cerebras

Deep Learning
Programming at Scale

Deep Learning Programming at Scale

Deep learning has become one of the most important computational workloads of our generation, advancing applications across industries from healthcare to autonomous driving. But it is also profoundly computationally intensive.

To train today's state-of-the-art neural networks, researchers often have to use large clusters of dozens to hundreds of graphics processing units (GPUs). These clusters are expensive to build, complicated to program for, and can still take days to weeks to train a network, dragging the pace of innovation. We founded Cerebras to solve this problem.

Most deep learning work begins in a prototype phase, when researchers want to experiment quickly and iterate often. When an experiment is in its earliest stages and being run with a limited subset of data, a small-scale hardware setup (a workstation with a single GPU, for example) can suffice.

But as research scales up or models move into production, the increased complexity and larger data-sets require vastly more compute. At this point, researchers have historically turned to large, scale-out GPU clusters. However, achieving good utilization across an enormous compute cluster is complicated and time consuming. It requires distributing a workload across many small devices, dealing with device memory sizes and memory bandwidth constraints, and carefully managing communication and synchronization overheads. Researchers often find themselves needing to pull in additional layers of software like Horovod and OpenMPI.

Furthermore, it is rare for massively distributed training setups to produce correct results right out of the box. Scale-out efficiency relies on using very large batch sizes, which can affect how models converge. To deal with accuracy drops, researchers run extensive experiment sweeps, tuning learning rates and trying different optimizers, in order to find an optimal training configuration that is then highly tailored to a specific hardware configuration.

So, while it is possible to bring more compute to neural network training by clustering large numbers of GPUs, it is complicated, time-consuming, and difficult for the entire organization — from ML researchers to production ML engineers, infrastructure and IT teams.

At Cerebras Systems, we believe that state-of-the-art deep learning should be simple and accessible to every ML practitioner, without requiring cluster or parallel programming expertise.

Powered by wafer-scale technology, Cerebras' CS-1 and CS-2 densify the compute and memory of an entire cluster onto a single chip in a single device, so ML researchers and practitioners can achieve cluster-scale performance with the programming ease of a single machine.

In the sections below, we highlight some of the most painful steps involved with distributed deep learning on GPU clusters and describe an easier, faster way to achieve deep learning at scale on Cerebras' CS-1 and CS-2 systems.

Traditional Approach: Distributed Training With a GPU Cluster

What does it take today to scale training across a large cluster of GPUs?

Model Distribution and Cluster Orchestration

ML Frameworks like PyTorch and TensorFlow have made it straightforward to build and run a model on a single GPU. But eventually you hit a performance wall and need to scale out to support full-scale experiments that use large amounts of data. At this point you are faced with the challenge of figuring out how to distribute your model across many GPUs.

Multi-GPU workload distribution requires thinking beyond the scope of a single neural network model and exploiting parallelism across devices. Typically, users start by changing their model code to train data-parallel across one or many multi-GPU machines, and using framework augmentations like Distributed TensorFlow or PyTorch Distributed for software configuration.

These frameworks make scaling out less painful than trying to manually implement data, or model-parallel execution, but it still takes time to tune the setups for them, and for them to learn. And getting the model running is just the first step — it is only rarely that a distributed model will run at good utilization and target accuracy right out of the box.

Device and Cluster Orchestration

Trying to run your distributed deep learning workload at higher device utilization becomes more involved. It's a complex challenge to harness maximum functional performance from many individual small processors, each with specific device constraints, which all need to be carefully managed and orchestrated.

Doing this involves figuring out how to allocate compute between devices, how to think about device memory sizes and memory constraints, and how to deal with the communication and synchronization overheads among them. This is the point at which many users bring in yet more frameworks like Horovod and libraries like OpenMPI — additional software to help with workload distribution, inter-process communication, and both inter- and intra-node communication.

But model parallelization is not deep learning research or engineering; it is supercomputer cluster engineering, and a fundamentally complex parallel programming problem. Even with the best tools, it is extremely time consuming, and often requires the expertise of supporting IT, HPC, and ML engineering teams. While this work is meant to **speed up** training across more GPUs, it is instead itself one of the main reasons why training state-of-the-art models remains such a slow and difficult process.

Convergence and Tuning

Successfully distributing a model across a cluster takes more than tuning the cluster setup and orches - trating the software; it requires researchers to change their actual model implementations.

