Breadth-First Pipeline Parallelism: paper presentation

Aleksandr Algazinov

Tsinghua University, Master in Advanced Computing

Motivation for Selecting the Paper

The paper "Breadth-First Pipeline Parallelism" (MLSys 2023) was chosen for its focus on optimizing the training of large language models (LLMs), which are foundational to key areas of interest:

- **Natural Language Processing (NLP):** LLMs power advanced NLP tasks like translation, summarization, and sentiment analysis. Efficient training is vital for their development.
- **Large Language Models (LLMs):** The paper introduces a novel method to reduce training time, cost, and memory use, directly advancing LLM technology.
- **AI-Generated Content (AIGC):** Faster, more efficient LLM training improves the quality and speed of generating text-based content.
- **Multimodal AI:** The training techniques could potentially extend to models integrating text with images or audio, broadening their impact.

Problem Statement

- **Significance:** Paper introduces Breadth-First Pipeline Parallelism, a novel training schedule that optimizes the combination of pipeline and data parallelism. Key features and results include:
  - Reduces training time, cost, and memory usage
  - Achieves high GPU utilization with small batch sizes per GPU
  - Utilizes fully sharded data parallelism
  - Experimental results: Up to 43% increase in training throughput for a 52B parameter model compared to Megatron-LM
  - This could reduce training time and cost by up to 43% on large GPU clusters

Parallelism in Deep Learning

Pipeline Parallelism

- Vertically splits model across devices
- Each device computes different layer(s)
- Sequential processing of micro-batches
- Requires careful scheduling to minimize idle time
- Example: GPipe, PipeDream

Data Parallelism

- Replicates model across devices
- Each device processes different data subset
- Gradients synchronized across devices
- Scales well with batch size
- Example: Distributed Data Parallel (DDP)

Key Difference

Pipeline parallelism splits the **model** across devices, while data parallelism splits the **data** across devices.

Our quest: high GPU utilization *and* small batch sizes

Problem: Current methods for training large language models need a high batch size per GPU to achieve a high GPU utilization (computational efficiency), yet Stochastic Gradient Descent runs faster with small batch sizes.

Larger batch sizes slow down the convergence of SGD. More training samples are needed to reach the same validation loss.

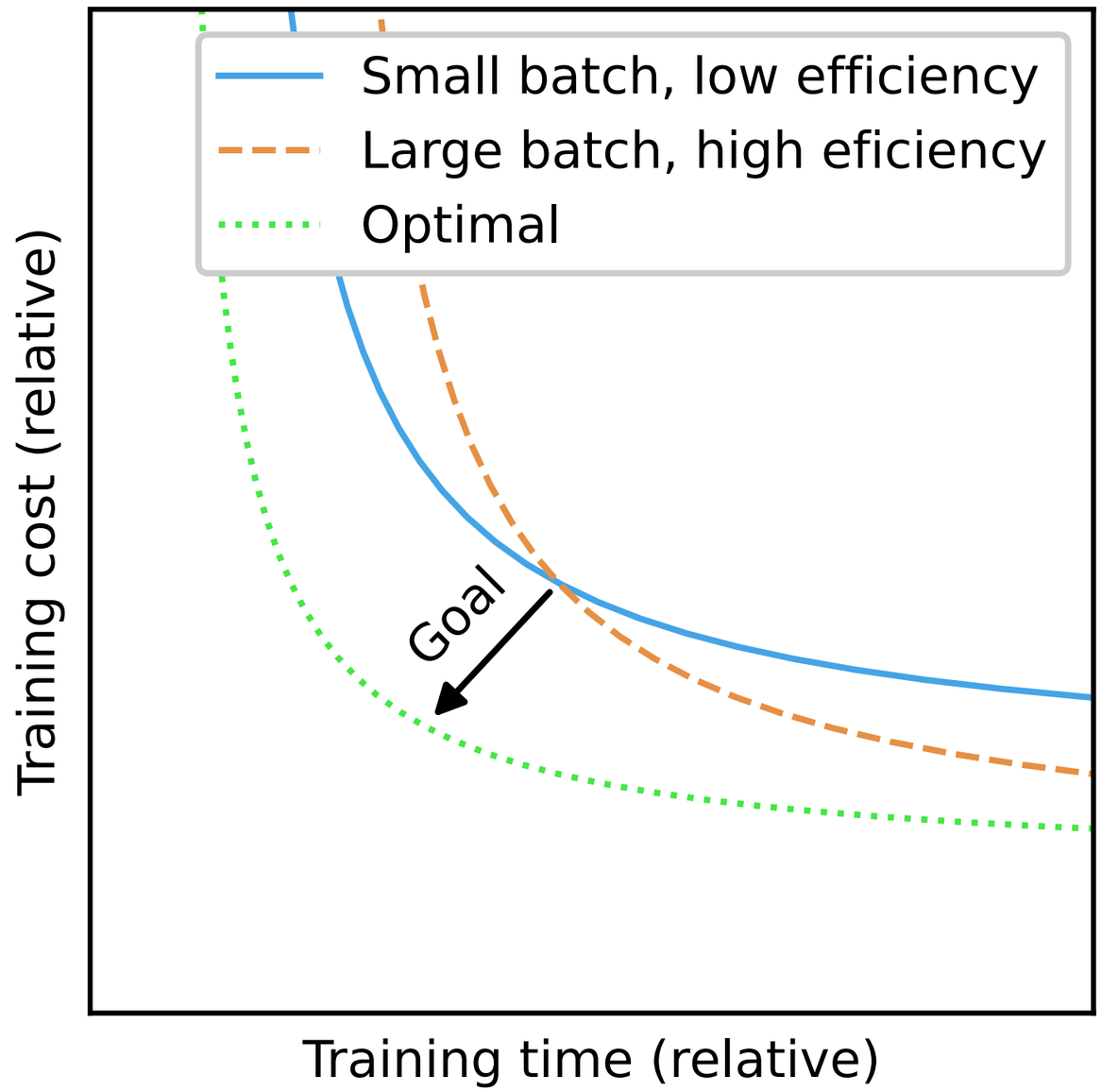
- **Empirical model:** The training length depends on the ratio between the batch size and the empirical critical batch size:

