Review

Not peer-reviewed version

A Survey on Advancements in Scheduling Techniques for Efficient Deep Learning Computations on GPUs

Rupinder Kaur *, Arghavan Asad, Seham Al Abdul Wahid, Farah Mohammadi

Posted Date: 20 February 2025

doi: 10.20944/preprints202412.0276.v2

Keywords: Deep Learning Algorithms; Deep-Neural Networks (DNN); Energy-efficient scheduling; Performance optimization; Heterogeneous computing; Machine Learning (ML)

Review

A Survey on Advancements in Scheduling Techniques for Efficient Deep Learning Computations on GPUs

Rupinder Kaur [1,*], Arghavan Asad [2], Seham Al Abdul Wahid [1] and Farah Mohammadi [1]

[1] Electrical, Computer and Biomedical Engineering Department, Toronto Metropolitan University, 350 Victoria St, Toronto, M5B2K3, Ontario, Canada

[2] School of Computer Science and Technology, Algoma University, Brampton, Ontario, Canada

* Correspondence: rupinder.kaur.ece@torontomu.ca

Abstract: This comprehensive survey explores recent advancements in scheduling techniques for efficient deep learning computations on GPUs. The article highlights challenges related to parallel thread execution, resource utilization, and memory latency in GPUs, which can lead to suboptimal performance. The surveyed research focuses on novel scheduling policies to improve memory latency tolerance, exploit parallelism, and enhance GPU resource utilization. Additionally, it explores the integration of prefetching mechanisms, fine-grained warp scheduling, and warp switching strategies to optimize deep learning computations. "These techniques" demonstrate significant improvements in throughput, memory bank parallelism, and latency reduction. The insights gained from this survey can guide researchers, system designers, and practitioners in developing more efficient and powerful deep learning systems on GPUs. Furthermore, potential future research directions include advanced scheduling techniques, energy efficiency considerations, and the integration of emerging computing technologies. By continuously advancing scheduling techniques, the full potential of GPUs can be unlocked for a wide range of applications and domains, including GPU-accelerated deep learning, task scheduling, resource management, memory optimization, and more.

Keywords: deep learning algorithms; Deep-Neural Networks (DNN); energy-efficient scheduling; performance optimization; heterogeneous computing; Machine Learning (ML)

1. Introduction

Deep learning has emerged as a powerful technique for solving complex problems and has achieved remarkable success in various domains, including computer vision, natural language processing, and autonomous systems. Graphics Processing Units (GPUs) have played a pivotal role in accelerating deep learning computations due to their parallel processing capabilities and high memory bandwidth. However, harnessing the full potential of GPUs for deep learning requires efficient scheduling techniques that optimize resource utilization and minimize latency [1-5].

The efficient execution of deep learning algorithms on GPUs faces several challenges. Firstly, deep learning computations often involve massive amounts of data and complex neural network architectures, resulting in high computational demands. Secondly, GPUs consist of numerous parallel processing units known as warps, which execute threads in parallel. However, inefficient scheduling of these warps can lead to underutilization of GPU resources and suboptimal performance. Moreover, memory latency, caused by data movement between the GPU and memory, can significantly impact the overall system performance [10-16].

To address these challenges, researchers have been actively developing advanced scheduling techniques specifically tailored for deep learning computations on GPUs. These techniques aim to maximize the efficiency and throughput of deep learning algorithms by effectively managing the

 doi:10.20944/preprints202412.0276.v2

execution of parallel threads, optimizing memory access patterns, and improving resource utilization [17-20]. Scheduling techniques for deep learning on GPUs encompass various aspects, such as warp scheduling policies, memory latency tolerance, memory bank parallelism, context switching strategies, and prefetching mechanisms. Warp scheduling policies focus on efficiently assigning threads to warps, considering factors like data dependencies and resource availability. Memory latency tolerance techniques aim to overlap memory access and computation to minimize the impact of memory latency on overall performance [21-25]. Memory bank parallelism techniques exploit the parallel memory access capabilities of GPUs to enhance memory throughput. Context switching strategies involve efficiently switching between different warps to optimize GPU resource utilization.

Additionally, prefetching mechanisms are employed to anticipate and fetch data in advance, reducing memory access latency and improving overall performance [26-31]. Fine-grained warp scheduling techniques consider the execution order and prioritization of individual threads within a warp, further enhancing parallelism and reducing resource contention. Furthermore, hardware and algorithm co-design approaches have been explored to optimize scheduling and execution for specific types of deep learning algorithms, such as sparse neural networks and Convolutional Neural Networks (CNNs). Techniques for optimizing cache data residency have been explored to enhance memory management and leverage cache locality. By analyzing data access frequencies and categorizing applications based on their access patterns, these methods improve data storage and boost cache efficiency, resulting in better overall performance. In this article, we provide a comprehensive survey of recent advancements in scheduling techniques for efficient deep learning computations on GPUs [32-37]. We delve into the details of various scheduling methodologies, their underlying principles, and their impact on system performance. Through experimental evaluations, we demonstrate the effectiveness of these techniques in improving throughput, reducing latency, and optimizing resource utilization. The insights gained from this survey can guide researchers, system designers, and practitioners in developing more efficient and powerful deep learning systems on GPUs [38-43]. Additionally, we discuss potential future directions for research, including advanced scheduling techniques, energy efficiency considerations, and the integration of emerging computing technologies. By continuously advancing scheduling techniques, we can unlock the full potential of GPUs and further optimize deep learning computations for a wide range of applications and domains [44-55].

Figure 1 illustrates the classification of scheduling techniques discussed in this survey, categorizing them based on their application and methodology for optimizing deep learning computations on GPUs.

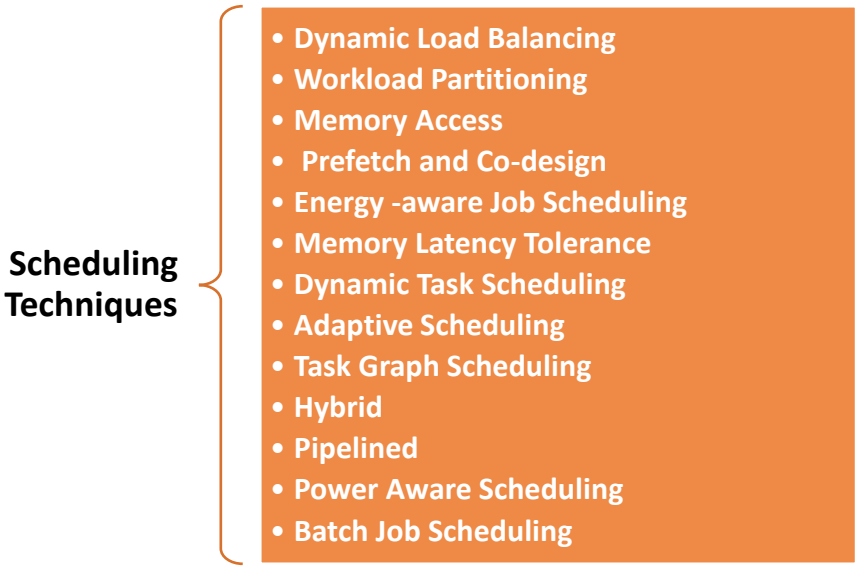

Figure 1. Classification of Scheduling Techniques discussed in this survey.

 doi:10.20944/preprints202412.0276.v2

At the core of this survey, we explore a variety of scheduling techniques critical for optimizing deep learning computations on GPUs. In Section II, the discussion begins with heterogeneous computing, highlighting its promises and challenges in combining CPUs and GPUs for improved performance and energy efficiency. Next, in section III we delve into diverse scheduling strategies, including static, dynamic, and adaptive approaches, focusing on their ability to optimize workload distribution and resource utilization. Dynamic load balancing techniques are examined for their real-time adaptability, while workload partitioning strategies emphasize efficient division of tasks between CPUs and GPUs. Memory access scheduling methods are explored to reduce latency and enhance throughput, complemented by fine-grained warp scheduling policies that prioritize threads to improve parallelism. Context switching strategies are discussed for their role in minimizing idle times and optimizing resource usage. Energy-aware and task graph scheduling techniques are presented to balance energy efficiency, manage dependencies, and improve completion times. Additionally, hybrid scheduling approaches, power-aware and deadline-aware scheduling, and batch job scheduling are analyzed for their ability to address complex resource management challenges in heterogeneous systems. Pipelined scheduling strategies are reviewed for their potential to overlap computation stages and boost throughput. This comprehensive discussion sets the stage for the subsequent sections, offering insights into the latest advancements and future directions in scheduling techniques for GPU-accelerated deep learning.

2. Heterogeneous Computing: Promises and Challenges

Heterogeneous computing aims to combine the strengths of both CPUs and GPUs to achieve improved computational performance. The promises of heterogeneous computing include utilizing the unique features of CPUs and GPUs, achieving load balancing between the processing units, and harnessing the computational power of both to strive for exascale performance.

However, there are several challenges in realizing the potential of heterogeneous computing. The vastly different architectures and programming models of CPUs and GPUs present obstacles in achieving collaborative computing. Optimization techniques designed for CPU-only or GPU-only systems may not be effective in heterogeneous systems. Therefore, novel techniques are required to optimize performance and energy efficiency by considering the characteristics of both CPUs and GPUs.

 doi:10.20944/preprints202412.0276.v2

3. Scheduling Strategies

The goal of heterogeneous computing is to leverage the unique features and strengths of both CPUs and GPUs to achieve high-performance computing. The increasing utilization of both CPUs and GPUs in a wide range of applications underscores the need for collaboration between these processing units, as highlighted in [1]. It discusses the challenges and promises of heterogeneous computing and clarifies the terminology used in the field. Regarding scheduling techniques, it categorizes the research works based on the nature of scheduling, such as dynamic or static load balancing. It explores various approaches to workload partitioning between CPUs and GPUs, considering factors like the division of work among processing units.

3.1. Dynamic Load Balancing

Dynamic load balancing involves dynamically redistributing the workload between CPUs and GPUs based on their capabilities and current load. This technique aims to optimize resource utilization and maximize overall performance by adapting to changes in workload and system conditions in real-time. It can help ensure that both CPUs and GPUs are effectively utilized and that the workload is distributed efficiently to achieve better performance.

The problem of online scheduling for task graphs on heterogeneous platforms, particularly those comprising CPUs and GPUs, is addressed by Canon et al. [7]. Their focus is on dynamic scheduling, where task graphs are revealed in real-time during computation, and the scheduler operates with limited information about the available tasks. The objective is to minimize the total completion time of the tasks.

1. Flexibility and Knowledge Impact: the paper examines how factors such as task migration, preemption, and partial graph knowledge influence scheduling efficiency and the algorithm's lower bounds.
2. Competitive Algorithm: The authors propose a competitive algorithm, called QA, and refine its analysis to achieve better performance.
3. Generalization to Multiple Processor Types: The authors extend their results to the case of platforms with multiple types of processors, adapting the lower bounds and the online algorithm accordingly.
4. Heuristic Approach: The authors propose a simple heuristic based on the QA algorithm and the system-oriented heuristic EFT. This heuristic is both competitive and performs well in practice.

The scheduling technique READYS, developed by Nathan et al. [10], aims to create a dynamic scheduling algorithm for heterogeneous computing systems utilizing reinforcement learning. This approach incorporates Directed Acyclic Graphs (DAGs) as a computational model for parallel applications. It emphasizes the importance of accurate task allocation and scheduling decisions in parallel and distributed computing systems. The authors propose using reinforcement learning, specifically a combination of Graph Convolutional Networks (GCN) and an Actor-Critic Algorithm (A2C), to address the dynamic scheduling problem. The READYS algorithm aims to minimize the makespan, which is the total time required to complete all tasks, by learning an adaptive scheduling strategy based on the current state of the system and the characteristics of the tasks. The use of reinforcement learning is particularly suitable in scenarios where task durations and communication times are stochastic. The paper focuses on task graphs derived from linear algebra factorization kernels (such as CHOLESKY, LU, and QR) and considers heterogeneous platforms with both CPUs and GPUs. Through simulations, the authors demonstrate that READYS achieves comparable or superior performance to existing scheduling algorithms, particularly in environments with high uncertainty. Furthermore, READYS is capable of generalizing its learned scheduling strategy to different application domains and problem sizes.

