Summary of ZeRO: Memory Optimizations Toward Training Trillion Parameter Models

1 The problem the paper is trying to solve

The paper addresses the memory limitations in training extremely large neural network models, particularly in a data parallel setting. As model sizes grow, the memory required to store model parameters, gradients, and optimizer states becomes a bottleneck, limiting the size of models that can be trained on available hardware. The goal is to enable training of trillion-parameter models using data parallelism without sacrificing computational efficiency.

2 Impact of the work

The ZeRO optimizer has had a significant impact on large-scale model training:

- It enabled training of much larger models on existing hardware by reducing per-GPU memory requirements.
- The work led to the development of the widely-used DeepSpeed framework.
- It influenced the design of PyTorch's FSDP (Fully Sharded Data Parallel) API.
- While not fully realized in practice, it theoretically enables training of trillion-parameter models.

3 Main proposed ideas

- ZeRO-DP: Sharding model states, gradients, and optimizer states across GPUs in data parallel training.
- ZeRO-R: Partitioning activation memory in model parallel training.
- Optimized communication using buffers to improve efficiency.
- Memory defragmentation techniques to manage GPU memory more effectively.

4 Strengths and weaknesses

Strengths:

- Enables training of larger models with existing hardware.
- Achieves good scaling efficiency, even demonstrating super-linear speedup in some cases.
- Compatible with existing model parallel techniques like Megatron-LM.
- Relatively simple concept that is easy to understand and implement.

Weaknesses:

- The paper only considers how to improve the training speed by increasing the batch size, but ignores the fact that increasing the batch size may lead to slower convergence. In other word, increasing the batch size brings performance gains, but may affect the speed of model convergence.

- The largest model actually trained (17B parameters) is much smaller than the theoretical trillion-parameter model discussed.
- Baseline comparisons against Megatron-LM for very large models may not be entirely fair. This comparison can be misleading because ZeRO uses a different approach to data parallelism than other systems use for model parallelism, so a direct comparison of model sizes is not fair

5 Future Directions

- Exploring the practical limitations of training trillion-parameter models, including data and computational constraints.
- Investigating the integration of ZeRO with other parallelism techniques for even larger models.
- Studying the impact of extreme sharding on model convergence and training dynamics.
- Developing more sophisticated memory management techniques to further reduce fragmentation.
- Exploring adaptive sharding strategies that adjust based on workload and available resources.

6 Understanding Superlinear Speedup in Model Training

The speed of model training doesn't just increase linearly; sometimes, a phenomenon of "superlinear speedup" can occur. Here's a step-by-step explanation:

1. **Theoretical Linear Growth**:
   - Theoretically, increasing the number of GPUs should result in linear growth of training speed. For example, increasing from 64 to 128 cards should double the speed; from 128 to 256 cards, the speed should double again; and when increasing to 400 cards, theoretically, the expected speedup should be close to 8 times.
   - However, in reality, the growth in training speed is not strictly linear.
2. **Reasons for Potential Superlinear Speedup**:
   - When using more GPUs (e.g., 128 GPUs), parts of the model can be distributed across more GPUs, making the portion each GPU is responsible for smaller. This reduces the model parameters each GPU needs to maintain, freeing up more memory.
   - With more available memory, it's possible to increase the **batch size** processed by each GPU, which brings two benefits:
     (a) **Better utilization of computing resources**: Increasing batch size means more data is processed in each computation, allowing for better utilization of GPU compute units (cores), improving the computational efficiency of each GPU.
     (b) **Improved computation/communication ratio**: The batch size increases, but the amount of communication doesn't change (communication is a necessary step in model training, such as distributing model weights to various GPUs). More computation with unchanged communication time means an improved ratio between computation and communication, resulting in better overall performance of the distributed system.
3. **Specific Impact**:
   - The increase in batch size allows each GPU to process more samples per second, thus speeding up computation. Although larger batch sizes might increase computation time, since communication time remains constant, computation and communication can overlap better, improving the efficiency of distributed training.
4. **Potential Drawbacks**:
   - While increasing batch size accelerates training speed, it may affect the **convergence rate**, i.e., the speed at which the model reaches desired accuracy. Too large a batch size can lead to slower convergence. Therefore, the issue of model convergence speed is not considered here; the focus is on how to accelerate training by increasing GPUs and batch size.

5. **Uncertainty of Superlinear Growth**:

   - Although in some cases, this strategy of increasing batch size can lead to superlinear speedup, in practical computations, due to potential issues like slower convergence, the overall effect may vary. Nevertheless, this acceleration method is still considered a highlight.