Breadth-First Pipeline Parallelism

Joel Lamy-Poirier

ServiceNow Research, Montreal, Canada

June 6, 2023

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.

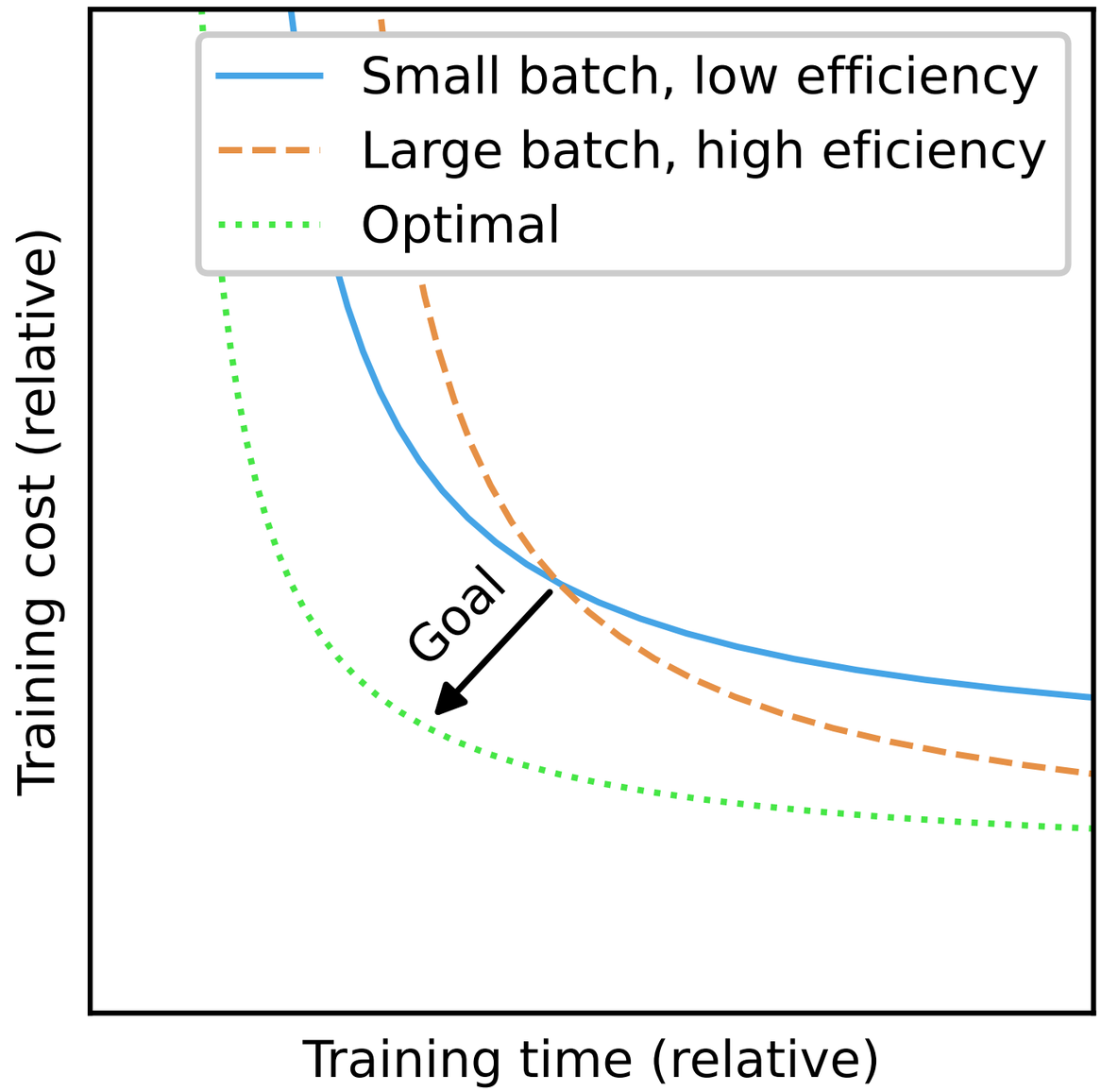
- **Empirical model:** The training length depends on the ratio between the batch size and the empirical critical batch size:

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

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

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

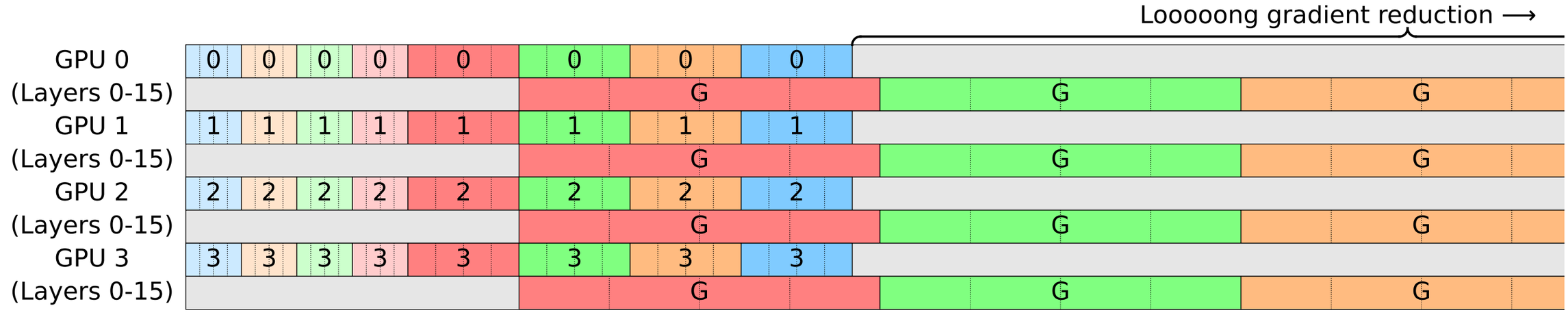
- **Trade-off:** The training time and cost cannot be minimized together. We want to mitigate the trade-off by maximizing the GPU utilization for a small batch size per GPU.



The good old methods won't do!

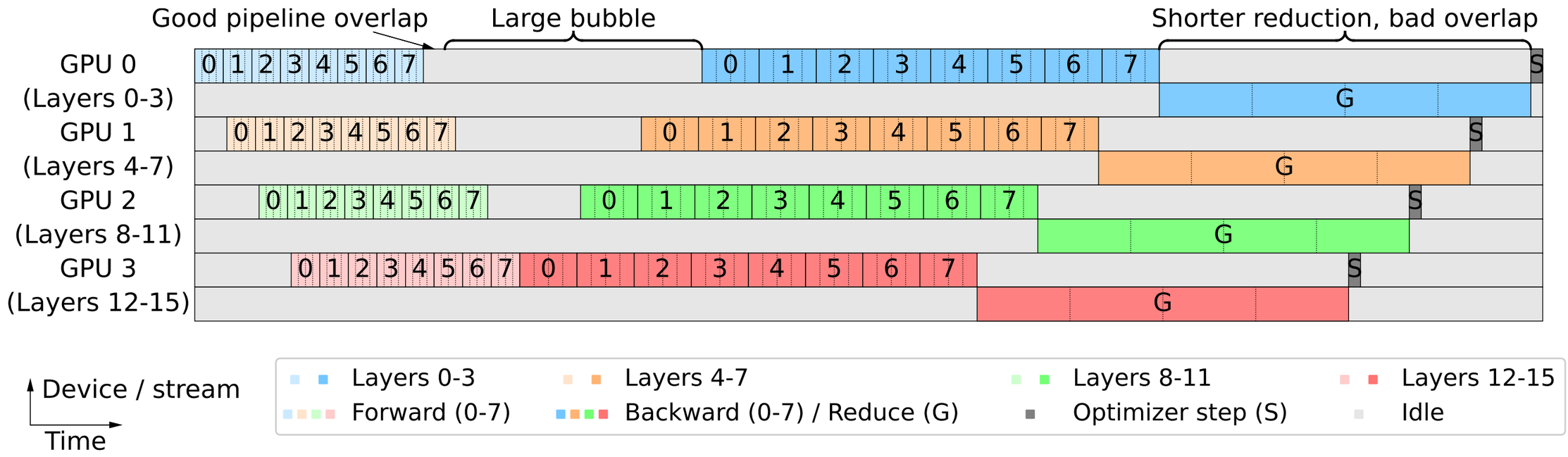
Data parallelism: needs help...

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



Pipeline parallelism: still struggling...