An improved scheduling mechanism for generative AI applications in cloud-native environments with heterogeneous resources is proposed by Chun et al. [28]. This approach employs

 doi:10.20944/preprints202412.0276.v2

dynamic programming as the scheduling technique. The study addresses the challenge of effectively scheduling multiple resources in such environments to enhance load balance and resource utilization. By applying the dynamic programming approach, the proposed scheduling mechanism outperforms native GPU scheduling strategies and the algorithm for the multidimensional knapsack problem. Experimental results demonstrate that the load balance can be improved by up to 64.1%, and the makespan (job completion time) can be reduced by up to 40.8%. This scheduling technique considers load balancing as a crucial metric and aims to achieve the highest total iteration count while optimizing job completion time. The approach effectively utilizes the available resources, including less powerful GPU nodes, and prevents resource idleness caused by unbalanced workloads.

On the other hand, Static load balancing involves pre-determining the workload division between CPUs and GPUs based on certain criteria or heuristics. This technique aims to distribute the workload evenly and efficiently at the beginning of the computation. It does not adapt to runtime changes and assumes a predetermined workload division that remains constant throughout the execution. Static load balancing can be useful in scenarios where the workload characteristics are known in advance or when the workload distribution is relatively stable.

3.2. Workload Partitioning

Effective workload division between CPUs and GPUs is essential in heterogeneous computing. Static partitioning predefines task allocation based on system characteristics, while dynamic partitioning adjusts allocations at runtime to optimize performance. Factors such as computation intensity, memory requirements, and inter-task dependencies are considered to ensure balanced resource usage. Workload partitioning focuses on dividing the workload between CPUs and GPUs in an effective manner. This technique considers factors such as the characteristics of the workload, the capabilities of the processing units, and the requirements of the deep learning computations. By carefully partitioning the workload, workload partitioning aims to achieve balanced utilization of CPUs and GPUs, optimize resource allocation, and improve overall performance.

The Constrained Autonomous Workload Scheduler (CAuWS), proposed by Justin et al. [46], is designed for Cyber-Physical Systems (CPS) with heterogeneous processing units. This technique addresses the challenge of creating efficient schedules that satisfy the physical requirements of CPS while accommodating the diverse capabilities of various hardware units. CAuWS utilizes a structured and system-agnostic approach, combining a representation language (AuWL), timed Petri nets, and mixed-integer linear programming. It enables the representation and scheduling of various CPS workloads, real-world constraints, and optimization criteria, resulting in optimal assignments of processing units to tasks. The technique is demonstrated using a drone simulation under multiple physical constraints. The scheduling technique aims to optimize resource utilization and meet the objectives of CPS while considering the heterogeneity of the processing units.

3.3. Memory Access Scheduling:

This section discusses memory latency tolerance techniques and memory bank parallelism as part of optimizing deep learning computations on GPUs. These techniques aim to schedule memory accesses effectively to minimize the impact of memory latency and improve overall performance.

A memory scheduling strategy that addresses the challenges encountered in shared memory environments of heterogeneous multi-core systems is proposed by Fang et al. [2]. It proposes a step-by-step approach to memory scheduling, focusing on mitigating interference between memory access requests from different cores and improving system performance. The memory scheduling strategy consists of three key steps:

1. Request Source Isolation: To prevent interference between CPU and GPU memory requests, a new memory request queue is created based on the request source. This isolation ensures that GPU requests do not interfere with CPU requests and vice versa, enhancing overall memory access performance.

 doi:10.20944/preprints202412.0276.v2

2. Dynamic Bank Partitioning: For the CPU request queue, a dynamic bank partitioning strategy is implemented. It dynamically maps the CPU requests to different bank sets based on the memory characteristics of the applications. By considering the access behavior and characteristics of multiple parallel applications, this strategy eliminates memory request interference among CPU applications without affecting bank-level parallelism.
3. Criticality-Aware Scheduling: The GPU request queue incorporates the concept of criticality to measure the difference in memory access latency among GPU cores. A criticality-aware memory scheduling approach is implemented, prioritizing requests based on their criticality level. This strategy balances the locality and criticality of application access, effectively reducing memory access latency differences among GPU cores.

The MixTran scheduling technique, introduced by Zhang et al. [24], aims for efficient and fair resource allocation in heterogeneous GPU clusters handling mixed deep learning workloads. The authors address the challenge of existing schedulers not being tailored to deep learning jobs, resulting in low resource efficiency and job performance. MixTran abstracts heterogeneous GPU resources and distributes them fairly to users using virtual tickets. It then formulates a global optimization model to efficiently allocate resources based on quantified resource requests, heterogeneous node constraints, and user fairness constraints. The technique employs a greedy resource trading mechanism to benefit multiple users. Experimental results demonstrate that MixTran significantly reduces the total execution time of deep learning workloads (up to 30%–50%) compared to traditional schedulers while maintaining user fairness. The scheduling technique employed in MixTran can be categorized as a combination of fair-share scheduling and resource trading.

3.4. Prefetch and Co-design strategies

3.4.1. Context Switching Strategies

These strategies involve efficiently switching between different warps to enhance GPU resource utilization and improve performance. On the similar terms, SkyByte, explained in [52], improves scheduling by enabling coordinated context switching between the host OS and the SSD controller. When a memory request to the CXL-SSD experiences a long delay, the SSD controller signals the CPU using a long-delay hint, triggering a hardware exception. The OS scheduler then switches to another thread to utilize the CPU while waiting for the data.

3.4.2. Hardware and Algorithm Co-Design

This method involves tailoring scheduling strategies to specific hardware architectures. For instance, co-designing algorithms for GPUs can leverage their strengths in matrix operations and parallelism. Techniques like cache data residency optimization analyze data access patterns to enhance memory locality, reducing cache misses and improving execution speed. Likewise, the approach discussed in [53] enhances scheduling by dynamically adapting algorithm complexity based on resource availability and deadline constraints. By virtualizing hardware resources and optimizing parallelism, it enables real-time SLAM processing while maintaining high accuracy in resource-constrained environments. This adaptability allows the system to efficiently allocate computational resources, reducing latency spikes caused by dynamic workloads. As a result, it ensures consistent performance while balancing power consumption and processing efficiency.

3.4.3. Prefetching Mechanisms

This technique aims to reduce memory access latency and enhance overall performance by overlapping data movement with computation Yüzügüler et al. present a prefetching mechanism in [54] that optimizes the inference of large language models (LLMs) by overlapping memory reads for model weights and key-value (KV) cache with communication operations. This approach allows the system to hide the latency of memory accesses during collective communication phases. By

 doi:10.20944/preprints202412.0276.v2

leveraging a graph optimization framework, the mechanism automatically inserts prefetch commands into the computation graph, ensuring efficient data retrieval without requiring user code modifications. Overall, this strategy significantly enhances performance and reduces inference latency in distributed LLM systems.

3.4.4. Fine-Grained Warp Scheduling

This technique allocates and prioritizes the execution of threads within warps (groups of threads executed simultaneously on a single GPU core). Advanced warp scheduling policies aim to minimize idle cycles caused by data dependencies or resource contention. For instance, fine-grained warp scheduling dynamically prioritizes threads based on their readiness to execute, which improves parallelism and reduces pipeline stalls. This is crucial for maintaining high throughput in GPUs designed for massively parallel workloads. Fang et al. [2] discuss fine-grained warp scheduling techniques that focus on the execution order and prioritization of individual threads within a warp. This approach enhances parallelism and reduces resource contention, resulting in improved performance. Tang et al. introduce gCom in [55] as a fine-grained compressor designed for mobile GPUs that optimizes memory usage during rendering. It features fine-grained warp scheduling by treating color channels as processing units and quads (2x2 pixels) as parallel units, enhancing data locality and repetition rates. gCom employs a Channel Decorrelator to minimize raw data size and a Hierarchical Delta mechanism to maximize channel locality. Additionally, it introduces the Parallel-Oriented Golomb-Rice (POGR) algorithm, which allows parallel execution of compression and decompression processes, ensuring efficient GPU throughput while minimizing performance degradation.

3.5. Energy-Aware Job Scheduling Techniques

With rising concerns about energy efficiency, these techniques aim to minimize power consumption while maintaining performance. Dynamic voltage and frequency scaling (DVFS), for example, adjusts the processor's power states based on workload requirements. Additionally, heuristic algorithms like energy-aware successor tree scheduling optimize task execution sequences to minimize energy usage in heterogeneous systems. Tang et al. [3] discuss scheduling techniques aimed at optimizing energy consumption in heterogeneous computing systems. The main objective is to address the challenges posed by energy consumption, dynamic variability of CPU-GPU utilization, and job scheduling in large-scale systems. It proposes a CPU-GPU utilization aware and energy-efficient heuristic greedy strategy (UEJS) as well as a hybrid particle swarm optimization algorithm (H-PSO) to solve the job scheduling problem. The proposed UEJS algorithm incorporates the CPU-GPU utilization awareness and energy efficiency considerations to schedule jobs. To enhance the algorithm's global optimization ability, it introduces the H-PSO algorithm, which combines the heuristic greedy strategy with a bio-inspired search optimization technique. Experimental results demonstrate that the H-PSO algorithm outperforms the heuristic greedy strategy, Max-EAMin, and Genetic Algorithm in terms of average energy consumption of jobs and system job rejection rate. Specifically, H-PSO achieves a 36.5% improvement over UEJS, a 36.3% improvement over Max-EAMin, and a 46.7% improvement over GA in terms of job average energy consumption for heterogeneous systems with high workload.

Roeder et al. [5] highlight the significance of optimizing energy consumption in computing devices, especially in embedded systems. The authors propose an offline scheduling algorithm that aims to minimize overall energy consumption while ensuring timing constraints on heterogeneous platforms. The scheduling strategy is based on Forward List Scheduling and takes into account Dynamic Voltage and Frequency Scaling (DVFS). The algorithm selects the most energy-efficient version for each task, considering different energy/time trade-offs offered by architecture-specific binary blocks. It also enables applications to dynamically adapt voltage and frequency during runtime. Additionally, the proposed heuristic is compared against an optimal solution derived by an Integer Linear Programming (ILP) formulation, with a deviation of only 1.6% on average. Overall, [5]

 doi:10.20944/preprints202412.0276.v2

emphasizes the need for energy-aware scheduling strategies on heterogeneous real-time systems and presents a scheduling algorithm that effectively reduces energy consumption while meeting timing constraints. The proposed approach considers multi-version tasks, DVFS, and heterogeneous platforms, resulting in significant energy savings compared to existing scheduling algorithms.

Seo et al. [11] propose scheduling techniques to tackle the challenges encountered by edge platforms that process multiple machine learning (ML) models concurrently. These platforms, equipped with heterogeneous computing processors, have limited computational and energy budgets compared to data center servers. The proposed scheduler uses pre-profiled behavior of ML models and routes requests to the most suitable processors, considering the regularity of computation in common ML tasks. It aims to reduce energy consumption while meeting the service-level objective (SLO) requirement for bounded response latency. To handle the inflexible pre-emption capability of GPUs and DSPs, the scheduler decomposes large ML tasks into sub-tasks based on the layers of the DNN model. The scheduling policies are evaluated on an edge platform with CPU, GPU, and DSP. Results show significant performance improvements and reduced SLO violations compared to naive scheduling. By considering the heterogeneity of ML models and computing processors, the proposed techniques optimize performance, reduce energy consumption, and meet SLO requirements in edge computing environments.

With rising concerns about energy efficiency, these techniques aim to minimize power consumption while maintaining performance. Dynamic voltage and frequency scaling (DVFS), for example, adjusts the processor's power states based on workload requirements. Additionally, heuristic algorithms like energy-aware successor tree scheduling optimize task execution sequences to minimize energy usage in heterogeneous systems.

3.6. Memory Latency Tolerance

Techniques in this category aim to overlap computation with memory access to reduce the impact of memory latency on overall system performance. For example, prefetching mechanisms predict data requirements and fetch data into memory before it is needed by the processing units. Similarly, memory bank parallelism ensures concurrent access to multiple memory banks by optimizing data placement, thereby increasing throughput and reducing access contention. Wang et al. [4] discuss the challenges and propose solutions for enhancing the performance of integrated CPU/GPU platforms, particularly focusing on the latency associated with data initialization. The authors in this work highlight the importance of these platforms in autonomous driving and edge intelligence applications and the limitations posed by the shared physical memory. It introduces the concept of unified memory (UM) model in GPU programming, which simplifies memory management and allocation. However, the conventional copy-then-execute model used in UM programming leads to significant initialization latency and hampers kernel execution. To address this issue, the authors propose a framework that enables latency-aware data initialization. The framework includes three data initialization modes: CPU initialization, GPU initialization, and hybrid initialization. It also incorporates an affinity estimation model that considers application characteristics and platform features to determine the most suitable initialization mode. The goal is to optimize the initialization latency performance of the application.