As GPU clusters scale to 10s to 100s to even 1,000s of individual GPUs over many multi-GPU servers, researchers are often forced to use extremely large batch sizes to mitigate massive communication overheads and still be able to achieve device utilization.

But extreme batch size training often has significant impact on model convergence. It can cause the total number of epochs needed to increase considerably and can even result in drops in model accuracy.

Achieving a fast, distributed model implementation that still converges to target accuracy can take days, weeks, or even longer. Researchers often need to run dozens of experiments to find the right combina - tion of hyperparameters (e.g. batch size, learning rate and momentum), optimizers, and more to achieve convergence and target accuracy at scale.

Meanwhile, wall-clock training time also scales very sub-linearly. For example, recent results from MLPerf™[1] show that a cluster of 32 DGX-A100 systems (256 A100 GPUs) is required to achieve wall-clock accel- eration of only 14.6x beyond that of a single DGX-A100. As the need for compute grows, researchers running distributed GPU implementations have to deal with increasing software and model convergence complexities, all while getting diminishing returns in performance.

And finally, the resulting distributed cluster implementation of a deep learning model is brittle. Training time-to-accuracy on a cluster is a function of the model, data, batch size, learning rate, and number of devices. This means that if a researcher needs to change their data dimensions, dataset, model archi - tecture, or neural network layer operations / optimizers, they need to reset their work and re-do the functional hyperparam - eter tuning, debugging, and performance tuning experiments.

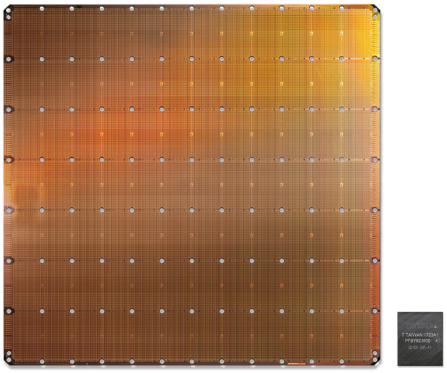
In summary, scaling deep learning training on a GPU cluster is time-intensive, complex, and often results in brittle, suboptimal solutions.

---

1 MLPerf™ v0.7 Training Closed NLP BERT. Retrieved from https:// mlcommons.org/en/training-normal-07/ 18 June 2021, entries 0.7-19 and 0.7-34. The MLPerf name and logo are trademarks of MLCommons Associa- tion in the United States and other countries. All rights reserved. Unautho - rized use strictly prohibited. See www.mlperf.org for more information.

The Cerebras Solution

At Cerebras, we built our systems to eliminate the challenges associated with scaling deep learning models across a GPU cluster.



Powered by the largest chip in the world (over 56x larger than the next-largest chip) — the second-generation wafer-scale engine (WSE-2) — the CS-2 system densifies the compute power of 850,000 AI-optimized cores onto a single chip. Keeping everything together on silicon means the CS-2 delivers not only enormous compute and on-chip memory, but also orders of magnitude more memory and interconnect bandwidth than a GPU. These purpose-built performance advantages allow the CS-2 to accelerate deep learning models far beyond what processors optimized for other applications, such as GPUs, are capable of.

A single CS-2 provides the wall-clock compute performance of an entire cluster of GPUs made up of dozens to hundreds of individual processors, at a fraction of the space and power.

For organizations, this means faster insights at lower cost. For the ML researcher, this means achieving **cluster-scale performance with the programming ease of a single device**. With the CS-2, researchers can accelerate state-of-the-art models without spending the days to weeks on the setup and tuning needed to run distributed training on large clusters of small devices.

Programming the CS-2

Because the CS-2 delivers cluster-scale acceleration in a single device, users can scale up performance and dramatically reduce time-to-solution, all while keeping their programming model simple. Data scientists and ML researchers can focus on working with their data, model, and application, rather than sinking time into addressing the challenges and idiosyncrasies of multi-device cluster orchestration and optimization.

Researchers can program the CS-2 using familiar ML frameworks like TensorFlow and PyTorch. After that, the Cerebras Graph Compiler (CGC) handles everything else to automatically translate the user's neural network graph into an optimized executable for the 850,000 cores of the CS-2.

Bringing an application to life on the CS-2 is as simple as adding a few lines of code, shown here in an example using TensorFlow.

The `CerebrasEstimator` is a wrapper class developed by our team for TensorFlow. Users simply import `CerebrasEstimator`, then define their model function, input function, relevant parameters, and training script as usual, using standard TensorFlow semantics.

