FPGAs versus GPUs in Datacenters

This article presents position statements and a question-and-answer session by panelists at the Fourth Workshop on Computer Architecture Research Directions. The subject of the debate was the use of field-programmable gate arrays versus GPUs in datacenters.

Babak Falsafi
EPFL

Bill Dally
Stanford University

Desh Singh
Amazon

Derek Chiou
Microsoft

Joshua J. Yi
Dechert

Resit Sendag
University of Rhode Island

We are generating a lot of data that is being fed to datacenters, and we need to find a way to turn that data into services and value. The datacenter space is growing quickly, but parts of that space are reaching their physical limits. A big datacenter consumes about 20 MW, is 17 times the size of a football field (regardless of what kind of football you play), and costs a few billion dollars. Furthermore, data from various think tanks (such as Energy Star) shows that electricity usage is shooting up. For example, in London, datacenter servers already consume 6 percent of electricity, and that's growing at 20 percent per year. Therefore, not only is the capital investment for a datacenter huge, but operational costs are also huge.

This has not always been a problem in platforms, because Moore's law came with Dennard scaling. Robert Dennard, who graduated from Carnegie Mellon University, basically explained that, given the power equation, $P = f^{*} C^{*} V^2$, you have a "silver bullet" in supply voltages (V), so that as you scale voltages down, you exponentially decrease your power. As such, this lets you limit the power per area as you scale down the area. But, if you look at some of the *International Technology Roadmap for Semiconductors* projections from 2014, voltages have leveled off.

What you end up doing is going from a modern server CPU, which is like a race car, to something like the EZ Chip/Cavium efficient core design, whose complexity is specialized for server workloads. The good news with this type of approach is that it reduces the power. But what do you do next? Core complexity is going down. Extrapolating into the future, you may eventually get to the five-stage pipeline. In fact, Nikos Hardavellas showed that, even for server workloads with abundant request-level parallelism, the server won't be able to power up the entire chip (see Figure 1).[1]

Another approach is to use accelerators, which was the focus of this panel. On the one hand, we have massive data, and on the other hand, we have diminishing efficiency. We'd like to bridge these two trends in datacenters, and field-programmable gate arrays (FPGAs) and GPUs are two possibilities. FPGAs comprise reconfigurable blocks, logic, interconnect, block RAM, and I/O. Their density has improved over time, and their libraries offer high-level functionality. They are suitable for spatial computation, and they support arbitrary parallelism. Examples of accelerators in servers using FPGAs, such as the Microsoft Catapult and Intel HARP, have already emerged.

Published by the IEEE Computer Society

0272-1732/17/$33.00 © 2017 IEEE

GPUs have been around for a long time. They were originally developed for graphics, and they have morphed into a dense grid of multithreaded SIMD cores with improved programmability. Datacenter workloads exist that can leverage the dense data parallel processing. For example, Google is using GPUs intensively for machine learning in its datacenters.

Table 1 lists the pros and cons of FPGAs and GPUs. The advantages of FPGAs are arbitrary logic and arbitrary parallelism that evolved over time to deal with rapidly changing datacenter algorithms. Even if you think about Catapult with big kernels, they keep getting updated. FPGAs do not do so well on the programmability side. Programming with today's abstractions and trying to map your algorithms to the fabric is not easy. The tools are fairly cumbersome, and you lose efficiency. In particular, lack of native support for high-level memory abstractions is a big disadvantage.

GPUs, on the other hand, offer high computational density and efficiency and higher-level programmability. However, they are efficient only for data-parallel algorithms, are not good for sparse computation, and are designed for algorithms that require very dense floating point.

In this panel, we had two experts on the topic: Bill Dally, a chief scientist and senior vice president of research at Nvidia, and Desh Singh, the director of virtualization labs for Altera. Their position statements, and an edited version of the question and answer session, follow. Video of the panel is available at www.ele.uri.edu/CARD. —Babak Falsafi

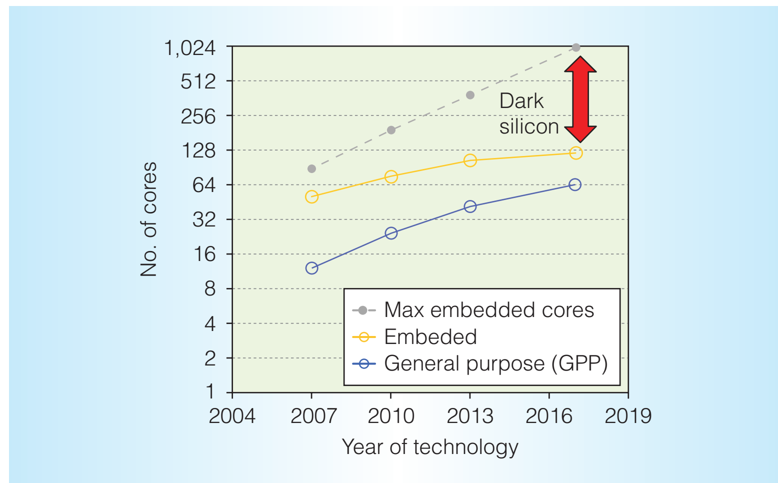

Figure 1. Dark silicon limitations that prevent powering up a whole processor are encountered even when using cores designed for efficiency. (Source: Nikos Hardavellas; used with permission.)

