Review ’FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness’

Adin Mauer - Student ID: 43514439

I. Summary

One of the main problems with Transformer models is that they have a large memory footprint and that footprint increases quadratically relative to sequence length. While previous attempts to accelarate and optimize Transformer models (using sparsity [keyformer], low-rank or other methods) attempted to improve performance and memory footprint by focusing on FLOP reduction, the FlashAttention paper presents a very successful solution by addressing the memory-bound elements of the algorithm instead of the compute-bound components of the algorithm. The reasons transformer algorithms have become memory-bound instead of compute-bound are:

1) HMB bandwidth is 1.5-2.0 TB/s while the GPU SRAM bandwidth is 19.0 TB/s, requiring high arithmetic intensity for efficiency.
2) The standard attention implementation naively requires $\Theta(Nd + N^2)$ HBM accesses compared to FlashAttention’s $\Theta(N^2d^2M^{-1})$ (note that $d^2 << M$)

FlashAttention does not modify the Transfomer model architecture, rather it addresses the execution mechanics of the central operations related to a single layer of a Transformer:

$$\text{Softmax}(QK^T)V$$

. Essentially, the optimization uses **matrix tiling**, **kernel fusion** and **recomputation** to maximize possible IO-aware locality and arithmetic intensity. **matrix tiling** and **kernel fusion** operations cannot be done in a naive way because intermediate values are required to compute gradients on the backward pass, so FalshAttention introduces and small overhead of $O(N)$ additional memory footprint to keep track of some local statistics that enable recomputation of the necessary matrcies on the backward pass (thereby avoiding the need to send them through HBM to store the matrices in memory).

Beyond this basic notion of IO-aware Attention, the paper extends its optimization by showing that adding Block-Sparsity to the algorithm improves runtime by a factor that is proportional to the fraction of non-zero blocks in the sparsity mask. The runtime improves because there is less computation and less HBM accesses. With the block-sparse implentation that number of HBM accesses goes to $\Theta(Nd + N^2d^2M^{-1}s)$, where s is the fraction of nonzero blocks.

The improvements in FlashAttention provide better runtimes, but they also present an opportunity to improve the quality of the model. Deep learning models enjoy the rare quality of improving proportionally to scale, and the smaller memory footprint for FlashAttention allows for running transformers with longer sequences on the same GPU, making the model more powerful.

The novelty in the paper comes from the well-rounded approach to algorithm implementation on hetrogenous computing. Tiling is a standard accelaration optimization for matrix multiplcation, and kernel fusion, in this case, is non-trivial because it required some additional considerations for both numerical stability and maintaing intermediate statistics for the backward pass. This kind of appoach, where the algorithm code is written and tailored to the specific system the code is running on is important when the runtimes become bound by the computer/system architecture (and not the algorithm computation itself).

II. Main Strengths

Who will this interest?

Anyone who is **only** interested in the ML/algorithmic side of Transformer architecture and algorithms will not find the insights in this paper of much interest. Alternatively, anyone who is interested in deploying models that use Transformers in the real world will definitely be interested in the results and methods used in this paper. At the point of time of reading this paper, some tech companies are exploring building nuclear power plants to power their data-center which train and deploy their LLMs (transformer models). This in itself is an expample that shows how likely is it that any imporvement of performance, runtime, or memory footprint of transformers is of great interest to many people today.

Strong Results

The strength of the paper really comes from the strong results which prove directly viability of the FlashAttention algorithm. It is shown that the **memory footprint** no longer scales quadratically with sequence but rather it scales linearly with FlashAttention. The **memory footprint** improves 20x in FlashAttention relative to exact attention (standard attention). Additionally, it is shown that - on average - the **runtime** improves 3x compared to standard attention with common sequence lengths. The performance of FlashAttention is also impressive, showinw that it performs on par with standard attention with similar sequence lengths and improves performance with the longer sequence lengths that it can now use.

To evaluate FlashAttention, the model was run with varying sequence length while measuring runtime and memory-footprint (number of HBM accesses). Then those metrics were assessed in the following ways:

1) The model runtime and footprint was compared relative to other implementations (standard attention, approximate attention, Linformer, etc.)
2) The model was benchmarked against scores like Perplexity and Path-X/Path-256. It is notable that FlashAttention is the first model to solve Path-X/Path-256 with non-random results (61.% accuracy).

III. Main Limitations

The main limitation in the paper is that the implementation of flash attention is highly system-architecture specific, and there is no generalized approach to provide a clear API for every Transformer architecture and every system. This means that the actual usage of FlashAttention requires significant engineering work and careful CUDA kernel implementations. Additionally, although the IO-aware approach seems to be novel, the fact that it is novel is a weakness. ML researchers neglect to address the HW/SW interface of their algorithmic implementations. This approach is only the beginning, and it is evident because at the time of writing this review FlashAttention2.0 and 3.0 have been released with further 2x and 4x average improvements, respectively.

Another more specific limitation is that even though it is trivial to parallelize several heads in a layer it is unclear how to parallelize single-head implementations of FlashAttention. In the algorithm, the matrix blocks/tiles are loopes over in 2 nested loops. It is hard to interpret the nested code as the code for a potential CUDA kernel because there are dependencies in between interations of the loop.

IV. Improvements

Some improvements to this paper may include:

1) Parallelize single heads in a layer by using scatter-gather methods of the local statistics (l and m vectors) and the O matrix.
2) Include nested tiling to take advantage of multiple memory hierarchy levels
3) Implement multi-gpu FlashAttention using non-shared memory parallel programming paradigms (like MPI).