Concurrency and Computation: Practice and Experience

WILEY

REVIEW ARTICLE

A Survey on Heterogeneous CPU–GPU Architectures and Simulators

Mohammad Alaei | Fahimeh Yazdanpanah

Computer Engineering Department, Faculty of Engineering, Vali-e-Asr University of Rafsanjan, Rafsanjan, Iran

Correspondence: Fahimeh Yazdanpanah (yazdanpanahf@vru.ac.ir)

Received: 31 March 2024 | **Revised:** 21 September 2024 | **Accepted:** 9 October 2024

Keywords: heterogeneous CPU–GPU architectures | heterogeneous CPU–GPU simulators | network-on-chip | shared resources management | task scheduling

ABSTRACT

Heterogeneous architectures are vastly used in various high performance computing systems from IoT-based embedded architectures to edge and cloud systems. Although heterogeneous architectures with cooperation of CPUs and GPUs and unified address space are increasingly used, there are still a lot of open questions and challenges regarding the design of these architectures. For evaluation, validation and exploration of next generation of heterogeneous CPU–GPU architectures, it is essential to use unified heterogeneous simulators for analyzing the execution of CPU–GPU workloads. This article presents a systematic review on challenges of heterogeneous CPU–GPU architectures with covering a diverse set of literatures on each challenge. The main considered challenges are shared resource management, network interconnections, task scheduling, energy consumption, and programming model. In addition, in this article, the state-of-the-art of heterogeneous CPU–GPU simulation platforms is reviewed. The structure and characteristics of five cycle-accurate heterogeneous CPU–GPU simulators are described and compared. We perform comprehensive discussions on the methodologies and challenges of designing high performance heterogeneous architectures. Moreover, for developing efficient heterogeneous CPU–GPU simulators, some recommendations are presented.

1 | Introduction

High performance computing systems, such as intelligent and IoT-based systems and cloud computing systems typically use heterogeneous architectures to exploit data-level, instruction-level and thread-level parallelism at the same time [1]. Heterogeneous architectures combine processing cores, such as CPUs and GPUs, with different characteristics and different instruction set architectures (ISAs) [2–6]. These heterogeneous architectures are evolving towards tighter integration to improve speed and throughput and also to reduce power consumption. The advancements in VLSI design and semiconductor manufacturing technologies allow multiple heterogeneous cores to be integrated in a single chip. Various algorithms such as artificial intelligent and machine learning algorithms [7–12], edge and cloud computing [13–16], neural and biological networks [17–20], robotics and signal processing [21–31], and query processing [32, 33] were presented for executing on heterogeneous architectures.

CPUs and GPUs have their specific characteristics and strengths; therefore, cooperation of CPU and GPU cores provides efficient parallel computing at different granularities. In heterogeneous architectures with off-chip GPUs (discrete GPUs), CPU and GPU cores have been interconnected through a PCI Express bus. This kind of data transferring inflicts considerable performance overheads. For reducing the communication costs, improving performance and programmability of emerging architectures, CPUs and GPUs have been integrated on the same die with shared last level caches (LLCs), off-chip memory controllers (MCs),

and on-chip network structure. Compared to discrete GPUs, the on-chip integration GPUs enables close collaboration between cores, better utilization of shared resources, and the elimination of costly data transfers between core and various memory modules. In shared memory systems, such as heterogeneous CPU–GPU architectures, on-chip interconnections are essential [34]. Nevertheless, single-die CPU–GPU architectures face various challenges because of heterogeneity, limited area and different memory requirements.

These challenges have motivated many researchers to investigate on various aspects of heterogeneous computing architectures for improving performance and removing restrictions. For instance, Mittal et al. [4] presented a survey on different aspects of heterogeneous computing techniques such as fused computing approaches in algorithms, programming models, compilers, and workload partitioning. Raju et al. [35] reviewed the various techniques for high performance and energy-efficient heterogeneous CPU–GPU architectures. A survey on software techniques for emulating heterogeneous memory systems in high performance computing was presented in [36]. A systematic review on memory management techniques in heterogeneous CPU–GPU architectures was presented in [37].

With emerging heterogeneous processors, it is necessary to develop cycle-accurate simulators which are able to emulate and evaluate the heterogeneous designs [34]. It is imperative to explore modern heterogeneous CPU–GPU architectures and understand any associated issues with designing these emerging many-core systems. Hence, full-system heterogeneous CPU–GPU simulators are essential to explore the behavior of the whole system by running various heterogeneous applications, considering the interconnection networks and the other shared resources [38–42].

This article concentrates on the challenges of heterogeneous CPU–GPU architectures and the structures of their simulation platforms. The main challenges include shared resource management, network interconnections, task scheduling, energy consumption, and programming model. For each challenge, the recent related works are reviewed and discussed. Then, this article presents a comprehensive review on various aspects of heterogeneous CPU–GPU simulation platforms. The structure and characteristics of the main examples of these simulators are reviewed and compared.

The main contributions of this article can be summarized as follows:

1. We classify and describe the main challenges in heterogeneous CPU–GPU architectures and cover a diverse set of literatures on each challenge.
2. The state-of-the-art of heterogeneous CPU–GPU simulators is reviewed. Then, the structure and the main aspects of five cycle-accurate heterogeneous simulation platforms are described.
3. We discuss the main issues on designing high performance heterogeneous CPU–GPU architectures and the methodologies and techniques for developing their simulators.
4. Considering the challenges and discussions, some recommendations are provided for developing effective heterogeneous CPU–GPU simulators.

The rest of this article is organized as follows. The GPU architecture and its simulators are briefly explained in Section 2. The heterogeneous CPU–GPU architectures are described in Section 3. Then, Section 4 presents challenges of heterogeneous CPU–GPU architectures. Heterogeneous CPU–GPU simulators are described in Section 5. Section 6 contains the descriptions of structure and features of the main examples of heterogeneous CPU–GPU simulators. Section 7 is dedicated to recommendations for developing efficient heterogeneous simulators. Finally, Section 8 presents the conclusions.

2 | Background: Graphics Processing Units

As technology has advanced, graphics processors went from being special-purpose, high-cost, and separated co-processors to becoming smaller and more affordable accelerators attached to CPUs. GPUs become general-purpose and energy-efficient many-core systems that provide high performance computing [43–51]. In GPUs, hundreds to thousands of threads can compute concurrently at data-level parallelism. GPUs contain an array of processing elements called stream multi-processors (SMs) in NVIDIA or compute units in AMD documents. Inside each SM, there are several stream processors (SPs) for arithmetic operations, which share caches and control logic in their SM. The number of SPs is different in each generation of GPUs. Figure 1 shows overall structure of NVIDIA GPUs. All SMs of a GPU share a DRAM memory. The SMs are connected to the DRAM controllers through a bus.

The bandwidth from SMs to DRAM is high, but for high computing applications, this bandwidth is still finite for thousands of threads accessing memory simultaneously. The main memory of GPUs is separated from the main memory of CPUs, and direct system calls are required for transferring data between these modules through the PCI Express bus [47, 49]. GPUs include an L1 data cache in each SM, and a global L2 cache that is shared between all SMs. In addition, a shared memory is embedded and controlled by software and can be tuned by programmers [36, 37, 43–56]. GPUs utilize large groups of data threads (called warps) for executing the same instruction on an array. Warp processing is similar to vector processing, but much more flexible and efficient, and does not have vector processing limitations. The main limitations of vector processing are fixed and pre-defined length and consecutive memory locations of vector elements [57–61]. SMs of the modern GPUs include several schedulers, execution units, and register file banks; hence, for decreasing area and energy-consumption, SMs are divided into sub-cores. Each sub-core includes a distinct warp scheduler and register file [57]. A lookup table within each SM cluster be used [52] to capture replicated cache requests and to alleviate congestion problems of the network interconnection. A systematic survey on the approaches for optimizing GPU performance was presented in [44]. Moreover, hardware GPU multitasking methods, including temporal, spatial, and simultaneous was reviewed in [62].

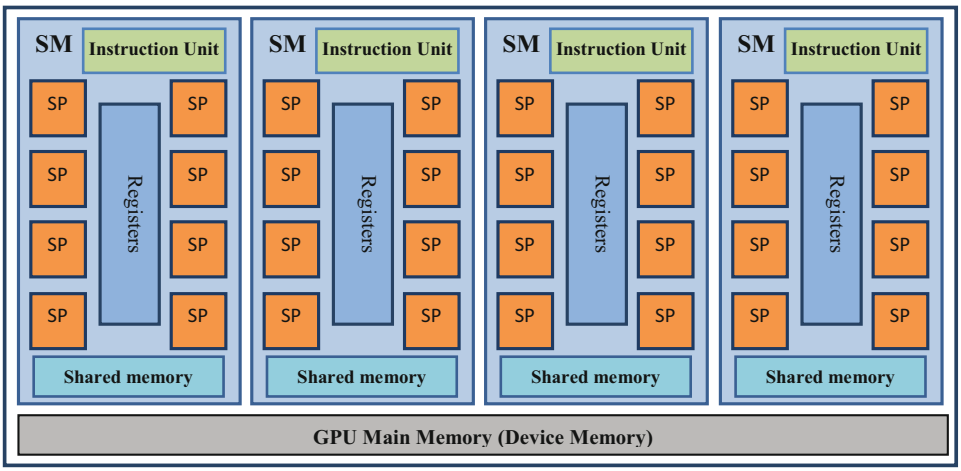


FIGURE 1 | General structure of NVIDIA GPU architectures.

OpenCL (open computing language) [63] and CUDA (compute unified device architecture) [64] are the most famous GPU programming models. OpenCL applications are usually executed on heterogeneous architectures [65]. CUDA programs include several kernels that are launched, and run concurrently on a large number of GPU threads [45, 46, 48, 66, 67]. By the programmer, the threads can be grouped into blocks and grids of blocks. The blocks are dynamically dedicated to SMs according to the existing registers and shared memory spaces.

Researchers have developed several simulation frameworks for modeling and analysis of GPU architectures [68–82]. For instance, Qsilver [69], Attila [70] and MIAOW [71] are cycle-accurate and execution-driven GPU simulators for OpenGL-based applications. Barra [72] and Multi2Sim Kepler [73] are functional GPU simulators for CUDA programs. Another functional GPU simulator is Ocelot [74], which is a dynamic compiler working at a virtual ISA level. GpuTejas [75] is a java-based, trace-driven GPU simulator. GPGPU-Sim is the most famous GPU simulator, which emulates various GPU micro-architectures in functional and timing modes. Accel-Sim [76] is a configurable industrial GPU simulation tool and is based on GPGPU-Sim. Daisen [77] is a framework that supports data collection from GPU simulators and provides visualization of the simulator-generated GPU execution traces. MGPUSim [78] was presented for performance analysis and optimization of multi-GPU with a customized multi-GPU memory management system. A tool for modeling performance and power consumption of GPU applications using static and dynamic analysis was presented in [79, 80]. PPT-GPU [81] is a scalable performance prediction toolkit for GPUs that presents a comprehensive report of GPU performance metrics. FLAME GPU2 [82] is an agent-based GPU simulators.

3 | Heterogeneous CPU–GPU Architectures

Heterogeneous CPU–GPU architectures, consisting of multi-core CPUs and many-core GPUs, have been popular in various environment from IoT-based embedded systems to grid processing and cloud computing environments to provide high performance and energy-efficient architectures [2–11, 13–18, 21–27, 83]. They take advantages of two different processors, provide low manufacturing cost and high performance architectures, and furthermore, reduce the limitations of CPUs and GPUs [3, 4]. CPUs and GPUs are able to cooperate to effectively execute both data-parallel applications and control-intensive workloads. CPUs, as the hosts, assign massive data-parallel tasks, that predefined by programmers, to GPUs, as the devices. For this, the CPU copies corresponding data and instructions from its main memory to the device memory. Then, the same instruction is executed by multiple threads with distinct data sets per GPU core. GPUs process large data sets concurrently, and return the results to CPUs [4]. Comparing with CPU architectures, GPUs have many more fine-grained processing units, larger memory buses, and specific operations for fast computations. Typically, the integrated heterogeneous cores share address spaces of the memory modules (e.g., cache-coherent memory hierarchy and shared LLCs), and need an on-chip interconnection structure [84]. Figure 2 shows a simple die map of an on-chip heterogeneous CPU–GPU processor.

Modern multi-core CPUs typically include superscalar and multi-instruction issue cores in out-of-order form, with various speculative and prediction mechanisms. In addition, CPUs use large caches, while GPUs employ a large number of in-order cores with smaller-sized caches. Thus, CPUs are suitable for latency-critical workloads, and GPUs are appropriate for throughput-critical workloads.

Heterogeneous CPU–GPU architectures can be discrete on separated chips (i.e., off-chip) or integrated on the same chip (i.e., on-chip). On-chip integrated CPU–GPU systems offer high performance heterogeneous computing systems because they replace the cost of PCI Express data transmission by efficient network-on-chip (NoC) architecture [85–101]. Low overhead of CPU–GPU data transmission is the most important advantage of on-chip heterogeneous architectures, and consumes less energy than discrete GPUs attached to CPUs [90]. Furthermore, placing CPU and GPU cores on an integrated platform and sharing the common network resources can decrease the data transfer latency.

Figure 3 presents two heterogeneous systems, one is discrete CPU–GPU architecture (Figure 3a), and the other is

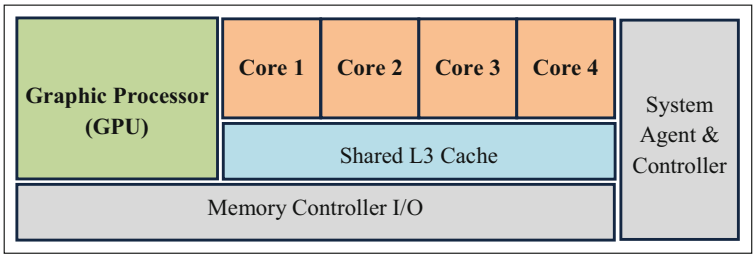
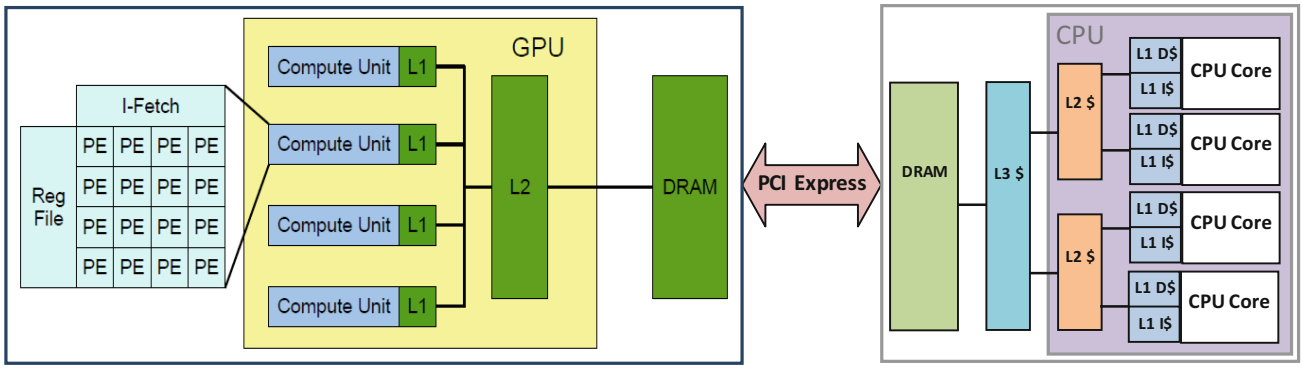
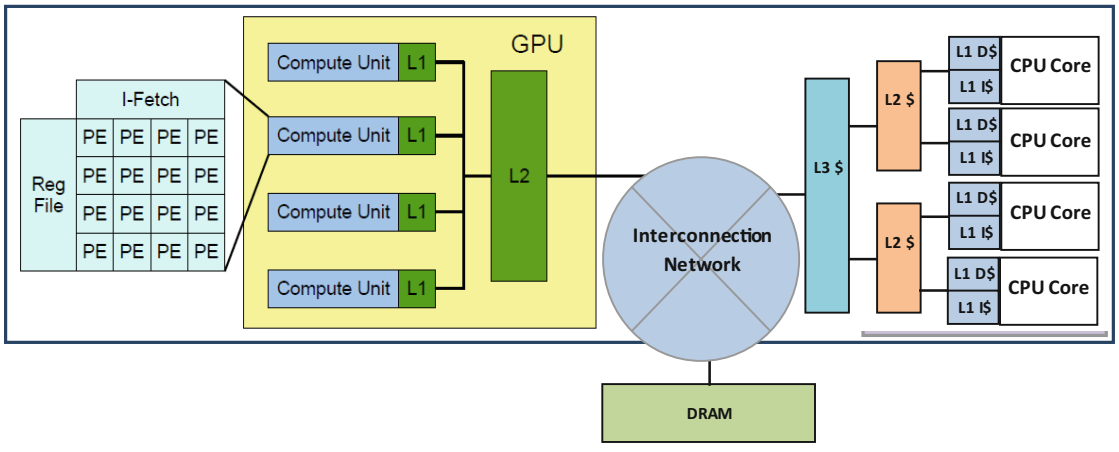


FIGURE 2 | A simple die map of an on-chip heterogeneous CPU–GPU processor.



