Unified Deep Learning with CPU, GPU, and FPGA Technologies

Allen Rush[1], Ashish Sirasao[2], Mike Ignatowski[1]

1: Advanced Micro Devices, Inc., 2: Xilinx, Inc.

Abstract

Deep learning and complex machine learning has quickly become one of the most important computationally intensive applications for a wide variety of fields. The combination of large data sets, high-performance computational capabilities, and evolving and improving algorithms has enabled many successful applications which were previously difficult or impossible to consider.

This paper explores the challenges of deep learning training and inference, and discusses the benefits of a comprehensive approach for combining CPU, GPU, FPGA technologies, along with the appropriate software frameworks in a unified deep learning architecture. Each of these hardware technologies offers unique benefits to the deep learning problem, and a properly designed system can take advantage of this combination. Moreover, the combination can provide unique capabilities that result in higher performance, better efficiency, greater flexibility, and a hedge against algorithm obsolescence compared to CPU/GPU and FPGA systems designed separately.

Aside from the underlying hardware approaches, a unified software environment is necessary to provide a clean interface to the application layer. This needs to account for several factors, including framework support, different compiler and code generator technologies, and optimization support for the underlying hardware engines. Higher-level frameworks (e.g., TensorFlow, Theano) can effectively hide most heterogeneity from application developers as well as enable portability across different systems. This is a powerful enabler for heterogeneous hardware. For application developers working below the framework level, the AMD ROCm and MIopen software frameworks are discussed as an example of a unified software environment applicable to a CPU and GPU solution. FPGAs are primarily used for inference, and the xfDNN middleware from Xilinx captures the software features essential for implementing deep learning inference on FPGAs.

A long-term vision for application developers is a full and seamless programing environment that works across CPUs, GPUs, and FPGAs. This could initially focus on support for a common language and runtime, such as OpenCL, and later be extended to additional languages. The language support would hide any internal differences in compilers and runtimes between the CPU, GPU, and FPGA implementations. This seamless programming environment will facilitate the full end-to-end optimization of resource allocation.

Background

Deep learning has emerged as the most effective method for learning and discerning classification of objects, speech, and other types of information resolution. A brief review of deep learning is useful, although there are many good references that cover the historical and state-of-the-art technology. The main purpose here is to illustrate the compute and data management challenges that exist as a result of implementing successful deep learning systems.

Some basic terms commonly used in this field are defined below:

Machine Intelligence (MI): A program that can respond to inputs, analyze data, recognize patterns, or develop strategies, in ways that would be considered as displaying intelligence when done by humans. For our purposes, we consider machine intelligence (MI) and artificial intelligence (AI) to be interchangeable terms.

Machine Learning (ML): A subset of machine intelligence algorithms that improve their performance over time, typically when they are exposed to more data.

Neural Network (NN): A data structure that represents artificial neurons interconnected by artificial synapses having various weights or strengths, similar to biological neural networks present in the brain. A neural network is "trained" by adjusting the weights of the various artificial synapses so that the network produces a desired output for various input data.

Deep Neural Networks (DNN): Multilayered neural networks with a large number of hidden layers used by deep learning algorithms. Two commonly used DNN variations are convolutional neural networks (CNN) and recurrent neural networks (RNN). Typically, CNNs are used for image processing related tasks whereas RNNs are used for speech and natural language processing tasks.

Deep Learning (DL): Machine learning algorithms with multilayered neural networks that learn from exposure to vast amounts of data.

DL Training: Using a set of training sample data to determine the optimal weights of the artificial neurons in a DNN. Modern DL models use a deep network with hidden layers and a process called *stochastic gradient descent* to train the network. To achieve acceptable accuracy, many training samples are needed in the training process. DL training times can range from days to weeks for large training sets involving millions of data samples.

DL Inference: Analyzing specific data using a previously trained DNN. For most applications, the latency for inference operations is typically less that a second. DNNs trained for image classification have improved to the point that they are virtually identical to (or in some cases exceed) human classification accuracy.

Software for Deep Learning

Machine learning applications are typically built using a collection of tools. Deep learning *frameworks* are sets of software libraries that implement the common training and inference operations. Examples of these include Caffe/Caffe-2 (Facebook), TensorFlow (Google), Torch, PyTorch, and MxNet (used by Amazon). All of these are available as open source software. Such frameworks are supported across both GPUs and FPGAs, and can effectively hide most heterogeneity from application developers as well as enable portability across different systems.

