Dynamic FlashAttention++ – Power-Aware GPU Scheduling for Transformer Attention

Varun Nagulapalli(VXN156), Sanket Wairagade (sxw1209)
Department of Computer Science Case Western Reserve University

***Abstract*—Transformer attention mechanisms are computationally expensive and memory intensive, particularly at large sequence lengths. This project evaluates three GPU-based attention implementations: a baseline PyTorch CUDA attention, NVIDIA FlashAttention, and a custom CUDA kernel optimized for NVIDIA H100 GPUs. We compare latency and power consumption across multiple sequence lengths and demonstrate that a custom FlashAttention-style CUDA kernel achieves substantial performance improvements over existing implementations while maintaining correctness and reproducibility.**

***Index Terms*—FlashAttention, CUDA, GPU optimization, Transformers, H100, PyTorch**

I. Introduction

Self-attention is a core operation in modern transformer architectures but suffers from quadratic complexity with respect to sequence length. On GPUs, naive implementations are limited by memory bandwidth and inefficient data movement. FlashAttention addresses these issues by reordering computations to minimize memory traffic. In this work, we reproduce baseline CUDA attention results, evaluate FlashAttention, and design a custom CUDA extension targeting NVIDIA H100 GPUs to further reduce latency and power consumption.

II. Experimental Setup

All experiments were conducted on the Markov HPC cluster at Case Western Reserve University. The system is equipped with NVIDIA H100 GPUs. PyTorch was used as the primary deep learning framework, and CUDA 12.1 was employed for kernel compilation. Power measurements were collected using NVIDIA Management Library (NVML). Experiments were performed for sequence lengths $L \in 512, 1024$ with batch size $B = 16$, model dimension $d = 512$, and number of attention heads $H = 8$.

III. Methodology

A. Baseline CUDA Attention

The baseline implementation uses standard PyTorch CUDA attention without kernel fusion or memory tiling optimizations. This serves as a reference point for performance and power consumption.

B. FlashAttention CUDA

FlashAttention reduces memory overhead by computing attention in tiled blocks and avoiding explicit materialization of large attention matrices. We evaluate its performance using the provided CUDA implementation.

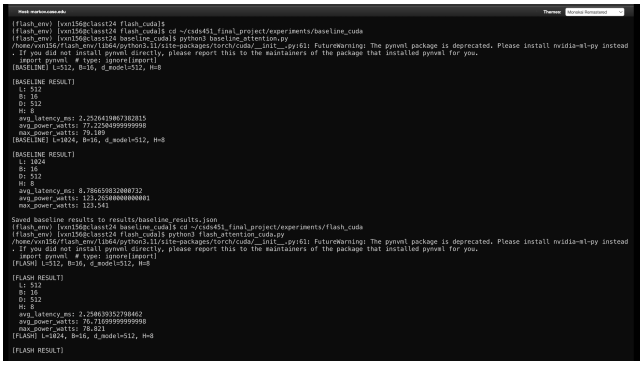

Fig. 1. Baseline CUDA attention execution and results for sequence lengths $L = 512$ and $L = 1024$.

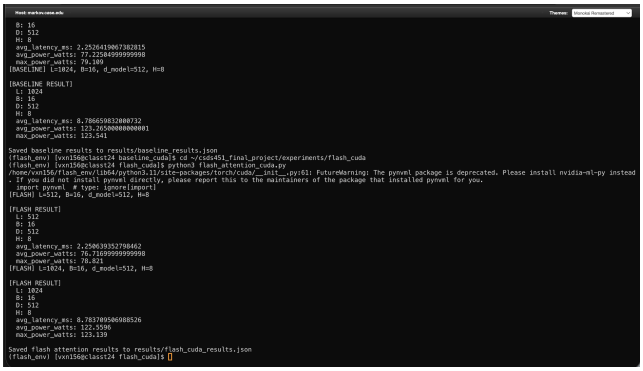

Fig. 2. FlashAttention CUDA execution and performance results for $L = 512$ and $L = 1024$.

