MOOR INSIGHTS & STRATEGY

WAVE COMPUTING: DESIGNED TO SCALE

WAVE'S AI SYSTEM IS NOW IN INITIAL TESTING AT CUSTOMER SITES

INTRODUCTION

Modern data scientists have an insatiable appetite for more performance to train and run deep neural networks (DNNs) for artificial intelligence (AI). In fact, research by Open.ai[1] has shown DNNs are doubling their performance requirements every three and a half months compared to the traditional Moore's Law rate for central processing units (CPUs), which have historically doubled every 18 months. While NVIDIA graphics processing units (GPUs) have largely enabled this advancement, some wonder if a new, grounds-up approach to silicon and system design might be better suited for this task. Given the growth prospects for AI, it's no surprise there are scores of startups and large companies like Intel readying new silicon to enter the race. Wave Computing ("Wave") believes their early time to market and novel "data flow" architecture will pave their way to success. In particular, Wave's system design has the potential to improve scalability, which is essential for large model training for AI. This article will look at Wave's architectural foundation for performance and scalability.

Wave Computing, which recently acquired MIPS Technologies for CPU IP, has begun initial testing of their first-generation custom systems at end users' sites for evaluation and feedback. While it is too soon to learn any results from these early experiments, the company's implementation of a dataflow architecture seems to present an elegant alternative to train and process DNNs for AI, especially when models require a high degree of scaling across multiple processing nodes. Instead of building fast parallel processors to act as an offload math acceleration engine for CPUs, this dataflow machine directly processes the flow of data of the DNN itself. The CPU is only used as a pre-processor to initiate runtime, not as a workload scheduler, parameter server, or code/data organizer. But, the real potential that we believe may have excited Wave's investors and prospective customers is the architecture's apparent ability to scale "super-linearly" for large DNN training and inference jobs.

A DATAFLOW PRIMER

Computer scientists use "dataflow" to refer to a model that represents a program as a graph of data flowing between operations. Data is processed when it "shows up" at the

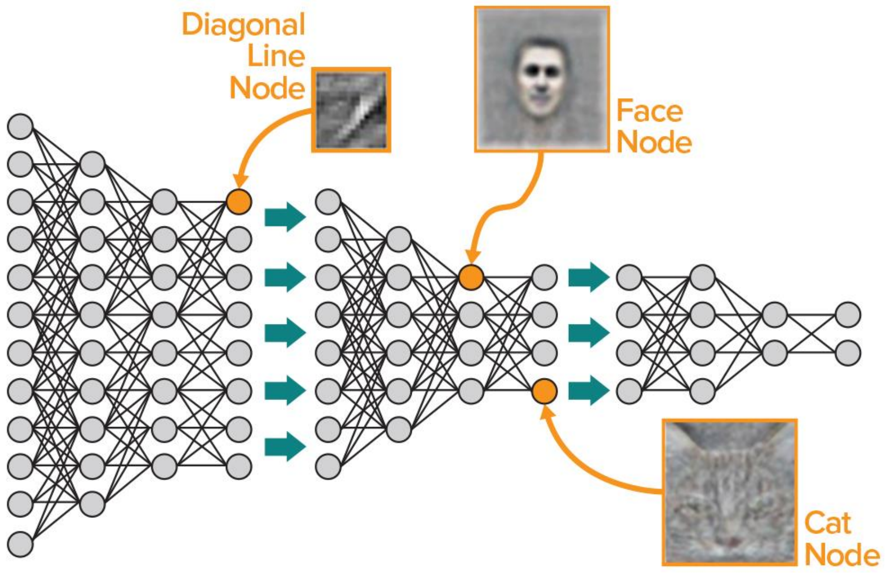
[1] https://blog.openai.com/ai-and-compute/

next node of the directed graph, leaving that processor idle until it receives a "fire" signal that data is ready.

Dataflow processing has been around for decades as a useful concept, but Wave appears to be one of the first companies to figure out how to build one that works, using a hybrid approach that combines standard instruction principles with dataflow concepts. The need for exponentially-increasing data distribution across the graph and activation signaling were inhibitors that stymied earlier attempts, such as those by MIT in the 1970s. The data flows asynchronously, enabling very fast and effective clock rates, initially up to 10 gigahertz with Wave's first chip. The company believes the typical throughput will be approximately two-thirds this peak rate.

What strikes many when they look at Wave's approach is that their dataflow machine looks exactly like the dataflow graph of a DNN. Such a system's architecture can natively process a dataflow graph such as a DNN. But, where does the data reside? How do you trigger a processing element to act? How do you manage memory and graphs larger than the physical machine? Where does a processor get and store its code? The dataflow approach seems to align elegantly with the DNNs it is being designed to process.