$$\text{Samples} \propto 1 + \frac{\text{Batch Size}}{\text{Critical Batch}}$$

- **Scaling the cluster:** When scaling the cluster, the GPU utilization mainly depends on the batch size per GPU β:

$$\text{Cost} \propto \text{Utilization}^{-1}(\beta)\left(1 + \beta\frac{\text{Num GPUs}}{\text{Critical Batch}}\right),$$
$$\text{Time} \propto \frac{\text{Cost}}{\text{Num GPUs}}.$$

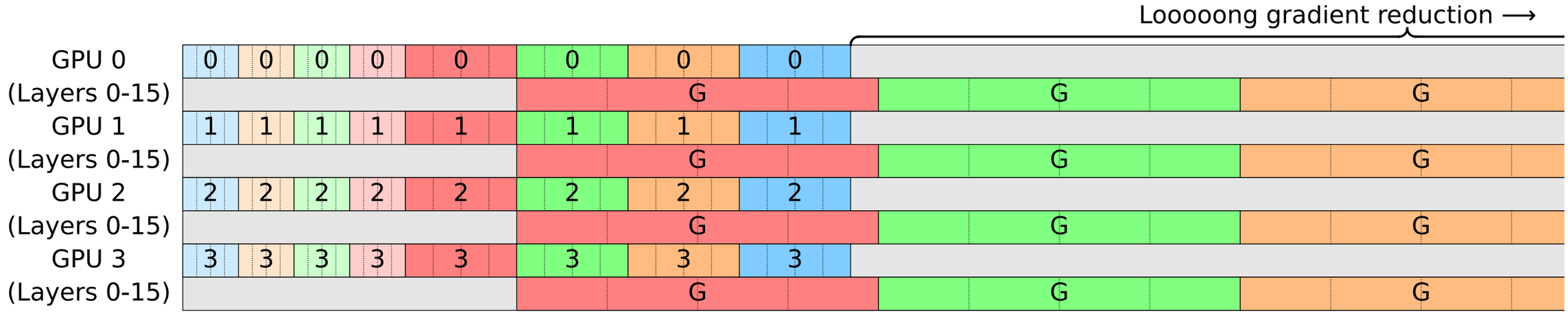
- **Trade-off:** The training time and cost cannot be minimized together. We want to mitigate the trade-off by maximizing the GPU utilization for a small batch size per GPU.



The good old methods won't do!