(a)



(b)

FIGURE 3 | Heterogeneous CPU and GPU architectures (a) discrete, (b) on-chip integrated structures.

an on-chip integrated heterogeneous CPU–GPU architecture (Figure 3b). Each CPU core contains its own private instruction and data caches (L1-I and L1-D) while an L2 cache is shared by all cores. In discrete heterogeneous architectures, CPUs and GPUs are placed on separated chips and need distinct memory spaces communicating through PCI Express link with limited bandwidth. Off-chip interconnections increase delay and power consumption considerably. This problem is highly alleviated by integrating CPUs and GPUs on the same chip using an efficient heterogeneous network infrastructure (Figure 3b).

The memory structure of almost all heterogeneous CPU–GPU architectures is similar to the AMD Fusion architectures; the main memory address space is separated into two partitions: one partition is the host memory and controlled by the operating system, and the device memory is managed by GPU kernels. Due to shared memory space, contentions between CPUs and GPUs in heterogeneous systems result penalties. The main contention penalties are delay and power consumption [102]. These penalties are considerably mitigated by an efficient run-time task scheduler [14, 103–109]. Heterogeneous CPU–GPU architectures face several challenges particularly because of the limited shared resources and chip area, and also, because of scheduling a large number of parallel tasks that should be concurrently executed. The main challenges of heterogeneous CPU–GPU architectures are discussed in the following section.

4 | Challenges of Heterogeneous CPU–GPU Architectures

This section concentrates on the main challenges of cooperating of CPU and GPU cores. We have classified these challenges by reviewing related literatures on heterogeneous computing and architectures. The challenges of designing heterogeneous CPU–GPU architectures and consequently, the challenges of designing their simulators originate from inherent differences of CPUs and GPUs.

The main challenges of heterogeneous CPU–GPU architectures are as follows:

1. Shared resources management;
2. Network interconnection;
3. Task management and scheduling;
4. Energy consumption;
5. Programming model.

4.1 | Shared Resources Management

Data movements between the CPUs and GPUs and on-chip or off-chip memory impose a major bottleneck in overall performance. Shared memory modules reduce data transfer overheads but due to different memory access patterns of heterogeneous cores, contention and load balancing issues are increased [37, 102, 110]. One important challenge of heterogeneous CPU–GPU architectures is optimal utilizing of various on-chip or off-chip shared resources, mainly memories, without hindering the performance [12, 32, 36, 50, 53, 57, 111]. Cache accesses of CPU workloads are inherently different from GPU applications. GPU computing needs a huge amount of memory requests and high memory bandwidth, because GPU applications have huge amount of data-parallelism. On the other hand, the CPUs with high-capacity caches at different levels require strong coherency and consistency. Most CPU cores only have a few concurrent threads so they do not create a lot of memory requests simultaneously. As a result, integrating CPUs and GPUs, with different memory access requirements, introduces special memory management policies. Recent heterogeneous CPU–GPU processors provide hardware CPU–GPU shared virtual memory, atomic operations, and memory coherency for efficient management of shared resources, synchronization and coherency [36, 54, 56, 112–116].

In heterogeneous CPU–GPU architectures, cores share large LLCs, and hence, it is important to improve the efficiency of LLC. Without carefully design of LLC management, data transmissions between CPUs and GPUs cause considerable performance loss [53, 55, 56]. The problem of shared LLC can be mitigated using non-volatile memories such as spin-transfer torque RAM that offer higher cache capacity and near-zero leakage power consumption [117]. A systematic review on memory management techniques in heterogeneous CPU–GPU architectures was presented in [37]. The survey includes an analysis of various DRAM architectures, different memory-centric heterogeneous architectures and processing-in-memory systems.

4.2 | Network Interconnection

The interconnection structure, for connecting various on-chip or off-chip cores and shared memory modules, is the most important issue in designing heterogeneous CPU–GPU architectures, and consequently, in designing their simulators. Obviously, efficient and well-designed network interconnection significantly affects the performance metrics such as latency, power consumption, dependability, reliability, throughput, and cost of heterogeneous architectures [87–94, 118–125].

Traditional interconnection structure of system-on-chip architectures (SoCs), such as point-to-point and buses, are not scalable and efficient for many-core heterogeneous systems. As the number of cores increases, point-to-point interconnections impose a large amount of cost and overhead to the architecture. A bus-based network for connecting various cores, due to the parasitic capacitance and arbitration complexity, has severe limitations [126]. Instead of these communication fabrics, NoCs are much more efficient for interconnecting CPUs, GPU cores and shared memory modules [87–94, 118–124]. Figure 4 shows a mesh-based 2D NoC configuration of heterogeneous CPU–GPU architectures. The circles, labeled with *R*, are NoC routers. The processing and storage cores including CPUs, GPUs, memory controllers and LLCs are connected to their adjacent routers. The routers communicate with each other through links that can be wired, wireless or photonic lines.

Wired 2D NoC architectures have constrains including long transmission latency, high power consumption, and low bandwidth, especially for collective communications. Therefore, alternative approaches such as photonic interconnections, wireless network-on-chips (WNoCs) and 3D NoCs were presented to reduce latency and power dissipation, and to increase throughput [93, 98, 99, 124, 127–134]. Photonic interconnections provide high bandwidth, low latency, and energy-efficient architectures but with high implementation costs [93, 128]. WNoCs provide high transmission performance and low-power architectures; therefore, hybrid wired/wireless NoCs have the potential of being customized for various heterogeneous CPU–GPU architectures [21, 98, 99, 129–134].

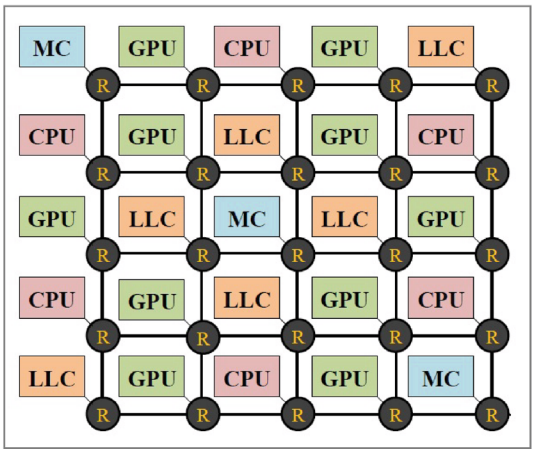


FIGURE 4 | An instance configuration of a heterogeneous CPU–GPU architecture with a mesh NoC.

NoC topology can be 2D or 3D structures in form of mesh, torus, ring, tree, honeycomb, and custom-designed configurations. Traditionally, mesh topology is simple, scalable, and regular, therefore mesh-based NoCs provide higher performance [87–91, 95, 96, 120–124, 131, 135, 136] rather than other 2D topologies. Off course, it has been demonstrated that NoCs with honeycomb topology [132–134, 137] outperform mesh-based NoCs in terms of throughput, latency, and energy consumption. Furthermore, the amount of hardware resources of NoC affects the performance of heterogeneous CPU–GPU architectures [87, 88, 134].

In addition to topology, the routing algorithm and the router structure have direct effect on the NoC performance [87, 88, 121–123, 129]. Various adaptive and semi-adaptive congestion-aware routing algorithms have been proposed [87, 88, 121–123, 132–134, 137]. The NoC routing algorithms based on fuzzy-logic [121, 123, 137] and metaheuristic methods [132–134] present high performance and low-power interconnection structure for heterogeneous architectures. Moreover, configurable, distributed and hardware-efficient microarchitecture of NoC routers provides low-power and high performance NoC architectures [87, 121, 122, 130, 134]. HiFMP [87], is a multicrossbar prioritized NoC router that provides configurable low-cost and low-power NoC architecture for efficient implementing on ASIC and FPGA devices.

Different techniques were presented to reduce NoC power consumption [87, 88, 129, 130, 132–134, 137]. Power-gating is the most efficient strategy for reducing the power consumption of NoCs. Using this strategy, according to the NoC state and packet distribution, idle modules are put into sleep mode. Slept modules are turned on when it is required. Most of the recent works focus on configurability and versatility of NoCs in order to design high performance heterogeneous communication infrastructure. Moreover, in heterogeneous NoC architectures, using adaptive and dynamic components is effective to exploit all the computing resources for impressive processing of data-intensive applications. In adaptive NoCs, voltage level, frequency and energy consumption are managed according to the network load and activity of components [87–91, 102, 120, 130, 135, 138].

4.3 | Task Management and Scheduling

Although heterogeneous architectures, with architecturally different processing cores, provide high performance and energy-efficient architectures, scheduling and task allocating on appropriate cores are very challenging. The reasons are different features and capabilities of heterogeneous cores, various workload characteristics, and also, large amount of parallel tasks, with different memory access patterns, that should be executed concurrently [2, 14, 27, 103–109, 139–154]. Task schedulers are software or software/hardware run-time modules attached to heterogeneous CPU–GPU architectures. These modules dynamically cooperate with the operating systems, compilers and also with the simulators.

Executing multiple tasks targeted CPUs or GPUs in unpredictable environments has leaded to self-adaptive and configurable systems that dynamically reorganize the optimal use of resources. The topic of task management and scheduling in heterogeneous CPU–GPU architectures has been explored in various research works. Summary of instances are presented here. These research works tackle the problems of run-time resource management for a wide range of heterogeneous systems and applications. AIRA [141] is a compiler and run-time for dynamic heterogeneous task mapping and flexible executing of applications. A specialized run-time system for accelerating data analytics on heterogeneous CPU–GPU architectures was presented in [142]. A contention-aware scheduling system for integrated CPU–GPU systems was explored in [143], which incorporates instrumentation-driven optimizations to improve the throughput. A self-synchronizing heterogeneous task scheduler was presented in [150]. SnuCL [151] is a framework for executing OpenCL kernels in heterogeneous clusters. A theoretical framework using StarPU for task mapping on heterogeneous CPU–GPU was presented in [152]. Nanos++, the run-time system of OmpSs programming model, is a framework for managing CPUs and GPUs by composing GPU kernels into a task graph. The scheduler of OmpSs can be implemented in hardware that provides less latency and less power consumption than the software run-time scheduler [155, 156].

4.4 | Energy Consumption

In heterogeneous CPU–GPU architectures, energy-efficiency is a key design goal because it affects the cost, dependability, maintainability, and fault tolerance of the whole devices for utilizing in diverse embedded systems such as IoT-based and distributed wireless systems [43, 47, 102, 129, 146, 148, 149, 157–161]. GPU cores utilize many fine-grained processing elements to concurrently execute on multiple data sets in an energy-efficient approach. With larger data bus, the energy consumption of the heterogeneous architectures decreases because of better exploitation of the GPU data-parallel resources. There are few works concentrating on reducing energy consumption of heterogeneous CPU–GPU architectures such as controlling task sizes and reducing redundant data transfer between CPU and GPU core [12, 20, 35, 146, 148, 162–165]. A survey on the various techniques for high performance and energy-efficient CPU–GPU cooperative computing was presented in [35]. A comprehensive energy management framework for heterogeneous CPU–GPU architectures was developed in [162]. An low-power AES algorithm for parallel executing on heterogeneous CPU–GPU architecture was presented in [165]. Power saving techniques for WNoCs was systematically analyzed in [129]; these techniques are applicable in designing energy-efficient heterogeneous CPU–GPU architectures.

4.5 | Programming Model

The programming model of heterogeneous applications is a challenge due to different kinds of ISAs and execution models of GPUs and CPUs [31, 166–168]. Basically, execution of GPU programming models, such as OpenCL, CUDA and HIP [169], is very different from the execution of CPU parallel programming models, such as OpenMP [170] and OmpSs. Moreover, in designing heterogeneous CPU–GPU architectures and their schedulers, workload features such as amount and levels of parallelism, presence of branch divergence, data dependency analysis are

important [3, 51, 115, 168, 171–174]. Most of the applications for executing in heterogeneous systems are C-based, combining CUDA or OpenCL with one of the parallel programming models such as OpenMP, MPI or OmpSs. In addition, SYCL [175] is a standard and portable code for every type of heterogeneous system especially for heterogeneous CPU–GPU architectures including Intel GPU, NVIDIA and AMD architectures, using the same syntax and the same compiler. Fang et al. [168] presented a comprehensive review on heterogeneous parallel programming models and the compiling techniques of improving programmability and portability for minimizing the communicating overhead. AutoNUSC [31] is a heterogeneous parallel programming model and runtime framework for task scheduling, data division, node communication, and fault-tolerant recovery.

4.6 | Discussions

CPU and GPU cores have their own specific characteristics and strengths; hence, cooperation of these cores provides high performance architectures. In designing heterogeneous CPU–GPU architectures and their simulators, it is important to consider constrains and limitations of processing units such as bandwidth and levels of memory modules, number of GPU threads and number of CPU cores. Furthermore, number of cycles and amount of data blocks that are read from or written to memory modules by CPUs or GPUs are not equal and not coherent with respect to each other. Consequently, CPU and GPU cores in heterogeneous architectures, possess diametrically opposite network demands. Therefore, several critical challenges such as task allocation, shared resource management, on-chip interconnection, dependability, and energy consumption optimization still require to be further investigated. Addressing each of these challenges will help to improve overall performance of heterogeneous CPU–GPU architectures. Table 1 presents the solutions for improvement the performance and functionality of heterogeneous CPU–GPU architectures for each challenge.

TABLE 1 | Challenges and the solutions for improvement.

Shared resources management

- Using hardware shared virtual memory, atomic operations, and memory coherency for efficient management of shared resources, synchronization and cache coherency [36, 53–56, 112–116].
- Using non-volatile memories such as spin-transfer torque RAM for offering higher cache capacity and near-zero leakage power consumption [117].

Network interconnection

- Efficient topologies such as mesh [87–94, 118–124] or honeycomb [132–134, 137].
- Photonic interconnections [93, 128] or WNoCs (hybrid wireless/wired NoCs) [21, 98, 99, 129–134].
- Reducing the amount of hardware resources of NoC [87, 88, 134].
- Efficient adaptive congestion-aware routing algorithms and configurable routers [87, 88, 121–123, 129, 132–134, 137].
- Using fuzzy-logic [121, 123, 137] and metaheuristic methods [132–134].
- Configurable NoC Architectures [87–91, 102, 120–122, 130, 134, 135, 138].

Task management and scheduling

- Dynamic and self-synchronizing heterogeneous task mapping and allocation [141, 142, 151, 152].
- Contention-aware scheduling [143].
- Software run-time scheduler such as Nanos++ [12, 155, 156].

