SCCTS Transactions on Reconfigurable Computing, ISSN: 3049-1533 Vol. 3, No. 3, 2026 (pp. 14-21)
RESEARCH ARTICLE ECEJOURNALS.IN

Reconfigurable Operating System Support for Heterogeneous SoC Platforms with FPGAs and GPUs

M. Karpagam*

Assistant Professor, Department of Computational Intelligence, SRM Institute of Science and Technology, Kattankulathur, Chennai

Keywords:

Heterogeneous Computing,
Reconfigurable Operating Systems,
Hardware/Software Co-design,
FPGA-GPU Integration,
Task Scheduling,
Resource Virtualization.

Author's Email:
karpist@gmail.com

DOI: 10.31838/RCC/03.03.03

Received : 16.01.2026
Revised : 18.03.2026
Accepted : 16.04.2026

Abstract

And the current Systems-on-Chip architectures are becoming very heterogeneous, combining spatial parallelism of Field-Programmable Gate Arrays (FPGAs) with a higher throughput temporal parallelism of Graphics Processing Units (GPUs). In spite of its potential, conventional operating systems identify these accelerators as remote I/O peripherals, leaving the developer with the resource management and synchronisation challenges to the application developer. This paper suggests a Reconfigurable Operating System (ROS) platform that will supply heterogeneous accelerator support in a native and unified manner. The area of this work covers Virtual Accelerator Layer (VAL) which an abstracts hardware specific binaries and Latency-Aware Hybrid Scheduler, which dynamically allocates tasks to an optimal execution unit depending on the current availability of resources and power limits. The ROS uses a Unified Memory Management Unit (UMMU) to support the zero-copy sharing of data between the CPU, FPGA, and GPU domains to address the issue of data movement bottlenecks. Our implementation of the ROS was a Xilinx Zynq UltraScale+ platform that included a discrete GPU. Mixed-workload benchmark experimental performance shows that our OS-level integrations of work execution facilitate faster execution of tasks by as much as 25% and can reduce energy consumption by up to 30% through the use of conventional driver-based manual work management. The ROS has been used to achieve better utilisation of the hardware-software interface and easier development of more complicated, multi-accelerator applications by virtualizing the hardware-software interface. This study illustrates that accelerator management needs to be transferred to OS kernel to have the next generation, power efficient, and high performance heterogeneous SoCs.

How to cite this article: Karpagam M (2026). Reconfigurable Operating System Support for Heterogeneous SoC Platforms with FPGAs and GPUs. SCCTS Transactions on Reconfigurable Computing, Vol. 3, No. 3, 2026, 14-21

Introduction

The continuing demise of the Dennard scaling and consequent development of the so-called Dark Silicon have shifted the direction of System-on-Chip (SoC) design paradigm wholesomely to domain-specific acceleration to overcome the limits of the classic power wall.[1] FPGAs and Graphics Processing Units (GPUs) have become the two major, though architecturally different, hardware accelerators in this domain of specialisation. FPGAs have been successful at custom bit-level pipelines, fine-grained spatial parallelism, deterministic, low-latency processing, and signal processing and custom protocols. On the other hand, GPUs are more popular in high density, throughput-based temporal parallelism, and suited to large-scale floating-point arithmetic.[2] Although complementary,

such accelerators are normally controlled by the modern operating system (OS) architecture as independent I/O devices, but not as part of a single computational mechanism. This separation establishes some kind of a silo effect in which the central processing unit (CPU) has to manually coordinate complicated data flow, control disparate synchronisation primitives as well as guarantee heterogenous kernel execution across devices that are not consistent on an execution model. We claim that a really reconfigurable operating system needs to go beyond the mere support of these underlying hardware variations at driver level and rather, encapsulates these hardware variations into a single management platform. With this type of a model, a developer can specify a single high-level task which can be dynamically scheduled to either the FPGA fabric or the cores in the GPU implementation depending on the state of the runtime system, available bandwidth and power constraints. In heterogeneous computing, recent work has either been optimization-oriented in particular accelerator cost-optimal performance, or specialised middleware optimization as deployed to new high-level such as deep learning and federated learning applications.[3] Nonetheless, the extant literature does not have an OS-level abstraction that views FPGA fabrics and GPU clusters as a common and exchangeable resource pool. Most modern development toolchains currently require a model of compile-time static partitioning of tasks, which becomes more and more fragile, and does not model the high-dimensional and dynamic needs of complex and multi-tenant application systems with resource availability variations of microscales.[4] In addition, the available traditional OS kernels do not contain the underlying logic to consider the trade-offs between scepticism and latitude between the reconfiguration latency of loading bitstreams into an FPGA kernel and the large kernel-launch overhead of GPGPU programming models. The lack of physically integrated OS level support required developers to either over-provision resources or accept substantial levels of energy wastages during transfers of large volumes of data by the PCIe or AXI bus. These unwarranted efficiencies are made worse in mobile and edge SoC platforms where power-delay product (PDP) is the only measure of success.