DNN training is very computationally intensive and typically makes use of common floating-point computation functions such as Basic Linear Algebra Subprograms (BLAS), GEMV and GEMM routines, Conv2D operations, batch normalization, SGD, SVD, element wise matrix operations, and non-linear functions such as Softmax and ReLU. Availability of these building blocks along with an efficient data transfer mechanism between GPU and CPU is essential for the efficient implementation of these algorithms. Training is typically done on a combination of GPUs and CPUs using single-precision floating-point arithmetic (fp32), though the trend is to move towards half-precision (fp16) floating-point arithmetic. Recently, frameworks like MxNet allow scalable parallelization of training tasks using multiple CPUs and GPUs.

Middleware libraries provide operations to the frameworks that are tuned to specific hardware for maximum performance. MIOpen from AMD support the Radeon Instinct line of AMD GPUs. cuDNN from NVIDIA supports similar functions on NVIDIA GPUs. BLAS, FFT, and Random Number Generator (RNG) are a set of standard math library routines. NVIDIA Collective Communications Library (NCCL) implements multi-GPU and multi-node collective communication primitives.

AMD has recently announced the ROCm platform for GPUs. The ROCm platform includes system runtime functions and drivers, compilers, and porting software for CUDA programs to AMD's Radeon line of GPUs. ROCm supports the common ML frameworks such as Caffe/Caffe-2, TensorFlow, Torch, and MxNet to provide a "full-stack" solution. ROCm is also compliant with the Heterogeneous Systems Architecture (HSA) open standard, and scales to multi-GPU configurations.

The complete machine learning software stack, featuring AMD's ROCm and MIopen, is shown below.

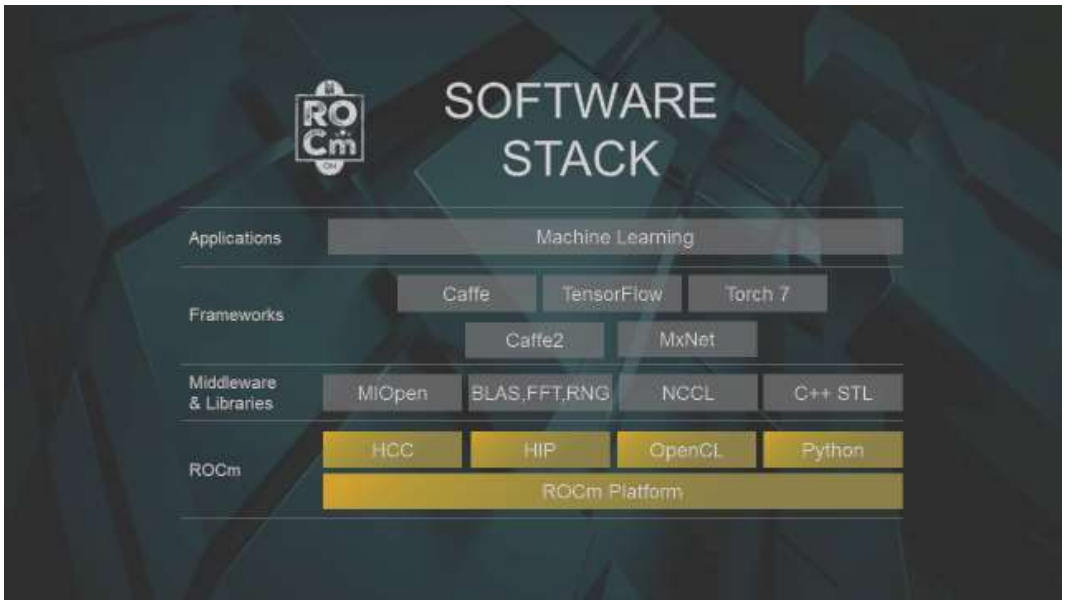

Figure 1: ROCm Software Stack including MIOpen and common MI frameworks

Inference implementation on FPGAs typically uses fused mode of operations, where most of the network is implemented using large and flexible on-chip memory and flexible arithmetic capabilities which are unique to Xilinx FPGA. Xilinx and its partners have implemented optimized overlay engines for implementing CNN and RNN engines. xfDNN from Xilinx provides a software middleware which runtime,

compiler, and a quantizer to map and run DNN models trained on GPUs and CPUs onto optimized overlay engines for CNN and RNN processing. Xilinx also provides Reconfigurable Acceleration Stack, which consists of the pre-built hardware platform on FPGA, a runtime and design tools to enable development of a broad set of customized deep learning solutions.

