International Journal of Emerging Engineering and Technology (IJEET)
ISSN (e): 2958-3764
Volume: 5, Number: 1, Pages: 1 - 5, Year: 2026

Programming Limitations and Challenges on Consumer Hardware GPU

Muhammad Ahmad *, Nabeel Akram, Hira Arif

Department of Computer Science, Superior University, Lahore, Pakistan

*Corresponding author: Muhammad Ahmad (Email: muhammadahmad2001318@gmail.com)

Received: 12/03/2026, Revised: 22/05/2026, Accepted: 15/06/2026

***Abstract*— Traditionally, high-performance computing was dominated by processors specifically designed to execute highly efficient mathematical calculations. But there are major architectural, thermal and software challenges involved in using consumer-grade and edge GPU hardware. We discuss the structural shortcomings in GPU programming, including a lack of fine-grained preemptive multitasking, thermal throttling under tight power limits, and memory vulnerabilities in the absence of enterprise-grade Error Checking and Correction on graphics cards. We also examine new software bottlenecks, specifically in LLM serving, and address challenges in distributed orchestration. In this work, we review existing frameworks such as FLEP, Phoenix and FlowPrefill to exemplify the need for full-stack solutions that are holistic in scope (encompassing hardware-level pre-emption via resource-efficient flow control, spatial sharing among diverse workloads, as well as dynamic scheduling) if we want to navigate around limitations posed by emerging heterogeneous compute architectures.**

***Index Terms*—Distributed orchestration, GPU computing, heterogeneous architectures, LLM serving, preemptive multitasking, soft errors, thermal throttling.**

I. Introduction

The transition from rendering pipelines to general-purpose, massively parallel accelerators has radically transformed the computing landscape. As Moore's law approaches its physical limits, architectural scaling has shifted towards radical heterogeneity to meet the gargantuan demands of Big Data, Artificial Intelligence, and scientific simulations [1-8]. Modern hardware environments consist of highly heterogeneous combinations of elements, including Central Processing Units (CPUs), GPUs, Field-Programmable Gate Arrays (FPGAs) and specialised neural or quantum accelerators, as attested in [8-10]. CPUs are engineered as MIMD (Multiple Instruction, Multiple Data) devices that must optimise extraordinarily complex control logic and minimise the time between task switching [11], while GPUs follow a SIMD (Single Instruction, Multiple Data) philosophy [12-14]. This binary nature means that GPUs are potentially only suitable for data-parallel situations, where the same kernel function is performed simultaneously on large, equivalent datasets [15].

Theoretical models of parallel computing help us understand the exact limits of GPU hardware. Abstract concepts such as the PRAM (Parallel Random Access Machine) and UPMH (Uniform Parallel Memory Hierarchy) describe idealised conditions for parallel execution [16]. So, the PRAM model operates under paradigms such as Exclusive Read Exclusive Write (EREW) and Concurrent Read Concurrent Write (CRCW), providing mathematical bounds on algorithm scaling [17]. But consumer hardware is a big departure from those abstractions. Problems with strict time dependencies, such as those in which successive iterations critically depend on prior iterations (as in the Newton-Raphson method), are inherently unparallelizable. Even more crucially, real-world GPU deployment requires complex manual cache management, mesh mapping among threads, and alignment of memory access patterns to avoid draconian performance drops [17].

II. Literature Review

The issues of GPU programming on consumer and edge hardware have been considered in several computational domains in the literature, from runtime orchestration to hardware preemption to power policy management to memory reliability and novel LLM bottlenecks. The field of distributed orchestration and task management has diverse approaches to architectural heterogeneity in the literature. However, single-node runtime systems do not natively support inter-node coordination, necessitating middleware layers such as D-IRIS to transparently manage distributed tasks and associated data across a multi-node cluster [2]. Meanwhile, with the advent of heterogeneous environments featuring Quantum Processing Units (QPUs), orchestration frameworks like Q-IRIS have emerged to coordinate hybrid classical-quantum workflows by treating quantum accelerators in a manner similar to CPUs and GPUs [1]. To prevent further CPU bottlenecks in distributed scenarios, RDMA networking models designed for GPUs, which allow direct access from the network interface to GPU memory [14], have been proposed.