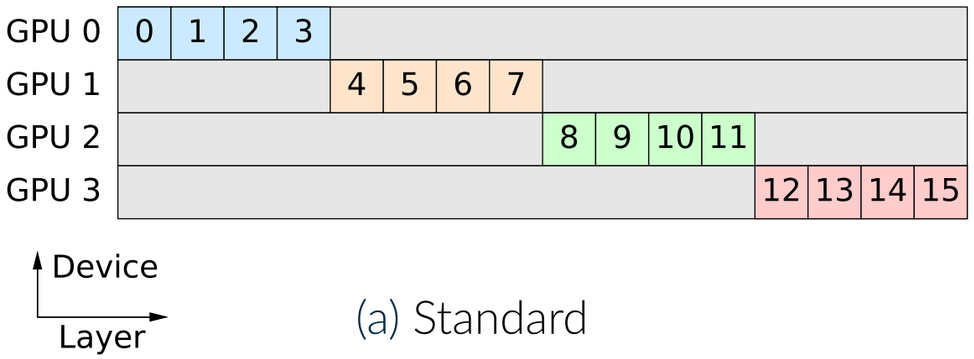
At small batch sizes, adding pipeline parallelism (GPipe or 1F1B) leads to a large pipeline bubble and poor gradient reduction overlap.



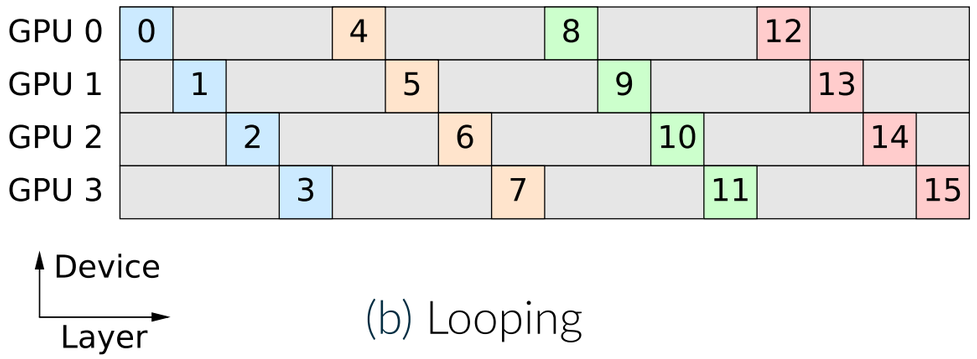
Bending the pipes

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



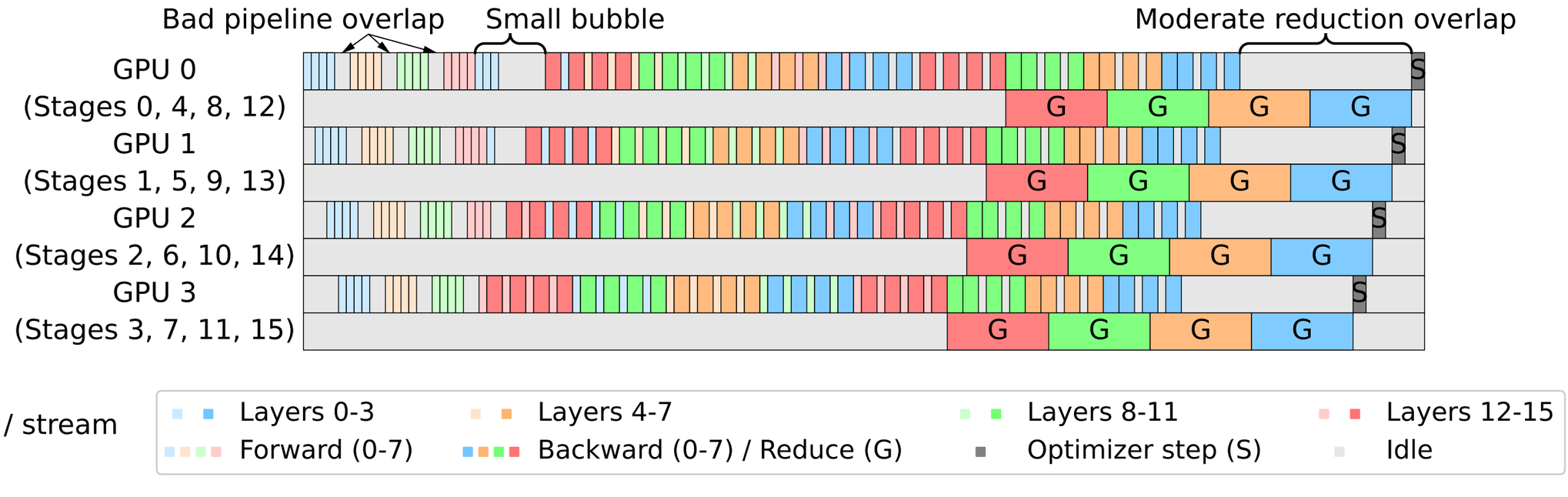
(a) Standard



(b) Looping

Depth-first (interleaved): almost there!

