M. Dupuis, J. Rys, and H. F. King, *Evaluation of molecular integrals over gaussian basis functions*, The Journal of Chemical Physics, 65 (1976), pp. 111–116.

[5] H. C. Edwards, C. R. Trott, and D. Sunderland, *Kokkos: Enabling manycore performance portability through polymorphic memory access patterns*, Journal of Parallel and Distributed Computing, 74 (2014), pp. 3202 – 3216. Domain-Specific Languages and High-Level Frameworks for High-Performance Computing.

[6] S. Hooker, *The hardware lottery*, 2020.

[7] N. P. Jouppi et al., *In-datacenter performance analysis of a tensor processing unit*, 2017.

[8] C. E. Leiserson, N. C. Thompson, J. S. Emer, B. C. Kuszmaul, B. W. Lampson, D. Sanchez, and T. B. Schardl, *There's plenty of room at the top: What will drive computer performance after moore's law?*, Science, 368 (2020).

[9] J. Nickolls, I. Buck, M. Garland, and K. Skadron, *Scalable parallel programming with cuda: Is cuda the parallel programming model that application developers have been waiting for?*, Queue, 6 (2008), p. 40–53.

[10] R. Nigam et al., *Predictable accelerator design with time-sensitive affine types*, in Proceedings of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2020, New York, NY, USA, 2020, Association for Computing Machinery, p. 393–407.

[11] J. Noguera, S. Neuendorffer, K. Vissers, and C. Dick, *Wireless mimo sphere detector implemented in fpga*, Xcell Journal, 74 (2011), pp. 38–45.

[12] R. Raina, A. Madhavan, and A. Y. Ng, *Large-scale deep unsupervised learning using graphics processors*, in Proceedings of the 26th Annual International Conference on Machine Learning, ICML '09, New York, NY, USA, 2009, Association for Computing Machinery, p. 873–880.

[13] I. S. Ufimtsev and T. J. Martínez, *Quantum chemistry on graphical processing units. 1. strategies for two-electron integral evaluation*, Journal of Chemical Theory and Computation, 4 (2008), pp. 222–231. PMID: 26620654.

[14] J. Wang, L. Guo, and J. Cong, *Autosa: A polyhedral compiler for high-performance systolic arrays on fpga*, in Proceedings of the 2021 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, 2021.

[15] J. Zhao, B. Korpan, A. Gonzalez, and K. Asanovic, *Sonicboom: The 3rd generation berkeley out-of-order machine*, (2020).