FIGURE 1: A TYPICAL NEURAL NETWORK FOR DEEP LEARNING



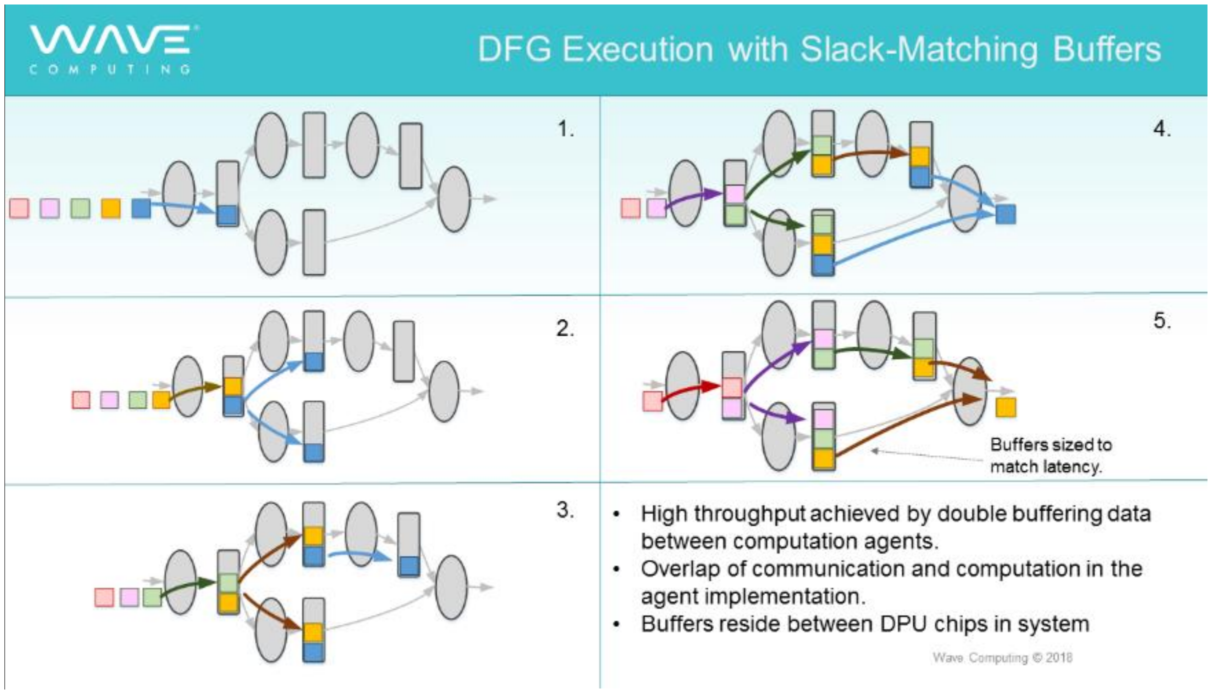
A neural network is usually represented as a dataflow graph showing how inputs combine with weights to determine the outcome—in this case, whether an image is a face or a cat.

Source: Wave Computing

Beyond Dataflow: System-level Scalability

Wave believes their novel architectural innovations will help them outperform their competitors for building large scale DNNs that typically require scores or hundreds of accelerators (GPU's or application specific integrated circuit or ASICs). Wave's approach is holistic, using highly parallel processors called dataflow processing units (DPUs) and a systems architecture (memory, interconnects, and software) that ensure these DPUs are kept busy to maximize performance at high node counts. Each DPU provides parallel processing across 1,024 clusters and each cluster has 16 processing elements. But, tying these processors together into a dataflow *system* is what may make Wave special. Three innovations combine to enable a high degree of scaling in a Wave system, according to Chris Nicol, CTO, at Wave.

Figure 2: Slack-matching Buffers



DFG Execution with Slack-Matching Buffers

- High throughput achieved by double buffering data between computation agents.
- Overlap of communication and computation in the agent implementation.
- Buffers reside between DPU chips in system

Wave uses "slack-matching buffers" to improve computation utilization in building (training) and processing (inference) DNNs.

Source: Wave Computing

Slack-matching Buffers

One of the challenges for dataflow processing is that the uneven nature of many graphs can lead to performance issues – as the processing of one flow is suspended in order to synchronize with the flow of an impending merging flow. Wave addresses this challenge by optimizing the size of buffers residing in the multi-ported hybrid memory cubes. The processing of the lower branch can continue as the processing of the upper branch in Figure 2 continues. The lower branch buffers its output for the final processing agent to begin once the data from the upper branch is ready to fire to node where the two sub-graphs merge on the right. Since the dataflow graph is static, the optimization of the buffer sizes can be configured a priori by the dataflow graph session manager.

