Why use FPGAs for Computing?

By
Steve Casselman, CEO
HotWright Inc.

Abstract:

FPGAs have been around for over 35 years, and they have been used for computing from the beginning. [1] [2] This paper outlines the basic, fundamental, reasons why FPGA technology is better than any other technology for computing at all scales. FPGA performance on many different algorithms has ranged from 0.1x to 1,000x compared to top-of-the-line processors. At first, FPGAs could only beat processors on integer-based algorithms. The classic four algorithms that show FPGAs are interesting compute engines are sort, search, encrypt and compress. FPGAs started to beat CPUs at single-precision floating-point algorithms in the 2001 time frame and double precision in 2003. [3] This was all without hardwired floating-point support. There are major reasons to use FPGA technology. These include device physics and intrinsic usability that make FPGAs the right path to take into the future of computing.

This paper will first talk about the physical reasons why FPGAs are better than ALU based CPUs and GPUs, next we'll talk about the intrinsic benefits FPGAs provide, then we will talk about the problems FPGAs currently have, and finally, we'll discuss why, in the near future, FPGAs will overcome these problems to become a first-class citizen in the computing world.

Physical advantages:

Dark silicon:

Dark silicon is the phenomenon of starving the silicon of electrons because the power usage in an area is too high.
From the group that coined the term "dark silicon":

"Our team was among the first to demonstrate the existence of a utilization wall which says that with the progression of Moore's Law, the percentage of a chip that we can actively use within a chip's power budget is dropping *exponentially*! The remaining silicon that must be left unpowered is now referred to as Dark Silicon." [4] [5]

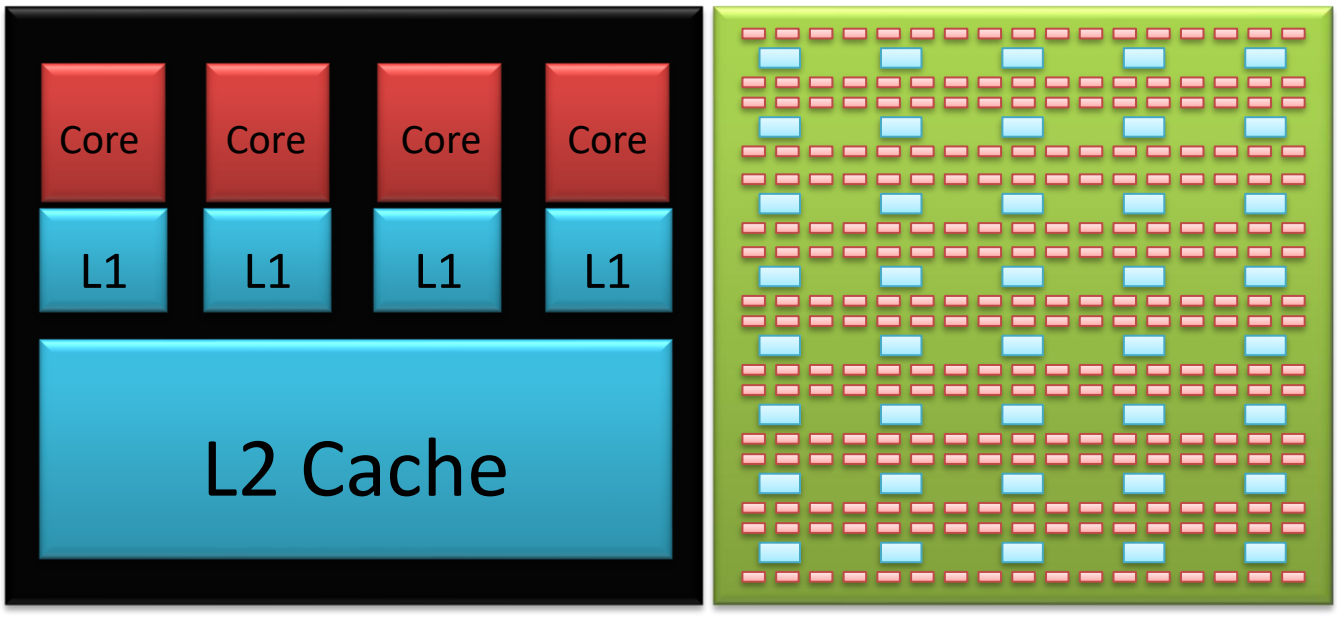


Figure 1 on the left, high speed cores get very hot. On the right, compute power is spread out and performance comes from pipelining. The logic is in pink and memory in blue.

