A survey on Multi-GPUs

ATIYEH GHEIBI-FETRAT, Sharif University of Technology, Iran
ARAD MALEKI, Sharif University of Technology, Iran
AMIRSAEED AHMADI-TONEKABONI, Sharif University of Technology, Iran
MASOUD MOHAMMADI-LAK, Sharif University of Technology, Iran
SAHAND ZOUFAN, Sharif University of Technology, Iran
MAHDI ALINEJAD, Sharif University of Technology, Iran
MOHAMMAD ALIZADEH, Sharif University of Technology, Iran
KOMEIL YAHYAZADEH, Sharif University of Technology, Iran
NEGAR AKBARZADEH, Sharif University of Technology, Iran
SINA DARABI-MOGHADAM, IPM, Iran
SHAAHIN HESSABI, Sharif University of Technology, Iran
HAMID SARBAZI-AZAD, Sharif University of Technology and IPM, Iran

Graphics Processing Units (GPUs) play a critical role in contemporary applications that handle large data volumes. Nonetheless, the performance enhancements provided by GPUs are constrained by their memory capacity and bandwidth limitations, which can hinder their efficiency in processing extensive datasets. Multi-GPU architectures are increasingly recognized as an effective approach to meet the growing demands of modern parallel applications. These systems facilitate a scalable high-performance computing environment by harnessing the processing capabilities of multiple GPUs. This configuration not only delivers remarkable computational throughput but also enhances memory bandwidth and capacity, which are crucial for efficiently managing extensive parallel data-intensive workloads. Multi-GPU architectures provide substantial benefits for deep learning, scientific simulations, and large-scale data processing tasks that demand the efficient handling of terabytes of data. These systems leverage parallel processing capabilities to enhance computational throughput, significantly accelerating model training and simulation times while optimizing resource utilization. Unfortunately, multi-GPU systems grapple with critical challenges including inefficient memory management, communication bottlenecks, and uneven distribution of workloads. This paper presents a comprehensive review of recent software and hardware techniques addressing prevalent performance bottlenecks of multi-GPU systems. We specifically focus on methodologies optimizing cache management, improving load balancing, enhancing inter-GPU communication, and refining memory migration. Furthermore, the paper identifies emerging trends and unresolved challenges, offering insights into potential future directions for research and development. This

Authors' addresses: Atiyeh Gheibi-Fetrat, atiye.gheibi@gmail.com, Sharif University of Technology, Tehran, Tehran, Iran; Arad Maleki, arad.maleki02@gmail.com, Sharif University of Technology, Tehran, Tehran, Iran; Amirsaeed Ahmadi-Tonekaboni, amirsaeed.ahmadi99@sharif.edu, Sharif University of Technology, Tehran, Tehran, Iran; Masoud Mohammadi-Lak, masoudmohammadilak@gmail.com, Sharif University of Technology, Tehran, Tehran, Iran; Sahand Zoufan, sahandzoufan79@gmail.com, Sharif University of Technology, Tehran, Tehran, Iran; Mahdi Alinejad, , Sharif University of Technology, Tehran, Tehran, Iran; Mohammad Alizadeh, alizadehmohammad1011@gmail.com, Sharif University of Technology, Tehran, Tehran, Iran; Komeil Yahyazadeh, komeilyahyazade@gmail.com, Sharif University of Technology, Tehran, Tehran, Iran; Negar Akbarzadeh, , Sharif University of Technology, Tehran, Tehran, Iran; SINA DARABI-MOGHADAM, IPM, Tehran, Tehran, Iran; Shaahin Hessabi, hessabi@sharif.edu, Sharif University of Technology, Tehran, Tehran, Iran; Hamid Sarbazi-Azad, azad@sharif.edu, Sharif University of Technology and IPM, Tehran, Tehran, Iran.

Manuscript submitted to ACM

comprehensive study is intended to serve as a foundational resource for both academics and industry professionals seeking to advance the efficiency and scalability of multi-GPU systems.

CCS Concepts: • **Do Not Use This Code → Generate the Correct Terms for Your Paper**; *Generate the Correct Terms for Your Paper*; Generate the Correct Terms for Your Paper; Generate the Correct Terms for Your Paper.