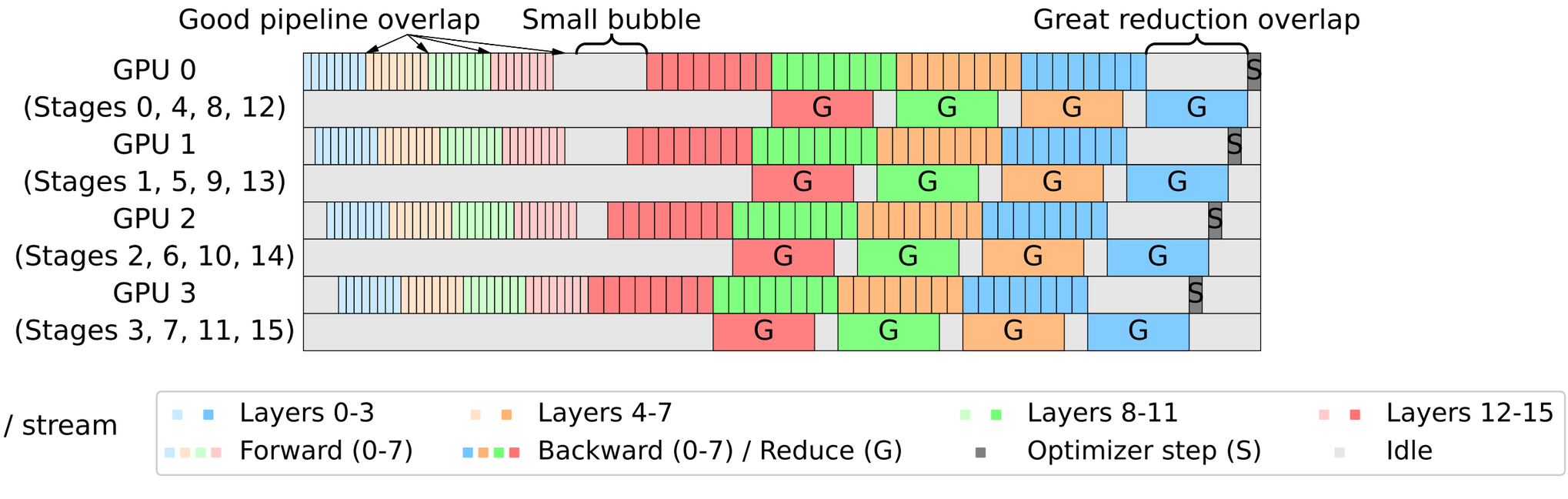
The depth-first schedule (Megatron-LM), running earlier micro-batches first, shrinks the bubble but has a limited data- and pipeline-parallel network overlap.



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.

