Batched Speculative Decoding Verification Kernels

Steven Kolawole

URL: https://stevenkolawole.github.io/15618/

Abstract

We implement and optimize batched speculative decoding verification kernels in CUDA, targeting the parallelism challenges that arise when verifying draft tokens across multiple sequences simultaneously—specifically warp divergence, non-coalesced KV cache writes, and dynamic load imbalance caused by sequences accepting different numbers of draft tokens at runtime. We profile these bottlenecks with Nsight Compute, implement optimizations using warp-level primitives and prefix sums, and evaluate performance across batch sizes and acceptance rates on GPU.

Background

Speculative decoding [1] accelerates autoregressive LLM inference by using a small draft model to propose γ tokens speculatively, then verifying all $\gamma + 1$ positions in a single parallel forward pass of the larger target model. This verification pass looks structurally similar to a prefill operation and is compute-bound rather than memory-bandwidth-bound, which is why it yields speedups over serial decoding.

However, there is a pitfall in research publications: most tend to assume a batch size of 1. In production, where batching multiple sequences simultaneously is expected, verification introduces a hard parallelism challenge: different sequences in the batch accept different numbers of draft tokens at runtime. Sequence A may accept 4 out of 5 draft tokens while sequence B accepts only 1. This produces *ragged outputs* since the accepted suffix length k_i per sequence is unknown until the verification pass is completed, creating irregular work distribution, non-coalesced KV cache updates, and warp divergence during the token acceptance scan.

The Challenge

The core parallelism difficulties are:

- **Divergent control flow during acceptance scanning.** Since each sequence independently walks its draft tokens and rejects at the first mismatch, threads handling different sequences diverge in the number of iterations, causing serialization within warps.
- **Irregular KV cache updates.** After verification, each sequence must update its KV cache by exactly k_i accepted tokens, where k_i varies per sequence. Writes are non-contiguous across sequences, degrading memory coalescing.
- **Dynamic work imbalance.** Sequences with low acceptance rates finish early, leaving execution resources idle while others are still scanning.

Resources

Hardware: MLD Babel Cluster (via Prof. Virginia Smith's lab access); PSC Bridges-2 as backup.

Starting point: We begin from the LLMSpeculativeSampling codebase, which implements single-sequence ($b = 1$) speculative decoding. We extend this to the batched setting and replace the verification step with custom CUDA kernels.

Key references:

- Leviathan et al. (2023) — original speculative decoding algorithm [1]
- Speculative Decoding and Beyond: An In-Depth Survey [3]
- SEED: Scheduled Speculative Decoding [2]
- BASS: Batched Attention-optimized Speculative Sampling [4]
- Batch Speculative Decoding Done Right [5]
- MagicDec [6]

Goals and Deliverables

Plan to Achieve

- PyTorch baseline implementation (correctness reference and performance floor)
- Naive CUDA verification kernel with per-sequence thread assignment
- Nsight Compute profiling characterization: warp execution efficiency, memory throughput, occupancy
- Roofline analysis across $b \in \{1, 4, 16, 32\}$ and $\alpha \in \{0.3, 0.6, 0.9\}$
- Opt A: Ballot-based acceptance scan using `__ballot_sync` and `__builtin_ctz`
- Opt B: Coalesced KV cache writes via prefix sum over accepted lengths
- Speedup curves vs. PyTorch baseline across all batch sizes

Hope to Achieve

- Opt C (stretch): Sorted batching by predicted acceptance rate to reduce intra-warp divergence
- Integration with real HuggingFace models (GPT-2 / GPT-2-XL) and real datasets

If Work Goes Slowly

- Synthetic inputs only; Opt A implementation; Nsight characterization as primary contribution

Poster Session

We plan to show: (1) Nsight profiling diff between naive and optimized kernels, (2) roofline plots across configurations, and (3) speedup curves (ablation table).

Platform Choice

We target NVIDIA GPUs on the Babel cluster using CUDA C++. This platform is the natural choice because the phenomena we are studying—warp divergence, memory coalescing, and occupancy—are GPU-specific architectural effects. CUDA gives us direct access to the warp-level primitives (`__ballot_sync`, `__shfl_up_sync`) required for Opt A and Opt B, and Nsight Compute provides the hardware performance counters needed to precisely attribute bottlenecks. A CPU implementation would not exhibit the same parallelism challenges and would not let us study the relevant phenomena.

Related course lectures: Lecture 5 (Work Distribution and Scheduling), Lecture 7 (GPU Architectures), Lecture 11 (Performance Measurement and Tuning).

Schedule

Dates	Tasks
Mar 25–28	PyTorch baseline, HuggingFace decoding loop, synthetic input harness
Mar 29 – Apr 4	Naive CUDA kernel: compiling, correct, callable from Python
Apr 5–11	Nsight profiling of naive kernel; roofline analysis
Apr 12–14	Opt A implementation; Milestone report
Apr 15–21	Opt B implementation; ablation experiments
Apr 22–24	Full sweep; poster preparation; Opt C if on track
Apr 25–30	Final report

References

[1] Leviathan, Y., Kalman, M., & Matias, Y. (2023). *Fast Inference from Transformers via Speculative Decoding*. ICML 2023.

[2] SEED: Accelerating Reasoning Tree Construction via Scheduled Speculative Decoding. COLING 2025.

[3] Speculative Decoding and Beyond: An In-Depth Survey of Techniques. arXiv:2502.19732, 2025.

[4] BASS: Batched Attention-optimized Speculative Sampling. ACL Findings 2024.

[5] Batch Speculative Decoding Done Right. arXiv:2510.22876, 2025.

[6] MagicDec: Breaking the Latency-Throughput Tradeoff for Long Context Generation with Speculative Decoding. arXiv:2408.11049, 2024.