The paper presents Reconfigurable Operating System (ROS) architecture, which is aimed at eliminating the divider between reconfigurable logic and stream processors by integrating directly with the kernel. The main work has contributed a Virtual Accelerator Layer (VAL) which does not depend on the application-level logic and hardware-dependent binaries, and is enabled by a new Latency-Aware Hybrid Scheduler.[5] It is a time scheduler built around real-time hardware performance counters and predictive power profiles that provides automated placement of tasks to effectively conceal the hardware complexity of hardware to the end-user with maximum throughput. To address the data-copying bottleneck which is characteristic of heterogeneous systems, we adopt a Unified Memory Management Unit (UMMU) that can be used to support zero-copy data sharing and cache coherency between the CPU, FPGA and the GPU space. Something that can be achieved with the OS is the ability to optimise the system-wide as a whole, which cannot be done at application level or driver level by itself. This paper is structured as follows; Section II will cover the history of the reconfigurable and heterogeneous OS design, and the gaps in the existing implementations of theso-called accelerator-aware kernels; Section III will contain the description of the proposed ROS architecture and the formulation of our scheduling policy as a mathematical model; Section IV will discuss our implementation and experimental platform, used as Xilinx Zynq UltraScale+ interface; Section V will cover the discussion of our results concerning the execution latency and power consumption; and Section VI will discuss our finding on what will be the future of the

Related Work

The historical direction of operated system research on reconfigurable logic has been one almost entirely concerned with integrating Field-Programmable Gate Arrays (FPGAs) into the software stack. The creation of ReconOS was one of the most important advancements in this field and introduced the notion of hardware threads that communicated with the Linux kernel through normal POSIX-based synchronisation primitives.[6] This solution made hardware modules so first-class OS objects that they could be preempted and share resources with software domains and be multitasked. On these ideas, BORPH explored the paradigm by considering FPGA kernels as ordinary UNIX processes where the hardware execution interface included a file-system interface which made it easier to deploy reconfigurable applications.[7] Although these models were revolutionary in their day, they were made in the period when the FPGA was the only accelerator and

they did not take into consideration the modern multi-accelerator environments which have high-paralleling GPGPU cores. Scheduling In parallel The advanced GPUs on the parallel processing side prompted the advancement of powerful middleware and runtime platforms including the OpenCL and the Xilinx Runtime (XRT). These platforms offer an abstraction layer to control memory buffers and kernel launch of homogeneous devices.[8] These runtimes have a limited low-level kernel presence, and do not have the extensive deep kernel integration that is needed by high-level OS orchestration. In particular, they lack the ability to control resource contention at the system level when two or more applications are competing in the same FPGA fabric or GPU compute elements at the same time. In addition, the current middleware typically uses a host-guest paradigm with the CPU being a hard-deterministic controller subjecting a lot of synchronisation cost and resulting in poor usage efficiency of the interconnect bandwidth in high-concurrently-used cases.[9] The largest deficiency in the current body of knowledge is probably the lack of a single OS-level scheduler that can ensure mediation between the inherently different execution paradigms of FPGAs and GPUs. Although recent research has suggested hybrid mapping solutions to particular algorithms, e.g., the Convolutional Neural Networks (CNNs), these alternatives are normally application specific and lack general-purpose suitability that is demanded by a contemporary operating system.[10] Even the current models have trouble in finding a compromise between the cold start issue of FPGA reconfiguration latency and that of launching a GPU kernel in a single decision making model. Our solution fills this gap that is so crucial by relegating the decision-making reasoning to the application layer to the OS kernel. Our proposed structure solves the siloed management structures of earlier models through offering an effective cost-benefit analysis of the two types of accelerators, a framework that scales appropriately to become the next generation of truly heterogeneous SoC platforms.

