Chapter 6

Reflecting on Self-Aware Systems-on-Chip

Bryan Donyanavard, Tiago Mück, Kasra Moazzemi, Biswadip Maity, Caio Batista de Melo, Kenneth Stewart, Saehanseul Yi, Amir M. Rahmani, and Nikil Dutt

6.1 Introduction to Self-Aware Systems-on-Chip

We are seeing an increasing number of complex cyber-physical systems (CPS) deployed for various applications, such as road-traffic control involving communicating autonomous cars and infrastructure, or smart grids controlling energy delivery down to the individual device. These distributed applications follow common design objectives, such as energy-efficiency, and require guarantees for high availability, real time or safety. In this context, autonomy is crucial: multiple system goals varying over time need to be adaptively managed and objectives holistically coordinated. By empowering future CPS with self-awareness, these systems promise to dynamically adapt, learn, and manage unforeseen changes [6].

6.1.1 Computational Self-Awareness

Computational self-awareness is the ability of a computing system to recognize its own state, possible actions, and the result of these actions on itself, its operational goals, and its environment, thereby empowering the system to become autonomous [6]. Computational self-awareness in itself is not a new field, but

B. Donyanavard (✉) · T. Mück · K. Moazzemi · B. Maity · C. B. de Melo · K. Stewart · S. Yi · N. Dutt
University of California, Irvine, CA, USA
e-mail: bdonyana@uci.edu; tmuck@uci.edu; moazzemi@uci.edu; maityb@uci.edu; cbatista@uci.edu; kennetms@uci.edu; saehansy@uci.edu; dutt@uci.edu

A. M. Rahmani
Technische Universität Wien, Vienna, Austria
e-mail: amirr1@uci.edu

rather a unification of subjects studied disjointly in various fields including control systems, artificial intelligence, autonomic computing, software engineering, among others, and how such research can be applied toward building computer systems with varying degrees of self-awareness in order to accomplish a task [8].

6.1.2 *Cyber-Physical Systems-on-Chip*

Battery-powered devices are the most ubiquitous computers in the world. Users of battery-powered devices expect support for various high-performance applications running on same device, potentially at the same time. Applications range from interactive maps and navigation, to web browsers and email clients. In order to meet performance demands by users utilizing complex workloads, increasingly powerful hardware platforms are being deployed in battery-powered devices. Systems-on-chip (SoCs) can integrate hundreds of heterogeneous cores and uncore components on a single chip. Such systems are constrained by a limited amount of shared system resources (e.g., power, interconnects). Simultaneously, the systems are expected to support workloads with diverse characteristics and demands that may conflict with system constraints. These platforms include a number of configurable knobs throughout the system stack and with different scope that allow for a trade-off between power and performance, e.g., dynamic voltage and frequency scaling (DVFS), power gating, idle cycle injection. These knobs can be set and modified at runtime based on the workload demands and system constraints. Heterogeneous many-core processors (HMPs) have extended this principle of dynamic power-performance trade-offs by incorporating single-ISA, architecturally differentiated cores on a single processor, with each of the cores containing a number of independent trade-off knobs. All of these configurable knobs allow for a huge range of potential trade-off. With such a large number of possible configurations, SoCs require intelligent runtime management in order to achieve system goals for complex workloads. Additionally, the knobs may be interdependent, so the decisions must be coordinated.

Cyber-physical systems-on-chip (CPSoC) [21] provide an infrastructure for system introspection and reflective behavior, which is the foundation for computational self-awareness. Figure 6.1 shows the infrastructure of a sensor-actuator rich platform, integrated with decision-making entities that observe system state through virtual and physical sensors at various layers in order to set the system configuration through actuators. The actuations are determined by policies that enforce the overall application goals while considering system constraints. Such an infrastructure can deploy *reactive* policies through the traditional *Observe, Decide, and Act* (ODA) feedback loop, as well as *proactive* policies through the augmented *self-aware* feedback loop. Figure 6.2 shows how the traditional ODA loop is augmented with reflection to provide self-aware adaptation. In this chapter we explore the

Fig. 6.1 CPSoC infrastructure: sensors and actuators throughout the system stack, with support for adaptive policies that enforce a given goal (from [3])

Fig. 6.2 Self-aware feedback loop. Policies are deployed to make action decisions toward achieving a goal by controlling the CPSoC based on observations and self-aware adaptation

use of computational self-awareness to address challenges of adaptive resource management in cyber-physical systems-on-chip.¹

¹Throughout the remainder of this chapter we use SoC as an umbrella term that includes CPSoC.