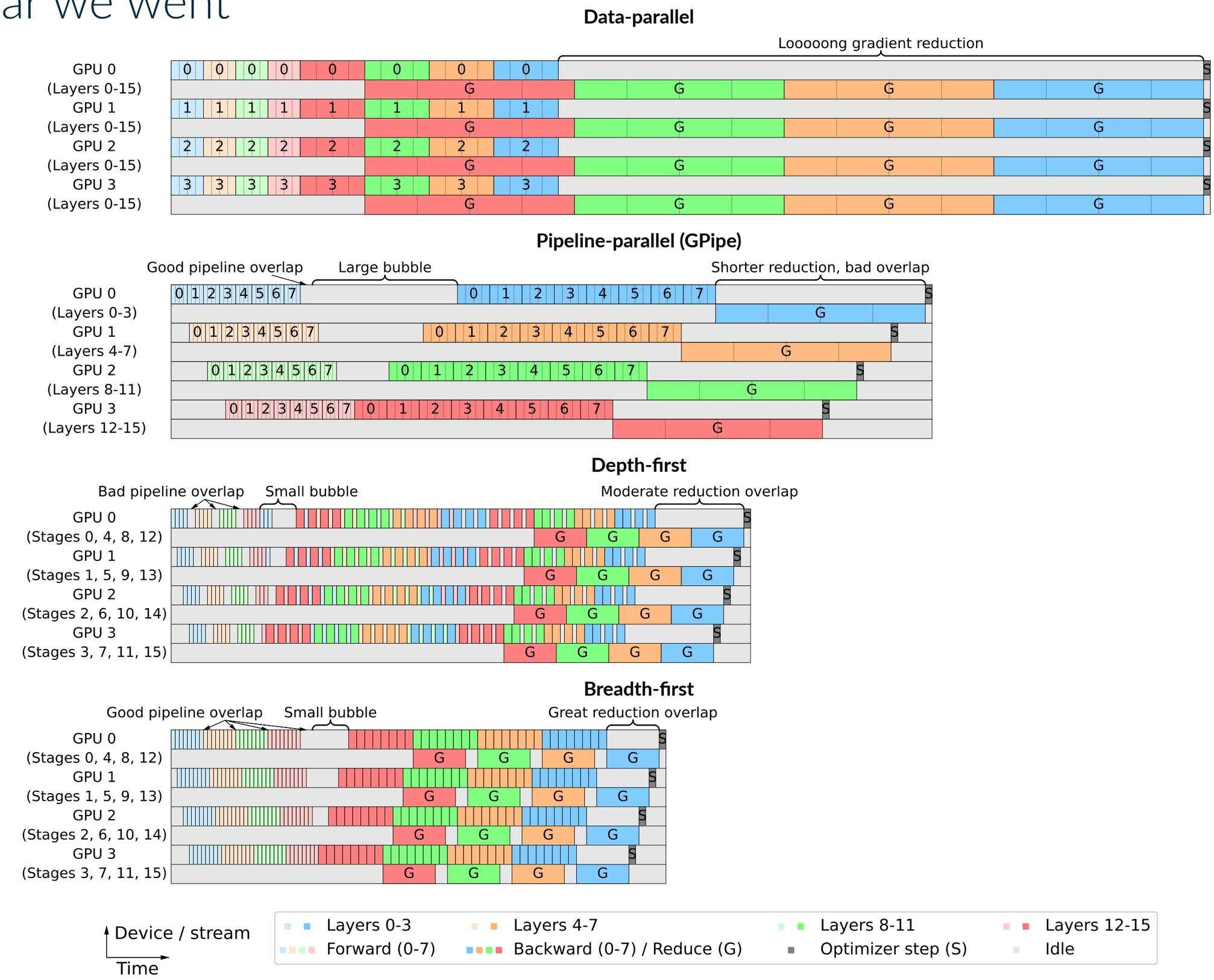


How about memory?

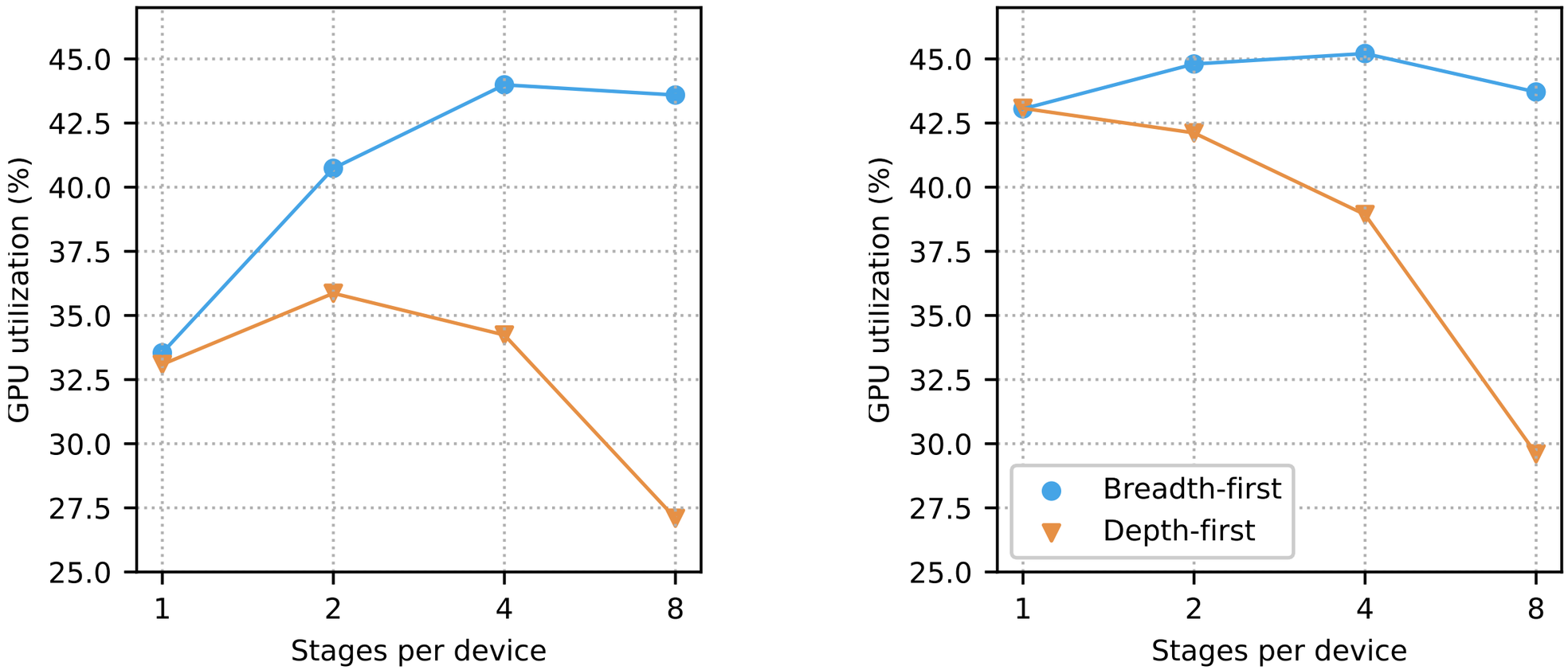
For small batch sizes, our method has the lowest memory usage of all pipeline-parallel methods, providing extra flexibility for choosing better training configurations:

