GPipe: Easy Scaling with Micro-Batch Pipeline Parallelism

Yanping Huang, et al.

NIPS 2019

Presenter: Constant (Sang-Soo) Park

sonicstage12@naver.com

April 14, 2020

Neural Acceleration Study Season #2

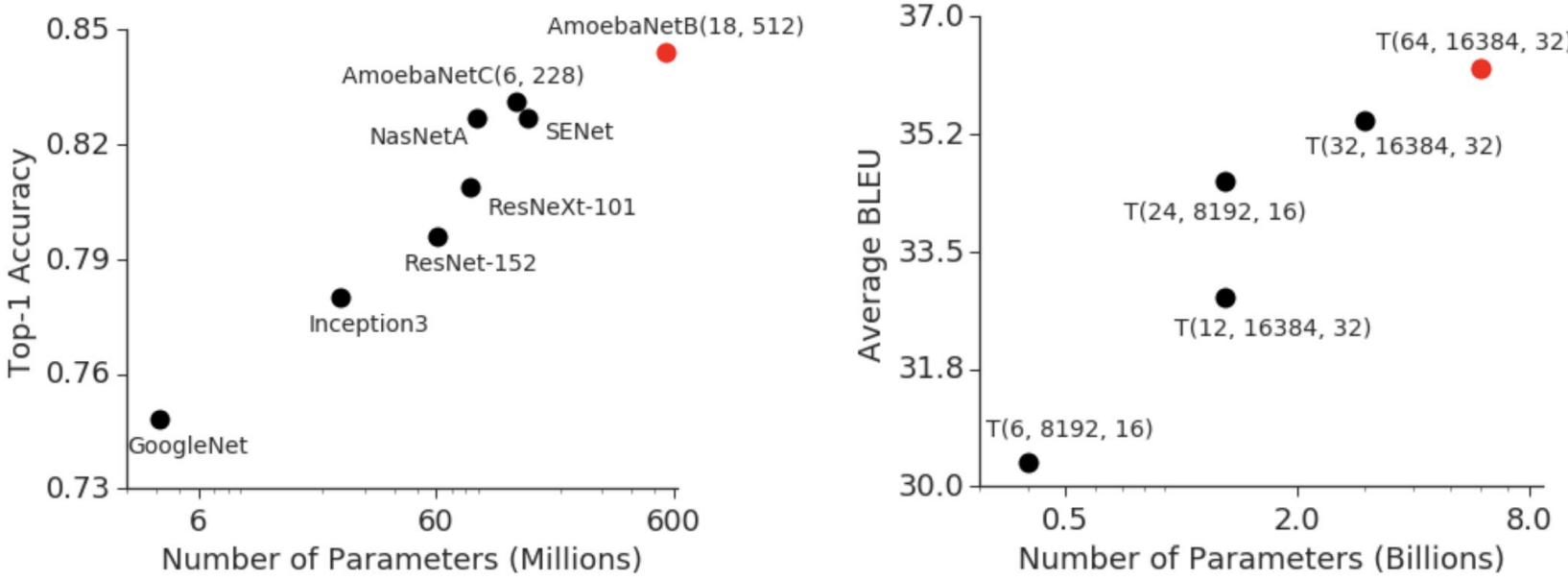
Contents of presentation

- **Introduction**
- **Gpipe concept**
  - Interface, Algorithm, Performance optimization
- **Performance analysis**

Introduction: Becoming more larger

- **Bring remarkable quality improvements to several fields**
  - Trend: **accuracy improvements** on ImageNet with increase in **model capacity**
  - Similar phenomenon be overserved in **context of natural language processing**

$$T(L, H, A) = transformer\ with\ L\ encoder\ and\ decoder\ (dimension\ of\ H\ and\ A\ attention\ heads)$$



[1] https://www.synopsys.com/designware-ip/technical-bulletin/building-efficient-deep-learning-dwtb_q318.html

Introduction: Becoming more larger

- **Practical challenges with larger models**
  - HW constraints (**memory limitation**, communication bandwidth on accelerator)
  - **Dividing larger models** into partitions and assignment of different partitions to different accelerators[1]

