A FLOPs Calculation

For FLOPs calculations, we follow the derivation from Narayanan, et.al.(Narayanan et al., 2021) and only consider the matrix multiplications (GEMMs) which are the main contributors to the number of floating-point operations. For the attention block, the main contributors to floating-point operations are: key, query, and value transformation ($6Bsh^2$ operations), attention matrix computation ($2Bs^2h$ operations), attention over values ($2Bs^2h$ operations), and post-attention linear projection ($2Bsh^2$ operations) where B is the microbatch size.

For the feed-forward network that increases the hidden size to $4h$ and then reduces it back to h, we have $16Bsh^2$ floating-point operations. Summing these together, each transformer layer results in $24Bsh^2 + 4Bs^2h$ FLOPs for the forward pass. The other main contributor to the number of floating-point operations is the logits layer in the language model head, which transforms features of dimension h to the vocabulary dimension v. The required FLOPs for this operation is $2Bshv$.

The backward pass requires double the number of FLOPs since we need to calculate the gradients with respect to both input and weight tensors. Summing all the contributions, the number of FLOPs required to do one forward and one backward pass (denoted by model FLOPs) is:

$$\text{model FLOPs per iteration} = 72BLsh^2\left(1 + \frac{s}{6h} + \frac{v}{12hL}\right). \quad (7)$$

Selective activation recomputation requires an additional forward pass attention matrix computation ($2Bs^2h$ operations) and attention over values ($2Bs^2h$ operations). Adding these required FLOPs to Equation 7, the total number of FLOPs we require per iteration (denoted by hardware FLOPs) is:

$$\text{hardware FLOPs per iteration} = 72BLsh^2\left(1 + \frac{2s}{9h} + \frac{v}{12hL}\right). \quad (8)$$

One can see that in our approach, model FLOPs are very close to the hardware FLOPs. Assuming $9h \gg 2s$ and $12hL \gg v$ and only considering main terms, the ratio of the hardware to model FLOPs can be approximated as:

$$\frac{\text{hardware FLOPs}}{\text{model FLOPs}} \approx 1 + \frac{s}{18h}. \quad (9)$$

B Pipeline Parallelism Memory Optimization

As mentioned in Section 4.2.3, efficient pipeline parallelism strategies do not uniformly divide the total memory across pipeline ranks. Figure 9 illustrates this memory pattern for the 530B model detailed in Table 3, showing the memory

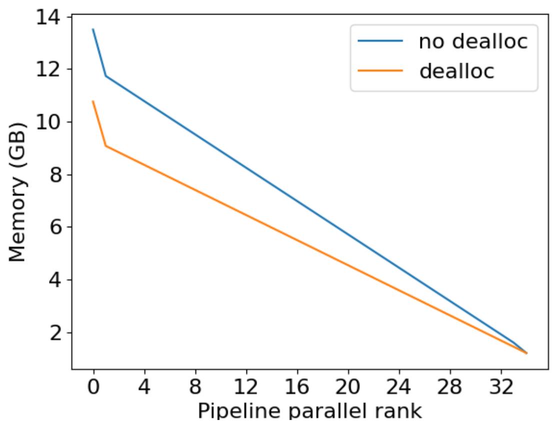

Figure 9. Activation memory of each pipeline parallel rank, shown for both an unoptimized case (blue), and with our memory optimization that deallocates the output tensor of each pipeline rank (yellow). The 530B model (see Table 3) is used for this experiment.

imbalance across pipeline ranks for both an unoptimized case, and when deallocating each microbatch's output tensor after its forward pass. This optimization relies on the fact that, after a microbatch's forward pass, the output tensor's data is redundant with the input data of the following stage, thereby making it safe to deallocate. We should note that in all other results shown in this paper, this output-tensor-deallocation optimization is used.

In this figure, we can observe the linear memory decrease along the pipeline ranks (directly observable for pipeline ranks 1 and above), along with the additional $sbhp$ memory spike on pipeline rank 0 due to the embedding layer's activation memory (see Section 4.3).