Bill Dally: Use the Right Tool for the Job

This is really a case of using the right tool for the job. If you need to drill round holes, which tool would you use? You would use the drill. If you want to cut straight lines, you use the saw. The problem facing datacenters is analogous. If you need lots of arithmetic performance, integer, floating point, memory bandwidth, and/or tight integration with the CPU to read and write single words in the CPU's memory—or if you need the CPU to read and write single words into your memories—you would use a GPU, because it does all these things better than anything else.

Nvidia's P100 GPU is an example of how you get all these things. It has a terabyte per second of memory bandwidth. The single-precision floating-point throughput is on the order of 10 teraflops. The efficiency fully loaded is on the order of 35 gigaflops per watt, which includes the memory bandwidth, instruction interpretation, registers, and power supplies.

But suppose you need to build something to process gene sequences. Now, you are not operating on numbers, so you do not need the arithmetic. Your memory bandwidth needs are modest, and you do not need tight integration with the CPU. Given those properties, use an ASIC [application-specific integrated circuit]. That's what people build when they're building things to do encoding and decoding and do not need the high bandwidth and arithmetic. If you do not have the volume to justify an ASIC—and turning an ASIC around these days is easily a $50 million proposition—then use an FPGA. But then you are down by something like a factor of 20 to 100 times on the nonhardwired logic that is implemented in FPGA fabric.[2] So, although the FPUs on an FPGA are great, you have to feed those FPUs by building multiplexors, sequencers, and other structures in the reconfigurable fabric. The overhead is enormous and, therefore, you never get the efficiency of a GPU on an FGPA.

So, for arithmetic and memory-intensive problems, use a GPU. They are also much more programmable. You can program them

Table 1. Pros and cons of field-programmable gate arrays and GPUs.

	FPGAs	GPUs
Pros	• Arbitrary logic • Arbitrary forms of parallelism • Can evolve over time	• Computational density and efficiency • Higher-level programmability
Cons	• Computational density and efficiency • Programmability • No memory abstraction	• Data parallel only • Not so suitable for sparse data structures • Much density for floating point

in CUDA, OpenACC, and OpenMP, and soon in extended C++. For nonarithmetic logic with modest memory bandwidth demands, use an ASIC. If you have low volume, use an FPGA, but realize you're going to get this huge overhead and have a really hard time programming it. You basically have to write Verilog or the equivalent. If you do use a GPU, you're riding this constant power scaling up—you will have ever-increasing performance going forward. For example, one company we deal with started out doing neural network training on FPGAs. As soon as it tried using cuDNN on GPU hardware, it scrapped all of its FPGAs and moved over, because it was such an overwhelming advantage using GPUs. So, for workloads like training neural networks, doing large data analytics, and so on, GPUs are really the instrument of choice.

Desh Singh: Specialization for Efficiency

I would like to give you my perspective on GPUs versus FPGAs in the datacenter. The first thing about the FPGA that I would like to explain to people is that you do not really have software running on hardware. The FPGA enables software to define the hardware. Let me explain that. Suppose you have the following high-level code: `Mem[100] += 42 * Mem[101]`. Compilers can take this code and break it down to lower-level assembly instructions.

To run this on a simple processor, the processor would need something that fetches instructions from memory (a load unit), a simple arithmetic unit, and a register file. If you were to implement an instruction on this processor, it would basically fetch an opcode that tells the processor how to configure the datapath for that opcode.

Now, if we executed the assembly language instructions on the simple processor, basically what you are doing is multiplexing that same piece of hardware across time. As we go ahead in time, we are loading from the memory locations, and we are loading the constant values all on the same hardware. We are doing the multiplication, addition, and storage back, again, all on the same hardware. This is essentially what a processor does: it takes the same piece of hardware and multiplexes it to implement different kinds of instructions.

Instead of doing this, suppose we distribute the processor across space. Now we have six processors, each dedicated to its own instruction. If we were able to "unroll" the processors in this way, we would have something that's fundamentally different than a processor. Because each processor is dedicated to a single fixed function, we can remove the fetch unit, because every processor knows what it is going to be able to do. Similarly, we can remove the unused ALU [arithmetic logic unit] operations and the unused loads and stores.

But, because we started with six individual processors (even before we removed anything), we need them to communicate with each other somehow. We can wire up the registers and have them communicate with each other in some fashion. What you are starting to get now is something that looks more and more like a hardware circuit. But this hardware circuit is somewhat inefficient. We do not really need to match the latency of the original program running on the processors; we just have to match the functionality. As such, we can retime and reschedule

this datapath to be more efficient. This is essentially what you do when you try to transform the program to run on an FPGA. You are essentially creating a custom piece of hardware to implement your algorithm. You use the FPGA's programmable resources to implement that circuit. You have millions and millions of programmable logic elements that you can join to form higher-level functionality using our configurable routing structure.

And you can do all this automatically. You do not have to code this Verilog or VHDL. We now support software technologies that can take your programs and compile them to your hardware. For example, suppose that you are coding something in OpenCL for a heterogeneous environment. We have a compiler that automatically does exactly what was just described. And people are now starting to build higher-level abstractions on top of this technology. They have built a Java compiler and things as complex as Spark and Hadoop implementations that directly map to FPGA hardware.