Efficient Execution of Model and Data Parallelism

Efficient model development (training) requires both model and data parallelism. Tim Dettmers, a computer science doctoral candidate at the University of Washington states: "*Data parallelism is when you use the same model for every thread, but feed it with different parts of the data; model parallelism is when you use the same data for every thread, but split the model among threads*." In practice, data parallelism speeds parallel computation of parts of an image, for example. Model parallelism speeds computation of different parts of a model across a (large) number of processors.

Data Parallelism

Data parallelism is typically implemented using a CPU as a parameter or central repository of the model parameters. The model and parameters are sent to each of the multiple accelerators (GPU's or ASICs) and different minibatches (small subsets of the training data). The accelerator then calculates the gradients using nonlinear transformation functions such as TanH or rectified linear units for the parameters based on the tagged data in its minibatch. The gradients are transmitted back to the parameter server on the CPU where they are averaged and the parameters are updated according to all of the minibatches that were sent to the accelerators. The updated parameters are then redistributed across the network of accelerators, along with the next minibatches. This is data parallelism using spatial separation across accelerators attached to a CPU.

In a data flow processing unit, the parameters are updated locally in the DPU. The minibatches are entered into the data flow graph in series. The "mean" block in the DPU performs an average calculation based on the gradients calculated in time (like a pipeline). The averaged gradients are then sent to the variable update agent that updates the parameters with the average of the gradients. This is performing the same operation as with the distributed accelerators, but the minibatches are pipelined through the graph and so there is no communication to the parameter server. Once again, the CPU is not relevant and data parallelism is implemented in a DPU without any CPU intervention.

Model Parallelism

Model parallelism is used in cases where the model does not fit onto a single accelerator. When using GPU or ASIC accelerators, the model is distributed across multiple compute nodes. There are various ways to partition the model across the nodes to minimize inter-node traffic. When combined with data parallelism, a parameter server will send distributed copies of the model parameters (with separate minibatches of training data) to servers containing multiple accelerators. These servers calculate the loss and new sets of gradients (while sharing large amounts of data between them) and then send the gradients back to the parameter server. Clearly, this approach gives rise to scalability issues, both within a server and in the need for a central repository of the network parameters.

In a DPU system, the layers of the network are distributed across multiple DPU chips. The updating of the parameters in the network are also distributed. The graph can be arbitrarily distributed across as many DPUs as can be connected together in a system. The inter-agent buffers that are used between the DPU chips by the session manager provide an overlap across chip boundaries. The network latency between the chips can even be exploited by the session manager when implementing the slack-matching algorithm mentioned earlier.

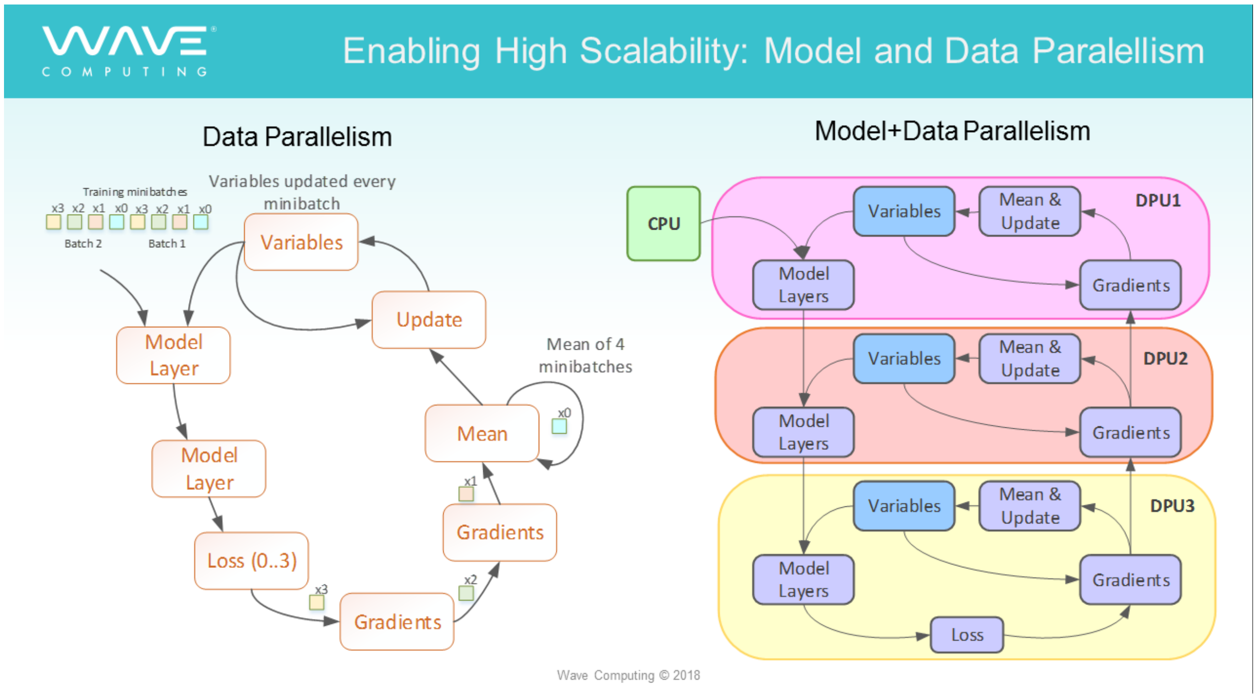
Data parallelism in such a system is still achieved by sequencing minibatches into the graph as before and the updating of the parameters in the graph will occur independently and in a distributed manner. In fact, there is no reason for sections of the graph to all be processing the same batch at the same time. The pipelining in the system enables autonomous and distributed training of the graph in a correct manner.

