Blocked Dynamic Sparse Attention

Mehul Goel (mehulg), Saransh Agrawal (saransh2)
Website: https://mehulgoel873.github.io/15418-final-project/

Updated Schedule

This is our updated schedule for the remaining weeks of the project.

- **Week 4:** Implement static sparse attention patterns (e.g., local, strided) that prune blocks of the attention matrix based on fixed heuristics. Evaluate accuracy and performance tradeoffs. Mehul: focus on sparse data structures and matmul kernel. Saransh: focus on softmax kernel optimizations and integration with sparse matmul.
- **Week 4.5:** Implement dynamic sparse attention patterns (e.g., top-k) that adaptively prune blocks based on input data. This will involve additional computation to determine which blocks to keep, as well as handling irregular memory access patterns. Mehul: focus on matmul and softmax kernel for dynamic patterns. Saransh: focus on block selection and scheduling strategies.
- **Week 5:** Finish up dynamic sparse attention implementation and test for correctness and performance. Conclude final optimizations and tweaks based on observations from testing. Mehul: focus on final correctness validation and documentation. Saransh: focus on final performance tuning and benchmarking.
- **Week 5.5:** Perform a comprehensive performance analysis comparing all implementations across varying sequence lengths and sparsity levels. Prepare final deliverables and documentation.

Current Progress

We have established a functional baseline and begun the first round of optimizations. First, we implemented a naive CUDA attention kernel that exploits data-level parallelism across the GPU's thread hierarchy, serving as a correctness reference and performance baseline. To validate subsequent optimizations, we developed a testing framework that compares kernel outputs against the naive implementation using numerical tolerance checks.

We then implemented a shared-memory tiled matrix multiplication kernel, which reduces redundant global memory traffic by staging tiles of the input matrices into on-chip shared memory before computing partial products. This optimization yielded approximately a 10% reduction in execution time on small-scale inputs. To systematically characterize performance across problem sizes, we extended our benchmarking infrastructure to sweep over multiple sequence lengths and report averaged latency across repeated trials, enabling more statistically stable comparisons between implementations.

We are currently in the process of brainstorming approaches for optimizing the softmax kernel and managing sparsity in the attention mechanism. We are exploring techniques such as warp-level

primitives for efficient reduction operations in the softmax computation, as well as data structures for tracking non-zero blocks in the sparse attention matrix.

Progress Estimate

The project is on track with respect to its revised scope. Based on feedback received after the proposal phase, we narrowed our focus from reproducing FlashAttention-style fused kernels to concentrating on dynamic sparse attention; specifically, the algorithmic and implementation challenges of adaptively selecting and computing only the most relevant attention blocks at runtime. The benchmarking and correctness infrastructure developed to date will directly support this investigation.

For the poster session, we plan to present performance profiles comparing dense, statically sparse, and dynamically sparse attention kernels across varying sequence lengths and sparsity budgets. If time permits, we will also include visualizations of the attention patterns generated by our dynamic sparsity algorithm, as well as a breakdown of the computational overhead introduced by the block selection process. Further, if time permits, we will present a demo showcasing the dynamic sparse attention kernel in action in a transformer inference setting, potentially using GPT-2 small as a test case.

Concerns

The primary technical risk is that dynamic sparse attention may offer limited wall-clock speedup in practice. The overhead of computing block-level importance scores at runtime and the resulting irregular memory access patterns can offset the savings from skipping low-weight blocks. Because this remains an active research area, state-of-the-art implementations typically couple sparsity with additional optimizations (e.g., specialized CUDA kernels for irregular gather/scatter, hardware-aware block scheduling) that are non-trivial to reproduce independently. We are not certain whether a clean implementation will achieve speedups that clearly justify the added complexity relative to our dense baseline.