When you start using software to define hardware like this, it is not just about defining the accelerator itself; you will find that there are other fundamental things you can now do with the FPGA. For example, you can customize how you move data around. With other accelerator technologies, you have to move data from the external world, put it into memory, and then process it; that is why you need really good high-memory bandwidth. But one compelling value proposition of FPGAs is that you can just process the data directly inline. Many algorithms process the data directly from network streams, video streams, and external devices of all sorts. With an FPGA, this data can bypass memory and directly enter the FPGA, which gives you very efficient types of computation.

FPGAs are also evolving to match the needs of today's workloads. They are not just traditional LUTs [lookup tables] and flip-flops. Many years ago, FPGAs included hardened dual-ported SRAMs. In addition, they now contain hardened PCI Express interfaces, external memory interfaces, and processors. Altera has introduced a new floating-point DSP [digital signal processing] block, resulting in tremendous resource and cost savings over implementing floating point in FPGA logic.

In short, the FPGA enables the implementation of custom data and control paths from standard software languages. I think that makes it a formidable component for the datacenter.

Moderator and Audience Questions

This section presents an edited transcript of the question-and-answer session.

Programmability

Babak Falsafi: Let's talk about programmability. Clearly, FPGAs are behind in this respect. GPUs already have higher-level interfaces and rich libraries. Is that really a showstopper for FPGAs? Is there nothing that FPGAs can use to bridge the gap? Is there any fundamental advantage to GPUs when it comes to programming interfaces?

Bill Dally: In the last panel, somebody said, "It's not the ISA, it's the ecosystem." I think that programming systems means both the programming tools and the ecosystem. In a GPU, you have your choice of programming languages. People have data-parallel programs. They most often get by with OpenMP or OpenACC. They take their original code, Fortran, often without changes, put in the directives, and get an effective GPU program.

For many applications—for example, neural networks—developers don't even need to go to that level of programming. There are very fast GPU ports of both Caffe and Theano, so you can basically download software from the web and be up and running instantly. There are other standard GPU libraries for other vertical areas, such as in oil and gas or the financial markets. There are also effective standard numerical libraries for GPUs, so the entire ecosystem is there. It's not so much a matter of it being easy to write OpenACC code, or CUDA code for that matter. It's that there's a huge ecosystem built up that makes it extremely easy to take whatever your problem is and get a large fraction of peak out of a GPU.

Falsafi: Desh, why are FPGAs behind when it comes to programming interfaces, and can they actually catch up? Is this a

showstopper? Is there something fundamental there?

Desh Singh: Well, I will admit the FPGA programming environment is, today, less mature than the GPU's programming environment. I think it really is just a matter of time. FPGAs have made a strategic shift in the kinds of markets that we're trying to address. The datacenter has become a large and important market for us. As that became true, we had to create both suitable hardware and software tools. Datacenters are a large space that others believe will continue to grow, as evidenced by Intel's record-breaking acquisition of Altera.

If you look at both of the big FPGA vendors, Xilinx and Altera, they have compilers that will compile OpenCL, C/C++, as well as Matlab and SimuLink to Verilog that is then passed through the standard toolchain. You can only imagine what would happen in the future with a larger software effort focused on enabling these devices. Another thing I'll note is that when customers are going off and implementing their own special code, they often can't find exactly what they want in existing libraries. If you can take those libraries and automatically translate them to Verilog, are GPUs really that much easier to program? Can I actually optimize my code for an FPGA or optimize it for a GPU? The difficulty is in the types of optimizations you'll encounter when you're going for maximal performance. So, I think FPGAs are now on good footing.

Falsafi: So, you both agree that eventually they are going to converge. I'd like to try to push this question toward killer apps in the datacenter world and the fact that obviously for programming to happen, we have to push the abstraction levels higher. DSLs [domain-specific languages] are also emerging as an interesting opportunity to try to map programs and algorithms to FPGAs or GPUs. Do you see a difference there in using DSLs and tool chains that will eventually map all the way down to FPGAs and GPUs?

Dally: Take training neural networks, which I think is one of the big killer applications in the datacenter today. You have the choice of a mature set of libraries running the GPU or an emerging set of libraries with the FPGA. The only reason you would go for the emerging set of libraries is that there's something else you would gain, but because you have better computational density on the GPUs, better flops per watt, better memory bandwidth, better integration with the CPU, there's no motivation to cause yourself a lot of pain in programming, because there's nothing to be gained. So, I can see if there's something to be gained, that they may eventually catch up on the ecosystem front, but unless there's some compelling advantage, it doesn't motivate people to endure pain.

Singh: I would disagree with that. I think a lot of people are looking at FPGAs because they find that they can reach levels of efficiency in both performance and power that they can't achieve with GPU technology today, so we have a lot of customers that actually moved away from GPUs just because of power reasons and moved to lower-cost and efficient implementations on the FPGA. That's really what drives people to FPGAs, and there are many success stories.

Dally: What are they doing? Because clearly if they're doing floating point, they would get better performance per watt on the GPU.

Singh: Even with floating point, now that we have hardened blocks that can do floating point, I think the performance per watt is actually much more favorable to the FPGA than it is for the GPU.

Dally: I think the performance per watt on the floating-point units themselves is about the same, but when we quote our numbers, we're quoting a fully loaded number, and when you try to implement the rest of the circuit in LUTs, you wind up way down.

