A. Planner, Additional Details

For every possible configuration of width and depth, PipeDream-2BW's planner explores the benefit of pipelining and each space-saving optimization. For example, with activation recomputation as a target memory-savings optimization, PipeDream-2BW considers three possible executions:

- Model and data parallelism without pipelining (with the largest per-GPU microbatch size that fits in memory).
- Hybrid parallelism with pipelining and without activation recomputation (all required weight versions and activation stashes in memory for in-flight microbatches).
- Hybrid parallelism with pipelining and recomputation.

PipeDream-2BW's planner estimates the throughput and memory footprint of each of these possible executions using a cost model. PipeDream-2BW's planner then tries to find the configuration with highest throughput that also fits in main device memory of the accelerators used (memory capacity provided as input). In this section, we show one such cost model for throughput and memory.

A.1. Closed-Form Cost Functions

In our experiments, we used profile-based cost functions that run configurations end-to-end for a couple of hundred iterations. However, performance of different parallel configurations can also be estimated using closed-form expressions that use more fine-grained profile information (e.g., time and memory footprint of each transformer block). We present one such cost model here.

A.1.1. THROUGHPUT(.) COST FUNCTION

The throughput of various hybrid-parallel setups with and without pipelining can be modeled using the times of forward and backward passes obtained from a simple profiling step. Let b be the largest per-GPU microbatch size without additional weight and activation versions, and b' be the largest per-GPU microbatch size that can fit on the device when multiple versions are needed ($b' \leq b$). As before, w and d are the pipeline width and depth.

Let $T_i^{\text{comp}}(b, w, d)$ represent the compute time of stage i with a per-GPU microbatch size b, $T_{i \rightarrow j}^{\text{comm}}(b, w, d)$ represent the communication time of activations and gradients between stages i and j with microbatch size b, and $T_i^{\text{comm}}(b, w, d)$ represent the communication time of exchanging gradients between w replicas of stage i with microbatch size b. We assume that the global batch size used is B. With pipeline width w and microbatch size b, data-parallel communication is required every $m(b) = B/(w \cdot b)$ microbatches.

Then, without pipelining, each microbatch of size b takes the following computation time, t:

$$t = \sum_i \max(T_i^{\text{comp}}(b, w, d) + \sum_j T_{j \rightarrow i}^{\text{comm}}(b, w, d), \frac{1}{m(b)} \cdot T_i^{\text{comm}}(b, w, d))$$

With pipelining, computation of different stages can be overlapped. A microbatch of size b' can then be processed every t seconds, where t is given by the expression:

$$t = \max_i \max(T_i^{\text{comp}}(b', w, d) + \sum_j T_{j \rightarrow i}^{\text{comm}}(b', w, d), \frac{1}{m(b')} \cdot T_i^{\text{comm}}(b', w, d))$$

With activation recomputation, the number of floating point operations increases, since forward passes need to be repeated to recompute the activation stashes needed in the backward pass. We use a constant multiplier c^{extra} to represent this. $c^{\text{extra}} = 4/3$ is a reasonable value for this constant, since the backward pass typically takes twice as long as the forward pass. c^{extra} can also be measured empirically. Arithmetic intensity might also increase, which is captured by $T_i^{\text{comp}}(.)$ being a function of the microbatch size b. Communication time remains unchanged from before. Every b inputs can now be processed in time t, where t is given by,

$$t = \max_i \max(c^{\text{extra}} \cdot T_i^{\text{comp}}(b, w, d) + \sum_j T_{j \rightarrow i}^{\text{comm}}(b, w, d), \frac{1}{m(b)} \cdot T_i^{\text{comm}}(b, w, d))$$

The throughput in samples per second of each of these setups is then the corresponding per-GPU microbatch size (b or b') divided by t.

Estimating $T^{\text{comp}}(.)$. $T_i^{\text{comp}}(b, w, d)$ is the compute time of stage i with per-GPU microbatch size b, and can be computed by summing up the forward and backward pass times of all blocks within the stage. If the depth of the pipeline is d and the total number of blocks in the model is B, then the total number of blocks in a given stage is B/d. Forward and backward pass times for each stage can be estimated by profiling 100–200 iterations of training.