3.7. Dynamic Task Scheduling

Zhang et al. [6] discuss task scheduling strategies for heterogeneous computing systems that utilize both CPU and GPU resources. It proposes two scheduling techniques: a load-aware strategy for the single task model and a genetic algorithm-based strategy for the multi-task model.

1. Load-Aware Scheduling Strategy:

This strategy focuses on addressing load balancing issues in CPU-GPU heterogeneous computing systems. It aims to allocate computing tasks to the CPU and GPU based on their computing power and a perception ratio. The strategy uses a bidirectional queue to store tasks, reducing additional overhead caused by scheduling. By assigning parallel computing tasks to both

 doi:10.20944/preprints202412.0276.v2

CPUs and GPUs, this strategy enhances performance, reduces load imbalances, and minimizes idle time.

2. Genetic Algorithm-Based Scheduling Strategy:

This strategy targets improving the overall operating efficiency of CPU-GPU heterogeneous computing systems in multi-task scenarios. It uses a genetic algorithm to determine the execution relationship between different types of tasks and heterogeneous processing cores. The strategy aims to optimize the allocation of computing resources and fully utilize the collaborative computing capabilities of CPUs and GPUs.

Experimental evaluations demonstrate that this strategy outperforms traditional scheduling algorithms, static and dynamic scheduling algorithms, and hybrid scheduling algorithms, resulting in higher system performance.

These scheduling techniques contribute to tapping into the performance potential of CPU-GPU heterogeneous computing systems, achieving load balance, reducing idle time, and improving system efficiency.

Yang et al. [18] propose a scheduling technique designed to minimize energy consumption in a CPU-GPU cloud computing platform while managing intermittent real-time tasks. The technique formulates the problem as an integer nonlinear programming problem and utilizes a dynamic programming approach inspired by the 0-1 knapsack problem. By dividing the hardware computing resource into virtual CPUs, the technique dynamically adjusts the task schedule based on the solution. Experimental results demonstrate that the proposed algorithm effectively reduces energy consumption and outperforms other greedy methods in terms of computation time. The scheduling technique contributes to optimizing system performance in cloud computing environments.

Hsu et al. [26] introduce a programming and execution model known as SHMT (Simultaneous and Heterogeneous Multithreading), which facilitates parallel processing with heterogeneous processing units. The paper focuses on the scheduling technique used in SHMT to coordinate the execution on different hardware components efficiently. SHMT introduces a low-overhead scheduling policy that considers both results and performance. It utilizes a set of virtual operations (VOPs) and High-Level Operations (HLOPs) as an intermediate layer between programming languages and hardware instructions. This intermediate layer facilitates task matching and distribution, allowing SHMT to divide equivalent operations and data on different computing resources. The scheduling technique employed in SHMT can be categorized as a dynamic scheduling policy. It dynamically adjusts workloads on various hardware units to maximize hardware efficiency while providing flexibility in scheduling policies. By effectively coordinating the execution on heterogeneous hardware, SHMT achieves significant speedup and energy reduction compared to GPU baseline, as demonstrated in evaluations on an embedded system platform. Overall, the paper emphasizes the importance of scheduling techniques in enabling parallel execution across heterogeneous processing units, and SHMT's dynamic scheduling policy plays a crucial role in achieving efficient utilization of hardware resources.

Zhou et al. [37] propose a resource scheduling system named FILL, aimed at enhancing the performance of GROMACS simulations through the effective utilization of heterogeneous hardware resources. The scheduling technique used in FILL is space partitioning technology, which allows for precise allocation of resources between CPU and GPU devices.The article highlights that previous research focused mainly on co-running multiple GROMACS simulations using time-slice technology, which introduced context-switching overhead and neglected collaborative scheduling of CPU and GPU devices. FILL addresses this limitation by leveraging hardware partitioning technologies and optimizing the allocation of resources for multiple GROMACS jobs. Experimental results demonstrate significant improvements in system throughput using FILL on both NVIDIA and AMD GPU servers compared to baseline approaches and state-of-the-art alternatives. FILL achieved an impressive improvement of up to 167% on NVIDIA GPU servers and demonstrated significant enhancements of 459% on AMD GPU servers. Overall, FILL's space partitioning-based scheduling technique enhances resource utilization and improves system throughput for multiple GROMACS

 doi:10.20944/preprints202412.0276.v2

simulations. The FILL mechanism, as shown in Figure 2, effectively addresses the challenge of resource utilization in heterogeneous computing by employing space partitioning technology to optimize CPU and GPU allocation for GROMACS simulations

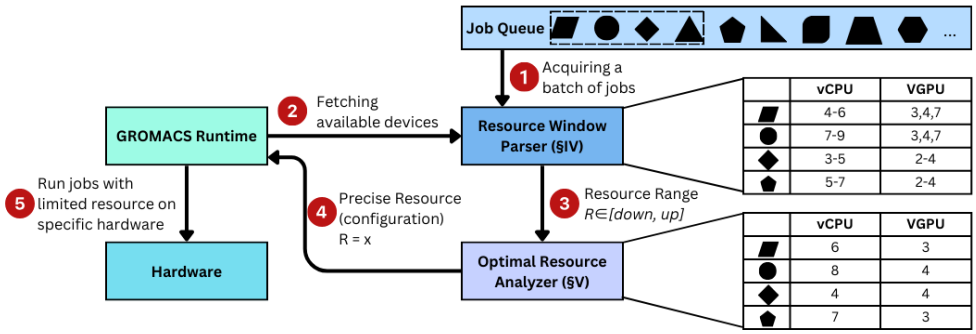

Figure 2. FILL mechanism presented in [37].

Wang et al. [39] introduce a novel scheduling technique named GPARS, designed for efficient resource allocation in heterogeneous graphics processing unit (GPU) clusters. The scheduling technique leverages spatiotemporal correlations among jobs and utilizes graph attention networks (GANs) for precise job duration prediction. The GPARS algorithm addresses the limitations of prior research by considering the performance variations among different GPU types within heterogeneous clusters and the presence of spatiotemporal correlations among jobs. It predicts job durations using GANs and dynamically allocates suitable GPU types for newly submitted jobs based on the prediction results. The effectiveness of GPARS is evaluated using real traces from Alibaba and Philly, demonstrating a significant reduction in waiting time (10.29%) and an average improvement in resource utilization (7.47%) compared to the original scheduling method. Overall, GPARS is a prediction-based scheduling technique that efficiently schedules resources in heterogeneous GPU clusters, considering the unique characteristics and variations among different GPU types.

Ndamlabin et al. [45] introduce a scheduling technique called RSCHED, aimed at managing the concurrent execution of task-based applications in heterogeneous computing environments. The authors address the challenge of efficiently utilizing modern parallel architectures with complex memory hierarchies and heterogeneous processors. They propose RSCHED as a framework to minimize overall makespan and maximize resource utilization. The scheduling technique aims to improve the execution of task-based applications by dynamically distributing resources and orchestrating their execution. The authors implemented RSCHED using the StarPU runtime system and evaluated its performance on real applications. The results showed that RSCHED significantly reduced the overall makespan compared to consecutive execution, with an average speedup factor of 10x. It also demonstrated the potential to increase resource utilization. RSCHED is a dynamic resource allocation strategy that enhances the scheduling of task-based applications on heterogeneous systems.

3.8. Adaptive Scheduling

Adaptive scheduling dynamically modifies task priorities, resource allocations, or scheduling policies in response to changing workloads or system states. For example, temperature-aware scheduling ensures that tasks are allocated in a way that prevents overheating in edge devices, prolonging hardware lifespan and maintaining stability under thermal constraints. Constantinescu et al. [8] discuss efficiency and productivity in decision-making processes involving low-power heterogeneous CPU+GPU SoCs. It specifically focuses on the evaluation of scheduling strategies for executing value iteration, a core procedure in decision-making methods, on a low-power CPU+GPU SoC. It compares off-line-tuned static and dynamic scheduling strategies with adaptive

 doi:10.20944/preprints202412.0276.v2

heterogeneous scheduling strategies. The experiments show that using CPU+GPU heterogeneous strategies significantly reduce computation time and energy requirements. The CPU+GPU strategies can be up to 54% faster and 57% more energy-efficient compared to multicore or GPU-only implementations. Additionally, it explores the impact of increasing the abstraction level of the programming model to ease programming efforts. The comparison between TBB+OpenCL and TBB+oneAPI implementations of heterogeneous schedulers shows that the oneAPI versions result in up to 5 times less programming effort with only 3-8% overhead if the scheduling strategy is selected carefully.

The discussed strategies in [8] are:

1. Off-line-Tuned Static and Dynamic Scheduling Strategies:

This study evaluates off-line-tuned static and dynamic scheduling strategies for executing value iteration, a core procedure in decision-making methods, on a low-power CPU+GPU System-on-Chip (SoC). The comparison is made between different scheduling strategies to determine their impact on computation time and energy requirements.

2. Adaptive Heterogeneous Scheduling Strategies:

The study also explores adaptive heterogeneous scheduling strategies for decision-making on a low-power CPU+GPU SoC. These strategies aim to dynamically allocate tasks between the CPU and GPU, taking advantage of their respective capabilities. The experiments demonstrate that using CPU+GPU heterogeneous strategies significantly reduce computation time and energy requirements compared to multicore or GPU-only implementations.

3. Increasing Abstraction Level of the Programming Model:

The article explores the impact of increasing the abstraction level of the programming model to ease programming efforts. Specifically, it compares the TBB+OpenCL and TBB+oneAPI implementations of heterogeneous schedulers. The results show that the oneAPI versions can result in up to 5 times less programming effort with only 3-8% overhead if the scheduling strategy is selected carefully.

Albahar et al. [16] propose a scheduling technique named SCHEDTUNE, designed to tackle the challenges of managing and scheduling heterogeneous GPU resources in cluster management systems such as Kubernetes. The existing resource schedulers in these systems do not effectively differentiate between different types of GPUs or support GPU sharing, resulting in low GPU utilization, queuing delays, and increased application makespan. SCHEDTUNE is a machine-learning-based scheduler that aims to improve GPU memory utilization, reduce out-of-memory (OOM) failures, and enhance overall makespan. It achieves this by profiling and analyzing deep learning (DL) jobs on heterogeneous GPUs to understand interference caused by collocating jobs and predict GPU memory demand and job completion times. By leveraging this information, SCHEDTUNE optimizes resource allocation and GPU sharing, resulting in higher GPU utilization and improved performance compared to the default Kubernetes scheduler. The proposed technique fills the gap between the capabilities of container orchestrators and the complex requirements of DL applications, providing a management-level solution that learns system behavior and efficiently manages GPU resources in a heterogeneity-aware manner.

He et al. [17] present a CPU-GPU heterogeneous computation offloading and resource allocation scheme tailored for the Industrial Internet of Things (IIoT). With the increasing complexity of IIoT tasks and the demand for delay-sensitive and computing-intensive applications, heterogeneous platforms integrating CPUs and GPUs have become essential. However, the three-stage heterogeneous computing process, including task preprocessing, hybrid computing, and result aggregation, poses challenges for task offloading and resource allocation. The proposed scheme introduces a three-stage heterogeneous computing model and formulates the joint task offloading and heterogeneous resource allocation problem. The authors employ the Lyapunov optimization method and propose the multihead proximal policy optimization (MH-PPO)-based algorithm to

 doi:10.20944/preprints202412.0276.v2

minimize the average task delay. Simulation results demonstrate the effectiveness of the scheme in reducing task delays. The study contributes to the development of efficient scheduling techniques in CPU–GPU heterogeneous computing environments for IoT applications. Figure 3 depicts the three-stage heterogeneous computing model designed for task offloading and resource allocation in Industrial Internet of Things (IIoT) systems, demonstrating the sequential steps of preprocessing, hybrid computing, and result aggregation.