6.2 Reflective System Models

Traditionally, resource managers deploy an ODA feedback loop (lower half (in black) of Fig. 6.3) to manage systems at runtime. However, recent works [1, 27] have shown that a runtime model of the system can better manage the unpredictable nature of workloads.

Reflection can be defined as *the capability of a system to reason about itself and act upon this information* [26]. A reflective system can achieve this by maintaining a representation of itself (i.e., a self-model) within the underlying system, which is used for reasoning. Reflection is a key property of self-awareness. Reflection enables decisions to be made based on both *past* observations, as well as *predictions* made from past observations. Reflection and prediction involve two types of models: (1) a self-model of the subsystem(s) under control, and (2) models of other policies that may impact the decision-making process. Predictions consider *future* actions, or events that may occur before the next decision, enabling “what-if” exploration of alternatives. Such actions may be triggered by other policies invoked more frequently than the decision loop. The top half of Fig. 6.3 (in blue) shows prediction enabled through reflection that can be utilized in the decision-making process of a feedback loop. The main goal of the predictive model is to estimate system behavior based on potential actuation decisions as well as system dynamics.

6.2.1 Middleware for Reflective Decision-Making

The increasing heterogeneity in a platform’s resource types and the interactions between resources pose challenges for coordinated model-based decision-making in the face of dynamic workloads. Self-awareness properties address these challenges for emerging SoC platforms through reflective resource managers. Reflective resource managers build a model of the system which represents the software organization or the architecture of the target platform. Resource managers can use reflective models to anticipate the effects of changing the system configuration

Fig. 6.3 Feedback loop overview. The bottom part of the figure represents a simple observe–decide–act loop. The top part (in blue) adds the reflection mechanism to this loop, enabling predictions for smart decision-making

Fig. 6.4 MARS framework overview from [10]. Different layers of the system stack coordinate through policies to orchestrate the management of resources: sensors inform policies of the system state; policies coordinate with models to perform reflective queries, and make resource management decisions; policies set actuators to enact changes on the system

at runtime. However, with SoC computing platform architectures evolving rapidly, porting the self-aware decision logic across different hardware platforms is challenging, requiring resource managers to update their models and platform-specific interfaces. To address this problem, we propose MARS (Middleware for Adaptive and Reflective Systems), a cross-layer and multi-platform framework that allows users to easily create resource managers by composing system models and resource management policies in a flexible and coordinated manner.

Figure 6.4 shows an overview of the MARS framework (shaded), with *Sensors* and *Actuators* interfacing across multiple layers of the system stack: *Applications*, *Linux kernel*, and *HW Platform*. The components of MARS are explained next.

1. **Sensors and actuators:** The sensed data consists of performance counters (e.g., instructions executed, cache misses, etc.) and other sensory information (e.g., power, temperature, etc.). The collected data is used to assess the current system state and to characterize workloads. Any updates to the system configuration (e.g., CPU core frequency, GPU frequency, memory controller frequency, task-to-core mapping) happen through system knobs. Actuators allow system configuration changes to optimize operating point or control trade-offs.
2. **Resource Management Policies:** They are platform agnostic user-level daemons implemented in MARS using supported sensors, actuators, and reflective system models.
3. **Reflective system model** is used by the policies to make informed decisions. The reflective model has the following subcomponents:

- (a) Models of *policies implemented by the underlying OS kernel* used for coordinating decisions made within MARS with decisions made by the OS.
 - (b) Models of *user policies* that are automatically instantiated from any policy defined within MARS.
 - (c) The baseline *performance/power model*. This model takes as input the predicted actuations generated from the policy models and produces predicted sensed data.
4. **The policy manager** is responsible for reconfiguring the system by adding, removing, or swapping policies to better achieve the current system goal.