Native preemption is emphasised in research on GPU hardware constraints. Tanasic et al. showed the excessive overhead of context switching and suggested hardware extensions for draining SMs and Dynamic Spatial Sharing (DSS) [10]. Wu et al. took a software perspective on this with FLEP, a compilation and runtime system that provides fully transparent and highly efficient spatial and temporal preemption, thus greatly reducing priority inversion [12]. Thermal factors and power management are still major

This work is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License, which permits Unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited

bottlenecks, especially for integrated CPU+GPU chips and mobile systems. Building on this, Dev and Reda used infrared imaging to show that chip power caps and CPU loads greatly change the optimal scheduling device [11], and proposed dynamic machine-learning-based schedulers. Jeon et al. proposed the Phoenix framework, which uses reinforcement learning and multi-exit networks to delay thermal throttling during constant DNN inference [3]. Kim et al. addressed execution optimisation by scheduling at the operation level [5].

The quality of consumer GPU memory has been examined by Haque and Pande, using the MemtestG80 tool throughout the Folding@home network to measure real-world soft error rates [7]. Sullivan et al. further examined HBM2 memory [13], showing that soft errors are infrequent yet highly severe and exhibit multi-bit corruption with respect to byte alignment; hence, sophisticated ECC schemes such as TrioECC and DuetECC need to be deployed to avoid silent data corruption [13]. Lastly, the emergence of Large Language Models (LLMs) has created certain scheduling bottlenecks. Hsieh et al. decoupled preemption from execution granularity [9] to address head-of-line blocking in disaggregated serving. Similarly, Liao et al. introduced a laser system to provide flexible scheduling with layer-level granularity, rather than inflexible iteration-level scheduling, to satisfy various service-level objectives (SLOs) [16]. Thermal-aware profiling with Nsight Python [15] and compensation for high-speed atmospheric turbulence using the WOLF method [4] are examples of domain-specific tools that underscore the need for software solutions that circumvent hardware limitations.

III. The Preemption Conundrum in Massively Parallel Architectures

The lack of fine-grained preemptive multitasking is one of the most stubborn structural limitations of modern GPU architectures. Traditional operating systems rely on hardware-supported preemptive scheduling to enable the CPU to quickly switch between tasks and to guarantee system responsiveness and fairness. By contrast, workload execution on GPUs proceeds by dispatching large arrays of thread blocks to Streaming Multiprocessors (SMs), executing them uninterrupted until completion [10]. Commands issued to the GPU execution engine have exclusive access, and a single long-latency kernel can corrupt the entire hardware array with fatal priority inversion [10]. Such situations lead to an indefinite blockage of high-priority, low-latency applications by low-priority, multicast kernels [10].

A. Context Switching Overhead versus SM Draining

Preemption on GPUs comes with significant hardware and latency overheads. In architectures like NVIDIA GK110 (Kepler), a full context switch involves saving the complete state of all running threads. Accordingly, transferring this state to off-chip memory may take up to 44 microseconds at peak memory bandwidth; a single GPU core can contain up to 256 KB of register file storage and 48 KB of on-chip shared memory [10]. This introduces a significant latency bottleneck that inherently limits real-time scheduling, as modern CPUs complete context switches in less than 1 microsecond [10]. Two diverse preemption mechanisms have been proposed and implemented to address these limitations: the conventional stop-and-save context switch and SM draining. The latter technique avoids moving huge amounts of data by stopping new threads from being issued to the SM, so that currently executing threads may run to completion before yielding control of the hardware to a higher-priority task [10].

B. Flexible and Spatial Preemption (FLEP)

Such limitations have been characterised in detail by advanced frameworks such as FLEP (Flexible and Efficient Preemption), which use both temporal and spatial preemption. Traditional software-based approaches, such as kernel slicing, use artificial sub-kernels [12] to set preemption boundaries, leading to significant overhead due to repeated invocations into the kernel. FLEP circumvents this compiler quirk by generating preemptable GPU programs during compilation, using Clang LibTooling to transparently manipulate the abstract syntax tree [12]. It produces preemptable kernel versions that add only 2.5% runtime overhead [12]. Importantly, FLEP develops the notion of spatial preemption. A running kernel can yield only a subset of SMs via the %smid register on NVIDIA architectures [12], rather than the entire GPU. Spatial preemption reduces preemption latency for a waiting, high-priority kernel that uses only a few SMs by up to 41% [12]. Experimental results show FLEP achieves 3X to 24.2X speedup for top-20 kernels [12].