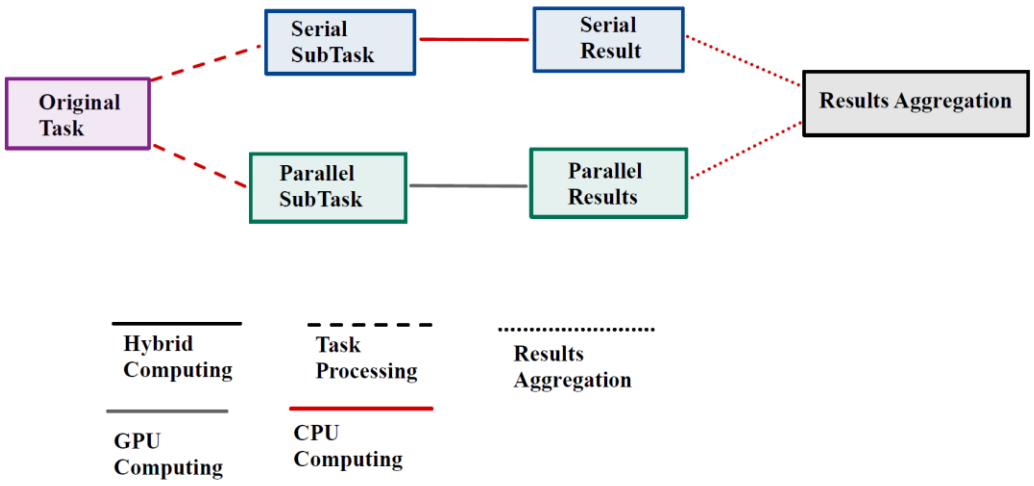

Figure 3. Three stage Heterogeneous Computing Model depicted in [17].

Sharma et al. [40] propose a novel scheduling technique specifically designed for distributed deep learning (DDL) workloads in GPU clusters. The primary focus of the scheduling technique is to minimize communication overhead and reduce training times by consolidating jobs on physically close GPUs. The scheduling technique consists of three major components: a classical delay scheduling algorithm for job placement and consolidation, a network-sensitive job preemption strategy, and an auto-tuner mechanism to optimize delay timers. By considering the anticipated communication-network delays and the sensitivities of DDL jobs to these delays, the scheduler intelligently places and consolidates jobs based on their network requirements. The proposed technique takes into account the performance characteristics of different network tiers, leveraging modern networking hardware advancements. It dynamically adjusts consolidation based on the network sensitivity of individual jobs, aiming to minimize queueing delays and overall training times. The evaluation results demonstrate significant improvements in end-to-end makespan, job completion time, and communication overhead compared to existing consolidation-based scheduling methods. It presents a network-sensitive scheduling technique that optimizes GPU cluster scheduling for DDL workloads, considering communication overhead and job consolidation based on network sensitivities. Scheduling techniques tailored for distributed deep learning workloads are summarized in Figure 4, emphasizing strategies that minimize communication overhead and improve job placement within GPU clusters

 doi:10.20944/preprints202412.0276.v2

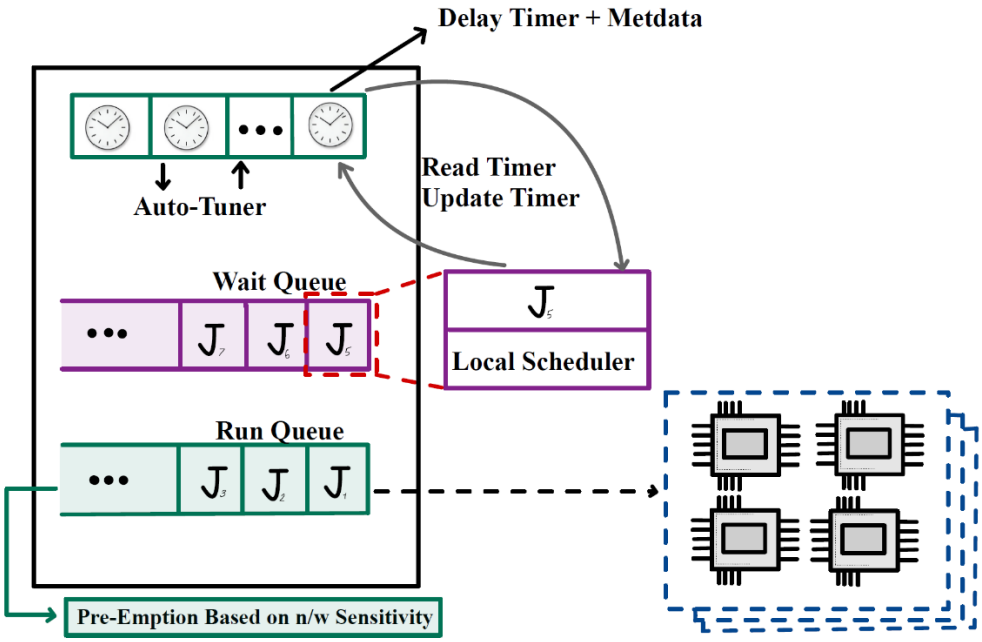

Figure 4. Scheduling Techniques proposed in [40].

3.9. Task Graph Scheduling

Li et al. [9] discuss scheduling techniques aimed at efficient task mapping on heterogeneous CPU/GPU platforms. The paper presents a theoretical framework and three practical mapping algorithms with low time complexity to minimize completion time in GPU-based embedded systems. The challenges in task mapping arise from the large size of the policy space and the need to consider factors such as performance characteristics, task dependencies, and data transfer costs. The proposed mapping algorithms address these challenges and outperform state-of-the-art techniques. Experimental results demonstrate up to 30% faster completion time across different workloads. This work in [9] extends the algorithms to enhance runtime performance in resource-limited infrastructure and evaluates them using a benchmark testing suite for simulating real-world runtime neural networks. The extended algorithm achieves significantly faster completion time (averaging 30% to 37% improvement) compared to existing techniques. Figure 5 presents the scheduler application framework designed to enhance task mapping on heterogeneous CPU/GPU platforms, ensuring efficient resource utilization and reduced completion times.

 doi:10.20944/preprints202412.0276.v2

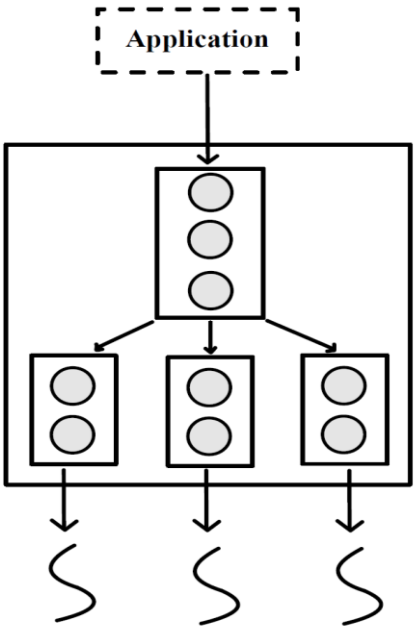

Figure 5. Scheduler Application prsented in [9].

Harichane et al. [12] focus on scheduling techniques in heterogeneous computing environments, particularly in the context of Kubernetes. The paper addresses the challenge of efficiently utilizing multiple CPUs and GPUs in cloud-based deployments by proposing the KubeSC-RTP scheduler. KubeSC-RTP stands for "Kubernetes Scheduler based on RunTime Prediction" and employs machine learning algorithms for runtime estimation of deployed applications. By accurately predicting the execution time of applications, the scheduler aims to optimize resource consumption and improve overall system performance. The article outlines the steps involved in implementing the KubeSC-RTP scheduler and highlights its intelligent scheduling approach based on machine learning techniques. It emphasizes the importance of selecting the appropriate device (CPU or GPU) for each application to minimize execution time and maximize resource utilization. The proposed scheduling technique offers potential benefits for cloud service providers, including reduced processing time, increased customer satisfaction, and improved resource management. The article concludes by discussing the experimental results and providing insights into the strengths and limitations of the KubeSC-RTP scheduler, as well as suggesting avenues for future research and enhancements in this field.

The FastGR framework in [18] employs a heterogeneous task graph scheduler as its scheduling technique. This scheduler efficiently distributes tasks between the CPU and GPU components of the system, ensuring workload balancing and resource utilization optimization. By leveraging the processing power of both CPU and GPU, FastGR achieves significant improvements in performance and solution quality for global routing. The task graph scheduler plays a crucial role in coordinating the execution of tasks, managing dependencies, and allocating resources effectively. It dynamically assigns tasks to the most suitable processing unit, taking into account the computational capabilities and workload distribution across the system. This approach enables FastGR to harness the massive parallelism offered by GPUs while effectively utilizing the CPU's capabilities. Through balanced task scheduling, FastGR maximizes the utilization of available resources, minimizes idle time, and enhances the overall efficiency of the global routing process. The overall workflow of the FastGR global routing framework is shown in Figure 6, highlighting the role of a heterogeneous task graph scheduler in balancing workloads across CPU and GPU components.

 doi:10.20944/preprints202412.0276.v2

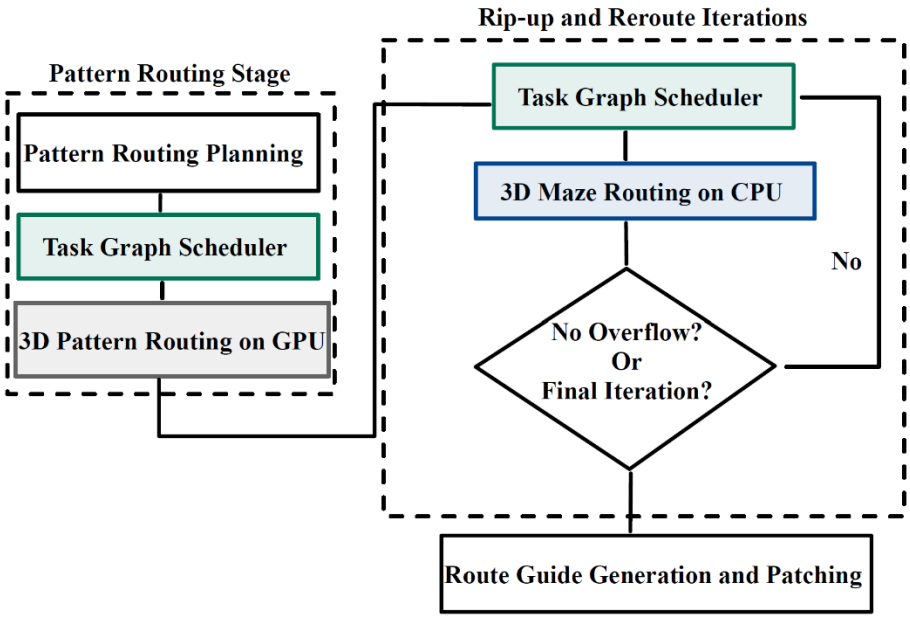

Figure 6. Overall flow of Fast GR presented in [18].

Gao et al. [27] propose a temperature-aware scheduling technique for heterogeneous computing in edge scenarios. The authors address the challenge of jointly processing inference tasks using CPU, GPU, and NPU while considering the impact of ambient temperature on edge devices. They establish a temperature perception model based on the ambient temperature and introduce a TAS (temperature-aware schedule) algorithm to control the running speed of the heterogeneous device. Additionally, they propose a task scheduling algorithm called TASTS (TAS-based task schedule) and utilize a Hungarian matching algorithm to optimize the final results. The article demonstrates that the proposed technique improves performance by 20-50% compared to conventional methods under temperature constraints. The scheduling technique used in this study is the TAS (temperature-aware schedule) algorithm.

Wang et al. [38] propose a GPU scheduling technique known as GCAPS, designed for real-time GPU task execution. The scheduling technique addresses the challenges of prioritization and preemption in GPU tasks, aiming to ensure timely execution and meet stringent timing requirements. GCAPS operates at the device driver level and enables control over GPU context scheduling by adding one-line macros to GPU segment boundaries. It allows preemption of GPU execution based on task priorities, improving schedulability and response time. The approach requires minimal modifications to the user-level GPU access code and provides fine-grained and efficient control of the GPU. Through empirical evaluations, the proposed approach demonstrates significant improvements over prior work, achieving up to 40% higher schedulability. GCAPS utilizes a preemptive scheduling technique for GPU tasks in real-time systems.