A long-term vision for application developers is a full and seamless programing environment that works across CPUs, GPUs, and FPGAs. This could initially focus on support for a common language and runtime, such as OpenCL, and later be extended to additional languages. The language support could hide any internal differences in compilers and runtimes between the CPU, GPU, and FPGA implementations.

Moreover, this seamless programming environment will facilitate the full end-to-end optimization of resource allocation to take maximum advantage of the collection of available compute and memory capability of the system.

DNN Implementation and Optimization

Training makes heavy use of floating-point operations for most applications, but for inference operations many network weights can be represented with considerably less precision than their original training output. Recent results show very good results at 8 bits of precision, and in some cases even binary networks can give desired accuracy.

In cases where network weights are zero or near zero, they can be eliminated, or "pruned," from the graph. Static reduction of weights can account for a modest reduction in weights, while dynamic pruning (done during the inference operation) can achieve considerably higher levels of pruning. Such pruning can considerably reduce the computation requirements for inference operations.

Once a model for deep learning is trained, deploying it for inference in deep learning applications typically has additional requirements. While the compute demand for inference is less in terms of complexity, training data, and long training times, there are several key requirements for inference. In cloud and server systems, inference requirements fall into two categories: low latency single batch, and very large batch high throughput. Inference solutions focus on efficiency of implementation in terms of these requirements. Historically, these tasks have been implemented on CPUs, but GPUs are increasingly being used, and more recently FPGAs have emerged as an efficient solution for inference. FPGAs take advantage of their flexibility to implement lower precision arithmetic, pruned networks, local SRAM, and a wide variety of DRAM standards to provide broad performance, latency, and power envelopes. Inference solutions implemented on FPGAs can be deployed in a datacenter environment as well as in embedded environments like surveillance, self-driving cars, IoT edge servers, etc.

Increasing Computational Challenges

There are a number of standard DNN configurations, each differing by the number of layers and interconnect patterns between layers. The total number of layers has been increasing over time at a rate faster than the "Moore's Law growth rate" as illustrated in the table below. Between 2012 and 2016, the number of layers in common DNNs have grown by over 2.3x per year. There have been recent discussions of extending the number of layers in future versions of ResNet to 1,000.

DNN training is achieved by fitting the parameters (or weights) using a method called stochastic gradient descent and other optimization techniques to arrive at parameter updates. Iterative updates are required to improve the model accuracy with a small incremental improvement rate. Training with large sample sizes such as ImageNet (containing over 14 million images) and a model such as Resnet can take ~30-40K iterations to converge to a stable solution. As the DNN complexity and the size of the training set both increase, the total computational load, as a high-level estimation, can exceed 10^{20} FLOPS.

As early as 2008, GPUs were used to speed up the training to an acceptable level. Even with GPU support, it is not unusual for a training session to take days or weeks. There are many optimization strategies being developed to reduce the long learning time. However, as DNNs continue to increase in size and complexity, the computational requirements are expected to grow.

Network	Application	Layers
LeNet-5 for MNIT (1998)	Handwritten Digital Recognition	7
AlexNet (2012)	ImageNet	8
DeepFace (2014)	Face recognition	7
VGG-19 (2014)	ImageNet	19
GoogLeNet (2015)	ImageNet	22
ResNet (2015)	ImageNet	152
Inception-ResNet (2016)	ImageNet	246

Figure 2 Training Layers Increasing Over Time

In addition to the computational load, the memory capacity and bandwidth are a heavy influence on the overall performance of training, and to a lesser degree, inference. The computation for many network layers has a high compute-to-bandwidth ratio; there is a large amount of data reuse, and the parameters are shared across each layer of input data. Conversely, the more fully connected network layers have very low compute-to-bandwidth ratios; the data reuse is very low and there are no shared parameters.

As GPU processing has improved to address the compute challenges for these networks, many algorithms or models have become bandwidth bound, and the peak performance of the GPU (or CPU) is not realized. In some worst-case conditions the efficiency of the GPU is in the range of 15-20% of peak theoretical performance. This need for increased memory bandwidth being addressed with the use of high bandwidth HBM stacked memory as discussed in the GPU section below.