Estimating $T^{\text{comm}}(.)$. Communication times can be similarly modeled. Let the size of the associated parameter with B total blocks be $|W|$, and the size of the block's input and output activations be $|A^{\text{inp.+out.}}(b)|$. With a pipeline of depth d, each pipeline stage has $1/d$ of the total model parameters.

Algorithm 1 Partitioning Algorithm

Input: Model m, memory capacity M, m's associated search function SEARCH(.), m's associated throughput cost function THROUGHPUT(.), m's memory footprint cost function MEMORY(.), maximum safe batch size B.
Return: Optimal width and depth w^{opt} and d^{opt}, optimal per-GPU microbatch size b^{opt}, boolean whether activations should be recomputed r^{opt}, optimal degree of gradient accumulation g^{opt}.

Initialize $t^{\text{max}} = 0, w^{\text{opt}} = \text{NULL}, d^{\text{opt}} = \text{NULL}$
for $w = 1$ **to** N **do**
  for $d = 1$ **to** N/w **do**
    // For given width w, depth d, and batch size B, find optimal microbatch size and whether activation recomputation should be performed.
    $b, r = m.\text{SEARCH}(w, d, B)$

    $t = m.\text{THROUGHPUT}(w, d, b, r)$
    if $m.\text{MEMORY}(w, d, b, r) > M$ **then**
      continue
    end if
    if $t > t^{\text{max}}$ **then**
      $t^{\text{max}} = t, w^{\text{opt}} = w, d^{\text{opt}} = d, b^{\text{opt}} = b, r^{\text{opt}} = r$
    end if
  end for
end for
$g^{\text{opt}} = B/(N \cdot b^{\text{opt}})$ *// To reach batch size B.*

The time to communicate activations across stages can be computed as (factor of 2 for gradients in the backward pass),

$$T^{\text{comm}}_{i \to j}(b, w, d) = \frac{2|A^{\text{inp.+out.}}(b)| \cdot \mathbb{I}(d > 1)}{\text{bwdth}_{\text{depth}}(d)}$$

The time to communicate weight gradients across stage replicas can be computed similarly given a bandwidth function $\text{bwdth}_{\text{width}}(w)$, and the number of bytes communicated during all-reduce. The number of byes communicated in an all-reduction can either be explicitly measured, or estimated using a closed-form expression (Narayanan et al., 2019).

$\text{bwdth}_{\text{depth}}(d)$ and $\text{bwdth}_{\text{width}}(w)$ represent the bandwidths for inter-stage and intra-stage communication. These bandwidth functions can respect hierarchical network topologies. For example, if w is less than the number of workers in a single server, communication can be performed entirely within a server, using the higher intra-server bandwidth.

$$\text{bwdth}_{\text{width}}(w) = \begin{cases} B_{\text{high}} \text{ if } w < \text{number of GPUs in server} \\ B_{\text{low}} \text{ otherwise} \end{cases}$$

A.1.2. MEMORY(.) COST FUNCTION

The memory footprint can similarly be modeled using the sizes of activations and weights obtained from a profiling step. Let the total size of the weight parameters for the entire model be $|W|$, let the total size of the activations given a microbatch size b for the entire model be $|A^{\text{total}}(b)|$, and let the size of the input activations for a single stage be $|A^{\text{input}}(b)|$. With a pipeline of d stages, each pipeline stage has weight parameters of size $|W|/d$, and activations of size $|A^{\text{total}}(b)|/d$.

Without Activation Recomputation. As discussed in §3.1, `2BW` maintains 2 different versions of the weight parameters. PipeDream-`2BW` also maintains d versions of activations (the total number of in-flight activations). This means the total PipeDream-`2BW` memory footprint is:

$$\frac{2|W|}{d} + \frac{d|A^{\text{total}}(b)|}{d} + d|A^{\text{input}}(b)|.$$

With Activation Recomputation. With activation recomputation, the total number of activation versions in GPU memory at any point in time is 1. This means that the PipeDream-`2BW` memory footprint with d stages is:

$$\frac{2|W|}{d} + \frac{|A^{\text{total}}(b)|}{d} + d|A^{\text{input}}(b)|.$$

A.2. Partitioning Algorithm

We show pseudocode for the full partitioning algorithm in Algorithm 1.

