
ACCELERATING LARGE LANGUAGE MODEL TRAINING WITH 4D PARALLELISM AND MEMORY CONSUMPTION ESTIMATOR

Kazuki Fujii¹ Kohei Watanabe² Rio Yokota³

ABSTRACT

In large language model (LLM) training, several parallelization strategies, including Tensor Parallelism (TP), Pipeline Parallelism (PP), Data Parallelism (DP), as well as Sequence Parallelism (SP) and Context Parallelism (CP), are employed to distribute model parameters, activations, and optimizer states across devices. Identifying the optimal parallelization configuration for each environment while avoiding GPU memory overflow remains a challenging task. In this study, we provide precise formulas to estimate the memory consumed by parameters, gradients, optimizer states, and activations for 4D parallel training (DP, TP, PP, CP) in the Llama architecture. We conducted 454 experiments on A100 and H100 GPUs, incorporating often neglected factors such as temporary buffers and memory fragmentation into our analysis. Results indicate that when the estimated memory usage is below 80% of the available GPU memory, the training never encounters out-of-memory errors. This simple yet effective formula allows us to identify parallelization configurations that could lead to memory overflow in advance, significantly reducing the configuration search space. Additionally, through a comprehensive exploration of optimal configurations in 4D parallelism, our analysis of the 454 experimental results provides empirical insights into optimal 4D parallelism configurations.

1 INTRODUCTION

Training large language models (LLMs) requires model parallelism to distribute the model’s parameters, optimizer states, and activations across devices to fit within GPU memory constraints. Data parallelism is also employed to enable training within a realistic time. Efficient LLM training utilizes not only Tensor Parallelism (TP) (Shazeer et al., 2018; Shoeybi et al., 2019; Xu et al., 2021), Pipeline Parallelism (PP) (Huang et al., 2019; Li et al., 2021; Narayanan et al., 2019), and Data Parallelism (DP), but also Sequence Parallelism (SP) (Korthikanti et al., 2023) and Context Parallelism (CP) (Megatron-LM-team, 2024; Liu et al., 2023). However, determining the optimal parallelism configuration for a given environment while avoiding GPU memory overflow is challenging.

Excessive amounts of tensor parallelism can lead to smaller matrix multiplications, which reduce GPU utilization and degrade training throughput, as measured in TFLOP/s. Similarly, an oversized pipeline parallel configuration can increase pipeline bubbles, further reducing TFLOP/s. Additionally, when the global batch size is fixed, scaling the

number of GPUs and employing pipeline parallelism while increasing the data parallel size reduces the number of microbatches, making pipeline bubbles prominent and causing performance degradation beyond the communication overhead of added GPUs. Consequently, identifying an optimal parallelism configuration is non-trivial.

Moreover, when utilizing supercomputers with GPU memory capacities and per-node GPU counts different from those typically offered by cloud services—such as the NVIDIA H100 (SXM) 94GB×4 that we used in our experiments—one cannot reuse configurations that were optimal in common environments like H100 80GB×8. For example, even if setting TP=8 is optimal on an H100×8 setup, when using a cluster configured with H100 94GB×4, specifying a tensor parallel size exceeding the per-node GPU count of 4 leads to significant training speed degradation due to the additional communication cost introduced by tensor parallelism. Additionally, configurations that result in out-of-memory with 80GB memory may be trainable with 94GB memory. Therefore, a setting optimized in one environment does not necessarily apply to another. However, indiscriminately sweeping all parallelism configurations for each training environment requires substantial computational cost and time.

In the training of Llama 3 (Dubey et al., 2024), they addressed this issue by developing a memory consumption estimator, but its implementation has not been made public,

¹School of Computing, Institute of Science Tokyo, Tokyo, Japan ²Turing. inc, Tokyo, Japan ³Super Computing Research Center, Institute of Science Tokyo, Tokyo, Japan. Correspondence to: Kazuki Fujii <kazuki.fujii@rio.ssrc.iir.isct.ac.jp>.

and the open community cannot benefit from it. ZeRO (Rajbhandari et al., 2020) provides equations to calculate the memory size consumed by parameters, gradients, and optimizer states, but these equations are limited to the GPT architecture, and the calculation for activation memory remains approximate, posing practical difficulties. Other prior work (Korthikanti et al., 2023) presented equations for computing activations in GPT models, but these equations are for the case of 3D parallelism (DP, TP, PP) with sequence parallelism (SP), and have practical issues such as lack of support for the Llama architecture, which has been widely adopted recently, nor do they consider context parallelism, which is essential for handling long contexts.

In this work, we provide equations for the memory consumption of parameters, gradients, optimizer states, and activations when training with 4D parallelism in the Llama architecture. Additionally, based on empirical insights from 454 experiments conducted on A100(40GB) and H100(94GB) GPUs, we provide analytical results that consider factors such as temporary buffers and fragmentation, which are difficult to theoretically calculate in terms of memory usage. From our experiments, we found that when the memory consumption estimated by our estimator is 80% or less of the GPU memory, training succeeds in all cases. This simple yet effective equation not only allows us to detect parallelism configurations that would result in out-of-memory beforehand and reduce the configuration search space, but also enables exploration without relying on empirically optimal configurations. Furthermore, we analyzed the results obtained from 454 experiments to explore optimal settings in 4D parallelism, for which comprehensive configuration insights are lacking.

2 RELATED WORK

2.1 Parallelism

Model parallelism enables training large models across multiple GPUs. Model parameters, optimizer states, and activations require a huge amount of memory and do not fit on a single GPU. Even if we are able to fit the model on a single GPU (e.g., by using CPU offloading (Ren et al., 2021)), the high number of compute operations required can result in unrealistically long training times.

The 3D parallelism adopted in Megatron-LM (Narayanan et al., 2021) employs tensor parallelism and pipeline parallelism to distribute model parameters and optimizer states, enabling training within GPU memory constraints. Tensor parallelism splits and parallelizes each layer’s parameters across multiple GPUs, while pipeline parallelism partitions the model along the layer dimension. The integration of ZeRO Stage 1 (Rajbhandari et al., 2020) and Megatron-LM’s distributed optimizer, which shard optimizer states

across data-parallel ranks, further enhances memory efficiency and training scalability.

Despite its benefits, tensor parallelism in Megatron-LM does not partition activations from layers such as Dropout and LayerNorm, leading to redundant activation memory across tensor parallel ranks. Sequence Parallelism (Korthikanti et al., 2023) addresses this limitation by partitioning these activations along the sequence dimension, significantly reducing activation memory without incurring additional computational or communication overhead. By effectively combining sequence parallelism with tensor parallelism, it is possible to enhance memory efficiency when training large models. Consequently, modern LLM training often employs a combination of 3D parallelism, distributed optimizers, and sequence parallelism.

Context Parallelism Context Parallelism (Megatron-LM-team, 2024) (CP) is a method that performs parallelization along the sequence length dimension. Unlike Sequence Parallelism (Korthikanti et al., 2023), which only parallelizes the activations of Dropout and LayerNorm, Context Parallelism enables partitioning of the model’s network inputs and all activations along the sequence dimension.

Llama 2 (Touvron et al., 2023) had a sequence length of 4,096 tokens, but Llama 3 (Dubey et al., 2024) increased this to 8,192 tokens, and Llama 3.1 further extended it to 131,072 tokens. As efficient training that supports long contexts is increasingly demanded, context parallelism—which allows partitioning along the sequence dimension—is extremely useful for reducing the activations per GPU.

In components other than self-attention, there are no inter-token operations; thus, introducing context parallelism does not alter the operation. However, in self-attention layers, inter-token operations occur, necessitating the gathering of the full sequence, which requires additional all-gather communications between GPUs in the forward pass. During backpropagation, reduce-scatter is applied to the activation gradients, and each GPU stores only its sequence chunk to reduce the activation memory footprint.

To date, there is no comprehensive performance evaluation of 4D parallelism (DP, TP, PP, CP) utilizing context parallelism, and knowledge for applying it to actual LLM training is lacking.

2.2 Model states’ memory consumption

Model states—including parameters, gradients, and optimizer states (such as momentum and variance in the Adam optimizer) are the primary consumers of GPU memory during training. Efficient management of these states is critical to scaling up LLM training without exceeding memory constraints.

In FP16/FP32 mixed-precision training, if the number of model parameters is Ψ , the parameters and gradients are stored in FP16 format (2 bytes per value), consuming 2Ψ bytes each. Additionally, the optimizer states—including parameters, momentum, and variance—are stored in FP32 format (4 bytes per value), consuming 4Ψ bytes each. Therefore, the total estimated memory consumption is 16Ψ bytes (Rajbhandari et al., 2020).

With the advent of A100 GPUs that support BF16 training, FP16/FP32 mixed-precision training has transitioned to BF16/FP32 mixed-precision training. Furthermore, performing gradient accumulation in FP32 for numerical stability, as adopted in Llama-3 (Dubey et al., 2024) and implemented in widely used pre-training libraries like Megatron-LM¹, necessitates improvements to the formula presented in (Rajbhandari et al., 2020) to be applicable for training the latest LLMs.

2.3 Activation memory consumptions

The estimation of activations has been provided by (Korthikanti et al., 2023). However, this formula is not only specific to the GPT architecture but also does not assume the use of FlashAttention (Dao et al., 2022; Dao, 2023), which is utilized for faster and more memory-efficient training. Furthermore, it does not consider context parallelism (Megatron-LM-team, 2024). As a result, it cannot accurately calculate the activation memory when training Llama architecture models using 4D Parallelism.