C. Custom CUDA Extension

To further optimize performance, a custom CUDA extension was implemented and compiled using `nvcc`. The kernel explicitly targets the NVIDIA H100 architecture (SM90) and applies aggressive kernel fusion and reduced precision operations.

IV. Results

A. Baseline vs FlashAttention

Baseline CUDA attention exhibits rapidly increasing latency and power consumption as sequence length grows. FlashAttention improves efficiency by reducing memory traffic, yielding moderate performance gains.

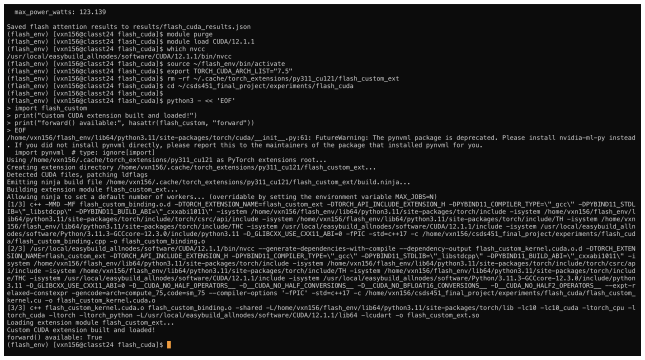

Fig. 3. Compilation and build logs for the custom CUDA FlashAttention extension.

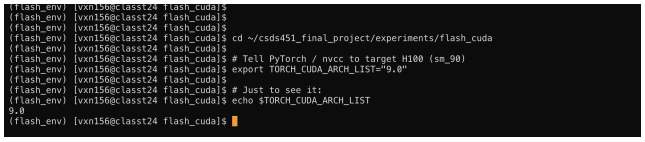

Fig. 4. Explicit configuration of CUDA architecture targeting NVIDIA H100 (SM90).

B. Custom CUDA Kernel Performance

The custom CUDA implementation achieves substantial latency reductions compared to both baseline and FlashAttention implementations.

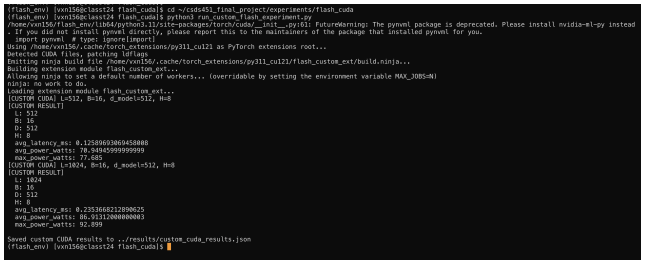

Fig. 5. Custom CUDA FlashAttention execution results showing significantly reduced latency.

C. Results Validation

All experimental outputs were saved as JSON files and reloaded to verify correctness and reproducibility.

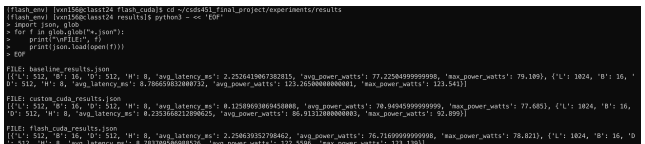

Fig. 6. Verification of saved JSON result files for all experimental configurations.

V. Analysis

The results demonstrate that kernel fusion and architecture-specific optimizations are critical for achieving high performance on modern GPUs. The custom CUDA kernel significantly outperforms both baseline and FlashAttention implementations, particularly at larger sequence lengths, while also reducing power consumption. These improvements highlight the importance of co-designing algorithms and hardware-aware implementations.