B. Evaluation, Additional Graphs

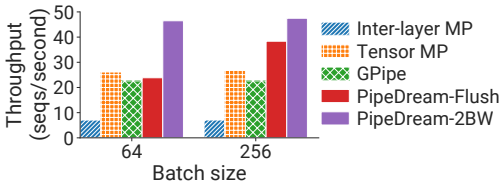
In this section, we present additional results we could not fit in the main paper due to space.

B.1. Throughput and Memory Footprint with BERT Models

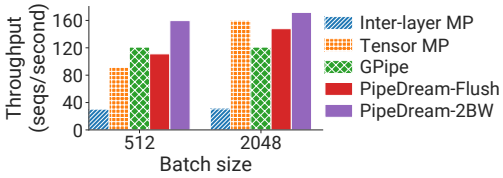
We also ran PipeDream-`2BW` on two BERT models: one with 2.2 billion parameters, and another with 3.8 billion parameters. Figure 10 compares PipeDream-`2BW`'s throughput against the same baselines as before, and Figure 11 compares PipeDream-`2BW`'s memory footprint for these BERT models. We see that results are similar to GPT. One point of difference is that GPipe does not run out of memory at the batch size of 64 (for GPT, only a batch size of 32 fits in memory, leading to a larger pipeline bubble); however, GPipe still has higher memory footprint compared to all other baselines.

B.2. Impact of Activation Recomputation

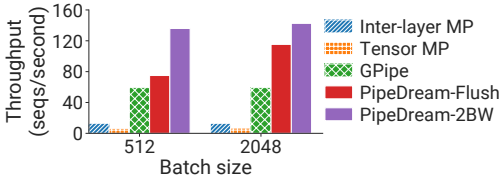
Figure 12 shows the effect of activation recomputation on throughput for various GPT models. For a given per-



(a) BERT, 2.2B, 8-way model parallelism (8×V100s).



(b) BERT, 2.2B, 8-way model parallelism (64×V100s).



(c) BERT, 3.8B, 16-way model parallelism (64×V100s).

Figure 10. Throughput of various systems for different batch sizes for BERT models. Results are shown with a single 8×V100 server, and with eight 8×V100 servers (with 16GB).

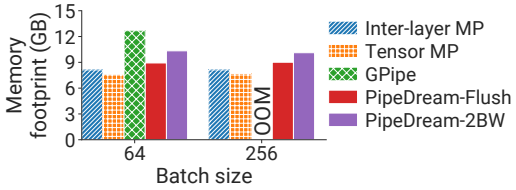
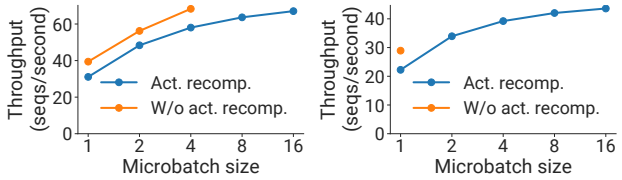


Figure 11. Worst-case memory footprint (in GB) of various systems with 8 V100 GPUs for a BERT model with 2.2B parameters.



(a) GPT, 1.3B. **(b)** GPT, 2.2B.

Figure 12. Throughput of $(1, 8)$ PipeDream-`2BW` configurations vs. per-GPU microbatch size for GPT models using a maximum sequence length of 512 and 8 16-GB-V100 GPUs, with and without activation recomputation. Activation recomputation helps increase the maximum per-GPU microbatch size that fits, especially for larger models, leading to higher throughput in some cases.

GPU microbatch size, recomputation introduces overhead (capped at 33% since the backward pass takes twice as long as the forward pass for most operators). However, recomputation allows for a larger per-GPU microbatch to fit on the worker, sometimes leading to higher throughput than without activation recomputation: activation recomputation leads to higher throughput in Figure 12b, but not in Figure 12a. In the extreme case (not pictured), recomputation makes it possible to train large models by reducing peak memory footprint of training, at the cost of extra compute operations due to an extra forward pass.