Mipsology:
The Future of FPGA-Based Machine Learning

Abstract

A.I. is an exploding market, projected to grow at a compound annual rate of 62.9 percent from 2016 to 2022. Neural networks are in greater demand than ever, appearing in an ever-growing range of consumer electronics. Even so, the processing demands of Deep Learning and inference exceed the capacity of traditional CPUs and GPUs – and the shortfall becomes even more pronounced at scale, in data centers. FPGAs, by contrast, are already used in many data centers, and measure up to the processing demands of Deep Learning and inference – but until now,
they’ve been very difficult to use without expertise. Mipsology’s team of expert founders, leveraging more than 20 years of experience designing high-performance electronic systems, have brought the cutting-edge technology of FPGA-based acceleration to bear on neural network processing, delivering a fully integrated FPGA replacement for GPUs, which works seamlessly and easily with all neural networks, and is ready-made for use in data centers.

Why use an FPGA for Your Neural Network Inference?

Machine learning is moving from GPU to FPGA… and momentum is growing. Field Programmable Gate Array components (FPGAs) have already proven to be the fastest, most flexible, power-efficient chips on the market.

That means they’re ideally suited to be used in data centers, especially in the fast-moving world of machine learning.

FPGA is crucial for Deep Learning

Demand for Deep Learning continues to skyrocket, spurring the development of a variety of new technologies. Some of these applications are very visible – for example, the self-driving car. Others, such as speech recognition and face recognition, are less obvious, but nonetheless essential to our daily lives. Many more exist to serve specialized but important verticals, such as identifying cancers on medical images, identifying breaches into computer networks, adapting efficiently power consumption in data centers, attempting to predict election results, or improving the internet ad profiling.

Deep Learning is a method of computerized learning performed by neural networks, which are simplified mathematical simulations of how brains function. By looking at many examples or associations, the neural network is able to “learn” connections and relationships much more quickly than a traditional recognition program could.

For example, a neural network might listen to many vocal samples, and use Deep Learning to learn to “recognize” the sounds of specific words. This neural network could then sift through a list of brand-new vocal samples, and correctly identify the samples containing the words it has learned, using a technique called inference.

Even the simplest neural network requires a tremendous amount of mathematical “heavy lifting.” Deep Learning requires an even greater number of calculations, and the computing demands of inference are greater still. This is because, whereas Deep Learning typically only needs to be performed one single time, a neural network must perform inference again and again, for each new sample it receives.
The fact that many popular applications use inference to perform common tasks like speech recognition and image tagging – millions upon millions of times every day – highlights an obvious need for a categorical upgrade in computing power.

The future of Deep Learning is on FPGA
We founded Mipsology in 2015 to deliver fully integrated solutions that accelerate the computing of Deep Learning and inference. Our focus has always been to deliver full solutions for data centers, with the goal of supporting very large loads and high throughput. Rather than attempting to accelerate one neural network for a single application, our technology, called Zebra, can accelerate all neural networks for all applications – effectively replacing the GPUs currently used by most scientists.

Mipsology’s founders bring together more than 20 years of experience designing high performance systems based on many FPGAs. In the past, several of our team members led the design of multiple emulators for high-speed ASIC verification, based on thousands of Xilinx FPGAs.

In January 2017, our team demonstrated Zebra executed on a KU115 – showcasing the fastest 100 percent FPGA-based solution computing any neural network inference (as far as we’ve been able to determine).

We ported Zebra to AWS in just two months, and will soon offer access to our Zebra f1 implementation on AWS Marketplace. In Q2, we will support all convolutional neural networks (CNN) similar to AlexNet and GoogLeNet, delivering inference performance reaching 4000 images per second and more than 100 images per second per watt. We will also add support for Caffe and MXNET infrastructures, so our users will never have to worry about their FPGAs running “undercover.”

In the near future, we’ll continue to add new features that deliver stronger performance, while supporting more Deep Learning infrastructures and innovations.