C. Dynamic Spatial Sharing (DSS)

Running alongside FLEP is a policy called Dynamic Spatial Sharing (DSS) that divides hardware resources by assigning non-overlapping sets of GPU cores to different processes based on their operating system priority. In DSS, a token-based algorithm is implemented where the budget of an SM is allocated as tokens to kernels that are decremented (when assigned) and returned (after releasing the SM) [10]. Based on this insight, a new hardware scheduling policy was proposed that increases the execution time of high-priority processes and improves system fairness by up to 15.6x and 3.4x respectively [10]. But these enhancements come at a price — they lead to an overall throughput drop of 12% to 35%, primarily due to the underutilization of streaming multiprocessors as partitions are dynamically shifted [10]. Table I summarizes the trade-offs among these preemption mechanisms.

TABLE I
COMPARISON OF GPU PREEMPTION MECHANISMS

Preemption Mechanism	Primary Operational Advantage	Primary Architectural Disadvantage	Empirical Performance Impact
Stop-and-Save (Context Switch)	Predictable preemption boundaries.	High latency due to off-chip memory transfers.	15.6x speedup for high priority, 12% throughput degradation.
SM Draining	No massive data movement off-chip.	Unpredictable latency based on active thread execution duration.	Eliminates state-saving delays but risks delayed preemption.
Kernel Slicing	Programmable without hardware modifications.	High overhead from repeated API invocations and kernel launches.	Trade-off between responsiveness and severe runtime overhead.
Spatial Preemption (FLEP)	Yields only required SMs, leaving others active.	Requires deep compiler transformation and AST manipulation.	41% latency reduction, 2.5% runtime overhead, 24.2X speedup.

IV. THERMAL DYNAMICS AND POWER CONSTRAINTS

Thermal throttling and strict power budgets become significant limiting factors as heterogeneous computing extends from large data centres down to embedded devices and consumer electronics. In integrated CPU+GPU processors, where separate architectural devices exist on the same silicon die, making a scheduling decision fundamentally changes the thermal and power profile of the entire chip [11].

A. Infrared Thermal Profiling and Device Mapping

Infrared imaging of integrated CPU+GPU processors shows tremendous spatial asymmetries in thermal generation. For instance, mapping a vector-multiplication OpenCL workload to the GPU rather than the CPU can reduce peak die temperatures while consuming much less total power and delivering 5.5x better performance [11]. However, the best mapping target is not fixed; it depends strongly on dynamic, time-variable physical conditions including, e.g., the total power limit of the chip [11]. Using machine learning models monitorsuch as Support Vector Machine (SVM) classifiers, dynamic scheduling frameworks observe hardware performance counters and available power budgets to decide how to remap workloads over time. Such adaptive mapping achieves up to 31% runtime and 10% energy efficiency gains over static scheduling [11].

B. Mobile and Embedded Constraints: The Phoenix Framework

It is not feasible to keep running multi-instance Deep Neural Networks (DNNs) on mobile consumer hardware because they produce continuous output that can quickly exceed the device's Thermal Design Power (TDP) [3]. When thermal throttling kicks in, computing power is reduced across heterogeneous processors in a nonuniform manner, thereby annihilating frame-rate consistency [3].

Frameworks like Phoenix overcome this by employing DQN-based reinforcement learning agents to generate thermally aware allocation policies [3]. For example, in experimental setups, this approach delayed the onset of thermal throttling from 61 seconds to 217 seconds by mapping tasks to processors based on their heat-generation profiles [3]. When throttling will occur is dependent on the particular hardware's thermal dynamics; as such, Phoenix implements multi-exit networks with configuration via neural architecture search (NAS) in such a way that it can adaptively perform inference at shallower exit layers to maintain a consistent frame rate [3].

V. MEMORY RELIABILITY: SOFT ERRORS AND ECC MECHANISMS

Consumer GPUs lack the error-checking and correcting (ECC) memory modules found on enterprise servers, making them prone to hardware corruption. A broad analysis of the Folding@home distributed computing network, using the MemtestG80 tool and other hardware tests, showed that 66% of tested consumer graphics chipsets exhibited a significant, pattern-sensitive rate of memory soft errors [7]. Without ECC, these consumer boards are very prone to silent data corruption [7].