3 MEMORY USAGE

Previous studies (Rajbhandari et al., 2020; Korthikanti et al., 2023) have attempted to consider GPU memory usage stemming from parameters, gradients, and optimizer states, as well as to calculate activation memory. However, none have simultaneously considered both aspects to estimate and discuss the per-GPU memory requirements in the context of 4D parallelism. Moreover, prior works have primarily focused on LLMs based on the GPT architecture and have not provided memory estimates that take into account the Llama architecture, which has been adopted by many open-source models.

In this study, we provide practical formulas to predict the memory consumption by estimating both parameter and activation memory using generalized equations that can handle the Llama architecture.

3.1 Model states' memory

We consider a single-stack transformer decoder model as shown in Figure 1. The input tensor of size $b \times s \times h$ passes

through an embedding layer of size $v \times h$, followed by L transformer layers, and then through RMSNorm, a linear layer, and a Softmax function to produce the output. Each transformer layer consists of self-attention, a feed-forward network (FFN), and RMSNorm. Notably, the Llama architecture does not include Dropout layers, which are present in the GPT architecture. Additionally, in the transformer layer's FFN, the GPT architecture increases the hidden size to $4h$ and then reduces it back to h . To accommodate the Llama architecture, we introduce h_{ffn} without assuming it to be $4h$, thereby defining a generalized variable. Furthermore, in the Llama architecture, the weights of the embedding layer and the language model head (output layer) are not shared; thus, we perform calculations under the assumption that they are unshared.

For reference, the variable names are listed in Table 1. To estimate the memory size consumed by parameters and optimizer states, we first calculate the number of model parameters.

Let the weights of the self-attention be $Q = XW_Q$, $K = XW_K$, $V = XW_V$, and let W_O be the weights of the linear layer after self-attention. Considering the group size in Grouped Query Attention as $g = \frac{a}{k}$, the sizes of W_Q , W_K , W_V , and W_O are (h, h) , $(h, h/g)$, $(h, h/g)$, and (h, h) , respectively. Therefore, the number of attention parameters per layer is:

$$\text{Attention parameter per layer} = 2h^2(1 + \frac{k}{a}) \quad (1)$$

Similarly, in the FFN layer, the sizes of the up projection $W_{\text{mlp_up}}$, gate projection $W_{\text{mlp_gate}}$, and down projection $W_{\text{mlp_down}}$ are (h, h_{ffn}) , (h, h_{ffn}) , and (h_{ffn}, h) , respectively. Therefore, the number of FFN parameters per layer is:

$$\text{FFN parameter per layer} = 3hh_{\text{ffn}} \quad (2)$$

Taking into account the embedding layer, language model head, RMSNorm layers, and the final RMSNorm, the total number of parameters can be expressed as:

$$\begin{aligned} & 2hv + L(2h^2(1 + \frac{k}{a}) + 3h^2\frac{h_{\text{ffn}}}{h} + 2h) + h \\ &= 2hv + h + 2Lh^2(1 + \frac{k}{a} + \frac{3}{2}\frac{h_{\text{ffn}}}{h} + \frac{1}{h}) \end{aligned} \quad (3)$$

In Equation (3), the term $2hv$ accounts for the embedding and output layers, L is the number of transformer layers, and h is the hidden size. The factor $2h^2(1 + \frac{k}{a})$ comes from the attention parameters per layer (Equation (1)), and $3hh_{\text{ffn}}$ is from the FFN parameters per layer (Equation (2)).

¹<https://github.com/NVIDIA/Megatron-LM>

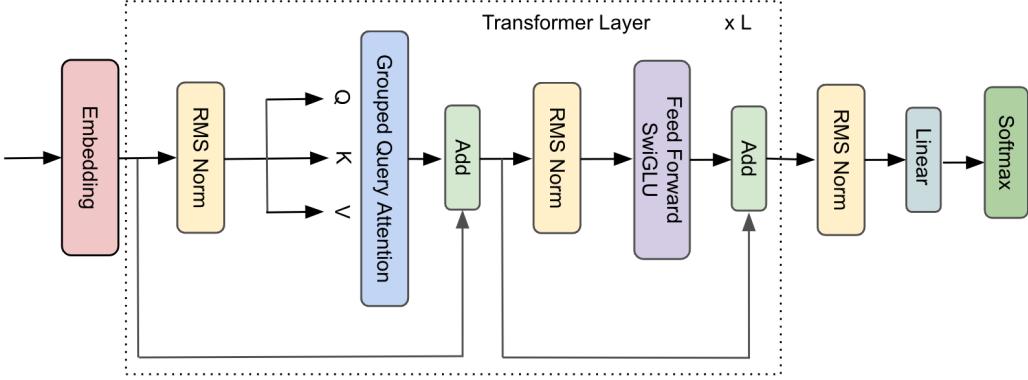


Figure 1. Llama Architecture

Table 1. Variable names.

a	number of attention heads	p	pipeline parallel size
b	microbatch size	s	sequence length
h	hidden size	t	tensor parallel size
d	data parallel size	c	context parallel size
h_{ffn}	FFN hidden size	k	number of key-value heads
L	number of transformer layers	v	vocabulary size

The additional $2h$ within the parentheses accounts for the RMSNorm layers in each transformer layer, and the final h outside accounts for the last RMSNorm layer after the transformer stack.

From the above calculations, we have determined the number of parameters. Based on this, we can calculate the memory consumed by the parameter weights and optimizer states. In ZeRO (Rajbhandari et al., 2020), since gradients are assumed to be stored in FP16 or BF16, the memory size required during training when using the Adam (Kingma & Ba, 2017) optimizer is considered to be 16Ψ , where Ψ is the number of parameters. However, as reported by Llama 3 (Dubey et al., 2024), gradients may be accumulated in FP32 to ensure convergence during training. In this paper, we assume that gradients are accumulated in FP32, resulting in a memory consumption of 18Ψ . Note that FP16/BF16 consumes 2 bytes per value, while FP32 consumes 4 bytes per value. The breakdown of the coefficient 18 is as follows:

weight:	BF16 (2 bytes)
gradients:	FP32 (4 bytes)
optimizer states:	
parameter (master weight):	FP32 (4 bytes)
gradient momentum:	FP32 (4 bytes)
gradient variance:	FP32 (4 bytes)

Therefore, the memory consumed by the parameters, gradients, and optimizer states is given by:

$$18(2hv + h + 2Lh^2(1 + \frac{k}{a} + \frac{3}{2} \frac{h_{\text{ffn}}}{h} + \frac{1}{h})) \quad (4)$$

This equation holds in cases where model parallelism methods, such as Tensor Parallelism and Pipeline Parallelism, are not applied. In the following sections, we will sequentially demonstrate how the per-GPU memory consumption changes when each parallelization method is introduced.

3.1.1 Data Parallelism

By utilizing techniques such as ZeRO Stage 1 (Rajbhandari et al., 2020) and Megatron-LM’s distributed optimizer, which shard the optimizer states across data parallel processes, the optimizer states amounting to 12Ψ are distributed among the data parallel processes. As a result, the memory consumption per GPU becomes:

$$(6 + \frac{12}{d})(2hv + h + 2Lh^2(1 + \frac{k}{a} + \frac{3}{2} \frac{h_{\text{ffn}}}{h} + \frac{1}{h})) \quad (5)$$

3.1.2 Tensor Parallelism

Tensor Parallelism divides the parameters of the Attention, FFN, embedding, and language model head layers by t . Specifically, the Attention parameters become $(2h^2(1 + \frac{k}{a})) / t$, and the MLP parameters become $(3hh_{\text{ffn}}) / t$. Therefore, when both Data Parallelism and Tensor Paral-

lelism are applied, the memory consumption of parameters, gradients, and optimizer states per GPU is:

$$(6 + \frac{12}{d})(\frac{2hv}{t} + h + 2Lh^2(\frac{1 + \frac{k}{a} + \frac{3}{2}\frac{h_{\text{ffn}}}{h}}{t} + \frac{1}{h})) \quad (6)$$

3.1.3 Pipeline Parallelism

In Pipeline Parallelism, for the first pipeline stage—which includes the embedding layer—the memory consumption is calculated as follows. Since the final LayerNorm does not pertain to the first pipeline stage, only the embedding layer’s parameters hv and the parameters of $\frac{L}{p}$ layers are considered. In this paper, we assume the 1F1B pipeline schedule developed in PipeDream (Narayanan et al., 2020) for pipeline scheduling.

$$(6 + \frac{12}{d})(\frac{hv}{t} + 2\frac{L}{p}h^2(\frac{1 + \frac{k}{a} + \frac{3}{2}\frac{h_{\text{ffn}}}{h}}{t} + \frac{1}{h})) \quad (7)$$

For the intermediate pipeline stages, which do not include the embedding or language model head layer, the memory consumption per GPU is:

$$(6 + \frac{12}{d})(2\frac{L}{p}h^2(\frac{1 + \frac{k}{a} + \frac{3}{2}\frac{h_{\text{ffn}}}{h}}{t} + \frac{1}{h})) \quad (8)$$

For the last pipeline stage, considering the language model head and the final LayerNorm (RMSNorm), the memory consumption per GPU is:

$$(6 + \frac{12}{d})(\frac{hv}{t} + h + 2\frac{L}{p}h^2(\frac{1 + \frac{k}{a} + \frac{3}{2}\frac{h_{\text{ffn}}}{h}}{t} + \frac{1}{h})) \quad (9)$$

3.1.4 Context Parallelism

Context Parallelism not only partitions the inputs and activations along the sequence dimension but also distributes the optimizer states across processes. Consequently, the optimizer states are divided among $d \times c$ processes. Considering this, the memory consumption per GPU (here shown for the first stage of Pipeline Parallelism) is:

$$(6 + \frac{12}{dc})(\frac{hv}{t} + 2\frac{L}{p}h^2(\frac{1 + \frac{k}{a} + \frac{3}{2}\frac{h_{\text{ffn}}}{h}}{t} + \frac{1}{h})) \quad (10)$$