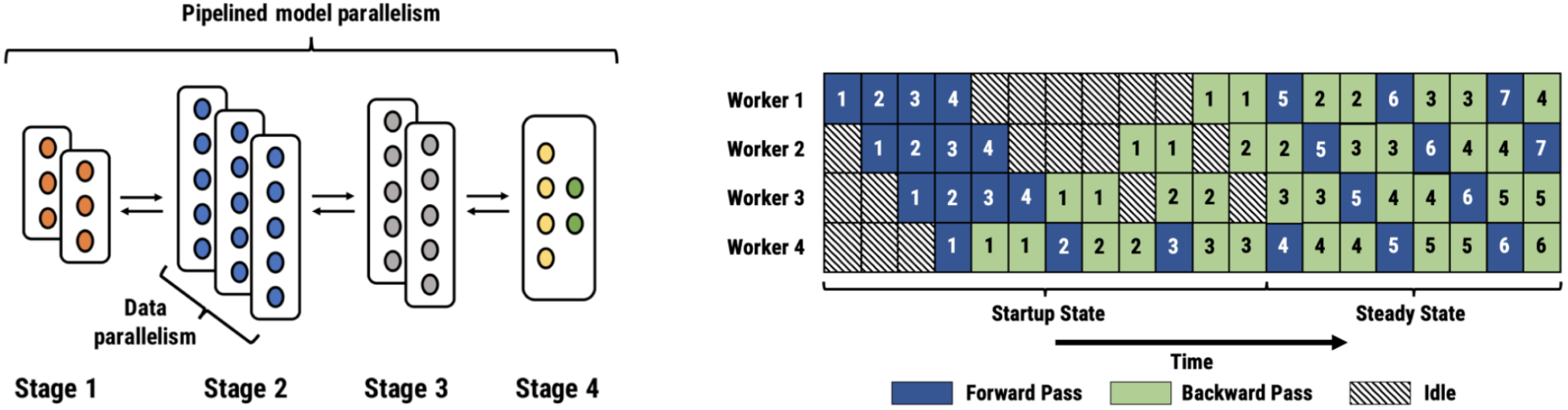
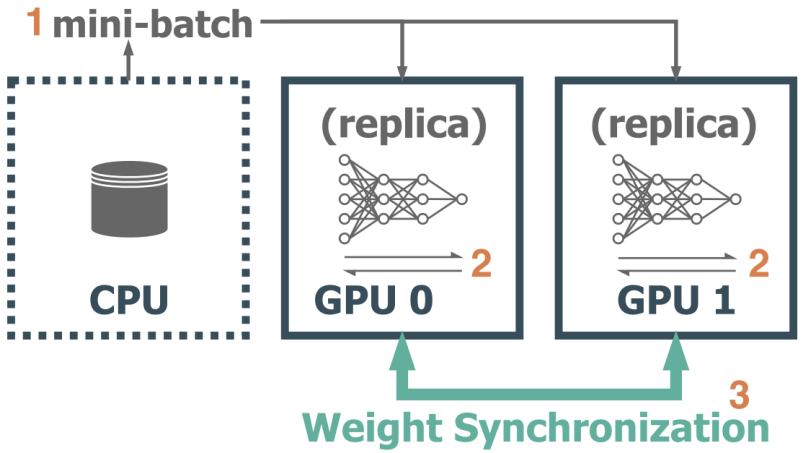


Figure 2: In the figure on the left, we show an example pipeline-parallel assignment with eight GPUs split across four stages. Only activations and gradients at stage boundaries are communicated. Stages 1, 2, and 3 have replicated stages to ensure a load-balanced pipeline. The figure on the right shows another pipeline with 4 workers, showing start-up and steady states. In this example, the backward pass takes twice as long as the forward pass.

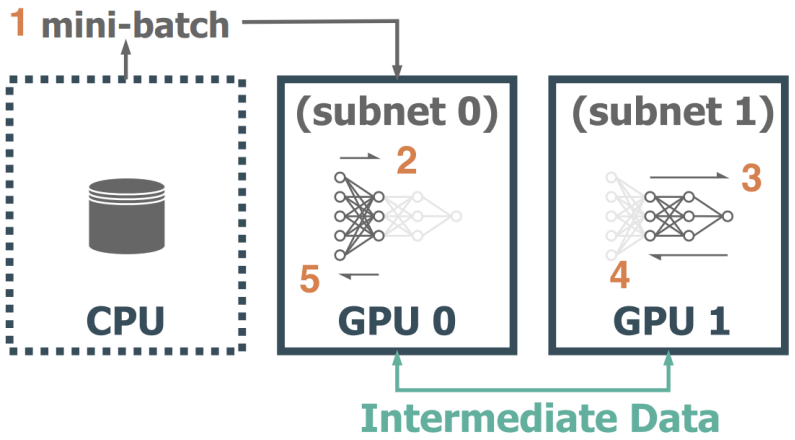
[1] https://www.microsoft.com/en-us/research/blog/pipedream-a-more-effective-way-to-train-deep-neural-networks-using-pipeline-parallelism/

