International Journal on Recent and Innovation Trends in Computing and Communication
ISSN: 2321-8169 Volume: 11 Issue: 5
Article Received: *25 March 2023* ***Revised:*** *12 April 2023* ***Accepted:*** *30 May 2023*

Hardware/Software Co-design: Addressing Uncertainty in Platform Development through Workload Modeling and Bottleneck Feedback Loops

Ankush Jitendrakumar Tyagi[1*]

[1]University of Texas at Arlington, Texas, USA

Correspondence to:

Ankush Jitendrakumar Tyagi[*]

[*]University of Texas at Arlington, Texas, USA, ankush8tyagi@gmail.com

Abstract

Hardware/software co-design is now a vital paradigm of next-generation computing platforms, especially in applications where efficiency, power, and flexibility are of paramount importance. Due to the increased complexity of applications and the evolution of platforms to support varying and changing workloads, the problem of design uncertainty multiplies. This ambiguity, which most commonly arises due to the unknown nature of workload, changing requirements on the amount and type of resources, as well as hardware limitations, manifests itself in the sub-optimality of the resulting system and slackened development schedules. One possible solution to alleviate these problems is by including in-depth workload predictive modeling and feedback loops of bottlenecks into the co-design process. Workload modeling allows abstracting the behavior and simulating a real-world application at an early stage of design. Proper workload models capture patterns of data flow, levels of computation, and access patterns to memory, enabling platform architects to make great hardware settings and software scheduling choices. To supplement this, bottleneck feedback loop mechanisms, which are iterative systems to detect performance-constraining elements and then react upon them, are proposed as a continuous design improvement system. These loops offer us suggestions to act by identifying constraints that exist in systems so that specific actions can be taken to refine the systems, both hardware and software. Such an integrated approach allows for improving the predictability and flexibility of platform design by matching the capability of hardware with what the applications need. It is also flexible to validate iteratively, which means any differences in the theoretical performance with the observed one can be eliminated early and efficiently. Real-world applications in edge computing, autonomous applications, and high-performance embedded systems show how this method achieves extraordinary savings in design risk, increased resource utilisation, and faster design cycles. Teaming workload modelling and feedback processes in a framework of coded subsystems of hardware/software design provides a prospective approach to the uncertainty hurdle with the view of a robust, energy-efficient, and application-sensitive computing pathway. This paper explores how hardware/software co-design supports real-world systems, including industrial IoT, edge computing, and autonomous vehicles. Through workload analysis and adaptive feedback, co-design aligns hardware capabilities with evolving software demands, helping to manage uncertainty, boost energy efficiency, and deliver scalable, high-performance solutions in complex operational settings.

Keywords: Hardware/Software Co-design, Workload Modeling, Bottleneck Feedback Loops, Platform Development Uncertainty, Embedded System Optimization

International Journal on Recent and Innovation Trends in Computing and Communication
ISSN: 2321-8169 Volume: 11 Issue: 5
Article Received: *25 March 2023* ***Revised:*** *12 April 2023* ***Accepted:*** *30 May 2023*

Abbreviation	Full Form
AI	Artificial Intelligence
API	Application Programming Interface
ASA	Application-Specific Architecture
ASIC	Application-Specific Integrated Circuit
CPU	Central Processing Unit
CPS	Cyber-Physical Systems
DSE	Design Space Exploration
DVFS	Dynamic Voltage and Frequency Scaling
FPGA	Field-Programmable Gate Array
GPU	Graphics Processing Unit
HLS	High-Level Synthesis
HPC	High-Performance Computing
HW/SW	Hardware/Software
I/O	Input/Output
IEC	International Electrotechnical Commission
IEEE	Institute of Electrical and Electronics Engineers
IoT	Internet of Things
ISO	International Organization for Standardization
ML	Machine Learning
MPSoC	Multiprocessor System on Chip
OCP	Open Compute Project
QEMU	Quick Emulator
RISC-V	Reduced Instruction Set Computing – Version Five
RTOS	Real-Time Operating System
SoC	System on Chip
TEE	Trusted Execution Environment
TLM	Transaction Level Modeling
UAV	Unmanned Aerial Vehicle
VLSI	Very-Large-Scale Integration
WCET	Worst-Case Execution Time

1. Introduction

The accelerating change in computing technology has drastically replaced the picture of embedded systems, high-performance platforms, and edge computing architectures. Since software capabilities have evolved faster than those of hardware due to new developments in artificial intelligence (AI), real-time processing, and ubiquitous connectivity, the mismatch has been aggravated. To fill this gap, hardware/software (HW/SW) co-design as a system design approach is one way in which developers are going to get around. Contrary to historic linear development models in which hardware is systematically developed and then software, HW/SW co-design establishes a kind of engineered cooperation, in the same way that both circles of work are done along with each other. This approach is becoming a key to constructing high-performance yet flexible, scalable, and energy-efficient computing platforms [1-4]. In the past, the system design was a serial task. Microarchitecture decisions were made by the hardware teams following specifications, and those decisions were set in stone; software teams subsequently adjusted the applications to execute on the products that were created. Although it is appropriate in fixed-function computers with determinate workloads, the method is not appropriate in modern computing facilities. Various systems supporting dynamic workloads exist in edge AI, autonomous vehicles, smart manufacturing, and 5G networking applications, where system designs need to support a wide set of dynamic and changing workloads. Such workloads have different levels of computation intensity, the requirement of memory and

communication requirements. This forces platforms to be highly varied, something that can hardly be accommodated without initial software and hardware collaboration. Another key issue that emerges in such contexts is called design uncertainty, which refers to the changes in platform development at the beginning which are unpredictable and incomplete knowledge. Uncertainty is caused by many factors: the changing software needs, the emerging hardware parameters, the unknown target deployment conditions, and the changing non-functional requirements such as power budgets, latency bounds, and so on. Consider a scenario where a neural network operates on an embedded GPU. As the model size increases or input complexity rises with the release of new software versions, overall performance may degrade. When non-anticipatory in nature, these mismatches may lead to expensive redesigns and performance deficit [3-5].

HW/SW co-design provides the solution to this problem since, at its base, it is suggested to implement a cross-domain decision-making. It allows system architects a chance to collectively optimise processing elements, memory hierarchies, communication channels, and software task mapping. Instead of siloing hardware and software, co-design considers the platform a novel, design-determined ecosystem in which design decisions in any one area have an immediate impact on design choices in the other. In particular, such co-processor collaboration is critical in heterogeneous computing systems where CPUs are united with GPUs, FPGAs, as well as other accelerators in satisfying various computational needs. The HW/SW co-design has its complexity regardless of the benefits it poses. The design space is exponentially expanded and difficult to explore due to the necessity to examine numerous different configurations and take into consideration unpredictable behaviour. This is where bottleneck feedback loops and predictive workload modeling prove to be a very useful tool. Workload modeling allows computer programmers to project application behaviours in the initial phases of design, including the patterns of data movement, concurrency levels, as well as memory access distributions. The models assist in making educated guesses regarding the performance of an application in a certain hardware configuration, to make better architecture, scheduling, and task partitioning choices [1,2,5]. In complement with modeling, there is the principle of the bottleneck feedback loops, which consists of an iterative system to check the performance during deployment, then detect the bottleneck in real-time and suggest or enforce the cure. The loops play a major role in ensuring the performance of systems in various operating conditions. In cases where the factors that determine the nature of workloads change because of forces in the environment, user input, or updates in the application, the feedback mechanisms will determine that the platform adjusts, instead of becoming worse. This approach transforms the perception of system design from a static, one-time process into a continuous cycle of observation, diagnosis, and improvement. The integration of modeling and feedback as core mechanisms within a unified hardware/software co-design framework significantly advances the development of resilient and future-ready platforms. When combined, these elements enable the creation of systems that are not only dynamic but also capable of being tested with high precision and adapted progressively over time. This methodology proves particularly valuable in critical sectors such as automotive safety, medical diagnostics, and aerospace engineering, where system failure is unacceptable and adaptability holds paramount importance [1-5].