Additional Key Words and Phrases: GPUs

ACM Reference Format:
Atiyeh Gheibi-Fetrat, Arad Maleki, Amirsaeed Ahmadi-Tonekaboni, Masoud Mohammadi-Lak, Sahand Zoufan, Mahdi Alinejad, Mohammad Alizadeh, Komeil Yahyazadeh, Negar Akbarzadeh, SINA DARABI-MOGHADAM, Shaahin Hessabi, and Hamid Sarbazi-Azad. 2018. A survey on Multi-GPUs. In *Proceedings of Make sure to enter the correct conference title from your rights confirmation emai (Conference acronym 'XX)*. ACM, New York, NY, USA, 33 pages. https://doi.org/XXXXXXX.XXXXXXX

1 Introduction

Recently, several technologies have started to utilize Graphics Processing Units (GPUs) as their main processor to enhance throughput and performance. GPUs, originally designed to handle the intense parallelism required for rendering images and videos, have evolved into powerful general-purpose accelerators. This transformation has revolutionized fields such as scientific computing, artificial intelligence (AI), machine learning (ML), and high-performance simulations, enabling advancements that were previously unattainable with traditional CPU-based systems. Their highly parallel architecture, capable of processing thousands of threads simultaneously, makes GPUs ideal for tasks involving large-scale data processing and computationally intensive applications.

This evolution has unlocked significant opportunities in research and technology, making complex simulations and data analyses feasible. Fields such as climate modeling, molecular dynamics, and astrophysics now benefit from the power of GPU clusters, which were once accessible only via supercomputers. In deep learning and neural network training, GPUs have drastically reduced training times by accelerating matrix computations. Continuous advancements in GPU design, such as the increase in the number of Streaming Multiprocessors (SMs) and link bandwidth of individual GPUs, have further propelled this progress, driven by higher transistor density through CMOS technology. However, the limitations imposed by the end of Moore's Law [1] constrain the scalability of single GPUs [2], making it no longer cost-effective to add more computing resources to individual GPUs [3]. As a result, numerous research efforts have been conducted to explore new approaches for scaling GPU performance.

Multi-GPU systems, inspired by multicore architectures, have emerged as a promising solution to address the increasing computational demands of modern applications. Single-processor architectures face challenges in scaling to accommodate the demands of modern workloads. Meanwhile, multi-GPU systems offer a compelling solution by providing higher computational power and larger memory capacities. These systems are particularly advantageous for deep neural networks, large-scale simulations, and complex data processing tasks. Nevertheless, the transition to multi-GPU systems introduces new challenges, particularly regarding inefficient GPU-to-GPU communication and memory management which limited the performance gains of these systems [4]. Current techniques, such as first-touch demand paging, often result in imbalanced memory page distribution and excessive overhead due to frequent data migrations. These inefficiencies lead to load imbalances and degraded system performance [4]. Recent advancements have introduced unified memory models and programmer-transparent solutions to simplify multi-GPU programming and enhance performance. These solutions minimize programmer intervention by automating data placement and migration processes. For instance, Griffin, a holistic hardware-software solution, addresses these challenges by improving

the efficiency of page migration in NUMA-based multi-GPU systems. By employing novel mechanisms for runtime page migration and load balancing, Griffin significantly boosts system performance while minimizing implementation overhead [4]. Despite these advancements, multi-GPU systems still face persistent challenges, especially concerning the need to re-engineer applications originally optimized for single-GPU environments. Techniques such as Peer-2-Peer (P2P) access [5] offer solutions to these challenges, yet they often require extensive modifications to existing codebases, hindering their widespread adoption [6]. The shift towards multi-socket designs may also demand new programming paradigms, requiring developers to adapt to complex NUMA configurations [6].

The increasing popularity of multi-GPU systems can be attributed to advancements in simulation tools, such as those developed in [7], which have made exploring multi-GPU architectures more feasible. Historically, multi-GPU research was limited, with early studies offering only broad ideas and theories about their potential, often without extensive experimentation or detailed exploration [8–10]. A key bottleneck identified early on was the complexity of communication and memory management between GPUs, as well as between GPUs and CPUs. Notably, Kim et al. proposed the Unified Memory Network to address this challenge, integrating CPU and GPU memory into a single network to reduce data transfer bottlenecks [11]. This approach continues to inspire current solutions [4].