FPGAs do not have "dark silicon" like multicore processors. This is because while multicore designers pushed the frequency so high there are not enough electrons to power the high-density cores. FPGA designers spread the computing power over the entire chip.

Multicores also have caches and network on chip structures which can take lots of power. FPGAs have direct pipelined connections and take less power to move the data than multicores. Think of dark silicon as an extension of the frequency wall. If a device is 1 square centimeter and takes 1 watt, then, when you shrink the geometry by half, you have one watt in ¼ square centimeter. That is a 4x power per area increase. This is a fundamental barrier to shrinking traditional compute architectures like von Neumann cores whether they are multi-core, many-core, or SIMD.

Rent's Rule:

Rent's rule describes the relationship between the amount of logic in a partition and the amount of communication into that partition to ensure the logic can be used. FPGAs are architected based on Rent's rule and CPUs and GPUs are not. FPGAs are designed to get the data to the logic and so have way more routing and bisectional bandwidth than multicore systems such as CPUs and GPUs. You can see how this works in figure 2. The logic cores of CPUs and GPUs are connected to caches through which the data must pass. FPGAs, on the other hand, have 1000's of wires coming into a logic partition from all directions. In FPGAs data is managed through 100's to 1000's of multi-ported memories instead of a hierarchical memory using different levels of cache.

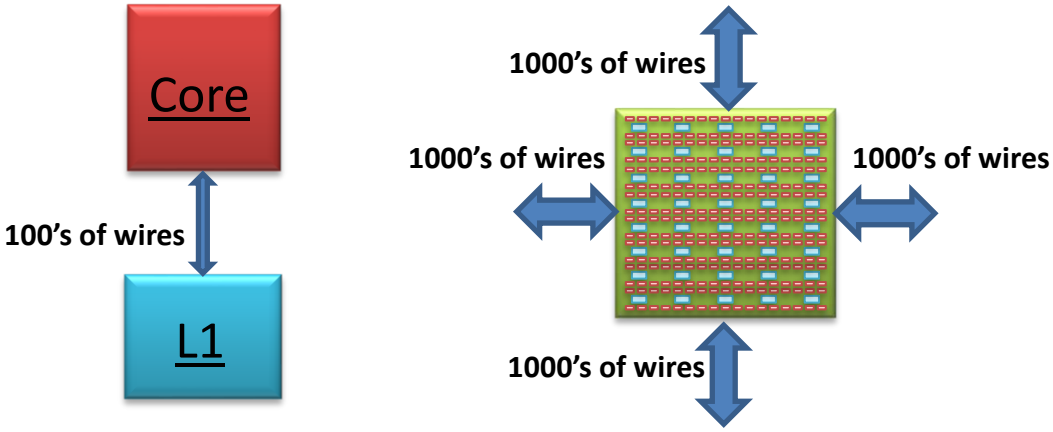


Figure 2 Rent's rule more wires go to the logic in FPGAs than CPUs

Because FPGAs have many more wires than CPUs or GPUs the overall bisectional bandwidth of FPGAs is impressive. If you make the partition the whole chip you see that FPGAs have multiple PCIe busses as well as 400 Gbit transceiver capabilities, something that CPUs and GPUs don't have.

Other fundamental observations:

Central Processing Unit (CPU) based computing devices:
CPUs have very high speed and power-hungry compute units. They use a register file and an ALU to implement their data flow graphs. A high portion of the device is dedicated to retrieving and decoding instructions thereby implementing their control flow graph. This subsystem must move billions of instructions per second generating lots of heat. CPUs have hierarchical levels of memory which data must travel over. Some of these caches are shared among the cores and this causes memory contention.

