GPipe Analysis

Raghav Mishra

1 First Pass

- It is used mainly for efficient and task-independent model parallelism.
- GPipe, a pipeline parallelism library that allows scaling any network that can be expressed as a sequence of layers.
- Provides the flexibility of scaling a variety of different networks to gigantic sizes efficiently.
- Training large-scale neural networks on two different tasks:
  - (i) Image Classification: Training a 557-million-parameter AmoebaNet model that attains a top-1 accuracy of 84.4% on ImageNet-2012.
  - (ii) Multilingual Neural Machine Translation: Training a single 6-billion-parameter, 128-layer Transformer model on a corpus spanning over 100 languages, achieving better quality than all bilingual models.

2 Second Pass

- GPipe allows scaling arbitrary deep neural network architectures beyond the memory limitations of a single accelerator by partitioning the model across different accelerators and supporting re-materialization on every accelerator.
- Gradient updates using GPipe are consistent regardless of the number of partitions, allowing researchers to easily train increasingly large models by deploying more accelerators.
- Experimental results on two tasks:

(i) Image classification: Training the AmoebaNet model on 480×480 input from the ImageNet 2012 dataset. By increasing the model width, they scale up the number of parameters to 557 million and achieve a top-1 validation accuracy of 84.4%.

(ii) Machine translation: Training a single 128-layer 6-billion-parameter multilingual Transformer model on 103 languages (102 languages to English).

- Model is capable of outperforming the individually trained 350-million-parameter bilingual Transformer Big models on 100 language pairs.
- Introduction of GPipe : a scalable model-parallelism library for training giant neural networks
- Novel batch-splitting pipeline-parallelism algorithm that uses synchronous gradient updates, allowing model parallelism with high hardware utilization and training stability.
- GPipe performance with two very different types of model architectures: an AmoebaNet convolutional model and a Transformer sequence-to-sequence model.
- Study of scalability , efficiency and communication cost
- Amoeba Net : experiments on Cloud TPUv2s 8GB memory accelerator
- GPipe reduces the intermediate activation memory requirements from 6.26GB to 3.46GB.
- Enables a 318M parameter model on a single accelerator
- Trained Transformer model on Cloud TPUv3s with 16GB memory per accelerator core. Fixed vocab size of 32k, seq length of 1024 and batch size of 32.
- Transformer layer has 2048 for model dimension, 8192 for feed-forward hidden dimension and 32 attention heads
- Re-materialization allows training a $2.7\times$ larger model on a single accelerator. With 128 partitions, GPipe allows scaling Transformer up to 83.9B parameters, a $298\times$ increase.
- the number of micro-batches M is at least $4\times$ the number of partitions, the bubble overhead is almost negligible

- Transformer model, there is a 3.5× speedup when it is partitioned across four times more accelerators

Table 1: Normalized training throughput using GPipe with different number of partitions (K) and 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.

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

Table 2: Normalized training throughput using GPipe on GPUs without high-speed interconnect.

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

Table 3: Image classification accuracy using AmoebaNet-B (18, 512) first trained on ImageNet 2012 then fine-tuned on others. Fine-tuned results averaged across 5 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 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]

3 Third Pass

1. GPipe enables large-scale model parallelism using pipeline parallelism with micro-batches.
2. Models are divided into sequential partitions that run on multiple accelerators.
3. Micro-batch scheduling ensures high utilization and reduces idle time.
4. It eliminates the memory constraints of single-device training.
5. GPipe supports scaling models both in width and depth.
6. The approach reduces communication overhead compared to naive model parallelism.
7. GPipe works efficiently with synchronous training.
8. Pipeline parallelism overlaps computation and communication.
9. Checkpoints allow efficient memory use via re-materialization.
10. GPipe scales to very large models with minimal code modifications.
11. Experiments with CIFAR-10 validated the effectiveness of GPipe.
12. AmoebaNet was used as a benchmark model for image classification.
13. Scaling depth and width improved test accuracy significantly.
14. GPipe achieved near-linear speedup with more devices.
15. Partitioning into 4 accelerators improved throughput compared to 2.
16. Even with communication overhead, training remained efficient.
17. AmoebaNet models trained with GPipe reached state-of-the-art accuracy on CIFAR-10.
18. Training larger networks became feasible without GPU OOM errors.
19. Memory savings allowed deeper models to fit within hardware limits.
20. GPipe provides flexibility for both research prototypes and production systems.