Singh: I'm also including full system power and not just the FPUs. The numbers you quoted earlier were done by academic researchers. One of the things that you find about FPGA companies is the secret sauce is actually in the software. I got my PhD at the University of Toronto, where those researchers came from, and the software they have available in mapping circuits to FPGAs is so inferior to the things that we have in the industry now. That's part of the reason that they were not able to really find efficiency in taking circuits and mapping them to the FPGAs and then comparing them to ASICs.

So, I would say internally we have our own numbers. We're able to do these comparisons much more favorably.

Dally: But if the customers trying to put the stuff into the datacenter can't do that, then it doesn't do them any good.

Singh: That's right, but the customer is trying to put the stuff in the datacenter, not off using academic tools. They're using industry tools that Altera and Xilinx are providing.

Parallel and Nonparallel Computation

Falsafi: On that note, I'd like to turn the question to Bill. Bill, GPUs are amazing for data-parallel computation. What happens if your computation is not data parallel? It's still parallel, and you can map it spatially to a fabric, but it's just not data parallel.

Dally: It's interesting. One of our big success stories with GPU programming is actually ray tracing, things that studios like Pixar use to render their feature films. They actually don't use the graphics hardware on the GPUs, because that texture-maps polygons. They basically run ray tracing as CUDA programs on the GPUs. When we first started looking at ray tracing, it was the poster child for non-data-parallel programming. GPU's SMs are 32 SIMT [single instruction, multiple thread] lanes wide. The average occupancy was 1.5 for the ray-tracing program, and now it's 31. Although there were a lot of rays operating in parallel, it wasn't data parallel, because different rays were processed differently. The solution was to sort the rays so that, at any point in time, that SM would be doing 32 rays that all needed to have the same operations applied. Most programs that aren't completely data parallel have similar properties, where there's a lot of parallelism and you're simply performing a bunch of different cases, one case per data element. You can mechanically transform them so that they get good performance on SIMT.

Falsafi: So, if I summarize that, if you have a lot of data, all computation is data parallel.

Dally: That's a good way to put it. So you have data parallelism, just not regular.

Falsafi: Desh, do you want to respond?

Singh: Let me take it to a more practical algorithm. Compression, for example, where you're trying to take in a screen in real time and do compression, things like gzip, where it's based on the history of what you've seen before. If you look at how that algorithm is written, it looks very sequential and, therefore, it's hard to make it something that's purely data parallel. It is actually much more pipeline parallel, and when you take that algorithm and take its serial implementation and just run it through an OpenCL compiler for an FPGA, you get an efficient pipeline-parallel implementation that has incredible performance.

Dally: And I think you're making my point exactly. These are the cases where you're not really doing arithmetic and you don't need a lot of memory bandwidth because you're just processing a stream that's coming in—unlike, for example, ray tracing, where you have a huge scene and a big bounding volume hierarchy to traverse. In those cases, I would say, yeah, if you can afford it, do an ASIC, and if you can't, do an FPGA. But if you've got lots of floating point, you need lots of memory bandwidth. It's also things like a memory system where, because graphics demands high bandwidth memory, we build a memory system that, in the Pascal generation with stacked memory, consumes five picojoules per bit. If you have an FPGA with an SDDR memory system attached to it, you're off by a factor of five. You're at 20, 25 picojoules per bit just accessing the memory, and you won't get a terabyte per second. You just don't have enough pins to stack enough SDDR around it to do that. So, if you're doing a gzip-like thing, use an ASIC or an FPGA. If you're doing a numerical computation, like training a neural network, use a GPU. I'm not saying one is better than the other. I'm saying that they're different tools for different jobs.

Floating Point

Falsafi: Bill, with respect to neural nets, I know that you've been talking about floating point. The Chinese Academy of Sciences has been working on ASICs for neural nets, and one of the things it shows is that it's much better to do fixed point than floating point for neural nets. GPU fabrics are definitely designed for high-density floating point. If you're actually doing high-density fixed point, you may lose.

Dally: It's really a question of precision and not float versus fixed, and depending on whether you're doing training or inference, there's lots of papers I can point you to that show you can get by with 16—and in some cases even 8—bits of precision. That's one reason the TX1 that we're shipping right now has support for FP16, because a lot of the neural nets in our Drive PX package for self-driving cars are actually done using 16-bit fixed point. You can imagine that future GPUs may have very good support for lower precision, both float and fixed, for the same reason. So, I don't think that the type of arithmetic you're doing matters very much. You can build dense arithmetic units and make them so you can segment them up for different-precision arithmetic. What really matters is that you have a fabric that is optimized for executing arithmetic codes at really high memory bandwidth and close CPU integration.

Memory Bandwidth

Falsafi: Desh, do you want to comment on higher-level abstractions in FPGAs and DSPs, and how we could bridge that gap with efficiency?

Singh: Sure. I just want to get back to the comment about neural nets. If you look at the way GPUs implement these styles of algorithms, it comes down to an SGEMM [single-precision floating general matrix multiply] essentially, where you're just bottlenecked by external memory bandwidth. If you were to think about a slightly different way of architecting these algorithms using things like sliding windows and such, you actually get tremendously efficient implementations on the FPGA without having to go to external memory as well.

Dally: The leading-edge neural net for vision, like VGG net, has something on the order of 130 million parameters. Multiply that by 4 [bytes per parameter], that's over 500 megabytes of data. Now, you have to fetch that every cycle. That's memory bandwidth. You can't keep that all on your FPGA, even if you want to.