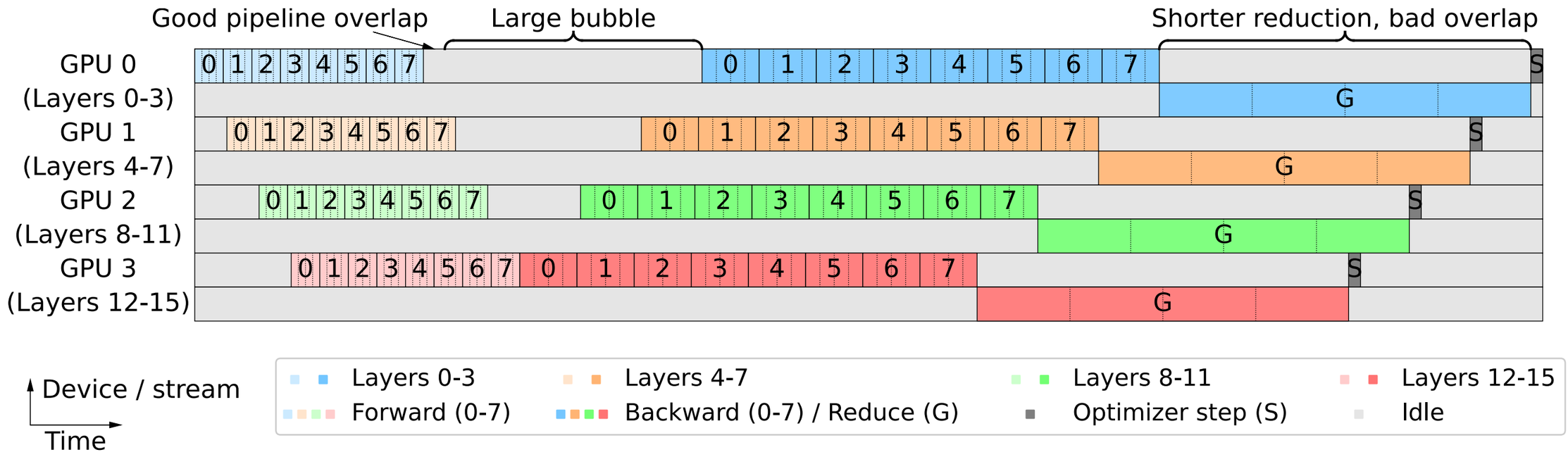
Data parallelism: needs help...

At small batch sizes, data-parallel training is bottlenecked by the long gradient reduction.



Pipeline parallelism: still struggling...

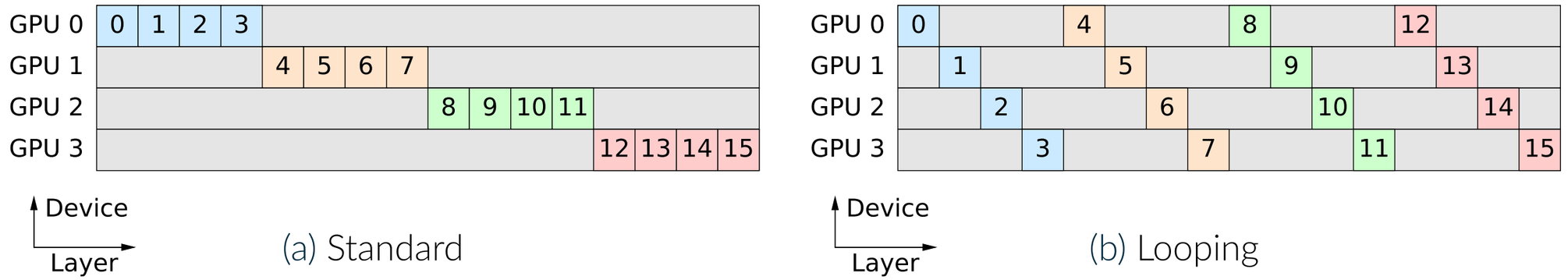
At small batch sizes, adding pipeline parallelism (GPipe or 1F1B) leads to a large pipeline bubble and poor gradient reduction overlap.