Energy consumption

- Using scheduling and allocation algorithms for small tasks, controlling the size of tasks and also reducing redundant data transferring between CPU and GPU core [12, 20, 35, 146, 148, 162–165].
- Different techniques such as power-gating, voltage and frequency scaling, clock-gating for reducing NoC power consumption [87, 88, 129, 130, 132–134, 137].

Programming model

- Combining CUDA or OpenCL with one of the parallel programming models such as OpenMP, MPI or OmpSs.
- Using SYCL as a standard and portable code for every type of heterogeneous system.

5 | Heterogeneous CPU–GPU Simulation Frameworks

To explore and validate the efficiency of different aspects of heterogeneous CPU–GPU architectures, researchers have developed several simulating frameworks. Cycle-accurate simulators are essential for computer architects and researchers to evaluate and optimize their designs at functional and architectural levels. Moreover, the simulators are used for workload characterizations and design space exploration of hardware/software co-design. Heterogeneous simulators include various units with different inherent characteristics that work cooperatively to execute a heterogeneous application effectively. These simulators usually integrate one of the existing CPU simulation frameworks with one of the existing GPU simulators.

Figure 5 illustrates the main modules of heterogeneous CPU–GPU simulators. Heterogeneous CPU–GPU simulators should be able to emulate different modules for modeling, analyzing and exploring various aspects of emerging heterogeneous architectures. These modules include multi-core CPU simulator, many-core GPU simulator, reconfigurable network interconnection simulator, task scheduling system, simulator for off-chip memory accesses, cache memory hierarchy models, coherency protocols, and energy consumption analyzer and profiler. The heterogeneous CPU–GPU simulators were developed in one of the following forms: a single and unified CPU–GPU cycle-accurate simulator; or an integration of GPU simulators and CPU simulators along with network simulators and energy consumption models. Many CPU simulators as well as a lot of GPU simulators have been developed and employed by researchers, but just a few cycle-accurate heterogeneous CPU–GPU simulators have been presented.

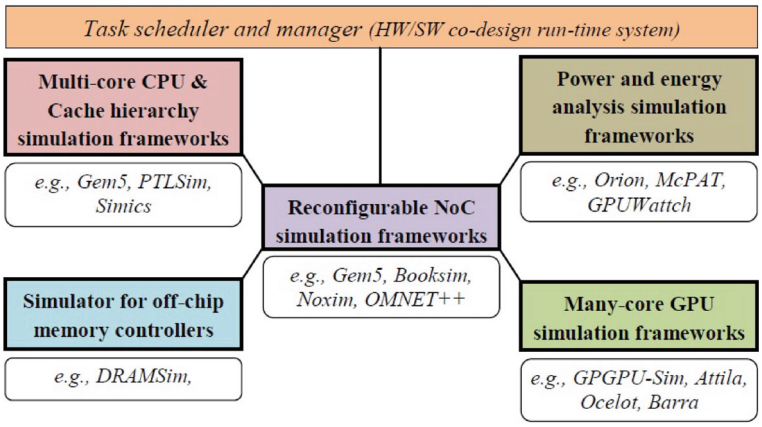


FIGURE 5 | Overall view of a heterogeneous CPU–GPU simulation framework.

Main instances of heterogeneous CPU–GPU simulators are as follows. Gem5-GPU [38] and FusionSim [39, 176] both integrate GPGPU-Sim with a cycle-accurate CPU simulator. MacSim [41] is another timing model and trace driven cycle-level simulator of heterogeneous architectures which can simulate ×86 and NVIDIA PTX traces simultaneously. MCMG [40] is a CPU–GPU simulator integrates Attila and Gem5 simulation frameworks. Multi2Sim [42] is an open-source simulator that supports ISA-level simulation of an ×86 CPU and an AMD Evergreen GPU. Multi2Sim is a unified CPU–GPU cycle accurate simulator which does not integrate any existing simulation frameworks. An analysis of Gem5-GPU and Multi2Sim compared with Gem5 was presented in [177].

Besides the aforementioned heterogeneous simulators, there are some other heterogeneous CPU–GPU platforms [5, 28, 36, 84, 178–185] which are not vastly used, or are not as mature and complete as the above mentioned simulators. For instance, SCHP [180] integrates Gem5 for modeling CPUs, and GPGPU-Sim for emulating GPUs. The CPU and GPU simulators of SCHP are concurrently executing on independent threads. GLTraceSim [181] is a memory tracing framework for studying the behavior of graphics workloads in heterogeneous CPU–GPU systems. In addition, Wang et al. [182] presented a full system heterogeneous CPU–GPU simulator. Gera et al. [84] presented an open-source microarchitectural simulator with trace generation for Intel's integrated GPUs. Furthermore, a full-system simulator for mobile heterogeneous CPU–GPU architectures was developed in [183] which enables users to run a complete and unmodified OpenCL programs of mobile ARM CPU and Mali-G71 GPU powered device. An approach for heterogeneous parallel Simulink models of image processing algorithm was presented in [28]. HEROv2 [184] is an open-source FPGA-based heterogeneous CPU–GPU simulator that includes a configurable NoC, a unified heterogeneous programming interface, and a mixed-ISA heterogeneous compiler based on LLVM. Guo et al. [185] presented an efficient static timing analysis engine for heterogeneous CPU–GPU computing by developing high-performance GPU kernels and data structures. A heterogeneous computing platform was presented in [5] for non-experts to deploy, run, and evaluate high-computational-demand algorithms. Simeuro [19] is a system-level heterogeneous CPU–GPU simulator for neuromorphic chips for accelerating the simulations of DNNs on huge datasets. Simeuro consists of two major sections: the neurosynaptic core simulator executing on GPUs and the NoC simulator runs on CPU cores.

6 | Examples of Heterogeneous CPU–GPU Simulators

In this section, the structure and features of the main instances of heterogeneous CPU–GPU simulators are described. These simulators are able to emulate the overall heterogeneous CPU–GPU architectures including CPUs, GPUs, various memory modules, NoCs and power consumption models. Then, in Section 6.6, different characteristics of these simulation frameworks are compared.

6.1 | Fusionsim

FusionSim [39, 176] is a heterogeneous cycle-accurate simulation framework for concurrently modeling of both GPUs and ×86-based out-of-order CPUs. FusionSim is built on PTLsim and GPGPU-Sim simulators. In addition, in FusionSim, the cache hierarchy is simulated by a configurable cache system called MARSSx86 [186] and the heterogeneous NoC infrastructure is simulated by Intersim tool [187]. FusionSim models ordering of memory accesses, overlaps synchronous and asynchronous operations, manages memory transfers and processing concurrent tasks. For communicating between CPU and GPU cores, the global address space of GPUs is placed in the memory address space of CPUs.

FusionSim has two modes: (1) discrete mode where CPUs and GPUs connect as disjoint units via traditional PCI Express bus for simulating discrete heterogeneous architectures (see Figure 6a); and (2) fused mode where one CPU and several GPUs are integrated on the same die (see Figure 6b). In the fused mode, one of the processing clusters of GPGPU-Sim is substituted with the PTLsim CPU core and its cache memories; hence, only one CPU can be used. This feature limits heterogeneous system architectural exploration. FusionSim has been used for modeling and analysis heterogeneous architectures in some researches [84, 188–190].

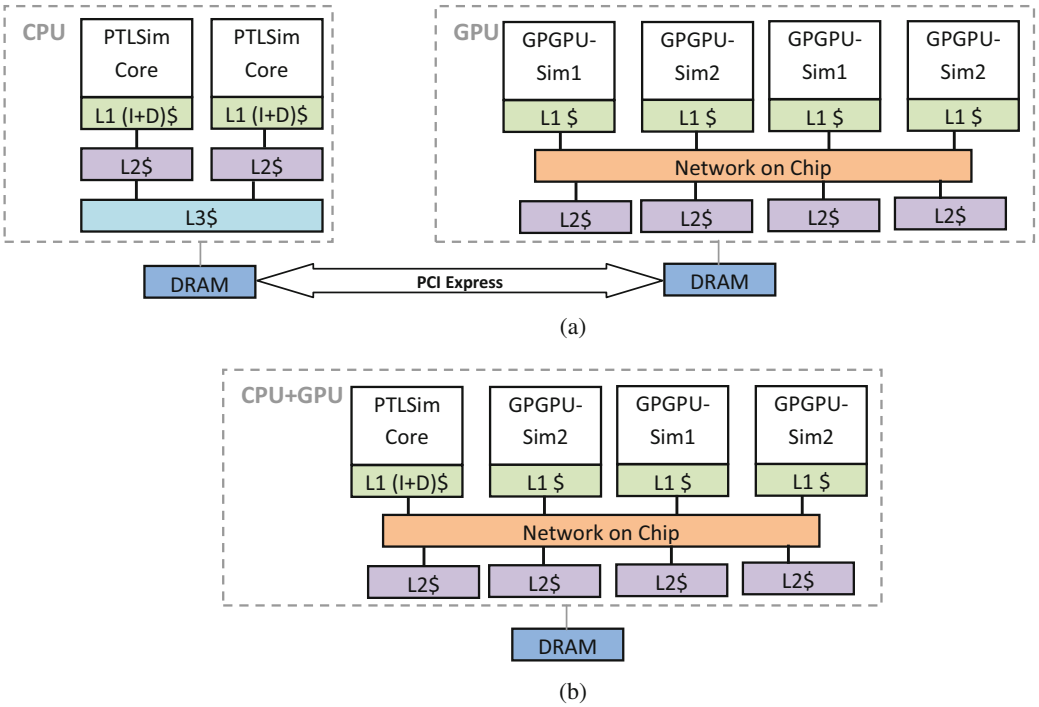


FIGURE 6 | (a) Discrete FusionSim, (b) fused FusionSim.

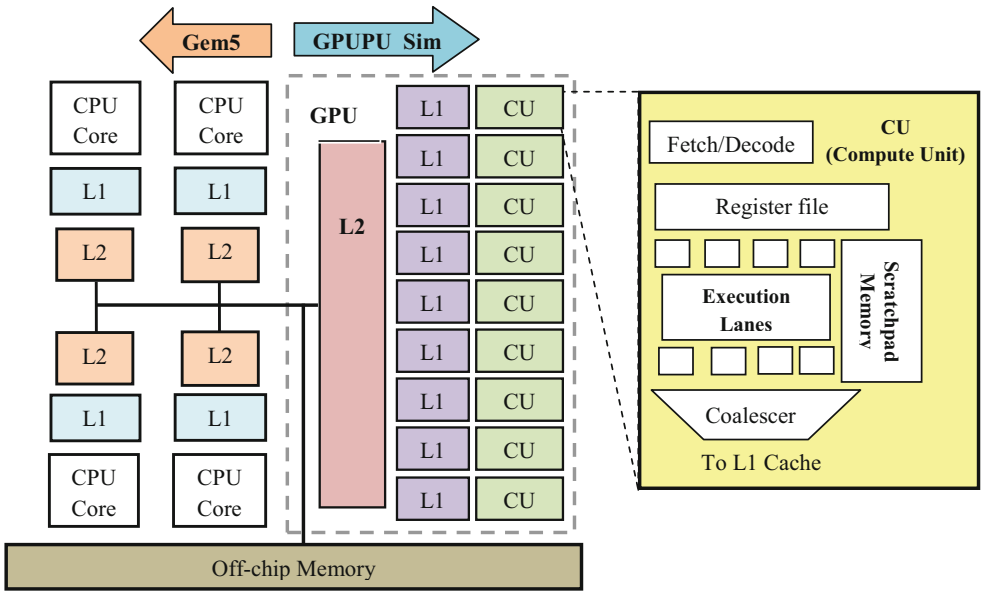


FIGURE 7 | Gem5-GPU block diagram.

6.2 | Gem5-GPU

Gem5-GPU [38] is a heterogeneous CPU–GPU simulation framework according to Gem5 and GPGPU-Sim platforms. It has been developed by the same researchers that provided Gem5. The structure of Gem5-GPU is illustrated in Figure 7.

Figure 8 illustrates the Gem5-GPU software architecture including the logical copy engine for transferring data from CPU to GPU memories in a split virtual memory mode. Similar to Gem5 simulator, Gem5-GPU manages its configurable memory accesses using Ruby. Gem5-GPU emulates shared and/or discrete virtual address spaces between CPU and GPU cores in order to simulate diverse system configurations such as systems with single virtual address space and systems with separated GPU and CPU real address spaces. In Gem5-GPU, the number of CPUs, the number of GPUs, memory organization and interconnections are completely configurable.

Several performance analysis of heterogeneous architectures using Gem5-GPU platform have been done [97, 100, 114, 120, 191–198]. For instance, using this simulator, Choi et al.

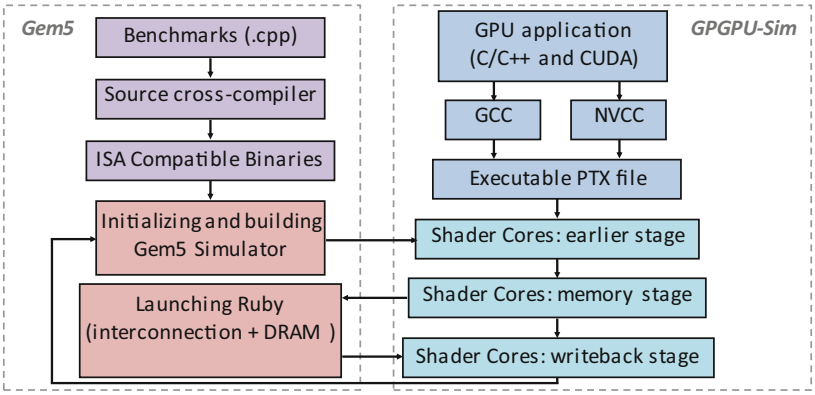


FIGURE 8 | Gem5-GPGPU software architecture.

[100] explored an energy-efficient and specialized heterogeneous CPU–GPU many-core architecture for training convolutional neural networks. Gao et al. [114] presented a dynamic cache replacement and migration policy for heterogeneous CPU–GPU architectures. While, by default, Gem5-GPU execute the CUDA applications, Wang et al. [192] presented an extension for Gem5-GPU to support OpenCL programming model. In [120], heuristic-based optimization approaches were developed and explored using Gem5-GPU to obtain a near-optimal low power heterogeneous NoC architecture. A specialized heterogeneous CPU–GPU system for energy-efficient training of CNNs using Gem5-GPU was presented in [100].

6.3 | MacSim

MacSim [41] is a heterogeneous trace-driven and cycle-accurate simulator. It models different architectural behaviors, including in-order and out-of-order architectures, multi-threading, cache and memory structures, memory controller and NoC infrastructure. Figure 9 illustrates the structure of MacSim. This simulator is able to simulate homogeneous and heterogeneous ISA architectures. Multiple MacSim simulators can run concurrently. This framework uses ×86 traces created by Pin and PTX GPU traces generated by Ocelot. MacSim implements a five-stage processor including fetch, decode, schedule, execute and retire. In MacSim, the GPU includes various streaming multi-processors (SMs), each including multiple scalar processors (SPs) and specific function units (SFUs), and also, a multi-threaded fetch and issue unit, a cache memory for constants, and a shared memory. The hierarchical memory structure of MacSim is very flexible and configurable.

For simulating accesses to DRAM, DRAMSim is incorporated to MacSim. Three different configurations of connection between MacSim and DRAMSim are shown in Figure 10. As the figure shows, different network configurations based on IRIS [199] model such as point-to-point, bus and NoC structure are possible to configure the interconnection between DRAMSim and one or more MacSim. IRIS is a portable framework for cross-platform heterogeneous architectures. Using shared virtual memory and automatic resizing tasks, IRIS manage various resources and different programming models at the same time for efficient execution on heterogeneous architectures.

An example of using MacSim is the research of [89] which is a design space exploration of NoCs in heterogeneous CPU–GPU architectures. Another example is an exploration on interconnections with MacSim was done in [118]. Another research that has used MacSim is an approach for a heterogeneous 3D NoC architecture [124]. In [200], using MacSim, an analysis of LLC and memory controller in heterogeneous CPU–GPU architecture was presented.