This survey aims to provide a comprehensive overview of the advancements in multi-GPU technology, focusing on architectural innovations and software-level adaptations necessary for optimizing multi-GPU performance. By exploring these advancements, we aim to clarify the potential of multi-GPU systems and their role in modern computing applications. The rest of the paper is organized as follows: Section 2 provides an in-depth background on GPU architecture, highlighting critical differences between single-GPU and multi-GPU systems. Section 3 discusses the challenges associated with multi-GPU systems, including memory management, communication inefficiencies, and programming complexities. Section 4 reviews recent technological advancements addressing these challenges, with a focus on hardware-software co-design, interconnection improvements, and dynamic memory optimization techniques. Section 5 explores key use cases of multi-GPU systems in domains such as artificial intelligence, scientific computing, and large-scale simulations while discussing emerging trends and future directions. Finally, Section 6 concludes the paper by summarizing the findings and proposing areas for further research.

Table 1. GPU Generations Specifications [12–23].

GPU Generations	#SM	BW (GB/s)	#Transistors (B)	Tech. Node (nm)	Chip Size (mm^2)	Memory Size (GB)
Fermi (GF100)	15	177	3	40	529	Up to 6
Kepler (GK110)	15	288	7.1	28	551	Up to 12
Maxwell (GM200)	24	288	8	28	601	Up to 24
Pascal (GP100)	56	720	15.3	16	610	16
Volta (GV100)	80	900	21.1	12	815	16/32
Ampere (GA100)	108	1555	54.2	7	826	40
Hopper (H100 SMX5)	132	3000	80	4N customized for NVIDIA	814	80
Blackwell (B200)	160	16000 (2x8000)	208 (2x104)	4N customized for NVIDIA	N/A	384 (2x192)

2 Background

Graphics Processing Units (GPUs) are specialized electronic circuits designed to accelerate graphics display and complex calculations. Originally designed to enhance 3D graphics and visual effects in video games, GPUs have evolved into versatile parallel processors that can handle a variety of computer tasks. Modern GPUs have specialized cores which consist of thousands of small discrete units that can run multiple tasks simultaneously, making it especially efficient for

parallel processing. The architecture is different from CPUs in the sense that they are optimized for sequential work. We cover the important components of the GPU that are needed for a better understanding of Multi-GPU systems in section 2.1.

Following the current industry trends and research requirements, multi-GPU systems are becoming increasingly critical for handling large-scale computations. Single-GPU setups often face memory and performance bottlenecks, particularly in areas like AI training [24], scientific simulations [10], and deep learning [25], where datasets and models are growing rapidly. Multi-GPU configurations allow parallel processing, which drastically reduces training time and improves computational efficiency. Studies show that multi-GPU systems significantly boost performance in complex tasks like natural language processing [26, 27], image recognition [25, 28], and even traffic management [29].

Multi-GPU architectures, which integrate multiple GPU chips to enhance computational performance, are categorized into two primary types: multi-socket GPUs and Multi-Chip-Module (MCM) GPUs [3]. Multi-socket GPUs consist of several GPU and memory chips connected via a Printed Circuit Board (PCB), While MCM GPUs involve multiple GPU and memory dies interconnected within a single package using silicon interposers or organic substrates. The significant distinction between these architectures lies in the inter-chip bandwidth; MCM GPUs provide higher bandwidth but at a higher cost and with limited memory capacity. Conversely, multi-socket GPUs are more cost-effective and offer greater memory capacity, albeit with lower inter-chip bandwidth. A key challenge for both architectures is managing the bandwidth disparity between inter-chip and intra-chip connections.

2.1 Over view of GPUs

Over the years, GPUs have expanded their utility beyond graphics rendering to include scientific research, artificial intelligence (AI), and data analysis. In scientific research, GPUs accelerate simulations and data processing, facilitating breakthroughs in fields such as physics, biology, and climate science. The rise of AI and machine learning has further amplified the importance of GPUs, as they provide the computational power necessary for training complex models and handling large datasets efficiently. For instance, NVIDIA's CUDA (Compute Unified Device Architecture) platform and AMD's ROCm framework enable developers to leverage GPU power for a wide range of applications beyond traditional graphics.