MARS is implemented in the C++ language following an object-oriented paradigm and works on hardware (e.g., Odroid-XU3, Nvidia Jetson TX2), simulated (e.g., gem5), and trace-based offline [11] platforms. The framework is open source and available online.² While the current version of MARS targets energy-efficient heterogeneous SoCs, we believe the MARS framework can be ported to a wider range of systems (e.g., web servers, high-performance clusters) to support self-aware resource management.

6.3 Managing Energy-Efficient Chip Multiprocessors

Dynamic resource management for HMPs is a well-known challenge: integration of hundreds of cores running various workloads with conflicting constraints increases the pressure on limited shared system resources. A promising and well-established approach is the use of control-theoretic solutions based on rigorous mathematical formalisms that can provide bounds and guarantees for system resource management. In this context, we discuss efforts that deploy control-theoretic-centric runtime resource management of HMPs, from simple Single Input Single Output (SISO) controllers to more complex Supervisory Control Theory (SCT) methods.

6.3.1 Single Input Single Output Controllers

Conventional control theory methods proposed for resource management use Single Input Single Output (SISO) controllers for the ease in deployment and the guarantees they provide in tracking the target output. These SISO controllers use Proportional Integral (PI), Proportional Integral Derivative (PID), or lead-lag implementations [22]. Figure 6.5 depicts a first-order feedback SISO controller which can be deployed either as a PI or a PID controller. The error e is the input to the controller. Note that to compute the current control input u , the controller

²Code repository at <https://github.com/duttresearchgroup/MARS>.

Fig. 6.5 *Single Input Single Output (SISO) feedback loop*

needs to have the current value of the error e along with the past value of the error and the past value of the control input. It is this memory inherent in the controller that makes it dynamic.

6.3.2 Multiple Input Multiple Output Controllers

Modern HMPs execute diverse set of workloads with varying resource demands, which sometimes exhibit conflicting constraints. In this context, the use of SISO controllers might not be effective as multiple system goals varying over time need to be managed in a coordinated and holistic manner. Multiple Input Multiple Output (MIMO) control theory is able to coordinate and prioritize multiple design goals and actions. MIMO controllers have proven effective for coordinating management of multiple goals in uncore processors [17] and HMPs [12].

6.3.3 Adaptive Control Methods

Ideally, control-theoretic solutions should provide formal guarantees, be simple enough for runtime implementation, and handle nonlinear system behavior. Static linear feedback controllers such as SISO and MIMO can provide robustness and stability guarantees with simple implementations, while adaptive controllers modify the control law at runtime to adapt to the discrepancies between the expected and the actual system behavior. However, modifying the controller at runtime is a costly operation that also invalidates the formal guarantees provided at design time. In order to be able to take predicted responsive actions against nonlinear behavior of the computer systems, a well-established and lightweight adaptive control-theoretic

technique called *Gain Scheduling* can be used. This method is used for dynamic power management in chip multiprocessors in [2].

6.3.4 Hierarchical Controllers

Supervisory Control Theory (SCT) [19, 30] provides formal and systematic supervision of classical MIMO/SISO controllers. SCT uses modular decomposition of control problems to manage their complexity. Specifically, supervisory control has two key properties: (1) rapid adaptation in response to abrupt changes in management policy and (2) low computational complexity by computing control parameters for different policies offline. New policies and their corresponding parameters can be added to the supervisor on demand. Therefore, SCT is suitable for resource management problems (such as managing power, energy, and quality-of-service metrics) that can be modeled using logic and discrete system dynamics.

Figure 6.6 depicts a high-level view of supervisory control for HMP resource management. Either the user or the system software may specify *Variable Goals and Policies*. The *Supervisory Controller* aims to meet system goals by managing the low-level controllers. High-level decisions are made based on the feedback given by the *High-level Plant Model*, which provides an abstraction of the entire system. Various types of *Classic Controllers*, such as PID or state-space controllers, can be used to implement each low-level controller based on the target of each subsystem. The flexibility to incorporate any pre-verified off-the-shelf controllers without the need for system-wide verification is essential for the modularity of this approach. The supervisor provides parameters such as output references or gain values to each low-level controller during runtime according to the system policy. Low-level controller subsystems update the high-level model to maintain global system state,

Fig. 6.6 High-level view of Supervisory Control Theory

and potentially trigger the supervisory controller to take action. The high-level model can be designed in various fashions (e.g., rule-based or estimator-based) to track the system state and provide the supervisor with guidelines. Supervisory control provides the opportunity to benefit from both classical control-theoretic methods and heuristics in a robust fashion. The SCT hierarchy in Fig. 6.6 is successfully used to manage quality-of-service (QoS) goals within a power budget on an HMP in [18].