6.4 | MCMG

MCMG [40] is a framework for simulating heterogeneous CPU–GPU systems. It integrates Attila GPU simulator [70] with Gem5 CPU simulator. Figure 11 shows the general structure of MCMG framework. In this framework, the number of CPU and GPU cores and also levels of cache hierarchy are configurable. The heterogeneous cores and LLC are connected by a bus interconnection. The LLC is connected to off-chip DRAM module. MCMG is a Linux-based platform that runs OpenGL applications, and offers many instrumentation features for analyzing workloads and hardware characteristics. In MCMG, the GPUs run independently from CPUs, and communicate with the other parts through a bus adapter that includes two agencies reside in Attila for GPUs and Gem5 in CPUs.

6.5 | Multi2Sim

Multi2Sim [42] is a heterogeneous CPU–GPU simulator at ISA-level. This project was started as an open-source, cycle-accurate CPU simulator in 2007 [201]. Then, in 2012 it was extended as cycle-accurate heterogeneous CPU–GPU simulation platform. Multi2Sim supports simulation of different CPUs such as ARM, ×86 and MIPS, and various GPUs like NVIDIA Fermi and Kepler, AMD Evergreen, and Southern Islands. When an application is started to be executed on this simulator, the host parts of the application are executed on the CPU simulation modules. When CUDA or OpenCL API calls are met, the GPU simulator starts to execute the kernels. Figure 12 shows the comparison of running a program on Multi2Sim simulator and a native AMD heterogeneous system.

The CPU simulation part of Multi2Sim is composed of the functional and the architectural simulators. The functional part is

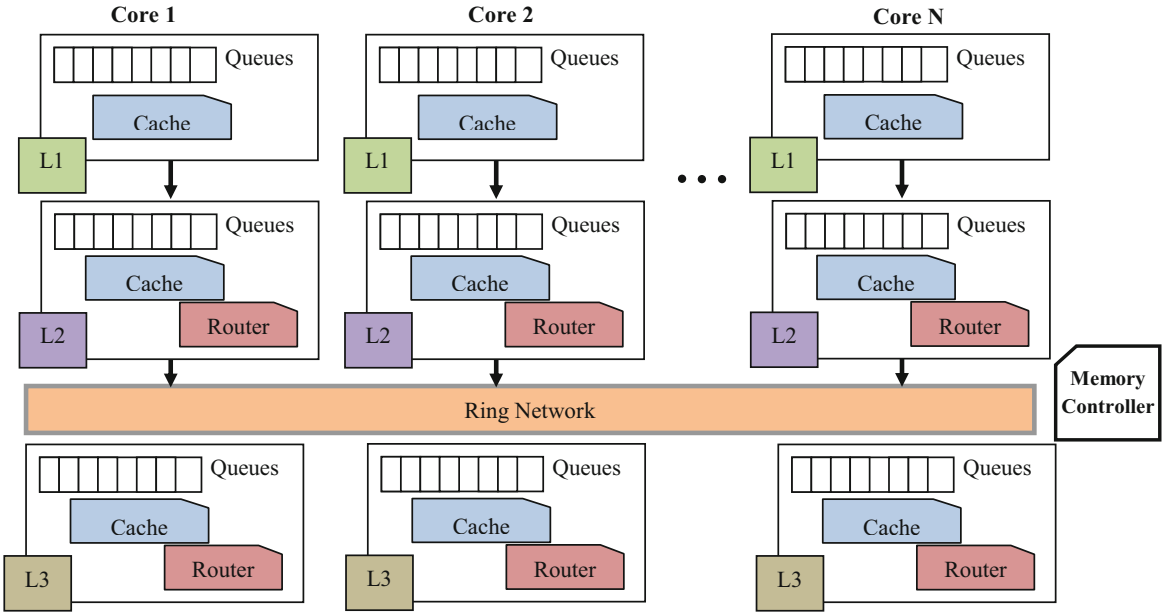


FIGURE 9 | Structure of MacSim.

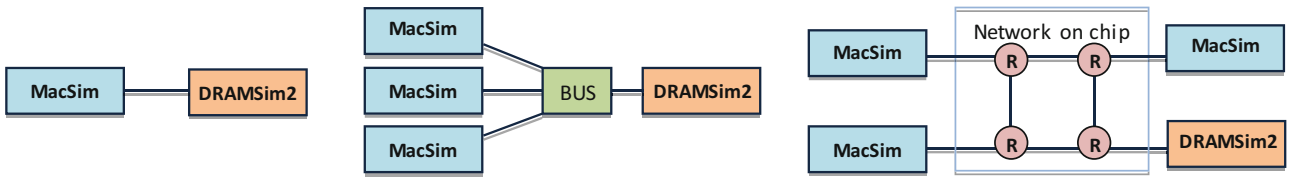


FIGURE 10 | Different configurations of MacSim with DRAMSim.

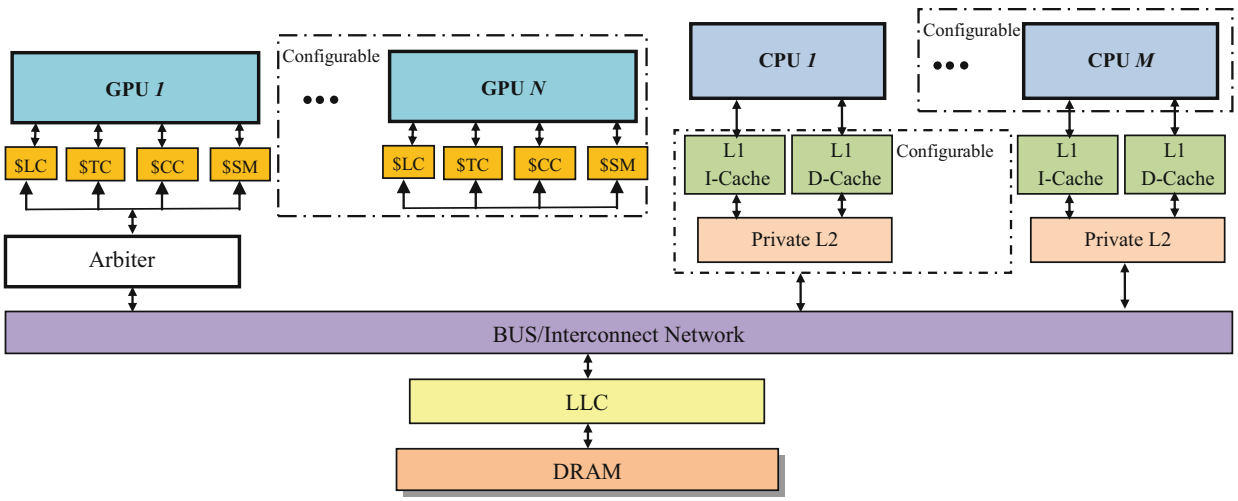


FIGURE 11 | Structure of MCMG framework.

similar to the execution of a program on a native ×86 processor. The architectural part gets ×86 traces from the functional part, and tracks execution cycle-by-cycle. The ×86 CPU model of Multi2Sim is a customizable architecture and is not according to any of the existing architectures. It is highly configurable for simulating different heterogeneous CPU–GPU architectures. It is capable to run OpenCL and CUDA applications, without any source code modifications, by launching host and device binaries. M2S-Visual is a visual extension tool of Multi2Sim that provides the visual presentation of running instructions and the states of CPU and GPU entities, memory hierarchy and also interconnection entities. M2S-Visual tool consists of plots that reveal the activities of various modules, memory access patterns and the network traffic according to static and dynamic

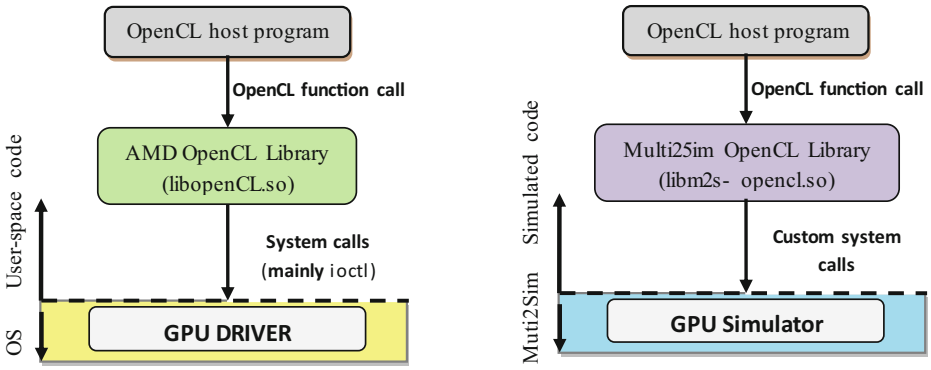


FIGURE 12 | Running a program on a native AMD heterogeneous system (left) compared to Multi2Sim simulator (right).

features of the running kernels after the application execution is complete.

An emulated approach for on-chip and off-chip GPU memory modules using Multi2Sim was proposed in [202]. A memory analyzer for heterogeneous architectures was presented in [178], which was designed based on Multi2Sim. M2S-CGM [203, 204] is an architectural simulator for coherent CPU–GPU systems that designed based on Multi2Sim with a custom-built memory system model. A wireless NoC to provide energy-efficient, high-performance communication for CPU–GPU heterogeneous multi-chip systems was presented in [205] and was analyzed with Multi2Sim. A research on cache optimization for CPU–GPU heterogeneous architectures using Multi2Sim was presented in [206].

6.6 | Comparison

In this section, the described CPU–GPU simulators are compared from different aspects. General information of the heterogeneous CPU–GPU simulators are presented in Table 2. All the simulators have been developed since year 2012, and their source codes are available. The user/product supports for Gem5-GPU are very high while they are low for FusionSim and MCMG. All of the simulators accept binary files for input except MacSim. In addition to binary file and traces, Gem5-GPU accepts unmodified CUDA or OpenCL source code without modifications. As the table shows, all of these simulators are able to execute CUDA and OpenCL applications in the form of binary executable or traces. The only exception is MCMG which supports OpenGL APIs.

6.6.1 | Capabilities

Table 3 presents a comparison of the heterogeneous CPU–GPU simulators. The table shows the CPU and GPU simulators that used for forming the heterogeneous simulation platforms. FusionSim, Gem5-GPU and MCMG were developed by integrating two existing frameworks, one CPU simulator and one GPU simulator. MacSim is basically a trace-driven simulator, and obtains the traces from two existing tools: Pin for ×86 traces and Ocelot for PTX traces. In MacSim, CPU and GPU traces are executed simultaneously under control of a common scheduler. Multi2Sim is the only unified heterogeneous framework which is not an integration of any existing frameworks and is highly configurable. Therefore, we used customizable terms for this simulator in the table.

MacSim simulates an integrated CPU–GPU architecture with shared LLC by executing the CPU and GPU traces separately. FusionSim replaces one of the processing clusters in the GPU (GPGPU-Sim) with a CPU core (PTLsim). Multi2Sim models ×86 CPU cores and AMD Evergreen GPUs at ISA level with OpenCL supports. However, it cannot execute CPU and GPU applications concurrently. Multi2Sim is an ISA-level CPU–GPU heterogeneous framework simulator. In this simulator, when executing OpenCL programs, the host (i.e., CPU) portions of the program are run using the CPU simulation modules. When OpenCL API calls are encountered, they are intercepted and used to setup or begin GPU simulation. Multiple concurrent executions mode of Gem5-GPU and also MacSim frameworks is possible. In most of these frameworks, such as Gem5-GPU and FusionSim, CPU and GPU modules run concurrently, but the simulation processes in Multi2Sim are sequential.

The table presents the CPU and GPU architectures that the platforms can modeled. If their default models are not the existing architecture, it is mentioned as custom. In addition, the table states the type of input entries, memory model, network model and power consumption analyzer of each simulator.

6.6.2 | Configurability Features

Configurability is an important feature for a simulation framework. As it was described in Section 5, heterogeneous CPU–GPU simulators include several components with different capabilities. These components should be flexible and configurable in diverse aspects for effective simulation and analysis. Number of CPU and GPU cores, cache hierarchy (number of cache levels, capacities, bandwidth and etc.), memory structure and accesses (number of cycles for reading, writing and waiting, width and depth of memory modules and etc.), NoC and other interconnection structures should be configurable (bandwidth, bit rate, buffer size, number of router ports, topology, routing algorithm and etc.). In Table 3 configurable components of each simulator are listed. Gem5-GPU and Multi2Sim are more configurable than the others. These frameworks provide

TABLE 2 | General information of the heterogeneous CPU–GPU simulators.

Heterogeneous simulator	FusionSim [39, 176]	Gem5-GPU [38]	MacSim [41]	MCMG [40]	Multi2Sim [42]
Year	2012	2014	2012	2015	2012
Github address	(1) https://github.com/kevjosey/fusion-sim, (2) https://github.com/joulesmith/fusion-sim	https://github.com/gem5-gpu	https://github.com/gthparch/macsim	https://github.com/jkwang93/MCMG	https://github.com/Multi2Sim/
Support	Low	Very high	High	Low	High
Input	Binary files	Source code, binary files, traces	x86 CPU traces from Pin + PTX GPU traces from Ocelot	Binary files	Binary files and traces
GPU languages	CUDA and OpenCL	CUDA and OpenCL	CUDA and OpenCL	OpenGL	CUDA and OpenCL
Simulation model	Execution-driven	Execution-driven + Trace-driven	Trace-driven	Execution-driven	Execution-driven + Trace-driven

many options for simulating and analyzing various features of incoming heterogeneous CPU–GPU architectures.

6.6.3 | Speed Capabilities

The simulators cannot be compared in terms of speed because their capabilities and configurability features are different. Hence, the speed comparison does not present any performance information. Some of these simulators should run more than two powerful and configurable simulation frameworks. Management simultaneous execution, memory management, scheduling, synchronization and data transfers between frameworks have effects on the speed of the more powerful and configurable simulators. Moreover, trace-driven and functional simulators are faster than the execution-driven full-system simulators due to the amount of processes and details that should be handled. Furthermore, the more configurable and more flexible NoC and memory simulation frameworks present more information on simulation but consume more time than a simple and static simulator that does not consider the details.

6.6.4 | Usability

In this section, the usability of the intended heterogeneous simulators by considering the number of citations of the main reference(s) of are discussed. Table 4 presents the citation details of the main reference(s) extracting from the valid research databases: Researchgate, IEEE Xplore, ACM Digital Library and SEMANTICscholar. In addition to total citations, SEMANTICscholar presents highly influential, background, methods and results citations. Since the main reference of MacSim is a user guide and there is not any published conference or journal article about MacSim, only Researchgate provides the number of citations to the user guide.

The usability raw in Table 4 presents the percentages of usage of each simulator for analyzing heterogeneous CPU–GPU architectures among the references of this article. As the table shows Gem5-GPU and Multi2Sim have been cited more than the others. Gem5-GPU is a comprehensive configurable simulator and has been used in a large number of researches for analysis the behaviors of the memory structure and accesses, cache hierarchy, network interconnection and core architectures of CPU–GPU heterogeneous architectures. Multi2Sim has been used vastly for analysis NoC and network architectures of CPU–GPU heterogeneous architectures. Compared to Gem5-GPU, Multi2Sim is simpler for usage and analysis. Also it is a unified configurable framework for simulating diverse CPU and GPU architectures.

For both of these simulators, code available to all researchers without restrictions. Gem5-GPU is a fully system framework that executes both the kernel-level and user-level instructions and emulates a complete system (i.e., OS, devices, exceptions, interrupts and I/O devices). Unlike Gem5-GPU, Multi2Sim has a functional simulator. Both of them can accept binary file or traces from various CUDA and OpenCL applications. Both of them are execution- and trace- driven. In addition, Gem5-GPU can run un-modified source code.

TABLE 3 | Comparison of the heterogeneous CPU–GPU simulators.