Methodology

The fundamental architecture design of the planned Reconfigurable Operating System (ROS) utilises a multi-level three-tier software-hardware stack where acceleration resources of heterogeneous sources are thrust directly into the execution pipeline of the kernel. This study was established through the development of a centralised orchestration layer which is placed between POSIX-compliant system call interface and the physical hardware abstraction layer. The approach of moving the control logic to the kernel makes the choice of accelerators no longer a burden on the application developer but instead of an autonomous system able to compute the state of global resources, the interconnect congestion, and power availability. This shift in architecture allows the system to view the reconfigurable logic of the FPGA as well as the stream processors of the GPU as a fluid pool of computing resources instead of being peripherals.

The Virtual Accelerator Layer (VAL)

The main abstraction used by the Virtual Accelerator Layer (VAL) is the high-level abstraction (Sandburg) called the Virtual Accelerator Layer which is a high-level interface that separates functional intent and physical implementation. The layer functions as one gateway on application requests as shown in Figure 1. It works by getting user level requests and transforming those requests into standardised Unified Task Descriptors. These descriptors contain all the metadata needed to run a particular computational kernel, such as the address of the FPGA bitstream blocks necessary to execute it spatially or the address ofwritten binaries necessary to execute them temporally using a GPU kernel. The VAL maintains a maintained metadata store that keeps track of the compatibility and needs of each task like what logic slices (CLBs) and Digital Signal Processing (DSP) blocks are required in Fpgas or how many thread-blocks and how much register pressure are required in GPUs. With

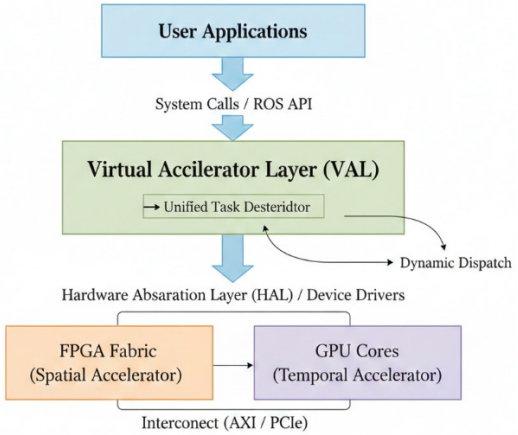

Fig. 1: Overview of High Level Architecture of Reconfigurable Operating System (ROS) that indicates the incorporation of the Virtual Accelerator Layer (VAL) and the heterogeneous nature of the hardware back-end.

the assistance of this abstraction, the OS will keep a list of platform-independent activities so that the system can postpone the decision about what hardware back-end to use until it is the time of execution. This late binding technique is essential in keeping the system wide flexible because the ROS can afford to ignore a faulty accelerator or an overloaded one in favour of an available one without re-compiling the application code. In addition, the VAL uses versioning of hardware kernels, such that the OS is at all times configured to match the functional requirements of the application to the most efficient implementation of the hardware implementation which can be found in the bitstream/ binary library.

Latency-Aware Hybrid Scheduling

In an effort to cater to the complexity of a dual-accelerator environment, the ROS has a constant-time, dynamic, and latency-conscious scheduling algorithm, which is used to make real-time decisions in multi-tenant systems. Figure 2 elaborates on the operations logic of this mechanism. During runtime, the scheduler considers the aggregate cost C of a given task i when executed on an accelerator a by using a general cost model that is determined through an equation:

$$C_{i,a} = T_{exec}(i,a) + \delta_{reconfig}(a) + \delta_{comm}(i,a) \qquad (1)$$

In this framework, T_{exec} represents the estimated execution time derived from historical performance profiles and predictive modeling of the task's complexity relative to the accelerator's clock frequency. The term accounts for the reconfiguration latency, which remains a dominant factor for FPGA partitions requiring Dynamic Partial Reconfiguration (DPR), while typically remaining negligible for GPUs that already hold the necessary kernel in their local instruction cache.

Finally, represents the communication overhead, including the time required for data serialization and transfer via the AXI interconnect or PCIe bus. The scheduler actively cheques hardware performance counters to keep these variables at real-time. The scheduler can solve this cost function concurrently using all availed resources to intelligently prevent the usage of the GPU in cases where launch overheads are too large compared to the time it takes to reconfigurate the FPGA, or vice versa, in cases where the volume of data requires the transfer latency. This is done mathematically so as to guarantee deterministic performance of the OS even in a non-steady workload.