3.2 Activation Memory

In the previous section, we calculated the memory size consumed by parameters, gradients, and optimizer states. In

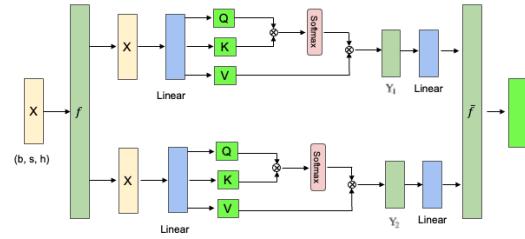


Figure 2. Attention Block (when tensor parallel size = 2)

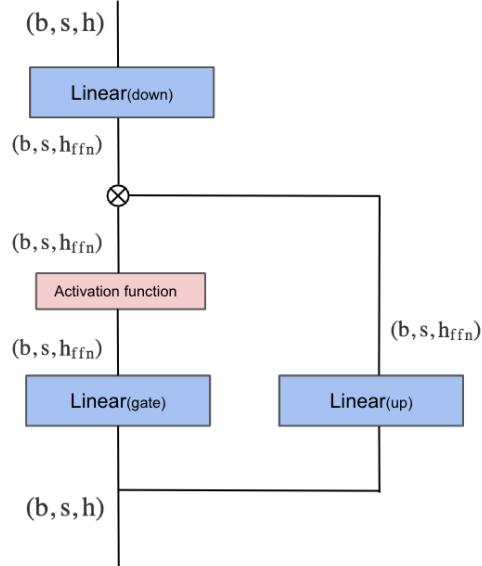


Figure 3. FFN (Feed Forward Network) in the Llama architecture.

this section, we compute the memory size consumed by activations. As shown in Figure 1, the primary contributor to activation memory is the Transformer layer. Therefore, we first calculate the activations for the Attention block, Feed-Forward Network (FFN), and RMSNorm within the Transformer layer, and then consider the embedding layer and the language model (LM) head. As depicted in Figure 2, the attention block consists of self-attention followed by a linear projection.² In practical LLM training, FlashAttention 2 (Dao, 2023) is used to improve training speed and reduce memory consumption. Therefore, we perform the activation calculations assuming that FlashAttention 2 is being utilized.

• Query (Q), Key (K), and Value (V) matrix mul-

²While the GPT architecture includes Dropout layers, the Llama architecture does not include Dropout.

tiples: The size of the Query is (b, s, h) , so the required activations are $2sbh$. However, when employing Grouped Query Attention (Ainslie et al., 2023), the Key and Value tensors have sizes of $(b, s, h \times \frac{k}{a})$, so the activations required for each are $2sbh \times \frac{k}{a}$.

- **QK^\top matrix multiply:** FlashAttention does not store the results of the large QK^\top matrices and recomputes them during the backward pass. Therefore, the activation memory required is zero.
- **Softmax:** Since FlashAttention recomputes during the backward pass, the activation memory required for the Softmax is also zero.
- **Attention over Values (V):** The output tensor size is (b, s, h) , so it consumes $2sbh$ of memory.

Including the input X , the total activation memory consumed by the self-attention is:

$$6sbh + 4sbh \times \frac{k}{a} \quad (11)$$

FFN: The structure of the Llama architecture’s FFN is shown in Figure 3. Therefore, we require activations for backpropagation as follows: $2sbh$ for the input, $2sbh_{\text{ffn}}$ for the output of the up projection, $2sbh_{\text{ffn}}$ for the output of the gate projection, and $2sbh_{\text{ffn}}$ for the activation function non-linearity. Additionally, $2sbh_{\text{ffn}}$ is needed for the input to the down projection. In total, $2sb(h + 4h_{\text{ffn}})$ activations are required.

Layer Norm: Each RMSNorm stores its input with size $2sbh$, so in total, we need $4sbh$ of storage. Adding up the memory required for the attention, FFN, and the RMSNorms, the total activation memory required to store the activations for a single layer of the Transformer network is:

$$\text{Activation memory per layer} = sbh \left(12 + 4\frac{k}{a} + 8\frac{h_{\text{ffn}}}{h} \right). \quad (12)$$

The above equation applies to the case where no form of model parallelism is applied.

Next, we consider the activation memory consumed by the embedding layer and the language model head.

Embedding: The input to the embedding layer has a numerical precision of `int 64`, which consumes 8 bytes per value. To keep the pipeline fully utilized and avoid idle time, the first stage must store activations for p microbatches (for more details, see Figure 4 top of (Narayanan et al., 2021)). Therefore, the activation memory consumed by the embedding layer is:

$$8sbhp \quad (13)$$

LM Head: The cross-entropy loss requires FP32 numerical precision, consuming $4sbv$ bytes of memory. Additionally, $2sbh$ is needed for the output RMSNorm, and $2sbh$ for the output linear layer, resulting in a total memory consumption of:

$$4sbh \left(1 + \frac{v}{h} \right) \quad (14)$$

For more details, see Section 4.3 of (Korthikanti et al., 2023).

3.2.1 Tensor Parallelism

We use the tensor parallelism developed by (Shoeybi et al., 2019) to parallelize the attention and FFN modules, as illustrated in Figure 2. This form of parallelism introduces two additional communication operations, f and \bar{f} , per layer. For more details, please refer to (Shoeybi et al., 2019).

As described in Section 3.1.2, tensor parallelism parallelizes the model states, and similarly, the activations are also parallelized. However, certain elements, such as the input to the FFN, cannot be parallelized. Additionally, RMSNorm is not parallelized. By enabling Sequence Parallelism, we can parallelize these components as well. For more details, please see Figure 6 in (Korthikanti et al., 2023).

Considering the above, the activation memory size consumed by the Transformer block can be expressed using the tensor parallel size t as follows:

$$\text{Transformer layers} = sbh \left(12 + 4\frac{k}{a} + 8\frac{h_{\text{ffn}}}{h} \right) L/t \quad (15)$$

3.2.2 Pipeline Parallelism

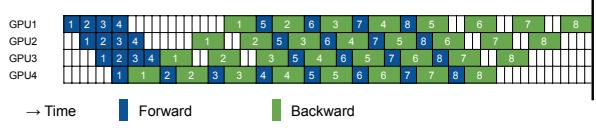


Figure 4. 1F1B (One Forward One Backward) Pipeline schedule. Blue represents the forward pass, green indicates the backward pass, and uncolored spaces represent pipeline bubbles.

Pipeline Parallelism divides the Transformer layers, which consist of L layers, into p groups, each containing L/p layers. However, unlike in Section 3.1.3, the total activation consumed by the Transformer layers is not evenly divided by p . This is due to the pipeline parallel scheduling. In this paper, we assume the 1F1B pipeline schedule, as shown in Figure 4. In the 1F1B schedule, the GPU assigned to the first pipeline stage has up to p microbatches worth of activations, while the GPU assigned to the last pipeline stage needs to store activations for only one microbatch because

the backward pass starts immediately. Therefore, although each stage is responsible for L/p layers, the first stage holds activations equivalent to $L/p \times p = L$ layers. Since this does not depend on p , the first stage always needs to store the same amount of activations for L layers, regardless of the pipeline parallel size p .

Considering the above and introducing the Kronecker delta $\delta_{p,1}$, the maximum activation memory per GPU when using Pipeline Parallelism can be expressed as follows. Here, we focus only on the first pipeline stage, which is critical in determining whether an out-of-memory (OOM) occurs.

$$\frac{sbh}{t} \left((12 + 4\frac{k}{a} + 8\frac{h_{\text{ffn}}}{h})L + 8p + \delta_{p,1}4(1 + v/h) \right) \quad (16)$$

3.2.3 Context Parallelism

As explained in Section 2.1, context parallelism partitions the network inputs and all activations along the sequence dimension. Therefore, the activation memory consumption per GPU can be expressed using the context parallel size c as:

$$\frac{sbh}{tc} \left((12 + 4\frac{k}{a} + 8\frac{h_{\text{ffn}}}{h})L + 8p + \delta_{p,1}4(1 + v/h) \right) \quad (17)$$

3.3 Total Memory

The majority of the required activation memory per GPU is captured by Equation 17. Similarly, the required memory per GPU for model states (parameters, gradients, optimizer states) is calculated using Equation 10. Therefore, excluding residual memory consumption due to temporary buffers and memory fragmentation—which are difficult to calculate theoretically—the total memory consumption is given by:

$$\begin{aligned} & \left(6 + \frac{12}{dc} \right) \left(\frac{hv}{t} + 2\frac{L}{p}h^2 \left(\frac{1 + \frac{k}{a} + \frac{3}{2}\frac{h_{\text{ffn}}}{h}}{t} + \frac{1}{h} \right) \right) \\ & + \frac{sbh}{tc} \left((12 + 4\frac{k}{a} + 8\frac{h_{\text{ffn}}}{h})L + 8p + \delta_{p,1}4(1 + \frac{v}{h}) \right) \end{aligned} \quad (18)$$

4 EVALUATIONS

4.1 Memory Usage

We utilized the memory consumption estimator implemented based on the equations presented in this paper to conduct experiments on NVIDIA A100 SXM (40GB) and

NVIDIA H100 SXM (94GB) GPUs. We conducted experiments training models with the same architecture as Llama-3.1-8B. On the A100, we used a sequence length of 8,192, while on the H100, we experimented with sequence lengths of 8,192, 16,384, and 32,768. Additionally, we trained models with the same architecture as Llama-3.1-70B on the A100 with a sequence length of 8,192 only. In all experiments, the global batch size was fixed at 1,024.