Heterogeneous simulator	FusionSim [39, 176]	Gem5-GPU [38]	MacSim [41]	MCMG [40]	Multi2Sim [42]
CPU simulator	PTLSim	Gem5	Pin	M5	Customizable simulator
GPU simulator	GPGPU-Sim	GPGPU-Sim	Ocelot	Attila	Customizable simulator
CPU architecture	×86 out-of-order	×86 out-of-order	×86 out-of-order	Customizable CPU architecture	Customizable CPU architecture
Number of CPUs	1	Several	Several	Several	Several
GPU architecture	Customizable GPU architecture	Customizable GPU architecture	NVIDIA Tesla and Fermi	Customizable GPU architecture	AMD EverGreen and Southern-Island, NVIDIA Kepler and Fermi
Number of GPUs	Several	Several	Several	Several	Several
Heterogeneity modes	Fused and discrete	Fused	Fused	Fused	Fused
Configurability	High	Very high	High	Medium	Very high
Configurable components	Number of GPUs, cache hierarchy, NoC	Number of GPUs and CPUs, CPU and GPU architecture, memory accesses model and cache hierarchy, NoC	Number of GPUs and CPUs, CPU architecture, cache hierarchy, NoC	Number of GPUs and CPUs, cache hierarchy	Number of GPUs and CPUs, CPU and GPU architecture, cache hierarchy, NoC and other interconnection structures
Memory model	MARSSX86 for all cache levels	Ruby of Gem5 for all cache levels	DRAMSim for all cache levels and DRAM	Simple Gem5 memory model	Customizable model
Network model	Intersim	NoC structure of Gem5	IRIS	Gem5 bus structure	Customizable model
Power analyzer	GPUWatch of GPGPU-Sim	GPUWatch of GPGPU-Sim+ Gem5 energy model	McPAT	Gem5 energy model	Customizable model
Network structure	PCI Express, NoC	Configurable NoC	Point-to-point, bus, NoC	Bus, simple NoC	Point-to-point, bus, configurable NoC

TABLE 4 | Citations details.

Heterogeneous simulator	FusionSim [39, 176]	Gem5-GPU [38]	MacSim [41]	MCMG [40]	Multi2Sim [42]
Researchgate citations	23	204	66	8	373
IEEE Xplore citations	10	149	—	8	26
ACM Digital Library citations	10	32	—	3	393
SEMANTICscholar total citations	20 + 10	229	—	9	464
SEMANTICscholar highly influential citations	2 + 1	14	—	—	54
SEMANTICscholar background citations	5 + 3	74	—	2	111
SEMANTICscholar methods citations	10 + 5	119	—	2	261
SEMANTICscholar result citations	1 + 0	—	—	—	4
Usability	13%	43%	15%	0%	29%

6.6.5 | Limitations

In this section, the limitations of the intended simulators are presented. Unlike the other simulators, FusionSim cannot run more than one CPU simulator, therefore it simulates only one CPU with several GPUs. MacSim, FusionSim, Multi2Sim and MCMG are able to run un-modified source code of the applications. Moreover, MacSim cannot run binary files as input. MCMG only supports OpenGL programs; it cannot run CUDA and OpenCL applications. Only FusionSim can simulate both fused and discrete CPU–GPU architectures, the others can simulate only fused architecture. Unlike the others, the configuration features of MCMG are very limited. The network interconnection and memory structure of MCMG are very simple compared with the others. As Gem5-GPU is a comprehensive framework, compared with the other intended simulators, it is more difficult to learn and configure this simulator.

7 | Recommendations

In this section, recommendations for designing high performance heterogeneous architectures and also for developing efficient and flexible heterogeneous CPU–GPU simulators are presented. Based on the challenges of heterogeneous CPU–GPU architectures, and also, according to various proposals and researches, the following recommendations are presented for designing flexible and efficient heterogeneous CPU–GPU simulators. These recommendations are in the direction of designing full-system, dependable and cycle-accurate simulators that cover all the architectural aspects of heterogeneous CPU–GPU architectures.

Researchers tend to use micro-architectural simulators for deep analysis, modeling, and simulating different aspects of their proposals. In order to address the challenges and to determine the requirements of future on-chip heterogeneous CPU–GPU architectures, parameterizable, reliable and flexible simulation frameworks are needed. As heterogeneous CPU–GPU architectures are growing more complex; their simulators should become more configurable and more flexible than the existing ones. Further researches and explorations on developing heterogeneous CPU–GPU simulators are required to be performed.

There are two approaches for developing a heterogeneous simulator. In the first approach, two or more existing simulators, one for CPU and one for GPU along with a network simulator, shared memory simulator and energy consumption analyzer are integrated. Most of the existing heterogeneous CPU–GPU simulators such as Gem5-GPU were designed based on this approach. In the other approach, a heterogeneous CPU–GPU simulator is designed from scratch as a standalone framework. Multi2Sim is the main instance of this approach.

The simulation methodology affects speed and amount of overhead caused during emulating, profiling and logging processes. Furthermore, the simulation approach influences precision, fidelity, accuracy and validity of simulation process. The key issue is providing parameterizable and configurable platforms with low run-time overheads. In designing a heterogeneous simulator, the methodology should be efficient, reproducible, reliable, extensible and deterministic. In addition, the simulation methodology should be able to utilize the capabilities of the real computing systems. It is recommended to use various strategies such as dynamic binary translation and parallel simulation in order to improve the speed and the efficiency of simulation procedure.

It is important to note that heterogeneous simulators should be able to execute CPU and GPU parts simultaneously for independent tasks. Moreover, they should be able to run several CPU cores along with several various GPU cores. Scalability, dependability and flexibility features of the simulator are also important especially for evaluation of critical run-time systems. It is significant to consider huge amount of state entities that should

be saved and processed during simulation procedure. The state of software entities includes work-groups, contexts, wave-fronts, and work-items, and also the memory entities. Furthermore, the network entities and the memory hierarchy states should be considered. Because of the different nature of schedulers from other modules of heterogeneous CPU–GPU architectures, the functionality of these modules is not completely embedded in the simulation frameworks. It is effective for heterogeneous simulators to join to a configurable task scheduler for simultaneous execution and synchronization.

7.1 | Recommendations for Shared Resources Management

Fully configurable and parameterized memory modules are essential for microarchitecture simulators especially for heterogeneous CPU–GPU simulators. In addition, the heterogeneous simulators should have various options for diverse memory structures. For instance, shared virtual memory, atomic operations, and memory coherency help to efficient manage different shared resources, synchronization and cache coherency.

7.2 | Recommendations for Network Interconnection

The most important challenge in designing and developing heterogeneous CPU–GPU simulators is modeling and managing accesses to shared resources through a common, high-speed and low-power network interconnection. The heterogeneous simulators can be constructed by integrating two or more existing simulators, or can be designed as a single unified heterogeneous simulator.

NoC infrastructure deals with the functionality of all components of the heterogeneous CPU–GPU architectures. NoCs directly affect the performance parameters such as speed, energy consumption, throughput, area, overhead, and even cost of the whole heterogeneous CPU–GPU architectures [34, 91, 96, 102, 120]. NoC architectures need to be well-adapted to the application requirements while providing high performance, dependable, energy-efficient and sustainable interconnection infrastructure. Therefore, simulating of NoC architectures is very important especially with heterogeneous processing elements. The NoC structure can be investigated using an adaptive and configurable NoC simulator as a part of a heterogeneous CPU–GPU simulator.

An efficient NoC structure, embedded in a heterogeneous CPU–GPU architecture, should dynamically balance the packet distributions using congestion-optimized interconnections and congestion-aware routing algorithms [24, 121, 123]. A reconfigurable network is effective when it is able to activate extra channels under high load and congestions, and also is able to shut them off when the network is unloaded [87, 90, 130]. A heterogeneous NoC should operate at nominal-voltage and high frequency under CPU-dominated low-load situations while under high-load and complicated traffic patterns of GPU applications, it should operate in near-threshold mode [90].

It has been demonstrated that most of the NoC architectures are not able to manage both CPU and GPU communication requirements efficiently because of different memory access patterns [100]. One solution is to divide the CPUs and GPU cores into two (or more) clusters connecting through a common NoC. Clustering strategies are used to schedule effectively and also to balance load and energy consumption to conserve the limited energy of the nodes and chips. In designing a task scheduler for heterogeneous CPU–GPU architectures, it should be taken into account that suitable applications for CPUs are very different from applications that be effectively executed on GPUs. In heterogeneous architectures, we have the chance to employ the high computational GPU strength for regular workloads, and transformative and flexible execution of CPUs for irregular control flow applications.

The NoC simulator in heterogeneous CPU–GPU simulators is a key feature. As it was described, the interconnection structure of the heterogeneous architectures exerts direct impact on performance and power consumption of the whole system. If the interconnection simulator is completely configurable and parameterized with many options, various circumstances would be prepared to investigate and evaluate the influences of diverse new proposals. For instance, it is highly effective that a NoC simulator has options for selecting wired, wireless or 3D NoC configurations, enabling power-gating mode, selecting buffer and bufferless routers. The simulator should support several adaptive and semi-adaptive NoC routing algorisms for investigating the NoC characteristics in order to find approaches for obtaining better performance and higher throughput. Another capability can be added to the heterogeneous simulator is machine learning approaches for bandwidth scaling and power supply scaling at each router within a specified reservation window. Therefore, the NoC model of a heterogeneous CPU–GPU simulator should be completely parameterized and configurable providing various options to investigate different aspects of heterogeneous architectures.

7.3 | Recommendations for Energy Consumption

In designing heterogeneous CPU–GPU architectures and developing their simulators, it is important to determine the most energy-hungry modules that consume a large amount of energy in heterogeneous CPU–GPU architectures. The simulator should model these modules specifically; then, the designer would be able to analyze them in details. The simulation framework should present various options for management and control and also for analysis the power consumption of different components of heterogeneous CPU–GPU architectures.

Determining a unified static and dynamic energy model, and also a framework for profiling and analysis power consumption of heterogeneous CPU–GPU architectures is very important. This is expected that the energy model of a heterogeneous simulator be able to analysis the energy consumption of the whole system including the energy consumption of CPUs, GPU cores, memory modules, links and interconnections and also run-time software/hardware scheduler system. It is noticeable that the obtained results of joining two or more energy

analysis tools may be incorrect or low accurate because each individual tool has its own estimation approach according to different energy models, different assumptions, and different simplifications.

7.4 | Recommendations for Programming Model

In heterogeneous architectures, the programming model challenge can be viewed from two points of view: (1) the programming model of the applications that are allocated to CPUs and GPUs, and (2) the source code of the simulation platform. However, the programming model of heterogeneous applications is not a critical challenge in designing a heterogeneous CPU–GPU simulator, because in the case of trace-driven simulator, ×86 traces and PTX traces are fed to the simulator regardless to the source code programming languages of the applications. In the cycle-accurate simulators that execute binary files of the applications, the input binary files for CPU and GPU are separately prepared before giving them to the simulator. The choice of programming languages of the source code of heterogeneous simulators is a key issue for software developers. Most of the existing simulation frameworks have been written with portable and flexible high-level languages such as C/C++. In integrating two or more simulators, MPI APIs have the capabilities to interface between GPU and CPU simulators by introducing features such as shared virtual memory and memory coherence [44, 45, 50].

8 | Conclusion

Heterogeneous CPU–GPU architectures provide low-latency and energy-efficient systems; hence, they are widely used in high performance emerging architectures, especially in critical and IoT-based embedded systems. The main reason of efficiency of these architectures is benefiting from the specific features of GPUs, unified address space, and configurable on-chip networks. Due to high computing potential and energy-efficiency of GPUs, they are vastly utilized as accelerators. Moreover, integration of GPUs and CPUs on one chip enables unified memory spaces including various memory modules communicating through an efficient interconnection structure such as NoC architectures. Configurable NoCs are able to improve the performance in terms of latency, throughput, fault tolerance, and energy efficiency.

The main objectives of this research are classifying and discussing the challenges of design heterogeneous CPU–GPU architectures and their simulators. Heterogeneous multi-core architectures can be discrete with off-chip GPUs or integrated CPU and GPU cores on the same chip. Off course, they are separated logically due to various programming models and different instruction sets. The significant conclusions of this article are as follows. From the hardware point of view, network interconnection for handling different data transferring between heterogeneous cores is the main challenge of designing both heterogeneous architectures and simulators. From the software point of view, task mapping and scheduling policies for accessing to the shared resources are the main challenges in designing heterogeneous CPU–GPU architectures. Usually, schedulers are developed in software and attached to heterogeneous CPU–GPU simulators.

Data Availability Statement

The data that support the findings of this study are available on request from the corresponding author. The data are not publicly available due to privacy or ethical restrictions.

References

1. F. Yazdanpanah, C. Alvarez, D. Jimenez-Gonzalez, and Y. Etsion, "Hybrid Dataflow/von-Neumann Architectures," *IEEE Transactions on Parallel and Distributed Systems* 25 (2014): 1489–1509.

2. G. Campeanu, J. Carlson, and S. Sentilles, "Component-Based Development of Embedded Systems With GPUs," *Journal of Systems and Software* 161 (2020): 110488.

3. Y. Cho, J. Park, F. Negele, C. Jo, T. R. Gross, and B. Egger, "Dopia: Online Parallelism Management for Integrated CPU/GPU Architectures," in *Proceedings of the 27th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming* (New York: ACM, 2022), 32–45.

4. S. Mittal and J. S. Vetter, "A Survey of CPU-GPU Heterogeneous Computing Techniques," *ACM Computing Surveys* 47, no. 4 (2015): 1–35.

5. J. Garcia-Hernandez, M. Morales-Sandoval, and E. Elizondo-Rodriguez, "A Flexible and General-Purpose Platform for Heterogeneous Computing," *Computation* 11 (2023): 97.

6. W. Li, T. Liu, Z. Xiao, H. Qi, W. Zhu, and J. Wang, "TCADer: A Tightly Coupled Accelerator Design Framework for Heterogeneous System With Hardware/Software Co-Design," *Journal of Systems Architecture* 136 (2023): 102822.

7. M. Riahi, A. Savadi, and M. Naghibzadeh, "Comparison of Analytical and ML-Based Models for Predicting CPU-GPU Data Transfer Time," *Computing* 102 (2020): 2099–2116.

8. V. Bui, T. Pham, H. Nguyen, H. Gia, and T. K. Mohd, "Heterogeneous Computing and the Real-World Applications," in *2021 IEEE 12th Annual Ubiquitous Computing, Electronics & Mobile Communication Conference (UEMCON)* (New York: IEEE, 2021), 0747–0751.

9. P. Song, Z. Zhang, Q. Zhang, L. Liang, and Q. Zhao, "Implementation of the CPU/GPU Hybrid Parallel Method of Characteristics Neutron Transport Calculation Using the Heterogeneous Cluster With Dynamic Workload Assignment," *Annals of Nuclear Energy* 135 (2019): 106957.

10. M. Gowanlock, "Hybrid KNN-Join: Parallel Nearest Neighbor Searches Exploiting CPU and GPU Architectural Features," *Journal of Parallel and Distributed Computing* 149 (2021): 119–137.

11. F. Su, C. Liu, and H. Stratigopoulos, "Testability and Dependability of AI Hardware: Survey, Trends, Challenges, and Perspectives," *IEEE Design & Test* 40 (2023): 8–58.

12. F. Yazdanpanah and M. Alaei, "An Approach for Low-Power Heterogeneous Parallel Implementation of ALC-PSO Algorithm Using OmpSs and CUDA," *Parallel Computing* 120 (2024): 103084.

13. Y. Huang, X. Zheng, and Y. Zhu, "Optimized CPU–GPU Collaborative Acceleration of Zero-Knowledge Proof for Confidential Transactions," *Journal of Systems Architecture* 135 (2023): 102807.