Verma et al. [47] provide a comprehensive overview of scheduling techniques aimed at optimizing energy consumption in cloud computing. The authors discuss state-of-the-art algorithms for scheduling workflow tasks to cloud resources, with a specific focus on reducing energy consumption. The article categorizes different workflow scheduling algorithms based on the scheduling approaches used and provides an analytical discussion of the covered algorithms. Additionally, the authors classify various energy-efficient strategies employed by cloud service providers (CSPs) for energy saving in data centers. The article also highlights popular real-world workflow applications and identifies emerging trends and open issues in cloud computing for future research directions.

Fang et al. [48] propose a scheduling technique designed to alleviate resource contention in heterogeneous systems. The article addresses the challenge of shared resource utilization between CPUs and GPUs in network-on-chip (NoC) architectures. It introduces the concept of LLC/MC

 doi:10.20944/preprints202412.0276.v2

CENTER architecture and analyzes the impact of different placement methods on system performance. To optimize the network's performance, the article presents a task-based routing algorithm that plans the path based on different tasks and a Task-Based-Partition (TBP) routing algorithm that allocates routing algorithms of different tasks into separate virtual channels. The proposed scheduling technique aims to improve system performance by enhancing resource allocation and reducing network latency. Overall, the article focuses on task-based scheduling techniques for managing resources in heterogeneous CPU-GPU architectures within a network-on-chip framework.

The framework allows a computational kernel to span across multiple devices on a node and enables multiple applications to be scheduled on the same node. It dynamically migrates kernels between devices, expands or contracts the kernel to utilize more or fewer devices, and optimizes scheduling decisions based on different objectives such as job throughput and job priorities. The authors evaluate the framework on a CPU+GPU+FPGA platform and demonstrate speedups of 2.26X over different applications and up to 1.25X for co-scheduled workloads over baselines. The major contribution lies in the ease of programmability with a single code base across different execution devices. Overall, the proposed framework provides an efficient scheduling technique for maximizing the utilization of heterogeneous devices in cloud and HPC environments.

3.10. Hybrid Scheduling

Combining static and dynamic strategies, hybrid scheduling offers flexibility and efficiency across diverse workloads. For example, a hybrid approach might predefine high-priority tasks while dynamically adjusting the execution order of lower-priority jobs based on real-time resource availability. Fang et al. [14] focus on addressing the challenges encountered in heterogeneous systems comprising CPUs and GPUs. The authors propose resource scheduling strategies to improve the utilization and efficiency of heterogeneous cores, thereby optimizing system performance. Two main strategies are presented: a combination strategy for single-task scheduling and a multi-task scheduling strategy. The combination strategy involves optimizing task execution efficiency on GPUs by modifying thread organization structures and developing workload balancing schemes for efficient core utilization. The multi-task scheduling strategy utilizes task samples to gather information about the execution efficiency of heterogeneous cores and global task information. An improved ant colony algorithm is then employed to quickly allocate tasks to the most suitable cores, taking advantage of the characteristics of heterogeneous cores. Experimental results demonstrate that the combination strategy reduces task execution time by an average of 29.13%, while the multi-task scheduling strategy reduces execution time by up to 23.38% compared to the combined strategy. These strategies effectively utilize the resources of heterogeneous systems and significantly improve task execution times.

Yan et al. [42] present a task scheduling technique called HSAS, which optimizes the processing efficiency of deep neural network models on heterogeneous systolic array accelerator clusters. The authors address the challenge of significant differences among models and layers by proposing a heterogeneous architecture that consists of systolic arrays with different scales. HSAS considers factors such as task priority, prediction, preemption, load balance, and layer-level scheduling. The scheduling technique employs a task decomposition algorithm and a subtask priority management table to enable fine-grained subtask-level scheduling. The authors validate the performance and energy models for systolic arrays and demonstrate their accuracy. Experimental results show that HSAS outperforms classic and state-of-the-art methods in terms of average normalized turnaround time, system throughput, and fairness, achieving improvements of over 80% with task-level scheduling and 18% to 63% with subtask-level scheduling. HSAS is a sophisticated scheduling technique that optimizes the utilization of heterogeneous systolic array architectures for processing deep neural network models, leading to significant performance improvements.

Mohammad Jafari et al. [44] provide a thorough analysis of task scheduling techniques in heterogeneous computing environments. The study evaluates the performance of four scheduling

 doi:10.20944/preprints202412.0276.v2

algorithms: First-Come, First-Served (FCFS), FCFS with No Queuing (FCFS-NQ), Minimum Expected Completion Time (MECT), and Minimum Expected Execution Time (MEET). The goal is to optimize resource utilization and minimize task completion times. Three workload scenarios representing different computational demands are considered: low, medium, and high. Through rigorous experimentation, the authors assess the effectiveness of each algorithm in terms of total completion percentage, energy consumption, wasted energy, and energy per completion. The findings highlight the strengths and limitations of each algorithm. MECT and MEET emerge as strong contenders, dynamically prioritizing tasks based on comprehensive estimates of completion and execution times. These algorithms exhibit superior energy efficiency compared to FCFS and FCFS-NQ, making them suitable for resource-constrained environments. The study provides valuable insights into task scheduling algorithms, enabling informed decision-making for enhancing resource allocation, minimizing task completion times, and improving energy efficiency. CPU scheduling techniques are visually summarized in Figure 7, providing insights into strategies for optimizing task execution in resource-constrained environments.

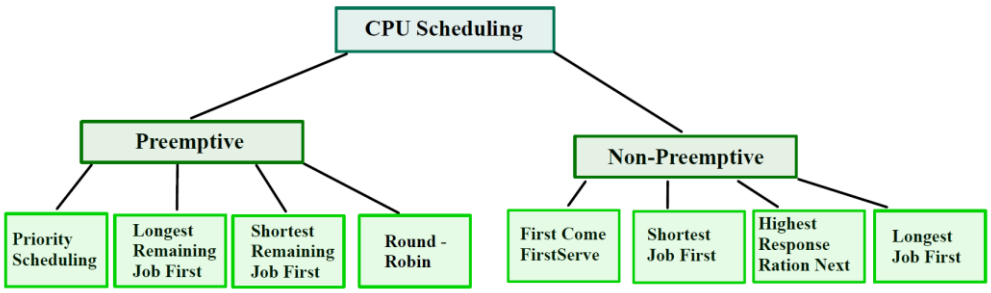

Figure 7. Different CPU Scheduling presented in [44].

3.11. Pipelined Scheduling

This strategy overlaps different stages of task execution to maximize system throughput. By decomposing tasks into smaller stages that can execute concurrently on different processing units (e.g., CPUs and GPUs), pipelined scheduling minimizes idle time and maximizes hardware utilization. Garcia-Hernandez et al. [19] introduce a platform that simplifies the implementation of computationally demanding algorithms in heterogeneous computing environments. The platform utilizes a scheduling technique to distribute tasks across different computing devices, including CPUs, GPUs, and FPGAs, in order to achieve algorithm acceleration. By leveraging the OpenCL framework, the platform offers generality and flexibility, allowing non-experts in heterogeneous computing to deploy and run algorithms without extensive knowledge of specific implementation frameworks. The platform automatically generates parallel code by analyzing algorithms implemented in C, adapting it to target the available computing devices. Experimental results using the Polybench/C suite demonstrate the effectiveness of the proposed platform, achieving accelerations of up to 270× for parallel-friendly algorithms. The scheduling technique used in the platform is not explicitly mentioned in the provided summary.

Liu et al. [20] present a distributed framework known as HeterPS, which utilizes a reinforcement learning (RL)-based scheduling method to optimize the training process of deep neural network (DNN) models in heterogeneous computing environments. The scheduling technique used in HeterPS is based on RL and utilizes an LSTM model. The framework addresses the challenge of effectively utilizing diverse computing resources, such as CPUs and GPUs, for training large-scale DNN models. HeterPS schedules each layer of the DNN model to the appropriate computing resource to minimize costs and satisfy throughput constraints. It also manages data storage and communication among distributed computing resources. Experimental results demonstrate that HeterPS outperforms existing approaches in terms of throughput, achieving a 14.5 times higher

 doi:10.20944/preprints202412.0276.v2

throughput, and monetary cost, resulting in a 312.3% reduction in cost. Figure 8 outlines the HeterPS framework, showcasing its reinforcement learning-based scheduling approach to optimize deep learning model training across diverse computing resources.

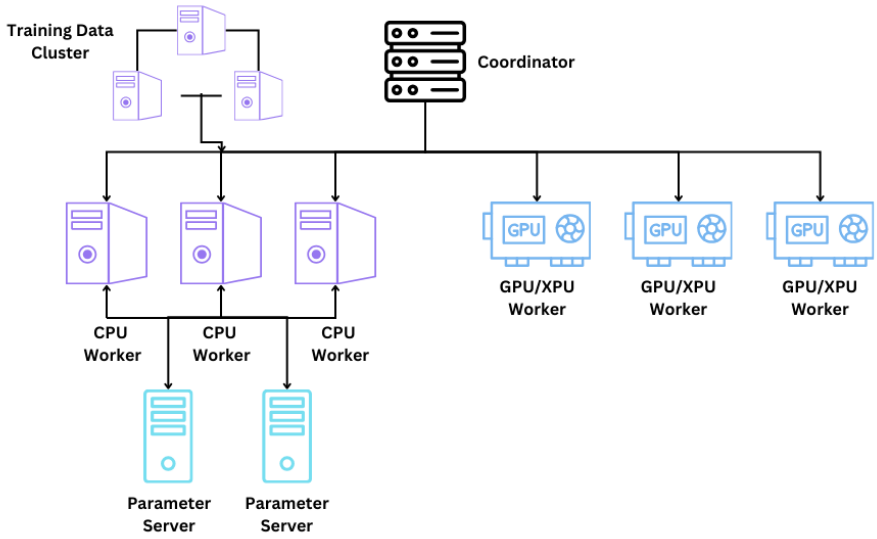

Figure 8. HeterPS framework presented in [20].

Zou et al. [25] introduce a scheduling technique called RTGPU for efficiently scheduling multiple GPU applications with hard real-time guarantees. The article addresses the challenge of scheduling highly parallel applications on graphics processing units (GPUs) while meeting stringent timing constraints. RTGPU combines fine-grain GPU partitioning with a novel scheduling algorithm based on federated scheduling and grid search with uniprocessor fixed-priority scheduling. The proposed technique leverages persistent threads and interleaved execution to improve GPU partitioning and performance. It provides real-time guarantees to meet hard deadlines and achieves significant improvements in system throughput (over 11%) and schedulability (up to 57%) compared to previous approaches. The RTGPU scheduling technique is validated and evaluated on NVIDIA GPU systems. It can be applied to mainstream GPUs and general heterogeneous computing platforms with similar task execution patterns. This technique contributes to enabling the efficient execution of computation-intensive parallel tasks on GPUs within real-time constraints.

Xue et al. [32] propose a performance model for estimating the performance of heterogeneous implementations and providing guidance for computer architecture design. They apply the CPU-GPU heterogeneous computing to a CFD test case and compare its performance with a pure GPU implementation. By assigning proper workload ratios to the CPU and GPU workers, the heterogeneous version outperforms the pure GPU version by up to 23%. The authors leverage MPI (Message Passing Interface) and OpenACC to facilitate the CPU-GPU heterogeneous computing workflow in the CFD solver. These frameworks are commonly used for parallel computing and task distribution in heterogeneous systems.

3.12. Power Aware/Deadline Aware Scheduling

Yang et al. [23] introduce Hydra, a scheduling technique designed for deep learning (DL) jobs running on heterogeneous GPUs. The primary goal of Hydra is to efficiently schedule DL jobs while considering deadline requirements and reducing job completion time (JCT). Existing approaches focusing on efficiency or deadline requirements alone are inadequate for this task. Hydra introduces a novel quantitative cost comparison approach that incorporates total JCT and a dynamic penalty based on tardiness, i.e., the delay in meeting deadlines. It leverages a sampling approach to estimate job execution times accurately on heterogeneous GPUs and utilizes an efficient branch-and-bound

 doi:10.20944/preprints202412.0276.v2

algorithm to find the optimal job-GPU combinations. Evaluation experiments using Alibaba traces demonstrate that Hydra significantly reduces total tardiness by 85.8% while minimizing JCT compared to state-of-the-art efforts. The technique used in Hydra can be categorized as deadline-aware scheduling for deep learning jobs on heterogeneous GPUs.

