15-418/618 Fall 2023 Project Milestone Report
Sparse Attention in CUDA

Full Names: Sarah Di, Jinsol Park

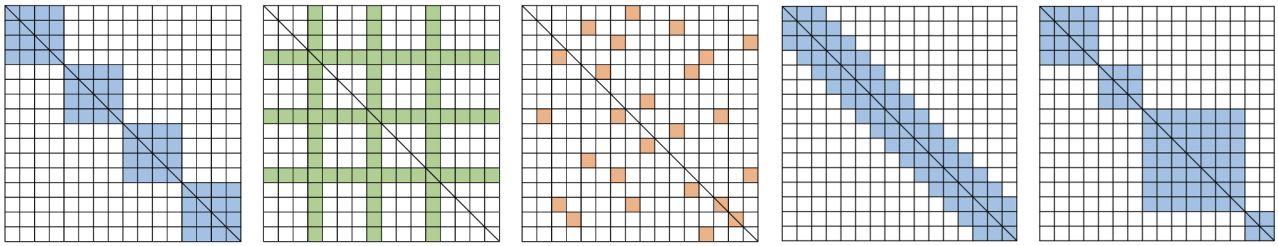
Andrew IDs: sarahdi, jinsolp

Project Page URL: `https://github.com/disarah/15618_Project`

1 Work Completed so far

1.1 Summary of work

Following the work of Child et al. [2019] with Sparse Attention Transformers, we first validate the motivation for sparse attention with empirical data. The graphs which confirm that attention is a performance bottleneck can be seen in the Preliminary Results section.



(a) Window (b) Global (c) Random (d) Sliding Window (e) Variant-sized Window

Figure 1: Different factorization methods of sparse attention. Each connectivity matrix shows whether an i^{th} token (row) of a sequence refers to a j^{th} token (column) to compute an attention score. These can be combined to be a single sparse attention method.

We implemented the naive attention and the sparse attention on CPU and CUDA. For the naive attention, we implemented general matrix multiplication and the softmax using a tiling scheme to exploit shared memory efficiently.
For this milestone, we focused on the window attention method as seen in (a) of Figure 1 for sparse attention. Since query and key matrices are dense but the resulting attention scores matrix is sparse, we cannot use general matrix multiplication. Instead, we implemented a `MultiHeadDDS` kernel, which performs dense $\times$ dense = sparse matrix multiplication based on the given window size. We also implemented a `MultiHeadSoftMaxSparse` kernel, which performs sparse softmax on the sparse attention scores matrix. Finally, we also have a `MultiHeadSDD` kernel to perform matrix multiplication between the sparse attention scores matrix and the dense value matrix, returning a dense matrix as the final output. All matrix multiplication kernels use tiling optimizations, and uses the window size configuration to efficiently exploit shared memory.

1.2 Being on Schedule

We believe we will be able to finish all "plan to achieve" goals. We also plan to work on the "hope to achieve" goals for the remaining time. A specified schedule for the rest of the semester is in section 2

1.3 Plans for the Poster Session

At the poster session, we are planning on showing multiple graphs showing the speedup of our implementations with respect to sequence length. Additionally we will include visualizations of the final tiling methods used and visualizations of how shared memory is exploited in our CUDA implementations.

1.4 Preliminary Results

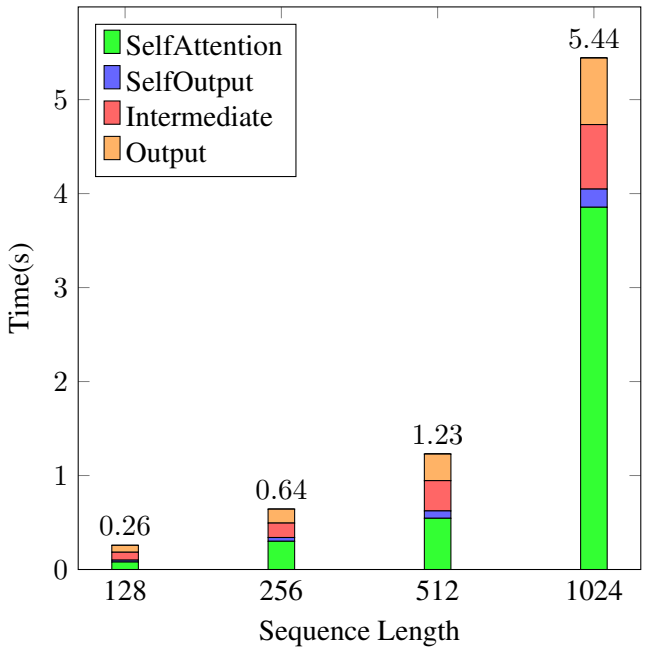


Figure 2: Computational Times for layers in a transformer vs Sequence Length

In Figure 2, we show the computational time of the layers present within a BERT encoder scaled by the sequence length of the inputs (ranging from 128 to 1024). Our visualization confirms that the computation time of the encoder's attention layer, as shown in green, scales quadratically with respect to sequence length, and also that is takes up a large proportion of the computation time. We also include the computational times of other layers of the encoder, such as the Intermediate and Output (Add & Norm) layers, as shown in red and orange respectively, which combined comprise the Feed Forward part of an encoder. The SelfOutput layer, shown in blue, represents the Add & Norm Layer of Self Attention.