Singh: Certainly you can't keep that all on the same FPGA, but you can also look at multi-FPGA systems where you stream between chips and things of that nature, and then compare that to a multi-GPU system. I think in those cases you can actually see really good results from doing that sort of thing. So, I think it really comes down to thinking about your algorithm, thinking about what is the optimal architecture to implement these sorts of things, and I think all of us are right in a sense. There are tradeoffs, right? It really depends on what you're after and how you can actually architect it to take advantage of these things.

Falsafi: I want to ask a related question, because Bill keeps bringing it up: the advantage GPUs have with respect to memory bandwidth. Obviously, GPU memory systems are highly specialized for the GPU style of data-parallel execution, but is that really a showstopper of FPGAs? Can't we adopt some of these technologies for FPGAs? These can certainly benefit from wide I/O and 3D.

Dally: This is where you get to the issue of building things out of LUTs, and Desh may be sort of refuting the paper from his colleagues in Toronto without giving me another paper that cites better results, which doesn't seem kosher to me. It still boils down to the fact that I can build, say, a six-input AND gate in something like 8-by-10 tracks on an ASIC. When you look at the amount of area it takes on an FPGA to build the same thing, it's on the order of about 80 times the area, and power tends to go with area, so you're going to be at 80 times the power. So, if you're building a memory controller out of LUTs rather than having it built out of gates the way we build the controllers on the GPU, you're going to wind up burning—and I was taking the 20 to 100 to be sort of a nice number—you can come up with a lot of worse numbers. So there's overhead for the FPGA implementations, and these memory systems are really high bandwidth. If you have a little inefficiency in there, it's going to destroy your power efficiency, and they wind up needing to be tuned in a way that requires lots of logic to queue up many things, so it's not just a few LUTs. It's going to be a big hunk of logic, and it's going to be quite inefficient to build out of FPGA fabric.

Falsafi: Desh, that's certainly a strong argument for GPUs when it comes to memory bandwidth. Do you want to rebut that?

Singh: First, I don't remember the last time a customer came to me and asked to implement an AND gate. It's usually a lot more complex things, and you can pack these things together and you get more efficient results than looking at a little microbenchmark, so that's number one. Number two is FPGA architecture is evolving, like you said. We recently announced 3D integration with system package technology, so I think FPGAs will have the same high memory bandwidth capabilities that GPUs currently have. At that point, FPGAs will be competitive on GPU-compatible applications, especially in the world of high-performance CPUs. I think once that happens, you will see a long-term bifurcation in terms of what an FPGA is good at versus what a GPU is good at. I mean, an FPGA is not a GPU, and a GPU is not an FPGA, and I think we have to come to the realization that in the world of heterogeneous computing, there's probably room for all of these different kinds of devices in a datacenter. We just have to figure out how do we programmatically create tools that take advantage of them.

Complex Algorithms

Audience member: In terms of complexity of algorithms, are there algorithms that are more suitable to spatial computing versus algorithms that are more suitable to GPUs? In the ISA panel, the ecosystem was an important factor. Is it possible that people are developing high-complexity algorithms like $O(n^2)$ algorithms because they can run on the GPU, whereas that could be an $O(n)$ algorithm that could actually efficiently run on FPGA? So, the choice of algorithm is driven by the ability of GPUs, right? Rather than what's the most efficient algorithm?

Dally: I think people find the most efficient algorithm and then they look at how to map it to hardware. It's really hard to overcome a weakness in asymptotic complexity with just brute force hardware. If you've got a lot of data to process, going from n to n^2 is a loser no matter what you're doing.

Singh: I would agree with that, but I would think that people choose implementations of algorithms ahead of time before trying to target technologies, so you'll often find that people will write an algorithm in such a way that it doesn't consume enormous amounts of memory bandwidth, whereas if they can restructure it even very slightly, it could have a lot more efficient implementation on architecture that's much more suited for stream or dataflow styles of computation. So, I don't think it's essentially the algorithm that people struggle with, but it's more of how to implement it for maximum efficiency.

Falsafi: Can I push that question again toward datacenters? What do you see as killer applications or algorithms that would be more suitable to FPGAs versus GPUs? Is it the complexity of the algorithm or is it just better suitability in terms of mapping when it comes to FPGAs?

Singh: One thing that you can immediately come up with for the FPGA is anything that's being processed directly from the networks, relieving the operating system from having to do that sort of thing; the FPGA can just take that data and direct it to the network and do a lot of network offload in addition to acceleration on that data before you even have to send it back to the processor. So, anything of that style.

Falsafi: Do you have examples of prototypes that actually can map the network stack or are network-oriented services?

Singh: Well, people are doing lots of stuff like this today. If you look in the world of high-frequency trading, you want to offload things like processing of TCP and UDP, put that directly in hardware, get deterministic results from it without it having to make a processor handle that kind of stuff.

Falsafi: Bill, what are examples of algorithms that you foresee emerging that are of interest in datacenters today?

Dally: For GPUs, things that are higher arithmetic intensity and higher memory bandwidth, the big one today is training neural networks, but also almost any numerical computation falls into that. I mentioned ray tracing, which you actually may not even think of as a high numerical bandwidth, high memory intensity thing, but it is. Let me also comment on the streaming issue that Desh has brought up a couple times. It's really an issue of reuse. Right? You always try to structure your code so you need to make the

smallest number of main memory accesses, because those are expensive, so modern GPUs that have many megabytes of L2 cache can be structured to stream as well. You simply order your code in such a way that as you produce something, the working set remains in L2, and you fetch it from there and you don't go back out to memory bandwidth. And so it might be apologizing a little bit for having weak memory systems in modern FPGAs, but FPGAs aren't uniquely able to stream. In any machine you can restructure any computation so you can get reuse out of your storage hierarchy.