A. Displacement Damage versus Cosmic Soft Errors

In high-performance compute GPUs, which may be equipped with High Bandwidth Memory (HBM2), ensuring memory reliability is especially challenging. Cosmic rays are simulated with a high-energy neutron beam testing [13], which has also uncovered unique failure modes. One key observation is the difference between standard soft errors and so-called intermittent errors due to "displacement damage," which occurs when significant energy radiation physically damages the DRAM access transistor, increasing cell leakage current [13]. Because faulty cells that produce such errors are damaged, they exhibit intermittent, single-bit, unidirectional faults that can be filtered out during testing to isolate true soft-error patterns [13].

B. Spatial Locality of HBM2 Errors and Advanced ECC

HBM2 has a hierarchical, massively wide interface for each 32-byte memory access from the same DRAM device [13]. HBM2 soft errors are moderately rare but very severe. According to [13], approximately 31.5% of SEUs affect more than one bit in at least one word. Since the HBM mats allow for 8-bit-granularity access, as in most commercial HBM, almost 75% of severe errors occur in a logically contiguous byte within a word of memory [13]. The common Single-Error-Correcting, Double-Error-Detecting (SEC-DED) codes utilized inside modern-day GPUs are orders of magnitude worse for all such byte-aligned corruptions. For example, proposed mitigations like TrioECC take advantage of the byte-aligned geometry of HBM2 failures to work as a drop-in replacement for traditional SEC-DED, reducing uncorrectable errors by 7.87x and the chance of SDC by two orders of magnitude [13]. Other symbol-based architectures reduce the risk of SDC by five orders of magnitude [13]. Table II compares these ECC architectures.

TABLE II
ECC ARCHITECTURES FOR GPU MEMORY

ECC Architecture	Target Environment	SDC Risk Reduction (vs SEC-DED)	Key Characteristics
SEC-DED	Standard GPU DRAM	Baseline	Suboptimal for byte-aligned multi-bit errors.
TrioECC	HBM2 GPU Memory	100x Reduction	7.87x fewer uncorrectable errors; no extra hardware footprint.
DuetECC	HBM2 GPU Memory	>1,000x Reduction	Prioritizes SDC reduction over aggressive correction.
SSC-DSD+	HBM2 GPU Memory	100,000x Reduction	Symbol-based design; larger decoder footprint.

VI. LARGE LANGUAGE MODEL (LLM) SERVING BOTTLENECKS

The explosion of Large Language Models (LLMs) has created whole new classes of workload bottlenecks on GPU hardware. LLM inference breaks computation into a heavily compute-bound prefill phase and a strictly memory-bandwidth-bound decode phase [9]. To mitigate this imbalance, contemporary architectures use Prefill-Decode (PD) disaggregation that decouples workloads at the physical level, but this localizes huge resource contention to prefill instances [9].

A. Head-of-Line Blocking and Operator-Level Preemption

While handling a long-context prefill request, the GPU incurs HoL blocking that inhibits scheduling for recently arrived latency-sensitive requests [9]. Conventional systems try to address this with chunked prefill, but small chunk sizes incur significant kernel launch overhead, degrading computational efficiency, while larger chunk sizes reintroduce HoL blocking [9].

Operator-Level Preemption [9] is an advanced scheduling approach designed to resolve this conflict. By injecting lightweight preemption checks at natural boundaries of the core operators, the system can interrupt a long-running prefill almost immediately [9]. In addition, FlowPrefill employs Event-Driven Scheduling by activating a schedule only during request arrival and completion events, with enhancements from Slack-aware Earliest-Deadline-First (S-EDF) to maximise goodput for up to 5.6x improvement [9].

B. Layer-Level Scheduling for Multi-SLO Workloads

The diversity of user expectations makes multi-SLO serving necessary. This approach breaks down for systems that use iteration-level scheduling, since they schedule requests in fixed-length intervals [16]. Laser solves this via layer-level scheduling: it breaks down the computation into fine-grained operations that correspond to individual layers [16]. During the decode phase, layer-level decode batching provides fine-grained control over how many layers execute in a single run for each request [16]. Laser specializes in serving requests by tailoring execution depth to the specific response time SLO target of each request, thus achieving an over 1.67x improvement in overall serving goodput while strictly guaranteeing multi-SLO attainment [16].

VII. DISTRIBUTED ORCHESTRATION AND DATA MOVEMENT

Computational demands can now easily overtake single-node capability, so applications must speak the language of clusters. However, existing single-node runtime systems lack native mechanisms for coordinating across nodes [2].