FPGA solves your toughest Deep Learning processing challenges
Traditional central processing units (CPUs), while very flexible, fail to measure up in high-demand computing tasks of Deep Learning and inference. Many neural network developers quickly recognized this, and switched to graphics processing units (GPUs), which allow more computation bandwidth, at the cost of decreased flexibility.
This trade-off worked reasonably well at first – but it, too, didn’t last for long. As neural networks continued to increase in complexity, demanding ever more calculation bandwidth, even GPUs reached the limits of their abilities. GPUs also turned out to present other problems as well: they consume quite a bit of power and demand a lot of cooling, making them less-than-ideal for deployment in the data centers used in big-data cloud applications.

In fact, while custom ASICs may seem to present an ideal solution, they also create significant challenges for data centers. For one thing, ASICs are not yet available in any data center. If available, they must comply with data centers’ strict requirements in order to be deployed, which will be another challenge before large adoption. But mainly, they are highly dedicated to a specific task, which makes them even less flexible than CPU, GPU and FPGA.

Developing a specific ASIC takes several years – but the Deep Learning and neural network fields are evolving rapidly, with ongoing breakthroughs making last year’s technology irrelevant. Plus, in order to compete with a CPU or a GPU, an ASIC would need to use a large silicon area using with the thinner manufacturing technology. This makes the upfront investment expensive, without any guarantee of long-term relevancy. The neural network field evolves rapidly, and the acceleration provided by a given ASIC may not always be as powerful as that of an equivalent GPU, CPU or FPGA.

Many of these challenges have already been solved by FPGAs.

An FPGA is programmable at the hardware level, making it highly parallel, to a greater degree than a CPU or a GPU. FPGAs use less power than a GPU or CPU, and are proven to comply with data center requirements. Unlike a specialized ASIC, an FPGA is hardware programmable, permitting acceleration of many different loads, and enabling quick updates to accelerate new neural network architectures without having to wait for the next ASIC to be released. It’s true that, until very recently, FPGAs lacked high computation bandwidth – but that has changed. In 2017, the two major FPGA design companies are releasing the most powerful FPGAs so far, which offer computation bandwidth equivalent to that of a GPU, with the additional potential to use the millions of logic cells and flip-flops necessary to match the requirements of a neural network.

### Design and Methodology

**How we chose our FPGA**

Our choice of FPGA has been driven by our design goal: to create a neural network computation system that can run any neural network and reach the highest potential performance on a single FPGA.

Several research teams have concluded that high GPU precision is not required for high accuracy in neural network computation. In other words, it’s possible to obtain accurate results using fixed-point 16 bits, 8 bits, or even fewer. Thus, although we do support 32-bit floating-point calculation as a reference when comparing results to GPU, our focus has been to design a system that values performance over precision.
By the same token, we value fixed-point throughput over floating-point. While Intel’s Stratix-10 GX2800, for example, natively supports floating-point calculations in the silicon, the fixed-point Xilinx Ultra-Scale Plus VU13P actually offers more fixed-point calculations per cycle, thanks to a higher number of DSPs.

Of course, maximum frequency must also be considered, because higher frequency correlates with high performance. While some FPGAs may run faster than others, the constraint of fitting into a data center requires to tightly control the power then to limit the frequency. Since even a higher-frequency FPGA would also be subject to the same power constraints, frequency was not a differentiator in our choice of FPGA.

<table>
<thead>
<tr>
<th>FPGA</th>
<th>Intel Stratix-10 GX2800</th>
<th>Xilinx VU9P</th>
<th>Xilinx VU13P</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSP</td>
<td>5,760 (11,520 mult)</td>
<td>6,840</td>
<td>12,288</td>
</tr>
<tr>
<td>Boolean Logic</td>
<td>2,753k LE</td>
<td>2,586k LC</td>
<td>3,780k LC</td>
</tr>
<tr>
<td>Flip-Flops</td>
<td>3,732k</td>
<td>2,592k</td>
<td>3,456k</td>
</tr>
</tbody>
</table>

*Table 1: Logic available in newest FPGA*

Over time, the performance and functionality of Intel’s and Xilinx’s FPGAs will continue to improve, which may lead to a different choice. In all cases, we are convinced that next generations, with more DSP and logic, will offer even better solution for neural network in data centers. However, with our current considerations in mind, and current components available, we chose the Xilinx KU115 as the basis for developing a prototype of the system, and the Xilinx VU9P and VU13P as targets for our production FPGA.