This paper discusses how synergy between workload modeling and feedback loop can be an effective method of uncertainty management in platform development. To begin, an examination of uncertainty in hardware/software systems is necessary, along with an exploration of the limitations inherent in traditional approaches when addressing such uncertainty. It then explores the rationale and values of HW/SW co-design, followed by elaborate accounts of workload modeling and feedback loop systems. Thus, methods for integrating these concepts into design processes are examined, along with an overview of broader trends that suggest the future direction of adaptive platform engineering. The latter boils down to this article suggesting that the next generation of intelligent, efficient, and reliable computing systems can be unlocked, not only through selection of improved hardware and by writing smarter software, but also through a top-to-bottom system co-design between these two components that has the topology of adaptability, guided by predictive insights and feedback through the bottom.

International Journal on Recent and Innovation Trends in Computing and Communication
ISSN: 2321-8169 Volume: 11 Issue: 5
Article Received: *25 March 2023* ***Revised:*** *12 April 2023* ***Accepted:*** *30 May 2023*

2. Understanding Uncertainty in Hardware/Software Platform Development

As was described above, uncertainty is problematic on many levels-application, architecture, timing, and system environment in modern computing systems. This uncertainty is especially common in the hardware/software co-design, where it is necessary to make the decision regarding hardware architecture simultaneously with software development. When wrongly matched, such decisions may result in poor system efficiency and performance of the system, with some bottlenecks and incompatibility among other factors that may only show during the later testing stages. This complexity is compounded by the increasing heterogeneity of hardware elements as well as the unpredictability of workloads. One of the main reasons for uncertainty is the variability of workloads in nature. In contrast to the relatively lodged functionality of traditional embedded designs, the current platforms now host a broad spectrum of dynamic applications, including real-time video processing and AI inference, edge analytics, and autonomous control. Such workloads change frequently following the changing data entry or user behaviour. An example is when a sensitive surveillance system encounters a surge in processing as a result of time or environmental activity. Likewise, edge applications based on AI-based models are highly extensible, making the requirements on the resources used dramatically different [5]. This brings in the functional uncertainty as discussed by design specialists, where the exact behaviour of the software is largely not known at the design stage. This has necessitated the hardware platforms to be adequately malleable in order to adapt to future changes with minimal cost of reconfiguration. This is unlike classical ASIC design when all the transistors are optimised around known and fixed functions [6]. Such inflexibility in today's systems would cause the hardware to become obsolete much earlier than its hardware demise.

The third type of uncertainty is the one that is caused by non-deterministic timing behaviours, especially in tools concerned with real-time embedded applications. A version of overhead and latency due to operating systems, middleware, and communication protocols depends on the context of execution. Once these factors cannot be considered adequately in the early design stages, it is likely that the final system would not adhere to deadlines, particularly when the behaviour of workloads is rendered unpredictable. This explains why temporal predictability is a key issue with respect to systems such as automotive ECUs or medical devices with deadlines becoming missed, which may lead to disastrous effects [7]. There is also some ambiguity in design brought on by energy consumption and thermal profiles. The over-use of power budgets may limit the usability of an item or in some cases, simply fail the system in battery-powered or heat-sensitive applications, such as wearables or drones. However, energy use is usually workload-specific and cannot be accurately forecasted until the software is in operation. This complicates the estimation of initial design and causes a demand of modelling methods that can imitate realistic consumption profiles and environmental bounds [8].

Furthermore, the emergence of multi-core and heterogeneous processing processors such as the CPUs, GPUs, DSPs, and FPGAs introduces an uncertainty in resource consumption and parallelisation performance. In the absence of concrete insight about how the work should be split and compliant with time by the processing elements, the developers will either over-provision on the hardware equipment or under-provision, which ends up in bottlenecks and poor performance. The task-to-resource mapping problem, an issue of allocating tasks to the hardware components expediently, becomes a non-trivial issue in the presence of such uncertainty [9]. External dependencies and environmental conditions also bring in a multiplier of design uncertainty. As an example, systems installed in distant settings (e.g., satellites or offshore platforms) need to cope with changes of temperature, radiation, or network latency. Such circumstances affect the reliability or performance of both hardware and software, and there is a strong need to consider uncertainty in the environmental conditions at the beginning of the design process.

Not to make things any easier, system requirements tend to be in flux. The priorities of the stakeholders might change in the middle of its development, e.g., switching between throughput focus to energy efficiency, which would require system architects to change the strategy of the hardware/software divisions. This creates the cycle of re-designs and re-testing that only grows the scope of the project, both in time and budget, unless managed with more pliable design practices. All these types of uncertainty show that the situation of fixed hardware at the beginning and developing the software after it cannot be applied anymore. Rather, there is a need to design

International Journal on Recent and Innovation Trends in Computing and Communication
ISSN: 2321-8169 Volume: 11 Issue: 5
Article Received: *25 March 2023* ***Revised:*** *12 April 2023* ***Accepted:*** *30 May 2023*

concurrently, which involves iterative, predictive, and results in feedback via simulation-enabled decision-making at each step in the development process. It requires the mechanisms to enable developers to model, observe, and act upon the way systems change in real time as the system evolves. This sets the stage for the following section, which explores the fundamentals and motivations of hardware/software co-design as a core paradigm for addressing ambiguity in the context of system-wide performance optimisation.

3. Principles and Motivations of Hardware/Software Co-design

As application complexities grow and requirements in responsive, adaptable computing platforms rise, the notion of hardware/software (HW/SW) co-design has come up as a central outcome in the development of embedded systems and high-performance systems. HW/SW co-design in the basics is the simultaneous and collaborative designing of hardware and software parts of a system with the goal of optimisation of the performance, energy and cost. This mode of design takes issue with the conventional tech stack in which the software is built over a pre-existing hardware, making the process quite inefficient or a lack of equality. Figure 1 highlights core principles like concurrent development, partitioning, and co-verification, alongside key motivations such as performance optimization, power efficiency, and design flexibility. Co-optimisation: this is the crunch concept of HW/SW co-design, making trade-offs and decisions that do not ignore either the software behaviour or the hardware constraints. This involves the removal of the walls between the hardware engineers and the software developers and the encouragement of a feedback-driven iterative design cycle. In this cycle, the needs for software applications influence the needs of hardware, and hardware capabilities inform assumptions about software structure and software performance expectations [10]. The most important result is a higher-level system architecture that fits well with the actual requirements of the application, as opposed to a one-size-fits-all.

The capability of addressing strict performance and power budgets is also one of the strongest reasons to resort to HW/SW co-design. Implantable medical devices, aerospace, automotive, and mobile computing systems are limited to egregious size, weight, energy, and real-time processing constraints. It is sometimes necessary to use domain-specific hardware accelerators (such as GPUs, FPGAs) or even custom ASICs, in order to best optimize performance within such limits. The effectiveness of such accelerators, however, lies much in the structuring of software to efficiently make use of these accelerators. The HW/SW co-design allows the designer to develop custom instruction sets, memory hierarchy, or interconnect that corresponds to the access pattern and computational requirements with software [11]. Also, time-to-market pressure is one of the forces that result in the codeign movement. Competition is growing, and the iteration cycles during product development are becoming too long due to late-discovered hardware/software incompatibilities which cannot be afforded by companies anymore. The risk is overcome by HW/SW co-design, which promotes co-validation as early as possible (using simulation, prototyping, and emulation). Software and hardware synthesis thereupon can take place together on the same platform, and feedback is instantaneous; examples include Xilinx Vitis or Intel OneAPI [12].