Introduction: Becoming more larger

- **Model parallelism**
  - Hard to design and implement **efficient model parallelism algorithm**[2]
  - **Using multiple GPUs,** Each GPU is responsible for weight updates of assigned model layers
  - Architecture and task-specific and Increasing demand for reliable and flexible infrastructure



(a) Data Parallelism.

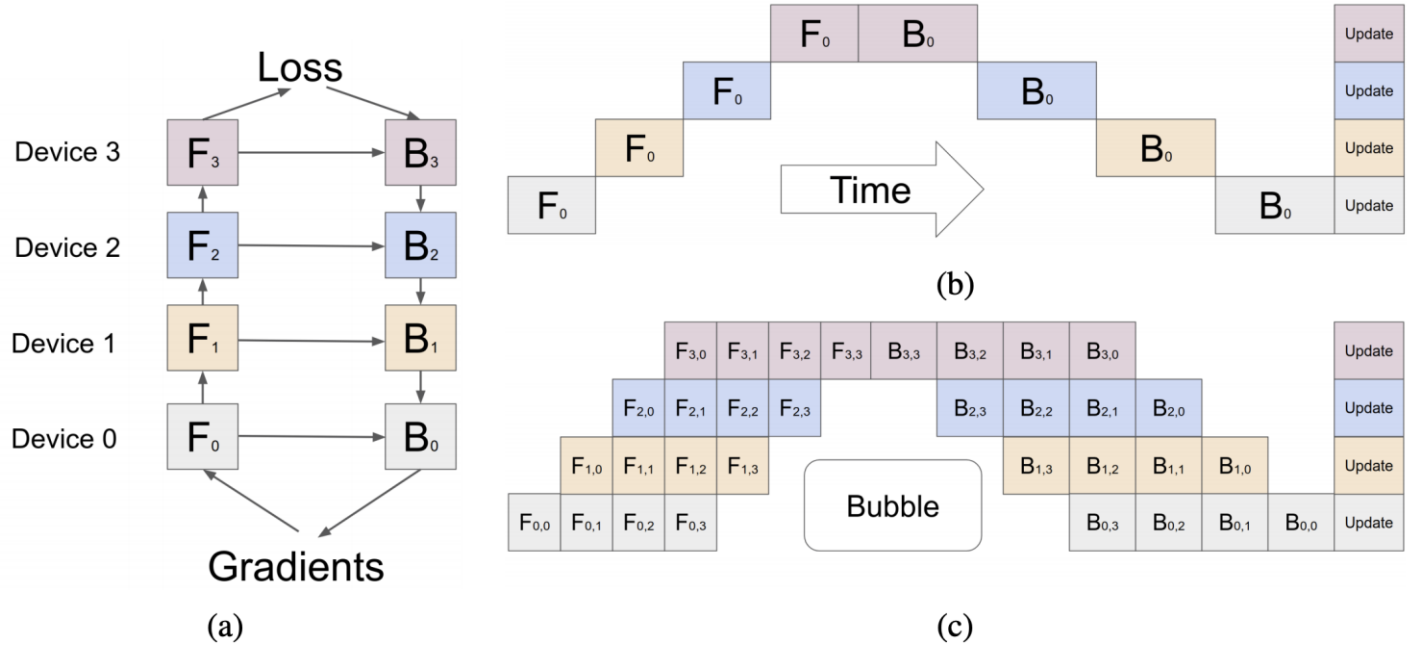


(b) Model Parallelism.