The entire process is captured below:

```
from cerebras.tf.cs_estimator import CerebrasEstimator
from cerebras.tf.run_config import CSRunConfig

   est_config = CSRunConfig(
      cs_ip=params["cs_ip"],
      cs_config=cs_config,
   )
   est = CerebrasEstimator(
      model_fn=model_fn,
      model_dir=`./out`
      config=est_config,
      params=params,
      use_cs=True
   )
   est.train(input_fn, max_steps=100000, use_cs=True)
```

`CerebrasEstimator` is subclassed from the official TensorFlow Estimator class, to keep the workflow easy and familiar. The user need only instantiate the `CerebrasEstimator`, provide an IP address for their Cerebras system, and set a flag `use_cs=True` to direct training or inference to the CS-2. At runtime, `CerebrasEstimator.train()` will automatically invoke CGC and handle the rest of what is needed to prepare a model for CS-2.

Because the CS-2 is such a powerful single system, no additional work is needed to scale the network across multiple small devices or to deal with communication and synchronization issues. Users can quickly experiment with alternative model architectures, hyperparameters, and different batch sizes by changing just a few lines of code.

With a CS-2, end-to-end model development tasks such as model setup, hyperparameter optimization, scaling, and performance optimization can be done in hours or days, as compared to the weeks they can take on a traditional GPU cluster.

Time-to-Solution Advantage

CS-2's unique combination of tremendous performance and single-node simplicity not only avoids parallel programming complexity, but also unlocks much faster time-to-solution, from research concept to model-in-production.

In a typical GPU cluster setup, an ML engineer may spend days or weeks choosing and tuning hyperparameters to achieve acceptable device utilization while maintaining model accuracy at extreme batch sizes.

Because the CS-2 is a single powerful device, there is no such batch size requirement. On CS-2, researchers can train models with high utilization at any batch size, avoiding the need to run additional hyperparameter sweeps to avoid accuracy drops. This means that users can achieve not only enormous acceleration right out of the box, but can also increase model convergence to target accuracy.

In partnership with one of our life sciences customers, we recently compared the time-to-solution for a domain-specific BERT NLP model development project from concept to production using a GPU cluster versus using a Cerebras system. See **Figure 1**.

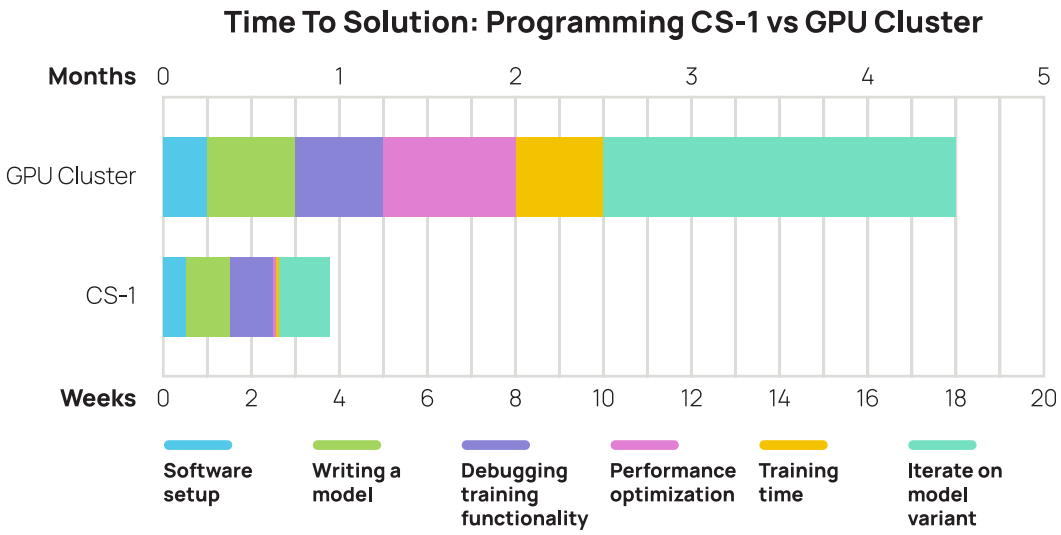


Figure 1. Overall time-to-solution from research concept to model-in-production — including programming and computation steps shown in the figure key - Cerebras system vs customer GPU cluster. The combination of Cerebras' performance and programming ease of use save researchers here more than 14 weeks of time.