How does one verify such a system? Wave has developed a mechanism to confirm network parameters without stalling the execution of the graph. The extracted parameter values will be synchronized to a batch boundary. The host can load new parameter values using the same mechanism.

When combined with other features supporting scalability, Dr. Nicol believes that the system will improve performance at least linearly, if not better, as one scales the system out with a large number of nodes across inter-chassis and rack networks.

Once the system is processing with data- and model- parallelism and has the buffering in place to keep them busy, the system has to move processing through the data to execute the mathematical transformations associated with each node and layer. Wave has created an efficient model whereby a signal manager activates a page of agents (called an "agent overlay") from a DPU's local DDR memory, overlaying and replacing the previously active agent on that DPU. There are multiple pages of agent overlays in each DPU to help keep the DPU busy. Each agent represents a set of distinct mathematical operations that are used to calculate the operation of a graph node in a DPU before sending the data to the next processor/node in the graph.

FIGURE 3: DISTRIBUTED AGENT MANAGEMENT



Source: Wave Computing

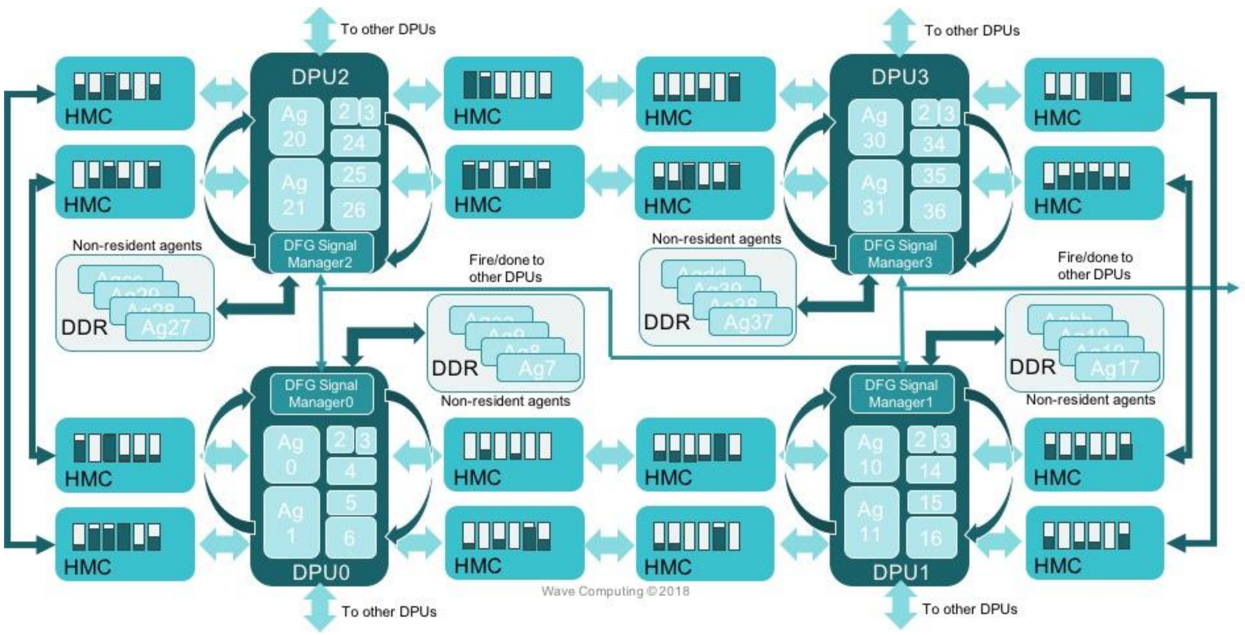
A distributed signal manager, which is a dataflow process, keeps track of the state of each of the agents and their queues of impending processing. If an active agent's queue drops below a threshold and an inactive agent has worked queued up, one agent is quiesced and the other agent is swapped into the DPU. The distributed management of agent overlays is based on the flow of data and enables rapid autonomous time multiplexing of portions of the network processing without the intervention of a CPU.