In the case of applying the output-tensor-deallocation optimization, we save $sbhr$ memory per pipeline rank, where r is the number of microbatches *in flight* on each rank, peaking at $r = p$ on the first pipeline stage. Using the hyperparameters for the 530B model (see Table 3), and this setup's 2 bytes per data element, the theoretical savings for this optimization on the first pipeline stage is $sbhp = 2.73$ GB, which closely matches the difference between the lines shown in the figure for rank 0.

C Microbatch Level Activation Recomputation

GPUs used in pipeline parallel model training store the input activations of layers until they are consumed at the gradient computation during back-propagation. As discussed in Section 4.2.3, the first pipeline stage stores the most activations, an equivalent of storing activations for all of the transformer layers in the model.

The computation and memory usage patterns of the first pipeline stage are illustrated in Figure 10. Yellow, red, and blue boxes indicate the execution of forward, recomputation, and back-propagation of one microbatch, respectively. This example uses the pipeline-parallel size of four and nine microbatches per iteration for training. Here the recomputation can be either full or selective. Figure 10.a depicts the execution flow of the baseline approach. Although there is some GPU memory left unused, if it is not large enough to store all of the activations and thus not require any recomputation, the activations of every microbatch is checkpointed and recomputed during back-propagation.

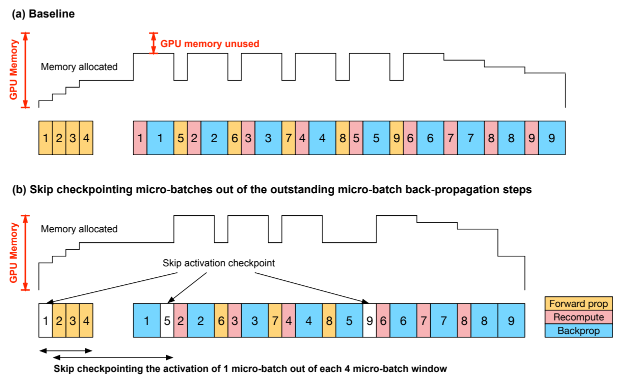
(a) Baseline
GPU Memory
GPU memory unused
Memory allocated
(b) Skip checkpointing micro-batches out of the outstanding micro-batch back-propagation steps
GPU Memory
Memory allocated
Skip activation checkpoint
Forward prop
Recompute
Backprop
Skip checkpointing the activation of 1 micro-batch out of each 4 micro-batch window

Figure 10. Computation and memory usage patterns of the baseline activation recomputation and microbatch level activation recomputation. Yellow boxes are a forward pass with activations checkpointed (i.e. only some activations are saved), red boxes are activation recomputation, blue boxes are backpropagation, and white boxes are a foward pass with all activations saved.

Microbatch level activation recomputation uses all the available device memory to store activations of some outstanding microbatches, and checkpoint the rest until the backpropagation frees enough memory to store another full layer of activations. We skip checkpointing microbatches (i.e. store all their activation) until all device memory is used. In Figure 10.b, the activations of the first microbatch are stored out of four microbatches. Once the activations of the first microbatch are consumed by the its back-propagation, enough memory is now free to store all activations of the fifth microbatch. This can be seen as a moving window of microbatches in which a certain number of them can have all activations stored.

Later pipeline stages have fewer outstanding backpropagation steps thus we can save all activations for the same number of microbatches within a smaller microbatch window. The size of outstanding microbatch backpropagation steps at each pipeline stage is calculated as $\max(0, p - S)$, where S is the stage number. Based on our observations, many of later pipeline stages do not need any activation recomputation.

Microbatch level activation recomputation increases the model FLOPs utilization of the 175B and 530B parameter models to 52.3% (+0.7%) and 56.4% (+0.4%), respectively, compared to the baseline with both sequence parallelism and selective activation recomputation. The gain is small because the selective recomputation overhead is as small as $\sim$2%. However, one can imagine model parallel configurations where there is nearly enough memory for no recomputation, in which case microbatch level activation recomputation could provide more improvement to training speed.