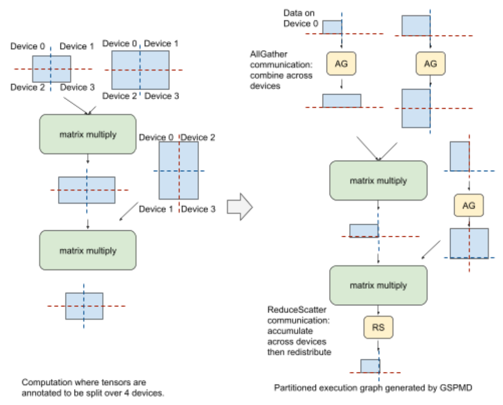
Research

Home Blog

General and Scalable Parallelization for Neural Networks


Device 0
Device 1
Device 2
Device 3
Device 0
Device 1
Device 2
Device 3
matrix multiply
Device 0
Device 2
Device 1
Device 3
matrix multiply
Computation where tensors are annotated to be split over 4 devices.
Data on Device 0
AllGather communication: combine across devices
AG
AG
matrix multiply
AG
matrix multiply
ReduceScatter communication: accumulate across devices then redistribute
RS
Partitioned execution graph generated by GSPMD

December 8, 2021

Posted by Yuanzhong Xu and Yanping Huang, Software Engineers; Google Research, Brain Team

or the computation being utilized, has been critical for improving model quality in many real-world machine learning applications, such as computer vision, language understanding and neural machine translation. This, in turn, has motivated recent studies to scrutinize the factors that play a critical role in the success of scaling a neural model. Although increasing model capacity can be a sound approach to improve model quality, doing so presents a number of systems and software engineering challenges that must be overcome. For instance, in order to train large models that exceed the memory capacity of an accelerator, it becomes necessary to partition the weights and the computation of the model across multiple accelerators. This process of parallelization increases the network communication overhead and can result in device under-utilization. Moreover, a given algorithm for parallelization, which typically requires a significant amount of engineering effort, may not work with different model architectures.

To address these scaling challenges, we present "GSPMD: General and Scalable Parallelization for ML Computation Graphs", in which we describe an open-source automatic parallelization system based on the XLA compiler. GSPMD is capable of scaling most deep learning network architectures and has already been applied to many deep learning models, such as GShard-M4, LaMDA, BigSSL, ViT, and MetNet-2, leading to state-of-the-art-results across several domains. GSPMD has also been integrated into multiple ML frameworks, including TensorFlow and JAX, which use XLA as a shared compiler.

Overview

GSPMD separates the task of programming an ML model from the challenge of parallelization. It allows model developers to write programs as if they were run on a single device with very high memory and computation capacity — the user simply needs to add a few lines of annotation code to a subset of critical tensors in the model code to indicate how to partition the tensors. For example, to train a large model-parallel Transformer, one may only need to annotate fewer than 10 tensors (less than 1% of all tensors in the entire computation graph), one line of additional code per tensor. Then GSPMD runs a compiler pass that determines the entire graph's parallelization plan, and transforms it into a mathematically equivalent, parallelized computation that can be executed on each device. This allows users to focus on model building instead of parallelization implementation, and enables easy porting of existing single-device programs to run at a much larger scale.

example, the model code that powered the GShard-M4 and LaMDA models can apply a variety of parallelization strategies appropriate for different models and cluster sizes with the same model implementation. Similarly, by applying GSPMD, the BigSSL large speech models can share the same implementation with previous smaller models.