Falsafi: We have a question from Tom Wenisch of Michigan. There are a lot of algorithms that you would like to accelerate in the datacenter space; these algorithms are evolving, but would it be practical to go to an ASIC in terms of cost as compared to an FPGA, which requires only reconfiguration?

Dally: Yeah, I think it's a huge advantage to be able to respin your hardware on a regular basis, but you are respinning with this huge disadvantage. I mean, we can argue about whether you're implementing a six-input AND gate or whether it's a tree of logic with six inputs, but you're still off by orders of magnitude—at least one, and probably more like one and a half—in area and energy for that logic implementation. So if you can be down by a factor of 30, yeah, go do that, but if you need to be at a factor of one, you really need something that is a little bit more "performant."

Falsafi: Desh, do you want to comment on any of the customers in the datacenter space that benefitted from being able to reconfigure algorithms?

Singh: I think that's the fundamental reason that people use FPGAs rather than ASICs in any environment, really. Algorithms are constantly changing. Standards that people are interested in are constantly evolving. You can't just figure out today "this is exactly what I want to implement," and it's going to stay the same until the rest of time, especially in datacenters, where you don't know the workload ahead of time. And even if people do know what they're trying to implement, it's going to change in a couple weeks or months, and that's where you get a big value proposition from the FPGA. I think we keep coming back to the question of the efficiency of the implementation on the FPGA. I would point out that anything that's programmable has inherent inefficiency. People can now measure how inefficient an FPGA is because it's programmable hardware; something that's software programmable in the conventional sense has inefficiency. It has circuitry that goes off and fetches instructions. It doesn't just have the pure functional units that are off doing the actual computation. It has all the stuff around it that's routing data to the computer, so everything has inefficiencies. It's a question of quantifying that.

Dally: Right, we can quantify that. If you look at the SM [streaming multiprocessor] of the Maxwell X1, the Tegra part—in the SM, half of the energy goes into the floating-point unit, so our inefficiency compared to wiring the floating point into one another is two. I think that for an FPGA, it would be very hard-pressed to come anywhere close to that number.

Singh: But again, you're doing this for one specific type of algorithm. Let's say we can't just measure the hardware itself. It's sequencing the instructions, and how much efficiency do you get over the lifetime of the algorithm itself?

Dally: That is running a particular actual application in X1.

Suitability for Datacenters

Falsafi: We have a question from James Hoe at Carnegie Mellon University. There's a lot going on in the datacenter world that may not be a good fit for either FPGA acceleration or a GPU. We've been working on this for the past 15 years. A lot of the services that are spending most of the resources in a server are actually just pointer chasing. In the modern world, there are a number of such services, such as data serving, memory caching, web serving, and web search. This is just basically unstructured data structure traversals, and those services are not data parallel. So, what are we even going after? Are these kinds of services a good fit for GPUs or FPGAs?

Singh: Think of a world where you have to do things like pointer chasing, like traversing some kind of graph, and you just have huge data structures stored in some kind of

memory system. Let's presume it's on the processor's memory system. The FPGA is hanging off like an accelerator. You do need to go to technologies like shared virtual memory. That's something that we've just introduced with our OpenCL compilation framework, and we can do this over various kinds of transport mechanisms, like QPI [QuickPath Interconnect]. So as you start to see FPGAs become more tightly integrated with processors, and especially having interconnect technologies where you can start to share some of the same memory space, I think it makes it a lot more attractive to do these sorts of algorithms. Before that, all of the stuff that you'd be able to do in FPGA was essentially offloading memory to some kind of accelerator, having some processing, then coming back or having that kind of disaggregated model, where the FPGA is off just doing its own thing, processing from the stream and sending results back. So this opens up a whole new world coupled with a much tighter integration, but I think it is really too early to say.

Dally: I think GPUs are already pretty well suited for datacenters. We've made successful deployments and they're actually very good at pointer chasing, and a lot of people who do graph algorithms think the current Graph 500 winners for the small size graphs, the things that will fit into the 32-gigabyte memory of the big GPU are fastest on GPUs because they have the fastest memory bandwidth. And if you have enough parallelism in the graph, you can be doing all those pointer chases in parallel. They're also good at chasing pointers in the CPU's memory; the upcoming Pascal generation of GPUs has NVLink, which makes that a sort of 150-gigabytes-per-second path to CPU memory. So, you can basically use all the bandwidth to the CPU memory from the GPU, just as if it were the GPU memory. The place I think the GPUs could do better at integrating into the dataserver really has to do with networking and access to the network. Right now, you tend to chop your problem up and have little pieces of the problem on GPUs. If people thought of datacenters more as if they were big HPC [high-performance computing] machines with a PGAS [partitioned global address space], you could do some really interesting things doing puts and gets over the network from GPUs and be able to run very large problems, very large graphs, at whatever the bandwidth your network would limit you to.

Customization

Falsafi: There was a comment and a question from Jason Cong of UCLA. I think each one deserves a bit of debate. The comment was that, in fact, FPGAs are not just a good fabric for accelerating your computation, but also for customizing your memory accesses. Would you like to comment on that?