The next important principle is design-space exploration (DSE). This includes testing several potential system designs, e.g., different types of processors, different memory hierarchy structures, or bus schemes, and comparing their tradeoffs in throughput, latency, area, power, and cost. With a co-design methodology, the DSE is much more effective as the hardware is modelled and so is the software. It enables architects to find a Pareto-optimal solution between conflicting goals, like performance vs. energy or costs vs. scalability [13]. The requirements of the heterogeneous computing environments, which require CPUs, GPUs, DSPs, and FPGAs to be utilized simultaneously, in order to satisfy various processing requirements, also benefit from the help of co-design. In those settings, every device has distinct advantages: CPU is general-purpose, GPU has high parallel, and FPGA and flexible. HW/SW co-design allows the designer to divide the work in an intelligent way amongst these elements based on issues that include cost of data movement, latency demands, and computational demand [14].

In addition, HW/SW co-design encourages the development of application-specific architectures (ASAs) and systems-on-chip (SoCs). Increasing applications are being found in fields such as autonomous vehicles and machine learning, where a general-purpose processor proves inadequate. Within this context, co-design enables programmable pipeline

International Journal on Recent and Innovation Trends in Computing and Communication
ISSN: 2321-8169 Volume: 11 Issue: 5
Article Received: *25 March 2023* ***Revised:*** *12 April 2023* ***Accepted:*** *30 May 2023*

structures, application-specific accelerators, and memory hierarchies closely integrated to the specific application computation graph, all optimised to the specific application [15]. The co-design paradigm has also received a boost with the bug toward an open hardware ecosystem, like the RISC-V. Designers are now able to change and extend hardware instruction sets, and, since it allows individual modification of hardware, co-design allows designers to optimise the processor to particular workloads. An example is that in cryptography, custom instructions to perform modular arithmetic can achieve much higher speed when co-designed with accompanying software library routines [16].

Notably, principles of the HW/SW co-design do not refer only to applications that are performance-oriented. The solutions are also advantageous in scenarios where cost is not the primary concern, particularly in contexts that prioritise security, reliability, or maintainability. An example of this is running the software in a safety-conscious system, such as avionics or medical equipment, where, to achieve software reliability, the software should be balanced by the hardware's fault tolerance. The co-development of error-checking software routines and redundancy mechanisms in hardware can be performed with co-design so as to meet the regulatory requirements [17]. Finally, HW/SW co-design attracts lifecycle flexibility. Systems developed under this paradigm offer greater ease of future upgrading, restructuring, or expansion, due to the presence of modular and parameterised segments. It is especially useful in products with long product lifetimes, such as industrial controllers or military equipment, where changing operational requirements or component obsolescence would repeatedly necessitate changes to systems without complete redesign [18].

In a nutshell, the concepts and objectives of hardware/software co-design all merge at one point: designing efficient, customized, and future evidence-based computing systems that could close the gap between the demands of contemporary software and the potential of hardware. This structure forms a basis to address the unpredictability mentioned in the previous section and present mechanisms, like workload modeling, which gives a better idea of application interactions with architecture establishment. The following section addresses this important aspect in greater detail.

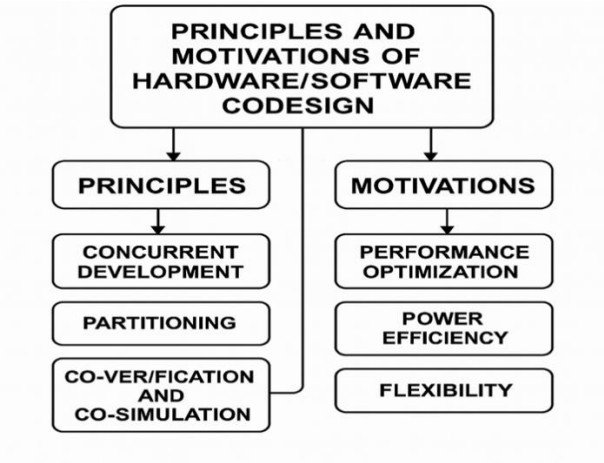

Figure 1: Principles and motivations underlying hardware/software co-design methodologies.

4. Workload Modeling for Informed Design Decision-Making

Workload modeling is one of the most potent tools that can be used by system architects to effectively navigate the uncertainty surrounding the development of platforms. Workload modeling involves an abstract representation and simulation of software application behaviour, illustrating interactions with hardware resources over time. Workload models serve as a critical component in hardware/software co-design, offering a forward-looking perspective on application requirements. This foresight enables consideration of alternative hardware and software configurations and supports optimisation well before final implementation is achieved. Figure 2 illustrates an iterative process involving workload definition, performance analysis, simulation, and evaluation of design alternatives leading to final design decisions. Fundamentally, workload modeling is meant to be concerned with the capturing of the main contours of the execution behaviour of an application. Some of these metrics include instruction mix, computational intensity, data access patterns, memory bandwidth use, parallelism, and communication overhead. By measuring these parameters, the designer is able to determine how various workloads will exercise the system and what resources, such as CPU cycles, memory hierarchy, and I/O bandwidth, will be the likely bottlenecks [19].

What is referred to as early-stage design space exploration (DSE) is made possible by a good workload model. This provides information on how software loads can be mapped to candidate hardware architectures,

enabling trade-off analysis based on energy consumption, performance, cost, and complexity. As an example, a workload model shows that a high level of data locality and a low level of control flow complexity could result in designers choosing FPGA acceleration. In contrast, general-purpose processors or reconfigurable platforms might be favoured by workloads with a nonstandard memory access pattern [20]. Workload modeling can be especially beneficial in an iterative co-design process to describe ever-changing hardware configurations to changing software requirements. Instead of inferring ad-hoc benchmarking measurements following deployment of the hardware, designers may model workloads in the design process, so as to make predictive comparisons of candidate architectures. This severely minimizes the instances of design failures and risks of encountering incompatibility with performance at the late stages of design [21].

The construction of a workload model typically begins with application profiling. Software is analysed using profilers such as GNU gprof, Intel VTune, or ARM Streamline, which generate execution profiles in the form of traces that report timing, instruction usage, and resource utilisation. Such traces may be translated into abstract models, in the form of control/data flow graphs (CDFGs), dependency graphs, or finite state machines [22]. Using new simulators such as gem5, Sniper, and SystemC-based environments, it is now possible to simulate these cache models under various hardware settings, which is useful to study the trade-offs between performance and energy consumption [23]. A still more potent tool is synthetic workload modeling in which models representing the whole application are synthesized when the entire application is not available. This comes in handy in cases where the software attacked is yet to be developed or may fall under privacy regulations. When that is the case, it is possible to model the probable behaviour of the workloads in terms of statistics or behaviour, which provides early indicators of design requirements. By taking into consideration known properties like the size of inputs, degree of concurrency, or complexity of algorithms, these models may be parametrised to achieve a reflective behaviour in the real world [24]. Besides directing hardware designs, workload modeling controls software scheduling techniques, as well as resource assignments. In a heterogeneous processor system, such as the model, can guide the designer on partitioning different components of an application across CPUs, GPUs, and FPGAs in such a way as to optimise the performance or energy consumption. Through such models, intelligent task partition is achieved, which involves the decomposition of software into small units whereby scheduling them in a resource-sensitive manner becomes achievable. This is especially applicable when there is a cloud and edge computing situation, where system resources are shared and are variable [25].