Computational Approaches – CPU, GPU, and FPGA

CPU Solutions

High performance CPUs nodes typically support large memory and storage capacities and robust network connectivity. They are more general purpose than GPUs or FPGAs but fail to match them in raw compute capabilities. However, they are typically the best solution for MI applications involving large scale data analytics where large memory capacity and network capabilities are required.

GPU Solutions

GPUs are very efficient for many of the underlying DL primitives involving highly parallel compute operations: convolutions, matrix multiply, and activation functions. With the addition of HBM stacked memory to the latest GPU models, the bandwidth is improved to the point where many of the primitives can effectively exploit all available GPU compute resources. For dense linear algebra operations, the performance improvement of GPUs over CPUs is typically in the range of 10-20:1.

The primary programming model for GPUs is based on maximizing parallel processing. An example GPU design may contain up to 64 compute units, each with 4 SIMD engines, with 16 lanes of floating point compute per SIMD. When the utilization approaches 100%, the peak performance is over 10 TFLOPS (single precision fp32) and 25 TFLOPS (fp16). Additional GPU performance could be realized when inner product instructions combine multiply and addition functions for vectors to match primitives for matrix operations.

While GPUs are typically thought of as optimal designs for DNN training, they can also provide substantial performance for inference operations.

FPGA Solutions

FPGAs are often used to implement inference accelerators. They can be efficiently configured to minimize the overhead or unnecessary functions that are included in GPU devices. However, they are limited to either integer inference or lower performance floating point compared to GPUs. The key attribute of FPGAs is the ability to configure the array with efficient design with little or no overhead, and the ability to dynamically (at run time) re-configure the array characteristics.

It has been demonstrated that FPGAs can provide both latency and performance per watt advantage over CPUs and GPUs in deep learning inference operations. FPGAs achieve this efficiency by enabling implementation of deep learning algorithms using reduced arithmetic precision, pruned networks, and custom high-performance hardware implementation. Recent implementation from Xilinx and partners has demonstrated that FPGAs can be used to implement CNN overlay engines with a peak performance of over 15 TOPs at the 8-bit accuracy with efficiency as high as 80% for some of the common convolutional neural networks. Pruning techniques used mainly in the context of LSTM and MLP neural processing has demonstrated that the model sizes can be effectively reduced by up to 20x providing a significant advantage in implementing an optimized solution. New research in the area of custom floating-point implementation and fixed-point precision lower than 8 bits is very promising and expects to provide further improvements to peak performance of FPGA implementation of DNN models.

Comparison

To understand the tradeoffs in determining the best way to configure a system using either CPU, GPU, or FPGA devices, Figure 3 shows a comparison chart for various attributes for the devices. An important observation is that there are complementary strengths and weaknesses; there are no obvious “one size fits all” solutions.

Feature	Analysis	Winner
DNN Training	GPU floating point capabilities are greater	GPU
DNN Inference	FPGA can be customized, and has lower latency	FPGA
Large data analysis	CPUs support largest memory and storage capacities. FPGAs are good for inline processing.	CPU/FPGA
Timing latency	Algorithms implemented on FPGAs provide deterministic timing, can be an order of magnitude faster than GPUs	FPGA
Processing/Watt	Customized designs can be optimal	FPGA
Processing/$$	GPUs win because of large processing capabilities. FPGA configurability enables use in a broader acceleration space.	GPU/FPGA
Interfaces	FPGA can implement many different interfaces	FPGA
Backward compatibility	CPUs have more stable architecture than GPUs. Migrating RTL to new FPGAs requires some work.	CPU
Ease of change	CPUs and GPUs provide an easier path to changes to application functionality.	GPU/CPU
Customization	FPGAs provide broader flexibility	FPGA
Size	CPU and FPGA's lower power consumptions leads to smaller volume solutions	CPU/FPGA
Development	CPUs are easier to program than GPUs, both easier than FPGA	CPU

Figure 3 Summary of CPU, GPU, and FPGA comparison

Scale-out for Training and Inference

One of the key challenges for increasing large scale performance is leveraging high-performance nodes, either with GPU or FPGA, to large network configurations. In general, two types of parallelism are considered: model parallelism and data parallelism. Model parallelism is implemented by coordinating the work of a DNN model spread across multiple hardware nodes, jointly operating on the same batch of data. A high degree of communication and synchronization among nodes is necessary, especially during training, to keep the information in each node consistent.