Singh: Certainly. I think one of the interesting things that you see with being able to customize and accelerate and improve your application is to be able to create a memory system that's ideally suited to feeding those functions. So, with applications in the datacenter for you to do things like ranking documents and scoring, we often find that the things that you're trying to do are very irregular lookups into large tables in off-chip memory, and one of the ways that you can get around that is to build an efficient hash table so that you don't have to go off to off-chip memory if you don't have to. When you look at implementing that stuff on an FPGA, you can make a gigantic, multiported hash table incredibly trivially in software, so being able to customize that stuff for the application gets you incredible efficiency in applications like that.

Dally: I actually tend to agree with that. You have a bunch of block RAMs and instructions that you want, you can build your storage hierarchy however you want. We have lots of L1 caches, scratchpads, and L2 cache, so we can, for the same size working set, build data structures that emulate that, but we don't have the ability to change or replace algorithms in the mapping. I think that actually is an advantage of FPGAs, but again it's something that when you're down to writing Verilog you're not going to get that out of OpenCL.

Form Factor

Falsafi: The question from Jason Cong was about form factor. Is there an advantage when it comes to form factor?

Dally: Yeah. If you saw the photo I had in my slide deck, that's what's called an xSM module, and that was a Pascal, which we're not shipping yet. We are shipping xSM modules with Kepler now, and so you don't need to have a PCI card form factor to put a GPU in your server. In fact, many people ship 1U servers with GPUs in them using our xSM modules, which are small daughter cards. They're smaller than the size of an index card with a GPU, all of its memory and its power supply on one module, so the small form factor exists.

Singh: I would just echo what Jason said. You'll find that a lot of customers are choosing FPGAs just because of the amount of density in terms of boards that that need to be added to these systems.

Dally: Form factor really comes down to density. What I've observed dealing with a bunch of different datacenter customers is density ultimately becomes a question of energy efficiency. Most datacenters are limited to so many kilowatts per rack, somewhere between 20 and 50 kilowatts depending on which customer you're dealing with, so it really depends on who's able to deliver the best performance per watt. But right now, GPUs are very good at doing that on things that involve stuff that looks like dense or sparse linear algebra.

Reconfiguration

Audience member: FPGAs in the past have had an advantage that you could dynamically reconfigure them, and a lot of effort was spent in reconfiguring them quickly, so you can deploy whatever new operations you have. What role does that play in a datacenter setting, and is that something that is actually of great value?

Singh: I think it's incredibly valuable, actually, because in certain aspects of datacenter computes, workloads are just completely elastic. You can't predict ahead of time exactly what a customer's going to be running, especially when you have multitenant systems. If you want to be able to allocate acceleration resources to these people—where you don't know exactly what's going to be on the FPGA, so you want to be able to dynamically swap things in and out, depending on the current needs of what the system is actually doing—things like dynamic or partial reconfiguration are big technologies that enable that sort of thing.

Falsafi: If I understand in the current setting, for example, with some of the accelerated neural nets or machine learning kernels, they're mostly accelerating the training, which is computationally intensive and spends much time on the offloaded reconfigurable fabric. So, the configuration time, if you're doing it once every 24 hours, is not a big deal. Do you see that reconfiguration speed playing a great role in the future?

Singh: I think in the future it will. Currently, the way the FPGAs are deployed, the configuration doesn't change all that often, but there was an earlier question about what kinds of things would you change in the FPGA or GPU architecture that would enable even more penetration into datacenter. That would be my answer. If we could have much faster reconfiguration speeds, a much finer grain as well, I think that would allow us to swap things in and out much quicker and enable us to do very fine-grained offload in FPGAs.

Dally: I just want to point out that in a GPU, this is called "jump," and it happens in a couple cycles.

Falsafi: Actually, I'd like to go back to that in GPUs, which basically means that you have to reprogram and launch your kernels in a programming interface.

Dally: You're not talking about writing the code. You're just talking about loading it.

Falsafi: Just loading it? But let's reformulate that question. Would GPUs actually have an advantage when you have a new algorithm and you want to just launch it?

Dally: Well, I think the original question was about dynamic reconfiguration. I mean, people basically have GPUs deployed to run whole algorithms today, and it literally is jump. The code is actually all there. They simply say, "okay, you're going to run this code, now jump to that thing," so the kernels are all good. Now, the length of time writing the original kernels is another issue, and that's where writing in Open MP—a directives language—is really the most productive mode of operation. Moving down to a language like CUDA or OpenCL would be next, and

if you have the right Verilog, like you would for a highly tuned FPGA algorithm, that would take the longest.

Integration

Falsafi: Can you comment on integration? We've seen GPUs in the discrete form and those integrated on chip now. Catapult is a discrete form of FPGA, and now we're seeing FPGAs integrated into the CPU. Integration of GPUs and FPGAs: is there an advantage to either one? How would you foresee that coming? Bill?

Dally: We see an advantage in integrating with CPUs. In fact, an approach we've taken is what I call "logical integration," where, through the development of NVLink and ultimately coherence technologies, you have tight coupling between the CPU and the GPU memory; both very high bandwidth, very low latency, and transparent sharing of memory. We think it's much less important whether things are physically integrated, on the same die, so we do that in our low-end process. Things like the TK1 and the TX1 for mobile are integrated, because economically it makes the right sense, especially where you're coming from one monolithic die with four or eight ARM cores and a GPU. At our high end, we actually see the economics pushing the other way, to motivate the logical integration but keep the chips physically separate, which lets you make both the CPU and the GPU chips more powerful and rev them independently of one another. As to an FPGA, it's interesting to think about. I haven't seen any applications that really demand that.