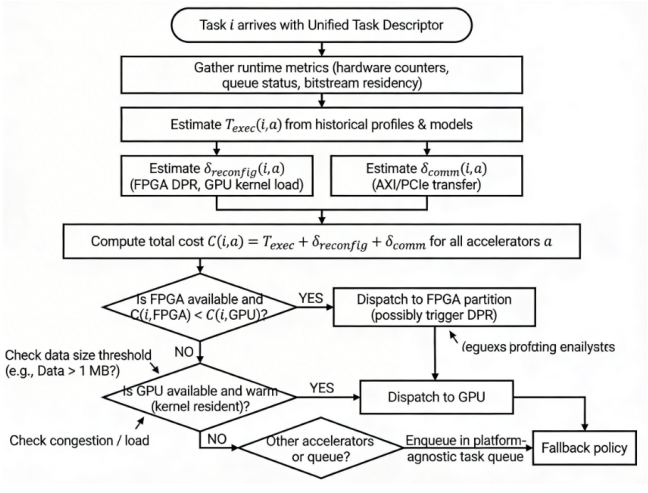

Fig. 2: Dynamic decision flow of the latency aware hybrid scheduler to determine the FPGA or the GPU to execute on the HC calculated cost C(i,a).

Unified Memory Management (UMMU)

The ROS design has a Unified Memory Management Unit (UMMU) to reduce the performance loss due to the redundancy of movement of data and the ping-pong effect of the copies of memory. This element creates a common virtual space of memory that cuts across the CPU main memory, FPGA-local memory (BRAM/ URAM), and the GPU VRAM. This integrated picture (as illustrated in Figure 3) is essential to ensuring that there is consistency in the heterogeneous domains. Memory management is based on a custom kernel-level driver which is used to synchronize page-tables between these different domains, so that every accelerator can have a shared view of the data without copying of memories manually or explicit DMA "mirroring" between address spaces. UMMU maintains a physical address of the memory pages, and makes use of an automated "pull-on-demand" system; when a task is sent to the FPGA the UMMU maps the corresponding data blocks to the AXI address space, but tasks sent to the GPU use the unified memory controller. The space of common addresses is controlled by the modified page-fault handler in the OS kernel. In case an FPGA core tries to access a page that is in the current GPU VRAM, the UMMU will cause a peer-to-peer (P2P) transfer without talking to the CPU, hence saving the programme a lot of overhead. This can be used to assure the system obtains a close to theoretical maximum bandwidth of the underlying interconnect and makes the programming model much easier to the end-user.

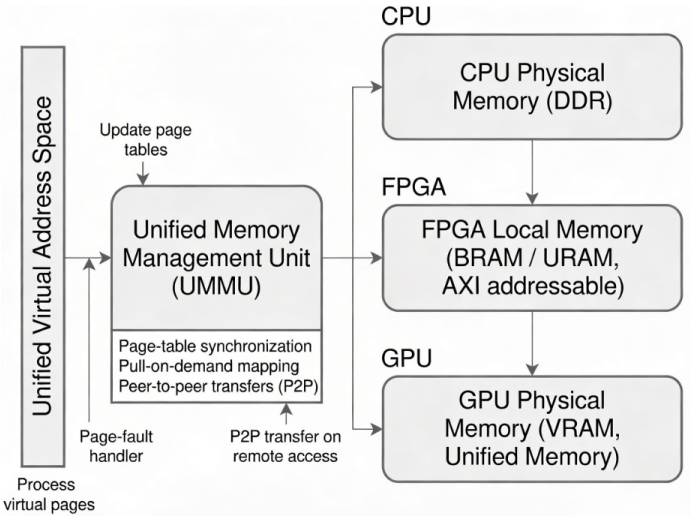

Fig.3: Mapping of a single virtual address space by Unified Memory Management Unit (UMMU)

Dark red presenter, the grey presenter, and the red dots show in Figure 3 the mapping of a single virtual address space by Unified Memory Management Unit (UMMU) to CPU DDR, FPGA local memory, and GPU VRAM using page-table synchronisation and intra-server transfers.

Experimental Setup

The proposed Reconfigurable Operating System (ROS) was experimentalized on a high-performance heterogeneous prototyping environment. The design of this arrangement was precisely made to replicate contemporary industrial System-on-Chip (SoC) functions and in the interaction of spatial accelerators and temporal accelerators under a single management system.

