Mesh-TensorFlow: Deep Learning for SuperComputers

Noam Shazeer, Youlong Cheng, Niki Parmar, Dustin Tran, Ashish Vaswani, Penporn Koanantakool,
Peter Hawkins, HyoukJoong Lee, Mingsheng Hong,Cliff Young, Ryan Sepassi, Blake Hechtman, **Google AI**

Massive Model-Parallelism (and/or data-parallelism) Made Easier

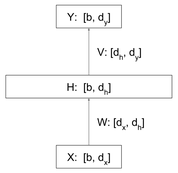
- Describe the overall computation in a TensorFlow-like language with named tensor-dimensions.
- Describe your physical cores as n-dimensional array of processors by specifying a “mesh”.
- Describe which tensor-dimensions should be split across which dimensions of the mesh of processors.
- You’re Done - Mesh-TensorFlow compiles your graph into Single-Program-Multiple-Data (SPMD) TensorFlow code plus collective communication primitives.

Motivation

- Ability to train **GIANT** multi-billion/trillion-parameter models which do not fit on one processor.
- Ability to process giant examples (spatial/temporal splitting of images/video, etc.)
- Low latency by splitting computation for one example across multiple processors.
- Traditional MIMD approaches to model-parallelism are:
  - Tricky to design
  - Create giant cumbersome graphs
  - Are prone to bottlenecks

Example Perceptron

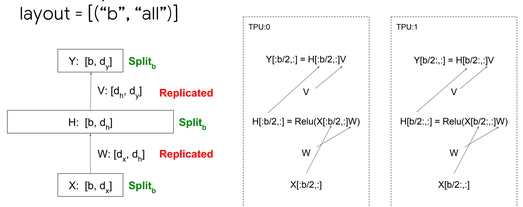
```
Y = (H = Relu(XW))V
```




```
# x.shape == [(“b”: 4096), (“d_x”: 1024)]
w = mtf.get_variable([(“d_x”: 1024), (“d_h”: 8192)]
v = mtf.get_variable([(“d_h”: 8192), (“d_y”: 1024)]
h = mtf.einsum([x, w], [(“b”: 4096), (“d_h”: 8192)])
h = mtf.relu(h)
y = mtf.einsum([h, v], [(“b”: 4096), (“d_y”: 8192)])
...
```

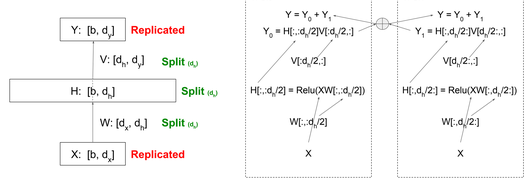
… with data-parallelism (split “b” dimension)

```
mesh_shape = [(“all”, 2)]
layout = [(“b”, “all”)]
```



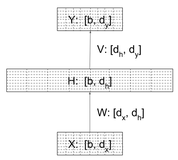
… with model-parallelism (split “d_h” dimension)

```
mesh_shape = [(“all”, 2)]
layout = [(“d_h”, “all”)]
```



… data+model parallelism on a 3d mesh of 512 processors

```
mesh_shape = [(“m_0”, 8), (“m_1”, 8), (“m_2”, 8)]
layout = [(“b”, “m0”), (“d_h”, “m_1”), (“d_x”, “m_2”), (“d_y”, “m_2”)]
```



Use a 3-dimensional mesh of processors with dimensions [m_0, m_1, m_2].

- Split “b” across mesh dimension m_0
- Split “d_h” across mesh dimension m_1
- Split “d_x”, “d_y” across mesh dimension m_2
- Each matrix is tiled across two mesh dimensions and replicated across the third.
- Each matmul consists of local matmuls, followed by allreduce across one mesh dimension.

Results

- Mesh-TF Transformer Model

$$\text{Attention}(Q,K,V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

$$\text{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2$$

- Transformer models working in MeshTF
- Layouts for Transformer model

```
layout_data_parallel=”batch:m0”
layout_model_parallel=”vocab:m0,heads:m0,d_ff:m0”
layout_dp_mp=”batch:m0,vocab:m1,heads:m1,d_ff:m1”
```

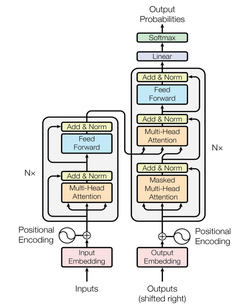


Figure 1: The Transformer - model architecture.

Table 3: Transformer Machine-Translation Results. $d_{model} = 1024$, $d_k = d_v = 128$

d_{ff}	$heads$	d_k, d_v	Parameters (Billions)	WMT14 EN-DE BLEU	WMT14 EN-FR BLEU	
2048	4	128	0.15	25.5	41.8	
4096	8	128	0.24	26.5	42.5	
8192	16	128	0.42	27.1	43.3	
16384	32	128	0.77	27.5	43.5	
32768	64	128	1.48	27.5	43.8	
65536	128	128	2.89	26.7	**43.9**	
4096	16	64	0.21	**28.4**	41.8	[21]

Table 2: Transformer-Decoder Language Models: $d_{model} = 1024$, $d_k = d_v = 256$

d_{ff}	$heads$	Parameters (Billions)	Billion-Word Benchmark Word-Perplexity	Wikipedia Subword-Perplexity
4096	4	0.14	35.0	8.74
8192	8	0.22	31.7	8.03
16384	16	0.37	28.9	7.44
32768	32	0.67	26.8	6.99
65516	64	1.28	25.1	6.55
131072	128	2.48	24.1	6.24
262144	256	4.90	24.0(**23.5**)	**6.01**
Prev Best DNN [20]		6.5	28.0	
Best DNN Ensemble [17]			26.1	
Best Ensemble (different methods)[17]		> 100	23.7	

- Trained Transformer models with up to 5B parameters on up to 512 TPU cores.
- >50% MXU utilization (6PFLOPS/possible 11.5 PFLOP/s)

Status

- Code is Open-Source on github - please contribute. https://github.com/tensorflow/mesh/tree/master/mesh_tensorflow
- Implementations to produce SPMD code for TPU or MIMD code for multi-CPU/GPU.
- Integrated on Google Cloud TPU along with examples like the Transformer.