Singh: I really can't comment on what's going to happen with Intel and Altera as our relationship gets closer, but today what you see with FPGAs is that we're incorporating things like ARM processors and having very tight integration on the same die between the processor and the FPGA, and that enables all kinds of classes of algorithms, because you have a very low-latency communication that allows us to do very fast offload on these kinds of devices. We're also looking at the same kinds of technologies for the logical integration, where things like QPI enable the sharing of virtual memory between the FPGA device and the processor, so I think in the future you'll probably see closer physical integrations.

Looking to the Future

Falsafi: We have a question from John Demme at Columbia University. We've talked about convolutional neural networks as an application today, and other machine learning kernels. Let's extrapolate to 10 years from now. What would be the advantages and disadvantages of GPUs and FPGAs? How would they fare? Let's go to Bill.

Dally: I think GPUs are going to continue evolving the way they have. I don't think it's a question of spatial versus temporal. GPUs get their performance from parallelism—that's spatial. And it's not spatial versus temporal at all; it's two different types of spatial architectures. The GPUs are going to evolve where they're basically going to have more streaming multiprocessors. They're going to be faster. You're going to have a memory system that's deeper, with more on-chip storage and higher bandwidth off chip, so I don't see any radical changes coming. I think it's going to be more and better of the same thing, because it works very well.

Falsafi: Okay, Desh, FPGAs on steroids 10 years from now?

Singh: I can't speak exactly to what will be created, but the way I see the world is that we're moving to more and more heterogeneous systems on chip. FPGA technology is really evolving now and incorporating things like hardened processors. You can imagine that trend even going further and further. And we might find that, 10 years down the line, we have an architecture that looks like hybrid GPUs/FPGAs from many different companies. We have multiple processors as well as some of the advantages that you get from hardware reconfiguration, so we'll see what evolves.

F*alsafi:* Let's have a closing statement, and since Bill went first, I'll ask Desh to issue a quick closing statement.

Singh: I think this has been a really good debate, and there's been lots of points of contention, but I think it just is a sign that there are applications for both GPUs and FPGAs. When it comes to picking the right balance for your particular datacenter and your

application, I think it's a big opportunity for all of the researchers out there to go off and build to this and figure out exactly how we can enable this.

Dally: I'll go back to my opening statement, which is "choose the right tool for the job." GPUs have the best performance per watt for things involving arithmetic, the highest memory bandwidth, the most efficiency in terms of picojoule per bit memory bandwidth of anything out there, and the tightest integration with the host CPU—so, if the problem you're doing involves those, use a GPU. If you're doing something like zip—I guess that was the example given—where none of those things fall into the category, again, use an FPGA. I think it's not a question of whether one is better than the other. They're completely different things. MICRO

References

1. N. Hardavellas, "Chip Multiprocessors for Server Workloads," PhD dissertation, Computer Science Dept., Carnegie Mellon Univ., 2009.
2. I. Kuon and J. Rose, "Measuring the Gap between FPGAs and ASICs," *Proc. ACM/SIGDA 14th Int'l Symp. Field Programmable Gate Arrays*, 2006, pp. 21–30.

Babak Falsafi is a professor in the Department of Computer and Communication Sciences at EPFL and the founding director of the EcoCloud research center. His research interests include datacenters, emerging technologies, and evaluation methodologies for servers. Falsafi received a PhD in computer science from the University of Wisconsin–Madison. Contact him at babak.falsafi@epfl.ch.

Bill Dally is the Willard R. and Inez Kerr Bell Professor of Computer Science and Electrical Engineering at Stanford University. He is also a chief scientist and senior vice president of research at Nvidia. Dally received a PhD in computer science from Caltech. He has received the ACM Maurice Wilkes Award, the IEEE Seymour Cray Award, and the ACM Eckert–Mauchly Award. He is a member of the National Academy of Engineering and a Fellow of IEEE, ACM, and the American Academy of Arts and Sciences. Contact him at dally@stanford.edu.

Desh Singh is a senior manager of software development at Amazon in Toronto. Singh received a PhD in computer engineering from the University of Toronto.

Derek Chiou is a partner hardware group engineering manager at Microsoft and a research scientist at the University of Texas at Austin. His research interests include accelerating datacenter applications and infrastructure, rapid system design, and fast, accurate simulation. Chiou received a PhD in electrical engineering and computer science from the Massachusetts Institute of Technology. Contact him at derek@ece.utexas.edu.

Joshua J. Yi is a patent litigation associate at Dechert. His research interests include high-performance computer architecture and performance methodology. Yi received a PhD in electrical engineering from the University of Minnesota, Minneapolis, and a JD from the University of Texas at Austin. Contact him at joshua.yi@dechert.com.

Resit Sendag is a professor of electrical and computer engineering at the University of Rhode Island. His research interests include microarchitecture, memory systems, and simulation techniques. Sendag received a PhD in electrical engineering from the University of Minnesota, Minneapolis. Contact him at sendag@ele.uri.edu.

myCS Read your subscriptions through the myCS publications portal at http://mycs.computer.org.