Bending the pipes

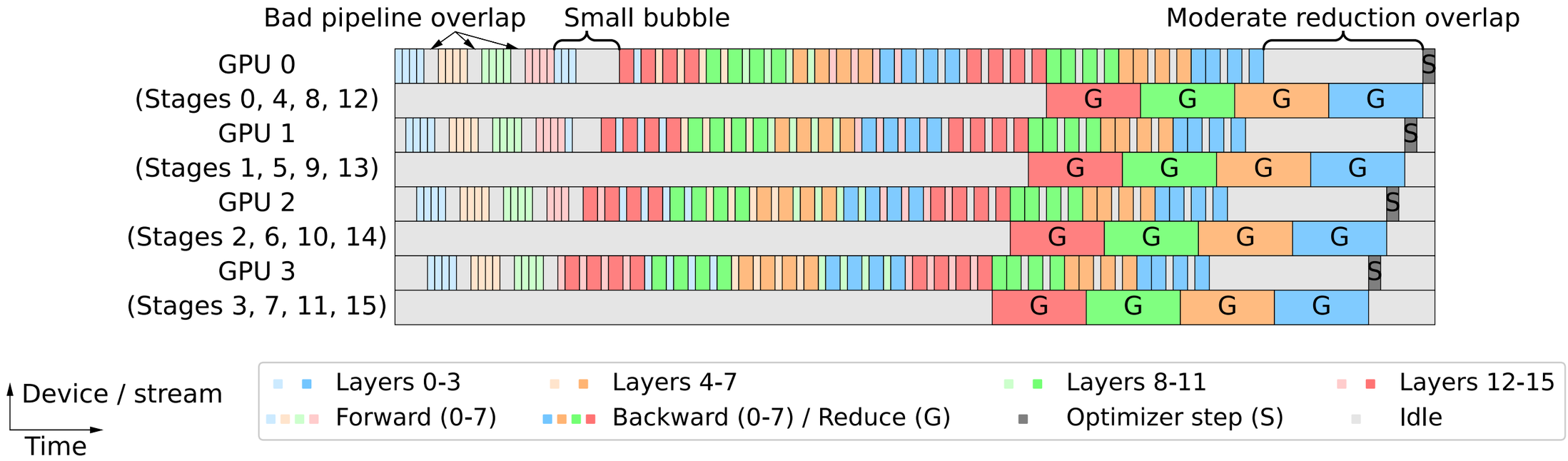
The solution: looping the pipeline

We replace the few large stages by many small stages, looping around the pipeline multiple times. This allows for a smaller pipeline bubble, even with a small batch size. This comes at the cost of extra pipeline-parallel communication



Depth-first (interleaved): almost there!

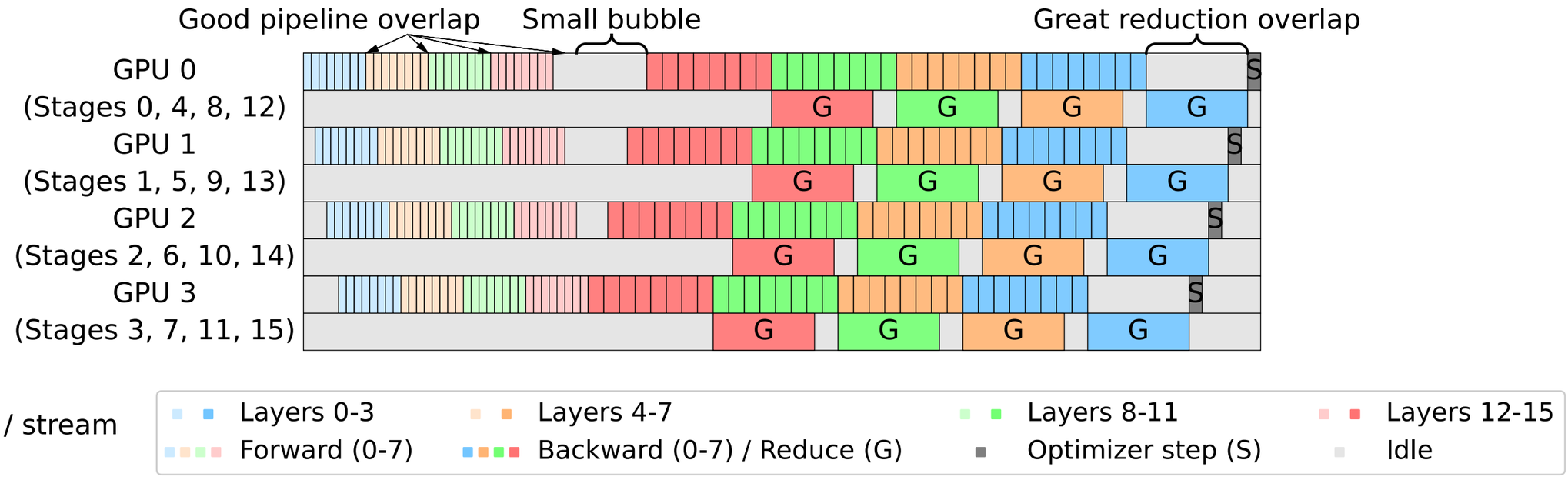
The depth-first schedule (Megatron-LM), running earlier micro-batches first, shrinks the bubble but has a limited data- and pipeline-parallel network overlap.



Our method: Breadth-First Pipeline Parallelism

Breadth-first schedule: that's the one!

A breadth-first, running earlier stages first, schedule keeps the small bubble but has a great data- and pipeline-parallel network overlap.