Lannurien et al. [41] introduce a scheduling technique aimed at optimizing the deployment of time-sensitive applications, specifically focusing on Intrusion Detection Systems (IDS) in a serverless edge computing environment. The proposed technique addresses the challenges of reducing initialization delays, minimizing communication delays, and leveraging heterogeneous resources to satisfy variable Quality of Service (QoS) requirements. The scheduling technique incorporates a storage-aware allocation and scheduling policy that aims to minimize task placement costs for service providers while optimizing QoS for IDS users. It includes a caching and consolidation strategy to minimize cold starts and inter-function communication delays. By leveraging the capabilities of heterogeneous edge resources, the strategy achieves better QoS performance and reduces the number of edge nodes required for application deployment. The article presents a simulation-based evaluation comparing the proposed technique with a vanilla Knative orchestrator and a storage-agnostic policy. The results demonstrate that the proposed strategy achieves 18% fewer QoS penalties and consolidates applications across 80% fewer edge nodes. Overall, the scheduling technique contributes to cost-effective deployment of IDS and other time-sensitive applications on unreserved edge resources, leveraging serverless computing paradigms.

Shobaki et al. [43] present a novel approach to instruction scheduling for GPUs using a parallelized Ant Colony Optimization (ACO) algorithm. The authors address the register-pressure-aware instruction scheduling problem, which involves optimizing the balance between schedule length and register pressure on a GPU target. They demonstrate that parallelizing the ACO algorithm on the GPU significantly improves scheduling performance compared to sequential CPU-based scheduling. The ACO algorithm, inspired by nature, utilizes a pheromone-based search technique to find optimal instruction orders. The authors propose several techniques to efficiently parallelize the ACO algorithm for the complex multi-objective optimization problem of register-pressure-aware scheduling on the GPU. Experimental results show that parallel ACO-based scheduling on the GPU achieves up to 27 times faster execution compared to sequential CPU-based scheduling, resulting in a 21% reduction in total compile time and up to 74% improvement in execution speed compared to AMD's production scheduler.

3.13. Batch Job Scheduling

Batch scheduling groups multiple tasks for execution, optimizing resource allocation and minimizing overhead. Composable schedules, for instance, allow for modular execution of subtasks, enabling efficient handling of large-scale workloads in data centers and accelerators. Chen et al. [29] focus on a programming model, Allo, for enhancing the efficiency of spatial accelerator architectures. One of the key aspects of Allo is its scheduling technique, which enables the construction of modular hardware accelerators through the composition of customized kernels and external IPs. The scheduling technique used in Allo is called composable schedules. This technique allows users to incrementally add customizations to kernels while validating the correctness of each submodule. Multiple schedules are progressively integrated into a complete design using the .compose() primitive. The composable schedules approach enhances productivity and debuggability, enabling the creation of high-performance designs. Allo also introduces a hierarchical dataflow graph to support the composition of multiple kernels within a complex design while maintaining function boundaries. By modeling the interface unification problem as a type inference problem and leveraging the hierarchical dataflow graph, Allo optimizes the scheduling of dataflow operations. Overall, Allo's composable scheduling technique provides a flexible and modular approach to designing high-performance spatial accelerators.

Ahmad et al. [30] analyze scheduling algorithms utilized in heterogeneous computing systems. The article reviews various list-based workflow scheduling algorithms over the past two decades and

 doi:10.20944/preprints202412.0276.v2

categorizes them based on their scheduling objectives. The main scheduling technique discussed in the article is list-based scheduling, specifically list scheduling with static priorities (LSSP). List-based scheduling algorithms are known for their efficiency in generating schedules for complex workflow applications in heterogeneous computing environments. These algorithms aim to minimize makespan, energy consumption, and maximize resource utilization and reliability. The article compares different list-based scheduling algorithms based on their objectives, merits, comparison metrics, workload type, experimental scale, experimental environment, and results. Additionally, the article conducts experimental analysis of seven state-of-the-art algorithms on randomly generated workflows to understand their working.

Belkhiri et al. [31] explore the challenges and advancements in GPU virtualization, with a specific focus on the scheduling technique employed in Intel's Graphics Virtualization Technology—Grid generation (GVT-g). The authors present a novel performance analysis framework that utilizes host-based tracing combined with the Linux Trace Toolkit Next Generation (LTTng) to collect performance data efficiently and with minimal overhead. The scheduling technique employed in GVT-g allows for the allocation and sharing of GPU resources among virtual machines. The framework incorporates a unified, stateful model for filtering and organizing trace data, enabling the computation of various performance metrics. The analysis is performed using Trace Compass, an open-source performance analyzer, which provides synchronized graphical views to aid in understanding GVT-g's internal mechanisms and diagnosing performance issues related to virtual GPU usage. This article provides valuable insights into GPU virtualization and offers a practical approach to performance analysis in virtualized environments using the GVT-g scheduling technique.

Ye et al. [33] provide a comprehensive overview of scheduling techniques for deep learning (DL) workloads in GPU datacenters. The article emphasizes the importance of efficient scheduling to reduce operational costs and improve resource utilization in DL model development. Traditional scheduling approaches designed for big data or high-performance computing workloads are inadequate for fully utilizing GPU resources in DL workloads. The survey examines existing research efforts for both training and inference workloads, categorizing them based on scheduling objectives and resource utilization. It discusses the challenges in designing satisfactory schedulers for DL workloads and identifies the common strategies employed by existing solutions.

Tariq et al. [35] present a scheduling technique known as Energy Efficient Successor Tree Consistent Earliest Deadline First (EESEDF) for Periodic Conditional Task Graphs (PCTGs) on Multiprocessor System-on-Chips (MPSoCs) with shared memory. The goal is to minimize energy usage by considering dynamic and static power models. The EESEDF approach maximizes the worst-case processor utilization by assigning tasks to processors and arranging them using the earliest successor tree consistent deadline-first strategy. To further minimize energy consumption, the technique solves a convex Non-Linear Program (NLP) to determine optimal task speeds. Additionally, an online Dynamic Voltage Scaling (DVS) heuristic is introduced, which dynamically adjusts task speeds in real-time. Experimental results demonstrate that EESEDF+Online-DVS outperforms existing techniques, achieving notable energy efficiency improvements over LESA and NCM. The proposed scheduling technique achieves significant energy efficiency gains compared to IOETCS-Heuristic, BESS, and CAP-Online. In summary, the article focuses on the EESEDF scheduling technique, which combines successor tree consistency, earliest deadline first strategy, and dynamic voltage scaling to optimize energy usage in scheduling PCTGs on MPSoCs with shared memory.

Kwon et al. [36] address the scheduling challenges encountered in multi-tenant cloud systems with heterogeneous GPUs while running various machine learning workloads. The authors propose a novel scheduling approach that utilizes a genetic optimization technique implemented within a process-oriented discrete-event simulation framework. The scheduling technique employed in the study is a genetic optimization technique, which involves using genetic algorithms to find optimal solutions for scheduling machine learning tasks on GPUs. The approach aims to improve GPU

 doi:10.20944/preprints202412.0276.v2

utilization in complex environments by effectively orchestrating the scheduling of machine learning workloads. Through extensive simulations using workload traces from Alibaba's MLaaS cluster, which consists of over 6000 heterogeneous GPUs, the proposed scheduling approach demonstrates a 12.8% improvement in GPU utilization compared to Round-Robin scheduling. The results highlight the effectiveness of the genetic optimization technique in optimizing GPU scheduling in cloud-based environments. It focuses on enhancing the scheduling of AI applications in multi-tenant cloud systems by employing a genetic optimization technique, which improves GPU utilization and optimizes resource allocation for machine learning workloads. Table 1 recaps the significant features of various scheduling techniques discussed in this survey focussing on their advanatges and challenges.

Table 1. Significant features of various Scheduling Techniques.

Paper	Scheduling Technique	Description	Focus	Flexibility	Advantages	Challenges
[55]	Fine-grained warp scheduling	prioritizes threads	Thread execution optimization	High	Enhanced parallelism, reduced resource contention	Increased complexity, potential overhead
[4]	Memory latency tolerance	Overlaps memory access with computation	Memory access optimization	Medium	Reduced memory latency impact	Complexity in implementation
[2, 24]	Memory bank parallelism	Boosts throughput	Memory throughput optimization	Medium	Improved memory access patterns	Potential conflicts in memory banks
[52]	Context switching	switches between warps	Resource utilization optimization	High	Reduced idle times, better utilization	Overhead and potential synchronization issues
[54]	Prefetching mechanisms	Anticipates and fetches data in advance	Data access optimization	Low	Reduced latency, improved data availability	Prediction overhead, redundancy risk
[7, 10]	Dynamic load balancing	Redistributes workload	Workload distribution	High	Improved responsiveness, balanced resources.	Potential runtime overhead
[6, 39]	Dynamic task scheduling	Real-time adaptation of task execution	Task execution optimization	High	Adaptability to workload changes	Higher runtime complexity
[16, 17]	Adaptive scheduling	Adjusts policies dynamically	Workload adaptation	High	Better resource utilization	May incur frequent adjustments
[14, 42]	Hybrid scheduling	Combines static and dynamic approaches	Mixed scheduling strategies	Medium	Flexibility, efficiency in diverse workloads	Requires careful design
[23]	Power-aware scheduling	Manages energy consumption	Energy efficiency	Medium	Reduced power usage	May compromise speed

 doi:10.20944/preprints202412.0276.v2

[25, 38]	Real-time GPU scheduling	Ensures deadlines.	Timing-critical applications	Medium	Ensures predictable performance	Limited to real-time systems
[33, 36]	Batch job scheduling	Batches tasks	Resource optimization	Medium	High throughput	May cause latency for smaller tasks
[20]	Reinforcement learning-based scheduling	Optimizes allocation.	Intelligent task allocation	High	Optimized scheduling in uncertain environments	Training overhead, scalability
[29]	Composable schedules	Builds schedules	Hardware optimization	High	Flexibility in hardware design	Limited generalizability
[31, 37]	Resource partitioning	Divides resources.	Resource allocation	Medium	Improved resource utilization	May require significant tuning
[41]	Storage-aware scheduling	Optimizes storage	Storage and task placement	Medium	Better QoS and reduced cold starts	Relies on caching performance
[8, 27]	Temperature-aware scheduling	Adapts task schedules	Thermal optimization	Medium	Prolonged hardware lifespan, performance stability	Overhead of temperature monitoring
[40]	Network-sensitive scheduling	Minimizes communication delays	Communication overhead optimization	Medium	Reduced communication delays, faster completion	Complex job placement algorithms
[43]	Register-pressure-aware scheduling	Balances execution.	Instruction scheduling	Medium	Improved GPU instruction performance	Algorithm complexity
[18, 44]	Task graph scheduling	Optimizes dependencies	Dependency management	Medium	Reduced completion times	High complexity for large workloads
[19, 32]	Pipelined task scheduling	Overlaps task execution stages	Staged execution optimization	High	Faster task completion	Increased task management complexity
[35]	Energy-efficient scheduling	Uses successor tree consistency	Power optimization	Medium	Reduced energy usage	May impact real-time performance
[45, 48]	Task-based routing	Plans resources	Network-on-Chip scheduling	Medium	Optimized resource allocation, reduced latency	High computational requirements

4. Review of Design and Implementation Methodologies

This review paper does not present original experimental evaluations but instead offers a critical analysis and summary of the methodologies, evaluation metrics, and datasets utilized in prior studies. This analysis deepens the understanding of experimental designs, thereby promoting reproducibility and facilitating further research. The articles examined in this review focus on key evaluation metrics such as throughput, latency, energy efficiency, resource utilization, and makespan

 doi:10.20944/preprints202412.0276.v2

[56]. Additionally, performance metrics like scalability, fault tolerance, and response time are identified as essential for assessing system effectiveness. Commonly utilized datasets in the reviewed studies include standard benchmarks like ImageNet, CIFAR, and COCO, which are frequently employed for evaluating deep learning workloads, including sparse neural networks, convolutional neural networks (CNNs), and transformers. Significant tools and frameworks discussed in the literature include GPU computing platforms like CUDA and OpenCL, as well as profiling tools such as NVIDIA Nsight and various simulation environments used for evaluating scheduling strategies.

5. Summary of the Review

The review provides a comprehensive survey of recent developments in scheduling techniques aimed at maximizing the efficiency of deep learning computations on GPUs. It highlights the challenges associated with parallel thread execution and resource utilization on GPUs, which can result in suboptimal performance. The surveyed research focuses on novel scheduling policies that improve memory latency tolerance, exploit parallelism, and enhance GPU resource utilization.