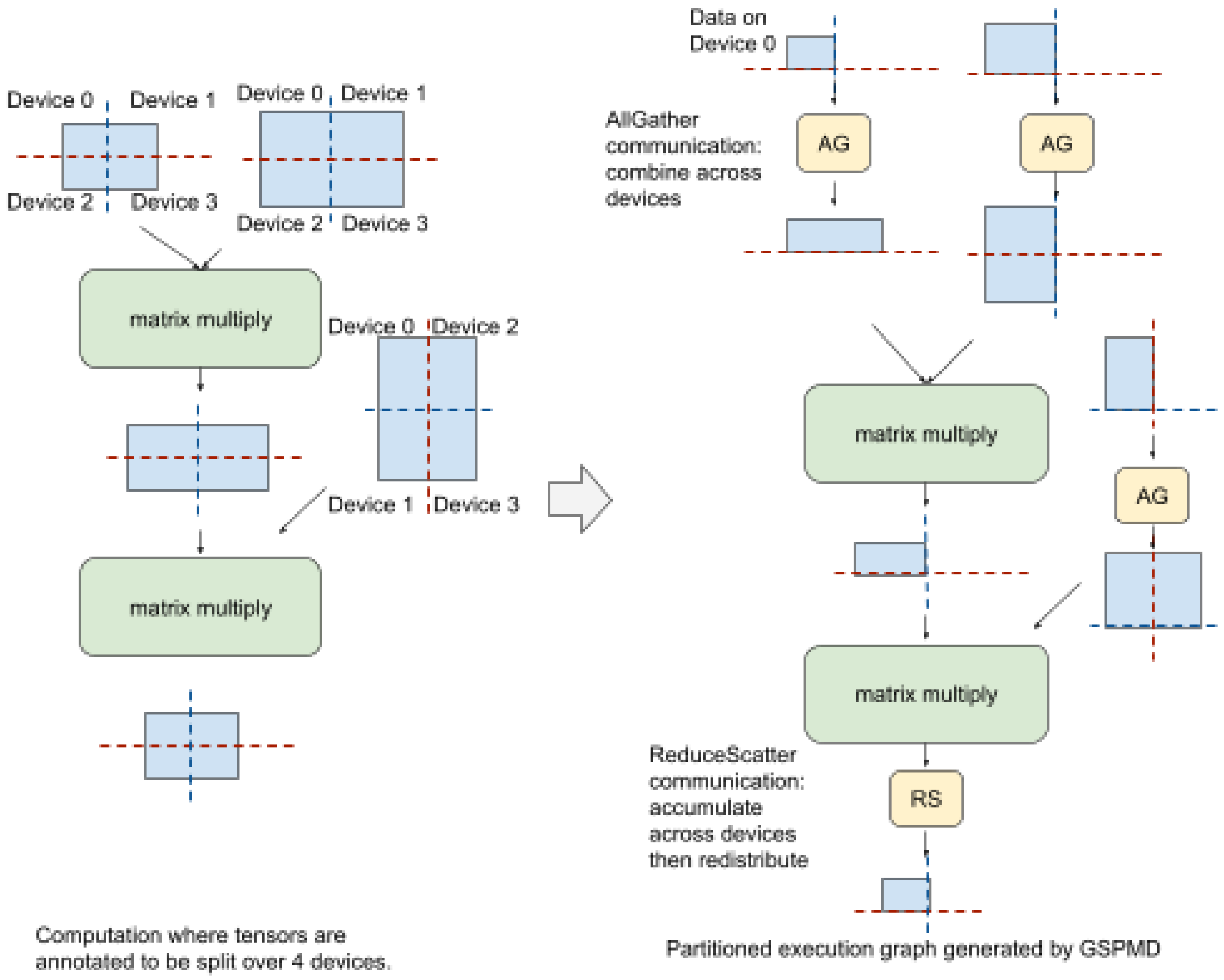
Generality and Flexibility

Because different model architectures may be better suited to different parallelization strategies, GSPMD is designed to support a large variety of parallelism algorithms appropriate for different use cases. For example, with smaller models that fit within the memory of a single accelerator, *data parallelism* is preferred, in which devices train the same model using different input data. In contrast, models that are larger than a single accelerator's memory capacity are better suited for a *pipelining algorithm* (like that employed by GPipe) that partitions the model into multiple, sequential stages, or *operator-level parallelism* (e.g., Mesh-TensorFlow), in which individual computation operators in the model are split into smaller, parallel operators.

GSPMD supports all the above parallelization algorithms with a uniform abstraction and implementation. Moreover, GSPMD supports nested patterns of parallelism. For example, it can be used to partition models into individual pipeline stages, each of which can be further partitioned using operator-level parallelism.

GSPMD also facilitates innovation on parallelism algorithms by allowing performance experts to focus on algorithms that best utilize the hardware, instead of the implementation that involves lots of cross-device communications. For example, for large Transformer models, we found a novel operator-level parallelism algorithm that partitions multiple dimensions of tensors on a 2D mesh of devices. It reduces peak accelerator memory usage linearly with the number of training devices, while maintaining a high utilization of accelerator compute due to its balanced data distribution over multiple dimensions.

