SPECTR: Formal Supervisory Control and Coordination for Many-core Systems Resource Management

Amir M. Rahmani[†‡*] Bryan Donyanavard[†*] Tiago Mück[†*] Kasra Moazzemi[†*] Axel Jantsch[‡]
Onur Mutlu[§] Nikil Dutt[†]
[†]University of California, Irvine, USA [‡]TU Wien, Austria [§]ETH Zurich, Switzerland

Abstract

Resource management strategies for many-core systems need to enable sharing of resources such as power, processing cores, and memory bandwidth while coordinating the priority and significance of system- and application-level objectives at runtime in a scalable and robust manner. State-of-the-art approaches use heuristics or machine learning for resource management, but unfortunately lack formalism in providing robustness against unexpected corner cases. While recent efforts deploy classical control-theoretic approaches with some guarantees and formalism, they lack scalability and autonomy to meet changing runtime goals.

We present SPECTR, a new resource management approach for many-core systems that leverages formal supervisory control theory (SCT) to combine the strengths of classical control theory with state-of-the-art heuristic approaches to efficiently meet changing runtime goals. SPECTR is a scalable and robust control architecture and a systematic design flow for hierarchical control of many-core systems. SPECTR leverages SCT techniques such as gain scheduling to allow autonomy for individual controllers. It facilitates automatic synthesis of the high-level supervisory controller and its property verification.

We implement SPECTR on an Exynos platform containing ARM's big.LITTLE-based heterogeneous multi-processor (HMP) and demonstrate that SPECTR's use of SCT is key to managing multiple interacting resources (e.g., chip power and processing cores) in the presence of competing objectives (e.g., satisfying QoS vs. power capping). The principles of SPECTR are easily applicable to any resource type and objective as long as the management problem can be modeled using dynamical systems theory (e.g., difference equations), discrete-event dynamic systems, or fuzzy dynamics.

ACM Reference Format:
Amir M. Rahmani[†‡*] Bryan Donyanavard[†*] Tiago Mück[†*] Kasra Moazzemi[†*] Axel Jantsch[‡] Onur Mutlu[§] Nikil Dutt[†] . 2018. SPECTR: Formal Supervisory Control and Coordination for Many-core Systems Resource Management. In *ASPLOS '18: Architectural Support for Programming Languages and Operating Systems, March 24–28, 2018, Williamsburg, VA, USA.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3173162.3173199

[*] These authors contributed equally to this work.

ASPLOS '18, March 24–28, 2018, Williamsburg, VA, USA

ACM ISBN 978-1-4503-4911-6/18/03…$15.00
https://doi.org/10.1145/3173162.3173199

1 Introduction

Runtime resource management for many-core systems is increasingly challenging due to the complex interaction of: i) integrating hundreds of (heterogeneous) cores and uncore components on a single chip, ii) limited amount of system resources (e.g., power, cores, interconnects), iii) diverse workload characteristics with conflicting constraints and demands, and iv) increasing pressure on shared system resources from data-intensive workloads.

In this context, autonomy is crucial: multiple system goals varying over time need to be adaptively managed and objectives holistically coordinated. As a result, designers face a large space of configuration parameters that often are controlled by a limited number of actuation knobs, which in turn generate a very large number of cross-layer actuation configurations. For instance, Zhang and Hoffman [93] show that for an 8-core Intel Xeon processor, combining only a handful of actuation knobs (such as clock frequency and Hyperthreading levels) generates over 1000 different actuation configurations; they use binary search to efficiently explore the configuration space for achieving a *single* goal: cap the Thermal Design Power (TDP) while maximizing performance. Searching the configuration space is common practice in many similar single-goal, heuristic-based, runtime resource management approaches [10, 11, 69, 82, 86]. While there is a large body of literature on ad-hoc resource management approaches for processors using heuristics and thresholds [15, 17, 24, 46], rules [18, 44], solvers [34, 65], and predictive models [7, 19–21], there is a lack of formalism in providing guarantees for resource management of complex many-core systems. We find that prior art focuses primarily on performance, reliability, and adaptivity (learning from feedback), with relatively little research on the following critical questions:

1. **Robustness:** How can we provide guarantees and perform robustness analysis?
2. **Formalism:** What formalisms facilitate reasoning about and synthesis of resource management strategies?
3. **Efficiency:** How can we design lightweight, yet responsive controllers?
4. **Coordination:** How do we control and coordinate (possibly conflicting) actuations while tracking multiple objectives simultaneously (e.g., frame rate and chip power)?
5. **Scalability:** How can we properly design control hierarchies to manage large and complex systems?
6. **Autonomy:** How can controllers automatically respond to abrupt runtime changes in objectives (e.g., changing the priority of objectives)?

