Summary of “ZeRO-Infinity: Breaking the GPU Memory Wall for Extreme Scale Deep Learning”

Yi Chen (yichencs), Yunchi Lu (yunchi), Kaiwen Xue (kaiwenx)

Problem and Motivation

Large language model training has hit the GPU memory wall because the increase of GPU memory cannot match the scale of the increase of the size of the models. Despite the proposals of 3D parallelism, large models must still be trained with hundreds to thousands of GPUs. With this scale, data scientists without sufficient resources cannot fine-tune their models. They also need to significantly refactor their code to implement 3D parallelism. Therefore, there is need for a solution that can:

- Support future scaling of model sizes,
- Make the model accessible to data scientists without access to huge number of resources, and
- Make training large models easier.

Related Works

3D parallelism combines data, tensor, and pipeline parallelism. This involves GPipe, PipeDream, Megatron-LM, Tofu, etc. However, they require the programmers to refactor their code with non-trivial effort. To perform pipeline parallelism, models must be split into load-balanced pipeline stages, which is hard when the computational dependencies are complex. ZeRO and ZeRO-offload propose mechanisms to save memory in lieu of model parallelism. However, they have not systematically discussed overlapping communication and computation when large amounts of data is offloaded to CPU and NVMe memory. Other work such as Checkmate and Chen et al. describes using activation checkpointing to save the storage of activation. This brings additional challenges in terms of communication overhead.

Solution Overview

The paper first provides an analysis of the memory and bandwidth requirements of DL training. For memory, it characterizes the memory needed for model states, activation states, and working memory (minimum size of the activations and model states needed to sustain propagation at the largest layer). For bandwidth, it calculates the computational efficiency in terms of bandwidth, arithmetic density, and the peak computational throughput. It then describes empirically that how much bandwidth is needed for computation of each type of data to be overlapped with communication should the data be offloaded to slower memory.

The analysis forms the basis of the ZeRO-Infinity. First, to offload model states and activations, the paper proposes an infinity offload engine, which offloads all the partitioned model states. To save working memory, it uses memory-centric tiling, which effectively splits large operators to small operators to be processed sequentially. To improve training efficiency, it uses bandwidth centric partitioning that utilizes the connection bandwidth better than existing solutions such as ZeRO-Offload. It also uses overlap centric design to separately transfer all the needed data when they are needed but asynchronously through a dynamic prefetcher.

ZeRO-Infinity is evaluated by comparing against various baselines, including traditional data parallelism, Megatron-LM, and other state-of-the-art 3D parallelism methods. Using the same amount of GPU memory, ZeRO-Infinity can support training of 50x more parameters than the state-of-the-art. This allows democratizing the model training to common programmers who would like to fine-tune their model but do not have sufficient resources. Because ZeRO-Infinity relies on offloading to save memory, it can scale superlinearly with the more bandwidth provided by more GPUs. Memory tiling can be used to increase the maximum hidden dimension. Communication overlap results in speedup in training, compared with the state-of-the-art.

Limitations

One drawback of ZeRO-Infinity or all the ZeRO papers is they are extremely slow if not enough compute resources can be used, despite the various optimizations. One potential solution to that is to still train on large clusters, but have data scientists who do not have sufficient resources to fine-tune them.

Another limitation inherent to the offloading approach is that although ZeRO-Infinity can sustain the computational efficiency by leveraging high bandwidth connection, this approach might not work when the model is no longer transformer and has a more random data access pattern. The overhead in that case will be to manage the data storage, which will make communication overhead more complicated.

Future Research Directions

ZeRO-Infinity claims that device memory is no longer a limitation. In the future, researchers can focus more on improving the bandwidth utilization and the bandwidth itself. In addition, in the future, the GPUs will be more compute-efficient, making it harder to overlap computation with communication. This will be a challenge in the future, and recomputation or even hardware innovation can help with this problem.

Summary of Class Discussion

Q: What are the differences between original ZeRO and 3D parallelism?
A: Original ZeRO can be considered as a predecessor of FSDP implemented on DeepSpeed.

Q: Why use NVMe instead of some networked storage? At what point would the latter be too slow?
A: NVMe is way faster than networked storage. The reason such high bandwidth is needed is because if the data movement is not fast enough, computation and communication cannot be well-overlapped. However, in the future, advanced memory disaggregation technology might be employed to achieve even better cost efficiency while maintaining performance.

Q: Why would increasing the hidden dimension reduce the overhead of offloading activation checkpoint?
A: Larger hidden dimension will result in more compute and more GPU utilization. Once the hidden dimension is increased, more communication can be overlapped, but potentially at the cost of doing more computation.

Q: Explain what pinned memory is.
A: Pinned memory is that the memory is held in the memory (main memory or NVMe) so that the CPU will not swap them to the disk. Since the memory is guaranteed not to be evicted, accessing them will be fast enough to achieve the overlapping of communication and computation.