The memory consumption estimator-predicted values represent the memory consumption of model states and activations, as described in Section 3.3, and do not account for temporary buffers and fragmentation. From our experimental results, we found that if there is 20% spare GPU memory, the impact of temporary buffers and fragmentation does not affect whether an out-of-memory (OOM) occurs, as demonstrated in Table 3 by actually experimenting with each configuration. This finding was consistent not only in the A100 (40GB) environment but also in the H100 (94GB) environment, as shown in Table 4, 10, 11.

Therefore, by using Equation 18, we demonstrated that it is possible to estimate, without the need for an accelerator device environment, which configurations are trainable within the memory of the devices used in any GPU environment, which configurations are not, and which configurations may be trainable depending on temporary buffers and fragmentation. By using this simple yet effective equation, we could preliminarily narrow down candidate parallelism configurations that have the potential to achieve high throughput (TFLOP/s) by effectively utilizing HBM memory. By experimenting only with these candidates, it became possible to discover the optimal parallelism configuration while saving computational resources.

4.2 Performance Analysis of Parallelism Configuration

In this section, we provide an empirical analysis of the results obtained from experiments using 4D parallelism on NVIDIA A100 SXM (40GB) and NVIDIA H100 SXM (94GB) GPUs. Throughput was measured in TFLOP/s, using the formula adapted for the Llama architecture based on prior work (Narayanan et al., 2021).

4.2.1 Throughput Analysis on A100(40GB)

As shown in Table 3 and Figure 5, the parallel configurations that achieve high TFLOP/s are those with the minimal combination of $TP \times CP \times PP$ that does not result in out-of-memory errors. This trend is also observed in Table 6, which presents the throughput for training Llama-3.1-70B in the Appendix, indicating its general applicability.

Comparing configurations using only tensor parallelism ($TP, CP, PP = (4, 1, 1)$) with those combining tensor parallelism and pipeline parallelism ($TP, CP, PP = (2, 1, 2)$) or

Table 2. Estimated Total Memory (GB) per GPU when training Llama-3.1-8B on A100(40GB) with sequence length 8,192. The vertical axis (TP, CP, PP, MBS) represents Tensor Parallel size (TP), Context Parallel size (CP), Pipeline Parallel size (PP), and Micro Batch Size (MBS), respectively. Red cells indicate settings that are estimated to exceed the 40GB GPU memory limit, likely resulting in Out of Memory (OOM) errors. Yellow cells denote configurations that may approach the limit due to temporary buffers and memory fragmentation, and green cells represent estimations at or below 80% of GPU memory (32GB or less), suggesting they are less likely to cause OOM.

(TP, CP, PP, MBS)	8 GPUs	16 GPUs	32 GPUs	64 GPUs	128 GPUs	256 GPUs
(4, 1, 2, 1)	27.2	21.59	18.79	17.39	16.69	16.34
(4, 1, 2, 2)	37.58	31.97	29.16	27.76	27.06	26.71
(4, 1, 2, 4)	58.33	52.72	49.91	48.51	47.81	47.46
(4, 2, 2, 1)	-	16.41	13.6	12.2	11.5	11.15
(4, 2, 2, 2)	-	21.59	18.79	17.39	16.69	16.34
(4, 2, 2, 4)	-	31.97	29.16	27.76	27.06	26.71
(4, 2, 2, 8)	-	52.72	49.91	48.51	47.81	47.46
(2, 2, 2, 1)	32.81	27.2	24.4	23.00	22.29	21.94
(2, 2, 2, 2)	43.19	37.58	34.77	33.37	32.67	32.32
(2, 4, 2, 1)	-	22.02	19.21	17.81	17.11	16.76
(2, 4, 2, 2)	-	27.2	24.40	23.00	22.29	21.94
(4, 2, 1, 1)	28.1	22.49	19.69	18.28	17.58	17.23
(4, 2, 1, 2)	33.76	28.15	25.35	23.94	23.24	22.89
(4, 2, 1, 4)	45.08	39.47	36.67	35.27	34.56	34.21
(2, 2, 4, 1)	-	23.19	20.01	18.43	17.64	17.24
(2, 2, 4, 2)	-	33.69	30.51	28.93	28.14	27.74
(2, 2, 4, 4)	-	54.69	51.51	49.93	49.14	48.74
(2, 4, 1, 1)	39.32	33.71	30.9	29.5	28.8	28.45
(2, 4, 1, 2)	44.98	39.37	36.56	35.16	34.46	34.11
(2, 4, 1, 4)	56.3	50.69	47.89	46.48	45.78	45.43
(4, 1, 1, 1)	33.76	28.15	25.35	23.94	23.24	22.89
(4, 1, 1, 2)	45.08	39.47	36.67	35.27	34.56	34.21
(2, 2, 1, 1)	44.98	39.37	36.56	35.16	34.46	34.11
(2, 2, 1, 2)	56.3	50.69	47.89	46.48	45.78	45.43
(2, 1, 2, 1)	43.19	37.58	34.77	33.37	32.67	32.32
(2, 1, 2, 2)	63.94	58.33	55.52	54.12	53.42	53.07

tensor parallelism and context parallelism (TP, CP, PP) = (2, 2, 1), we find that combining tensor parallelism with other parallelism methods achieves slightly higher TFLOP/s for the same micro batch size. However, since tensor parallelism can reduce both model states and activations memory, there is flexibility in choosing the micro batch size (MBS). In the case of (TP, CP, PP, MBS) = (4, 1, 1, 2), using only tensor parallelism becomes the optimal configuration on 256GPUs.

Because context parallelism does not reduce the memory consumed by model states' parameters and gradients, it may not be possible to sufficiently distribute the optimizer states among data parallel processes and context parallel processes when using a small number of GPUs. This can lead to out-of-memory, as observed in Table 3 for configurations with 8 to 64 GPUs. Additionally, in the configuration (TP, CP, PP) = (2, 1, 2), activation memory is not reduced in the first stage due to the nature of the 1F1B pipeline schedule (Narayanan et al., 2021).

This imposes constraints on the allowable micro batch sizes to avoid out-of-memory errors.

From the above observations, we conclude that if the tensor parallel size can be kept below the number of GPUs per node, increasing the micro batch size in configurations using only tensor parallelism—up to the point where the memory consumption estimator indicates an out-of-memory condition—is a useful method for discovering sub-optimal settings.

Furthermore, in the configuration (TP, CP, PP, MBS) = (2, 1, 2, 1), the number of microbatches decreases from 128 on 32 GPUs to 16 on 256 GPUs due to the increase in data parallel size. Consequently, the pipeline bubble fraction, given by $\frac{p-1}{m}$ where m is the number of microbatches, increases by a factor of 8 (for details on the pipeline bubble fraction formula, refer to Section 2.2 of (Narayanan et al., 2021)). As a result, while this configuration is the fastest at 32 and 64 GPUs, its throughput decreases by approx-

Table 3. Measured throughput(TFLOP/s) when training Llama-3.1-8B on A100 (40GB) with a sequence length of 8,192. The colors red, yellow, and green correspond to memory consumption levels predicted by the memory consumption estimator: red indicates configurations that likely exceed memory limits, yellow denotes configurations near the memory limit, and green represents configurations predicted to use 80% or less of GPU memory (32GB or less).

(TP, CP, PP, MBS)	8 GPUs	16 GPUs	32 GPUs	64 GPUs	128 GPUs	256 GPUs
(4, 1, 2, 1)	182.6	178.38	176.79	170.84	169.52	153.9
(4, 1, 2, 2)	OOM	189.79	187.01	185.69	182.28	178.62
(4, 1, 2, 4)	OOM	OOM	OOM	OOM	OOM	OOM
(4, 2, 2, 1)	-	151.65	148.7	142.34	134.34	116.28
(4, 2, 2, 2)	-	155.01	160.27	158.15	150.64	127.03
(4, 2, 2, 4)	-	158.99	156.36	154.88	160.62	160.83
(4, 2, 2, 8)	-	OOM	OOM	OOM	OOM	OOM
(2, 2, 2, 1)	187.83	192.62	185.07	178.58	165.2	142.61
(2, 2, 2, 2)	OOM	OOM	196.65	192.54	188.77	183.17
(2, 4, 2, 1)	-	133.54	128.95	124.56	108.73	98.79
(2, 4, 2, 2)	-	144.41	142.4	138.89	124.05	117.65
(4, 2, 1, 1)	171.16	174.07	171.99	168.66	162.15	153.03
(4, 2, 1, 2)	OOM	188.97	187.29	185.35	183.49	181.97
(4, 2, 1, 4)	OOM	OOM	OOM	OOM	OOM	OOM
(2, 2, 4, 1)	-	153.99	137.78	133.72	121.44	105.93
(2, 2, 4, 2)	-	157.73	143.15	135.33	126.91	118.14
(2, 2, 4, 4)	-	OOM	OOM	OOM	OOM	OOM
(2, 4, 1, 1)	OOM	OOM	163.1	153.42	144.44	131.53
(2, 4, 1, 2)	OOM	OOM	OOM	OOM	OOM	OOM
(2, 4, 1, 4)	OOM	OOM	OOM	OOM	OOM	OOM
(4, 1, 1, 1)	194.25	194.97	194.64	190.9	191.88	188.28
(4, 1, 1, 2)	OOM	OOM	OOM	OOM	201.37	197.12
(2, 2, 1, 1)	OOM	OOM	OOM	OOM	198.59	193.57
(2, 2, 1, 2)	OOM	OOM	OOM	OOM	OOM	OOM
(2, 1, 2, 1)	OOM	OOM	203.98	202.00	195.55	193.48
(2, 1, 2, 2)	OOM	OOM	OOM	OOM	OOM	OOM

imately 6.5 TFLOP/s and 8.5 TFLOP/s on 128 and 256 GPUs, respectively, compared to the 64 GPU configuration.