CUDA is NVIDIA's parallel computing platform and programming model designed for general-purpose computing on GPUs (Graphics Processing Units). CUDA provides developers with access to virtual instruction sets and memory of the parallel computational elements within NVIDIA GPUs, enabling high-performance parallel computing for tasks in fields such as deep learning, scientific computing, and real-time data processing. In parallel, AMD's ROCm (Radeon Open Compute) platform offers a similar suite for GPU programming on AMD hardware, providing tools and libraries for parallel computing, deep learning, and HPC applications on AMD GPUs. To optimize GPU utilization, Multi-Process Service (MPS) allows multiple CUDA applications to share a single GPU efficiently, which is particularly useful for maximizing resource usage on large data-center GPUs by reducing the overhead of GPU context switching. Programming for GPUs with CUDA involves writing parallelized code using extensions to languages like C, C++, or Python, where developers can manage thousands of threads running concurrently on GPU cores. This model enables massive parallelism, allowing complex computations to be broken into smaller tasks that can be executed in parallel, significantly boosting performance in tasks that benefit from high-throughput processing.

As GPU architectures have evolved, they have demonstrated significant advancements in processing units, memory bandwidth, and manufacturing technologies. Table 1 highlights the specifications of various NVIDIA GPU generations.

Modern GPUs are composed of several key components, each playing a critical role in its functionality. In the following subsections, we briefly introduce these components.

2.1.1 Streaming Multiprocessors (SMs)

The core computational units in a GPU are called **Streaming Multiprocessors (SMs)**. SMs are designed to execute a vast number of threads in parallel, forming the foundation of the GPU's unparalleled parallel processing capabilities. Within each SM are multiple **CUDA cores** (or their equivalents in non-NVIDIA architectures, such as AMD's stream processors). These CUDA cores execute the individual instructions of threads in a warp, leveraging **Single Instruction, Multiple Threads (SIMT)** architectures to maximize throughput.

Each SM also contains a variety of specialized components to support efficient execution:

- **Warp Schedulers**: Responsible for dispatching and managing instructions for groups of 32 threads called *warps*.
- **Register Files**: Large arrays of registers that store temporary data for threads, ensuring low-latency access to frequently used variables.
- **Shared Memory**: High-speed, programmer-controlled memory shared among all threads in the SM, reducing reliance on slower global memory.
- **Special Function Units (SFUs)**: Dedicated units for mathematical operations such as trigonometric functions, square roots, and exponentials.
- **Load/Store Units (LD/ST)**: Responsible for transferring data between the global memory and local thread memory.

SMs are highly scalable, with each new GPU generation typically increasing the number of SMs per GPU and enhancing their architecture for improved efficiency. Modern SMs can handle **thousands of threads concurrently**, organized into *thread blocks* and warps. This concurrency makes GPUs highly efficient for tasks such as matrix multiplications, neural network training, physical simulations, and other workloads involving massive datasets or highly repetitive computations.

Additionally, SMs are integrated with advanced features such as **tensor cores** (for accelerating deep learning computations) in NVIDIA GPUs, or **ray tracing cores** (for real-time ray tracing in graphics). These specialized cores further enhance the computational versatility of the GPU.

The combination of massive parallelism, high memory bandwidth, and specialized units within the SMs enables GPUs to deliver unparalleled performance for both traditional graphics rendering and general-purpose computing (GPGPU).

2.1.2 CUDA Cores

CUDA cores are the individual arithmetic logic units (ALUs) within an SM that perform basic calculations such as integer addition, floating-point operations, and logical operations. A GPU may have thousands of CUDA cores, enabling it to handle many simultaneous operations. These cores are optimized for parallel processing but are less powerful on a per-core basis than CPU cores, which are designed for more complex single-threaded tasks.

2.1.3 Tensor Cores

Tensor cores, introduced in more recent GPU architectures, are specialized processing units designed to accelerate deep learning computations, especially matrix multiplications used in neural networks. Tensor cores perform mixed-precision operations, meaning they can compute with lower precision values such as FP16 (16-bit floating point), allowing for faster computations without significant loss in accuracy. Tensor cores are integral to tasks like AI model training and inference.