A. Multi-Node Task Dependency and Centralized Data Handling

Distributed middleware layers, e.g., D-IRIS, dynamically intercept API calls through preloadable shared libraries, and a global task dependency Directed Acyclic Graph (DAG) is reconstructed without modifying the source code [2]. Although compute-intensive tasks show good scalability, this incurs extreme performance overhead in communication-heavy tasks because of centralized data handling bottlenecks [2].

B. GPU-Centric RDMA Networking

Fast paths can avoid OS networking stacks to remove the latency added by their hops, and modern deployments use Remote Direct Memory Access (RDMA). A GPU-centric RDMA approach accomplishes this by bypassing the CPU and moving data directly from the NIC to the GPU's memory [14]. This kernel-bypass mechanism saturates 100-Gbps network links and reduces machine learning inference response times by approximately 50%, thereby making additional CPU cycles available to other workloads [14].

VIII. DOMAIN-SPECIFIC BOTTLENECKS AND EMERGING WORKFLOWS

A. 5G O-RAN and DVFS Sensitivity

In 5G Open Radio Access Networks (O-RAN), meeting sub-millisecond timing requirements is highly sensitive to Dynamic Voltage and Frequency Scaling (DVFS) [6]. For instance, a 1410 MHz GPU paired with a 2.8 GHz CPU has a very stable coefficient of variation (cv = 0.2%), while changing the CPU frequency to 3.0 GHz resulted in at least cv = 60% [6], indicating non-monotonic behaviour between throughput stability and CPU frequency.

B. Spatial Independence in Mathematical Optimization

The Well Optimized Linear Finder (WOLF) method of compensating atmospheric turbulence is a prime example of an algorithm that must be reworked to fit the GPU hardware format. The WOLF method replaces global iterative loops with a structured pointwise autocorrelation of the generalized pupil function, which yields an 80x speedup in execution time compared to serial CPU implementations [4].

C. Embedded Operation-Level Scheduling

Deep learning models are represented as Directed Acyclic Graphs (DAGs), and operation-level scheduling frameworks have been developed primarily for embedded systems, such as the NVIDIA Jetson Nano and ODROID-XU4 [5]. In an offline profiling stage, regression models are trained to predict latency based on tensor dimensions. The scheduler uses these models to decide whether to run an operation on the CPU only, the GPU only, or both devices concurrently, resulting in up to 40% and 74% reductions in end-to-end inference time [5].

D. Python-Native Profiling and Thermal Management

Python Domain-Specific Languages have made GPUs much more accessible, but traditional profiling still incurs some cost to the surrounding Python ecosystem [15]. Toolkits such as Nsight Python provide a Python-first profiling interface based on decorators and context managers. It features "Thermovision" thermal management, which fully interfaces with NVML to mitigate GPU throttling that can severely skew performance metrics by up to 29% [15].

E. The Horizon: Hybrid Quantum-Classical Environments

Quantum Processing Units (QPUs) need to be integrated alongside a runtime that can orchestrate hybrid classical-quantum workflows [1]. For example, runtimes such as Q-IRIS treat the QPU as QRAM in an asynchronous, task-based environment [1]. For efficient throughput, techniques such as quantum circuit cutting break down complex circuits into smaller subcircuits that can run in parallel to address scaling challenges on massive hybrid infrastructure [1].

IX. Conclusion

Architectural deployment of GPU architectures is still constrained by a broad range of complexities across architecture, thermal, memory, and software domains. Due to the lack of native and hardware-level preemption, priority inversion is unavoidable, necessitating reliance on compiler-based spatial preemption and dynamic spatial sharing. As thermal limits necessitate highly dynamic scheduling decisions, solution approaches based on machine-learning or reinforcement-learning frameworks must be employed to avoid substantial thermal throttling in these processors. Moreover, the underlying physical properties of HBM2 memory make systems vulnerable to byte-aligned soft errors; consequently, ECC mechanisms must be highly specialised, such as TrioECC. In the end, there is no magic bullet to eliminate such limitations; fundamentally, we need a cross-layer, full-stack perspective, from GPU-centric RDMA networking to layer-level batching for LLMs, to extract maximum performance from the extreme heterogeneity of modern compute hardware.

Funding Statement

The authors received no specific funding for this study.

Conflicts of Interest

The authors declare no conflicts of interest to report regarding the present study.

Author Contributions

All authors contributed to the conception, literature review, drafting, and critical revision of this manuscript and approved the final version for submission.