14. H. Zhao, J. Li, G. Zhang, S. Li, and J. Wang, "An Energy-Efficient Task Scheduling Method for CPU-GPU Heterogeneous Cloud," in *IEEE Conference of High Performance Computing and Communications* (New York: IEEE, 2022), 1269–1274.

15. A. Gorobets and P. Bakhvalov, "Heterogeneous CPU+GPU Parallelization for High-Accuracy Scale-Resolving Simulations of

Compressible Turbulent Flows on Hybrid Supercomputers," *Computer Physics Communications* 271 (2021): 108231.

16. T. Liang, H. Li, Y. Lin, and B. Chen, "A Distributed PTX Virtual Machine on Hybrid CPU/GPU Clusters," *Journal of Systems Architecture* 62 (2016): 63–77.

17. Y. Fu and W. Zhou, "A Heterogeneous Parallel Implementation of the Markov Clustering Algorithm for Large-Scale Biological Networks on Distributed CPU–GPU Clusters," *Journal of Supercomputing* 78 (2022): 9017–9037.

18. B. Keomley-Horvath, G. Horváth, P. Polcz, B. Siklósi, K. Tornai, and J. Juhász, "The Design and Utilisation of PanSim, a Portable Pandemic Simulator," in *2022 First Combined International Workshop on Interactive Urgent Supercomputing (CIW-IUS)* (New York: IEEE, 2022), 1–9.

19. H. Zhang, N.-M. Ho, D. Y. Polat, P. Chen, M. Wahib, and T. T. Nguyen, "Simeuro: A Hybrid CPU-GPU Parallel Simulator for Neuromorphic Computing Chips," *IEEE Transactions on Parallel and Distributed Systems* 34, no. 10 (2023): 2767–2782.

20. Z. Ou, J. Chen, Y. Sun, et al., "AOA: Adaptive Overclocking Algorithm on CPU-GPU Heterogeneous Platforms," in *Algorithms and Architectures for Parallel Processing*, vol. 13777 (Cham, Switzerland: Springer, 2022), 253–272.

21. R. Laso, J. Cabaleiro, F. Rivera, M. Muaiz, and J. Alvarez-Dios, "IHP: A Dynamic Heterogeneous Parallel Scheme for Iterative or Time-Step Methods-Image Denoising as Case Study," *Journal of Supercomputing* 77 (2021): 95–110.

22. J. Liu, G. Xiao, F. Wu, X. Liao, and K. Li, "AAPP: An Accelerative and Adaptive Path Planner for Robots on GPU," *IEEE Transactions on Computers* 72, no. 8 (2023): 2336–2349.

23. J. F. Fabeiro, A. Gonzalez-Escribano, and D. Ferraris, "Distributed Programming of a Hyperspectral Image Registration Algorithm for Heterogeneous GPU Clusters," *Journal of Parallel and Distributed Computing* 151 (2021): 86–93.

24. M. Gowanlock and B. Karsin, "A Hybrid CPU/GPU Approach for Optimizing Sorting Throughput," *Parallel Computing* 85 (2019): 45–55.

25. F. Zhang, C. Zhang, L. Yang, et al., "Fine-Grained Multi-Query Stream Processing on Integrated Architectures," *IEEE Transactions on Parallel and Distributed Systems* 32 (2021): 2303–2320.

26. R. Borrell, D. Dosimont, M. Garcia-Gasulla, et al., "Heterogeneous CPU/GPU Co-Execution of CFD Simulations on the POWER9 Architecture: Application to Airplane Aerodynamics," *Future Generation Computer Systems* 107 (2020): 31–48.

27. L. Mabrouk, S. Huet, D. Houzet, S. Belkouch, A. Hamzaoui, and Y. Zennayi, "Efficient Adaptive Load Balancing Approach for Compressive Background Subtraction Algorithm on Heterogeneous CPU-GPU Platforms," *Journal of Real-Time Image Processing* 17 (2020): 1567–1583.

28. Z. Zhong and M. Edahiro, "Model-Based Parallelization for Simulink Models on Multicore CPUs and GPUs," *International Journal of Computers and Technology* 20 (2020): 1–13.

29. H. Ma, "Development of a CPU-GPU Heterogeneous Platform Based on a Nonlinear Parallel Algorithm," *Nonlinear Engineering* 11, no. 1 (2022): 215–222.

30. F. Li, Y. Wang, J. Jiang, H. Zhang, X. Wang, and X. Chi, "Heterogeneous Acceleration Algorithms for Shallow Cumulus Convection Scheme Over GPU Clusters," *Future Generation Computer Systems* 146 (2023): 166–177.

31. Y. Lu, C. Yu, J. Xiao, et al., "A Large-Scale Heterogeneous Computing Framework for Non-Uniform Sampling Two-Dimensional Convolution Applications," *CCF Transactions on High Performance Computing* 6 (2023): 221–239.

32. V. Rosenfeld, S. Bre, and V. Markl, "Query Processing on Heterogeneous CPU/GPU Systems," *ACM Computing Surveys* 55, no. 1 (2022): 1–38.

33. Z. Yang, Q. Pan, and C. Xu, "Fine-Grained Tuple Transfer for Pipelined Query Execution on CPU-GPU Coprocessor," in *Database Systems for Advanced Applications*, vol. 13943 (Cham, Switzerland: Springer, 2023), 19–34.

34. T. Krishna and S. Bharadwaj, "Interconnect Modeling for Homogeneous and Heterogeneous Multiprocessors," in *Network-On-Chip Security and Privacy* (Cham, Switzerland: Springer, 2021), 31–54.

35. K. Raju and N. Chiplunkar, "A Survey on Techniques for Cooperative CPU-GPU Computing, Sustainable Computing," *Info Systems* 19 (2018): 72–85.

36. C. Foyer, B. Goglin, and A. R. Proano, "A Survey of Software Techniques to Emulate Heterogeneous Memory Systems in High Performance Computing," *Parallel Computing* 116 (2023): 103023.

37. A. Hazarika, S. Poddar, and H. Rahaman, "Survey on Memory Management Techniques in Heterogeneous Computing Systems," *IET Computers & Digital Techniques* 14 (2020): 47–60.

38. J. Power, J. Hestness, M. S. Orr, M. Hill, and D. Wood, "Gem5-GPU: A Heterogeneous CPU-GPU Simulator," *IEEE Computer Architecture Letters* 14 (2014): 34–36.

39. V. Zakharenko, "Fusionsim: Characterizing the Performance Benefits of Fused CPU/GPU Systems" (PhD thesis, University of Toronto, 2012).

40. J. Ma, L. Yu, J. M. Ye, and T. Chen, "MCMG Simulator: A Unified Simulation Framework for CPU and Graphic GPU," *Journal of Computer and System Sciences* 81, no. 1 (2014): 57–71.

41. H. Kim, J. Lee, N. B. Lakshminarayana, J. Sim, J. Lim, and T. Pho, *MacSim: A CPU-GPU Heterogeneous Simulation Framework User Guide* (Atlanta, GA: Georgia Institute of Technology, 2012).

42. R. Ubal, B. Jang, P. Mistry, D. Schaa, and D. Kaeli, "Multi2Sim: A Simulation Framework for CPU-GPU Computing," in *2012 21st International Conference on Parallel Architectures and Compilation Techniques (PACT)* (New York: IEEE, 2012), 335–344.

43. S. Mittal and J. S. Vetter, "A Survey of Methods for Analyzing and Improving GPU Energy Efficiency," *ACM Computing Surveys* 47 (2015): 1–23.

44. A. Masola and N. Capodieci, "Optimization Strategies for GPUs: An Overview of Architectural Approaches," *International Journal of Parallel, Emergent and Distributed Systems* 38 (2023): 140–154.

45. M. Safari and M. Huisman, "Formal Verification of Parallel Prefix Sum and Stream Compaction Algorithms in CUDA," *Theoretical Computer Science* 912 (2022): 81–98.

46. W. Pang, X. Luo, K. Chen, D. Ji, L. Qiao, and W. Yi, "Efficient CUDA Stream Management for Multi-DNN Real-Time Inference on Embedded GPUs," *Journal of Systems Architecture* 139 (2023): 102888.

47. M. Khairy, A. Wassal, and M. Zahran, "A Survey of Architectural Approaches for Improving GPGPU Performance, Programmability and Heterogeneity," *Journal of Parallel and Distributed Computing* 127 (2019): 65–88.

48. X. Kong, X. Zheng, Y. Zhu, G. Duan, and Z. Chen, "I/O-Efficient GPU-Based Acceleration of Coherent Dedispersion for Pulsar Observation," *Journal of Systems Architecture* 142 (2023): 102958.

49. D. Ernst, M. S. Holzer, G. Hager, M. Knorr, and G. Wellein, "Analytical Performance Estimation During Code Generation on Modern GPUs," *Journal of Parallel and Distributed Computing* 173 (2023): 152–167.

50. J. Fang, Z. Zhu, S. Li, et al., "Parallel Training of Pre-Trained Models via Chunk-Based Dynamic Memory Management," *IEEE Transactions on Parallel and Distributed Systems* 34, no. 1 (2023): 304–315.

51. L. Hussein, B. Younis, and A. Younis, "Graphics Accelerators: A Review," *NTU Journal of Engineering and Technology* 2 (2023): 6–15.

52. B. Cheng, E.-M. Huang, C.-H. Chao, W.-F. Sun, T.-T. Yeh, and C.-Y. Lee, "COLAB: Collaborative and Efficient Processing of Replicated Cache Requests in GPU," in *2023 28th Asia and South Pacific Design Automation Conference (ASP-DAC)* (New York: IEEE, 2023), 314–319.

53. S. Darabi, M. Sadrosadati, N. Akbarzadeh, et al., "Morpheus: Extending the Last Level Cache Capacity in GPU Systems Using Idle GPU Core Resources," in *Proceedings of the 55th Annual IEEE/ACM International Symposium on Microarchitecture* (New York: ACM, 2022), 228–244.

54. I. Jeong, Y. Oh, W. Ro, and M. Yoon, "TEA-RC: Thread Context-Aware Register Cache for GPUs," *IEEE Access* 10 (2022): 82049–82062.

55. L. Yu, T. Chen, and M. Wu, "Last Level Cache Layout Remapping for Heterogeneous Systems," *Journal of Systems Architecture* 87 (2018): 49–63.

56. J. Hong, S. Cho, and G. Kim, "Overcoming Memory Capacity Wall of GPUs With Heterogeneous Memory Stack," *IEEE Computer Architecture Letters* 21, no. 2 (2022): 61–64.

57. A. Barnes, F. Shen, and T. G. Rogers, "Mitigating GPU Core Partitioning Performance Effects," in *IEEE Symposium on High Performance Computer Architecture (HPCA)* (New York: IEEE, 2023), 530–542.

58. F. Yazdanpanah, "An Approach for Analyzing Auto-Vectorization Potential of Emerging Workloads," *Microprocessors and Microsystems* 49 (2017): 139–149.

59. B. Simpson, M. Zhu, A. Seki, and M. Scott, "Challenges in GPU-Accelerated Nonlinear Dynamic Analysis for Structural Systems," *Journal of Structural Engineering* 149 (2022): 528–537.

60. R. Abdullah, H. Zhou, and A. Awad, "Plutus: Bandwidth-Efficient Memory Security for GPUs," in *IEEE Symposium on High Performance Computer Architecture (HPCA)* (New York: IEEE, 2023), 543–555.

61. P. Dalmia, R. Mahapatra, J. Intan, D. Negrut, and M. D. Sinclair, "Improving the Scalability of GPU Synchronization Primitives," *IEEE Transactions on Parallel and Distributed Systems* 34, no. 1 (2023): 275–290.

62. C. Zhao, W. Gao, F. Nie, and H. Zhou, "A Survey of GPU Multitasking Methods Supported by Hardware Architecture," *IEEE Transactions on Parallel and Distributed Systems* 33, no. 6 (2022): 1451–1463.

63. "OpenCL: The Open Standard for Parallel Programming of Heterogeneous Systems," accessed June 20 2024, https://developer.nvidia.com/opencl.

64. "NVIDIA CUDA Toolkit Documentation 12.1," accessed June 20 2024, https://docs.nvidia.com/cuda/.

65. A. Rodriguez, A. Navarro, R. Asenjo, et al., "Exploring Heterogeneous Scheduling for Edge Computing With CPU and FPGA MPSoCs," *Journal of Systems Architecture* 98 (2019): 27–40.

66. A. Nishio, M. D. C. Filho, J. S. de Souza, and E. W. Clua, "GPU Parallel Processing to Enable Extensive Criticality Analysis in State Estimation," *Concurrency and Computation: Practice and Experience* 36, no. 20 (2024): e8200.

67. H. J. F. Luz, P. Souza, and S. R. Souza, "Structural Testing for CUDA Programming Model," *Concurrency and Computation: Practice and Experience* 36, no. 14 (2024): e8105.

68. O. Villa, D. Lustig, Z. Yan, E. Bolotin, Y. Fu, and N. Chatterjee, "Need for Speed: Experiences Building a Trustworthy System-Level GPU Simulator," in *IEEE Symposium on High Performance Computer Architecture (HPCA)* (New York: IEEE, 2021), 868–880.

69. J. W. Sheaffer, D. Luebke, and K. Skadron, "A Flexible Simulation Framework for Graphics Architectures," in *ACM SIGGRAPH/EUROGRAPHICS Conference on Graphics Hardware* (New York: ACM, 2004), 85–94.

70. V. M. Barrio, C. Gonzalez, J. Roca, and A. Fernandez, "Attila: A Cycle-Level Execution-Driven Simulator for Modern GPU Architectures," in *IEEE Symposium on Performance Analysis of Systems and Software* (New York: IEEE, 2006), 231–241.

71. R. Balasubramanian, V. Gangadhar, Z. Guo, et al., "Enabling GPGPU Low-Level Hardware Explorations With MIAOW: An Open-Source RTL Implementation of a GPGPU," *ACM Transactions on Architecture and Code Optimization* 12, no. 2 (2015): 21.

72. S. Collange, M. Daumas, D. Defour, and D. Parello, "Barra: A Parallel Functional Simulator for GPGPU," in *2010 IEEE International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems* (New York: IEEE, 2010), 351–360.

73. X. Gong, R. Ubal, and D. Kaeli, "Multi2Sim Kepler: A Detailed Architectural GPU Simulator," in *IEEE Symposium on Performance Analysis of Systems and Software* (New York: IEEE, 2017), 269–278.

74. G. F. Diamos, A. R. Kerr, S. Yalamanchili, and N. Clark, "Ocelot: A Dynamic Optimization Framework for Bulk-Synchronous Applications in Heterogeneous Systems," in *2010 19th International Conference on Parallel Architectures and Compilation Techniques (PACT)* (New York: IEEE, 2010), 353–364.

75. G. Malhotra, S. Goel, and S. R. Sarangi, "GpuTejas: A Parallel Simulator for GPU Architectures," in *Conference on High Performance Computing* (New York: IEEE, 2014), 1–10.

76. M. Khairy, Z. Shen, T. M. Aamodt, and T. G. Rogers, "Accel-Sim: An Extensible Simulation Framework for Validated GPU Modeling," in *ACM/IEEE 47th Annual Symposium on Computer Architecture (ISCA)* (New York: ACM, 2020), 473–486.

77. Y. Sun, Y. Zhang, A. Mosallaei, M. D. Shah, C. Dunne, and D. Kaeli, "Daisen: A Framework for Visualizing Detailed GPU Execution," *Computer Graphics Forum* 40 (2021): 239–250.

78. Y. Sun, T. Baruah, S. A. Mojumder, S. Dong, X. Gong, and S. Treadway, "MGPUSim: Enabling Multi-GPU Performance Modeling and Optimization," in *2019 ACM/IEEE 46th Annual International Symposium on Computer Architecture (ISCA)* (New York: IEEE, 2019), 197–209.

79. G. Alavani, "Prediction of Performance and Power Consumption of GPGPU Applications" (PhD thesis, BIRLA Institute of Technology and Science, 2023).