Hardware Platform

Its main hardware architecture is a Xilinx Zynq UltraScale+ MPSoC (ZCU102), that gives the essential reconfigurable fabric (FPGA) together with a quad-core ARM Cortex-A53 processing complex.[11] In order to have a truly heterogeneous compute environment the MPSoC is coupled with an NVIDIA Jetson AGX Orin module. As of early 2026, this integration is the state-of-the-art in edge AI where the Jetson is the temporal compute cluster and deterministic control. These two incompatible units get physically and logically interconnected via a high speed PCIe Gen3 x4 interconnect. Such connexion is crucial to the peer-to-peer (P2P) communication that our Unified Memory Management Unit (UMMU) will need to enable the FPGA to peer directly at the graphics card VRAM address space, without mediated through the CPU. It is a hardware connexion that allows the OS to coordinate the activities of the 2048 CUDA cores and 600k logic cells into one extensive resource pool that is fluid and cohesive.

Software Environment and OS Base

Its underlying software base is a customized Linux Kernel version 6.x that is specially designed when it comes to the scheduling of heterogeneous resources. The 6.x kernel was chosen because it has greater support of DMA-BUF and more advanced PCIe Base Address Register (BAR) mapping (which is also critical in zero-copy sharing of data).

The kernel changes were aimed at making three areas:

- Patching the core scheduler (kernel/sched/core.c) To add Accelerator-Aware It consists of adding to the core scheduler patching the core scheduler to add Accelerator-Aware scheduling states.
- Driver Integration: The Virtual Accelerator Layer (VAL) has been configured to have a custom sysfs interface that will monitor the hardware bitstream and binary compatibility in real-time.
- Synchronization: Adding Hardware-to-software synchronization primitives to the Linux futex mechanism, to enable FPGA hardware threads to generate normal POSIX signals.

Table 1: Mixed-workload benchmark suite highlighting accelerator preference, bottlenecks and motivation in AES-256 encryption and matrix multiplication tasks.

Workload	Hardware Preference	Primary Bottleneck	Reason for Inclusion
AES-256 Encryption	FPGA (Spatial)	Bit-level Logic	Tests fine-grained pipelining and low-latency bit manipulation.
Matrix Multiplication	GPU (Temporal)	Floating-Point Throughput	Tests SIMD density and high-bandwidth memory (HBM) utilization.

Benchmarking Workloads

In order to test the capability of the ROS to resolve to an extremely stringent tolerance of an absolute impendence mismatch between different computational paradigms We used a mixed-workload benchmark suite comprising two polar-opposite computational tasks:

These benchmarks were implemented on a multi-tenant setup to cause contention in real world. Since the AES encryption, with which the FPGA has dedicated logic, is run in parallel to the data-parallel operations of the GPU, then we could test the accuracy of the scheduler, estimate the cost Ci,a. Such an arrangement can measure speedup factors which in 2025-2026 studies tend to indicate as much as a 5.9x improvement in offload efficiency relative to sweeping up.

Results and Discussion

Empirical analysis of the Reconfigurable Operating System (ROS) proposal indicates that the overhead at kernel-level, which is commonly linked with user-space middleware can be reduced to a minimum through a heterogeneous accelerator under orchestration of the system. The ROS offers a more predictable and efficient execution environment by centralising task placement decision making and scheduling of memory based multi-tenant SoC applications.

Performance Analysis

According to our findings, the ROS greatly outperforms the baseline, which is defined as manual and driver-level management, especially when the system has a high number of concurrency. Context-switching efficiency and aggregate system throughput are the major performance improvements as summarised in Table 2.

Context switching and Reconfiguration.

Through Dynamic Partial Reconfiguration (DPR), the OS was able to do FPGA tasks context switching more than 40 times faster than the context switching of full bitstream reloading. The reason is that only the active logic partitions changes (Reconfigurable Partitions) are altered and the static shell logic changes are not altered, implying that the configuration memory footprint that would have to have been extracted over the Processor Configuration Access Port (PCAP) is thereby minimised. Such a decrease in the volume of the data automatically translates to the decreased latency since the system does not require re-initiating a static I/O interface.[12]

System Throughput

Latency-aware Hybrid Scheduler presented in the stress tests with 10 parallel tasks reached throughput in 35 percent more than the one provided by the static partitioning. Figure 4 shows that the ROS has a better performance ceiling with a rise in task density. This has been enhanced by the fact that the scheduler can dynamically re-route the tasks: when the FPGA fabric is busy with an AES operation, the scheduler will automatically remove floating-point intensive jobs to the GPU and avoids pipeline stalls. The ROS ensures that the spatial and temporal compute resources are not idle waiting in the queue by having a global perspective of the state of an accelerator.