**HLS vs. RTL, and OpenCL vs. PCIe**

For a while, Xilinx and Intel are pushing the use of high-level synthesis (HLS) to design FPGA content. One goal is to allow a wider audience to develop FPGA content, particularly software developers who can write C/C++ code. When the goal is to create a design utilizing a lot of arithmetic, HLS allows design at a higher level of abstraction than register-level transfer (RTL). This can reduce the coding effort. However, if the goal is to push global performance as high as possible, HLS may fall short. Using RTL to design, on the other hand, allows for finer control of the implementation result, at the potential cost of a longer design time. Since our goal is to reach the highest possible performance, we decided to use RTL coding.

OpenCL is a framework for writing programs that execute across heterogeneous platforms. On a typical PC, this allows programs to transfer data over a PCIe bus without going into low-level details – which means OpenCL enables heterogeneous platforms to get up and running more quickly, with less coding effort from the developer.

However, like any generic solution, OpenCL may lack the highest throughput desirable for certain specific solutions. Because we sought the highest possible performance, we decided to invest in writing our own SW/HW interface.

In short, no perfect solution exists – it’s all a matter of choosing the right trade-offs.
We can imagine that a larger developer community will drive HLS and OpenCL to close the performance gap eventually. For those who are new to FPGA and acceleration, or those who want to design quickly with fewer performance requirements, both these options are probably equally good choices.

We’ve already solved the architecture and performance challenges
A growing number of companies are adding FPGAs to their data centers, for the primary reason of increasing calculation throughput. This means computation performance is the most important driver in FPGA choice (typically followed by the drive to reduce power consumption). In certain applications, the latency offered by FPGA could also be a primary driver, like in fast-trading and more generally financial markets. But in most application, the computation acceleration justifies by itself the use of FPGA.

Thus, while it’s mandatory to utilize the highest possible number of digital signal processors (DSPs) at high frequency, it’s equally important to feed each DSP with data. In particular, the ways in which look up tables (LUT), flip-flops and memories are used around DSPs create significant impacts on eventual performance. These facts drove many of our architecture decisions. Google, in its recent article, provides some interesting related information. If some may see specific ASIC as superior for performance, the architectural choice also impacts them greatly. Google’s TPU, for example, has a low actual MAC utilization rate with an average of 23% (and as low as 6.3% on one of the neural network used to measure performance). This does not differ for FPGA: the architecture can drastically impact the real eventual performance.

Our first focus was to choose between internal and external memory. While internal memory offers a large amount of bandwidth, it has limited capacity. External memory, on the other hand, offers high capacity at the costs of lower limited bandwidth and high latency, while also consuming more power. Thus, we decided to avoid external memory as much as possible in our architecture.

Another challenge was to reduce the logic per DSP. There are thousands of DSPs in the FPGA, driven by data busses. If those busses are 32-bits wide, for example, every extra LUT (flip-flop) on them costs not one LUT (flip-flop) but thousands multiplied by 32. This significantly increases the size of the logic used.

For example, on a VU9P, the addition of a mux on each bit in front of each of the 6840 DSP costs roughly 18 percent of LUTs on the FPGA.

Thus, we defined a budget-per-bloc and per-DSP before we began coding. We also took advantage of the large set of arithmetic operations offered by the Xilinx DSP to improve the per-cycle performance, which a CPU or a GPU cannot offer. Aggregating the DSPs to construct larger operations, without involving extra logic, allowed us to also reduce the logic involved for larger operations.