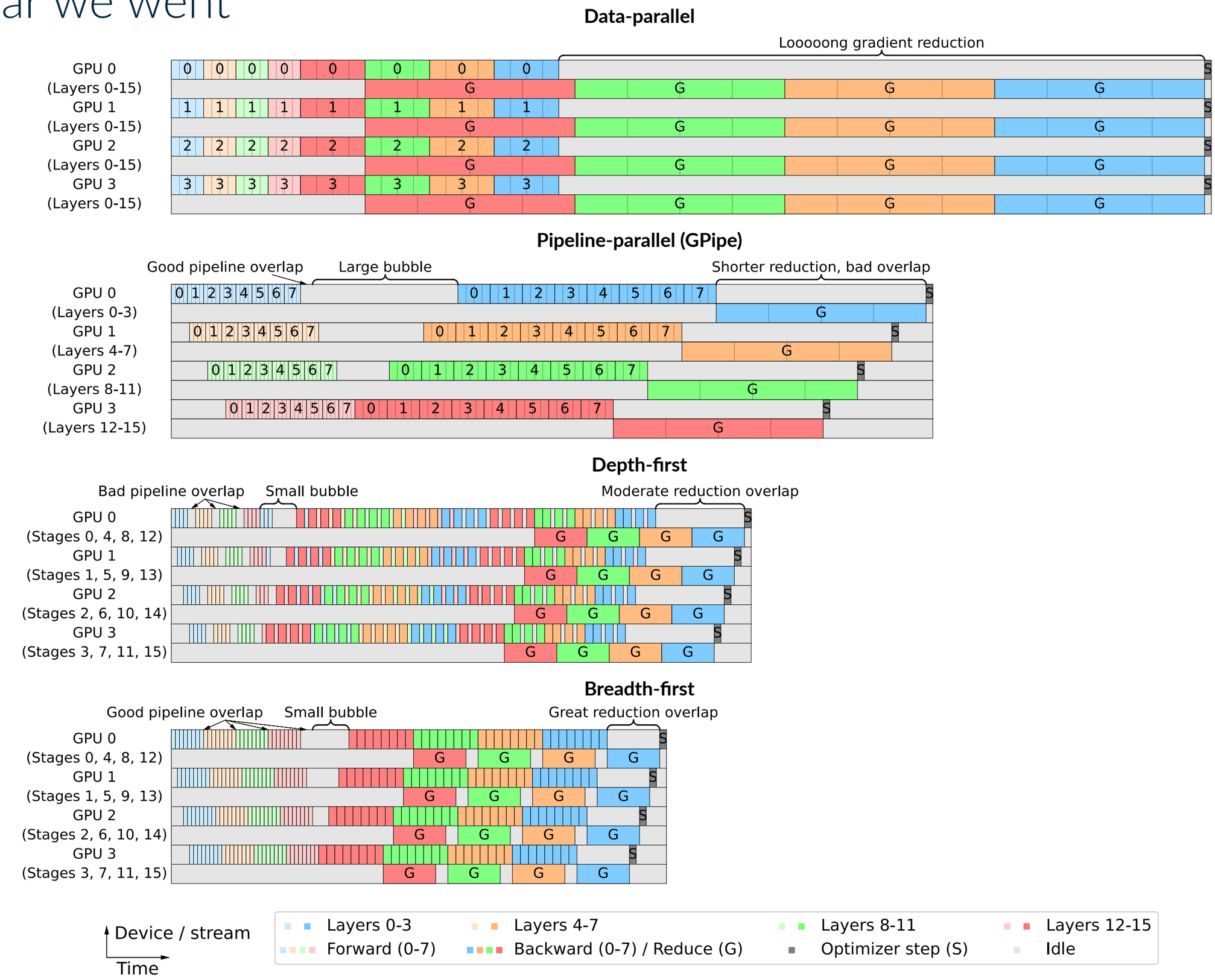


How about memory?