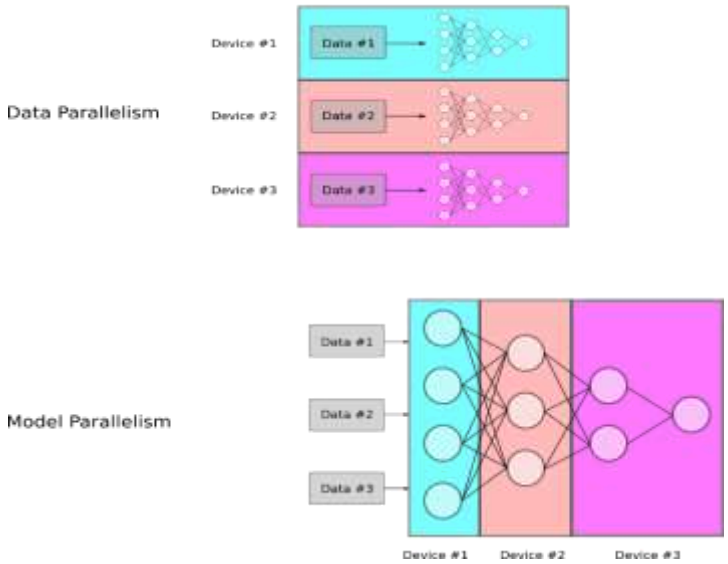

Figure 4 Model Parallelism vs Data Parallelism

Conversely, data parallelism is implemented by assigning a batch of data to each node, where a node implements that entire DNN model, end-to-end.

Within a single hardware node, scaling to multiple GPUs can be accomplished using a combination of PCIe links or custom fabrics. One such configuration is shown below using AMD Infinity Fabric™ links to interconnect multiple GPUs. NVIDIA GPUs can use their custom NVLink™ interconnect for similar configurations. Figure 5 shows a typical configuration of 4:1 GPU to CPU.

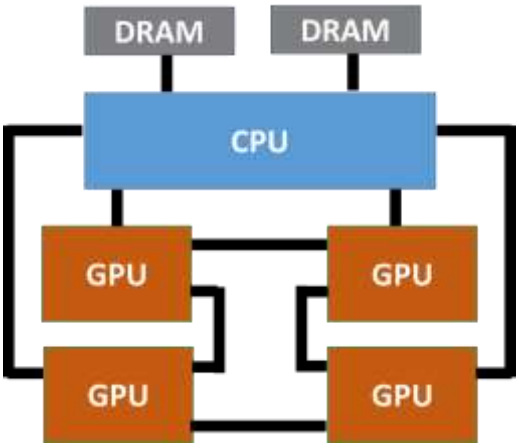

Figure 5 GPU Scale out using Vega20 and Epyc [Note: this image will be replaced in the final version]

What about ASICS? It is generally possible to take custom designs originally developed for an FPGA and implement them in an ASIC chip for improved performance and power efficiency. Google's Tensor Processing Unit (TPU) is an example of an ASIC custom designed for DL processing. For the purposes of this white paper, whether an accelerator for DL is implemented in an FPGA or ASIC has little impact to the system hardware and software architecture issues for the heterogeneous DL systems being discussed here.

Unified Deep Learning Configurations and Emerging Applications

Previous sections have described the complementary strengths of CPUs, GPUs, and FPGAs for different types of deep learning operations. With the emergence of new use cases, there will be a growing benefit of unified deep learning configurations combining all three types of hardware elements into single hardware nodes. This is illustrated with a few specific examples.

Training and inference are typically thought of as very distinct operations today, but there are emerging applications that will combine them. Continuous and reinforcement training can be used when a deep learning system is deployed in an environment where regular retraining and updating of a DNN is required. For example, systems that control devices in real time can be continuously learning and adapting based on the results of their previous actions. Similarly, systems that interact with humans can use feedback from humans to continuously improve their training. A unified deep learning platform that simultaneously employs GPUs for training updates and FPGAs for inference enables efficient implementation of such continuous training systems.