2.1.4 Memory Hierarchy

A GPU's memory hierarchy is crucial for its performance. It includes different types of memory optimized for various use cases:

- **Global Memory:** This is the largest pool of memory, accessible by all threads, but it has the highest latency.
- **Shared Memory:** Shared memory is a smaller, faster memory that is accessible by all threads within an SM. It is used for inter-thread communication and storing frequently accessed data.
- **Register Memory:** Registers are the fastest and smallest memory spaces available to each thread for temporary data storage during computation.
- **L1 and L2 Caches:** GPUs use a caching hierarchy similar to CPUs. L1 cache is faster but smaller, and each SM may have its own cache. L2 cache is larger and shared among multiple SMs.

2.1.5 Texture and Raster Units

In addition to raw computation, GPUs are optimized for rendering 3D graphics. Texture units are responsible for fetching and filtering texture data, while raster units convert vector graphics (represented by vertices and lines) into raster images (pixels). These units handle operations specific to graphics pipelines, such as shading and image rendering.

2.1.6 Memory Controllers and Bandwidth

The memory controller manages data flow between the GPU's core and its memory subsystem. High memory bandwidth is crucial for GPU performance, particularly for data-intensive applications such as gaming or scientific simulations. The faster the memory can communicate with the cores, the less time the cores spend idle, waiting for data.

2.1.7 PCI Express Interface

The PCI Express (PCIe) interface connects the GPU to the motherboard and, by extension, to the CPU and system memory. Data transfer between the CPU and GPU is handled via this interface. Though fast, PCIe bandwidth is often lower than the internal bandwidth within the GPU, making it a limiting factor in some workloads that require frequent data exchange between the CPU and GPU.

2.2 Multi-GPU Components

Inspired by the advances in multi core systems, the use of multiple GPUs in a system was proposed [9, 10]. Multi-GPU systems are designed to distribute computational workloads across several GPUs to increase performance. These systems consist of specific components that enable effective communication and workload distribution among GPUs. Below is a brief overview of them.

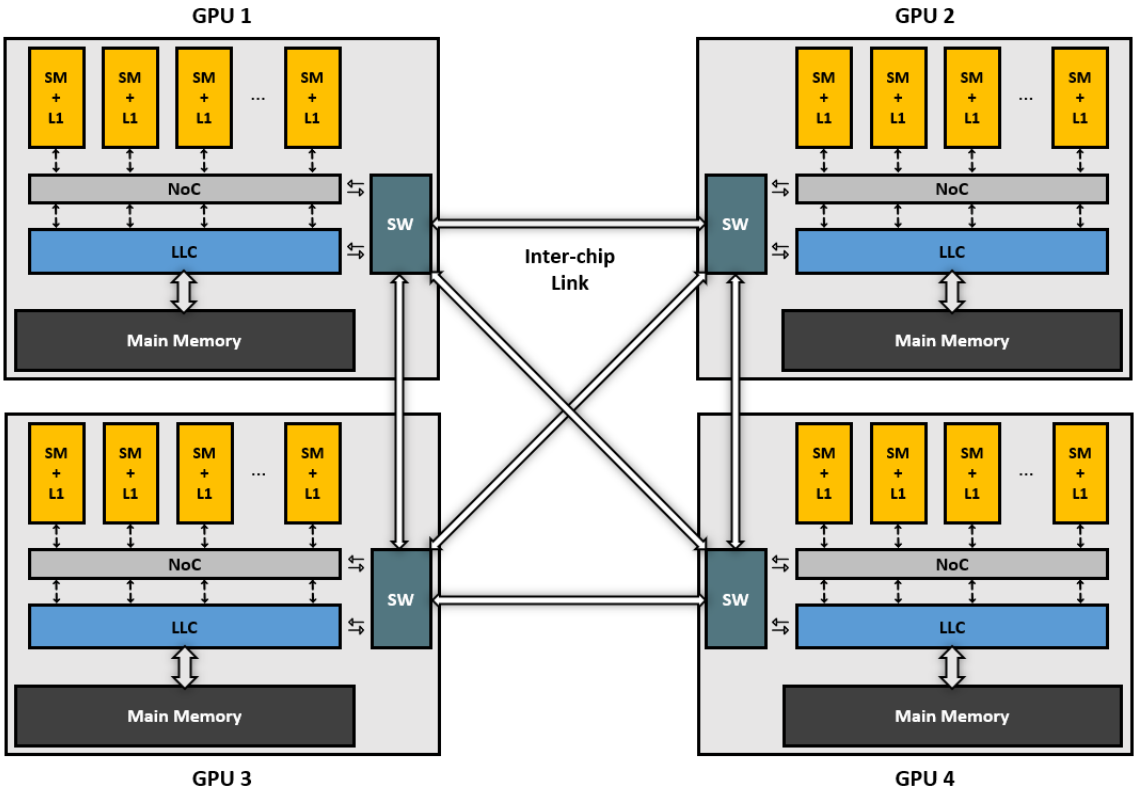

Fig. 1. Conventional MGPU system. Switch (SW) handles the remote access request from one GPU to another. PCIe or NVLink is used as the inter-chip link.

