**Introduction**

Transformers are the go-to architecture for many top AI models due to their outstanding performance across various applications. However, they struggle with memory demands when handling long sequences like videos and complex tasks. This paper introduces a new approach called Ring Attention with Blockwise Transformers, which improves how long sequences are managed across multiple devices. This method allows for much longer sequences without additional memory or computation costs, enabling better performance in tasks requiring large contexts.

**Key Concepts**

- **Transformers**: A popular AI model architecture known for its ability to handle diverse tasks using self-attention and feedforward mechanisms.

- **Memory Challenges**: Traditional Transformers require a lot of memory to process long sequences, limiting their application in handling extensive data.

- **Blockwise Computation**: A method to process sequences in smaller chunks (blocks) to distribute memory and computation needs across multiple devices.

- **Ring Attention**: A new technique that overlaps the computation and communication of these blocks among devices, significantly extending the sequence length Transformers can handle.

**Approach**

- **Blockwise Processing**: By breaking down the sequence into smaller blocks, each device handles a portion, sharing key-value blocks in a ring-like structure.

- **Overlap Communication and Computation**: Devices send and receive blocks simultaneously, ensuring no additional overhead compared to traditional methods.

- **Scalability**: This approach allows the sequence length to scale with the number of devices, achieving near-infinite context size.

**Results**

Experiments show that Ring Attention dramatically reduces memory requirements, enabling sequences up to 100 million tokens long without approximations. This method supports both training and inference on much longer sequences than previously possible.

**Contributions**

1. A new Transformer architecture that scales context length linearly with the number of devices, removing individual device memory constraints.

2. Demonstrating the method's effectiveness through extensive experiments in language modeling and reinforcement learning.

**Explanation of Ring Attention**

**Ring Attention** is a technique designed to handle very long sequences in Transformers by dividing the work among multiple devices efficiently. Here's how it works:

1. **Blockwise Computation**: The sequence is divided into smaller blocks, and each block is processed independently. This reduces the memory load on any single device.

2. **Distributed Processing**: Each device in a network handles one or more blocks, processing them in parallel.

3. **Ring Structure**: Devices form a ring where they pass blocks of data (key-value pairs) to their neighbors. Each device sends its processed data to the next device in line and receives data from the previous one.

4. **Overlap of Computation and Communication**: While a device processes its blocks, it simultaneously sends and receives blocks from other devices. This overlapping ensures that there's no waiting time, making the process as fast as traditional methods but much more memory efficient.

By using this approach, Ring Attention allows Transformers to handle sequences that are much longer than what was previously possible, without requiring more memory or computation time. This is especially useful for tasks that involve large amounts of data, such as language modeling and processing long videos or texts.

**Memory Constraints in Large Contexts**

When working with long sequences in Transformers, there are significant memory challenges. Here's a breakdown of the issues and current solutions:

1. **Self-Attention Computation**:

- **Inputs**: We have sequences Q, K, V (Query, Key, and Value) with a length s and dimension d .

- **Formula**: The output is calculated using Attention **(Q, K, V) = softmax (QKT/√d)**

- **Feedforward Network**: Each position in the sequence also passes through a feedforward network, which applies two linear transformations with a ReLU activation in between: **FFN(x) = max(0, xW1 + b1)W2 + b2.**

2. **Blockwise Parallel Transformers**:

- **Memory Efficiency**: Previous advancements reduced memory usage by computing attention in blocks rather than all at once. This means processing parts of the sequence separately and then combining them.

- **Memory Overhead**: This method brings down the memory requirement to 2bsh bytes per layer, where b is the batch size, s is the sequence length, and h is the hidden size of the model.

- **Feedforward Network in Blocks**: The Blockwise Parallel Transformers (BPT) method further reduces memory by applying the feedforward network in blocks, lowering the activation size from 8bsh to 2bsh.

3. **Storage Challenge:**

- **Layer Outputs**: Despite the memory savings, storing the output of each layer is still necessary. This is because self-attention involves interactions among all sequence elements, and not storing these outputs would mean re-computing them repeatedly, which is impractical.

- **Memory Requirements**: For example, processing 100 million tokens with a batch size of 1 for a model with a hidden size of 1024 requires over 1000GB of memory. Modern GPUs and TPUs typically have less than 100GB of high-bandwidth memory (HBM), making this requirement difficult to meet due to physical and cost constraints.

**Explanation of Memory Constraints and Solutions**

When dealing with long sequences in Transformers, memory usage is a major issue because of how self-attention and feedforward networks work. Here's a simplified explanation:

1. **Self-Attention**:

- This is a key part of Transformers, where each part of the sequence (like words in a sentence) looks at every other part to understand the context.

- The computation involves large matrices that grow with the sequence length, quickly consuming a lot of memory.

2. **Blockwise Computation**:

- To handle this, researchers divide the sequence into smaller blocks and process these blocks separately. This approach saves memory because we don't need to keep the entire sequence in memory at once.