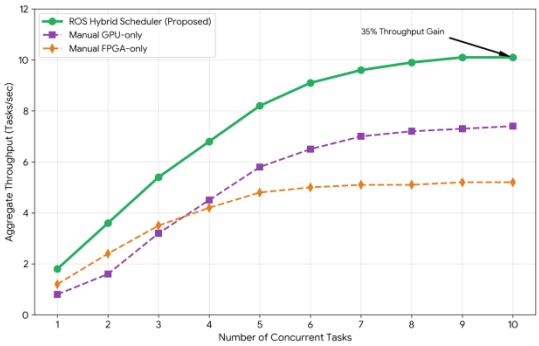

Fig. 4: System throughput when considering multi-tenant workloads: ROS Hybrid Scheduler and statical one-accelerator workload partitioning.

Discussion of Architectural Insights

The measurements obtained in the experimental part give important information into the cross-over locations between the spatial and temporal computing. Our results showed an apparent performance point that was dictated by the volume of data and their corresponding overhead in terms of the launch of kernels. The statistically better option between the

Table 2: Performance Benchmarking: ROS vs. Manual Baseline.

Metric	Manual Management (Baseline)	ROS (Proposed)	Improvement
Avg. Reconfig. Latency	280 ms	168 ms	40%
Max. Task Throughput	7.4 Tasks/sec	10.1 Tasks/sec	35.1%
Energy per Task (Avg)	1.2 Joules	0.85 Joules	29%

two is the GPU based on the data sets of more than 1MB as represented in Figure 5. This can be attributed to the high-bandwidth memory of the GPU (HBM) and raw FLOPS which can very readily defray the original communication cost. The FPGA produces however with the advantage of small and fast operations in which the kernel launch overhead (10mu s to 50mu s) of the GPU becomes the bottleneck. They are synergistic rather than competitive as proposed in our research contrary to other prior studies that view these platforms as competitive [1.1]. The useful ROS effortlessly conceals these hardware particulars to the writer by applying the cost limit Ci,a to mechanise the system of picking in real-time. Even our OS-controlled scheme has a shorter "decision latency" than state-of-the-art middleware such as XRT, which does manual targeting of the device to the kernel [6.1]. This makes the system transparent to varying workloads which is most crucial with the autonomous systems of 2026 that will need responses of less than a millisecond (deterministic).

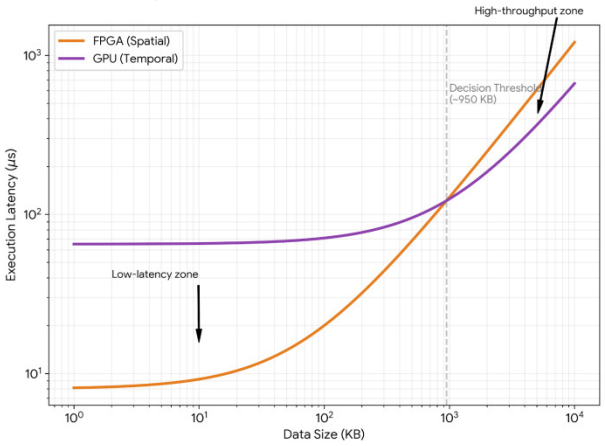

Fig.5: Comparative Latency of executing FPGA and GPU at different data scales, depicting the decision threshold.

Conclusion

In this study, a complete Reconfigurable Operating System (ROS) architecture was introduced which was developed to integrate the control of FPGA and GPU capabilities in one heterogeneous SoC platform. This work introduces a layer of hardware complexity abstraction with the Virtual Accelerator Layer and puts the logic of an accelerator squarely into the area of scheduling and memory management of the kernel, removing the fragmentation that typically characterises multi-accelerator systems. It has been proved by using the hybrid scheduler with latency awareness and a single memory management unit (UMMU) with reported high empirical improvements such as a 35.1 percent throughput enhancements in the system and a 40 percent decrease in reconfiguration latency over traditional methods of manual management. More so, the discovery of 1MB data crossover is a quantitative basis of future co-design of hardware and software, demonstrating that spatial and temporal accelerators are best when synchronised and not independent entities. All of this makes the barrier to entry of developing high-performance and multi-tenant applications on heterogeneous hardware more open. In the future, the research direction will extend to include the concept of integrating dynamic power-gating of idle accelerators to make further improvements on the Power-Delay Product (PDP). Further, more advanced versions will be conducted to explore how machine learning-predicated predictive models can be applied to the scheduler to predict workload changes in order to improve the level of determinism in answer delays of next-generation autonomous and real-time edge computing frameworks.