Moreover, the modeling of workload can assist in identifying the design-time bottlenecks in the workload, segments of the workload that overwhelm execution time or resource use. The discovery of these hot spots allows developers to conduct optimisation where it counts and not where generalised improvement is carried out. It is important in projects that are under a tight budget or schedule because it is vital to employ resources sparingly in order to optimise them [26]. One of the most important advantages of workload modeling is that it allows what-if analysis. Designers are able to assess performance changes in workload characteristics, whether a system is input size increased, higher concurrency, or no algorithmic improvement. This will make the system resistant to any change in application behaviour in the future, as well as sustainable of the platform as far as the platform will be in the future. To give just one example, the model can be used in the design of a video analytics edge device in order to measure the effect on CPU resource consumption and memory use of increasing resolution or frame rate so that design choices can be adjusted early [27]. Workload modeling can in addition, be combined with timing analysis frameworks to check that all safety-critical or real-time tasks are achieving their deadlines. This guarantees that there is budgeting of timing requirements as well as functional requirements, as far as co-design is concerned. Real-time schedulability testing and worst-case execution time (WCET) analysis are among the more common analysis techniques that may be employed on workload models as the starting point input [28].

Notably, workload modeling does not occur in intervals, but it is an ongoing process. When software changes, introduces new algorithms, or the pattern of usage of the system changes, the models need to be updated in accordance with new conditions. This dynamic adaptation will guarantee that the system will at all times stay at par with regards to performance objectives and providing effective operation throughout the lifecycle. With bottleneck feedback loops (next), such a synergy

International Journal on Recent and Innovation Trends in Computing and Communication
ISSN: 2321-8169 Volume: 11 Issue: 5
Article Received: *25 March 2023* ***Revised:*** *12 April 2023* ***Accepted:*** *30 May 2023*

between modelling and feedback effectively provides a strong tool of iterative system optimisation. In short, intelligent platform design is incomplete without workload modeling. It facilitates decisions based on knowledge and accurate data regarding the hardware architecture, software scheduling, and system settings. Workload modeling minimises uncertainty and better optimises the resource utilisation because it can simulate execution behaviours and identify early bottlenecks in the entire hardware/software co-design process, and it can establish a high-performance, resilient hardware/software co-design methodology. This section shifts focus to a description of how bottleneck feedback loops formalise workload modelling within a dynamic, runtime-aware optimisation process.

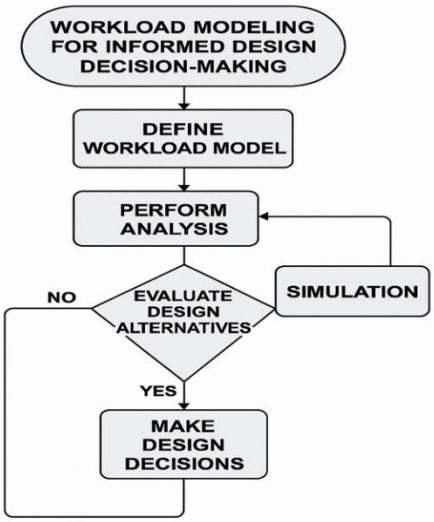

Figure 2: Workflow for workload modeling to support informed design decision-making.

In order to better depict the practical use of workload modeling in various fields, the table below summarizes the priority and characteristics of workload modeling determination per application class.

Table 1: Domain-Specific Workload Modeling Objectives and Characteristics

Application Domain	Primary Modeling Objective	Key Workload Characteristics	Example Tools/Techniques
Edge AI (e.g., smart cameras)	Real-time latency prediction	Irregular control flow, dynamic inputs	TensorFlow Lite Profiler, gem5
Autonomous Vehicles	Parallelism analysis and timing validation	Sensor fusion, high data rates, tight deadlines	Simulink, SystemC, OpenModelica
Financial Trading Systems	Throughput optimisation	Low-latency messaging, event-driven processing	Discrete Event Simulators, QEMU
Embedded Medical Devices	Power and thermal modeling	Predictable task sets, low duty cycles	ARM Streamline, Synopsys Virtualizer
Cloud-Edge Video Analytics	Bandwidth and compute resource modelling	High data throughput, inter-node communication	NS-3, Sniper, InfluxDB integration

Note: Each domain introduces unique challenges to workload modeling, requiring custom abstractions and metric tracking.

5. Bottleneck Feedback Loops: An Iterative Optimization Mechanism

Although workload modeling plays a vital role in predicting system behaviour and guiding decision-making during the early stages of design, real-world platforms often diverge from simulations once deployed. Non-determinism in data, state of a system, and environmental conditions may cause the emergence of new performance bottlenecks or constraints that could not have been noted in the modeling process. The

bottleneck feedback loops are the concept that has been proposed to solve these issues; these are the complementary mechanisms in hardware/software co-design. Such loops are more of iterative optimisation systems which help to detect, monitor performance degradation, and act on them; hence, the distinction of desired performance as compared to the performance of the system is bridged.

A bottleneck herein is a hardware component or a subsystem, software or their interface that hinders system bandwidth, latency, or wastes energy. Typical bottlenecks are CPU utilization, memory bandwidth, I/O, poor use of the cache, and communication delays among processing elements. These problems can severely degrade the throughput and violate real-time or energy requirements unless checked by constant monitoring and intervention, in embedded and high-performance software [29]. The actual concept of the bottleneck feedback loop is simple to some level: measure, analyse, act, and validate, and a repetition of these phases forms an endemic cycle in the life of the system. The measurement step gathers performance data in terms of sensors, profilers, or system logs. Internal tools like Linux perf, Intel Performance Counter Monitor (PCM), the NVIDIA product Nsight, and ARM DS-5 offer the raw information required to discern runtime behaviour. Performance counters are thus embedded in hardware components such as memory controllers or a communication bus in custom platforms, to offer low-level insight into an operation [30]. When data is gathered, the analysis stage points to the main bottlenecks. It entails anomaly detection of predicted throughput, latency spikes, idling cores or memory stalls. More advanced techniques can incorporate machine learning algorithms, decision trees or clustering to enable abnormal patterns to be detected. As an example, when memory latency distorts the proportional increase in a particular execution phase, the memory subsystem is identified as a bottleneck, which could be due the inefficient caching or data movement [31].

During the action phase, the system takes action by providing optimisations to combat the constraint identified. Response may range in complexity and severity as simple as migration of a task or changing the frequencies to as much of the complexity as reconfiguring an FPGA fabric, changing software flow of execution, or changing communication protocols. In a multicore scenario, such as this, the system may perform task migration to underutilised cores or bandwidth-intensive processes may be throttled to level the load [32]. The validation section will determine improvement in performance after the action is taken. Otherwise, the cycle represents additional measurements and improvements. Such iterative behavior makes the task performance tuning not a one-time effort; it is a dynamic process involving changes of system behavior as well as external changes in workload. More importantly, this will enable the platform to run as close as possible to its optimal configuration as the demands of the applications change with time [33]. These feedback-controlled mechanisms are particularly effective in highly heterogeneous and reconfigurable systems, e.g., hybrid systems with CPUs, GPUs, and FPGAs. When using technologies in such platforms, the relationship among components is intricate, and assumptions about the design made statically do not work. As an example, when GPU memory contention is the limiting factor in a vision processing application, the feedback loop can dynamically offload some of the preprocessing steps to a CPU or program the FPGA to do some of the filtering required, to balance the workloads [34]. End-to-end latencies can be surveyed by feedback loops, and the possibility of deadline violations can be announced. The system would then be able to modify scheduling techniques, priorities on tasks, or inactivation of non-essential parts to satisfy real-time requirements. Such a dynamism is crucial when applied in applications such as UAV controls or road safety systems, in which each millisecond can be critical [35].

Notably, such loops can also be used in long-term design choices. By recording the bottlenecks observed during operation, system architects will be able to sharpen their knowledge of workload behaviours and may use such information to refresh their workload models or the succeeding design cycle. This establishes a circle in-loop process involving running operational feedback to continue enhancing the accuracy of the design, efficiency of the design, and definition of resilience. In the long term, this minimises design risk, enhances resource usage, and mitigates changes in application demands [36]. Additionally, power-aware computing is supported by bottleneck feedback loops, where it is a matter of great concern in mobile, wearable, and edge devices. As renewable energy consumption is monitored in real time, the loop can also throttle or halt processes, even though in order to remain within power budgets, it can also reduce their responsiveness. Usual responses used in these loops are dynamic voltage and frequency scaling

(DVFS) and core parking and memory controller gating as the means of balancing performance and energy efficiency [37]. Research highlights the integration of machine learning to enhance feedback mechanisms. By analysing historical data, such systems can anticipate potential bottlenecks, enabling pre-emptive modifications before issues arise. As an example, algorithms of reinforcement learning have been used to optimise task allocation with heterogeneous processors, to learn over time about the optimal configurations to produce high performance in a given workload [38].