Simulations can also be used for training. There are many scenarios in the field of autonomous driving, surveillance, and other areas where large amounts of training data may not be available. Creating large datasets with labeled images or scene-specific pedestrian detectors is extremely costly. This also applies to designing deep learning based controllers for lane merging, and other situations. To circumvent this problem, modern computer games are being used to provide training data by simulating real life situations. A unified system with very robust hardware capabilities is needed for such combined simulation / deep learning systems.

Many systems can have serious negative consequences for certain types of wrong predictions. One example is an autonomous vehicle that fails to correctly identify the presence of a stop sign. There are cases where such failures occur when stop signs are slightly modified, such as when a small sticker is placed on them. One approach to reducing such errors is to combine multiple types of analysis: multiple DNNs independently trained with different data, plus the use of a digital map of all known stop signs in the area, plus an expert system that predicts circumstances where a stop sign is likely to be present. A supervising routine would evaluate the input from all these sources to make a more reliable determination about the presence of a stop sign. This type of hierarchy of heterogeneous machine intelligent systems mimics how many human decisions are made, and is expected to become more common over time.

Another active area for developing machine intelligence applications combines deep learning and data analytics on very large-scale datasets or real-time data streams.

These types of systems combining DNNs with other operations require the variety of rapid compute capabilities that a combination of CPU, GPU, and FPGA functions provides.

A heterogeneous configuration of components within a node can also optimize functions other than machine intelligence, as shown in the example in Figure 6. The FPGA in this configuration is used to implement a highly optimized intelligent interconnect between the server blades. It is easy to envision adding GPUs to such a blade, with possible additional support in the FPGA for machine intelligence related communication and collective functions, resulting in a highly optimized heterogeneous server for large datacenter machine intelligence operations.

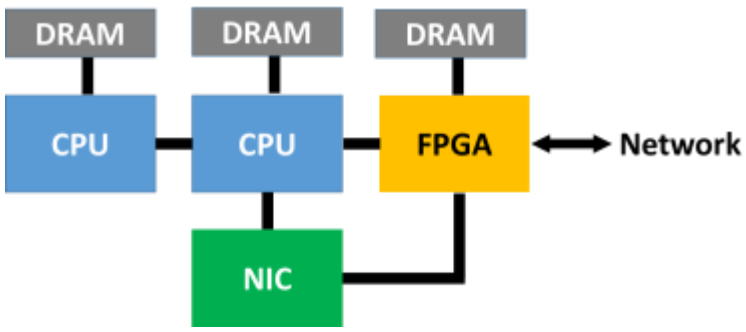

Figure 6 FPGA Exploitation

The CPU, GPU, and FPGA coupling provides for a software selection of a la carte solutions that can be optimized in a wide variety of ways that are hidden from the user. A unified software stack is not only elegant but provides for a maximum efficiency management of system resources. This also applies to

subsequent hardening of FPGA configurations in ASICS or other forms of efficient hardware embedded IP. It is also worth repeating that higher-level frameworks (e.g., TensorFlow, Theano) can effectively hide the heterogeneity from application developers as well as facilitate portability across different systems. This is a powerful enabler for heterogeneous hardware.

Summary

It is clear the use of deep learning applications is expanding. The combination of very large data sets, robust algorithms, and high-performance compute platforms enables a large variety of applications for commercial deployment. The compute requirements are substantial and growing for both training and inference. As a result, unified deep learning configurations combining CPU, GPU, and FPGA products are emerging as powerful solutions that enable the compute capabilities and flexibility that are needed to address these challenges.

Moreover, with a rich collection of optimization libraries that target all three compute elements, a heterogeneous system can be optimized with a hybrid of resources and application requirements that can adapt at run time. The ROCm software stack and MIOpen are excellent building blocks for this. They also support a growing community of open architecture application development that includes these different compute solutions.

Xilinx provides Reconfigurable Acceleration Stack, which consists of the pre-built hardware platform on FPGA, a runtime and design tools to enable development of a broad set of customized deep learning solutions. xfDNN middleware from Xilinx captures the software features essential for implementing deep learning inference on FPGAs.

We know that the solution landscape is evolving rapidly, and that flexibility is the key for maintaining relevance. A unified CPU, GPU, and FPGA solution provides a hedge against algorithm obsolescence and assures the best framework for future proofing.

The combination of high performance CPUs, GPUs, and FPGAs with heterogeneous support in the software stack results in a total system package that is greater than the sum of its parts. It is a solution in which emerging deep learning applications and system designs can be balanced in terms of flexibility and performance.