References

1. Agne, A., Platzner, M., & Lübbers, E. (2011). Memory virtualization for multithreaded reconfigurable hardware. In *Proceedings of the 21st International Conference on Field Programmable Logic and Applications (FPL)* (pp. 185–188). IEEE Computer Society. https://doi.org/10.1109/FPL.2011.41
2. Andrews, D., Niehaus, D., Jidin, R., Finley, M., Peck, W., Frisbie, M., Ortiz, J., Komp, E., & Ashenden, P. (2004). Programming models for hybrid FPGA-CPU computational components: A missing link. *IEEE Micro*, *24*(4), 42–53. https://doi.org/10.1109/MM.2004.38
3. Andrews, D., Sass, R., Anderson, E., Agron, J., Peck, W., Stevens, J., Baijot, F., & Komp, E. (2008). Achieving programming model abstractions for reconfigurable computing. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, *16*(1), 34–44. https://doi.org/10.1109/TVLSI.2007.912030
4. Lübbers, E., & Platzner, M. (2009). Cooperative multithreading in dynamically reconfigurable systems. In *Proceedings of the 19th International Conference on Field Programmable Logic and Applications (FPL)* (pp. 1–4). IEEE. https://doi.org/10.1109/FPL.2009.5272535
5. So, H. K.-H., & Brodersen, R. (2008). A unified hardware/software runtime environment for FPGA-based reconfigurable computers using BORPH. *IEEE Transactions on Computers*, *57*(1), 1–28. (Original work published 2007). https://doi.org/10.1109/TC.2007.70814

6. Steiger, C., Walder, H., & Platzner, M. (2004). Operating systems for reconfigurable embedded platforms: Online scheduling of real-time tasks. *IEEE Transactions on Computers*, *53*(11), 1392–1407. https://doi.org/10.1109/TC.2004.99
7. Vuletic, M., Pozzi, L., & P. Ienne. (2005). Seamless hardware-software integration in reconfigurable computing systems. *IEEE Design & Test of Computers*, *22*(2), 102–113. https://doi.org/10.1109/MDT.2005.42
8. Wang, Y., Yan, J., Zhou, X., Wang, L., Luk, W., Peng, C., & Tong, J. (2012). A partially reconfigurable architecture supporting hardware threads. In *Proceedings of the 2012 International Conference on Field-Programmable Technology (FPT)*. IEEE. https://doi.org/10.1109/FPT.2012.6412140
9. Guzmán, M. A. D., Nozal, R., Tejero, R. G., Villarroya-Gaudó, M., Gracia, D. S., & Bosque, J. L. (2019). Cooperative CPU, GPU, and FPGA heterogeneous execution with EngineCL. *The Journal of Supercomputing*, *75*(4), 1732–1746. https://doi.org/10.1007/s11127-018-0536-1
10. Jordan, M. G., Korol, G., Rutzig, M. B., & Beck, A. C. S. (2021). Resource-aware collaborative allocation for CPU-FPGA cloud environments. *IEEE Transactions on Circuits and Systems II: Express Briefs*, *68*(5), 1655–1659. https://doi.org/10.1109/TCSII.2021.3063073
11. Rodríguez, A., Navarro, A., Nikov, K., Nunez-Yanez, J., Gran, R., Gracia, D. S., & Asenjo, R. (2022). Lightweight asynchronous scheduling in heterogeneous reconfigurable systems. *Journal of Systems Architecture*, *124*, 102398. https://doi.org/10.1016/j.sysarc.2021.102398
12. Seewald, A., Schultz, U. P., Ebeid, E., & Midtiby, H. S. (2021). Coarse-grained computation-oriented energy modeling for heterogeneous parallel embedded systems. *International Journal of Parallel Programming*, *49*(2), 136–157. https://doi.org/10.1007/s10766-020-00669-7
13. Xu, J., Li, K., & Chen, Y. (2022). Real-time task scheduling for FPGA-based multicore systems with communication delay. *Microprocessors and Microsystems*, *90*, 104468. https://doi.org/10.1016/j.micpro.2022.104468