[2] https://arxiv.org/pdf/1809.02839.pdf

Introduction: Becoming more larger

- **Flexible library for efficient training of large neural networks**
  - Scaling arbitrary DNN architecture by partitioning models, supporting re-materialization on accelerators
  - Layers can be partitioned into cells and each cell is then placed on separate accelerator
  - **Pipeline parallelism algorithm with batch splitting (*mini-batch*, *micro-batches*)**

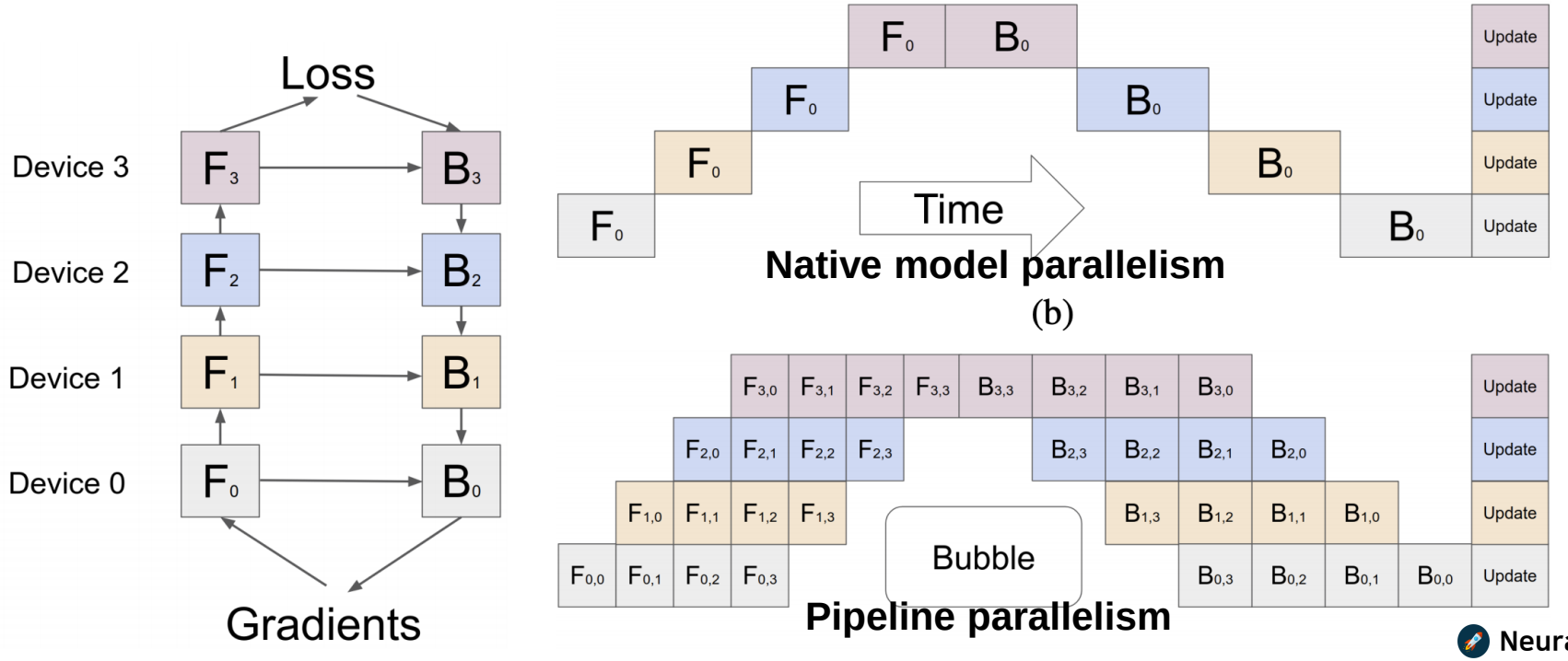


GPipe: Interface

- **Background**
  - Deep neural network can be defined as sequence of L layers
  - Each layer L_i is composed of forward computation (f_i) with set of parameters (w_i)
  - Sequence of L layers can be partitioned into K composite layers, or cells
  - Set of parameter (p_k) consist of consecutive layers between layers i and j.
  - $p_k = [w_i, w_{i+1}, \ldots, w_j]$, $F_k = f_j \circ \cdots \circ f_{i+1} \circ f_i$, B_k can be computed from F_k, cost estimation function (c_i)
  - Number of model partitions K, number of micro-batches M, sequence and definitions of L layers