6.4 Heterogeneous Mobile Governors: Energy-Efficient Mobile System-on-a-Chip

Mobile games stress modern SoCs by utilizing heterogeneous processing elements, CPUs and GPUs, concurrently. However, the utilization of each processing element may vary between games. Performance of these games that usually is measured in frames per second (FPS) can highly depend on the operating frequency of compute units. However, conventional DVFS governors conservatively choose high frequencies without considering the utilization pattern of the games [16]. In order to meet a performance goal while conserving energy, the frequency of each processing element should be as low as possible without an observable effect on the FPS.

6.4.1 *Sensors to Capture Dynamism*

To coordinate frequency configuration decisions, a cooperative CPU-GPU DVFS strategy, Co-Cap [14], limits the maximum frequency of CPUs and GPUs on a game-specific basis. Based on the utilization of each processing element, games are classified as one of the following classes: (1) No CPU-GPU Dominant; (2) CPU Dominant; (3) GPU Dominant; and (4) CPU-GPU Dominant. Figure 6.7 shows the classes and gives an example of each class. To determine a maximum frequency for each game class, Co-Cap implements a frame rate sensor, which is affected by both CPU and GPU frequencies. By limiting maximum frequencies for each game class, Co-Cap reduces energy consumption without observable performance degradation.

The assumption in Co-Cap is that games can only belong to one of the classes. However, some games might change their dynamic behavior throughout their life cycle. To proactively respond to the dynamic CPU and GPU frequency requirements of games, a DVFS governor policy requires more information about a game's workload dynamism. A Hierarchical Finite State Machine (HFSM) based CPU-GPU governor, HiCAP [13], models the dynamic behavior of mobile gaming workloads and applies a cooperative, dynamic CPU-GPU frequency-capping policy to conserve energy by adapting to a game's inherent dynamism. Using the HFSM, a DVFS governor can predict the next workload feature for a certain window

GPU workload \ CPU workload	GPU workload				
	low	...	med	...	high
CPU workload	low	Angry Birds		GFX bench	
	...	NO CPU-GPU Dominant		GPU Dominant	
	med	CPU Dominant		CPU-GPU Dominant	
...	...			GPU bench	
	high	Jetski Race			

Fig. 6.7 Classification of CPU-GPU workloads observed in mobile games from [14]

at a game’s runtime. Through this added self-awareness, HiCAP reduces energy consumption even further than Co-Cap.

Further dynamism exists in a game’s memory access patterns. Some scenes in mobile games read more graphics data than others, resulting in increased memory utilization. This may slow down the CPU portion of the game, but on the other hand when memory utilization is low, it may run faster than originally predicted by a conventional DVFS governor. A conventional DVFS governor cannot detect these memory utilization changes by sensing utilization, causing prediction errors to increase. MEMCOP, a Memory-aware Cooperative Power Management Governor for Mobile games [5], senses the number of last level cache misses to monitor the memory pressure of the system in addition to CPU, and GPU memory utilization. This prevents the CPU DVFS governor from increasing frequency due to inaccurate predictions caused by variation in memory access time.

6.4.2 Toward Self-Aware Governors

Co-Cap, HiCap, and MEMCOP DVFS policies are each steps toward a self-aware DVFS governor policy for heterogeneous SoCs. Each policy monitors system’s state using novel sensors, and defines runtime prediction rules to reflect and adapt to changes in mobile game behavior. However, the predictive models are generated statistically at design time, and remain the same during the execution. Moreover, as the predictive model becomes more complex, prediction errors increase due to the assumption of a linear relationship between the model’s input and output. ML-Gov, a machine learning enhanced integrated CPU-GPU governor [15], tries to address these issues by applying machine learning algorithms. This method does not require rule tuning at design-time. ML-Gov’s machine learning algorithm helps to exploit

nonlinear characteristics between frequency and performance. ML-Gov currently builds the model offline, but through enhanced self-awareness via online updates of the reflective model, could adapt to previously unknown games and classes.