Q: User experience says ZeRO-Infinity is very slow. Will it really democratize large model training? What if the models increase even more?
A: The user does not have to train the whole model. Instead, they can focus on fine-tuning, which is much faster while able to improve training performance. In the future, should the model become even larger, GPU-GPU communication, which is not considered problematic, will also potentially become a bottleneck.

Q: How can ZeRO-Infinity be combined with recomputation techniques?
A: When the model becomes even larger, a larger batch size must be used so that the model can converge. This will make the communication of even higher overhead. More recomputation can reduce the overhead in this regard.

Summary of “Reducing Activation Recomputation in Large Transformer Models”

Yi Chen (yichencs), Yunchi Lu (yunchi), Kaiwen Xue (kaiwenx)

Problem and Motivation

In training large LLMs, activation recomputation is commonly used to mitigate memory capacity constraints. By checkpointing activations of a subset of layers while re-computing the rest necessary activations in the backward pass, machines use those saved memory to fit a larger model. Yet recomputation is costly and mostly unnecessary, since some of their memory consumptions can be effectively reduced even without recomputation. The paper proposes *selective recomputation* that recomputes a selected set of layers to improve performance. Additionally, the paper proposes sequence parallelism, which in conjunction optimizes the transformer architecture.

Related Works

There is much work addressing training large models in a distributed manner. As the fundament, model parallelism involves *tensor parallelism* and pipeline parallelism. Yet in Megatron, not all tensors are partitioned across tp ranks, some of them are replicated. To address the issues, *sequence parallelism* is suggested to further partition activations along the sequence dimensions. The method originally requires parameters and optimizer states to be replicated, so it does not scale well for large models. Alternatively Sagemaker and GSPMD also propose memory efficient versions of tp, but the involvement of *multi-device layer normalization* incurs too much communication overhead. This paper proposes an optimal solution that combines benefits from tensor parallelism and sequence parallelism, while getting rid of their drawbacks. The main insight is that the communication overhead in distributed layernorm can be eliminated by sequence parallelism. As a result, it reduces the activation memory significantly thus saving a lot of additional recomputation, making training performance much faster.

Solution Overview

The paper starts by deriving an approximate formula for the memory required to store activations in the forward pass of a single stack transformer model, assuming none parallelism is applied. With the help of tp, only a part of memory consumption can be reduced by the number of tp ranks *t*. Also, additional communication primitives (all-reduce + identity/no-op) need to be injected before and after the tp block.

It is noticed that the layernorms and dropouts are left intact and replicated. Since operations are independent along the sequence dimension, the paper suggests sequence parallelism can help further parallelize those layers. Yet the sequence parallelism also needs to inject additional

communication primitives (all-gathers) around its blocks as converters between sp and tp, which introduce much overhead that will slow down the training.

The paper found that the additional collective communications from tp and sp can be combined together into existing efficient communication primitives: all-gather in forward pass, and reduce-scatter in backward pass. Under this design, the original memory consumption formula is now fully reduced by the number of tp ranks t.

The required activation memory under the above design is still large. So selective activation recomputation is applied. The idea is to only checkpoint enough activations, not all, to allow a given model -parallel configuration to train on the given device. And prioritize recomputing the layers that require much memory space but are not computationally expensive.

Limitations

The choice of layers for checkpointing or recomputation is manually done. This not only requires manual efforts, but also is not instructive to users. Also, the manual search only guarantees a near-optimal execution plan.

The paper only compares the design with model parallelism. It is not compared with other techniques' training performance, e.g. sharding model states, and CPU offloading. It is less instructive for readers to choose among different techniques that address memory constraints while optimizing training performance.

Future Research Directions

The current selective activation recomputation is based on manual search for the trade-off between memory constraints and recomputation overheads. Exploring automated methods deserves future work.

Researchers also plan to further reduce the activation memory by resolving the issues caused by memory fragmentation for large micro batches and non-uniform memory allocation due to pipeline parallelism.

Summary of Class Discussion

Q: Are there any additional communication overhead introduced by changing (all-reduce + no-op) to (all-gather + reduce-scatter)?
A: The original all-reduce is equivalent to reduce-scatter followed by an all-gather. So the change in communication is doing almost the same amount of work except for splitted and in a different order. So the communication cost should be the same.

Q: Does selective recomputation affect accuracy for training?

A: No. Selective recomputation is a technique optimizing runtime memory consumption. There is no approximation or violation on the math equivalence. Thus training accuracy should not be affected.

Q: Under memory constraints, how to choose selective checkpointing vs CPU-offloading? Any possibility of combining this paper's technique with Zero-infinity or other techniques?
A: This paper proposes two techniques: sp + tp, and selective activation recomputation. For the first, it comes almost at no cost (compared with only applying tp), so always applicable to transformer models. For the second, it is a performance-improved version of activation checkpointing. So both checkpointing and CPU offloading incur much overhead in order to fit large models. Yet, whether one would outperform the other under certain conditions is not well evaluated in this paper. Also, it is possible to combine sp+tp with Zero-infinity that offload states to CPU memory and NVMes, and leverages overlapped computation for performance optimizations.