Graphical Processing Unit (GPU) based computing devices:
GPUs have many small, fast compute units tied to a single instruction sequencing control unit. GPUs also have an instruction decode subsystem that moves billions of instructions per second generating lots of heat. GPUs are mostly floating-point intensive devices that are best at data parallel algorithms that do the same thing for each data point. Branching algorithms don't perform well in these devices. GPUs have hundreds of cores running at high speeds and are the most power-hungry devices used in computing today.

Field Programmable Gate Array (FPGA) based computing devices:
FPGAs have uncommitted logic and routing that gets "personalized" at run time. Since the logic and internal memory are spread out, there are always enough electrons to power the device. FPGAs don't suffer from the dark silicon bottleneck like CPUs and GPUs. FPGAs put the burden of instruction generation on the compiler and don't have a power-hungry instruction decode subsystem that uses lots of power. FPGAs are the most power-efficient compute devices available.

Intrinsic benefits:

Result Reuse:

Memory subsystems evolved from the single core architectures that produced both single threaded programs and programming languages. When CPUs hit the frequency wall, which ushered in the multicore age, memory subsystems tried to patch up performance with megabyte caches and more memory channels. This problem is exacerbated by the programmers trying to "optimize" programs by making them smaller and tighter. Functions of the form **result = f(x,y,z…)** dominate program libraries which means for each function the system takes data out of the memory, computes on it, then stores the result back in memory. Statements that are composites of functions still must take data from memory, compute on it, and then send it back to memory. If the data sets are small, the caches work fine. But if the data sets are larger, then there can be a huge degradation of performance. A composite function might look like this: **result0 = f(g(x,y),h(y,z))** which is broken down into this: **result1 = g(x,y)**, **result2 = h(y,z)**, then **result0 = f(result1,result2)**, where each result has to go back to memory, and data y must be read twice. Trying to parallelize this to make use of multiple cores just makes matters worse, or not much better, by creating memory contention where the cores must fight over access to the same sets of data.

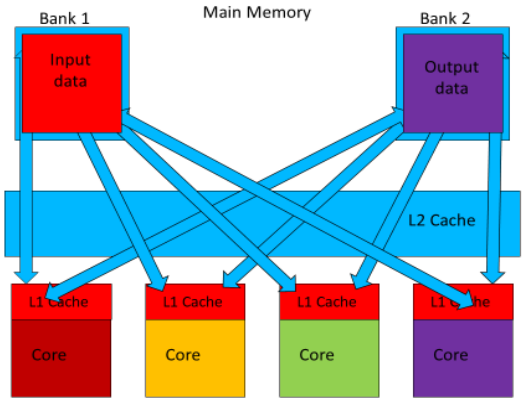


Figure 3 Memory collusions slow down multicore solutions

"First defined two decades ago, the memory wall remains a fundamental limitation to system performance. Recent innovations in 3D-stacking technology enable DRAM devices with much higher bandwidths than traditional DIMMs. The first such products will soon hit the market, and some of the publicity claims that they will break through the memory wall. Here we summarize our analysis and expectations of how such 3D-stacked DRAMs will affect the memory wall for a set of representative HPC applications. We conclude that although 3D-stacked DRAM is a major technological innovation, it cannot eliminate the memory wall." [6] [7]

FPGAs allow you to build hardware that can explicitly compose functions in a way that avoids excess memory reads and writes as seen in figure 4.

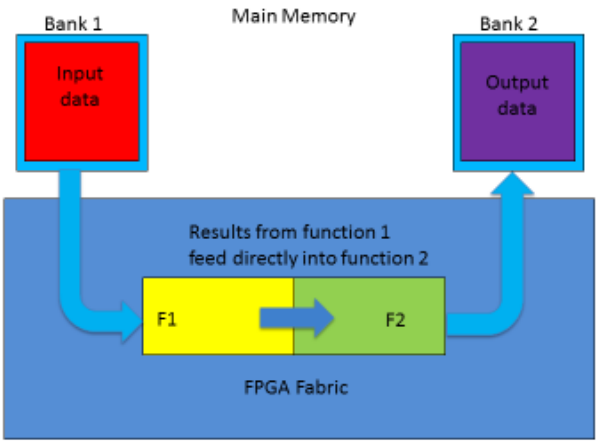


Figure 4 FPGAs eliminate collisions by using results from the previous function call.