To conclude, feedback loop bottlenecks are an important addition in the development and evolution of hardware/software co-design. Such mechanisms allow systems to detect and address performance challenges in real time while also adapting and evolving throughout their operational lifespan. These loops turn rigid designs into living organisms through incessant cycles of observation, analysis, and optimisation that allow a living system to self-optimise, endure, and sustain itself. The integration of predictive workload modeling with feedback mechanisms into a unified design process offers a powerful strategy for addressing uncertainty in platform engineering capabilities. This concept is examined in the subsequent section.

To understand how feedback loops function across the different stages of a system's lifecycle, refer to Table 2.

Table 2: Lifecycle View of Bottlenecks, Detection Methods, and Corrective Actions

System Stage	Common Bottlenecks	Detection Techniques	Typical Adaptive Response
Design-Time	CPU overcommit, poor memory map	Static profiling, simulation trace analysis	Task reallocation, memory hierarchy redesign
Integration	Cache conflicts, I/O contention	Emulation + synthetic workloads, cache simulators	Cache tuning, I/O priority adjustments
Runtime	Thermal throttling, latency jitter	On-chip sensors, runtime profilers	DVFS, task migration, thermal load balancing
Post-Deployment	Resource drift, workload skew	Performance logging, AI-based anomaly detection	Firmware updates, real-time FPGA reconfiguration

Note: Effective feedback loops are context-sensitive and may rely on hybrid data sources including on-chip telemetry and software instrumentation.

6. Integrating Modeling and Feedback into Co-design Workflows

The above paragraphs have served as a basis to comprehend the scheme of workload modeling and bottleneck feedback loops working independently towards reducing the uncertainty in platform development. The true potential lies in the manner of integration, where unifying these methods within the hardware/software co-design process enables the creation of adaptive, resilient, and high-performance platforms. Optimized composite of these mechanisms will help in ensuring the decisions made at design-time and those being made at runtime are based on constant learning, empirical data, and systemic understanding. Figure 3 continuous loop from the modeling phase through design execution, feedback gathering, and adaptation, enabling iterative improvement and informed design adjustments. Its most fundamental concept is that of iterative refinement of a system's design, which says that the architecture of a system is not established during the initial synthesis but is still subject to ongoing feedback. In the typical development flows, the hardware would be completed early in the flow, and this would happen before the software was comprehensively stipulated; this would create expensive incompatibilities and a low degree of adaptability. In contrast, an integrated modeling-feedback workflow views the

platform as a dynamic artefact, which is also subject to revision should the behaviours of workload be better understood [39]. History of bottlenecks analysis feedback is applied to update the workload models, which, in turn, suggests the new cycle of the hardware/software adaptation.

Toolchain interoperability is a requirement to support this workflow. Their development environments should enable smooth migration of the modeling, simulation, synthesis, and performance analysis. As an example, when using a system simulation framework such as SystemC TLM (Transaction Level Modeling), evaluation of system performance can be estimated early. Performance monitor traces can subsequently be used to augment these models during deployment, ensuring a continuous flow of information throughout the development process [40]. In the co-simulation environment (e.g Cadence Virtual System Platform or Synopsys Platform Architect), it is possible to perform a combined evaluation of workload and hardware behaviour in a single virtual space, and thus more convenient to coordinate the changes. After construction, workload models are applied within virtual prototypes and emulation setups, enabling early validation of architectural assumptions well in advance of physical hardware deployment. Through this, artificial or actual workloads in the form of user-trace, application benchmarks or generated input patterns are injected into the system. Such collected performance data can be used in bottleneck analysis at the early stage so as to recognize the hotspots or inefficiencies in the design prior to the actual implementation process [41]. This significantly increases the design-space exploration and lowers the peril of failure of performance after silicon.

When the system enters into implementation and deployment stages, the runtime bottleneck feedback loops replace them, monitoring the system behaviour in a real working environment. There will be metrics like CPU/GPU usage, memory bandwidth, task completion latency, power consumption and thermal conditions that will be constantly read. These values are contrasted to simulated workload expected baselines. All deviations, gradual (e.g., resource drift) and sudden (e.g. contention spikes), cause triggering of optimisation mechanisms and raising alerts. An effective workflow can then transfer this run-time information back into the development tool chain, completing the design and deployment loop. An example of this is provided when a memory-bound routine is regularly identified by a runtime profiler, the workload model can be updated to take account of that, the memory controller can be reconfigured, or software memory access patterns may be re-designed. In FPGA-based systems, such insights could result in the redistribution of logic resources or the synthesis of the data paths with HLS (High-Level Synthesis) tools [42]. Moreover, autotuners (script-based, AI-trained or operator expert systems) can be designed to either take action during the feedback in real time or in batch mode. For example, a reinforcement learning agent may adjust parameters such as cache size, clock speed, or task priorities, selecting combinations that minimise latency or energy consumption. Tools like AutoTVM, OpenTuner, or a custom heuristic engine can be integrated into the workflow, allowing configurations to improve progressively over time as performance is continuously monitored [43]. This consolidated workflow becomes vastly useful in the case of an edge computing situation. The devices in the field are highly likely to run in a non-deterministic setting, and updating them physically is hard. To achieve such responsiveness, these systems can auto-calibrate to variability (i.e., by means of lightweight feedback mechanisms) or to pre-modelled variability (i.e., integrating workload model predictions in firmware). The edge nodes that prevent the loading of tasks on the basis of local bottlenecks analysis not only increase efficiency but also minimize upstream transmission of data, which saves bandwidth [44]. More importantly, the integration also facilitates design traceability and design validation, especially in safety-key or regulated domains. As data needs are satisfied by each optimisation or design choice and can be traced through the workflow, audit logs and compliance reports can be created to cover all basic and advanced needs. Particular to this, it is pertinent in areas such as automotive (ISO 26262), medical (IEC 62304), or aerospace (DO-254), in which regulatory certification requires marked evidence of systematic design and validation methods [45].

In the management of a team and project, cross-functional teamwork increases when modeling and feedback are integrated into the co-design process. All engineers & architects work on a common source of truth: the performance model, and perform a common loop of iteration. This is enforced by using feedback mechanisms to ensure that no team works in a vacuum and that the validation process is ongoing to rebuff any bottlenecks that pass downstream into production

International Journal on Recent and Innovation Trends in Computing and Communication
ISSN: 2321-8169 Volume: 11 Issue: 5
Article Received: *25 March 2023* ***Revised:*** *12 April 2023* ***Accepted:*** *30 May 2023*

releases. In short, the combination of predictive modeling and implied runtime feedback in a hardware/software co-design flow forms a self-healing evolutionary design process. It can help platforms to adjust at design-time and placement, and guarantee continued performance, efficiency, and reliability. The combination of all these approaches expedites time-to-market as well as minimizing development costs, increases the adaptability of such systems after deployment, and future-proofs complex embedded systems. The given integrated approach being in place, the design can be viewed as a continual dialogue between the system intent and its reality, one which is responsive to significant change and uncertainty and which transforms accordingly.