These findings indicate that no specific parallelism configuration is universally optimal. However, when searching for the optimal experimental settings, it is important to avoid exploring configurations that unnecessarily increase $TP \times CP \times PP$.

As shown in Table 4, similar to the experiments on the A100 (40GB), focusing on the minimal $TP \times CP \times PP$ that does not result in out-of-memory is crucial for identifying optimal configurations. In the case of 4 GPUs, the configuration $(TP, CP, PP, MBS) = (1, 2, 1, 1)$ results in a CUDA out-of-memory error. The memory consumption estimator predicts a memory usage of 89.95 GB, which is close to the 94 GB limit. This occurs because, as pointed out in Section 4.2.1, context parallelism splits activations along the sequence dimension but does not partition the model states’ parameters and gradients. In contrast, the configuration $(TP, CP, PP, MBS) = (2, 1, 1, 1)$ has a memory consump-

tion of 67.52 GB on 4 GPUs according to the estimator, providing significantly more headroom.

However, while context parallelism is disadvantageous in terms of memory reduction, it has a lower communication overhead compared to tensor parallelism. Tensor parallelism requires two all-reduce operations during forward pass per transformer layer, whereas context parallelism requires only one all-gather operation during the forward pass per transformer layer. Therefore, in configurations where memory is sufficient, $(TP, CP, PP, MBS) = (1, 2, 1, 1)$ is faster than $(TP, CP, PP, MBS) = (2, 1, 1, 1)$ across all comparable GPU counts from 8 to 64 GPUs.

4.2.2 Micro Batch Size

In the experimental results where $TP \times CP \times PP$ was not increased unnecessarily, as shown in Tables 3 and 4, increasing the micro batch size consistently led to improved throughput across all configurations. Increasing the micro batch size enhances the arithmetic intensity of executed

Table 4. Measured throughput(TFLOP/s) when training Llama-3.1-8B on H100 (94GB) with a sequence length of 8,192. The colors red, yellow, and green correspond to memory consumption levels predicted by the memory consumption estimator: red indicates configurations that likely exceed memory limits, yellow denotes configurations near the memory limit, and green represents configurations predicted to use 80% or less of GPU memory (75.2GB or less).

(TP, CP, PP, MBS)	4 GPUs	8 GPUs	16 GPUs	32 GPUs	64 GPUs
(2, 1, 1, 1)	446.72	443.45	439.87	438.69	435.34
(2, 1, 1, 2)	OOM	479.16	475.2	471.82	469.01
(2, 1, 1, 4)	OOM	OOM	OOM	OOM	OOM
(2, 2, 1, 1)	396.83	393.96	391.4	383.28	382.72
(2, 2, 1, 2)	421.59	418.79	422.93	418.39	414.46
(2, 2, 1, 4)	OOM	452.21	456.01	447.8	446.01
(2, 2, 1, 8)	OOM	OOM	OOM	OOM	OOM
(4, 1, 1, 1)	408.29	406.13	402.58	402.04	397.55
(4, 1, 1, 2)	443.83	439.61	439.53	438.38	433.62
(4, 1, 1, 4)	OOM	448.51	446.4	446.24	443.92
(2, 1, 2, 1)	415.39	413.75	408.00	410.59	405.03
(2, 1, 2, 2)	449.1	445.74	434.68	435.92	426.5
(2, 1, 2, 4)	OOM	OOM	OOM	OOM	OOM
(1, 2, 1, 1)	OOM	495.9	487.58	481.69	483.56
(1, 2, 1, 2)	OOM	OOM	OOM	OOM	OOM
(1, 4, 1, 1)	449.58	441.25	436.39	425.73	415.52
(1, 4, 1, 2)	OOM	465.92	462.95	448.61	435.31

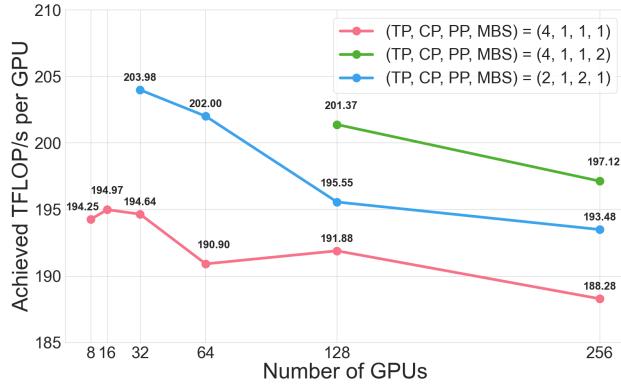


Figure 5. Training TFLOP/s of Llama-3.1-8B with sequence length 8,192 on A100 (40GB). This figure illustrates how the optimal parallelism configuration changes as the number of GPUs increases.

kernels, thereby increasing GPU utilization. While this improves GPU utilization, it simultaneously reduces the number of microbatches. When using pipeline parallelism, as noted in Section 4.2.1, this reduction can lead to increased pipeline bubbles, negatively affecting throughput. Which effect dominates depends on factors such as the number of GPUs used and the model size, making it challenging to identify the optimal setting without experimentation. However, in our experiments, the positive impact of increased GPU utilization from a larger micro batch size outweighed the negative effects. In cases where pipeline parallelism was not used, we exclusively benefited from the increased

GPU utilization, resulting in increased throughput when the micro batch size was increased.

5 CONCLUSIONS

In this paper, we addressed the challenge of selecting optimal parallelization configurations for large language model (LLM) training without causing GPU memory overflow. We introduced precise formulas to estimate memory consumption when using 4D parallelism (DP, TP, PP, CP) in the Llama architecture. Validated through 454 experiments on NVIDIA A100 and H100 GPUs, our results confirmed that training always succeeds when estimated memory usage is below 80% of GPU memory. We found that configurations minimizing $TP \times CP \times PP$ while avoiding memory overflow achieve higher throughput. Our work provides practical tools for efficiently identifying optimal configurations, enhancing resource allocation and accelerating LLM training.

6 REPRODUCIBILITY STATEMENT

The experiments in this study were conducted using a forked code of Megatron-LM mcore v0.8.0³ and TransformerEngine v1.9⁴. The experimental environment in-

³https://github.com/NVIDIA/Megatron-LM/releases/tag/core_r0.8.0

⁴<https://github.com/NVIDIA/TransformerEngine/releases/tag/v1.9>

cluded PyTorch 2.3.1+cu121, flash-attention 2.5.8, CUDA Toolkit 12.1, cuDNN 8.9.7, NCCL 2.20.5, and HPC-X 2.17.1.

