15-418/618 Fall 2023 Final Project Proposal
Sparse Attention in CUDA

Full Names: Sarah Di, Jinsol Park

Andrew IDs: sarahdi, jinsolp

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

1 Summary

We are going to implement sparse attention on CPU and GPU platforms using python and CUDA respectively, and perform analysis on both system's performance characteristics.

2 Background

Attention requires time and memory that that scales quadratically with sequence length Child et al. [2019].

Sparse attention is similar to the original attention, but it factorizes the attention matrix to compute certain parts of the matrix instead of computing the entire $n \times n$ matrix. Different factorization methods lead to different sparse attention modes.

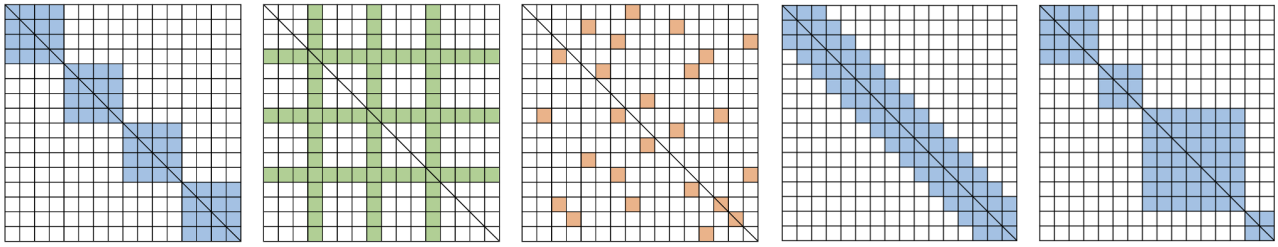

Figure 1: Different factorization methods of sparse attention. Each connectivity matrix shows whether an i^{th} token (row) of a sequence refers to a j^{th} token (column) to compute an attention score. These can be combined to be a single sparse attention method.

Figure 1 illustrates different sparse attention factorization methods. We think that instead of naively implementing this by checking every element, parallelizing matrix multiplication by taking sparsity into account will be important for gaining high speedup. Thus, we are planning to implement CPU and GPU versions of sparse attention by actually checking only the elements that matter, which are decided in advance by user configurations. Based on this implementation, we are planning to analyze how much speedup sparse attention brings compared to naive matrix multiplication.

3 The Challenge

First, implementing sparse attention in CUDA itself is challenging. As aforementioned, we are planning to consider sparsity itself when doing matrix multiplication. This means that we will be looking at elements in certain strides under different configurations. The main aspect that needs to be considered when implementing sparse attention will be locality. Since we will essentially be performing sparse matrix multiplication under a certain rule, exploiting shared memory will become much more complex compared to naive matrix multiplication.

Also, we plan to implement different sparse attention methods and analyze them against the CPU version. We believe that profiling the execution time and the kernel in detail will also be challenging.

Overall, we wish to deepen our understanding of attention and how matrix multiplication can be parallelized, while also learning how to exploit shared memory on CUDA.

4 Resources

4.1 Tutorials

We plan to start by looking into the following sparse attention tutorials in DeepSpeed.

- `https://www.deepspeed.ai/tutorials/sparse-attention/`
- `https://github.com/microsoft/DeepSpeed/tree/master/deepspeed/ops/sparse_attention`

Also, since we are planing to use the C++ front-end of PyTorch to make it compatible with our custom CUDA kernels, we plan to look into the following C++ PyTorch tutorial.

- `https://pytorch.org/tutorials/advanced/cpp_frontend.html`

4.2 Code Base

We plan to implement our own Attention module in C++ using PyTorch C++ front-end APIs. To analyze the bottleneck of attention within the entire transformers layer, we plan to use the Huggingface Transformers library, and run a simple model.

4.3 Papers

We plan to read and refer to the following papers; Vaswani et al. [2017], Child et al. [2019], Beltagy et al. [2020], Zaheer et al. [2020]

5 Goals and Deliverables

5.1 Goals

Plan to Achieve

- Read related papers and work on tutorials to deepen our understanding
- Implement sparse attention on CPU
- Implement sparse attention on GPU using CUDA
- Optimize GPU version of sparse attention by exploiting shared memory
- Analyze the speedup gains of sparse attention against naive attention, and also observe different gains on CPU vs GPU.
- Analyze different input sequence lengths to observe larger gains with longer sequence lengths.

Hope to Achieve

- Implement multiple versions of sparse attention, and compare their performance gains.
- Profile our sparse attention CUDA kernel with Nsight Compute to find room for further optimization.

5.2 Deliverables

At the poster session we will be showing speedup graphs. We plan to plot these graphs to show speedup against the naive matrix multiplication, and also to show different gains between CPU and GPU. We also plan to show different speedup gains depending on different sequence lengths. We hope to verify that sparse attention brings significant speedup compared to the original matrix multiplication.

6 Platform Choice

We plan to implement our own CUDA kernels for sparse attention. Also, we plan to use the C++ front-end of PyTorch to build our deep learning models since it is easy to use and directly compatible with CUDA kernels (so we also do not have to make python - C++ bindings).

7 Schedule

Our planned schedule is as follows:

Time	Plan
11/15	Project Proposal Due
11/19	Work on tutorials, understanding sparse attention
-	Get used to C++ frontend for PyTorch
11/26	Create CPU implementation
12/03	Milestone Report Due
-	Have 1 CUDA implementation & benchmark
12/10	Additional CUDA implementations & optimizations
12/14	Final Report Due
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.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Iz Beltagy, Matthew E Peters, and Arman Cohan. Longformer: The long-document transformer. *arXiv preprint arXiv:2004.05150*, 2020.

Manzil Zaheer, Guru Guruganesh, Kumar Avinava Dubey, Joshua Ainslie, Chris Alberti, Santiago Ontanon, Philip Pham, Anirudh Ravula, Qifan Wang, Li Yang, et al. Big bird: Transformers for longer sequences. *Advances in neural information processing systems*, 33:17283–17297, 2020.