Table 1 shows the coverage of existing on-chip resource management approaches in handling these key issues. Some machine learning based and heuristic approaches (e.g., [7, 21, 24, 32] in Rows A and B) focus on **efficiency (3)** and **coordination (4)**, but fail to address other attributes such as formalism in providing robustness

Table 1. Major on-chip resource management approaches and the key questions they address (∗ = partially addressed)

	Methods	1. Robustness	2. Formalism	3. Efficiency	4. Coordination	5. Scalability	6. Autonomy
A	Machine learning [7, 21, 32]		✓	✓	✓		
B	Estimation/Model based heuristics [15, 17, 19, 24, 46]			✓	✓		
C	SISO Control Theory [40, 55, 56, 70, 71]	✓	✓	✓		∗	
D	MIMO Control Theory [66, 67]	✓	✓	✓	✓		
E	Supervisory Control Theory [SPECTR]	✓	✓	✓	✓	✓	✓

against unexpected corner cases. Single-Input-Single-Output (SISO) control theoretic approaches (e.g., [56] in Row C) provide means to address **robustness (1)**, **formalism (2)**, and **efficiency (3)**, while lacking the ability to concurrently coordinate and control multiple objectives in a non-conflicting manner. Although multiple SISOs have been used in nested loops to achieve scalability in simple control problems [40, 55], they suffer from scalability issues in complex resource management problems for many-core systems where coordination of multiple actuators is necessary. Recently-proposed Multiple-Input-Multiple-Output (MIMO) control [66, 67] (Row D) enables **coordination (4)**, addressing attributes (1) to (4). For example, such control is able to simultaneously and robustly track (i.e., meet) the reference (i.e., target) power consumption and instructions per second (IPS) on a *single-core* processor. However, MIMO control lacks **scalability (5)** for heterogeneous multi-processing (HMP) architectures due to 1) the exponential growth in computational complexity with increasing numbers of inputs and outputs, and 2) the difficulty of performing Dynamic System Model identification for large systems. Furthermore, both SISO and MIMO controllers lack **autonomy (6)**, which enables rapid responses to abrupt runtime changes: while SISO is a *single objective* controller, MIMO deploys a design-time-configured Tracking Error Cost matrix that captures the priority of tracking each output; and both are unable to adapt to runtime changes. Autonomous operation of such classic controllers requires an external entity to coordinate their set-points and schedule their gains dynamically.[1]

Our goal is to address all six key challenges in HMP resource management. To this end, we propose SPECTR, a **scalable (5)** and **autonomous (6)** approach based on Supervisory Control Theory (SCT) [73, 84]. SPECTR provides formal and systematic supervision of classical MIMO/SISO controllers, thereby holistically addressing all six key attributes (Row E in Table 1). SCT uses modular decomposition of control problems to manage their complexity. SCT is widely used for higher-level control of complex systems such as communication and transportation networks, computer databases, and manufacturing systems in order to achieve higher performance and predictable operation [73]. SCT's application scope is wider than classical control theory, since supervisory controllers have the ability to integrate **logic** (i.e., discrete-event dynamic system) with **continuous/discrete dynamics** (i.e., differential/difference equations) in the control of complex systems [49]. Therefore, SCT is suitable for any resource management problem (such as managing power, thermal, QoS, and interconnects) that can be modeled using logic and discrete system dynamics.

[1] *Set-points* are reference values tracked by a controller to reach a particular target (e.g., a desired frame rate), and *gains* are internal controller parameters (e.g., the coefficients of the proportional, integral and derivative (PID) terms in a PID controller) [36].