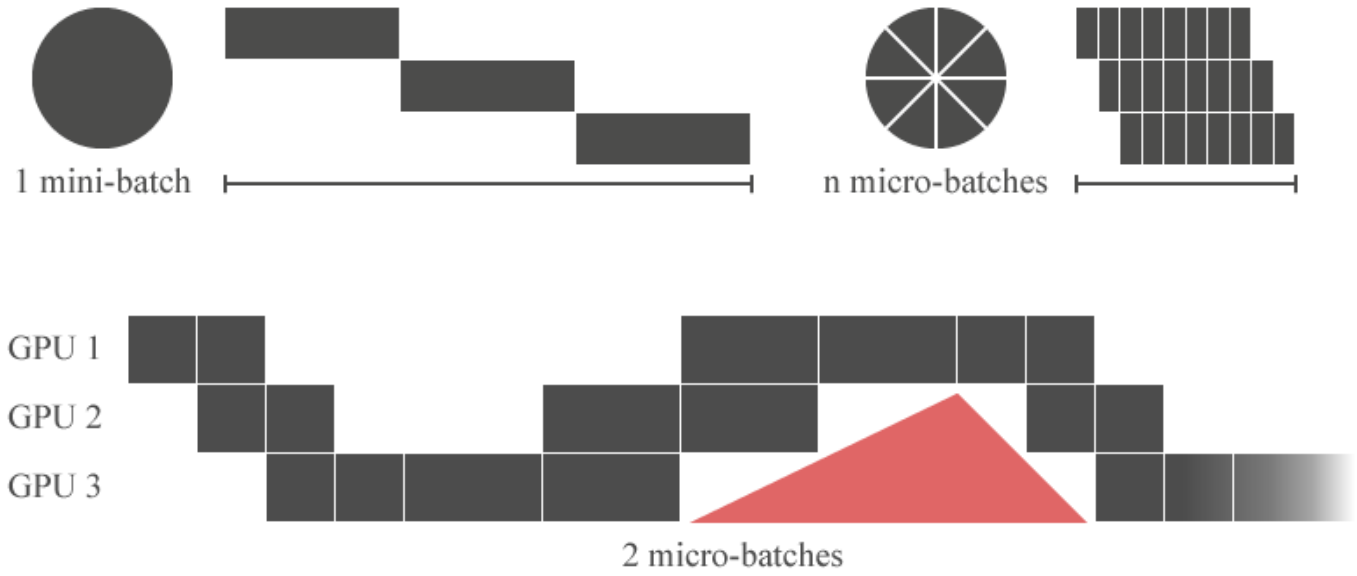
GPipe: Algorithm

- **Network into L cells and placing cells on k^{th} accelerator**
  - Forward: dividing every mini-batch of size N into M equal micro-batches (pipelined through K accelerators)
  - Backward, gradients for each micro-batch are computed on same model parameters used for forward
  - At end of each mini-batch, gradients from M micro-batches are accumulated/applied to update parameters



GPipe: Algorithm

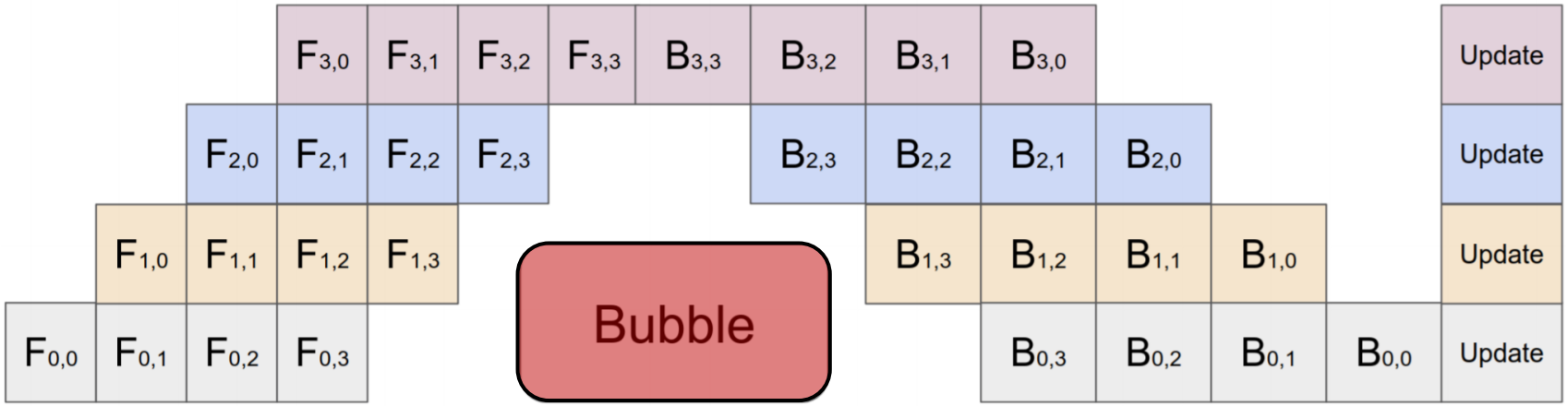
- **Pipeline parallelism**
  - Amount of overlapped processing increases, allowing same amount of data to be processed in less time[3]
  - As size of Micro-batch gets smaller, next GPU can start working faster (idle time gradually decreases)
  - However, size is too small, efficiency is reduced (Not enough to compute workload for GPU)