Data Availability Statement

Data is available on reasonable request.

INSTITUTIONAL REVIEW BOARD STATEMENT

Not applicable.

INFORMED CONSENT STATEMENT

Not applicable.

References

[1] N. R. Miniskar et al., "Q-IRIS: The Evolution of the IRIS Task-Based Runtime to Enable Classical-Quantum Workflows," in SCA/HPCAsia 2026 Workshops, Osaka, Japan, Jan. 2026.

[2] F. Ozdemir and I. Akturk, "Distributed Runtime Support for Portable and Scalable Execution of Heterogeneous Applications," in SCA/HPCAsia 2026 Workshops, Osaka, Japan, Jan. 2026.

[3] S. Jeon, J. Kim, J. Lee, and H. Cha, "Phoenix: Thermal-Aware On-device Inference of Multi-Instance DNNs for Mobile Video Applications," ACM Transactions on Embedded Computing Systems, Feb. 2026.

[4] T. E. Coon, "Advancing Real-World Implementation of the Well Optimized Linear Finder (WOLF) High-Speed Atmospheric Turbulence Compensation Method," Ph.D. dissertation, Florida Institute of Technology, Melbourne, FL, USA, Aug. 2025.

[5] M. Kim, S. Choi, S. Wang, and C. Y. Jeong, "Operation-level scheduling framework for efficient deep learning inference on embedded systems using directed acyclic graphs," ETRI Journal, Sep. 2025.

[6] A. Elango, E. Baena, and D. Koutsonikolas, "Exploratory Study of CPU-GPU Frequency Interactions and Throughput Stability in GPU-Accelerated 5G O-RAN," Institute for Intelligent Networked Systems, Northeastern University, Boston, MA, USA.

[7] I. S. Haque and V. S. Pande, "Hard Data on Soft Errors: A Large-Scale Assessment of Real-World Error Rates in GPGPU," arXiv:0910.0505v2, Nov. 2009.

[8] P. O. A. Navaux, A. F. Lorenzon, and M. d. S. Serpa, "Challenges in High-Performance Computing," Journal of the Brazilian Computer Society, vol. 29, no. 1, Aug. 2023.

[9] C. Hsieh et al., "FlowPrefill: Decoupling Preemption from Prefill Scheduling Granularity to Mitigate Head-of-Line Blocking in LLM Serving," arXiv:2602.16603v1, Feb. 2026.

[10] I. Tanasic et al., "Enabling Preemptive Multiprogramming on GPUs," in 2014 ACM/IEEE 41st International Symposium on Computer Architecture (ISCA), 2014.

[11] K. Dev and S. Reda, "Scheduling Challenges and Opportunities in Integrated CPU+GPU Processors," in ESTIMedia '16, Pittsburgh, PA, USA, Oct. 2016.

[12] B. Wu, X. Liu, X. Zhou, and C. Jiang, "FLEP: Enabling Flexible and Efficient Preemption on GPUs," in ASPLOS '17, Xi'an, China, Apr. 2017.

[13] M. B. Sullivan et al., "Characterizing and Mitigating Soft Errors in GPU DRAM," in MICRO '21, Virtual Event, Greece, Oct. 2021.

[14] M. Girondi, M. Scazzariello, G. Q. Maguire Jr., and D. Kostić, "Toward GPU-centric Networking on Commodity Hardware," in EdgeSys '24, Athens, Greece, Apr. 2024.

[15] B. Hagedorn and A. Collins, "Nsight Python: A Python-First Profiling Toolkit for Seamless GPU Kernel Analysis," in Proc. 35th ACM SIGPLAN International Conference on Compiler Construction (CC '26), Sydney, Australia, Jan. 2026.

[16] J. Liao, Q. Dong, Y. Liang, Z. Zhou, and X. Chen, "Laser: Unlocking Layer-Level Scheduling for Efficient Multi-SLO LLM Serving," in Proc. 31st ACM SIGPLAN Annual Symposium on Principles and Practice of Parallel Programming (PPoPP '26), Sydney, Australia, 2026, pp. 509–521.

[17] C. A. Navarro, N. Hitschfeld-Kahler, and L. Mateu, “A Survey on Parallel Computing and its Applications in Data-Parallel Problems Using GPU Architectures,” Communications in Computational Physics, vol. 15, no. 2, pp. 285–329, 2014.