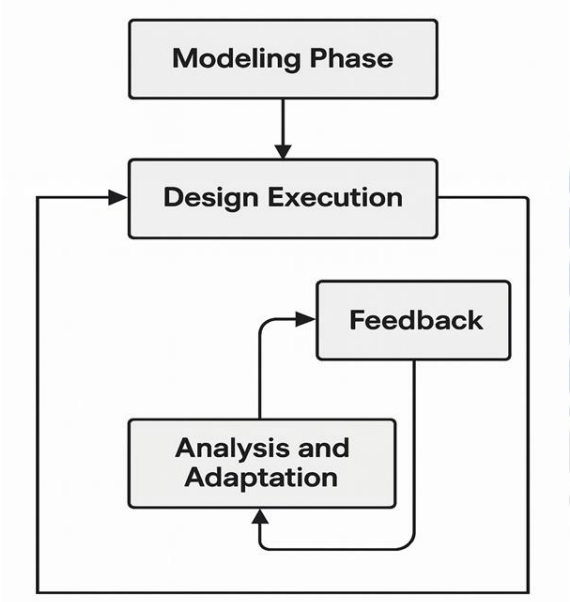

Figure 3: Workflow illustrating the integration of modeling and feedback within hardware/software co-design processes.

7. Future Trends in Adaptive Platform Engineering

The rise of both hardware heterogeneity and powerful and intelligent software systems, together with a dynamic set of application requirements, is driving platform engineering to a new era, one in which properties of adaptability, autonomy, and self-optimisation will be of paramount importance. Initial evidence of this shift is seen in the maturation of hardware/software co-design, driven by workload modeling and bottleneck feedback processes. Emerging trends are beginning to reshape the conceptualisation, design, and support of computing systems. These future trends in adaptive platform engineering are associated with a conversion of design movement to one comprising marginally static, deterministic, and design that can be conscious, the ascendancy of learning, and systems having the potential to develop after deployment.

Among the main trends in the near future, the emergence of self-adaptive systems is to be noted. These are systems capable of autonomously restructuring themselves according to actual changes of the workload or the environmental situation on a real-time basis. Self-adaptive architectures can be inspired by biological systems, making use of runtime monitors, predictive algorithms, and policy-based optimisers to change hardware parameters, task migrations or even change communication paths. As an example, an embedded system in a flying object could set the frequency of a CPU to low in a period where sensors note little activity to save energy or even offload heavy calculations to a cloudlet node when the activities are high [46]. Artificial intelligence (AI) and machine learning (ML) are also quickly becoming a trend where the machine learning (ML) and artificial intelligence (AI) are built into the platform engineering toolchain directly. This method, sometimes called AI-assisted design or learning enabled systems, applies ML models to help at all levels, including initial exploration of the design space, and optimisation at runtime. Rather than relying on manually constructed rules or heuristics, ML models will be trained on the history of performance to model the decision on potentially optimal configurations, compiler flags, task placements, or voltage/frequency settings. As another example, reinforcement learning has been applied (to dynamically schedule tasks across heterogeneous architectures to minimize latency and energy consumption) [47]. There is also a shift towards the full system digital twins-virtual models of the hardware/software systems that reproduce the behaviour of the systems in real time. These twins are an actual reflection of real-life situations as well as constant updating, the twins obtain through the flow of data from deployed devices. Digital twins can facilitate predictive maintenance, performance predictions, and identification of failures in a short period of time with the right models of workloads, thermal properties, and power consumption. Digital twins can be utilised in industrial environments during mission-critical operations, such as in industrial robotics or autonomous vehicles, serving as a sandbox environment for safely testing new workloads or firmware upgrades before deployment into active operation [48].

International Journal on Recent and Innovation Trends in Computing and Communication
ISSN: 2321-8169 Volume: 11 Issue: 5
Article Received: *25 March 2023* ***Revised:*** *12 April 2023* ***Accepted:*** *30 May 2023*

The other driver to adaptive platform design is open and composable hardware ecosystems. Companies focused on open hardware, such as RISC-V and Open Compute Project (OCP) have attempted to open the process so that designers can hybridize and fit or customise processors, memory interfaces, or accelerators to a particular workload. Such open ecosystems natively enable co-design at a level of granularity sufficient to enable developers to execute application-specific instruction sets, or reconfigurable pipelines that are optimised to specific niche domains like genomics or AI inference, or cryptography. Combined with workload-aware software stacks, these platforms go through improved performance per watt and quick response to new demands [49]. A second, more recent trend has been the reconfigurability of hardware at run time, especially when driven by features such as partial reconfiguration of FPGAs and the nascent dawn of programmable SoCs. Chip designs such as Xilinx Versal or the Intel Agilex now support on-the-fly reprogramming of parts of the chip and leaving others active. This enables systems to add new accelerators or update the protocols, or to provide security patches without reboot. When the results of bottleneck feedback loops are used to drive runtime reconfiguration, then a type of computational fluidity can occur, the flow of resources in response to changing needs [50].

Containerisation and microservices structure are at the software level, defining the future of adaptive systems. Through the breakdown of software into potentially deployable boxes that stand independently, systems can grow and change without major disturbances. Lightweight containers in the case of embedded and real-time systems, lightweight containers can now be used to roll out updates, a fresh AI model, or a change in operational parameters as needed. When combined with orchestration platforms such as Kubernetes or EdgeX Foundry, the adaptive systems are able to automatically manage load and find faults, and roll out new services in a distributed system [51]. Notably, security and trust will also emerge to become the centre pieces of adaptive platform engineering. As the platforms obtain the capacity to modify their own structure and behaviour, it is harder to guarantee their integrity. The integration of trusted execution environments (TEEs), runtime attestation, and AI-based anomaly detection represents a forward-looking approach to monitoring unauthorised changes and malicious behaviour. Protective mechanisms must be inherently embedded and adaptive, enabling detection and isolation of compromised components, as well as the rollback of updates in response to identified vulnerabilities [52].

Adaptive platform engineering overlaps with other fields of cross-domain/ interdisciplinary innovation, notably cyber-physical systems (CPS), edge AI, and neuromorphic computing. Such areas demand exceptional flexibility due to their inherent structural characteristics and high responsiveness to external influences. These are, e.g., in neuromorphic platforms which emulate brain-like behaviour and can rewire themselves with learned patterns, providing very energy-efficient adaptation mechanisms. In the same vein, CPS in smart grids or in smart factories has to readjust itself in real-time according to changes in sensor data or even in human interaction or power availability [53].

In the future, standardisation and interoperability frameworks will be necessary in scaling adaptive platforms in various industries. Interoperability frameworks in adaptive systems, to support plug-and-play hardware modules, standardised telemetry interfaces, and unified policy management, are already under development by organisations including IEEE, ISO, and the Industrial Internet Consortium. These conventions will allow the production of compatible modeling and feedback systems among dissimilar hardware agents and software platforms, even further speeding up the process of conquest [54]. Lastly, sustainability and empowering efficiency will emerge as the key design constraints to adaptive systems. The platforms of the future will have to fulfill not only their functional requirement but also operate under extreme environmental and regulatory conditions. The design will incorporate energy-aware workload modeling, green computing policies, and adaptive power gating methods to the design at base level. The bottleneck feedback loops will be imperative in the process of identifying the energy-hungry behaviours and implementing corrective actions in a dynamic manner [55].

Finally, intelligence, modularity, and resilience are the trends that characterise the future of adaptive platform engineering. The integration of modeling and feedback with AI and digital twins and open ecosystems will facilitate platforms responsive not only in real time, but also proactively optimized, naturally adaptable, and sustainable in terms of their environmental impact. The trends will climax into a new era of computing systems

that are in harmony with the complex and dynamic nature of the modern application.

At this stage, the article concludes by summarising the contributions and long-term value offered through the integration of workload modeling and bottleneck-driven feedback loops within hardware/software co-design strategies.