[3] https://www.kakaobrain.com/blog/66

GPipe: Performance optimization

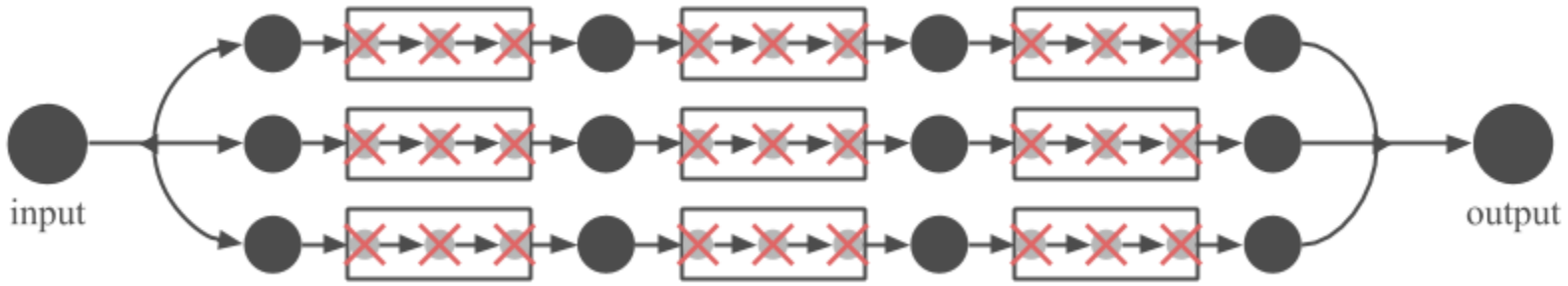
- **Re-materialization: to reduce activation memory requirement**
  - Forward, each accelerator only stores output activations a partition boundaries
  - Backward, k^{th} accelerator recompute composite forward function (F_k)
  - **Peak activation memory** is reduced to $\boldsymbol{O}\left(\boldsymbol{N}+\frac{\boldsymbol{L}}{\boldsymbol{K}}*\frac{\boldsymbol{N}}{\boldsymbol{M}}\right)$, $\frac{N}{M}$ (micro-batch), $\frac{L}{K}$ (# of layers per partition)
  - Bubble overhead: some idle time per accelerator



Bubble time $\boldsymbol{O}\left(\boldsymbol{N}+\frac{\boldsymbol{L}}{\boldsymbol{K}}*\frac{\boldsymbol{N}}{\boldsymbol{M}}\right)$ amortized over number of micro-steps $\boldsymbol{M}$
(to be negligible when $\boldsymbol{M \geq 4*K}$)

GPipe: Performance optimization

- **Re-materialization with check-pointing**
  - Checkpoint: point at division of model
  - **Only hidden layers connecting separate models to each other is placed in memory**
  - In forward propagation, **discard result of hidden layers inside separated models**[3]
  - **Peak activation memory** is reduced to $\boldsymbol{O}\left(\boldsymbol{N} + \frac{\boldsymbol{L}}{\boldsymbol{K}} * \frac{\boldsymbol{N}}{\boldsymbol{M}}\right)$, w/o $\boldsymbol{O}(\boldsymbol{N} * \boldsymbol{L})$



[3] https://www.kakaobrain.com/blog/66

GPipe: Performance analysis

- **Performance with two different types of models: AmoebaNet**
  - **AmoebaNet** (convolutional model), **Transformer** (sequence-to-sequence model)
  - Both re-materialization and pipeline parallelism **to benefit memory utilization**
  - **Biggest model size GPipe can support under reasonably large input size**
  - **25x** more memory than what is possible w/o GPipe

Table 1: Maximum model size of AmoebaNet supported by GPipe under different scenarios. Naive-1 refers to the sequential version without GPipe. Pipeline-k means k partitions with GPipe on k accelerators. AmoebaNet-D (L, D): AmoebaNet model with L normal cell layers and filter size D . Transformer-L: Transformer model with L layers, 2048 model and 8192 hidden dimensions. Each model parameter needs 12 bytes since we applied RMSProp during training.