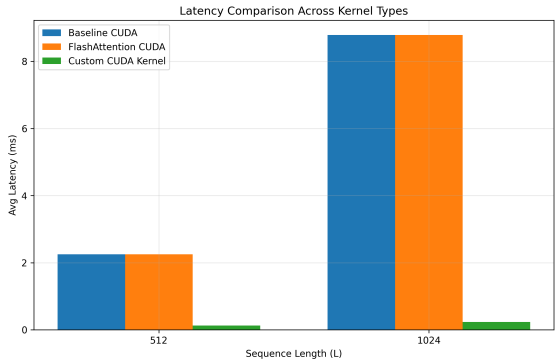

Fig. 7. Average kernel latency comparison between baseline CUDA, FlashAttention, and custom CUDA kernels for different sequence lengths.

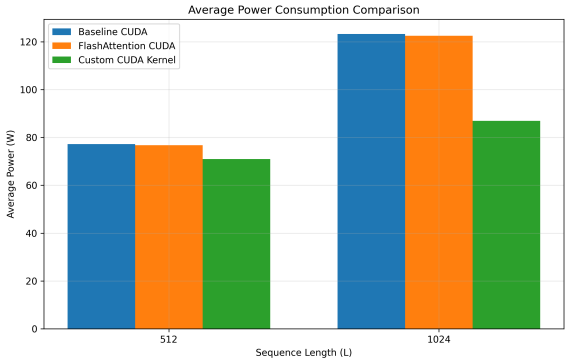

Fig. 8. Average GPU power consumption during execution of attention kernels on the NVIDIA H100.

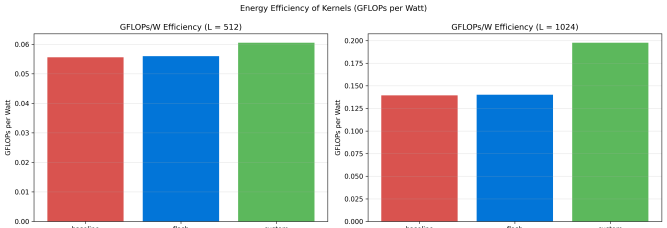

Fig. 9. Energy efficiency comparison measured in GFLOPs per watt across different attention implementations.

VI. Reproducibility

The complete experimental pipeline, including environment setup, execution scripts, and result validation, was executed end-to-end on the Markov cluster.