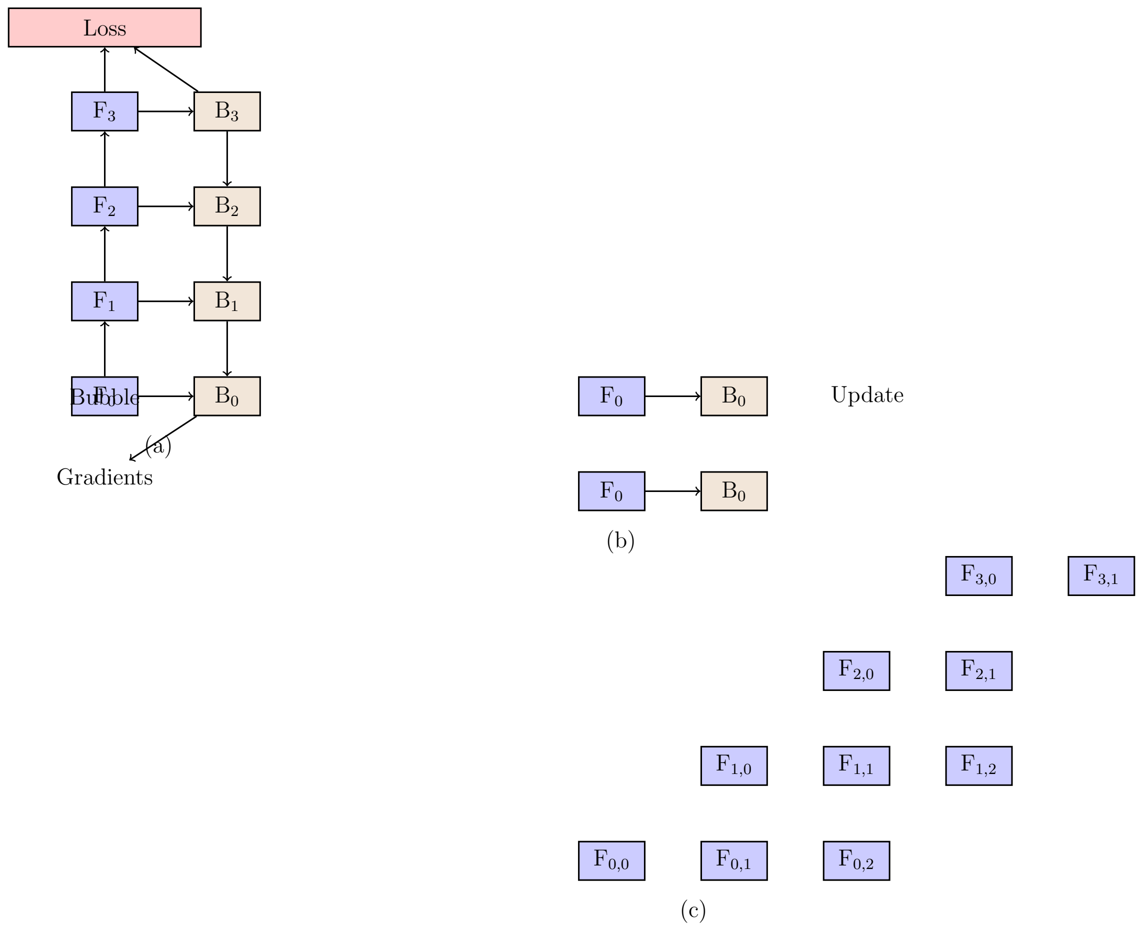
21. GPipe was further applied to large-scale NLP, specifically multilingual machine translation (MMT).
22. The dataset included 25 billion parallel sentences across 102 languages and English.
23. Data spanned both low-resource and high-resource languages.
24. Transformer models were scaled along depth (layers) and width (hidden size, attention heads).
25. The base Transformer Big model had 400M parameters, denoted as $T(6, 8192, 16)$.
26. Scaling produced models of 1.3B, 3B, and 6B parameters.
27. Increasing model capacity improved BLEU scores across all languages.
28. The 1.3B deep model outperformed the wide model for low-resource languages.
29. Depth provided stronger generalization and transfer learning benefits.
30. Larger models yielded diminishing returns beyond 3B parameters.
31. Training instability was observed with deeper models due to sharp activations and noise.
32. Logit clipping and scaled initialization were applied to stabilize training.
33. These methods reduced exploding gradients and preserved convergence.
34. Temperature-based sampling improved multilingual performance.
35. Larger models significantly improved translation for low-resource languages.
36. Bilingual baselines were outperformed by multilingual GPipe models.
37. Model parallelism allowed training across up to 16 accelerators.
38. BLEU scores increased consistently with model size.
39. Data parallelism with very large batch sizes was also explored.
40. Batch sizes scaled from 260K tokens to 4M tokens per step.