NVIDIA GPUs (8GB each)	Naive-1	Pipeline-1	Pipeline-2	Pipeline-4	Pipeline-8
AmoebaNet-D (L, D)	(18, 208)	(18, 416)	(18, 544)	(36, 544)	(72, 512)
# of Model Parameters	82M	318M	542M	1.05B	1.8B
Total Model Parameter Memory	1.05GB	3.8GB	6.45GB	12.53GB	24.62GB
Peak Activation Memory	6.26GB	3.46GB	8.11GB	15.21GB	26.24GB
Cloud TPUv3 (16GB each)	**Naive-1**	**Pipeline-1**	**Pipeline-8**	**Pipeline-32**	**Pipeline-128**
Transformer-L	3	13	103	415	1663
# of Model Parameters	282.2M	785.8M	5.3B	21.0B	83.9B
Total Model Parameter Memory	11.7G	8.8G	59.5G	235.1G	937.9G
Peak Activation Memory	3.15G	6.4G	50.9G	199.9G	796.1G

imbalanced distribution (model parameters)

GPipe: Performance analysis

- **Performance with two different types of models: Transformer**
  - **AmoebaNet** (convolutional model), **Transformer** (sequence-to-sequence model)
  - Both re-materialization and pipeline parallelism **to benefit memory utilization**
  - **Biggest model size GPipe can support under reasonably large input size**
  - **298x** more memory than what is possible w/o GPipe

Table 1: Maximum model size of AmoebaNet supported by GPipe under different scenarios. Naive-1 refers to the sequential version without GPipe. Pipeline-k means k partitions with GPipe on k accelerators. AmoebaNet-D (L, D): AmoebaNet model with L normal cell layers and filter size D . Transformer-L: Transformer model with L layers, 2048 model and 8192 hidden dimensions. Each model parameter needs 12 bytes since we applied RMSProp during training.

NVIDIA GPUs (8GB each)	Naive-1	Pipeline-1	Pipeline-2	Pipeline-4	Pipeline-8
AmoebaNet-D (L, D)	(18, 208)	(18, 416)	(18, 544)	(36, 544)	(72, 512)
# of Model Parameters	82M	318M	542M	1.05B	1.8B
Total Model Parameter Memory	1.05GB	3.8GB	6.45GB	12.53GB	24.62GB
Peak Activation Memory	6.26GB	3.46GB	8.11GB	15.21GB	26.24GB
Cloud TPUv3 (16GB each)	**Naive-1**	**Pipeline-1**	**Pipeline-8**	**Pipeline-32**	**Pipeline-128**
Transformer-L	3	13	103	415	1663
# of Model Parameters	282.2M	785.8M	5.3B	21.0B	83.9B
Total Model Parameter Memory	11.7G	8.8G	59.5G	235.1G	937.9G
Peak Activation Memory	3.15G	6.4G	50.9G	199.9G	796.1G

imbalanced distribution (model parameters)

GPipe: Performance analysis

- **Normalized training throughput of AmoebaNet and Transformer**
  - # of micro-batches M is al least 5x # of partitions, bubble overhand is almost negligible
  - Transformer, 3.5x speedup when it is partitioned across four times more accelerators
  - AmoebaNet, sub-linear speedup due to its imbalanced computation distribution
  - When M is relatively small, bubble overhead can no longer be negligible (M =1, no pipeline parallelism)

Table 2: Normalized training throughput using GPipe with different # of partitions K and different # of micro-batches M on TPUs. Performance increases with more micro-batches. There is an almost linear speedup with the number of accelerators for Transformer model when $M \gg K$. Batch size was adjusted to fit memory if necessary.

TPU	AmoebaNet			Transformer		
$K =$	2	4	8	2	4	8
$M = 1$	1	1.13	1.38	1	1.07	1.3
$M = 4$	1.07	1.26	1.72	1.7	3.2	4.8
$M = 32$	1.21	1.84	3.48	1.8	3.4	6.3

Table 3: Normalized training throughput using GPipe on GPUs without high-speed interconnect.