2.2.1 Interconnects

Interconnects are the primary component for enabling communication between GPUs. The most commonly used interconnects in multi-GPU systems are:

- **NVIDIA NVLink [30]:** A high-speed interconnect provides direct data transfer between GPUs with higher bandwidth and lower latency than PCIe.
- **PCI Express (PCIe) [31, 32]:** Standard interconnect used in most multi-GPU setups, though with lower bandwidth compared to NVLink. Refer to section 2.1.7 for more details.

- **AMD Infinity Fabric [33]:** A proprietary interconnect used in AMD GPUs for efficient communication between GPUs and CPUs.

2.2.2 Cross-GPU Memory Management

Memory management across multiple GPUs is essential for high-performance parallel computing. The two key models are:

- **Unified Memory:** GPUs share the same memory space, allowing seamless access to data across GPUs.
- **Discrete Memory:** Each GPU has its own memory, and explicit data transfer between GPUs is required when sharing data.

2.2.3 GPU Bridges

Physical bridges are often used in multi-GPU setups to ensure high-speed, direct communication between GPUs. Examples include:

- **NVLink Bridge:** Used to physically link NVIDIA GPUs in systems that support NVLink.
- **CrossFire and SLI Bridges:** Legacy bridge technologies for linking GPUs, primarily used for gaming setups.

2.2.4 Scalability Controllers

Scalability controllers are components within multi-GPU systems that handle workload distribution and resource management:

- **PCIe Switches:** These switches allow multiple GPUs to connect to a single CPU or shared bus, distributing tasks effectively across GPUs.
- **Multi-GPU Scheduling:** In some systems, hardware and firmware manage task scheduling across multiple GPUs to ensure efficient resource utilization.

2.3 MCM GPU

A Multi-Chip Module (MCM) GPU is an advanced architecture that leverages multiple silicon chips integrated into a single package to act as a cohesive GPU. This approach contrasts with traditional monolithic designs where all GPU components are placed on a single die. MCM architectures provide several benefits, including improved manufacturing yields, increased processing power, and greater flexibility for scaling. By allowing separate chips, or "chiplets," to handle distinct tasks, MCM GPUs can potentially reduce inter-chip latency, optimize energy efficiency, and offer enhanced parallel processing capabilities. As a result, MCM GPUs are well-suited for large-scale, high-performance applications, particularly in fields such as deep learning, scientific computing, and real-time rendering.

2.3.1 Compute Chiplets

Compute chiplets are individual processing units within an MCM GPU that contain GPU cores and memory interfaces. Each compute chiplet operates semi-independently, performing core GPU tasks and communicating with other chiplets to coordinate parallel processing workloads.

2.3.2 Interconnect

The interconnect is a high-speed communication link that facilitates data transfer between chiplets within the MCM GPU. It is designed to reduce latency and maximize bandwidth between chiplets, ensuring efficient data exchange and coherent memory access across the module.

2.3.3 Memory Cache and Interface

MCM GPUs often include a cache and memory interface layer, which manages data storage and retrieval between chiplets and the GPU's main memory. This component optimizes data locality and reduces the latency of memory access for chiplets, thereby enhancing computational efficiency.

2.3.4 Power Management

Power management is critical in MCM designs to balance energy efficiency and performance. Dedicated power management components regulate power distribution across chiplets, adjusting dynamically to workload demands to optimize overall GPU performance while minimizing energy consumption.

2.3.5 Control Logic

The control logic coordinates the tasks and scheduling between chiplets, overseeing load balancing and task allocation. This ensures that each chiplet operates efficiently and synchronizes processing tasks to avoid bottlenecks, improving the GPU's overall performance and scalability.