6.5 Adaptive Memory: Managing Runtime Variability

Heterogeneous processing elements on mobile SoCs share limited memory resources, leading to memory contention and stalled processes waiting for data. This performance degradation is exacerbated by the Von Neumann bottleneck, a prevalent problem in modern day computer systems. Data transfer speeds in memory have not been able to keep up with the performance gains of processors exemplified by Moore's law. However, with the end of Moore's law on the horizon there is an ever increasing need to alleviate the Von Neumann bottleneck to increase the performance of computer systems. There have been various approaches over the years to address the Von Neumann bottleneck such as putting critical memory in an easily accessible cache [25] and recently in an easily accessible Software-Programmable Memory (SPM), also known as a scratchpad, using multi-threading [9], and exploiting cache-coherency [7]. We address the bottleneck by providing self-awareness with respect to memory resource utilization.

6.5.1 *Sharing Distributed Memory Space*

Software-Programmable Memories are a promising alternative to hardware-managed caches in embedded systems. However, traditional approaches for managing SPMs do not support sharing of distributed memory resources, missing the opportunity to utilize those memory resources. Employing operating-system-level awareness of SPM utilization, memory resources can be shared by allowing threads to opportunistically exploit the entire memory space for unpredictable application workloads. Best-effort policies can be used to maximize the usage of on-chip SPMs. The policies can be supported by hardware via distributed memory management units (MMUs), an on-chip component that can be used to exchange information between the NoC and an MMU's local SPM. Sharing distributed SPM space reduces memory contention, resulting in reduced memory latency by reducing off-chip memory accesses by about 14%. The off-chip access reduction decreases average execution time by about 19.5%, which in turn reduces energy consumption [23, 29]. More intelligent policies that explore a mixed SPM/cache hierarchy for many-core embedded systems can yield further improvements.

6.5.2 Memory Phase Awareness

Modern mobile devices use multi-core platforms that allow for concurrent execution of multimedia and non-multimedia applications that enter and exit at unpredictable times. Each application also has variable memory demands during these unpredictable times. By being aware of the periodic patterns, or *phasic behavior*, of an application’s memory usage (memory phases), a system’s on-chip memory can be more efficiently utilized. Memory phases can be identified from memory usage information extracted on an application basis, and can be used to prioritize different memory pages in a multi-core platform without having any prior knowledge about running applications. The identification process can be integrated into the runtime system and done online. For example, memory phases can be used for effective sharing of distributed SPMs for multi-core platforms to reduce memory access latency and contention. Experiments on workloads with varying intra- and inter-application memory-intensity show that using phase detection schemes can reduce memory access latency up to 45% for configurations up to 16 cores [28]. Ongoing work investigates more aggressive use of memory phasic behavior in many-core architectures with hundreds of cores.

6.5.3 Quality-Configurable Memory

We have established how self-awareness can be achieved through formal control theory. Figure 6.8 shows a closed feedback control loop with a quality monitor that can measure memory utilization and processor usage with respect to a QoS goal to fit the runtime requirements of applications. The quality monitor gives a quality score and sends the collected data to a high-level controller. The controller reflects on the data, then tunes knobs to adapt the memory utilization and processor usage to minimize the error between the current quality and the quality-of-service goal. The self-aware approach enables dynamic convergence toward dynamic memory utilization and quality targets for unpredictable workloads. While current

Fig. 6.8 Control loop implementing a self-aware approach to memory utilization optimization. The controller optimizes memory knobs to improve application performance

results indicate that a self-aware memory controller outperforms a manual quality configuration scheme, there is much work to be done with to analyze energy trade-offs when using a self-aware memory controller, and whether a MIMO controller could be more effective for resource management in many-core systems with the self-aware approach [24].

6.6 What's Ahead?

Self-awareness enables a system to observe its context and make changes to optimize its execution at runtime. For instance, it is possible to allow a system to tune its execution to optimize power consumption. Through observing how it has reacted to past changes in certain conditions, the system can learn what the impact on the overall execution and power consumption was, and if a different adaptation would be more appropriate in the future. To further explore such opportunities in computing systems, we shift our focus to a new project: the Information Processing Factory (IPF). IPF is a step toward autonomous many-core platforms in cyber-physical systems (CPS) and the Internet of Things (IoT). It represents a paradigm shift in platform design, with robust and independent platform operation in the focus of platform-centric design rather than existing semiconductor device or software technology, as mostly seen today [4].

