15-418 Final Project Milestone Report

gxt minjeon2

1 Summary of Progress

Initially, we implemented a correct sequential implementation of the attention computation to serve as our performance baseline. To validate our results, we wrote a Python script to generate random Q, K, V matrices and compute the attention output using PyTorch. This result served as our correctness checker that would be used to evaluate future iterations. We designed the test suite to consist of realistic problem sizes with embedding dimension $D = 64$ and matrix dimension N ranging from 256 to 8192.

For the parallel version, we implemented the attention mechanism in CUDA. We split up the attention calculation into three kernels corresponding to the following stages in the computation: (1) matrix multiply $QK^\top$ in the numerator and scale by $\sqrt{d_k}$, (2) apply Softmax to the attention score, (3) matrix multiply the attention weights to the value vector V. For steps 1 and 3, we applied tiled matrix multiplication to increase memory reuse and fewer global memory accesses. Our primary optimization focused on the Softmax kernel, for which we implemented two versions. In the initial implementation, for the max/sum reduction step over each row, we assigned a single thread to scan through the row linearly and aggregate the result, which had $O(N)$ time complexity. Then, we implemented our tree-based strategy for the reduction step, in which half of the threads combine pairs of partial results simultaneously. Each time, the stride and number of active threads are halved, so it takes $O(\log N)$ steps to complete the reduction. Furthermore, since stride is always a multiple of 32, all threads in the same warp follow the same conditional branch, so the effects of warp divergence is minimized. Another optimization that we explored was unrolling the last warp in the reduction. At this point, we know that threads in a single warp will execute simultaneously, so we removed the barrier synchronization for the last step in the reduction. Lastly, we verified the correctness with the reference results and recorded the computation time, total execution time, and the Softmax kernel execution time of both implementations. We compared the results between the linear and tree reduction Softmax kernels and produced a speedup graph to examine the performance gain from our proposed strategy.

2 Goals and Deliverables

We believe that we've mostly met our 100% goal that we set in our project proposal. We've implemented a functional CUDA implementation, including tiled matrix multiplication kernels and two versions of Softmax reduction. We have verified the correctness of our implementation against the PyTorch baseline and analyzed the speedup to show the performance gain achieved by our tree-based approach on a single batch.

We believe we will be able to complete all of our goals with the remaining time. While implementing and testing our CUDA implementation, we noticed that the attention computation was heavily memory-bound. Thus, our steps will be to parallelize the computation across multiple heads or batches and pipeline the stages to hide the latency of memory operations and transferring between devices. Our goals for the remaining time are as follows:

1. Perform an in-depth performance analysis of our implementation to verify the bottlenecks in the program.
2. Measure the performance of our program on the PSC machines and produce performance scaling and speedup graphs.
3. Implement asynchronous pipelining to process multiple attention heads or batches concurrently.
4. If we have time, we will implement the attention mechanism using MPI and produce a speedup graph comparing the MPI and CUDA implementations.

3 Preliminary Results

N	**Linear (ms)**	**Tree (ms)**	**Speedup**
256	0.135	0.027	5.07×
512	0.156	0.044	3.56×
1024	0.300	0.088	3.40×
2048	0.619	0.205	3.02×
4096	1.456	0.641	2.27×
8192	3.719	2.071	1.80×

Table 1: Softmax execution times using Linear Reduction vs. Tree Reduction.

N	Comp. (Linear)	Comp. (Tree)	Total (Linear)	Total (Tree)
256	0.229	0.174	68.646	81.173
512	0.429	0.316	55.967	55.430
1024	1.158	1.109	59.466	61.391
2048	3.838	3.419	67.559	71.206
4096	14.000	13.163	72.863	77.665
8192	53.384	51.730	117.613	109.000

Table 2: Computation and Total execution times for Linear vs. Tree Reduction.

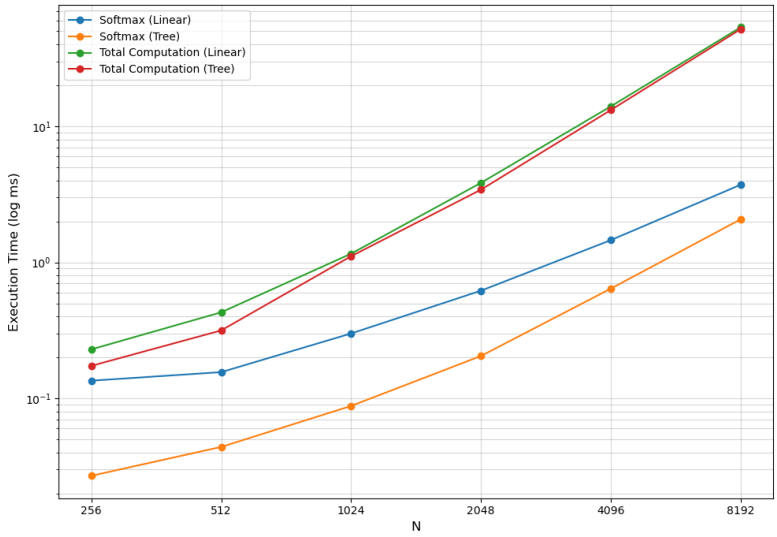

Figure 1: Performance scaling of the Linear vs. Tree Reduction implementations across matrix dimensions N.

The above plot shows the execution time scaling of our attention mechanism on a log scale across increasing matrix dimensions. The Softmax execution time shows that our tree-based reduction consistently outperforms the linear scan across all N. As N increases, the speedup of the tree-based approach decreases, suggesting that the Softmax kernel becomes limited by memory bandwidth. However, the convergence of the total computation lines at larger problem sizes indicates that the overall performance is increasingly dominated by the $O(N^2D)$ matrix multiplications, which we hope to address using pipelining.

4 Concerns

We do not currently face any major blockers, so our remaining work will mostly be profiling our code using tools like ncu to more precisely identify our bottlenecks and implementing the optimizations we discussed.

5 Timeline

1. **November 1 – December 3**
    - **Sunny:** Design and run experiments on the PSC machines to observe any performance differences.
    - **Grace:** Research tools (e.g. NVIDIA Nsight Compute) to profile the kernels to confirm main bottlenecks of our implementation.
2. **December 4 – December 6**
    - **Grace:** Implement pipelining to support concurrent batch and multi-head processing.
    - **Sunny:** Run final experiments and profile the optimized kernels.
    - **Both:** Begin drafting the final report and recording the results.
3. **December 7 – December 8**
    - **Sunny:** Write the "Summary" and "Background" sections of the report.
    - **Grace:** Write the "Results" section.
    - **Both:** Work on the "Approach" section.