MEMORANDUM

TO:	Professor Majid Shaalan, Ph.D.
FROM:	Kenneth Peter Fernandes
PROGRAM:	Ph.D. in Computational Sciences
DATE:	February 23, 2026
SUBJECT:	IMC05 Reflection – Parallel Sieve of Eratosthenes
COURSE:	CISC 719: Contemporary Computing Systems Modeling Algorithms (CCSM), Spring 2026
UNIVERSITY:	Harrisburg University of Science and Technology

1. What Was Easy vs. Hard in Adapting the Classical Sieve

What Came Naturally. The conceptual leap from sequential to parallel sieving was surprisingly intuitive. The Sieve of Eratosthenes is an embarrassingly parallel algorithm at its core: once the base primes up to $\sqrt{N}$ are known, every segment of the number line can be processed independently. Dividing the range $[2, N]$ into contiguous blocks — one per OpenMP thread or CUDA thread block — required no synchronization on the marking pass, which made the parallelization feel like a direct, almost mechanical translation. Similarly, adapting the sieve to identify *Sophie Germain primes* (p such that $2p + 1$ is also prime) and *safe primes* ($q = 2p + 1$ where p is prime) was straightforward once a complete primality bitset was in hand: the scan reduces to a single pass over the prime list checking membership of $2p + 1$ in the set, a problem perfectly suited to a parallel `for` loop.

Where Complexity Emerged. The hard parts were almost entirely about memory. Implementing segmented bit-packing simultaneously — one bit per odd number, processed in cache-sized chunks — introduced non-trivial index arithmetic. Mapping a global index into a segment-local bit position requires careful offsetting, and off-by-one errors at segment boundaries produced silent correctness failures that took time to diagnose. The GPU implementation added another layer of difficulty: Numba CUDA kernels require explicit reasoning about thread-to-data mapping, global memory coalescing, and the cost of host-to-device transfers. For small N, PCIe transfer overhead dominated runtime, making the GPU appear slower than the serial baseline — a counterintuitive result that required careful explanation in the benchmarking analysis.

The Cryptographic Extension Specifically. Adapting the sieve to cryptographic primality analysis exposed a fundamental algorithmic boundary. The sieve excels within a bounded range but is wholly impractical for the integer sizes used in modern RSA (1024–4096 bits). A 1024-bit number has on the order of 10^{308} digits; no sieve can enumerate primes in that space. Bridging this gap required understanding where sieve-based methods are genuinely useful — for example, as a *trial division prefilter* to rapidly eliminate small-factor composites before invoking Miller–Rabin for large candidate primes — versus where probabilistic primality tests are the only viable option. This boundary between exact bounded algorithms and

probabilistic unbounded ones is not obvious from the classical sieve formulation and required careful study.

2. How the PCAM Model Helps and Hinders Creative Thinking

Where PCAM Provided Clarity. The PCAM framework (Partitioning, Communication, Agglomeration, Mapping) proved most valuable during the design phase, before any code was written. For the segmented sieve, the Partitioning step made it immediately clear that the work decomposed along segment boundaries — a natural fit for both shared-memory (OpenMP) and GPU (CUDA block) models. The Communication analysis revealed that the *only* shared data is the list of base primes up to $\sqrt{N}$, which is read-only and small (at most $O(\sqrt{N})$ entries). This insight eliminated the need for complex synchronization primitives and guided the decision to broadcast base primes once before the parallel marking pass. PCAM essentially converted a vague instruction ("parallelize the sieve") into a concrete set of design decisions with clear rationale.

Where PCAM Felt Constraining. The model struggles when the problem does not decompose cleanly into four sequential phases. The cryptographic extension — scanning for Sophie Germain and safe primes, and reasoning about RSA factorizability — has a fundamentally different structure: the "communication" between finding a prime p and checking whether $2p + 1$ is prime is not data movement in the parallel sense but rather a lookup into a shared data structure. Forcing this into the PCAM vocabulary felt artificial. More broadly, PCAM offers limited guidance on *memory hierarchy* effects. For $N = 10^8$, the choice of segment size has a larger impact on performance than the number of threads, yet PCAM's Agglomeration phase addresses this only obliquely. A model that explicitly incorporates cache levels, NUMA topology, and GPU memory hierarchy would complement PCAM well in practice.

3. What I Would Do Differently with a Distributed GPU Cluster

Architecture and Algorithm Redesign. With access to a distributed GPU cluster — say, 8 nodes each equipped with 4× NVIDIA A100 80GB GPUs — the most impactful change would be a full hybrid MPI+CUDA design. Each MPI rank would own a non-overlapping subrange of $[2, N]$, with Rank 0 computing base primes up to $\sqrt{N}$ and broadcasting them via `MPI_Bcast`. Within each node, a CUDA kernel would execute the segmented sieve across the rank's local range, with each thread block processing one cache-aligned segment. The A100's 80 GB of HBM2e memory would allow segment sizes of several hundred megabytes, dramatically reducing the number of kernel launch cycles compared to the Colab T4 GPU used in this project. For $N = 10^{12}$, this architecture would distribute ∼125 GB of bitset across 32 GPUs at roughly 4 GB per device — entirely feasible with NVLink for fast intra-node aggregation.

Cryptographic Workloads at Scale. For the cryptographic extension, a distributed GPU cluster would unlock two genuinely new capabilities. First, exhaustive Sophie Germain and safe prime enumeration up to $N = 10^{13}$ or beyond would become tractable — a range of practical interest for studying the density of cryptographically strong primes. Second, GPU-accelerated Miller–Rabin primality testing could be parallelized across a batch of large RSA candidate integers, with each GPU thread testing a different candidate using independent witness bases. At scale, this hybrid approach — sieve prefilter on CPU to eliminate composites with small factors, followed by GPU-parallel Miller–Rabin for the survivors — mirrors production-grade prime generation pipelines used in cryptographic libraries. The key engineering challenge would shift from compute throughput to *pipeline staging*: keeping all GPU SMs saturated requires overlapping data preparation, kernel execution, and result aggregation across the cluster.

This memo was written as Part 4 of IMC05 for CISC 719 – Contemporary Computing Systems Modeling Algorithms (CCSM), Spring 2026.
Kenneth Peter Fernandes | Ph.D. in Computational Sciences | Harrisburg University of Science and Technology