8. Conclusion

The growing interest in flexible, efficient, and high-performance computing facilities has led to the necessity of abandoning the traditional and rigid design processes towards flexible and intelligence-based developmental processes. To find an answer to this question, the paper has explained a business-wide framework that covers the uncertainty in hardware/software platform development by combining workload modeling and feedback of the bottlenecks into the process of hardware/software co-design. The article brings to the fact that there is ambiguity in designing platforms due to many factors changing and a dynamic nature, including: variable and evolving workloads, unexpected execution behaviours, changing resource demands, and dynamic operating conditions. Such uncertainties produce design risks as evidenced in the delays, poor performance of systems, and system redesigns at high costs, which cannot be properly addressed through traditional sequential design methods. To overcome this the paradigm of hardware/software co-design provides a synchronous, iterative paradigm in which co-design of the hardware and the software occurs with each affecting the configuration of the other. Such co-development allows a more adequate matching between the features of the platform and application needs, exploration of design space, and usage of custom architectures, optimised to meet domain-specific requirements. In this perspective, design choices are lean, rational, and data-based.

Workload modeling in this process gives a base that is predictive. It enables the system architects to model and learn the application behaviours at an early stage of system design. Workload models provide guidance on architectural tuning, resource allocation, and software scheduling by capturing the execution characteristics of the workload (data access patterns, instruction distributions, and task-level parallelism) and easy-to-read descriptions of all parameters. These models are adaptable and can be updated on a constant basis, and are more suitable when dealing with a platform that is long-lived or swift in development. Simultaneously, the bottleneck feedback loops allow monitoring the system performance in real-time and optimising the system on the fly. These loops find the factors limiting performance in both software and hardware through the measurement of real performance during execution and comparison with appropriate expected performance baselines. An automated pathway for implementing corrective measures is provided, encompassing re-configuration, task migration, frequency scaling, and dynamic scheduling. These feedback loops function as more than just reactive tools; they contribute to a continuous improvement process that refines system behaviour post-deployment and informs the development of future system design iterations. Workload modeling and bottleneck feedback loops yield a closed-loop, adaptive development structure when combined. This system minimizes the unknown by repeatedly verifying the assumptions and refining performance both during design and operation. It allows systems to naturally develop according to the changing needs, thereby enhancing resiliency, reliability, and lifecycle effectiveness.

Execution of this approach in the form of deployments in the real world can be seen in the form of edge computing, autonomous systems, AI inference, and industrial IoT. The platforms in these areas have extreme limitations on power, latency, security, and adaptability, and yet, still have to achieve high performance in extremely dynamic environments. Such platforms reach the market more quickly, utilise resources more efficiently, and experience fewer failures due to the integration of workload-aware models and perceptual real-time feedback mechanisms. These foundations form the basis of the future of adaptive platform engineering, as mentioned in the earlier section. As the use of AI-aided co-design, digital twins, reconfigurable hardware, and standardised open hardware ecosystems has grown, the capacities to scale and intelligence systems are expanding with it. The prospects of these advances include a future where the given platforms will be not only capable of supporting the needs of the current workloads but also one that will be able to preempt and respond to the requirements of the future. It can be summed up that with the inclusion of workload modeling, as well as a bottleneck feedback system, in the hardware/ software co-design process, there is an evolutionary jump in the manner in which modern computing platforms are perceived and brought into

existence. It allows a more interactive, precise, modular approach to engineering workflow, which is fundamental to the needs of next-generation applications facing an ever-greater complexity, ambiguity, and dynamism. With the increasing growth of embedded systems and a broadening of computational demands, the implementation of such adaptive approaches will prove central to the development of sound, effective, and future-resistant platforms.

References

[1] S. S. Bhattacharyya, E. F. Deprettere, R. Leupers, and J. Takala, Eds., *Handbook of Signal Processing Systems*. Springer, 2018.

[2] W. H. Wolf, "Hardware-software co-design of embedded systems," *Proc. IEEE*, vol. 82, no. 7, pp. 967–989, 2002.

[3] W. Wolf, "What is embedded computing?," *Computer*, vol. 35, no. 1, pp. 136–137, 2002.

[4] A. P. Chandrakasan, S. Sheng, and R. W. Brodersen, "Low-power CMOS digital design," *IEICE Trans. Electron.*, vol. 75, no. 4, pp. 371–382, 1992.

[5] S. Mittal, "A survey of techniques for improving energy efficiency in embedded computing systems," *Int. J. Comput. Aided Eng. Technol.*, vol. 6, no. 4, pp. 440–459, 2014.

[6] K. Keutzer, A. R. Newton, J. M. Rabaey, and A. Sangiovanni-Vincentelli, "System-level design: Orthogonalization of concerns and platform-based design," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 19, no. 12, pp. 1523–1543, 2000.

[7] J. P. Erickson and J. H. Anderson, "Soft real-time scheduling," in *Handbook of Real-Time Computing*, Singapore: Springer Nature Singapore, 2022, pp. 233–267.

[8] V. Venkatachalam and M. Franz, "Power reduction techniques for microprocessor systems," *ACM Comput. Surv.*, vol. 37, no. 3, pp. 195–237, 2005.

[9] S. Murali, M. Coenen, A. Radulescu, K. Goossens, and G. De Micheli, "A methodology for mapping multiple use-cases onto networks on chips," in *Proc. Design Autom. Test Eur. Conf.*, vol. 1, Mar. 2006, pp. 1–6.

[10] J. Teich, "Hardware/software co-design: The past, the present, and predicting the future," *Proc. IEEE*, vol. 100, Special Centennial Issue, pp. 1411–1430, 2012.

[11] J. Fowers, K. Ovtcharov, M. Papamichael, T. Massengill, M. Liu, D. Lo, *et al.*, "A configurable cloud-scale DNN processor for real-time AI," in *Proc. 45th Annu. Int. Symp. Comput. Archit. (ISCA)*, Jun. 2018, pp. 1–14.

[12] V. Kathail, "Xilinx Vitis unified software platform," in *Proc. ACM/SIGDA Int. Symp. Field-Programmable Gate Arrays*, Feb. 2020, pp. 173–174.

[13] L. Andrade and F. Rousseau, *Multi-Processor System-on-Chip: Vol. 1 – Architectures*. 2022.

[14] K. Bertels, *Hardware/Software Co-Design for Heterogeneous Multi-Core Platforms*. Berlin/Heidelberg, Germany: Springer, 2012.

[15] C. Zhang, P. Li, G. Sun, Y. Guan, B. Xiao, and J. Cong, "Optimizing FPGA-based accelerator design for deep convolutional neural networks," in *Proc. ACM/SIGDA Int. Symp. Field-Programmable Gate Arrays*, Feb. 2015, pp. 161–170.

[16] K. Asanović and D. A. Patterson, "Instruction sets should be free: The case for RISC-V," EECS Dept., Univ. California, Berkeley, Tech. Rep. UCB/EECS-2014-146, 2014.

[17] H. Kopetz and W. Steiner, "Real-Time Systems: Design Principles for Distributed Embedded Applications". Springer Nature, 2022.

[18] P. Marwedel, "Embedded System Design: Embedded Systems Foundations of Cyber-Physical Systems and the Internet of Things". Springer Nature, 2021.

[19] J. C. Palencia and M. G. Harbour, "Schedulability analysis for tasks with static and dynamic offsets," in *Proc. IEEE Real-Time Syst. Symp.*, Dec. 1998, pp. 26–37.

[20] Y. Song, R. Xu, C. Wang, and Z. Li, "Improving data locality by array contraction," *IEEE Trans. Comput.*, vol. 53, no. 9, pp. 1073–1084, 2004.

[21] A. Das, A. Kumar, B. Veeravalli, R. Shafik, G. Merrett, and B. Al-Hashimi, "Workload uncertainty characterization and adaptive frequency scaling for energy minimization of embedded systems," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Mar. 2015, pp. 43–48.