41. The largest batch ever reported in NMT (4M tokens) improved BLEU performance.
42. Increasing batch size reduced validation loss significantly.
43. Both low-resource and high-resource language pairs benefited.
44. GPipe's efficiency enabled training such massive batches.
45. Depth scaling was particularly beneficial for scarce languages.
46. Width scaling primarily helped high-resource languages.
47. A trade-off exists: deeper models generalize better, wider models converge faster.
48. Pipeline parallelism enables distributed training with near-linear scaling.
49. The combination of GPipe and Transformer scaling demonstrated new state-of-the-art in MMT.
50. GPipe thus provides a general-purpose solution for scaling deep learning across domains.
51. GPipe enables large-scale model parallelism using pipeline parallelism with micro-batches.
52. Models are divided into sequential partitions that run on multiple accelerators.
53. Micro-batch scheduling ensures high utilization and reduces idle time.
54. It eliminates the memory constraints of single-device training.
55. GPipe supports scaling models both in width and depth.
56. The approach reduces communication overhead compared to naive model parallelism.
57. GPipe works efficiently with synchronous training.
58. Pipeline parallelism overlaps computation and communication.
59. Checkpoints allow efficient memory use via re-materialization.
60. GPipe scales to very large models with minimal code modifications.

61. Experiments with CIFAR-10 validated the effectiveness of GPipe.
62. AmoebaNet was used as a benchmark model for image classification.
63. Scaling depth and width improved test accuracy significantly.
64. GPipe achieved near-linear speedup with more devices.
65. Partitioning into 4 accelerators improved throughput compared to 2.
66. Even with communication overhead, training remained efficient.
67. AmoebaNet models trained with GPipe reached state-of-the-art accuracy on CIFAR-10.
68. Training larger networks became feasible without GPU OOM errors.
69. Memory savings allowed deeper models to fit within hardware limits.
70. GPipe provides flexibility for both research prototypes and production systems.
71. GPipe was further applied to large-scale NLP, specifically multilingual machine translation (MMT).
72. The dataset included 25 billion parallel sentences across 102 languages and English.
73. Data spanned both low-resource and high-resource languages.
74. Transformer models were scaled along depth (layers) and width (hidden size, attention heads).
75. The base Transformer Big model had 400M parameters, denoted as $T(6, 8192, 16)$.
76. Scaling produced models of 1.3B, 3B, and 6B parameters.
77. Increasing model capacity improved BLEU scores across all languages.
78. The 1.3B deep model outperformed the wide model for low-resource languages.
79. Depth provided stronger generalization and transfer learning benefits.
80. Larger models yielded diminishing returns beyond 3B parameters.

81. Training instability was observed with deeper models due to sharp activations and noise.
82. Logit clipping and scaled initialization were applied to stabilize training.
83. These methods reduced exploding gradients and preserved convergence.
84. Temperature-based sampling improved multilingual performance.
85. Larger models significantly improved translation for low-resource languages.
86. Bilingual baselines were outperformed by multilingual GPipe models.
87. Model parallelism allowed training across up to 16 accelerators.
88. BLEU scores increased consistently with model size.
89. Data parallelism with very large batch sizes was also explored.
90. Batch sizes scaled from 260K tokens to 4M tokens per step.
91. The largest batch ever reported in NMT (4M tokens) improved BLEU performance.
92. Increasing batch size reduced validation loss significantly.
93. Both low-resource and high-resource language pairs benefited.
94. GPipe's efficiency enabled training such massive batches.
95. Depth scaling was particularly beneficial for scarce languages.
96. Width scaling primarily helped high-resource languages.
97. A trade-off exists: deeper models generalize better, wider models converge faster.
98. Pipeline parallelism enables distributed training with near-linear scaling.
99. The combination of GPipe and Transformer scaling demonstrated new state-of-the-art in MMT.
100. GPipe thus provides a general-purpose solution for scaling deep learning across domains.

4 GPipe Parallelism Diagrams



(a)

(b)

(c)