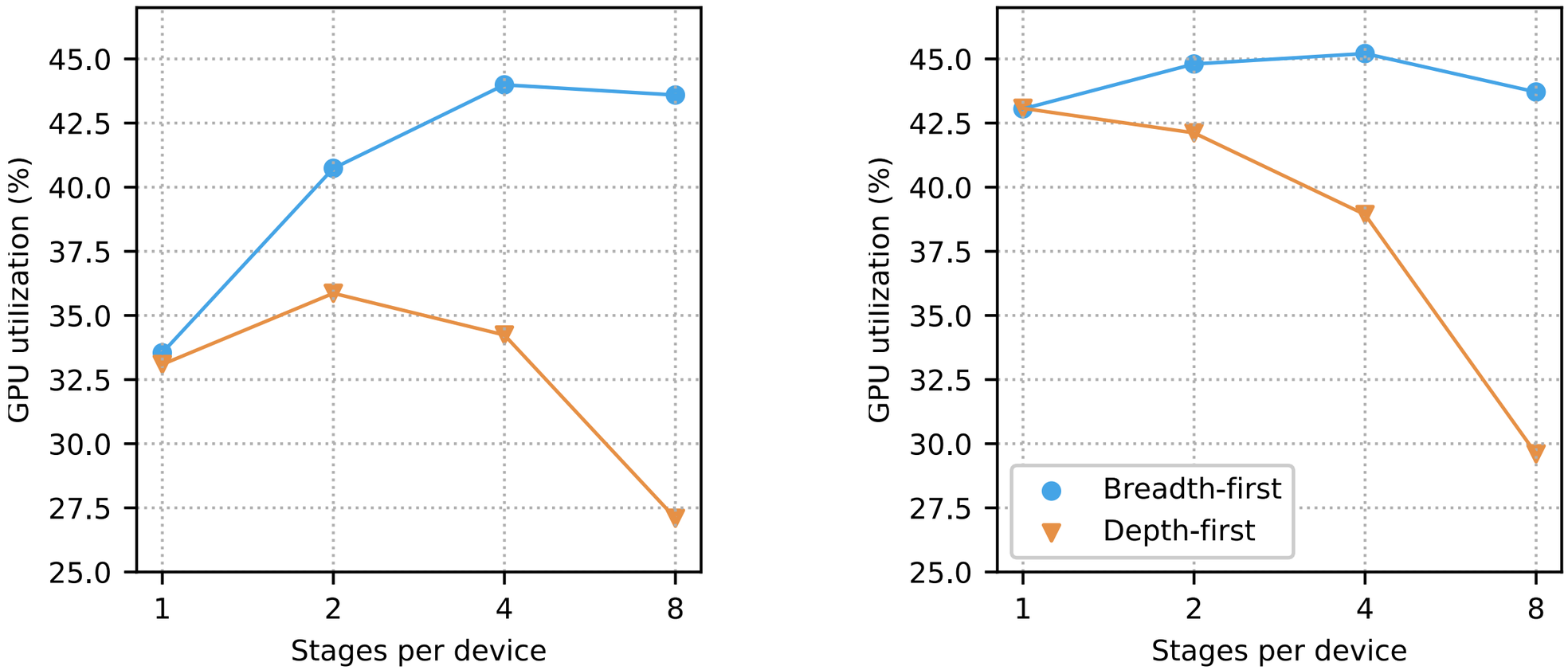
For small batch sizes, our method has the lowest memory usage of all pipeline-parallel methods, providing extra flexibility for choosing better training configurations:

- **Weights, gradients and training state:** Unlike other pipeline-parallel methods, Breadth-First pipeline parallelism combines well with Fully Sharded Data-Parallel (ZeRO-3). This allows training very large models with small pipelines.
- **Activations and checkpoints:** At small batch sizes, all pipeline-parallel methods use the same activation memory.