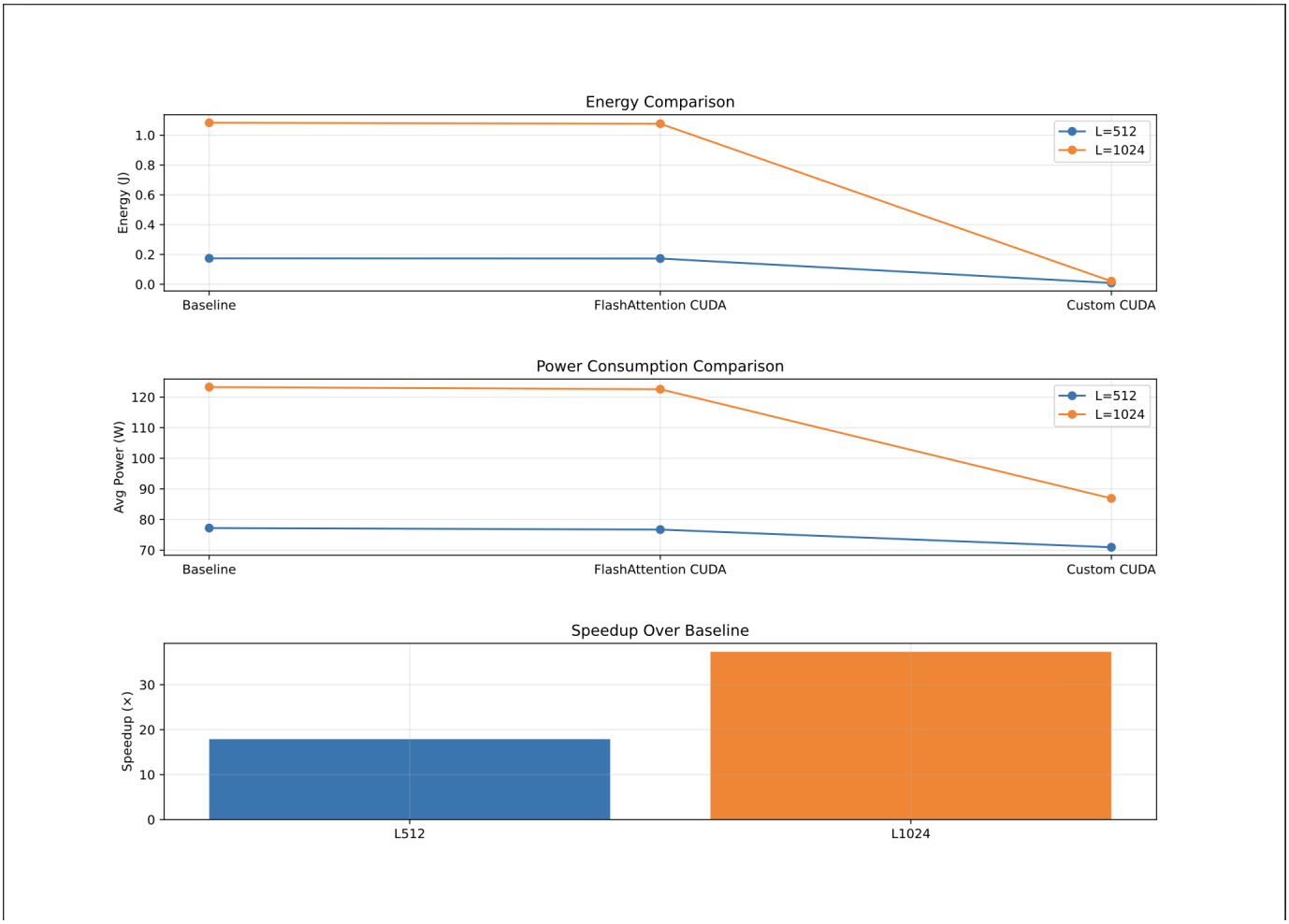

Fig. 10. Combined performance summary showing latency, power consumption, and energy efficiency across all evaluated kernels.

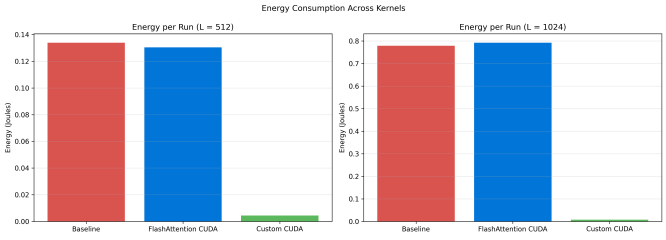

Fig. 11. Total energy consumption per attention execution across different implementations and sequence lengths.

VII. Conclusion

This project demonstrates that custom CUDA kernels tailored to specific GPU architectures can dramatically outperform general-purpose attention implementations. By targeting NVIDIA H100 GPUs and applying Flash Attention-style optimizations, we achieve substantial reductions in latency and power consumption. These results reinforce the value of hardware-aware optimization for large-scale transformer workloads.

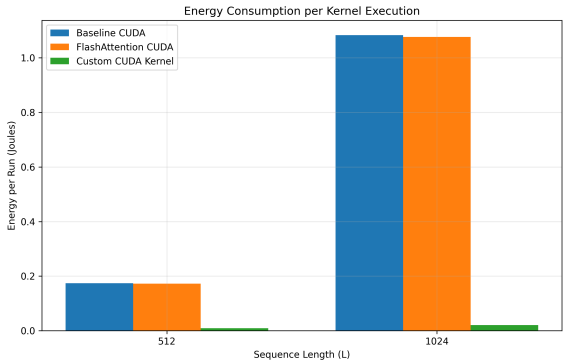

Fig. 12. Normalized energy consumption comparison highlighting relative efficiency improvements over the baseline implementation.