When coding using RTL or C, it is important to understand the results at the gate level. One may think that the synthesizer will do one thing, but the tools will sometimes produce a different result. This may be due to incorrect syntax, a missing option, an area optimization reducing frequency, or the opposite. However, when one can check the synthesis result, it’s possible to compare to the budgeted size and make sure the design will fit the targeted FPGA with the highest filling rate for each DSP, in order to obtain maximum performance.
Our final focus was frequency. Utilizing all the DSP and most of the other resources of the FPGA leads to challenging placement and routing problems for Vivado. We believe that RTL coding is quite helpful to reach higher frequencies – which is why we didn’t neglect this crucial step toward reaching our performance goal.

Running your FPGA in the Cloud provides many advantages

Amazon Web Services (AWS) offers FPGA-based instances, named AWS Elastic Compute Cloud (EC2) F1 instances, enabling users to accelerate algorithms in several advantageous ways. Each EC2 F1 instance is like a virtual PC in the cloud equipped with 1 or 8 VU9P FPGAs.

From the end user’s point of view
For end users, who are not familiar with FPGAs, but want to take advantage of their benefits, AWS provides an easy solution of an FPGA instance in the Cloud, similar to using a GPU-based or a CPU-based instance. This offers many benefits:

- **Zero up-front cost.** AWS provides access to one of the largest FPGAs, without requiring the user to buy it before using it. The board was designed to accelerate computations, so the end-user does not have to mind about which board is adequate.

- **Hardware comes pre-installed and pre-tested.** AWS has already taken care of compatibility with the computer and OS, which can consume a lot of time and money when a PCIe board (FPGA or otherwise) is not compatible with a certain machine.

- **High hardware quality.** AWS monitors the hardware on an ongoing basis, eliminating the impact of a failing board remaining undetected for a long time. If a board fails, you can simply switch to another instance and restart your application without delay.

- **Adaptable bandwidth.** After starting with one FPGA, it’s easy to scale up the bandwidth without any up-front cost – or to scale down if required. For applications with only a few hours of high peak activity, or applications that only require a few hours on a regular basis, the average cost will be reduced by only using the instances when required. This means there’s no need to buy many boards to use them only sparsely.

- **Simplified usage.** Since many users are not FPGA specialists, and prefer to focus on their applications more than on the FPGA, AWS provides applications that make using an FPGA as simple as using a CPU.

From the startup’s point of view
For companies in the business of providing FPGA-based acceleration, using a FPGA in the cloud also provides some significant advantages:

- **No up-front hardware investment.** If the acceleration can be performed on a generic board, there’s no need to invest in physical hardware.
• **Consistent, uniform hardware.** Since the hardware installed on AWS is uniform, you’ll be using the same hardware for development, evaluation and execution. This limits compatibility issues and risks of operating in an unknown environment, while focusing your development on one specific platform.

• **Immediate testing and evaluation.** You won’t have to spend weeks, or even months, installing software for a potential customer to evaluate. An evaluation can start immediately on the Cloud, yet the customer can still control the pace of testing.

• **No lead time for additional bandwidth.** Whenever a customer needs more bandwidth, new FPGAs are immediately available in volume – with no lead time for hardware manufacture, and no need to call in a team to install boards (or maintain existing ones).

• **No need to deal with borders, customs and shipping companies.** In fact, no need to ship hardware at all.

• **Unlimited number of customers.** All can share the same hardware, without a dedicated support team to look after them.

• **Automatic upgrades and deployments.** AWS’s own infrastructure manages every application upgrade, deployment and distribution.

• **Much broader customer reach.** Every customer can simply use the application on AWS, rather than having to order custom hardware and/or install custom software.

However, while those benefits are significant, it does take some work to integrate AWS requirements into the development process.

**We’ve already ported Zebra to EC2 F1**

Using OpenCL can significantly reduce the effort of porting to a cloud platform, because OpenCL is designed to be portable between platforms, as long as the API is used correctly.