We considered the same model and dataset, and included steps for software setup: model definition, functional debugging, performance optimization, initial model training, and subsequent experiments to develop a production-ready implementation.

This work showed that the Cerebras solution reduced end-to-end time to production solution from 18 weeks on a GPU cluster to four weeks on a Cerebras system. Programming and compute time were reduced by more than three months, saving our customer significant engineering costs and allowing them to accelerate new AI innovation.

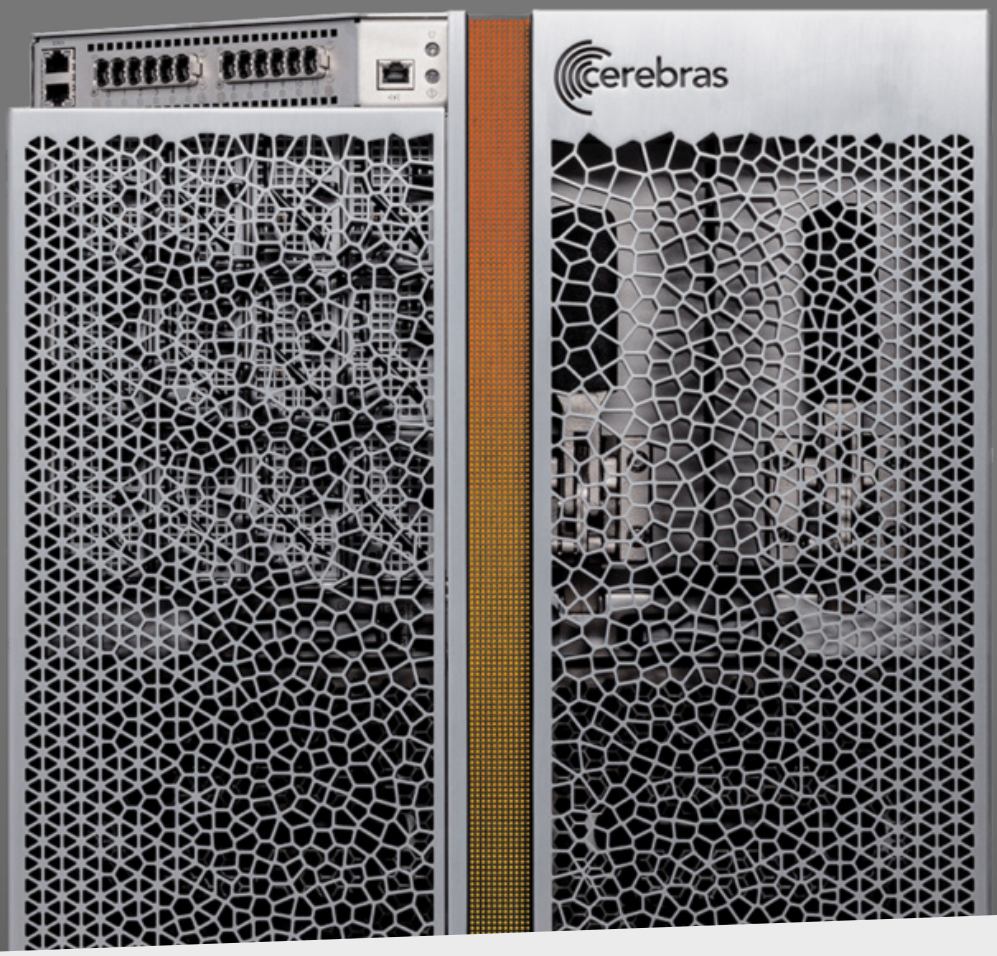
Conclusion

Deep Learning will continue to be one of the most important computational workloads of our time. Today's traditional systems are creating drag on the pace of innovation across countless industries.

At Cerebras Systems, we believe that state-of-the-art deep learning should be simple and accessible. We have been able to densify the compute and memory of an entire cluster's worth of compute onto a single chip in a single device.

We have created an easier, faster way to achieve deep learning at scale with Cerebras Systems.

To learn more or to see a demo, please contact us at cerebras.net/get-demo.



cerebras

Cerebras Systems is revolutionizing compute for Deep Learning with the CS-2 powered by the Wafer Scale Engine. The Wafer Scale Engine delivers more compute, more memory, and more communication bandwidth for artificial intelligence research at previously-impossible speeds and scale. Pioneering computer architects, computer scientists, and deep learning researchers have come together to build a new class of computer system that accelerates AI by orders of magnitude beyond the current state of the art.