How far we went



Breadth-first loops better



(a) Batch size 16

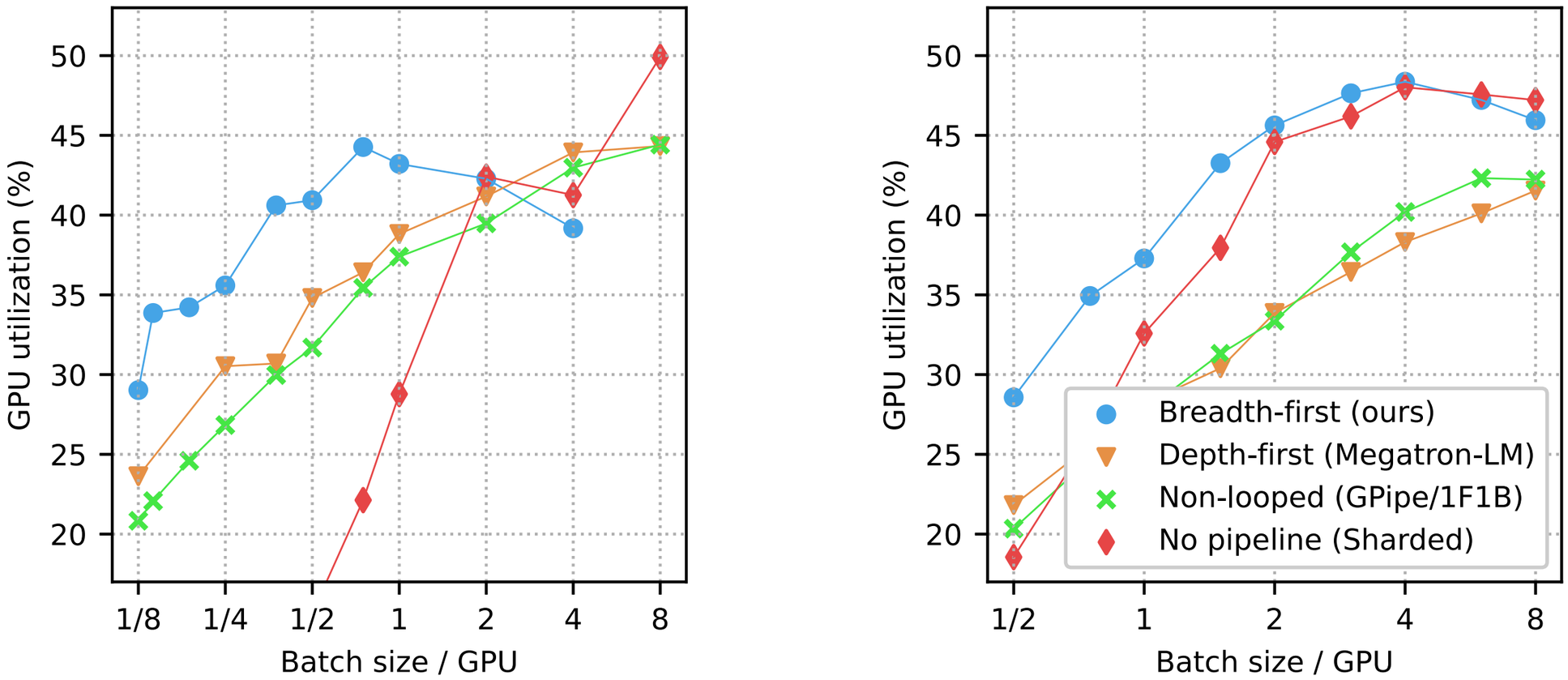
(b) Batch size 64

Figure 2. Comparison of looping schedule efficiencies for different number of loops. Both methods help with the pipeline bubble, which is higher for small batch sizes, but the depth-first does it at the expense of network overhead. (52 B model, TP = PP = 8, DP = 1, micro-batch size = 1)

Setup

We tested our methods for two models, with 52 and 6.6 billion parameters, on a cluster of 8 Nvidia DGX servers (64x V100-32GB GPUs) connected with InfiniBand. We used our custom implementation when possible (breadth-first, GPipe non-pipelined), otherwise we used Megatron-LM (depth-first, 1F1B).