We use the metaphor of an Information Processing Factory to draw similarities between microelectronics systems and factories as follows: in a factory, all components must adapt to the current workload [20]. Additionally, this adaptation cannot be done offline and must instead be done in real time without interrupting the baseline operations. Future microelectronic systems (e.g., MPSoCs) should operate in a similar manner.

Clusters of component-specific, uncorrelated control occurrences cannot handle operations of large scale systems with multi-criteria objective functions. Similarly, a centralized controller model is also inadequate in this case because it cannot scale. The goal of the IPF project is to demonstrate that a hybrid hierarchical approach, sporting as much modularity as possible and as much centralized as necessary, is a much more effective means of achieving the desired goal while maintaining cost efficiency, low overhead, and scalability.

Figure 6.9 depicts how we envision the platform to be structured. Information provided by sensors is gathered and merged into self-organizing, self-aware (SO/SA) control processing instances across different hardware/software abstraction layers comprising an MPSoC-based CPS system. The SO/SA instances generate actuation directives affecting the MPSoC system components at same or lower levels of abstraction. The SO/SA paradigm is not limited in scope to optimization of CPS operational parameters/metrics. In fact, self- and group-awareness can also enable higher level tasks such as self-protection of both the MPSoC and the overall CPS system.

Fig. 6.9 Idealized IPF Platform from [20]. Self-organizing and Self-aware (SO/SA) components are distributed throughout the system stack. Both hardware and application system state are monitored and configurations adjusted using sensors and actuators

6.6.1 Example Use Case: Autonomous Driving

The key innovation in automated driving as compared to driver assistance systems is the transition of decision-making from the driver to the vehicle. The application processing and communication requirements ask for platform performance, memory capacity, and communication bandwidth and latency far beyond the capabilities of current architectures. At the same time, these platforms must be highly reliable and guarantee sufficient functionality under platform errors, aging, and degradation to meet safety standards. That is, platforms and their components must be fail-operational, i.e., must be able to continue driving, instead of fail-safe, as today.

Thus, the automated driving requirements can be mapped to corresponding requirements of an Information Processing Factory. The system must be capable of in-field integration, i.e., able to adapt to changes in the workload of both critical and non-critical (best-effort) functions. The system must find a new suitable mapping and must prevent the changes from violating the guarantees of other software components. The software must be able to detect and to adapt to transient errors in order to provide a reliable service. This requires self- diagnosis and self-healing.

The system must be predictable and provide for minimum performance guarantees for all scenarios.

Allowing and exploiting dynamic system behavior through IPF can significantly improve platform performance and resource utilization. Thus, the system must be able to optimize the execution and mapping online: self-optimization. The optimization may target e.g. aging (temperature), power consumption, response time, and resource utilization.

6.7 Summary

Future cyber-physical systems will host a large number of coexisting distributed applications on hardware platforms with thousands to millions of networked components communicating over open networks. These distributed applications will include both critical and best-effort tasks, may be subject to permanent change, environment dynamics and application interference. Using wisdom gathered from our initial exploration into self-aware SoCs, we introduce a new Information Processing Factory paradigm to manage current and future cyber-physical systems.

Acknowledgments The authors would like to acknowledge Santanu Sarma, Chen-Ying Hsieh, JurnGyu Park, Majid Shoushtari, and Hossein Tajik for their research contributions. We acknowledge financial support by NSF grant CCF-1704859, and the Marie Curie Actions of the European Union's H2020 Program.