80. G. Alavani and S. Sarkar, "Performance Modeling of Graphics Processing Unit Application Using Static and Dynamic Analysis," *Concurrency and Computation: Practice and Experience* 34 (2021): e6602.

81. Y. Arafa, A.-H. Badawy, A. ElWazir, et al., "Hybrid, Scalable, Trace-Driven Performance Modeling of GPGPUs," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis* (New York: ACM, 2021), 1–15.

82. P. Richmond, R. Chisholm, P. Heywood, M. Chimeh, and M. Leach, "FLAME GPU 2: A Framework for Flexible and Performant Agent Based Simulation on GPUs," *Software: Practice and Experience* 53 (2023): 1659–1680.

83. Z. Li, B. Peng, and C. Weng, "XeFlow: Streamlining Inter-Processor Pipeline Execution for the Discrete CPU-GPU Platform," *IEEE Transactions on Computers* 69, no. 6 (2020): 819–831.

84. P. Gera, H. Kim, H. Kim, S. Hong, V. George, and C. Luk, "Performance Characterisation and Simulation of Intelś Integrated GPU Architecture," in *IEEE Symposium on Performance Analysis of Systems and Software (ISPASS)* (New York: IEEE, 2018), 139–148.

85. J. Gomez-Rodriguez, R. Sandoval-Arechiga, S. Ibarra-Delgado, V. Rodriguez-Abdala, J. Vazquez-Avila, and R. Parra-Michel, "A Survey of Software-Defined Networks-On-Chip: Motivations, Challenges and Opportunities," *Micromachines* 12, no. 2 (2021): 183.

86. I. A. Alimi, R. K. Patel, O. Aboderin, et al., "Network-On-Chip Topologies: Potentials, Technical Challenges, Recent Advances and Research

Direction," in *Network-On-Chip-Architecture, Optimization, and Design Explorations* (London: IntechOpen, 2021).

87. M. Alaei and F. Yazdanpanah, "A High Performance FPGA-Based Multicrossbar Prioritized Network-On-Chip," *Concurrency and Computation: Practice and Experience* 33, no. 6 (2021): e6055.

88. F. Yazdanpanah, "A Two-Level Network-On-Chip Architecture With Multicast Support," *Journal of Parallel and Distributed Computing* 172 (2023): 114–130.

89. J. Fang, Z. Leng, S. Liu, Z. Yao, and X. Sui, "Exploring Heterogeneous NoC Design Space in Heterogeneous GPU-CPU Architectures," *Journal of Computer Science and Technology* 30, no. 1 (2015): 74–83.

90. A. Mirhosseini, M. Sadrosadati, B. Soltani, H. Sarbazi-Azad, and T. F. Wenisch, "BiNoCHS: Bimodal Network-On-Chip for CPU-GPU Heterogeneous Systems," in *2017 Eleventh IEEE/ACM International Symposium on Networks-on-Chip (NOCS)* (New York: IEEE, 2017).

91. T. T. Le, D. Zhao, and M. Bayoumi, "Efficient Reconfigurable Global Network-On-Chip Designs Towards Heterogeneous CPU-GPU Systems: An Application-Aware Approach," in *2017 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)* (New York: IEEE, 2017), 439–444.

92. S. Qi, Y. Li, S. Pasricha, and R. G. Kim, "MOELA: A Multi-Objective Evolutionary/Learning Design Space Exploration Framework for 3D Heterogeneous Manycore Platforms," in *2023 Design, Automation & Test in Europe Conference & Exhibition (DATE)* (New York: IEEE, 2023), 1–6.

93. B. Asadi, S. M. Zia, H. Al-Khafaji, and A. Mohamadian, "Network-On-Chip and Photonic Network-On-Chip Basic Concepts: A Survey," *Journal of Electronic Testing* 39 (2023): 11–25.

94. B. K. Joardar, J. Rao Doppa, P. P. Pande, and K. Chakrabarty, "NoC-Enabled 3D Heterogeneous Manycore Systems for Big-Data Applications," in *23rd Symposium on Quality Electronic Design (ISQED)* (New York: IEEE, 2022), 1–6.

95. H. Zheng, K. Wang, and A. Louri, "Adapt-NoC: A Flexible Network-On-Chip Design for Heterogeneous Manycore Architectures," in *IEEE Symposium on High Performance Computer Architecture (HPCA)* (New York: IEEE, 2021), 723–735.

96. M. F. Reza, D. Zhao, H. Wu, and M. Bayoumi, "HotSpot-Aware Task-Resource Co-Allocation for Heterogeneous Many-Core Networks-On-Chip," *Computers and Electrical Engineering* 68 (2018): 581–602.

97. Y. Li and A. Louri, "ALPHA: A Learning-Enabled High Performance Network-On-Chip Router Design for Heterogeneous Manycore Architectures," *IEEE Transactions on Sustainable Computing* 6 (2020): 274–288.

98. R. Shruthi, H. Shashidhara, and M. Deepthi, "Comprehensive Survey on Wireless Network on Chips," in *Proceedings of the International Conference on Paradigms of Communication, Computing and Data Sciences* (Singapore: Springer, 2022), 203–218.

99. S. Pasricha, J. Jose, and S. Deb, "Electronic, Wireless, and Photonic Network-On-Chip Security: Challenges and Countermeasures," *IEEE Design & Test* 39 (2022): 90–98.

100. W. Choi, K. Duraisamy, R. G. Kim, et al., "On-Chip Communication Network for Efficient Training of Deep Convolutional Networks on Heterogeneous Manycore Systems," *IEEE Transactions on Computers* 67, no. 5 (2018): 672–686.

101. M. F. Reza, D. Zhao, and M. Bayoumi, "Energy-Efficient Task-Resource Co-Allocation and Heterogeneous Multi-Core NoC Design in Dark Silicon Era," *Microprocessors and Microsystems* 86 (2021): 104055.

102. M. Monil, M. Belviranli, S. Lee, J. Vetter, and A. Malony, "MEPHESTO: Modeling Energy-Performance in Heterogeneous SoCs and Their Trade-Offs," in *ACM Conference on Parallel Architectures and Compilation Techniques* (New York: ACM, 2020), 413–425.

103. L. Wan, W. Zheng, and X. Yuan, "Efficient Inter-Device Task Scheduling Schemes for Multi-Device Co-Processing of Data-Parallel Kernels on Heterogeneous Systems," *IEEE Access* 9 (2021): 59968–59978.

104. S. Yesil and O. Ozturk, "Scheduling for Heterogeneous Systems in Accelerator-Rich Environments," *Journal of Supercomputing* 78 (2021): 200–221.

105. O. Beaumont, L. Canon, L. Eyraud-Dubois, et al., "Scheduling on Two Types of Resources: A Survey," *ACM Computing Surveys* 53 (2020): 56.

106. T. Sung, J. Ha, J. Kim, A. Yahja, C. Sohn, and B. Ryu, "DeepSoCS: A Neural Scheduler for Heterogeneous System-On-Chip (SoC) Resource Scheduling," *Electronics* 9 (2020): 936.

107. S. Liu, Y. Pu, P. Liao, et al., "FastGR: Global Routing on CPU–GPU With Heterogeneous Task Graph Scheduler," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 42, no. 7 (2023): 2317–2330.

108. T. A. Rahmani, G. Belalem, and S. A. Mahmoudi, "RTLB_Sched: Real Time Load Balancing Scheduler for CPU-GPU Heterogeneous Systems," in *2023 International Conference on Smart Computing and Application (ICSCA)* (New York: IEEE, 2023), 1–6.

109. Q. Zeng, Y. Du, K. Huang, and K. Leung, "Energy-Efficient Resource Management for Federated Edge Learning With CPU-GPU Heterogeneous Computing," *IEEE Transactions on Wireless Communications* 20, no. 12 (2021): 7947–7962.

110. T. Cogumbreiro, J. Lange, D. Liew, and H. Zicarelli, "Memory Access Protocols: Certified Data-Race Freedom for GPU Kernels," *Formal Methods in System Design* 63 (2024): 134–171.

111. K. Zhang, J. Hong, Z. He, Y. Jing, and X. S. Wang, "AdaptChain: Adaptive Data Sharing and Synchronization for NFV Systems on Heterogeneous Architectures," *IEEE Transactions on Parallel and Distributed Systems* 35, no. 7 (2024): 1281–1292.

112. J. Fang, Q. Fan, X. Hao, Y. Cheng, and L. Sun, "Performance Optimization by Dynamically Altering Cache Replacement Algorithm in CPU-GPU Heterogeneous Multi-Core Architecture," in *IEEE/ACM Symposium on Cluster, Cloud and Grid Computing* (New York: ACM, 2017), 723–726.

113. C. Chen, K. Li, A. Ouyang, Z. Zeng, and K. Li, "Gink: An In-Memory Computing Architecture on Heterogeneous CPU-GPU Clusters for Big Data," *IEEE Transactions on Parallel and Distributed Systems* 29, no. 6 (2018): 1275–1288.

114. L. Gao, R. Wang, Y. Xu, et al., "SRAM- and STT-RAM-Based Hybrid, Shared Last-Level Cache for On-Chip CPU-GPU Heterogeneous Architectures," *Journal of Supercomputing* 74, no. 7 (2018): 3388–3414.

115. R. Ausavarungnirun, V. Miller, J. Landgraf, et al., "MASK: Redesigning the GPU Memory Hierarchy to Support Multi-Application Concurrency," in *ACM Conference on Architectural Support for Programming Languages and Operating Systems* (New York: ACM, 2018), 503–518.

116. L. Liang, Q. Zhang, P. Song, et al., "Overlapping Communication and Computation of GPU/CPU Heterogeneous Parallel Spatial Domain Decomposition MOC Method," *Annals of Nuclear Energy* 135 (2019): 106988.

117. S. Wang, H. Lee, F. Ebrahimi, P. K. Amiri, K. L. Wang, and P. Gupta, "Comparative Evaluation of Spin-Transfer-Torque and Magneto Electric Random Access Memory," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems* 6, no. 2 (2016): 134–145.

118. J. Lee, S. Li, H. Kim, and S. Yalamanchili, "Design Space Exploration of On-Chip Ring Interconnection for a CPU-GPU Heterogeneous Architecture," *Journal of Parallel and Distributed Computing* 73, no. 12 (2013): 1525–1538.

119. Z. Li, N. Goswami, and T. Li, "iConn: A Communication Infrastructure for Heterogeneous Computing Architectures," *ACM Journal on Emerging Technologies in Computing Systems* 11, no. 4 (2015): 42.

120. L. Alhubail, "Optimization of Heterogeneous NoC for Fused CPU-GPU Architecture" (PhD thesis, UC Irvine, 2019).

121. M. Alaei and F. Yazdanpanah, "A High Reliable Multicast Routing Algorithm for 2D and 3D Mesh-Based NoCs With Fuzzy-Based Load Control," *Journal of Control* 15, no. 1 (2021): 113–125.

122. M. Alaei and F. Yazdanpanah, "A Dynamic Congestion Management Method for Reconfigurable Network on Chip," *Journal of Soft Computing and Information Technology* 9, no. 2 (2020): 74–86.

123. F. Yazdanpanah, "An Adaptive Multicast Routing Algorithm for Network-On-Chip With Fuzzy-Based Load Control," in *Proceedings of the 4th Conference on Natural Sciences - Mathematics & Computer* (2019).

124. B. K. Joardar, W. Choi, R. G. Kim, J. R. Doppa, P. P. Pande, and D. Marculescu, "3D NoC-Enabled Heterogeneous Manycore Architectures for Accelerating CNN Training: Performance and Thermal Trade-Offs," in *2017 Eleventh IEEE/ACM International Symposium on Networks-on-Chip (NOCS)* (New York: IEEE, 2017), 1–8.

125. D. Xu, Y. Ouyang, W. Zhou, Z. Huang, H. Liang, and X. Wen, "RMC-NoC: A Reliable On-Chip Network Architecture With Reconfigurable Multifunctional Channel," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 31, no. 12 (2023): 2061–2074.

126. B. Boroujerdian, Y. Jing, D. Tripathy, et al., "FARSI: An Early-Stage Design Space Exploration Framework to Tame the Domain-Specific System-On-Chip Complexity," *ACM Transactions on Embedded Computing Systems* 22 (2022): 31.

127. S. Charles and P. Mishra, "A Survey of Network-On-Chip Security Attacks and Countermeasures," *ACM Computing Surveys* 54 (2021): 101.

128. M. Balti and A. Jemai, "Performance Survey of Classic and Optic Network-On-Chip," *IET Circuits, Devices & Systems* 15 (2021): 393–402.

129. F. Yazdanpanah and R. Afsharmazayejani, "A Systematic Analysis of Power Saving Techniques for Wireless Network-On-Chip Architectures," *Journal of Systems Architecture* 126 (2022): 102485.

130. F. Yazdanpanah, "A Low-Power WNoC Transceiver With a Novel Energy Consumption Management Scheme for Dependable IoT Systems," *Journal of Parallel and Distributed Computing* 172 (2023): 144–158.

131. A. Sarihi, A. Patooghy, A. Khalid, M. Hasanzadeh, M. Said, and A.-H. A. Badawy, "A Survey on the Security of Wired, Wireless, and 3D Network-On-Chips," *IEEE Access* 9 (2021): 107625–107656.

132. F. Yazdanpanah, R. Afsharmazayejani, A. Rezaei, M. Alaei, and M. Daneshtalab, "An Energy-Efficient Partition-Based XYZ-Planar Routing Algorithm for a Wireless Network-On-Chip," *Journal of Supercomputing* 75 (2019): 837–861.

133. R. Afsharmazayejani, F. Yazdanpanah, A. Rezaei, M. Alaei, and M. Daneshtalab, "HoneyWiN: Novel Honeycomb-Based Wireless NoC Architecture in Many-Core Era," in *Applied Reconfigurable Computing. Architectures, Tools, and Applications* (Cham, Switzerland: Springer, 2018), 304–316.

134. M. Alaei and F. Yazdanpanah, "H^2WNoC: A Honeycomb Hardware-Efficient Wireless Network-On-Chip Architecture," *Nano Communication Networks* 19 (2019): 119–133.

135. M. F. Reza, D. Zhao, and M. Bayoumi, "Power-Thermal Aware Balanced Task-Resource Co-Allocation in Heterogeneous Many CPU-GPU Cores NoC in Dark Silicon Era," in *2018 31st IEEE International System-on-Chip Conference (SOCC)* (New York: IEEE, 2018), 260–265.

136. J. Lee, S. Li, H. Kim, and S. Yalamanchili, "Adaptive Virtual Channel Partitioning for NoC in Heterogeneous Architectures," *ACM Transactions on Design Automation of Electronic Systems* 18, no. 4 (2013): 48.

137. M. Alaei and F. Yazdanpanah, "A Fuzzy-Based Routing Scheme for Network-On-Chip With Honeycomb Topology," *Computational Methods for Differential Equations* 7, no. 4 (2019): 511–520.

138. J. Fang, Z. Chang, Y. Cheng, and H. Zhao, "Exploration on Routing Configuration of HNoC With Reasonable Energy Consumption," in *2018 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)* (New York: IEEE, 2018), 744–749.

139. A. Rodriguez, A. Navarro, K. Nikov, et al., "Lightweight Asynchronous Scheduling in Heterogeneous Reconfigurable Systems," *Journal of Systems Architecture* 124 (2022): 102398.

140. A. M. Rahmani, B. Donyanavard, T. Mück, et al., "SPECTR: Formal Supervisory Control and Coordination for Many-Core Systems Resource Management," in *ACM Conference on Architectural Support for Programming Languages and Operating Systems* (New York: ACM, 2018), 169–183.

141. R. Lyerly, A. Murray, A. Barbalace, and B. Ravindran, "AIRA: A Framework for Flexible Compute Kernel Execution in Heterogeneous Platforms," *IEEE Transactions on Parallel and Distributed Systems* 29, no. 2 (2018): 269–282.