[22] V. Dunjko and H. J. Briegel, "Machine learning & artificial intelligence in the quantum domain: A review of recent progress," *Rep. Prog. Phys.*, vol. 81, no. 7, 074001, 2018.

[23] A. Makarov, V. Sverdlov, and S. Selberherr, "Emerging memory technologies: Trends, challenges, and modeling methods,"

Microelectron. Rel., vol. 52, no. 4, pp. 628–634, 2012.

[24] R. Wilhelm, "Schloss Dagstuhl Support Grant for Junior Researchers," *NSF Award Number 1257011*, Directorate for Computer and Information Science and Engineering, vol. 12, no. 1257011, p. 57011, 2013.

[25] D. Kılıçarslan, G. Gürler, Ö. Özkasap, and A. M. Tekalp, "Energy efficient video decoding on multi-core devices," in *Proc. ACM Int. Conf. Energy-Efficient Comput. Netw. (e-Energy)*, 2011.

[26] V. Tiwari, S. Malik, and A. Wolfe, "Power analysis of embedded software: A first step towards software power minimization," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 2, no. 4, pp. 437–445, 2002.

[27] K. Sreenivasan and A. J. Kleinman, "On the construction of a representative synthetic workload," *Commun. ACM*, vol. 17, no. 3, pp. 127–133, 1974.

[28] R. Wilhelm, J. Engblom, A. Ermedahl, N. Holsti, S. Thesing, D. Whalley, *et al.*, "The worst-case execution-time problem—Overview of methods and survey of tools," *ACM Trans. Embedded Comput. Syst. (TECS)*, vol. 7, no. 3, pp. 1–53, 2008.

[29] F. Balarin, P. Giusto, A. Jurecska, C. Passerone, E. Sentovich, B. Tabbara, *et al.*, *Hardware-Software Co-Design of Embedded Systems: The POLIS Approach*, vol. 404. Springer, 2012.

[30] M. L. Brodie, "The promise of distributed computing and the challenges of legacy information systems," in *Interoperable Database Systems (DS-5)*, North-Holland, 1993, pp. 1–31.

[31] S. Srikantaiah, M. Kandemir, and M. J. Irwin, "Adaptive set pinning: Managing shared caches in chip multiprocessors," *ACM SIGPLAN Notices*, vol. 43, no. 3, pp. 135–144, 2008.

[32] J. Haj-Yahya, M. Alser, J. S. Kim, L. Orosa, E. Rotem, A. Mendelson, *et al.*, "FlexWatts: A power- and workload-aware hybrid power delivery network for energy-efficient microprocessors," in *Proc. 53rd Annu. IEEE/ACM Int. Symp. Microarchitecture (MICRO)*, Oct. 2020, pp. 1051–1066.

[33] M. Naeem, G. De Pietro, and A. Coronato, "Application of reinforcement learning and deep learning in multiple-input and multiple-output (MIMO) systems," *Sensors*, vol. 22, no. 1, p. 309, 2021.

[34] N. K. Govindaraju, B. Lloyd, Y. Dotsenko, B. Smith, and J. Manferdelli, "High performance discrete Fourier transforms on graphics processors," in *Proc. ACM/IEEE Conf. Supercomputing (SC'08)*, Nov. 2008, pp. 1–12.

[35] G. C. Buttazzo, G. Lipari, M. Caccamo, and L. Abeni, "Elastic scheduling for flexible workload management," *IEEE Trans. Comput.*, vol. 51, no. 3, pp. 289–302, 2002.

[36] R. Nane, V. M. Sima, C. Pilato, J. Choi, B. Fort, A. Canis, *et al.*, "A survey and evaluation of FPGA high-level synthesis tools," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 35, no. 10, pp. 1591–1604, 2015.

[37] X. Cao, L. Liu, Y. Cheng, and X. Shen, "Towards energy-efficient wireless networking in the big data era: A survey," *IEEE Commun. Surv. Tutor.*, vol. 20, no. 1, pp. 303–332, 2017.

[38] H. Mao, M. Alizadeh, I. Menache, and S. Kandula, "Resource management with deep reinforcement learning," in *Proc. 15th ACM Workshop Hot Topics Netw.*, Nov. 2016, pp. 50–56.

[39] D. R. Kuhn, D. R. Wallace, and A. M. Gallo, "Software fault interactions and implications for software testing," *IEEE Trans. Softw. Eng.*, vol. 30, no. 6, pp. 418–421, 2004.

[40] M. Gries and K. Keutzer, *Building ASIPS: The MESCAL Methodology*. Springer, 2005.

[41] K. Karuri and R. Leupers, *Application Analysis Tools for ASIP Design: Application Profiling and Instruction-Set Customization*. Springer, 2011.

[42] W. Meeus, K. Van Beeck, T. Goedemé, J. Meel, and D. Stroobandt, "An overview of today's high-level synthesis tools," *Design Autom. Embedded Syst.*, vol. 16, pp. 31–51, 2012.

[43] T. Chen, T. Moreau, Z. Jiang, L. Zheng, E. Yan, H. Shen, *et al.*, "TVM: An automated end-to-end optimizing compiler for deep learning," in *Proc. 13th USENIX Symp. Operating Systems Design Implementation (OSDI)*, 2018, pp. 578–594.

[44] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE Internet Things J.*, vol. 3, no. 5, pp. 637–646, 2016.

[45] G. Heiser and B. Leslie, "The OKL4 Microvisor: Convergence point of microkernels and hypervisors," in *Proc. 1st ACM Asia-Pacific Workshop Syst.*, Aug. 2010, pp. 19–24.

[46] B. Cheng, G. Solmaz, F. Cirillo, E. Kovacs, K. Terasawa, and A. Kitazawa, "FogFlow: Easy programming of IoT services over cloud and edges

for smart cities," *IEEE Internet Things J.*, vol. 5, no. 2, pp. 696–707, 2017.

[47] K. S. Chatha, V. K. Prasanna, and S. Vrudhula, "Adaptive algorithms for task partitioning and scheduling for reconfigurable computing systems," *IEEE Trans. VLSI Syst.*, vol. 9, no. 6, pp. 1069–1080, 2001.

[48] S. Boschert and R. Rosen, "Digital twin—The simulation aspect," in *Mechatronic Futures: Challenges and Solutions for Mechatronic Systems and Their Designers*, pp. 59–74, 2016.

[49] J. Dean, D. Patterson, and C. Young, "A new golden age in computer architecture: Empowering the machine-learning revolution," *IEEE Micro*, vol. 38, no. 2, pp. 21–29, 2018.

[50] D. Park, Y. Xiao, and A. DeHon, "Fast and flexible FPGA development using hierarchical partial reconfiguration," in *Proc. Int. Conf. Field-Programmable Technol. (ICFPT)*, Dec. 2022, pp. 1–10.

[51] Q. Duan, "Intelligent and autonomous management in cloud-native future networks—A survey on related standards from an architectural perspective," *Future Internet*, vol. 13, no. 2, p. 42, 2021.

[52] M. Sabt, M. Achemlal, and A. Bouabdallah, "Trusted execution environment: What it is, and what it is not," in *Proc. IEEE Trustcom/BigDataSE/Ispa*, Aug. 2015, vol. 1, pp. 57–64.

[53] G. Indiveri and S. C. Liu, "Memory and information processing in neuromorphic systems," *Proc. IEEE*, vol. 103, no. 8, pp. 1379–1397, 2015.

[54] W. W. Diab, A. Ferraro, B. Klenz, S. W. Lin, E. Liongosari, W. E. Tannous, and B. Zarkout, "Industrial IoT artificial intelligence framework," Feb. 2022.

[55] S. P. Mohanty, U. Choppali, and E. Kougianos, "Everything you wanted to know about smart cities: The Internet of things is the backbone," *IEEE Consum. Electron. Mag.*, vol. 5, no. 3, pp. 60–70, 2016.