References

1. B. Donyanavard, T. Mück, S. Sarma, N. Dutt, Sparta: runtime task allocation for energy efficient heterogeneous many-cores, in *Proceedings of the Eleventh IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis, CODES '16* (ACM, New York, 2016), pp. 27:1–27:10. <https://doi.org/10.1145/2968456.2968459>
2. B. Donyanavard, A.M. Rahmani, T. Muck, K. Moazemmi, N. Dutt, Gain scheduled control for nonlinear power management in CMPs, in *2018 Design, Automation Test in Europe Conference Exhibition (DATE)* (2018)
3. N. Dutt, A. Jantsch, S. Sarma, Self-aware cyber-physical systems-on-chip, in *2015 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)* (2015), pp. 46–50. <https://doi.org/10.1109/ICCAD.2015.7372548>
4. N. Dutt, F.J. Kurdahi, R. Ernst, A. Herkersdorf, Conquering MPSoC complexity with principles of a self-aware information processing factory, in *Proceedings of the Eleventh IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis* (ACM, New York, 2016), p. 37
5. C.Y. Hsieh, J.G. Park, N. Dutt, S.S. Lim, MEMCOP: memory-aware co-operative power management governor for mobile games. *Des. Autom. Embed. Syst.* **22**(1-2), 95–116 (2018). <https://doi.org/10.1007/s10617-018-9201-8>
6. A. Jantsch, N. Dutt, A.M. Rahmani, Self-awareness in systems on chip— a survey. *IEEE Des. Test* **34**(6), 8–26 (2017). <https://doi.org/10.1109/MDAT.2017.2757143>

7. O. Kayiran, N.C. Nachiappan, A. Jog, R. Ausavarungnirun, M.T. Kandemir, G.H. Loh, O. Mutlu, C.R. Das, Managing GPU concurrency in heterogeneous architectures, in *2014 47th Annual IEEE/ACM International Symposium on Microarchitecture* (2014), pp. 114–126. <https://doi.org/10.1109/MICRO.2014.62>
8. S. Kounev, P. Lewis, K.L. Bellman, N. Bencomo, J. Camara, A. Diaconescu, L. Esterle, K. Geihs, H. Giese, S. Götz, P. Inverardi, J.O. Kephart, A. Zisman, *The Notion of Self-aware Computing* (Springer International Publishing, Cham, 2017)
9. C.-K. Luk, Tolerating memory latency through software-controlled pre-execution in simultaneous multithreading processors, in *Proceedings 28th Annual International Symposium on Computer Architecture* (2001), pp. 40–51. <https://doi.org/10.1109/ISCA.2001.937430>
10. T.R. Mück, Reflective on-chip resource management policies for energy-efficient heterogeneous multiprocessors. Ph.D. Thesis, University of California, Irvine (2018). <http://www.escholarship.org/uc/item/6n93v21h>
11. T. Mück, B. Donyanavard, N. Dutt, PoliCym: rapid prototyping of resource management policies for HMPs, in *Proceedings of the 28th International Symposium on Rapid System Prototyping: Shortening the Path from Specification to Prototype, RSP '17* (ACM, New York, 2017), pp. 23–29. <https://doi.org/10.1145/3130265.3130321>
12. T.R. Mück, B. Donyanavard, K. Moazzemi, A.M. Rahmani, A. Jantsch, N.D. Dutt, Design methodology for responsive and robust MIMO control of heterogeneous multicores. *IEEE Trans. Multi Scale Comput. Syst.* **4**, 944–951 (2018). <https://doi.org/10.1109/TMSCS.2018.2808524>
13. J.G. Park, N. Dutt, H. Kim, S.S. Lim, HiCAP: hierarchical FSM-based dynamic integrated CPU-GPU frequency capping governor for energy-efficient mobile gaming, in *ISLPED '16: Proceedings of the 2016 International Symposium on Low Power Electronics and Design* (2016), pp. 218–223. <https://doi.org/10.1145/2934583.2934588>
14. J.G. Park, C.Y. Hsieh, N. Dutt, S.S. Lim, Co-cap: energy-efficient cooperative CPU-GPU frequency capping for mobile games, in *SAC '16: Proceedings of the 31st Annual ACM Symposium on Applied Computing* (ACM, New York, 2016). <https://doi.org/10.1145/2851613.2851671>
15. J.G. Park, N. Dutt, S.S. Lim, ML-Gov: a machine learning enhanced integrated CPU-GPU dVFS governor for mobile gaming, in *ESTIMedia '17: Proceedings of the 15th IEEE/ACM Symposium on Embedded Systems for Real-Time Multimedia* (2017), pp. 12–21. <https://doi.org/10.1145/3139315.3139317>
16. J.G. Park, C.Y. Hsieh, N. Dutt, S.S. Lim, Synergistic CPU-GPU frequency capping for energy-efficient mobile games. *ACM Trans. Embed. Comput. Syst.* **17**(2), 1–24 (2017). <https://doi.org/10.1145/3145337>
17. R.P. Pothukuchi, A. Ansari, P. Voulgaris, J. Torrellas, Using multiple input, multiple output formal control to maximize resource efficiency in architectures, in *Proceedings of the 43rd International Symposium on Computer Architecture* (2016)
18. A.M. Rahmani, B. Donyanavard, T. Mück, K. Moazzemi, A. Jantsch, O. Mutlu, N. Dutt, Spectr: formal supervisory control and coordination for many-core systems resource management, in *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS '18* (ACM, New York, 2018), pp. 169–183. <https://doi.org/10.1145/3173162.3173199>
19. P.J. Ramadge, W.M. Wonham, The control of discrete event systems. *Proc. IEEE* **77**(1), 81–98 (1989)
20. A. Sadighi, B. Donyanavard, T. Kadeed, K. Moazzemi, T. Mück, A. Nassar, A.M. Rahmani, T. Wild, N. Dutt, R. Ernst, et al., Design methodologies for enabling self-awareness in autonomous systems, in *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)* (IEEE, Piscataway, 2018), pp. 1532–1537
21. S. Sarma, N. Dutt, P. Gupta, A. Nicolau, N. Venkatasubramanian, On-chip self-awareness using cyberphysical-systems-on-chip (CPSoC), in *Proceedings of the 2014 International Conference on Hardware/Software Codesign and System Synthesis, CODES '14* (ACM, New York, 2014), pp. 22:1–22:3. <https://doi.org/10.1145/2656075.2661648>

