[Milestone Report] Speculative Speculative Decoding: Hiding Draft Latency Through Asynchronous Speculation on Multi-GPU Systems

Marcus Alenius **William Chien**
Carnegie Mellon University
{malenius, wjchien}@andrew.cmu.edu

URL: https://alenius.io/15418-project

1 Work Completed

We have implemented the core infrastructure and two of the three decoding modes in our benchmark framework. We built a GEMM-only compute graph that captures the structure of a transformer decoder layer. Each layer executes seven FP16 matrix multiplications: four attention projections and three FFN projections. We created an autoregressive (AR) decode baseline benchmark that executes N sequential forward passes with batch size $M{=}1$ to simulate token-by-token generation. We also implemented tensor parallelism using NCCL.

Next, we implemented the full speculative decoding (SD) benchmark pipeline, where each SD round consists of K sequential draft model forward passes ($M{=}1$), followed by one batched target model verification pass ($M{=}K$), followed by synthetic Bernoulli acceptance sampling at rate α. In the tensor parallel case, the SD benchmark uses the tensor parallel forward pass for the target model verification, while the draft model runs on a single GPU. Finally, we created a CLI program and infrastructure that help us run tests and experiments.

2 Progress on Goal and Deliverables

All Week 1 deliverables are complete: single-GPU compute graph, PSC environment setup, NCCL microbenchmarks, AR decode baseline, and T_{SD} calibration on V100. For Week 2, tensor parallelism and the single-GPU SD pipeline are complete. We are almost done validating multi-GPU SD with tensor parallelism on the PSC machines, and this will be done by early Week 3.

Currently, we are on track to finish all "plan to achieve" deliverables. The SD infrastructure is complete, and SSD builds directly on the same primitives (forward passes, NCCL communication, threading) with additional coordination logic for concurrent draft/target execution. The "hope to achieve" items, like batch size scaling analysis, heterogeneity experiments, and a formal analytical model, will depend on how smoothly SSD integration and debugging proceeds in the second half of Week 3.

3 Poster Session

We plan on showing (a subset of) the following graphs during the poster session:

1. **AR vs. SD vs. SSD throughput.** A graph showing tokens/sec (or speedup over AR) across different α and K, with all three methods side-by-side.
2. **Cache hit rate sweep**. Holding the other parameters fixed, this will show how SSD's advantage over SD changes as p_{hit} varies.
3. **Fan-out sweep.** Captures the tradeoff between pre-speculation work and cache hit rate.
4. **Per-round latency breakdown.** A stacked bar chart with a bar for each method showing the wall-clock time of a single round in ms, broken into components. For AR it's just one T_{target} bar. For SD it's T_{draft} + T_{target} stacked sequentially. For SSD it's $\max(T_{\text{target}}, T_{\text{pre-spec}})$ + NCCL sync overhead.
5. **Verification window characterization.** Answers how many draft passes fit inside one target verification.
6. **Roofline-style analysis.** Plotting measured performance against the hardware roofline (using NCCL and cuBLAS microbenchmark numbers).
7. **Batch size scaling.** SSD speedup over SD as batch size increases, showing the degradation as p_{hit}^{b} shrinks.

8. **Throughput-latency Pareto frontier.** Similar to Figure 7 (right) in the SSD paper, plotting total throughput (tok/s) on the y-axis against per-sequence latency (1/latency) on the x-axis, with points for AR, SD, and SSD at different batch sizes.

4 Preliminary Results

4.1 T_{SD} on V100

As derived in the proposal, the theoretical speedup of SD over AR is

$$\text{speedup}_{\text{AR}\to\text{SD}} = \frac{E_{\text{SD}}}{1 + T_{\text{SD}}},$$

where E_{SD} is the expected number of generated tokens and $T_{\text{SD}} = T_{\text{draft}}/T_{\text{target}}$ is the ratio of time taken by the drafting and verification phases.

E_{SD} is calculated analytically as $E_{\text{SD}} = (1 - \alpha^{K+1})/(1 - \alpha)$ whereas T_{SD} must be measured on the given hardware. Table 1 shows measured T_{SD} on a V100-32 GPU.

Table 1: T_{SD} for different K values and two target/draft model pairs on one V100-32 GPU.

K	Llama-8B/1B	Llama-70B/1B*
2	0.36	0.04
4	0.69	0.08
6	1.02	0.12
8	1.35	0.15
10	1.67	0.19
12	2.01	0.23

* T_{target} for Llama-70B was estimated by running only one layer as the full model does not fit on the V100-32.