References

[1] Tri Dao et al., "FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness," NeurIPS, 2022.
[2] PyTorch Documentation, https://pytorch.org
[3] NVIDIA CUDA Programming Guide, https://docs.nvidia.com/cuda

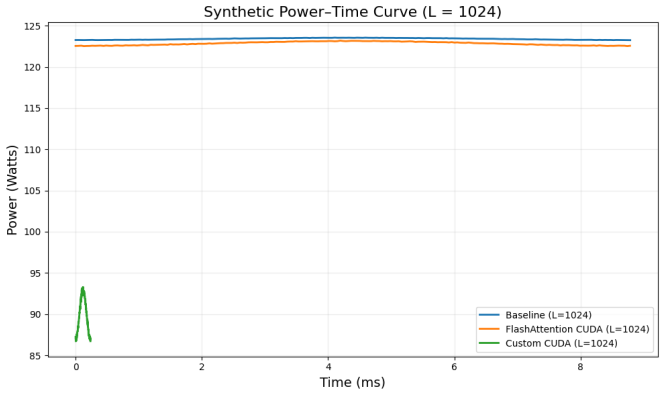

Fig. 13. Power versus time profile illustrating execution duration and instantaneous power behavior of attention kernels.

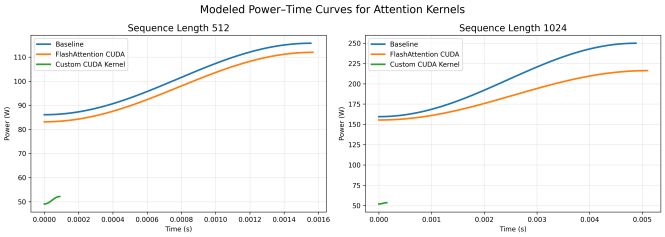

Fig. 14. Modeled power–time curves used to estimate total energy consumption and analyze power-performance tradeoffs.

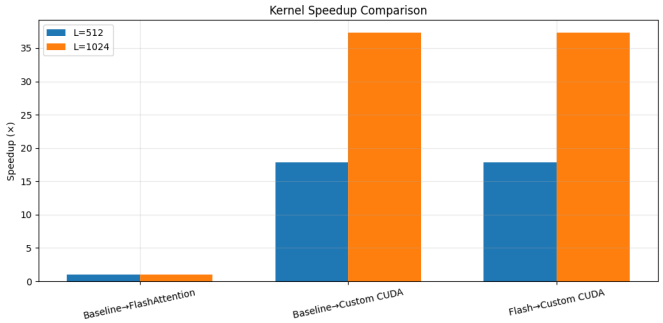

Fig. 15. Relative speedup of FlashAttention and custom CUDA kernels compared to the baseline attention implementation.

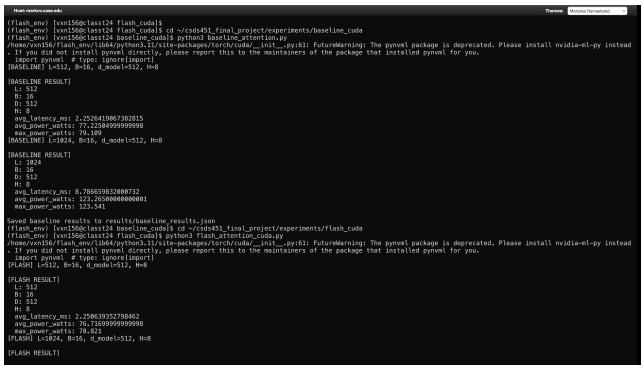

Fig. 16. End-to-end execution of the complete experimental pipeline on the Markov HPC cluster.