REFERENCES

- Ainslie, J., Lee-Thorp, J., de Jong, M., Zemlyanskiy, Y., Lebrón, F., and Sanghai, S. GQA: Training Generalized Multi-Query Transformer Models from Multi-Head Checkpoints, 2023. URL <https://arxiv.org/abs/2305.13245>.
- Dao, T. Flashattention-2: Faster attention with better parallelism and work partitioning. *arXiv preprint arXiv:2307.08691*, 2023.
- Dao, T., Fu, D., Ermon, S., Rudra, A., and Ré, C. Flashattention: Fast and memory-efficient exact attention with io-awareness. *Advances in Neural Information Processing Systems*, 35:16344–16359, 2022.
- Dubey, A., Jauhri, A., Pandey, A., Kadian, A., Al-Dahle, A., Letman, A., Mathur, A., Schelten, A., Yang, A., Fan, A., Goyal, A., Hartshorn, A., Yang, A., Mitra, A., Sravankumar, A., Korenev, A., Hinsvark, A., Rao, A., Zhang, A., Rodriguez, A., Gregerson, A., Spataru, A., Roziere, B., Biron, B., Tang, B., Chern, B., Caucheteux, C., Nayak, C., Bi, C., Marra, C., McConnell, C., Keller, C., Touret, C., Wu, C., Wong, C., Ferrer, C. C., Nikolaidis, C., Al-lonsius, D., Song, D., Pintz, D., Livshits, D., Esiobu, D., Choudhary, D., Mahajan, D., Garcia-Olano, D., Perino, D., Hupkes, D., Lakomkin, E., AlBadawy, E., Lobanova, E., Dinan, E., Smith, E. M., Radenovic, F., Zhang, F., Synnaeve, G., Lee, G., Anderson, G. L., Nail, G., Mialon, G., Pang, G., Cucurell, G., Nguyen, H., Korevaar, H., Xu, H., Touvron, H., Zarov, I., Ibarra, I. A., Kloumann, I., Misra, I., Evtimov, I., Copet, J., Lee, J., Geffert, J., Vranes, J., Park, J., Mahadeokar, J., Shah, J., van der Linde, J., Billock, J., Hong, J., Lee, J., Fu, J., Chi, J., Huang, J., Liu, J., Wang, J., Yu, J., Bitton, J., Spisak, J., Park, J., Rocca, J., Johnstun, J., Saxe, J., Jia, J., Alwala, K. V., Upasani, K., Plawiak, K., Li, K., Heafield, K., Stone, K., El-Arini, K., Iyer, K., Malik, K., Chiu, K., Bhalla, K., Rantala-Yeary, L., van der Maaten, L., Chen, L., Tan, L., Jenkins, L., Martin, L., Madaan, L., Malo, L., Blecher, L., Landzaat, L., de Oliveira, L., Muzzi, M., Pasupuleti, M., Singh, M., Paluri, M., Kardas, M., Oldham, M., Rita, M., Pavlova, M., Kambadur, M., Lewis, M., Si, M., Singh, M. K., Hassan, M., Goyal, N., Torabi, N., Bashlykov, N., Bogoychev, N., Chatterji, N., Duchenne, O., Çelebi, O., Alrassy, P., Zhang, P., Li, P., Vasic, P., Weng, P., Bhargava, P., Dubal, P., Krishnan, P., Koura, P. S., Xu, P., He, Q., Dong, Q., Srinivasan, R., Ganapathy, R., Calderer, R., Cabral, R. S., Stojnic, R., Raileanu, R., Girdhar, R., Patel, R., Sauvestre, R., Polidoro, R., Sumbaly, R., Taylor, R., Silva, R., Hou, R., Wang, R., Hosseini, S., Chennabasappa, S., Singh, S., Bell, S., Kim, S. S., Edunov, S., Nie, S., Narang, S., Raparthy, S., Shen, S., Wan, S., Bhosale, S., Zhang, S., Vandenhende, S., Batra, S., Whitman, S., Sootha, S., Collot, S., Gururangan, S., Borodinsky, S., Herman, T., Fowler, T., Sheasha, T., Georgiou, T., Scialom, T., Speckbacher, T., Mihaylov, T., Xiao, T., Karn, U., Goswami, V., Gupta, V., Ramanathan, V., Kerkez, V., Gonguet, V., Do, V., Vogeti, V., Petrovic, V., Chu, W., Xiong, W., Fu, W., Meers, W., Martinet, X., Wang, X., Tan, X. E., Xie, X., Jia, X., Wang, X., Goldschlag, Y., Gaur, Y., Babaei, Y., Wen, Y., Song, Y., Zhang, Y., Li, Y., Mao, Y., Coudert, Z. D., Yan, Z., Chen, Z., Papakipos, Z., Singh, A., Grattafiori, A., Jain, A., Kelsey, A., Shajnfeld, A., Gangidi, A., Victoria, A., Goldstand, A., Menon, A., Sharma, A., Boesenber, A., Vaughan, A., Baevski, A., Feinstein, A., Kallet, A., Sangani, A., Yunus, A., Lupu, A., Alvarado, A., Caples, A., Gu, A., Ho, A., Poulton, A., Ryan, A., Ramchandani, A., Franco, A., Saraf, A., Chowdhury, A., Gabriel, A., Bharambe, A., Eisenman, A., Yazdan, A., James, B., Maurer, B., Leonhardi, B., Huang, B., Loyd, B., Paola, B. D., Paranjape, B., Liu, B., Wu, B., Ni, B., Hancock, B., Wasti, B., Spence, B., Stojkovic, B., Gamido, B., Montalvo, B., Parker, C., Burton, C., Mejia, C., Wang, C., Kim, C., Zhou, C., Hu, C., Chu, C.-H., Cai, C., Tindal, C., Feichtenhofer, C., Civin, D., Beaty, D., Kreymer, D., Li, D., Wyatt, D., Adkins, D., Xu, D., Testuggine, D., David, D., Parikh, D., Liskovich, D., Foss, D., Wang, D., Le, D., Holland, D., Dowling, E., Jamil, E., Montgomery, E., Presani, E., Hahn, E., Wood, E., Brinkman, E., Arcaute, E., Dunbar, E., Smothers, E., Sun, F., Kreuk, F., Tian, F., Ozgenel, F., Caggioni, F., Guzmán, F., Kanayet, F., Seide, F., Florez, G. M., Schwarz, G., Badeer, G., Swee, G., Halpern, G., Thattai, G., Herman, G., Sizov, G., Guangyi, Zhang, Lakshminarayanan, G., Shojanazeri, H., Zou, H., Wang, H., Zha, H., Habeeb, H., Rudolph, H., Suk, H., Aspegren, H., Goldman, H., Damlaj, I., Molybog, I., Tufanov, I., Veliche, I.-E., Gat, I., Weissman, J., Geboski, J., Kohli, J., Asher, J., Gaya, J.-B., Marcus, J., Tang, J., Chan, J., Zhen, J., Reizenstein, J., Teboul, J., Zhong, J., Jin, J., Yang, J., Cummings, J., Carvill, J., Shepard, J., McPhie, J., Torres, J., Ginsburg, J., Wang, J., Wu, K., U, K. H., Saxena, K., Prasad, K., Khandelwal, K., Zand, K., Matosich, K., Veeraraghavan, K., Michelena, K., Li, K., Huang, K., Chawla, K., Lakhota, K., Huang, K., Chen, L., Garg, L., A, L., Silva, L., Bell, L., Zhang, L., Guo, L., Yu, L., Moshkovich, L., Wehrstedt, L., Khabsa, M., Avalani, M., Bhatt, M., Tsimpoukelli, M., Mankus, M., Hasson, M., Lennie, M., Reso, M., Groshev, M., Naumov, M., Lathi, M., Keneally, M., Seltzer, M. L., Valko, M., Restrepo, M., Patel, M., Vyatskov, M., Samvelyan, M., Clark, M., Macey, M., Wang, M., Hermoso, M. J., Metanat, M., Rastegari, M., Bansal, M., Santhanam, N., Parks, N., White, N., Bawa,

- N., Singhal, N., Egebo, N., Usunier, N., Laptev, N. P., Dong, N., Zhang, N., Cheng, N., Chernoguz, O., Hart, O., Salpekar, O., Kalinli, O., Kent, P., Parekh, P., Saab, P., Balaji, P., Rittner, P., Bontrager, P., Roux, P., Dollar, P., Zvyagina, P., Ratanchandani, P., Yuvraj, P., Liang, Q., Alao, R., Rodriguez, R., Ayub, R., Murthy, R., Nayani, R., Mitra, R., Li, R., Hogan, R., Battey, R., Wang, R., Maheswari, R., Howes, R., Rinott, R., Bondu, S. J., Datta, S., Chugh, S., Hunt, S., Dhillon, S., Sidorov, S., Pan, S., Verma, S., Yamamoto, S., Ramaswamy, S., Lindsay, S., Lindsay, S., Feng, S., Lin, S., Zha, S. C., Shankar, S., Zhang, S., Zhang, S., Wang, S., Agarwal, S., Sajuyigbe, S., Chintala, S., Max, S., Chen, S., Kehoe, S., Satterfield, S., Govindaprasad, S., Gupta, S., Cho, S., Virk, S., Subramanian, S., Choudhury, S., Goldman, S., Remez, T., Glaser, T., Best, T., Kohler, T., Robinson, T., Li, T., Zhang, T., Matthews, T., Chou, T., Shaked, T., Vontimitta, V., Ajayi, V., Montanez, V., Mohan, V., Kumar, V. S., Mangla, V., Albiero, V., Ionescu, V., Poenaru, V., Mihailescu, V. T., Ivanov, V., Li, W., Wang, W., Jiang, W., Bouaziz, W., Constable, W., Tang, X., Wang, X., Wu, X., Wang, X., Xia, X., Wu, X., Gao, X., Chen, Y., Hu, Y., Jia, Y., Qi, Y., Li, Y., Zhang, Y., Zhang, Y., Adi, Y., Nam, Y., Yu, Wang, Hao, Y., Qian, Y., He, Y., Rait, Z., DeVito, Z., Rosnbrick, Z., Wen, Z., Yang, Z., and Zhao, Z. The Llama 3 Herd of Models, 2024. URL <https://arxiv.org/abs/2407.21783>.
- Huang, Y., Cheng, Y., Bapna, A., Firat, O., Chen, D., Chen, M. X., Lee, H., Ngiam, J., Le, Q. V., Wu, Y., and Chen, Z. GPipe: Efficient Training of Giant Neural Networks using Pipeline Parallelism. In Wallach, H. M., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E. B., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pp. 103–112, 2019. URL <https://proceedings.neurips.cc/paper/2019/hash/093f65e080a295f8076b1c5722a46aa2-Abstract.html>.
- Kingma, D. P. and Ba, J. Adam: A Method for Stochastic Optimization, 2017. URL <https://arxiv.org/abs/1412.6980>.
- Korthikanti, V. A., Casper, J., Lym, S., McAfee, L., Andersch, M., Shoeybi, M., and Catanzaro, B. Reducing Activation Recomputation in Large Transformer Models. *Proceedings of Machine Learning and Systems*, 5: 341–353, 2023.
- Langley, P. Crafting papers on machine learning. In Langley, P. (ed.), *Proceedings of the 17th International Conference on Machine Learning (ICML 2000)*, pp. 1207–1216, Stanford, CA, 2000. Morgan Kaufmann.
- Li, Z., Zhuang, S., Guo, S., Zhuo, D., Zhang, H., Song, D., and Stoica, I. TeraPipe: Token-Level Pipeline Parallelism for Training Large-Scale Language Models. In Meila, M. and Zhang, T. (eds.), *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*, volume 139 of *Proceedings of Machine Learning Research*, pp. 6543–6552. PMLR, 2021. URL <http://proceedings.mlr.press/v139/li21y.html>.
- Liu, H., Zaharia, M., and Abbeel, P. Ring Attention with Blockwise Transformers for Near-Infinite Context. *arXiv preprint arXiv:2310.01889*, 2023.
- Megatron-LM-team. Megatron-lm. <https://github.com/NVIDIA/Megatron-LM>, 2024. Accessed: 2024/10/26.
- Narayanan, D., Harlap, A., Phanishayee, A., Seshadri, V., Devanur, N. R., Ganger, G. R., Gibbons, P. B., and Zaharia, M. PipeDream: generalized pipeline parallelism for DNN training. In Brecht, T. and Williamson, C. (eds.), *Proceedings of the 27th ACM Symposium on Operating Systems Principles, SOSP 2019, Huntsville, ON, Canada, October 27-30, 2019*, pp. 1–15. ACM, 2019. doi: 10.1145/3341301.3359646. URL <https://doi.org/10.1145/3341301.3359646>.
- Narayanan, D., Phanishayee, A., Shi, K., Chen, X., and Zaharia, M. Memory-Efficient Pipeline-Parallel DNN Training. *arXiv preprint arXiv:2006.09503*, 2020.
- Narayanan, D., Shoeybi, M., Casper, J., LeGresley, P., Patwary, M., Korthikanti, V. A., Vainbrand, D., Kashinkunti, P., Bernauer, J., Phanishayee, B. C. A., and Zaharia, M. Efficient Large-Scale Language Model Training on Gpu Clusters Using Megatron-LM. *ArXiv*, abs/2104.04473, 2021. URL <https://arxiv.org/abs/2104.04473>.
- Rajbhandari, S., Rasley, J., Ruwase, O., and He, Y. ZeRO: memory optimizations toward training trillion parameter models. In Cuicchi, C., Qualters, I., and Kramer, W. T. (eds.), *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, SC 2020, Virtual Event / Atlanta, Georgia, USA, November 9-19, 2020*, pp. 20. IEEE/ACM, 2020. doi: 10.1109/SC41405.2020.00024. URL <https://doi.org/10.1109/SC41405.2020.00024>.
- Ren, J., Rajbhandari, S., Aminabadi, R. Y., Ruwase, O., Yang, S., Zhang, M., Li, D., and He, Y. ZeRO-Offload: Democratizing Billion-Scale Model Training, 2021. URL <https://arxiv.org/abs/2101.06840>.
- Shazeer, N., Cheng, Y., Parmar, N., Tran, D., Vaswani, A., Koanantakool, P., Hawkins, P., Lee, H., Hong, M., Young,