4.2 Speedup: SD over AR

Given the measured T_{SD}, we can compute the theoretical speedup of SD over AR. Table 2 shows both the theoretical speedup and the achieved speedup for $\alpha = 0.9$.

We have yet to run these experiments with multiple GPUs and tensor parallelism, which we need to test the Llama-70B/1B model pair. Since this model pair has smaller T_{SD}, we would expect a greater speedup.

5 Concerns

PSC Multi-GPU Allocation Time. A practical concern is the time required to obtain interactive GPU allocations on PSC Bridges-2. We have found that requesting a single GPU through the `GPU-shared` partition is nearly instantaneous, while requesting two GPUs introduces a modest wait. Requesting four GPUs—the maximum allowed in `GPU-shared`—can take considerably longer, and we have not yet been able to obtain an interactive allocation of eight GPUs, which requires the full `GPU` partition. Since our SSD implementation requires at minimum $T + 1$ GPUs for T-way tensor parallelism plus a dedicated draft GPU, this poses a real bottleneck for development and evaluation. Going forward, we plan to submit batch jobs rather than relying on interactive sessions, as batch jobs can be scheduled to run when resources become available—including during off-peak hours—without requiring us to wait at a terminal. We will use interactive sessions with fewer GPUs for development and debugging, and reserve batch submissions for full-scale evaluation runs.

Table 2: Theoretical and achieved throughput speedup for different K values with the Llama-8B/1B target/draft model pair on one V100-32 GPU.

K	Theoretical	Achieved
2	2.12×	2.19×
4	2.57×	2.64×
6	2.73×	2.81×
8	2.75×	2.87×
10	2.69×	2.75×
12	2.60×	2.70×

Schedule. The SD pipeline was completed in early Week 3, slightly later than originally planned. The additional time was spent building a modular infrastructure—shared forward pass primitives, configurable model definitions, and reusable benchmarking harnesses—that both the SD and SSD pipelines build on. With this foundation in place, the SSD implementation can focus purely on the asynchronous coordination logic (draft-target overlap, cache management, NCCL send/recv) rather than reimplementing compute and timing infrastructure from scratch. This should put us back on track.

6 Updated Schedule

Week 1 (3/29–4/4)

- **[Complete]** Implement single-GPU cuBLAS GEMM sequence for one transformer layer

with correct dimensions for both draft and target.
- **[Complete]** Set up PSC environment, verify NVLink topology and run NCCL microbenchmarks (all-reduce, send/recv at various message sizes).
- **[Complete]** Stack L layers into a full forward pass, measure single-GPU AR decode throughput.
- **[Complete]** Measure T_{SD} for target model dimensions on V100 to calibrate performance targets.
- **[Complete]** Compare microbenchmark results to V100 roofline and document baseline numbers.

Week 2 (4/5–4/11)
- **[Complete]** Implement tensor-parallel target model (column/row-wise weight sharding, NCCL all-reduce after each layer).
- **[Complete]** Implement single-GPU SD pipeline (draft loop, batched verification, synthetic acceptance).
- Integrate tensor-parallel target with SD pipeline for multi-GPU SD. Validate SD speedups against performance targets.

Week 3 (4/12–4/18)—Milestone Report Due 4/14

4/12–4/15
- **[Complete]** Write milestone report.

4/16–4/18
- Implement target GPU loop for SSD (launch verification, determine outcome, NCCL send/recv with draft GPU).
- Implement draft GPU loop (pre-speculation with fan-out, cache storage and lookup).
- Implement communication protocol between draft and target (outcome transmission, speculation return, cache hit/miss branching).
- Integration testing: verify that SSD matches SD output under $p_{\text{hit}} = 0$ (pure fallback) as a correctness check.

Week 4 (4/19–4/25)

4/19–4/22
- Conduct throughput and latency sweeps over F, α, K, and p_{hit}.
- Perform verification window characterization (draft-to-target time ratio, maximum fan-out budget).

4/23–4/25
- Conduct roofline-style analysis using microbenchmark data.
- Perform batch size scaling experiments (hope to achieve).
- Debug and re-run any configurations with anomalous results.

Week 5 (4/26–5/2)—Final Report Due 4/30, Poster Session 5/1
- Generate final figures.
- Write final report.
- Prepare poster presentation.