Circuit Specialization:

FPGAs can be specialized even more than application specific integrated circuits (ASICs). This is because ASICs are inflexible once they are produced. Functionality is frozen. For example, DNA sequencing of amino-acid base pairs compare a sample strand against one or more databases of DNA strands to identify it or see how close it comes to a known sequence. You can get a "how close am I?" score. The highest score is the best-matched sequence.

There is a well-known algorithm to do this called the Smith-Waterman (SW) algorithm. SW is so compute-intensive that another algorithm, less accurate but faster, is used to compute the closest match. You can implement this in hardware. and there are several SW ASICs available. The SW uses a systolic architecture, so there are many little "processing elements", or PEs, tied together to their nearest neighbors. This algorithm's communication is between the PEs, so the wires are local, short, and fast. The ASIC version, which must load coefficients, runtime parameters, and do other activities that have nothing to do with the algorithm, requires many wires that are global, longer, and slower. This is the algorithm's hardware overhead which is not needed in the FPGA.

With FPGAs, you can build a custom sequencer for each DNA sequence. The resulting machine is smaller than the ASIC since each PE must look only for one DNA symbol and so the PE can be customized for performance. The hardware does not need to load the sequence which eliminates all hardware overhead. You can now fit more PEs into the FPGA and each PE runs faster. JBits can build and load the circuit in milliseconds as was shown by the Xilinx JBits project. [8] One FPGA was faster than 144 ASICs by 12x. [9]

This same technique can be used on neural networks and other algorithms.
The Xilinx FINN open-source project [10] takes a trained neural network and quantizes it. This allows FINN to trim the hardware design which then gets implemented. This saves power, and space, and improves performance in the FPGA.

Raw performance:

The industry consensus is that CPUs can't move into the future alone. Currently, in 2022, the computer hardware universe is fracturing. Intel and AMD are developing devices with both CPUs and GPUs in them. New compute devices like Google's TPU, Groq's GroqChip, SambaNova's RDU, Fungibles' DPU, and GraphCore's wafer-scale device are coming out to fill a gap in AI and ML.

The driving measure of raw performance today is the performance of AI workloads. AI is becoming so important that a growing percentage of the worldwide computing power is dedicated to these applications. [11]

FPGAs are better for AI inference than GPUs and CPUs. [12] [13]

FPGAs contain enormous amounts of logic, distributed memory, hardwired mathematics, and high-speed I/O.

FPGAs win the computing wars!

Why don't FPGAs own the computing world?

If FPGAs win on computing, then why aren't they everywhere?

There are systemic problems standing in the way before FPGAs can take over computing as we know it.

Systemic problems:

1. Device secrecy
2. The broken programming model
3. System architecture

Device secrecy:

Over the past 35 years, FPGA manufacturers have been extremely secretive about the programming of their technology. This crippled a developer's ability to implement runtime generation of bitstreams. [14] This is like having a computer where you can only assign memory at compile time. It takes away the ability to do dynamic allocation and generation of hardware and manage these dynamic resources at runtime.

Security through obscurity has been a way of life for FPGA manufacturers. Rules against reverse engineering the bitstream stopped any advancement or research in the area of runtime bitstream generation. There was a brief instant in time when Xilinx had a part with a fully documented bitstream, the XC6200. This openness inspired many FPGA advances such as Evolvable Hardware [15] and JBits [9].

The broken programming model:

Every algorithm can be decomposed into a data flow graph (DFG) and a control flow graph (CFG). When executed together the DFG and CFG implement the algorithm.

CPUs have evolved to implement the DFG and CFG using the ALU and register file, with other hardware, to manage the low-level mapping details. High level languages (HLLs) have evolved to match the ALU/regfile architecture. C/C++ and other current HLLs are structured to implement the DFG and CFG through the sequential execution of "instructions".