To illustrate this, consider a simplified feedforward layer in a Transformer model that has been annotated in the above way. To execute the first matrix multiply on fully partitioned input data, GSPMD applies an MPI-style AllGather communication operator to partially merge with partitioned data from another device. It then executes the matrix multiply locally and produces a partitioned result. Before the second matrix multiply, GSPMD adds another AllGather on the right-hand side input, and executes the matrix multiply locally, yielding intermediate results that will then need to be combined and are short-lived and the corresponding memory buffers will be freed after use, which does not affect peak memory usage in training.


Device 0
Device 1
Device 2
Device 3
matrix multiply
Computation where tensors are annotated to be split over 4 devices.
Data on Device 0
AllGather communication: combine across devices
AG
RS
ReduceScatter communication: accumulate across devices then redistribute
Partitioned execution graph generated by GSPMD

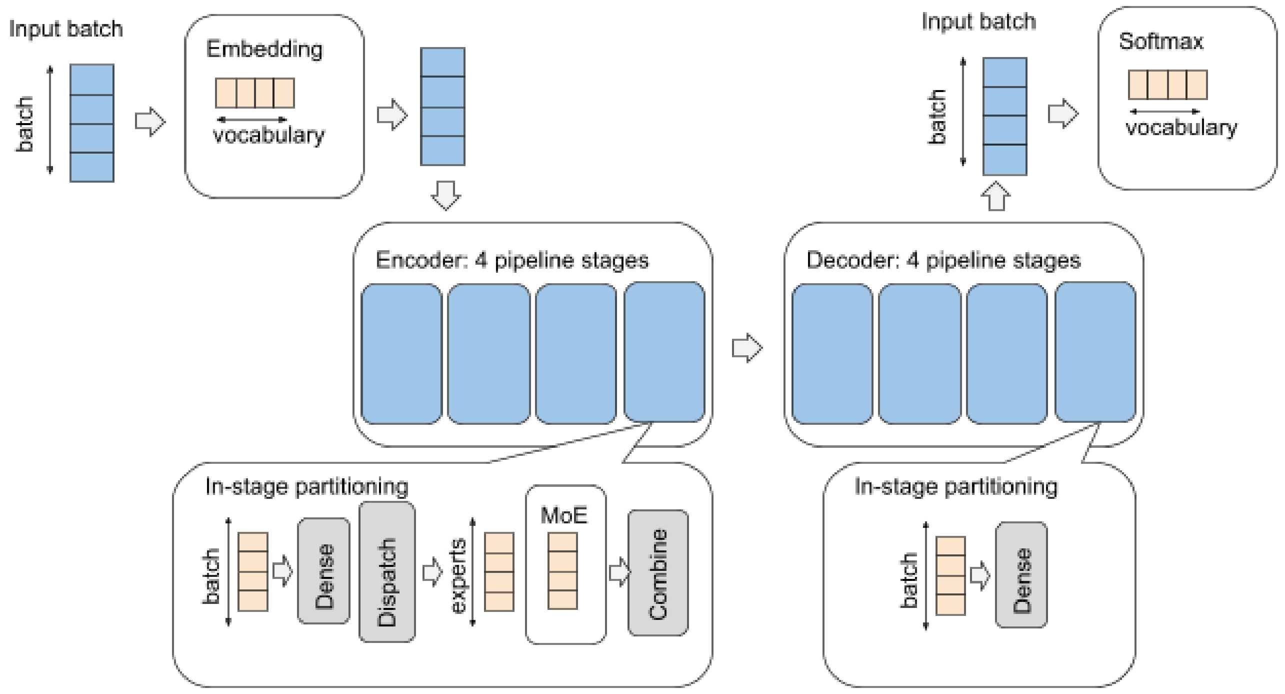
Left: *A simplified feedforward layer of a Transformer model. Blue rectangles represent tensors with dashed red & blue lines overlaid representing the desired partitioning across a 2x2 mesh of devices.* ***Right***: *A single partition, after GSPMD has been applied.*

A Transformer Example with Nested Parallelism

As a shared, robust mechanism for different parallelism modes, GSPMD allows users to conveniently switch between modes in different parts of a model. This is particularly valuable for models that may have different components with distinct performance characteristics, for example, multimodal models that handle both images and audio.

of several parallelism modes that treats each layer separately can be achieved with simple configurations.

In the figure below, we show a partitioning strategy over 16 devices organized as a logical 4x4 mesh. Blue represents partitioning along the first mesh dimension *X*, and yellow represents partitioning along the second mesh dimension *Y*. *X* and *Y* are repurposed for different model components to achieve different parallelism modes. For example, the *X* dimension is used for data parallelism in the embedding and softmax layers, but used for pipeline parallelism in the encoder and decoder. The *Y* dimension is also used in different ways to partition the vocabulary, batch or model expert dimensions.