142. N. Farooqui, I. Roy, Y. Chen, et al., "Accelerating Data Analytics on Integrated GPU Platforms via Runtime Specialization," *International Journal of Parallel Programming* 46, no. 2 (2018): 336–375.

143. L. Cheng, "Intelligent Scheduling for Simultaneous CPU-GPU Applications" (PhD thesis, Graduate College of the University of Illinois at Urbana-Champaign, 2017).

144. B. Perez, E. Stafford, J. Bosque, and R. Beivide, "Sigmoid: An Auto-Tuned Load Balancing Algorithm for Heterogeneous Systems," *Journal of Parallel and Distributed Computing* 157 (2021): 30–42.

145. X. Gao, "TAS: A Temperature-Aware Scheduling for Heterogeneous Computing," *IEEE Access* 11 (2023): 54773–54781.

146. J. Chen, P. Han, Y. Zhang, T. You, and P. Zheng, "Scheduling Energy Consumption-Constrained Workflows in Heterogeneous Multi-Processor Embedded Systems," *Journal of Systems Architecture* 142 (2023): 102938.

147. B. N. Chandrashekhar, M. Mohan, and V. Geetha, "Forecast Model for Scheduling an HPC Application on CPU and GPU Architecture," in *2023 3rd International Conference on Intelligent Technologies (CONIT)* (New York: IEEE, 2023), 1–5.

148. Y. Zhang, M. Li, and F. Tong, "Energy-Efficient Load Balancing for Divisible Tasks on Heterogeneous Clusters," *Transactions on Emerging Telecommunications Technologies* 34 (2023): e4829.

149. Y. Zhang, Z. Xu, Y. Zhang, et al., "FTSC: Fault-Tolerant Scheduling and Control co-Design for Distributed Real-Time System," *Journal of Systems Architecture* 142 (2023): 102934.

150. W. Jiang, M. Long, L. T. Yang, et al., "Fipip: A Novel Fine-Grained Parallel Partition Based Intra-Frame Prediction on Heterogeneous Many-Core Systems," *Future Generation Computer Systems* 78 (2018): 316–329.

151. J. Kim, S. Seo, J. Lee, J. Nah, G. Jo, and J. Lee, "SnuCL: An OpenCL Framework for Heterogeneous CPU/GPU Clusters," in *Proceedings of the 26th ACM international conference on Supercomputing* (New York: ACM, 2012), 341–352.

152. Z. Li, Y. Zhang, A. Ding, H. Zhou, and C. Liu, "Efficient Algorithms for Task Mapping on Heterogeneous CPU/GPU Platforms for Fast Completion Time," *Journal of Systems Architecture* 114 (2020): 101936.

153. I. Harichane, S. A. Makhlouf, and G. Belalem, "KubeSC-RTP: Smart Scheduler for Kubernetes Platform on CPU-GPU Heterogeneous Systems," *Concurrency and Computation: Practice and Experience* 34, no. 21 (2022): e7108.

154. M. Wilhelm, H. Geppert, A. Drewes, and T. Pionteck, "A Comprehensive Modeling Approach for the Task Mapping Problem in Heterogeneous Systems With Dataflow Processing Units," *Concurrency and Computation: Practice and Experience* 35, no. 25 (2023): e7909.

155. F. Yazdanpanah and M. Alaei, "Design Space Exploration of Hardware Task Superscalar Architecture," *Journal of Supercomputing* 71 (2015): 3567–3592.

156. F. Yazdanpanah, C. Alvarez, D. Jimenez-Gonzalez, R. Badia, and M. Valero, "Picos: A Hardware Runtime Architecture Support for OmpSs," *Future Generation Computer Systems* 53 (2015): 130–139.

157. M. Alaei and F. Yazdanpanah, "EELCM: An Energy Efficient Load-Based Clustering Method for Wireless Mobile Sensor Networks," *Mobile Networks and Applications* 24 (2019): 1486–1498.

158. M. Alaei and F. Yazdanpanah, "A Distributed Fuzzy-Based Clustering Scheme to Optimize Energy Consumption and Data Transmission in Wireless Sensor Networks," *Journal of Soft Computing and Information Technology* 9, no. 3 (2020): 229–243.

159. J. Hu, G. Dai, L. Wang, et al., "Adaptive Multidimensional Parallel Fault Simulation Framework on Heterogeneous System," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 42, no. 6 (2023): 1951–1964.

160. J. J. Escobar, J. Ortega, A. F. Diaz, J. Gonzalez, and M. Damas, "Energy-Aware Load Balancing of Parallel Evolutionary Algorithms With Heavy Fitness Functions in Heterogeneous CPU-GPU Architectures," *Concurrency and Computation: Practice and Experience* 31 (2019): e4688.

161. T. A. Rahmani, G. Belalem, S. A. Mahmoudi, and O. R. Merad-Boudia, "Equalizer: Energy-Efficient Machine Learning-Based Heterogeneous Cluster Load Balancer," *Concurrency and Computation: Practice and Experience* 36 (2024): e8230.

162. K. Ma, X. Li, W. Chen, C. Zhang, and X. Wang, "GreenGPU: A Holistic Approach to Energy Efficiency in GPU-CPU Heterogeneous Architectures," in *2012 41st International Conference on Parallel Processing* (New York: IEEE, 2012), 48–57.

163. A. Marowka, "Energy Consumption Modeling for Hybrid Computing," in *Euro-Par 2012 Parallel Processing* (Berlin, Germany: Springer, 2012), 54–64.

164. J. Escobar, J. Ortega, A. Deaz, J. Gonzelez, and M. Damas, "Time-Energy Analysis of Multilevel Parallelism in Heterogeneous Clusters: The Case of EEG Classification in BCI Tasks," *Journal of Supercomputing* 75 (2019): 3397–3425.

165. X. Fei, K. Li, W. Yang, and K. Li, "Analysis of Energy Efficiency of a Parallel AES Algorithm for CPU-GPU Heterogeneous Platforms," *Parallel Computing* 94-95 (2020): 102621.

166. L. Wan, X. Cui, Y. Li, W. Zheng, and X. Yuan, "HeteroPP: A Directive-Based Heterogeneous Cooperative Parallel Programming Framework," *Concurrency and Computation: Practice and Experience* 36, no. 11 (2024): e8014.

167. P. Incardona, A. Gupta, S. Yaskovets, and I. F. Sbalzarini, "A Portable C++ Library for Memory and Compute Abstraction on Multi-Core CPUs and GPUs," *Concurrency and Computation: Practice and Experience* 35, no. 25 (2023): e7870.

168. J. Fang, C. Huang, T. Tang, and Z. Wang, "Parallel Programming Models for Heterogeneous Many-Cores: A Comprehensive Survey," *CCF Transactions on High Performance Computing* 2 (2020): 382–400.

169. "HIP: C++ Heterogeneous-Compute Interface for Portability," accessed June 20 2024, https://rocm.docs.amd.com/projects/HIP/.

170. "OpenMP ARB Releases Technical Report 13," accessed June 20 2024, https://www.openmp.org/.

171. A. Navarro, F. Corbera, A. Rodreguez, A. Vilches, and R. Asenjo, "Heterogeneous Parallel-For Template for CPU-GPU Chips," *International Journal of Parallel Programming* 47 (2019): 213–233.

172. F. Ciccozzi, L. Addazi, S. A. Asadollah, B. Lisper, A. N. Masud, and S. Mubeen, "A Comprehensive Exploration of Languages for Parallel Computing," *ACM Computing Surveys* 55 (2022): 1–39.

173. S. Maheshwari, V. Y. Gudur, R. Shafik, I. Wilson, A. Yakovlev, and A. Acharyya, "CORAL: Verification-Aware OpenCL Based Read Mapper for Heterogeneous Systems," *IEEE/ACM Transactions on Computational Biology and Bioinformatics* 18 (2019): 1426–1438.

174. A. Bloch, S. Casale-Brunet, and M. Mattavelli, "Design Space Exploration for Partitioning Dataflow Program on CPU-GPU Heterogeneous System," *Journal of Signal Processing Systems* 95 (2023): 1219–1229.

175. "SYCL, C++ Programming for Heterogeneous Parallel Computing," accessed June 20 2024, https://www.khronos.org/sycl/.

176. V. Zakharenko, T. Aamodt, and A. Moshovos, "Characterizing the Performance Benefits of Fused CPU/GPU Systems Using Fusionsim," in *2013 Design, Automation & Test in Europe Conference & Exhibition (DATE)* (New York: IEEE, 2013), 685–688.

177. S. Shagufta, M. Aleem, M. Arshad, and M. Azhar, "A Comparative Study of Heterogeneous Processor Simulators," *International Journal of Computer Applications* 148 (2016): 5–11.

178. C. Lai, C. Yeh, C. Tu, and S. Hung, "Fast Profiling Framework and Race Detection for Heterogeneous System," *Journal of Systems Architecture* 81 (2017): 83–91.

179. R. I. Acosta-Quinonez, D. Torres-Roman, and R. Rodriguez-Avila, "HOSVD Prototype Based on Modular SW Libraries Running on a High-Performance CPU+GPU Platform," *Journal of Systems Architecture* 113 (2021): 101897.

180. H. Wang, V. Sathish, R. Singh, M. J. Schulte, and N. S. Kim, "Workload and Power Budget Partitioning for Single-Chip Heterogeneous Processors," in *Conference on Parallel Architectures and Compilation Techniques* (New York: IEEE, 2012), 401–410.

181. A. Sembrant, T. E. Carlson, E. Hagersten, and D. Black-Schaffer, "A Graphics Tracing Framework for Exploring CPU-GPU Memory Systems," in *2017 IEEE International Symposium on Workload Characterization (IISWC)* (New York: IEEE, 2017), 54–65.

182. P. H. Wang, G.-H. Liu, J.-C. Yeh, T.-M. Chen, H.-Y. Huang, and C.-L. Yang, "Full System Simulation Framework for Integrated CPU/GPU Architecture," in *Technical Papers of 2014 International Symposium on VLSI Design, Automation and Test* (New York: IEEE, 2014), 1–4.

183. K. Kaszyk, H. Wagstaff, T. Spink, B. Franke, M. O'Boyle, and B. Bodin, "Full-Ssystem Simulation of Mobile CPU/GPU Platforms," in *IEEE Symposium on Performance Analysis of Systems and Software (ISPASS)* (New York: IEEE, 2019), 68–78.

184. A. Kurth, B. Forsberg, and L. Benini, "HEROv2: Full-Stack Open-Source Research Platform for Heterogeneous Computing," *IEEE Transactions on Parallel and Distributed Systems* 33, no. 12 (2022): 4368–4382.

185. Z. Guo, T. Huang, and Y. Lin, "Accelerating Static Timing Analysis Using CPU-GPU Heterogeneous Parallelism," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 42 (2023): 4973–4984.

186. A. Patel, F. Afram, S. Chen, and K. Ghose, "MARSS: A Full System Simulator for Multicore x86 CPUs," in *Proceedings of the 48th Design Automation Conference* (New York: ACM, 2011), 1050–1055.

187. P. Chalise, R. Raghavan, and B. L. Fridley, "InterSIM: Simulation Tool for Multiple Integrative OMIC Datasets," *Computer Methods and Programs in Biomedicine* 128 (2016): 69–74.

188. F. Zhang, W. Liu, N. Feng, J. Zhai, and X. Du, "Performance Evaluation and Analysis of Sparse Matrix and Graph Kernels on Heterogeneous Processors," *CCF Transactions on High Performance Computing* 1 (2019): 131–143.

189. D. Kim, W. Choi, C. I. Lim, et al., "Towards Scalable and Configurable Simulation for Disaggregated Architecture," *Simulation Modelling Practice and Theory* 125 (2023): 102743.

190. C. Giles, C. Peterson, and M. Heinrich, "KnightSim: A Fast Discrete Event-Driven Simulation Methodology for Computer Architectural Simulation," *IEEE Transactions on Computers* 69 (2020): 65–71.

191. A. Arka, B. K. Joardar, R. G. Kim, D. H. Kim, J. R. Doppa, and P. P. Pande, "HeM3D: Heterogeneous Manycore Architecture Based on Monolithic 3D Vertical Integration," *ACM Transactions on Design Automation of Electronic Systems* 26 (2021): 16.

192. L. Wang, R.-W. Tsai, S.-C. Wang, K.-C. Chen, P.-H. Wang, and H.-Y. Cheng, "Analyzing OpenCL 2.0 Workloads Using a Heterogeneous CPU-GPU Simulator," in *IEEE Symposium on Performance Analysis of Systems and Software* (New York: IEEE, 2017), 127–128.

193. X. Cai, J. Yin, and P. Zhou, "An Orchestrated NoC Prioritization Mechanism for Heterogeneous CPU-GPU Systems," *Integration* 65 (2018): 344–350.

194. M. Sinha, S. Gupta, S. Rout, and S. Deb, "Sniffer: A Machine Learning Approach for DoS Attack Localization in NoC-Based SoCs," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems* 11 (2021): 278–291.

195. H. Tabani, F. Mazzocchetti, P. Benedicte, J. Abella, and F. Cazorla, "Performance Analysis and Optimization Opportunities for NVIDIA Automotive GPUs, Parallel and Distributed," *Computing* 152 (2021): 21–32.

196. H. Lahdhiri, J. Lorandel, S. Monteleone, E. Bourdel, and M. Palesi, "Framework for Design Exploration and Performance Analysis of RF-NoC Manycore Architecture," *Journal of Low Power Electronics and Applications* 10, no. 4 (2020): 37.

197. J. Hestness, S. W. Keckler, and D. A. Wood, "GPU Computing Pipeline Inefficiencies and Optimization Opportunities in Heterogeneous CPU-GPU Processors," in *2015 IEEE International Symposium on Workload Characterization* (New York: IEEE, 2015), 87–97.

198. J. Gomez-Luna, I. El Hajj, L.-W. Chang, V. García-Floreszx, S. G. de Gonzalo, and T. B. Jablin, "Chai: Collaborative Heterogeneous Applications for Integrated-Aarchitectures," in *IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)* (New York: IEEE, 2017), 43–54.

199. J. Kim, S. Lee, B. Johnston, and J. S. Vetter, "IRIS: A Performance-Portable Framework for Cross-Platform Heterogeneous Computing," *IEEE Transactions on Parallel and Distributed Systems* 35, no. 10 (2024): 1796–1809.

200. J. Fang, Z. Wei, Y. Liu, and Y. Hoe, "TB-TBP: A Task-Based Adaptive Routing Algorithm for Network-On-Chip in Heterogenous CPU-GPU Architectures," *Journal of Supercomputing* 80 (2024): 6311–6335.

201. R. Ubal, J. Sahuquillo, S. Petit, and P. Lopez, "Multi2Sim: A Simulation Framework to Evaluate Multicore-Multithreaded Processors," in *19th International Symposium on Computer Architecture and High Performance Computing* (New York: IEEE, 2007), 62–68.

202. F. Candel, S. Petit, J. Sahuquillo, and J. Duato, "Accurately Modeling the On-Chip and Off-Chip GPU Memory Subsystem," *Future Generation Computer Systems* 82 (2018): 510–519.

203. C. E. Giles and M. A. Heinrich, "M2S-CGM: A Detailed Architectural Simulator for Coherent CPU-GPU Systems," in *2017 IEEE International Conference on Computer Design (ICCD)* (New York: IEEE, 2017), 477–484.

204. C. Giles, "Simulation, Analysis, and Optimization of Heterogeneous CPU-GPU Systems" (PhD thesis, University of Central Florida, 2019).

205. S. H. Gade, M. M. Ahmed, S. Deb, and A. Ganguly, "Energy Efficient Chip-To-Chip Wireless Interconnection for Heterogeneous Architectures," *ACM Transactions on Design Automation of Electronic Systems* 24, no. 5 (2019): 1–27.

206. Z. Mann and L. Jani, "Cache Optimization for CPU-GPU Heterogeneous Processors," *American Journal of Algorithms and Computing* 2 (2015): 18–31.