C., Sepassi, R., and Hechtman, B. A. Mesh-Tensorflow: Deep Learning for Supercomputers. In Bengio, S., Wallach, H. M., Larochelle, H., Grauman, K., Cesa-Bianchi, N., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, pp. 10435–10444, 2018. URL <https://proceedings.neurips.cc/paper/2018/hash/3a37abdeef1dab1b30f7c5c7e581b93-Abstract.html>.

Shoeybi, M., Patwary, M., Puri, R., LeGresley, P., Casper, J., and Catanzaro, B. Megatron-LM: Training Multi-Billion Parameter Language Models Using Model Parallelism. *CoRR*, abs/1909.08053, 2019. URL <http://arxiv.org/abs/1909.08053>.

Touvron, H., Martin, L., Stone, K., Albert, P., Almahairi, A., Babaei, Y., Bashlykov, N., Batra, S., Bhargava, P., Bhosale, S., Bikell, D., Blecher, L., Ferrer, C. C., Chen, M., Cucurull, G., Esiobu, D., Fernandes, J., Fu, J., Fu, W., Fuller, B., Gao, C., Goswami, V., Goyal, N., Hartshorn, A., Hosseini, S., Hou, R., Inan, H., Kardas, M., Kerkez, V., Khabsa, M., Kloumann, I., Korenev, A., Koura, P. S., Lachaux, M.-A., Lavril, T., Lee, J., Liskovich, D., Lu, Y., Mao, Y., Martinet, X., Mihaylov, T., Mishra, P., Molybog, I., Nie, Y., Poulton, A., Reizenstein, J., Rungta, R., Saladi, K., Schelten, A., Silva, R., Smith, E. M., Subramanian, R., Tan, X. E., Tang, B., Taylor, R., Williams, A., Kuan, J. X., Xu, P., Yan, Z., Zarov, I., Zhang, Y., Fan, A., Kambadur, M., Narang, S., Rodriguez, A., Stojnic, R., Edunov, S., and Scialom, T. Llama 2: Open Foundation and Fine-Tuned Chat Models, 2023. URL <https://arxiv.org/abs/2307.09288>.

Xu, Y., Lee, H., Chen, D., Hechtman, B. A., Huang, Y., Joshi, R., Krikun, M., Lepikhin, D., Ly, A., Magoni, M., Pang, R., Shazeer, N., Wang, S., Wang, T., Wu, Y., and Chen, Z. GSPMD: General and Scalable Parallelization for ML Computation graphs. *CoRR*, abs/2105.04663, 2021. URL <https://arxiv.org/abs/2105.04663>.

A MEMORY CONSUMPTION ESTIMATION AND MEASURED TFLOP/S FOR PARALLELISM CONFIGURATIONS

A.1 A100 (40GB) Memory Consumption and TFLOP/s Measurements

Table 5 presents the memory consumption estimates for each parallelism configuration when training Llama-3.1-70B with a sequence length of 8,192 on A100 (40GB), as predicted by the memory consumption estimator. Additionally, Table 6 provides experimental results, including the observed TFLOP/s values for each configuration and whether an Out of Memory (OOM) occurred.

Table 5. Estimated Total Memory (GB) per GPU when training Llama-3.1-70B on A100(40GB) with sequence length 8,192. The vertical axis (TP, CP, PP, MBS) represents Tensor Parallel size (TP), Context Parallel size (CP), Pipeline Parallel size (PP), and Micro Batch Size (MBS), respectively. Red cells indicate settings that are estimated to exceed the 40GB GPU memory limit, likely resulting in Out of Memory (OOM) errors. Yellow cells denote configurations that may approach the limit due to temporary buffers and memory fragmentation, and green cells represent estimations at or below 80% of GPU memory (32GB or less), suggesting they are less likely to cause OOM.

(TP, CP, PP, MBS)	64 GPUs	128 GPUs	256 GPUs
(8, 1, 8, 1)	45.95	39.24	35.88
(8, 1, 16, 1)	-	34.48	33.76
(4, 2, 8, 1)	52.65	45.95	42.59
(4, 2, 16, 1)	-	41.2	37.48
(8, 2, 8, 1)	-	26.33	22.97
(8, 2, 8, 2)	-	39.24	35.88
(8, 2, 4, 1)	38.16	31.81	28.64
(8, 2, 4, 2)	50.94	44.6	41.42
(8, 4, 4, 1)	-	25.42	22.25
(8, 4, 4, 2)	-	31.81	28.64
(8, 4, 4, 4)	-	44.6	41.42
(8, 4, 2, 1)	43.32	37.16	34.08
(8, 4, 2, 2)	49.68	43.52	40.44

A.2 H100 (94GB) Memory Consumption and TFLOP/s Measurements

Tables 8, 9 show the estimated memory consumption, as predicted by the memory consumption estimator when training Llama-3.1-8B with sequence lengths of 16,386 and 32,768 on H100 (94GB). The measured TFLOP/s values for each comprehensive parallelism configuration in these settings are provided in Tables 10 and 11.

Table 6. Measured throughput(TFLOP/s) when training Llama-3.1-70B on A100 (40GB) with a sequence length of 8,192. The colors red, yellow, and green correspond to memory consumption levels predicted by the memory consumption estimator: red indicates configurations that likely exceed memory limits, yellow denotes configurations near the memory limit, and green represents configurations predicted to use 80% or less of GPU memory (32GB or less).

(TP, CP, PP, MBS)	64 GPUs	128 GPUs	256 GPUs
(8, 1, 8, 1)	OOM	OOM	172.31
(8, 1, 16, 1)	-	156.74	149.15
(4, 2, 8, 1)	OOM	OOM	OOM
(4, 2, 16, 1)	-	OOM	157.28
(8, 2, 8, 1)	-	156.11	148.02
(8, 2, 8, 2)	-	OOM	161.54
(8, 2, 4, 1)	OOM	170.3	166.32
(8, 2, 4, 2)	OOM	OOM	OOM
(8, 4, 4, 1)	-	126.6	123.89
(8, 4, 4, 2)	-	157.33	152.53
(8, 4, 4, 4)	-	OOM	OOM
(8, 4, 2, 1)	OOM	OOM	132.11
(8, 4, 2, 2)	OOM	OOM	OOM

Table 7. Estimated Total Memory (GB) per GPU when training Llama-3.1-8B on H100(94GB) with sequence length 8,192. The vertical axis (TP, CP, PP, MBS) represents Tensor Parallel size (TP), Context Parallel size (CP), Pipeline Parallel size (PP), and Micro Batch Size (MBS), respectively. Red cells indicate settings that are estimated to exceed the 94GB GPU memory limit, likely resulting in Out of Memory (OOM) errors. Yellow cells denote configurations that may approach the limit due to temporary buffers and memory fragmentation, and green cells represent estimations at or below 80% of GPU memory (75.2GB or less), suggesting they are less likely to cause OOM.