Computation Efficiency

GSPMD provides industry-leading performance in large model training. Parallel models require extra communication to coordinate multiple devices to do the computation. So parallel model efficiency can be estimated by examining the fraction of time spent on communication overhead — the higher percentage utilization and the less time spent on communication, the better. In the recent MLPerf[1] set of performance benchmarks, a BERT-like encoder-only model with ~500 billion parameters to which we applied GSPMD for parallelization over 2048 TPU-V4 chips yielded highly competitive results (see table instructions to run them on Google Cloud. More benchmark results can be found in the experiment section of our paper.

Model Family	***Parameter Count***	***% of model activated****	***No. of Experts****	***No. of Layers***	***No. of TPU***	***FLOPS utilization***
Dense Decoder (LaMDA)	137B	100%	1	64	1024 TPUv3	56.5%
Dense Encoder (MLPerf-Bert)	480B	100%	1	64	2048 TPUv4	63%
Sparsely Activated Encoder-Decoder (GShard-M4)	577B	0.25%	2048	32	1024 TPUv3	46.8%
Sparsely Activated Decoder	1.2T	8%	64	64	1024 TPUv3	53.8%

**The fraction of the model activated during inference, which is a measure of model sparsity.*
***Number of experts included in the Mixture of Experts layer. A value of 1 corresponds to a standard Transformer, without a Mixture of Experts layer.*

Conclusion

The ongoing development and success of many useful machine learning applications, such as NLP, speech recognition, machine translation, and autonomous driving, depend on achieving the highest accuracy possible. As this often requires building larger and even more complex models, we are pleased to share the GSPMD paper and the corresponding open-source library to the broader research community, and we hope it is useful for efficient training of large-scale deep neural networks.

Research

We wish to thank Claire Cui, Zhifeng Chen, Yonghui Wu, Naveen Kumar, Macduff Hughes, Zoubin Ghahramani and Jeff Dean for their support and invaluable input. Special thanks to our collaborators Dmitry Lepikhin, HyoukJoong Lee, Dehao Chen, Orhan Firat, Maxim Krikun, Blake Hechtman, Rahul Joshi, Andy Li, Tao Wang, Marcello Maggioni, David Majnemer, Noam Shazeer, Ankur Bapna, Sneha Kudugunta, Quoc Le, Mia Chen, Shibo Wang, Jinliang Wei, Ruoming Pang, Zongwei Zhou, David So, Yanqi Zhou, Ben Lee, Jonathan Shen, James Qin, Yu Zhang, Wei Han, Anmol Gulati, Laurent El Shafey, Andrew Dai, Kun Zhang, Nan Du, James Bradbury, Matthew Johnson, Anselm Levskaya, Skye Wanderman-Milne, and Qiao Zhang for helpful discussions and inspirations.

---

[1] The MLPerf name and logo are trademarks of MLCommons Association in the United States and other countries. All rights reserved. Unauthorized use is strictly prohibited. See www.mlcommons.org for more information. ↩

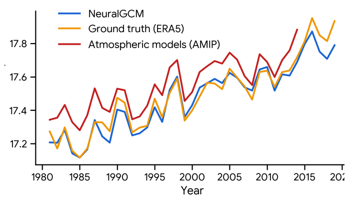
[2] FLOPS utilization is not an official MLPerf metric. ↩

Labels:

Distributed Systems & Parallel Computing

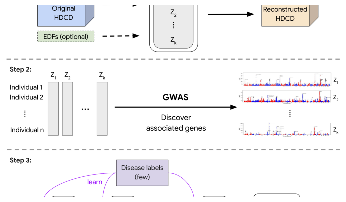
Machine Intelligence

Other posts of interest



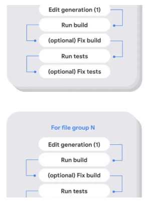
JULY 22, 2024

Fast, accurate climate modeling with NeuralGCM



JULY 18, 2024

Harnessing hidden genetic information in



JULY 18, 2024

Accelerating code migrations with AI

Google Research

Machine Intelligence · Open Source Models & Datasets

Generative AI · Health & Bioscience · Machine Intelligence · Open Source Models & Datasets

Software Systems & Engineering