PUTTING IT ALL TOGETHER

The diagram in Figure 4 shows a four DPU board. The DPUs are interconnected directly with each other over a fabric (used for signaling "Fire" and "Done") and through dual ported Hybrid Memory Cubes (HMC), which act both as fast memory and as shared data buffers between the DPUs. This allows shared double buffering to improve scalability by keeping the critical data close to the processors.

The bisection bandwidth of this approach is impressive, which supports the scale-up and scale-out thesis, delivering up to 7.25 terabytes per second. The company suggests that this would translate to approximately 300 gigabytes per second (GB/s) of user data on average, since much of the bandwidth is consumed by the movement of feature map data moving between the agents in the DPUs. The bandwidth needed for agent overlays is quite small because the agents are not swapped into or out of the machine with great regularity. High speed direct memory access (DMA) is used to transfer agent code into the DPU system. The dynamic reconfigurability of the DPU architecture enables the loading of overlays of agents to be done in parallel with the execution of other agents in the DPU chip. This is a benefit of the user-partitioned dynamic reconfiguration architecture.

FIGURE 4: DATAFLOW PROCESSING UNITS INTERCONNECTED THROUGH FABRIC



The first generation of Wave's dataflow processor units (DPUs) are interconnected directly through a fabric for signaling and hybrid memory cubes, enabling memory class interconnect latency acting as double buffers to feed the DPUs.

Source: Wave Computing

CONCLUSIONS

While it is still too early to say whether Wave can deliver on the “orders of magnitude” claims that most AI startups promise, we believe the company’s DPU architecture could be an elegant approach for training and inferencing using DNNs, especially for those networks which require a high degree of scaling.

The recent acquisition of MIPS should pave the way for future performance enhancements, especially with respect to floating point operations, while an agreement with Broadcom should smooth the transition to future 7 nanometer manufacturing technology.

MI&S will await real world use case data before we accept Wave’s performance claims, but we would expect them to perform well for network training and inferencing that requires significant scale where any offload accelerator will struggle. We are especially interested in learning how well this architecture can scale using techniques such as reinforcement learning for visual recognition and natural language processing, where accelerators sometimes struggle with more than a few nodes given the lack of memory locality in these models.

Note that improving scalability is where NVIDIA has targeted the 900 GB/s NVSwitch[2] in their own DGX-2 servers with Volta-based GPUs, so this will not be a space NVIDIA will lightly concede. In addition, plans for Intel’s Nervana chip has recently been shown to include on-die switching[3] to enable scalability, so this space will be hotly contended. MI&S also suspects other startups will enter this space in the next few years with innovative approaches to scalability. Wave has said it will soon enter beta testing with select customers. We would expect Wave to collaborate on testing and tuning its system software and performance at scale with these customers with results hopefully to follow later this year.

[2] https://www.forbes.com/sites/patrickmoorhead/2018/04/09/nvidia-reinforces-machine-learning-training-lead-via-platform-improvements-at-gtc-2018/#c9b6d9621a9f

[3] https://www.forbes.com/sites/moorinsights/2018/08/17/intel-updates-for-ai-1b-now-more-growth-ahead/#75cc99e21b81

Important Information About This Paper

Contributor

Karl Freund, Senior Analyst at Moor Insights & Strategy

Publisher

Patrick Moorhead, Founder, President, & Principal Analyst at Moor Insights & Strategy

Inquiries

Contact us if you would like to discuss this report, and Moor Insights & Strategy will respond promptly.

Citations

Licensing

Disclosures

This paper was commissioned by Wave Computing. Moor Insights & Strategy provides research, analysis, advising, and consulting to many high-tech companies mentioned in this paper. No employees at the firm hold any equity positions with any companies cited in this document.

Disclaimer