Figure 9 illustrates the distribution of various scheduling techniques based on their primary focus areas. The chart reveals that optimization techniques constitute the largest segment, accounting for 40% of the surveyed methods, highlighting their critical role in enhancing execution efficiency and resource utilization. Memory optimization techniques follow, comprising 25%, underscoring the importance of effective memory access and throughput. Workload adaptation strategies make up 20%, reflecting the need for dynamic responsiveness to varying workloads. Additionally, energy efficiency techniques and resource allocation methods account for 10% and 5%, respectively, indicating a growing emphasis on power management and effective resource distribution in scheduling practices.

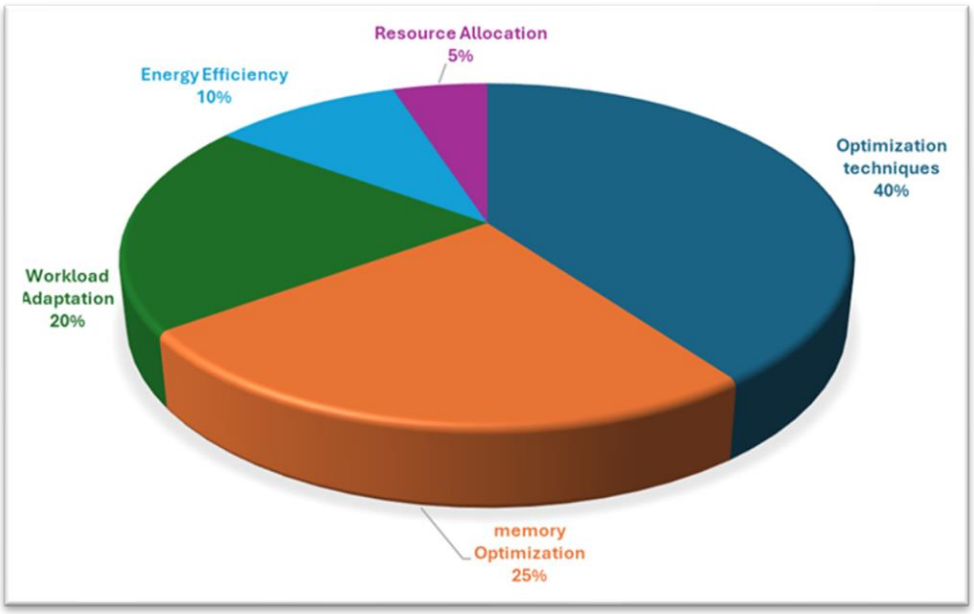

Figure 9. Key Areas of Emphasis in Modern Scheduling Approaches.

The review explores various aspects of scheduling techniques for deep learning on GPUs, including warp scheduling policies, memory latency tolerance, memory bank parallelism, context switching strategies, and prefetching mechanisms. These techniques aim to effectively manage the execution of parallel threads, optimize memory access patterns, and improve resource utilization. The integration of fine-grained warp scheduling, warp switching strategies, and prefetching mechanisms is also discussed to optimize deep learning computations on GPUs.

 doi:10.20944/preprints202412.0276.v2

Experimental evaluations demonstrate significant performance improvements in terms of throughput, memory bank parallelism, and latency reduction. The insights gained from this survey can guide researchers, system designers, and practitioners in developing more efficient and powerful deep learning systems on GPUs. The review also discusses potential future directions for research, such as advanced scheduling techniques, energy efficiency considerations, and the integration of emerging computing technologies.

Overall, the review provides a comprehensive overview of recent advancements in scheduling techniques for efficient deep learning computations on GPUs, highlighting their impact on system performance and suggesting future research directions. Table 2 summarizes various scheduling techniques, focusing on their strategies, key features, and applications, emphasizing advancements in optimizing GPU-based deep learning computations.

Table 2. Summary Table on Scheduling Techniques.

Ref No.	Scheduling Technique	Focus Area	Strategy Type	Key Features
[2,55]	Warp Scheduling	Thread assignment to warps	Dynamic	Efficient resource allocation and thread management
[2]	Memory Latency Tolerance	Minimizing memory latency	Dynamic	Overlapping memory access and computation
[2, 24]	Memory Bank Parallelism	Enhancing memory throughput	Dynamic	Exploiting parallel memory access capabilities
[52]	Context Switching	Optimizing resource usage	Dynamic	Efficient warp switching for resource optimization
[54]	Prefetching Mechanisms	Data access optimization	Dynamic	Anticipating and fetching data in advance
[55]	Fine-grained Scheduling	Individual thread prioritization	Dynamic	Enhancing parallelism and reducing contention
[46]	Workload Partitioning	CPU-GPU workload division	Static/Dynamic	Effective distribution based on workload and system conditions
[2, 24]	Memory Access Scheduling	Memory latency optimization	Dynamic	Effective scheduling to reduce memory latency
[9,12,18,27,38,47,48]	Task Graph Scheduling	Task graph optimization	Dynamic	Dynamic redistribution based on task graph characteristics
[7,10],28]	Dynamic Load Balancing	Workload redistribution	Dynamic	Adapting workload distribution in real-time
[53]	Hardware-Algorithm Co-design	Optimization for specific algorithms	Dynamic	Co-design approaches for algorithm and hardware optimization
[6,18,26,37,39,45]	Dynamic Task Scheduling	Real-time task optimization	Dynamic	Adapting task priorities based on system load
[8,16,17,40]	Adaptive Scheduling Policies	Dynamic scheduling adjustments	Dynamic	Real-time policy changes based on workload and system state

 doi:10.20944/preprints202412.0276.v2

[19,20,25,32]	Pipelined Task Scheduling	Task execution pipelining	Dynamic	Overlapping task execution stages for improved efficiency
[14,42,44]	Hybrid Scheduling Strategies	Combined scheduling approaches	Mixed	Utilizing a mix of static and dynamic scheduling techniques
[29,30,31,33,35,36]	Batch Job Scheduling	Job processing optimization	Static/Dynamic	Scheduling multiple jobs for optimized resource usage
[23,41,43]	Power-aware Scheduling	Energy-efficient task management	Dynamic	Adjusting scheduling based on power consumption considerations

6. Future Scope

In this survey paper, we have identified several promising future directions for advancing scheduling techniques in deep learning systems.

1. Hybrid Scheduling Algorithms: One key avenue is the development of hybrid scheduling algorithms that integrate artificial intelligence with traditional methods. By leveraging machine learning techniques, these algorithms can dynamically adapt to workload variations and resource availability, enhancing efficiency and responsiveness in real-time scenarios.
2. Scalability for Exascale Workloads: As computational demands grow, enhancing the scalability of scheduling strategies is crucial. Future research should focus on developing algorithms capable of managing exascale workloads, ensuring that scheduling techniques can handle vast data sets and complex models without sacrificing performance.
3. Emerging Hardware Technologies: The potential of emerging hardware technologies, such as neuromorphic and quantum processors, could transform scheduling approaches for deep learning. Research should explore how these novel architectures can be integrated into existing systems, optimizing scheduling to exploit their unique capabilities.
4. Energy Efficiency and Sustainability: With rising concerns about energy consumption in computing, future work should prioritize energy-efficient scheduling techniques. Investigating methods that minimize power usage while maintaining performance will be vital for sustainable computing in deep learning applications.
5. Robustness and Security: Tackling real-world deployment challenges, including fault tolerance and security, is essential. Future research should focus on developing scheduling techniques that ensure reliability in the face of hardware failures and security threats, particularly in critical applications such as autonomous systems and healthcare.
6. Interoperability and Standardization: As heterogeneous computing environments become more common, establishing standardized frameworks for scheduling across different platforms will be beneficial. Future research could aim to create guidelines and protocols that improve interoperability, facilitating smoother integration of various processing units.
7. Trends in Scheduling Techniques: Current trends indicate a shift towards adaptive and context-aware scheduling, which responds to real-time changes in workload and resource conditions. Future studies should investigate the effectiveness of these techniques in diverse scenarios, such as edge computing and federated learning, where resource constraints and latency are critical.
8. Artificial Intelligence in Scheduling: The increasing use of AI in scheduling presents an opportunity for research into intelligent scheduling systems that can predict future workload patterns and optimize resource allocation accordingly. This could involve the application of reinforcement learning and predictive analytics to enhance scheduling decisions.
9. Granularity and Precision in Scheduling: Future work should also explore fine-grained scheduling strategies that prioritize individual tasks based on their specific requirements.

 doi:10.20944/preprints202412.0276.v2

Investigating how to achieve optimal task granularity while minimizing overhead will be crucial for improving overall system performance.

10. User-Centric Scheduling: Finally, there's a growing need for user-centric scheduling approaches that consider user preferences and requirements. Future research could explore how to incorporate user feedback into scheduling algorithms, enabling more personalized and effective resource management.

By addressing these areas, researchers and practitioners can develop more robust, efficient, and adaptable scheduling techniques that meet the evolving needs of deep learning systems and their applications across various domains.

7. Conclusions

In conclusion, this review article provides a comprehensive survey of recent advancements in scheduling techniques aimed at maximizing the efficiency of deep learning computations on GPUs. The challenges associated with parallel thread execution, resource utilization, and memory latency in GPUs are highlighted, along with the need for efficient scheduling techniques to optimize performance.

The surveyed research focuses on novel scheduling policies that improve memory latency tolerance, exploit parallelism, and enhance GPU resource utilization. Various techniques such as prefetching mechanisms, fine-grained warp scheduling, and warp switching strategies are explored to optimize deep learning computations on GPUs. These techniques demonstrated significant performance improvements in terms of throughput, memory bank parallelism, and latency reduction.

The insights gained from this survey can guide researchers, system designers, and practitioners in developing more efficient and powerful deep learning systems on GPU. Furthermore, potential future directions are discussed, including advanced scheduling techniques, energy efficiency considerations, and the integration of emerging computing technologies, inspiring further research in this domain.

Future research in this domain can focus on several key areas to further optimize scheduling techniques for GPUs. First, the integration of energy-efficient algorithms with real-time scheduling methods could address the growing demand for sustainable computing. Second, exploring machine learning-based predictive models for dynamic scheduling can enhance adaptability and resource utilization in diverse workloads. Third, the convergence of scheduling strategies with emerging hardware technologies such as neuromorphic computing and quantum processors could unlock new possibilities for deep learning computations. Additionally, developing standardized frameworks to support heterogeneous systems and improve interoperability across platforms would be beneficial. Finally, addressing challenges in security-aware and fault-tolerant scheduling will ensure robustness in critical applications, opening avenues for deployment in industrial and edge computing environments. These advancements will continue to push the boundaries of deep learning and heterogeneous computing optimization.

Overall, the advancements in scheduling techniques presented in review offer valuable contributions to maximizing the efficiency of deep learning computations on GPUs, paving the way for improved performance and resource utilization in a wide range of applications and domains.