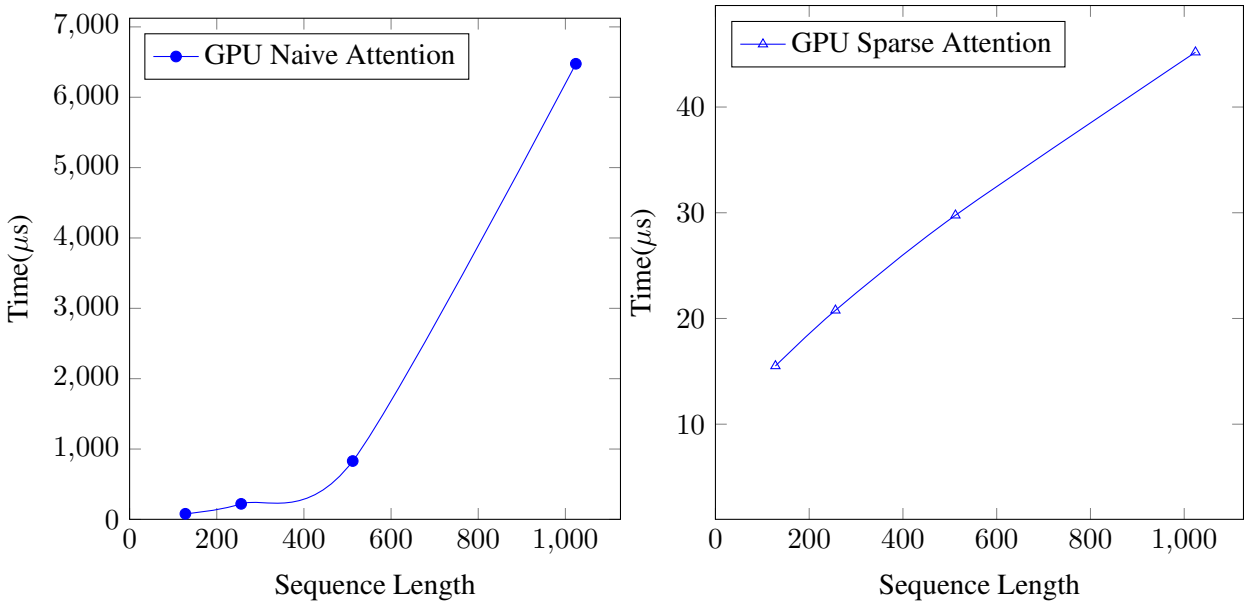


Figure 3: Execution Time vs Sequence Length for CPU and GPU implementations

Figure 3 shows the execution time of attention on GPU for different attention methods. We measured the execution time for sequence lengths 128, 256, 512, and 1024. It can be observed that the GPU Naive Attention scales quadratically with sequence length, while the GPU Sparse Attention implementation (with window size 16) scales linearly with sequence length. Also, it takes much less time compared to the full naive attention.

SeqLen	GPU Sparse	GPU Naive	CPU Sparse	CPU Naive
128	$\times$1	$\times$5.15	$\times$51.02	$\times$476.46
256	$\times$1	$\times$10.63	$\times$87.01	$\times$1238.01
512	$\times$1	$\times$27.88	$\times$181.65	$\times$3706.75
1024	$\times$1	$\times$143.32	$\times$332.90	$\times$8652.24

Table 1: Speedup of GPU window sparse implementation compared to other implementations

Table 1 provides further information about the execution time on CPU Sparse and Naive Attentions. We provide a speedup brought by GPU Sparse Attention compared to each implementation for different sequence lengths.

1.5 Remaining Challenges

We plan to implement additional sparse attention mechanisms from Figure 1. Unlike the window attention which intuitively aligns well with our tiling optimization, other versions are more difficult to optimize by exploiting shared memory.

Also, one of our "hope to achieve" plans is to use CUDA profiling tools (such as NSight Compute or Systems). Since neither of us are familiar with these tools, we will be treating this as an opportunity to learn, even if our kernel profiling results are unsuccessful.

2 Revised Schedule

Our finalized schedule for the remainder of the semester is as follows:

Time	Plan	Person
12/03	Milestone Report Due	Jinsol & Sarah
-	Have 1 CUDA implementation & benchmark	-
12/06	Read literature about different sparse attention mechanisms	Jinsol & Sarah
12/10	Additional CUDA implementations & optimizations	Jinsol & Sarah
-	Complete Background on Final Report	-
12/13	Finish Experiments/Discussion on Final Report	Jinsol & Sarah
12/14	Final Report Due	Jinsol & Sarah
-	Finish Poster	-
12/15	Poster Session	-

References

Rewon Child, Scott Gray, Alec Radford, and Ilya Sutskever. Generating long sequences with sparse transformers. *arXiv preprint arXiv:1904.10509*, 2019.