GPU	AmoebaNet			Transformer		
$K =$	2	4	8	2	4	8
$M = 32$	1	1.7	2.7	1	1.8	3.3

GPipe: Performance analysis

- **Image classification**
  - AmoebaNet with scaled input image size (480x480)
  - Single model: 4 partitions, 84.4$ (top-1), 97% (top-5)
  - Effectiveness of giant convolution networks on other image datasets through transfer learning

Table 4: Image classification accuracy using AmoebaNet-B (18, 512) first trained on ImageNet 2012 then fine-tuned on others. Please refer to the supplementary material for a detailed description of our training setup. Our fine-tuned results were averaged across 5 fine-tuning runs. Baseline results from Real *et al*. [12] and Cubuk *et al*. [26] were directly trained from scratch. *Mahajan *et al*.'s model [27] achieved 85.4% top-1 accuracy but it was pretrained on non-public Instagram data. Ngiam *et al*. [28] achieved better results by pre-training with data from a private dataset (JFT-300M).

Dataset	# Train	# Test	# Classes	Accuracy (%)	Previous Best (%)
ImageNet-2012	1,281,167	50,000	1000	**84.4**	83.9 [12] (85.4*[27])
CIFAR-10	50,000	10,000	10	**99.0**	98.5 [26]
CIFAR-100	50,000	10,000	100	**91.3**	89.3 [26]
Stanford Cars	8,144	8,041	196	94.6	**94.8*** [26]
Oxford Pets	3,680	3,369	37	**95.9**	93.8* [29]
Food-101	75,750	25,250	101	**93.0**	90.4* [30]
FGVC Aircraft	6,667	3,333	100	92.7	**92.9*** [31]
Birdsnap	47,386	2,443	500	**83.6**	80.2* [32]

GPipe: Performance analysis

- **Massive Massively Multilingual Machine Translation**
  - Scaled transformer: (1) depth by increasing number of layers, (2) width by increasing hidden dimensions
  - 1.3B wide model with T(12,16384,32), 1.3B deep model T(24,8192,16)
  - Quality of both model is similar, deeper model outperforms by huge margins on low-resource languages (suggesting that increasing model depth might be better for generalization)

Figure 3: Translation quality across all languages with increasing multilingual model capacity. Languages are arranged in the order of decreasing training dataset size from left to right. $T(L, H, A)$, depicts the performance of a Transformer with L encoder and L decoder layers, a feed-forward hidden dimension of H and A attention heads. We notice that increasing the model capacity, from 400M params ($T(6, 8192, 16)$) to 1.3B ($T(24, 8192, 16)$), and further, to 6B ($T(64, 16384, 32)$), leads to significant quality improvements across all languages. We also notice huge quality improvements for low-resource languages (right side of the plot), when compared against bilingual baselines, highlighting the significant transfer gains resulting from training a multilingual model.

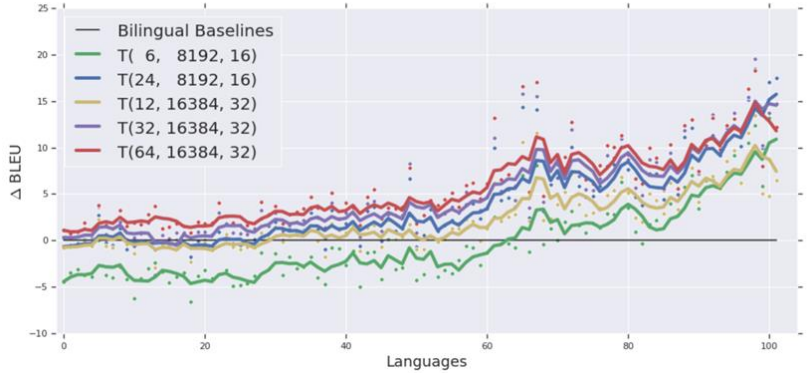


Table 5: The Effect of Batch Size

Batch Size	260K	1M	4M
BLEU	30.92	31.86	32.71
Loss (NLL)	2.58	2.51	2.46

*BLEU (Bilingual Evaluation Understudy) score: 번역에서 사용되는 성능 지표

Thank you