References

1. Mittal, S.; Vetter, J.S. A Survey of CPU-GPU Heterogeneous Computing Techniques. ACM Comput. Surv. 2015, 47, 1-35.
2. Fang, J.; Wang, M.; Wei, Z. A Memory Scheduling Strategy for Eliminating Memory Access Interference in Heterogeneous System. J. Supercomput. 2020, 76, 3129-3154.
3. Tang, X.; Fu, Z. CPU-GPU Utilization Aware Energy-Efficient Scheduling Algorithm on Heterogeneous Computing Systems. IEEE Access 2020, 8, 58948-58958.

 doi:10.20944/preprints202412.0276.v2

4. Wang, Z.; et al. Enabling Latency-Aware Data Initialization for Integrated CPU/GPU Heterogeneous Platform. IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. 2020, 39, 3433-3444.
5. Roeder, J.; Rouxel, B.; Altmeyer, S.; Grelck, C. Energy-aware Scheduling of Multi-version Tasks on Heterogeneous Real-time Systems. In Proceedings of the 36th Annual ACM Symposium on Applied Computing, March 2021; pp. 501-510.
6. Fang, J.; Zhang, J.; Lu, S.; Zhao, H. Exploration on task scheduling strategy for CPU-GPU heterogeneous computing system. In Proceedings of the 2020 IEEE Computer Society Annual Symposium on VLSI (ISVLSI), 306-311, IEEE, 2020.
7. Canon, L.-C.; Marchal, L.; Simon, B.; Vivien, F. Online scheduling of task graphs on heterogeneous platforms. IEEE Trans. Parallel Distrib. Syst. 2019, 31, 721-732.
8. Constantinescu, D.-A.; Navarro, A.; Corbera, F.; Fernández-Madrigal, J.-A.; Asenjo, R. Efficiency and Productivity for Decision Making on Low-Power Heterogeneous CPU+GPU SoCs. J. Supercomput. 2021, 77, 44-65.
9. Li, Z.; et al. Efficient Algorithms for Task Mapping on Heterogeneous CPU/GPU Platforms for Fast Completion Time. J. Syst. Archit. 2021, 114, 101936.
10. Grinsztajn, N.; et al. Readys: A Reinforcement Learning Based Strategy for Heterogeneous Dynamic Scheduling. In Proceedings of the 2021 IEEE International Conference on Cluster Computing (CLUSTER); IEEE, 2021.
11. Seo, W.; et al. SLO-aware Inference Scheduler for Heterogeneous Processors in Edge Platforms. ACM Trans. Archit. Code Optim. 2021, 18, 1-26.
12. Harichane, I.; Makhlouf, S.A.; Belalem, G. KubeSC-RTP: Smart Scheduler for Kubernetes Platform on CPU-GPU Heterogeneous Systems. Concurrency Comput. Pract. Exp. 2022, 34, e7108.
13. Asad, A.; Kaur, R.; Mohammadi, F. A Survey on Memory Subsystems for Deep Neural Network Accelerators. Future Internet 2022, 14, 146. https://doi.org/10.3390/fi14050146
14. Fang, J.; et al. Resource Scheduling Strategy for Performance Optimization Based on Heterogeneous CPU-GPU Platform. 2022.
15. Kaur, R.; Mohammadi, F. Power Estimation and Comparison of Heterogeneous CPU-GPU Processors. In Proceedings of the 2023 IEEE 25th Electronics Packaging Technology Conference (EPTC), 5 December 2023; pp. 948-951. IEEE.
16. Albahar, H.; et al. SchedTune: A Heterogeneity-Aware GPU Scheduler for Deep Learning. In Proceedings of the 2022 22nd IEEE International Symposium on Cluster, Cloud and Internet Computing (CCGrid); IEEE, 2022.
17. He, Z.; Sun, Y.; Wang, B.; Li, S.; Zhang, B. CPU-GPU Heterogeneous Computation Offloading and Resource Allocation Scheme for Industrial Internet of Things. IEEE Internet of Things J. 2023.
18. Hu, B.; Yang, X.; Zhao, M. Energy-Minimized Scheduling of Intermittent Real-Time Tasks in a CPU-GPU Cloud Computing Platform. IEEE Trans. Parallel Distrib. Syst. 2023, 34, 2391-2402.
19. Garcia-Hernandez, J.J.; Morales-Sandoval, M.; Elizondo-Rodríguez, E. A Flexible and General-Purpose Platform for Heterogeneous Computing. Computation 2023, 11, 97.
20. Liu, J.; Wu, Z.; Feng, D.; Zhang, M.; Wu, X.; Yao, X.; Yu, D.; Ma, Y.; Zhao, F.; Dou, D. Heterps: Distributed deep learning with reinforcement learning based scheduling in heterogeneous environments. Future Gener. Comput. Syst. 2023, 148, 106-117.
21. Kaur, R.; Mohammadi, F. Comparative Analysis of Power Efficiency in Heterogeneous CPU-GPU Processors. In Proceedings of the 2023 Congress in Computer Science, Computer Engineering, & Applied Computing (CSCE), 24 July 2023; pp. 756-758. IEEE.
22. Kaur, R.; Saluja, N. Comparative Analysis of 1-bit Memory Cell in CMOS and QCA Technology. In Proceedings of the 2018 International Flexible Electronics Technology Conference (IFETC), Ottawa, ON, Canada, 2018, pp. 1-3. doi: 10.1109/IFETC.2018.8584033.
23. Yang, Z.; Wu, H.; Xu, Y.; Wu, Y.; Zhong, H.; Zhang, W. Hydra: Deadline-aware and efficiency-oriented scheduling for deep learning jobs on heterogeneous GPUs. IEEE Trans. Comput. 2023, 72, 2224-2236.
24. Zhang, X. Mixtran: an efficient and fair scheduler for mixed deep learning workloads in heterogeneous GPU environments. Cluster Comput. 2024, 27, 2775-2784.

 doi:10.20944/preprints202412.0276.v2

25. A. Zou, J. Li, C. D. Gill, and X. Zhang, "RTGPU: Real-time GPU scheduling of hard deadline parallel tasks with fine-grain utilization," IEEE Trans. Parallel Distrib. Syst., vol. 34, no. 5, pp. 1450-1465, 2023.
26. Hsu, K.-C.; Tseng, H.-W. Simultaneous and Heterogeneous Multithreading. In Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture, 2023.
27. Gao, X. TAS: A Temperature-Aware Scheduling for Heterogeneous Computing. IEEE Access 2023, 11, 54773-54781.
28. Chun, C.-K.; Lai, K.-C. A Load Balance Scheduling Approach for Generative AI on Cloud-Native Environments with Heterogeneous Resources. In Proceedings of the 2024 10th International Conference on Applied System Innovation (ICASI); IEEE, 2024
29. Chen, H.; Zhang, N.; Xiang, S.; Zeng, Z.; Dai, M.; Zhang, Z. Allo: A Programming Model for Composable Accelerator Design. Proc. ACM Program. Lang. 2024, 8, PLDI, 593-620.
30. Ahmad, W.; Gautam, G.; Alam, B.; Bhati, B. S. An Analytical Review and Performance Measures of State-of-Art Scheduling Algorithms in Heterogeneous Computing Environment. Arch. Comput. Methods Eng. 2024, 1-23.
31. Belkhiri, A.; Dagenais, M. Analyzing GPU Performance in Virtualized Environments: A Case Study. Future Internet 2024, 16, 72.
32. Xue, W.; Wang, H.; Roy, C.J. CPU-GPU heterogeneous code acceleration of a finite volume Computational Fluid Dynamics solver. Future Gener. Comput. Syst. 2024, 158, 367-377.
33. Ye, Z.; Gao, W.; Hu, Q.; Sun, P.; Wang, X.; Luo, Y.; Zhang, T.; Wen, Y. Deep Learning Workload Scheduling in GPU Datacenters: A Survey. ACM Comput. Surv. 2024, 56, 1-38.
34. Tayeb, H.; Bramas, B.; Faverge, M.; Guermouche, A. Dynamic Tasks Scheduling with Multiple Priorities on Heterogeneous Computing Systems.2024.
35. Tariq, U.U.; Ali, H.; Nadeem, M.S.; Jan, S.R.; Sabrina, F.; Grandhi, S.; Wang, Z.; Liu, L. Energy-aware Successor Tree Consistent EDF Scheduling for PCTGs on MPSoCs. IEEE Access 2024.
36. Kwon, S.; Bahn, H. Enhanced Scheduling of AI Applications in Multi-Tenant Cloud Using Genetic Optimizations. Appl. Sci. 2024, 14, 4697.
37. Zhou, Y.; Ren, Z.; Shao, E.; Ma, L.; Hu, Q.; Wang, L.; Tan, G. FILL: a heterogeneous resource scheduling system addressing the low throughput problem in GROMACS. CCF Trans. High Perform. Comput. 2024, 6, 17-31.
38. Wang, Y.; Liu, C.; Wong, D.; Kim, H. GCAPS: GPU Context-Aware Preemptive Priority-based Scheduling for Real-Time Tasks. arXiv preprint arXiv:2406.05221, 2024.
39. Wang, S.; Chen, S.; Shi, Y. GPARS: Graph predictive algorithm for efficient resource scheduling in heterogeneous GPU clusters. Future Gener. Comput. Syst. 2024, 152, 127-137.
40. Sharma, A.; Bhasi, V.M.; Singh, S.; Kesidis, G.; Kandemir, M.T.; Das, C.R. GPU Cluster Scheduling for Network-Sensitive Deep Learning. arXiv preprint arXiv:2401.16492, 2024.
41. Lannurien, V.; Slimani, C.; d'Orazio, L.; Barais, O.; Paquelet, S.; Boukhobza, J. HeROcache: Storage-Aware Scheduling in Heterogeneous Serverless Edge-The Case of IDS. In Proceedings of the 24th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing, 2024.
42. Yan, K.; Song, Y.; Liu, T.; Tan, J.; Wei, X.; Fu, X. HSAS: Efficient task scheduling for large scale heterogeneous systolic array accelerator cluster. Future Gener. Comput. Syst. 2024, 154, 440-450.
43. Shobaki, G.; Muyan-Özçelik, P.; Hutton, J.; Linck, B.; Malyshenko, V.; Kerbow, A.; Ramirez-Ortega, R.; Gordon, V.S. Instruction Scheduling for the GPU on the GPU. In Proceedings of the 2024 IEEE/ACM International Symposium on Code Generation and Optimization (CGO), 435-447, IEEE, 2024.
44. Mohammadjafari, A.; Khajouie, P. Optimizing Task Scheduling in Heterogeneous Computing Environments: A Comparative Analysis of CPU, GPU, and ASIC Platforms Using E2C Simulator. arXiv preprint arXiv:2405.08187, 2024.
45. Ndamlabin, E.; Bramas, B. RSCHED: An effective heterogeneous resources management for simultaneous execution of task-based applications. 2024.
46. McGowen, J.; Dagli, I.; Dantam, N.T.; Belviranli, M.E. Scheduling for Cyber-Physical Systems with Heterogeneous Processing Units under Real-World Constraints. In Proceedings of the 38th ACM International Conference on Supercomputing, 298-311, 2024.

47. Verma, P.; Maurya, A.K.; Yadav, R.S. A survey on energy-efficient workflow scheduling algorithms in cloud computing. Softw. Pract. Exp. 2024, 54, 637-682.
48. Fang, J.; Wei, Z.; Liu, Y.; Hou, Y. TB-TBP: a task-based adaptive routing algorithm for network-on-chip in heterogeneous CPU-GPU architectures. J. Supercomput. 2024, 80, 6311-6335.
49. Al Abdul Wahid, S.; Asad, A.; Mohammadi, F. A Survey on Neuromorphic Architectures for Running Artificial Intelligence Algorithms. Electronics 2024, 13, 2963. https://doi.org/10.3390/electronics13152963.
50. Asad, A.; Kaur, R.; Mohammadi, F. Noise Suppression Using Gated Recurrent Units and Nearest Neighbor Filtering. In Proceedings of the 2022 International Conference on Computational Science and Computational Intelligence (CSCI), Las Vegas, NV, USA, 2022, pp. 368-372. doi: 10.1109/CSCI58124.2022.00072.
51. Kaur, R.; Asad, A.; Mohammadi, F. A Comprehensive Review of Processing-in-Memory Architectures for Deep Neural Networks. Computers 2024, 13, 174. https://doi.org/10.3390/computers13070174.
52. Zhang, H.; Xue, Y.; Zhou, Y.E.; Li, S.; Huang, J. SkyByte: Architecting an Efficient Memory-Semantic CXL-based SSD with OS and Hardware Co-design. 2025 IEEE International Symposium on High-Performance Computer Architecture (HPCA), 2025, 1-23. https://doi.org/10.1145/3711819.
53. Kim, S.; Hsiao, R.; Nikolić, B.; Demmel, J.; Shao, Y.S. SuperNoVA: Algorithm-Hardware Co-Design for Resource-Aware SLAM. Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '25), 2025, 1-17. https://doi.org/10.1145/3669940.3707258.
54. Yüzügüler, A.C.; Zhuang, J.; Cavigelli, L. PRESERVE: Prefetching Model Weights and KV-Cache in Distributed LLM Serving. Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '25), 2025, 1-16. https://doi.org/10.1145/3669940.3707258.
55. Tang, D.; Wu, Z.; Wang, Y.; Gu, Y.; Liu, F.; Qi, Z. gCom: Fine-grained Compressors in Graphics Memory of Mobile GPU. ACM Trans. Arch. Code Optim. 2025, 21, 1-23. https://doi.org/10.1145/3711819.
56. Hennessy, J.L.; Patterson, D.A. Computer Architecture: A Quantitative Approach; Elsevier: Amsterdam, The Netherlands, 2011.