- **Example**: Imagine reading a book in sections instead of trying to hold the entire book in your hands.

3. **Feedforward Network**:

- After self-attention, each part of the sequence goes through a feedforward network that processes it individually. By also processing this part in blocks, memory usage is further reduced.

4. **Challenges with Long Sequences**:

- Even with these blockwise methods, storing the outputs of each layer is still necessary because self-attention needs to reference previous layers' outputs.

- Without storing these, we'd need to re-compute everything from scratch for each layer, which isn't feasible.

- For very long sequences, the required memory can exceed the capacity of current hardware, creating a significant bottleneck.

In summary, while blockwise computation significantly reduces memory demands, the need to store outputs for each layer still limits how long sequences can be without exceeding the memory limits of modern hardware.

**RingAttention with Blockwise Parallel Transformers**

The aim is to make a system that can handle really long sequences of data across multiple devices without slowing down or using up too much memory. Here's how we do it:

1. **Blockwise Parallel Transformers (BPT):** We start with a framework called BPT, which splits up the sequence into blocks and lets each device handle its own block separately. This way, each device can work on its part without needing to talk to the others for most operations.

2. **Ring-Based Blockwise Attention:** Now, here's the tricky part. Normally, when devices need to talk to each other to get information, it slows things down and uses up memory. So, we use a clever trick based on the idea that it doesn't matter what order devices talk to each other in for certain operations. We organize the devices in a ring shape, like a circle. Each device talks to the one next to it in the circle while also getting information from the one before it. This way, the devices can keep working while they share information, without slowing down.

3. **Calculating the Right Block Size:** We need to make sure the size of each block is just right, so devices can keep working smoothly without waiting too long for information. We figure this out based on how fast each device can do calculations and how fast they can share information.

4.**Memory Requirement:** Each device needs to store some blocks of data while it's working. We calculate how much memory each device needs and make sure it's not too much.

5. **Algorithm and Implementation:** We lay out the steps for how the system works, from splitting up the data to doing the calculations and sharing information between devices.

Overall, this method lets us handle really long sequences of data across multiple devices without slowing down or using too much memory.

**Setting**

We're testing how much better Transformer models perform when we use Ring Attention. We're looking at two main things: the maximum length of sequences the models can handle and how efficiently they use computing power.

1. **Model Configuration:** We're using a specific architecture called LLaMA, and we're testing four different sizes of models: 3B, 7B, 13B, and 30B. These numbers indicate the complexity and size of the models we're dealing with.

2. **Baselines:** To see how much our method improves things, we compare it with different setups. We look at vanilla transformers, which are the basic versions, along with transformers that use memory-efficient attention and ones that use both memory-efficient attention and feedforward. These comparisons help us see exactly where our method shines.

3. **Training Configuration:** We're using a technique called full gradient checkpointing, which helps manage memory usage during training. We're applying this technique to both the attention and feedforward parts of the models. We're testing our setups on both GPUs and TPUs, considering different configurations like single GPUs and distributed setups with multiple GPUs. We're also testing on different generations of TPUs, from older versions to newer ones. Importantly, all our results are based on using full precision, not mixed precision, in computations.

Overall, we're putting Ring Attention to the test against various setups to see how it improves both the handling of long sequences and the efficient use of computing power in Transformer models.

**Results:**

In our experiments, we aim to thoroughly evaluate Ring Attention's performance across various metrics. We compare it with several baseline models, including vanilla transformers, transformers with memory-efficient attention, and transformers with both memory-efficient attention and feedforward. Our evaluation covers different model sizes and accelerator configurations.

**Evaluating Max Context Size:**

We assess the maximum supported context length using fully sharded tensor parallelism (FSDP). This method ensures a fair comparison by maintaining the same batch size in tokens for both baselines and our approach. We utilize FSDP along with Ring Attention to extend the sequence length significantly, while keeping the total batch size in tokens constant.

**Evaluating Model Flops Utilization:**

We evaluate the model flops utilization (MFU) of Ring Attention using standard training settings. Our goal is to understand the impact of model size and context length on MFU, a critical performance metric.

**Impact on In-Context RL Performance:**

We apply Ring Attention to learning trial-and-error RL experiences using Transformers. We evaluate our proposed model on the ExoRL benchmark across six different tasks, showing improved performance compared to baseline models.

**Impact on LLM Performance:**

We evaluate Ring Attention by applying it to finetune LLaMA models to longer contexts. Despite limitations in compute budget, our approach enables training with millions of context tokens, leading to improved accuracy in tasks requiring long-context understanding.

**Related Work:**

We discuss prior research on memory-efficient techniques for Transformers and various parallelism methods, highlighting the novelty and effectiveness of our approach.

**Conclusion:**

In conclusion, our memory-efficient approach enables training with longer context lengths, surpassing previous limitations. We demonstrate its effectiveness across language modeling and reinforcement learning tasks, paving the way for exciting future applications in various domains.