Whether you choose to use OpenCL or not, the principle is the same for the FPGA integration: AWS’s Hardware Design Kit provides a shell for the FPGA. This shell is the layer around your IP, providing access to PCIe and DDR memories. While the shell runs permanently on the EC2 F1 instance, the application the user runs is loaded using partial reconfiguration of the FPGA. Since we’ve already tested essential blocks and made them ready for use, running a neural network on Zebra essentially functions like running a program on a CPU, using only about 20 percent of the FPGA’s resources.
Porting our application to the EC2 F1 board turned out not to be too complex. We started from a design running on a KU115 coupled to a SW stack under Linux, and ported them to the VU9P available on EC2 F1 instances with our software stack, within four weeks. Adapting the design to a different FPGA was made easy as the architecture is scalable (see Figure 2). Most of the work was focused on bridging our control bus to the shell, and making sure that routing and timing would be achievable.

Then, we spent another four weeks optimizing the FPGA compiles to reach higher performances. We performed the software porting in parallel without major issue.

Although we began with no knowledge of how to use AWS, we found it to be very easy to use, and found the AWS team to be very reactive and knowledgeable. They automated the process of building the Amazon FPGA Image (AFI) which is required to run on the EC2 F1. This AFI cannot be accessed directly by the F1 instance user, allowing us to protect our IP, which – along with the fact that no one else offers cloud access to FPGA as AWS does – was one of our most important criteria in choosing AWS.

Next, we actually ran our design and debug it. For newbies – who most likely use HLS design – this can be a shock. The High-Level Synthesis tool significantly transforms the source code, resulting in a low-level gate netlist. Nothing looks like software debug anymore. Instead, it is millions of gates executed in parallel, which are typically debugged with waveforms.

**Benefits of running Mipsology Zebra on FPGAs in your data center**

Every Cloud-based FPGA designed by Mipsology processes over 24,000 operations at a time. It works with all neural networks, boasting compatibility with any typical CNN or RNN. It requires, from the user, no FPGA knowledge, no FPGA tool knowledge, no hardware knowledge; in fact, the user can just forget there is an FPGA. It runs on “standard” PCIe board ready to be used in data centers. It’s also low power, consuming 6 times less power than an equivalent GPU.

Plus, since Mipsology provides FPGAs as Software as a Service (SaaS) for data centers, you’ll save significant overhead on servers, software licenses and staff. All you’ll have to do is to connect to the Cloud and start running your neural networks.
Data Center Benefits

Mipsology Zebra runs in high density in data centers, without consuming excess power. The hardware works seamlessly with the data center’s loads, and complies with all the usual HVAC requirements.

Neural Network Technology Benefits

Mipsology Zebra doesn’t require any FPGA-related knowledge from the end user. It is already integrated with deep learning infrastructures, which means it can simply replace your existing GPU without any changes. Mipsology Zebra will also support new neural network technologies more quickly than a GPU as the same silicon can be reprogrammed at the hardware level to accommodate new needs, no re-spin, no new silicon, no tape-in tape-out delays, acceleration can be made available in few weeks.

Performance and Power Benefits

Mipsology Zebra boast higher performance than equivalent CPUs or GPUs. It will pass 100 images/s/W in 2017 in comparison to only 14.2 images/s/W from the newest GPU. As the power used is lower than GPU, having 8 boards per host is simple, allowing to increase the density of computation in data center with hundreds thousand images per host within a slim power budget below 700W. They support various precisions, and can even handle other processing demands on the same FPGA.

Cost Benefits Using AWS and Zebra

Mipsology Zebra runs on AWS EC2 F1 which is available with no upfront cost to the user. As switching to Zebra can be done in minutes, there is no engineering cost related to changing the neural networks or the code in place for GPU or CPU. The billing being done per hour, the hardware and software costs can be adapted to the real loads and needs, so no hardware is sitting in a computer doing nothing. No need to mind about availability, hardware compatibility, hardware support and failures, depreciation, deployment, IT costs, all that is made transparent and managed by AWS. You’ll be able to use the same hardware for inference and learning, all for a smaller R&D budget than ever before.

Mipsology stands at the forefront of FPGA-based machine learning. In terms of cost, power, flexibility and adaptability, the answer to your Deep Learning challenges is clear: Mipsology Zebra.