FPGA have evolved by implementing the DFG and CFG as hardware. They were first programmed via a schematic. The same "language" is used to create ASICs and PCBs. The next level of abstraction for FPGAs was hardware description languages (HDLs). Languages like Verilog and VHDL take years to learn and are very different from HLLs. The HDL tools are used to design ASICs and provide low-level flexibility of the design implementation of the DFG and CFG.

HDLs allow you to implement the DFG and CFG separately. You can parallelize and optimize both the DFG and CFG for performance or power or other reasons. These design decisions are already made for you in CPUs. Because HLLs and modern CPU architectures evolved together, HLLs map to the CPU hardware organically.

Programming an FPGA with an HLL is accepting the fact that the programming system was meant for a fixed implementation of control and data computation circuity. The one thing that these mappings seem to be able to do right, is pipelining the data graph to the point where the control graph looks like a go or no-go CFG.

Another broken part of programming FPGAs is that all current software packages[1] take hours to place and route a design/program. This breaks the quick design/test cycle that makes programming CPUs so great.

System architecture:

A computer system is more than just the CPU and a handful of speeds and feeds. Overall system performance depends on the full system architecture, with CPU just being one part of that system. FPGAs have been finding their way into current, advanced, computer systems in the form of SmartNICs and Computational Storage. SmartNICs and Computational Storage systems are being used by the industry. [14] [15] [16]

There have been some projects that put FPGA technology in the CPU socket [18] but for the most part, FPGAs surround CPUs in vital subsystems. FPGAs are usually found on a PCB that attaches to a PCIe bus and is treated as a slave accelerator.

The near future for FPGAs:

For nearly 35 years people have agreed that FPGAs could be the future of computing. While the duopoly of Xilinx and Altera made billions of dollars replacing ASICs a small group of engineers was working to make computers out of FPGAs. [1] For a long time, it seemed like a battle to advance Reconfigurable Computing against the duopoly.

Device secrecy crumbles:

The FPGA duopoly is now broken with the acquisition of Xilinx and Altera by the top CPU manufacturers. The tacit agreement to keep information secret has dissolved. This leaves the field of independent FPGA manufacturers to innovate into the vacuum left by the ensuing chaos. One of the projects stepping up to address this is the Open-Source FPGA Foundation. [20] OSFPGA is a project that takes an XML file and generates an FPGA all the way to the layout. Rapid Silicon is commercializing OpenFPGA and has raised $20 million dollars to democratize the creation of FPGAs and FPGA fabrics. [21] The CHIPS Alliance is bringing together companies like Intel and Xilinx (AMD) and universities to create standards for low-level FPGA portability. [22]

Fixing the programming model:

Momentum is building from the energy that is pouring into Reconfigurable Computing from many different directions; the day is at hand to form the perfect wave of computer rebirth. Giant companies like Google, Intel, AMD, Microsoft, and community-driven open-source projects are making a difference in the way are being programmed.

---

[1] Except the Xilinx open-source Rapidwright project

The most annoying programming problems with FPGAs are the long compile times to get any feedback on your code changes. Most programmers will reject a coding path that takes hours to complete for one change in the code!

FPGA based, Reconfigurable Computing needs a different programming model from CPU and GPU based machines. FPGA based programming systems need to support DFG and CFG tradeoff optimizations to get the best performance. Doing this automatically is an NP-complete problem. The best way is to separate the DFG and CFG completely. The Hotstate machine only implements the CFG with concurrency being achieved by using multiple state machines. The DFG is implemented in Verilog or VHDL. This gives the programmer complete control over the behavior of the hardware.

New languages like Scala based, Chisel, and Python based, MyHDL, are being accepted by software programmers as a hardware programming language. [24] [25]

System architecture:

Instead of having FPGAs be bit players, our vision is that FPGAs become the system stars. New advances in chiplet technology will enable systems on a chip to be held together with FPGA fabric. Both AMD and Intel could put together the systems that can drive the future of computing.

