Batched GPU Execution for High-Throughput Combinatorial Search

Minh Phuc Nguyen
nguyenp@union.edu
Union College
Department of Computer Science
Schenectady, New York, USA

Matthew Anderson (advisor)
andersm2@union.edu
Union College
Department of Computer Science
Schenectady, New York, USA

Abstract

Simplifiable Strong Uniquely Solvable Puzzles (SUSPs) connect combinatorial search to upper bounds on the matrix-multiplication exponent [1]. The process of evaluating each candidate requires expensive parallel workloads, which have been handled efficiently using GPUs. However, the existing search sent one candidate at a time to a GPU, so overhead limited the benefits of fast evaluation on a GPU. We redesigned the search by batching same-size candidates into one kernel launch and decoupling GPU work from multi-threaded search expansion on the CPU. Our pipeline increased end-to-end throughput by up to 5.75× with one GPU and reached over 11× with two. With a better search program, we found a $(79, 10)$-SUSP, improving on Anderson and Le's best width-10 result [1].

CCS Concepts

• **Computer systems organization** → **Heterogeneous (hybrid) systems**; • **Computing methodologies** → *Massively parallel algorithms.*

Keywords

GPU computing, CUDA, combinatorial search, heterogeneous computing, strong uniquely solvable puzzles, matrix multiplication

ACM Reference Format:
Minh Phuc Nguyen and Matthew Anderson (advisor). 2026. Batched GPU Execution for High-Throughput Combinatorial Search. In *ACM Student Research Competition at SC '26 (SC '26 SRC), November 17–19, 2026, Chicago, Illinois, USA.* ACM, New York, NY, USA, 2 pages.

1 Problem and Motivation

The exponent ω describes how matrix-multiplication cost grows. In the Cohn–Umans framework, a SUSP with a better size-to-width ratio can tighten an upper bound on ω [2]. An (s, k)-puzzle is a 2D array with s distinct rows of width k, and every cell is chosen from $\{1, 2, 3\}$. Only a tiny fraction are SUSP, while the search space grows as 3^{sk}. Anderson and Le's work focused on Simplifiable SUSPs, which can be verified to be SUSPs in polynomial time, while general SUSPs cannot yet. They used an iterated local search whose main scoring routine has a worst-case cost of $O(s^{5.5})$, and long runs call it billions of times. The systems problem is how to increase search throughput without changing the exact evaluation results [3].

SC '26 SRC, Chicago, Illinois, USA
2026.

2 Background and Bottleneck

Although candidate evaluation is highly parallel, per-candidate GPU offloading brought little improvement to full-search throughput. Kernel launch overhead, repeated memory allocation, host-device synchronization, and small grids left the GPU mostly idle while CPU workers responsible for search-frontier expansion waited for the GPU. These limitations motivated us to design a system that overlaps CPU-thread work with GPU computation [3].

3 Approach and Uniqueness

Firstly, we package candidate evaluations into one large batch per kernel launch. This amortizes overhead costs across many candidates, reducing the average overhead per candidate. To achieve this, we serialized each raw candidate input into a one-dimensional array and placed these arrays contiguously in a pinned host buffer. This buffer is reused, so it is allocated only once.

Based on Anderson and Le's original work, we use OpenMP with multithreaded search-expansion workers. When CPU threads independently expand the search and launch kernels for GPU evaluation, each worker must wait until the GPU finishes the evaluation and returns the results. When multiple threads launch work simultaneously, their tasks queue on the GPU, causing many CPU threads to wait at the same time. To address this problem, we designed a pipeline that decouples CPU search expansion from GPU evaluation through a shared queue [3]. A single manager thread is responsible only for launching kernels and collecting results. After enqueueing a task, each worker begins preparing the next batch. It later processes the previous batch once the GPU has evaluated it and the manager thread has returned its results. Each worker has two buffer slots, allowing these operations to overlap. This workflow also allows the search program to scale to multiple GPUs. Adding more GPUs gives the CPU manager additional devices on which to launch kernels. The manager dispatches successive batches to the GPUs in round-robin order.

4 Results and Contribution

Our experiments used a 24-thread AMD Ryzen 9 9900X and two NVIDIA RTX 5070 Ti GPUs with a batch size of 256. We tested various puzzle sizes, from small widths (5, 6, ...) to large widths (14, 15), and many row configurations. The returned result was consistent with the baseline in all tests.

We also measured throughput at the sizes of the largest Simplifiable SUSPs of widths 6 through 9 published by Anderson and Le. The single-GPU search achieved end-to-end speedups ranging from 3.09× to 5.75×, while the double-GPU search achieved more than 11×.

Through separate profiling measurements, we found that average streaming multiprocessor (SM) activity tended to increase with puzzle size. Larger puzzles require more computation, suggesting that the GPU is better utilized when more work is available.

We also tested (52, 9), the largest size in the benchmark group, on Modal with 80 OpenMP threads and up to 4 L40S GPUs to see how far we could increase search throughput. Throughput and CPU core usage increased sharply from 1 to 2 GPUs, with diminishing returns beyond 2 GPUs.

We have not yet been able to use Nsight Systems or Nsight Compute in this environment, so the cause remains unclear. We plan to measure how long CPU workers wait for results and GPUs wait for work to identify what limits further scaling.

Acknowledgments

OpenAI's ChatGPT was used to generate and revise wording in the Abstract and Sections 1–4 [3]. The authors reviewed the text and take full responsibility for its content.

References

[1] Matthew Anderson and Vu Le. 2023. Efficiently-Verifiable Strong Uniquely Solvable Puzzles and Matrix Multiplication. In *Computing and Combinatorics: 29th International Conference (Lecture Notes in Computer Science, Vol. 14423)*. Springer, Cham, 41–54. doi:10.1007/978-3-031-49190-0_4

[2] Henry Cohn, Robert Kleinberg, Balázs Szegedy, and Christopher Umans. 2005. Group-Theoretic Algorithms for Matrix Multiplication. In *46th Annual IEEE Symposium on Foundations of Computer Science*. IEEE, Pittsburgh, 379–388. doi:10.1109/SFCS.2005.39

[3] OpenAI. 2026. ChatGPT. https://chatgpt.com/ Used for writing assistance; accessed August 2, 2026.