- **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

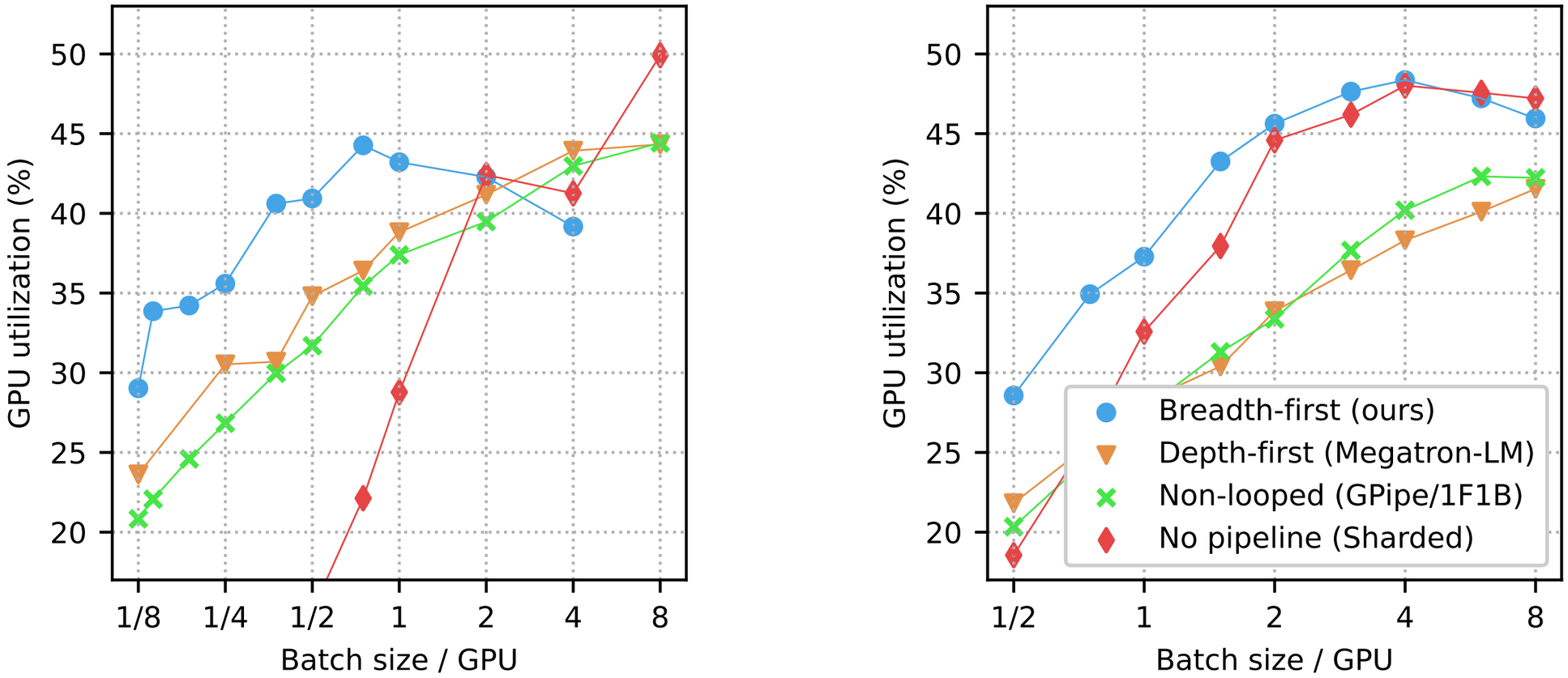
(b) Batch size 64

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

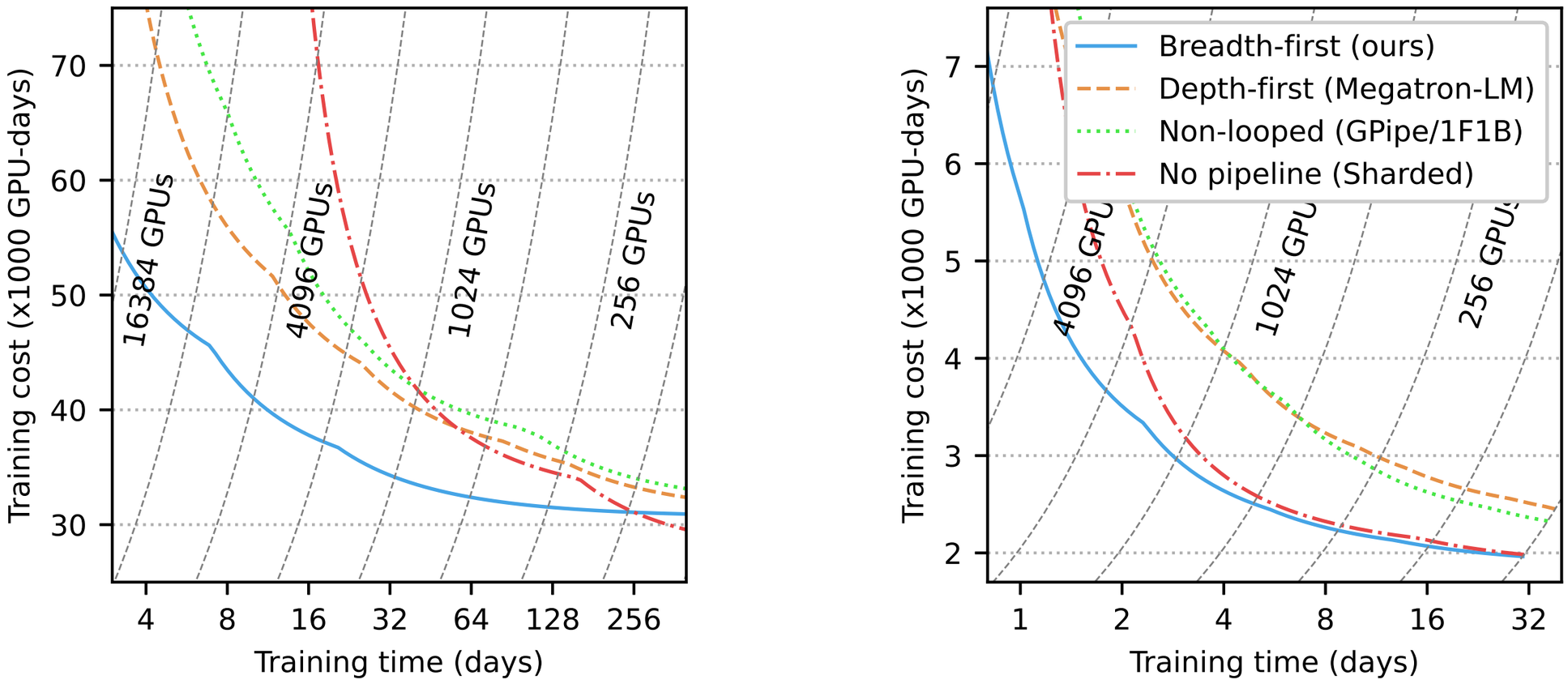
(b) 6.6 B model

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).