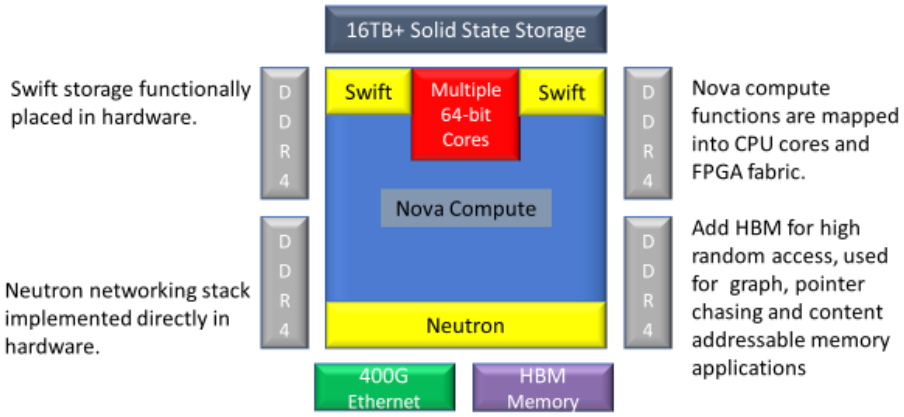


Figure 5: Software Defined Everything (SDE) Vision. Here is an example mapping of OpenStack to an optimized SDE FPGA system architecture. "OpenStack controls large pools of compute, storage and networking resources." [2]

FPGAs are manufactured with multiple high-speed interfaces that allow each device to be self-contained and totally reconfigurable. Combining compute, storage, security, and networking in one device and having protocols you can swap in and out will give the FPGA based systems an unbeatable advantage in performance and latency.

Conclusion:

Computing is at a crossroad. On one side are computer architectures that have been around for 50 or more years. The von Neumann architectures of multi-core CPUs and SIMD architecture of GPUs are facing insurmountable limits of physics and usability that will not carry us into the future. Only FPGA SoC-based devices that contain CPU, FPGA accelerators, storage, security, and communications all in one package have a chance at delivering power appropriate systems capable of Exascale performance and beyond.

About the Author:

Steve Casselman is CEO of HotWright Inc. In 1987 when he won his first SBIR contract to build an FPGA based supercomputer. Steve has 14 issued patents including putting the FPGA in the processor socket and combining FPGAs and CPUs on a single die.

[Two] https://www.openstack.org

[1] S. Casselman, "Virtual Computing and the Virtual Computer," *FPGAs for Custom Computing Machines,* pp. 43-48, 1993.

[2] S. Casselman, "Who is Steve Casselman?," 2013. [Online]. Available: http://bit.ly/WhoIsSteveCasselman.

[3] K. Underwood, "FPGAs vs. CPUs: trends in peak floating-point performance," in *FPGA '04: Proceedings of the 2004 ACM/SIGDA 12th international symposium on Field programmable gate arrays*, Monterey California USA, 2004.

[4] M. B. Taylor, "Center for Dark Silicon," [Online]. Available: http://darksilicon.org/.

[5] E. B. R. S. A. K. S. a. D. B. H. Esmaeilzadeh, "Dark silicon and the end of multicore scaling," *8th Annual International Symposium on Computer Architecture (ISCA),* pp. 365-376, 2011.

[6] D. Z. D. R. B. R. d. S. S. A. M. Milan Radulovic, "Another Trip to the Wall: How Much Will Stacked DRAM Benefit HPC?," in *MEMSYS '15: Proceedings of the 2015 International Symposium on Memory Systems*, Washington DC USA , 2015.

[7] S. Casselman, "Memory architecture optimized for random access". USA Patent 8,977,930, 10 March 2015.

[8] S. A. G. a. C. Patterson, "JBits Design Abstractions," *Field-Programmable Custom Computing Machines, Annual IEEE Symposium on,* pp. 251-252, 2001.

[9] S. A. G. a. E. Keller, "Gene Matching Using JBits," in *FPL*, 2002.

[10] D. G. G. G. M. B. a. M. P. Syed Asad Alam, "ON THE RTL IMPLEMENTATION OF FINN MATRIX VECTOR," *Preprint https://arxiv.org/pdf/2201.11409.pdf,* p. 1, 2022.

[11] T. D. Victor Schmidt, "Quantifying the Carbon Emissions of Machine Learning," https://arxiv.org, 2019.