Breadth-first is better at small batch sizes



(a) 52 B model

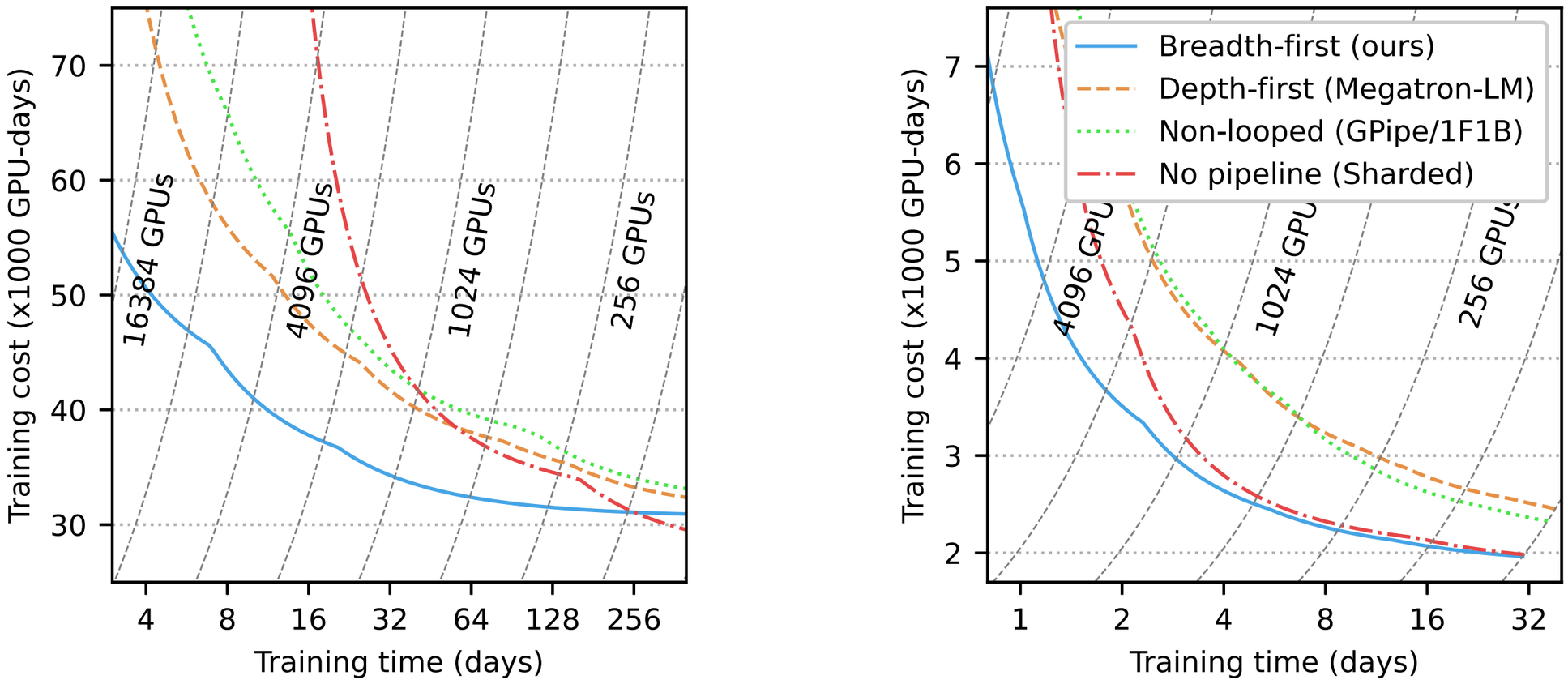
(b) 6.6 B model

Figure 3. Comparison the efficiency of each method as a function of the batch size (per GPU). Each data point represents an optimal configuration found through an extensive search over the configuration space. Breadth-First Pipeline Parallelism outperforms other methods for smaller batch sizes.

Setup

We tested our methods for two models, with 52 and 6.6 billion parameters, on a cluster of 8 Nvidia DGX servers (64x V100-32GB GPUs) connected with InfiniBand. We used our custom implementation when possible (breadth-first, GPipe non-pipelined), otherwise we used Megatron-LM (depth-first, 1F1B).

Breadth-first trains faster and for cheaper



(a) 52 B model

(b) 6.6 B model

Figure 4. Breadth-First Pipeline Parallelism outperforms other methods for smaller batch sizes per GPU, resulting in smaller training times and costs.

Setup

We tested our methods for two models, with 52 and 6.6 billion parameters, on a cluster of 8 Nvidia DGX servers (64x V100-32GB GPUs) connected with InfiniBand. We used our custom implementation when possible (breadth-first, GPipe non-pipelined), otherwise we used Megatron-LM (depth-first, 1F1B).

Conclusion

- Breadth-First Pipeline Parallelism reduces training time and cost for large language models
- Minimizes network overhead and pipeline bubble (period during computation where some stages of the pipeline are idle, waiting for others to complete their tasks.)
- Compatibility with DPFS (Data Parallel Fully Sharded) enables:
  - More efficient configurations
  - Training of much larger models (tens to hundreds of trillions of parameters)

Limitations and Future Work (discussed in the paper)

- **Limitations:**
  - Optimized for small N_{PP} and β, may not scale to large configurations [1]
  - Untested on newer hardware (A100, H100) and large clusters [1]
  - Fundamental challenges remain for extremely large models [1]
- **Future Work:**
  - Evaluate on larger models and modern hardware (A100, H100) [1]
  - Integrate with Flash Attention [1,2], sequence parallelism [1,3], and selective activation recomputation [1,3]

Limitations and Future Work (my thoughts)

Limitations:

- **Generalizability to Different Model Architectures:** The method may not perform as well for non-transformer models like CNNs or RNNs, having different computational patterns.
- **Hyperparameter Sensitivity:** The method's efficiency depends on specific hyperparameter settings, which may require extensive tuning and expertise to optimize.

Additional Future Work Directions:

- **Application to Other Domains:** Exploring the method's use in fields like CV or RL could reveal new optimization opportunities.
- **Automated Hyperparameter Tuning:** Developing tools to automatically adjust hyperparameters could make the method more accessible and easier to apply.
- **Support for Diverse Hardware:** Extending the method to work efficiently on various hardware types, such as TPUs or CPUs, could broaden its applicability.

References

1 Lamy-Poirier, J. "Breadth-First Pipeline Parallelism." Proceedings of Machine Learning and Systems, 5, 2023.

2 Dao, T., et al. "FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness." arXiv preprint arXiv:2205.14135, 2022.

3 Korthikanti, V., et al. "Reducing Activation Recomputation in Large Transformer Models." arXiv preprint arXiv:2205.05198, 2022.

Thank you for your attention!