22. S. Shahosseini, K. Moazzemi, A.M. Rahmani, N. Dutt, Dependability evaluation of SISO control-theoretic power managers for processor architectures, in *2017 IEEE Nordic Circuits and Systems Conference (NORCAS): NORCHIP and International Symposium of System-on-Chip (SoC)* (2017)
23. M. Shoushtari, B. Donyanavard, L.A.D. Bathen, N. Dutt, Shave-ice: sharing distributed virtualized SPMs in many-core embedded systems. *ACM Trans. Embed. Comput. Syst.* **17**(2), 47:1–47:25 (2018). <https://doi.org/10.1145/3157667>
24. M. Shoushtari, A. Rahmani, N. Dutt, Quality-configurable approximate memory hierarchy: a formal control theory approach. Workshop on Approximate Computing Across the Stack (2018). <http://approximate.computer/wax2018/papers/wax2018-paper12.pdf>
25. A.J. Smith, Cache memories. *ACM Comput. Surv.* **14**(3), 473–530 (1982). <https://doi.org/10.1145/356887.356892>
26. B.C. Smith, *Reflection and Semantics in a Procedural Programming Language*. Ph.D. (MIT Press, Cambridge, 1982)
27. V. Spiliopoulos, S. Kaxiras, G. Keramidas, Green governors: a framework for continuously adaptive DVFS, in *Proceedings of the 2011 International Green Computing Conference and Workshops, IGCC '11* (IEEE Computer Society, Washington, 2011), pp. 1–8. <https://doi.org/10.1109/IGCC.2011.6008552>
28. H. Tajik, B. Donyanavard, N. Dutt, On detecting and using memory phases in multimedia systems, in *Proceedings of the 14th ACM/IEEE Symposium on Embedded Systems for Real-Time Multimedia, ESTIMedia'16* (ACM, New York, 2016), pp. 57–66. <https://doi.org/10.1145/2993452.2993566>
29. H. Tajik, B. Donyanavard, N. Dutt, J. Jahn, J. Henkel, Smpool: runtime SPM management for memory-intensive applications in embedded many-cores. *ACM Trans. Embed. Comput. Syst.* **16**(1), 25:1–25:27 (2016). <https://doi.org/10.1145/2968447>
30. J. Thistle, Supervisory control of discrete event systems. *Math. Comput. Model.* **23**(11), 25–53 (1996)

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (<http://creativecommons.org/licenses/by/4.0/>), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.