(TP, CP, PP, MBS)	4 GPUs	8 GPUs	16 GPUs	32 GPUs	64 GPUs
(2, 1, 1, 1)	67.52	56.30	50.69	47.89	46.48
(2, 1, 1, 2)	90.16	78.94	73.34	70.53	69.13
(2, 1, 1, 4)	135.45	124.23	118.62	115.82	114.42
(2, 2, 1, 1)	56.20	44.98	39.37	36.56	35.16
(2, 2, 1, 2)	67.52	56.30	50.69	47.89	46.48
(2, 2, 1, 4)	90.16	78.94	73.33	70.53	69.13
(2, 2, 1, 8)	135.45	124.23	118.62	115.82	114.42
(4, 1, 1, 1)	44.98	33.76	28.15	25.35	23.94
(4, 1, 1, 2)	56.30	45.08	39.47	36.67	35.27
(4, 1, 1, 4)	78.95	67.73	62.12	59.31	57.91
(2, 1, 2, 1)	54.41	43.19	37.58	34.77	33.37
(2, 1, 2, 2)	75.16	63.94	58.33	55.52	54.12
(2, 1, 2, 4)	116.66	105.43	99.83	97.02	95.62
(1, 2, 1, 1)	89.95	78.74	70.32	68.92	68.22
(1, 2, 1, 2)	112.60	101.38	95.77	92.97	91.56
(1, 4, 1, 1)	78.63	67.41	61.80	59.00	57.60
(1, 4, 1, 2)	89.95	78.74	73.13	70.32	68.92

Table 8. Estimated Total Memory (GB) per GPU when training Llama-3.1-8B on H100(94GB) with sequence length 16,384. The vertical axis (TP, CP, PP, MBS) represents Tensor Parallel size (TP), Context Parallel size (CP), Pipeline Parallel size (PP), and Micro Batch Size (MBS), respectively. Red cells indicate settings that are estimated to exceed the 94GB GPU memory limit, likely resulting in Out of Memory (OOM) errors. Yellow cells denote configurations that may approach the limit due to temporary buffers and memory fragmentation, and green cells represent estimations at or below 80% of GPU memory (75.2GB or less), suggesting they are less likely to cause OOM.

(TP, CP, PP, MBS)	4 GPUs	8 GPUs	16 GPUs	32 GPUs	64 GPUs
(2, 1, 1, 1)	90.16	78.94	73.34	70.53	69.13
(2, 1, 1, 2)	135.45	124.23	118.62	115.82	114.42
(2, 1, 1, 4)	226.03	214.81	209.20	206.40	205.00
(2, 2, 1, 1)	67.52	56.30	50.69	47.89	46.48
(2, 2, 1, 2)	90.16	78.94	73.34	70.53	69.13
(2, 2, 1, 4)	135.45	124.23	118.62	115.82	114.42
(2, 2, 1, 8)	226.03	214.81	209.20	206.4	205.00
(4, 1, 1, 1)	56.30	45.08	39.47	36.67	35.27
(4, 1, 1, 2)	78.95	67.73	62.12	59.31	57.91
(4, 1, 1, 4)	124.24	113.02	107.41	104.6	103.2
(2, 1, 2, 1)	75.16	63.94	58.33	55.52	54.12
(2, 1, 2, 2)	116.66	105.44	99.83	97.02	95.62
(2, 1, 2, 4)	199.66	188.44	182.83	180.02	178.62
(1, 2, 1, 1)	112.6	101.38	95.77	92.97	91.56
(1, 2, 1, 2)	157.89	146.67	141.06	138.26	136.85
(1, 4, 1, 1)	89.95	78.74	73.13	70.32	68.92
(1, 4, 1, 2)	112.60	101.38	95.77	92.97	91.56
(1, 4, 1, 4)	157.89	146.67	141.06	138.25	136.85
(2, 4, 1, 1)	-	44.98	39.37	36.56	35.16
(2, 4, 1, 2)	-	56.30	50.69	47.89	46.48
(4, 2, 1, 1)	-	33.76	28.15	25.35	23.94
(4, 2, 1, 2)	-	45.08	39.47	36.67	35.27

Table 9. Estimated Total Memory (GB) per GPU when training Llama-3.1-8B on H100(94GB) with sequence length 32,768. The vertical axis (TP, CP, PP, MBS) represents Tensor Parallel size (TP), Context Parallel size (CP), Pipeline Parallel size (PP), and Micro Batch Size (MBS), respectively. Red cells indicate settings that are estimated to exceed the 94GB GPU memory limit, likely resulting in Out of Memory (OOM) errors. Yellow cells denote configurations that may approach the limit due to temporary buffers and memory fragmentation, and green cells represent estimations at or below 80% of GPU memory (75.2GB or less), suggesting they are less likely to cause OOM.

(TP, CP, PP, MBS)	4 GPUs	8 GPUs	16 GPUs	32 GPUs	64 GPUs
(2, 1, 1, 1)	135.45	124.23	118.62	115.82	114.42
(2, 1, 1, 2)	226.03	214.81	209.20	206.40	205.00
(2, 1, 1, 4)	407.19	396.97	390.36	387.55	386.15
(2, 2, 1, 1)	90.16	78.94	73.34	70.53	69.13
(2, 2, 1, 2)	135.45	124.23	118.62	115.82	114.42
(2, 2, 1, 4)	226.03	214.81	209.2	206.40	205.00
(2, 2, 1, 8)	407.19	395.97	390.36	387.55	386.15
(4, 1, 1, 1)	78.95	67.73	62.12	59.31	57.91
(4, 1, 1, 2)	124.24	113.02	107.41	104.60	103.20
(4, 1, 1, 4)	214.81	203.59	197.99	195.18	193.78
(2, 1, 2, 1)	116.66	105.44	99.83	97.02	95.62
(1, 4, 1, 1)	112.6	101.38	95.77	92.97	91.56
(2, 4, 1, 1)	-	56.30	50.69	47.89	46.48
(2, 4, 1, 2)	-	78.94	73.34	70.53	69.13
(4, 2, 1, 1)	-	45.08	39.47	36.67	35.27
(4, 2, 1, 2)	-	67.73	62.12	59.31	57.91
(2, 2, 2, 1)	-	63.94	58.33	55.52	54.12
(1, 4, 2, 1)	-	75.15	69.55	66.74	65.34

Table 10. Measured throughput(TFLOP/s) when training Llama-3.1-8B on H100 (94GB) with a sequence length of 16,384. The colors red, yellow, and green correspond to memory consumption levels predicted by the memory consumption estimator: red indicates configurations that likely exceed memory limits, yellow denotes configurations near the memory limit, and green represents configurations predicted to use 80% or less of GPU memory (75.2GB or less).

(TP, CP, PP, MBS)	4 GPUs	8 GPUs	16 GPUs	32 GPUs	64 GPUs
(2, 1, 1, 1)	OOM	497.56	494.06	493.34	492.28
(2, 1, 1, 2)	OOM	OOM	OOM	OOM	OOM
(2, 1, 1, 4)	OOM	OOM	OOM	OOM	OOM
(2, 2, 1, 1)	425.29	432.15	420.1	388.7	365.75
(2, 2, 1, 2)	453.76	463.88	459.94	452.71	436.77
(2, 2, 1, 4)	OOM	OOM	OOM	OOM	OOM
(2, 2, 1, 8)	OOM	OOM	OOM	OOM	OOM
(4, 1, 1, 1)	442.54	448.18	438.37	406.59	384.14
(4, 1, 1, 2)	451.10	468.49	462.28	453.71	457.36
(4, 1, 1, 4)	OOM	OOM	OOM	OOM	OOM
(2, 1, 2, 1)	448.13	450.76	433.43	399.61	365.75
(2, 1, 2, 2)	OOM	OOM	OOM	OOM	OOM
(2, 1, 2, 4)	OOM	OOM	OOM	OOM	OOM
(1, 2, 1, 1)	OOM	OOM	OOM	OOM	OOM
(1, 2, 1, 2)	OOM	OOM	OOM	OOM	OOM
(1, 4, 1, 1)	OOM	461.64	439.89	409.41	363.59
(1, 4, 1, 2)	OOM	OOM	OOM	OOM	OOM
(1, 4, 1, 4)	OOM	OOM	OOM	OOM	OOM
(2, 4, 1, 1)	-	288.42	290.7	267.03	237.84
(2, 4, 1, 2)	-	272.96	293.26	279.53	273.22
(4, 2, 1, 1)	-	303.46	295.71	295.71	255.45
(4, 2, 1, 2)	-	297.73	299.95	299.95	282.24

Table 11. Measured throughput(TFLOP/s) when training Llama-3.1-8B on H100 (94GB) with a sequence length of 32,768. The colors red, yellow, and green correspond to memory consumption levels predicted by the memory consumption estimator: red indicates configurations that likely exceed memory limits, yellow denotes configurations near the memory limit, and green represents configurations predicted to use 80% or less of GPU memory (75.2GB or less).

(TP, CP, PP, MBS)	4 GPUs	8 GPUs	16 GPUs	32 GPUs	64 GPUs
(2, 1, 1, 1)	OOM	OOM	OOM	OOM	OOM
(2, 1, 1, 2)	OOM	OOM	OOM	OOM	OOM
(2, 1, 1, 4)	OOM	OOM	OOM	OOM	OOM
(2, 2, 1, 1)	OOM	336.94	308.35	288.00	229.30
(2, 2, 1, 2)	OOM	OOM	OOM	OOM	OOM
(2, 2, 1, 4)	OOM	OOM	OOM	OOM	OOM
(2, 2, 1, 8)	OOM	OOM	OOM	OOM	OOM
(4, 1, 1, 1)	363.68	340.51	318.83	290.18	234.85
(4, 1, 1, 2)	OOM	OOM	OOM	OOM	OOM
(4, 1, 1, 4)	OOM	OOM	OOM	OOM	OOM
(2, 1, 2, 1)	OOM	OOM	OOM	OOM	OOM
(1, 4, 1, 1)	OOM	OOM	OOM	OOM	OOM
(2, 4, 1, 1)	-	179.97	170.11	152.09	135.51
(2, 4, 1, 2)	-	178.79	173.15	162.30	151.39
(4, 2, 1, 1)	-	181.36	174.21	156.90	146.43
(4, 2, 1, 2)	-	181.56	176.28	162.30	145.65
(2, 2, 2, 1)	-	178.22	170.10	154.70	142.97
(1, 4, 2, 1)	-	169.49	170.70	151.90	131.50