[12] R. D. M. W. Eriko Nurvitadhi, "Real Performance of FPGAs Tops GPUs in the Race to Accelerated AI," 2020. [Online]. Available: https://www.intel.com/content/www/us/en/products/docs/programmable/fpga-performance-tops-gpus-white-paper.html.

[13] T. Ray, "Xilinx and Numenta claim dramatic speed-up of neural nets versus Nvidia GPUs," 18 May 2021. [Online]. Available: https://www.zdnet.com/article/xilinx-and-numenta-claim-dramatic-speed-up-of-neural-nets-versus-nvidia-gpus/.

[14] S. Casselman, "FPGA virtual computer for executing a sequence of program instructions by successively reconfiguring a group of FPGA in response to those instructions". USA Patent 5,684,980, 4 November 1997.

[15] Wikipedia, "Evolvable hardware," [Online]. Available: https://en.wikipedia.org/wiki/Evolvable_hardware.

[16] S. Casselman, "Reconfigurable-logic-based fiber channel network card," in *Proceedings of SPIE - The International Society for Optical Engineering* , 1996.

[17] S. Casselman, "Computer network of distributed virtual computers which are EAC reconfigurable in response to instruction to be executed". USA Patent 5,802,290, 1 September 1998.

[18] A. M. C. e. al., "A Cloud-Scale Acceleration Architecture," in *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2016.

[19] S. Casselman, "FPGA co-processor for accelerated computation". USA Patent 7,856,545, 21 December 2010.

[20] "Open Source FPGA foundation," 2022. [Online]. Available: https://osfpga.org/.

[21] "RapidSilicon," 2022. [Online]. Available: https://rapidsilicon.com.

[22] "CHIPS Alliance," 18 Feb 2022. [Online]. Available: https://chipsalliance.org/category/announcement/.

[23] L. J. G. A. C. P. I. a. J. Jianyi Cheng, "Combining Dynamic & Static Scheduling in High-level," in *ACM/SIGDA International Symposium on Field-Programmable*, Seaside, CA, USA, 2020.

[24] "Chisel," [Online]. Available: https://www.chisel-lang.org/.

[25] "MyHDL From Python to Silicon," [Online]. Available: https://www.myhdl.org/.

[26] Nvidia, "Cuda programming guide," [Online]. Available: http://docs.nvidia.com/cuda/cuda-c-programming-guide/. [Accessed October 2013].

[27] D. Z. D. R. B. R. d. S. S. A. M. P. R. a. E. A. Milan Radulovic, Another Trip to the Wall: How Much Will Stacked DRAM Benefit HPC?, New York, NY, USA: AMC, 2015.

[28] S. Trimberger, "Retrospective on the First Thirty Years of FPGA Technology," in *Proceedings of the IEEE*, 2015.

[29] S. Casselman, "Computer with programmable arrays which are reconfigurable in response to instructions to be executed". USA Patent 6,023,755, 8 February 2000.

[30] S. Casselman, "Modular, hybrid processor and method for producing a modular, hybrid processor". USA Patent 6,178,494, 23 January 2001.

[31] S. Casselman, "Virtual computer of plural FPG's successively reconfigured in response to a succession of inputs". USA Patent 6,289,440, 11 September 2001.

[32] S. C. a. S. Sample, "Configurable processor module accelerator using a programmable logic device". USA Patent 7,856,546, 21 December 2010.

[33] S. Casselman, "Reconfiguration of an accelerator module having a programmable logic device". USA Patent 8,145,894, 27 March 2012.

[34] M. W. e. a. Steven Casselman, "Accelerator system for remote data storage". USA Patent 8,824,492, 2 September 2014.

[35] Xilinx, "Xilinx Extends Edge Compute Leadership with World's Highest AI Performance-per-Watt," Xilinx, 9 June 2021. [Online]. Available: https://www.xilinx.com/news/press/2021/xilinx-extends-edge-compute-leadership-with-world-s-highest-ai-performance-per-watt.html.

[36] Intel, "TACC: engineering Research in HPC," [Online]. Available: https://www.intel.com/content/www/us/en/customer-spotlight/stories/tacc-engineering-research-in-hpc-video.html.