SPECTR enables dynamic management of multiple shared many-core system resources in a coordinated and autonomous fashion by enforcing higher-level objectives using the formal SCT approach for global resource management at the highest level, with the local resource allocators formulated as traditional control problems (e.g., PID-based SISO [36], LQG-based MIMO [75]). To our knowledge, SPECTR is the first attempt to leverage SCT in the on-chip resource management domain to achieve both scalability and autonomy.

We implement SPECTR on an ODROID-XU3 platform [35] which contains an ARM big.LITTLE based Exynos 5422 Octa-core SoC that has heterogeneous multi-processing cores.[2] We show experimental results to demonstrate the effectiveness of SPECTR in orchestrating multiple low-level classic controllers using SCT techniques such as gain scheduling, precompensation, and reference regulation. Our experiments compare SPECTR's ability to meet quality of service (QoS) references (i.e., targets) while operating under a power budget against two alternative resource-management techniques adapted from state-of-the-art solutions [66, 93]. The resource managers control both the operating frequency (DVFS) and the number of active cores in an HMP. We evaluate the different resource managers on their accuracy and autonomy when managing a system under varying conditions with different goals. Our results show that SPECTR not only matches or surpasses the performance of the best state-of-the-art techniques in *all* cases, but also uniquely and efficiently adapts to both workload variation and dynamic system requirements.

Key contributions of this paper are:

- **Scalability:** We quantify the scalability deficiency of classical control-theoretic approaches in managing resources of complex many-core systems. We do so by analyzing the challenges of building models (i.e., system identification) for such controllers. We present the benefits of SCT in managing the complexity of control problems for HMPs, primarily via modular decomposition.
- **Autonomy:** We show that lack of resource management autonomy in response to changing system goals can either endanger the safety of the system or under-utilize shared resources. We deploy the idea of gain scheduling from SCT to achieve autonomy through dynamic goal management by updating the policy (i.e., parameters) of low-level controllers according to high-level goal(s).
- **Experimental Case Study:** We demonstrate the effectiveness of SPECTR via real system implementation on the Exynos platform, showing SPECTR's accuracy in maintaining QoS and responsiveness to dynamic power constraints for workloads with unpredictable background task interference.
- **Systematic Design Flow:** We present a systematic design flow for HMP architects to ease the task of hierarchical SCT design and verification, simplifying the development of resource managers with autonomy and scalability while preserving the other beneficial properties of control theory such as formalism, robustness analysis, and efficiency.

[2] In the remainder of this work, we refer to this as the Exynos platform.

2 Motivation

We motivate the need for supervisory control of multiple low-level controllers (e.g., MIMOs) to provide autonomy and scalability in resource management.

2.1 Autonomy: Managing Dynamic System-Wide Goals

Controllers may behave non-optimally, or even detrimentally, in meeting a shared goal without knowledge of the presence or behavior of seemingly orthogonal controllers [7, 13, 24, 25, 72, 86]. Consider the MIMO controller in Figure 1 that controls a single-core system with two control inputs ($u(t)$) and interdependent measured outputs ($y(t)$) [66]. The controller tracks two objectives (frames per second, or FPS, and power consumption) by controlling two actuators (operating frequency and cache size). We implement the MIMO using a Linear Quadratic Gaussian (LQG) controller [75] similarly to [66]:

$$x(t+1) = A \times x(t) + B \times u(t) \quad (1)$$

$$y(t) = C \times x(t) + D \times u(t) \quad (2)$$

where x is the system state, y is the measured output vector, and u is the control input vector.[3]

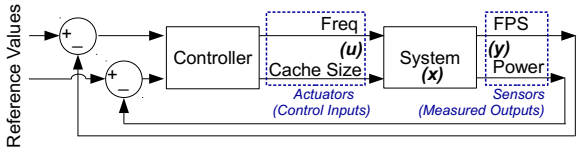


Figure 1. Basic 2×2 MIMO for single-core system. Clock frequency and cache size are used as control inputs. FPS and power are measured outputs that are compared with reference (i.e., target) values.

LQG control allows us to specify 1) the relative sensitivity of a system to control inputs, and 2) the relative priority of measured outputs. This is done using 1) a weighted Tracking Error Cost matrix (Q) and 2) a Control Effort Cost matrix (R). The weights are specified during the design of the controller. While this is convenient for achieving a fixed goal, it can be problematic for goals that change over time (e.g., minimizing power consumption before a predicted thermal emergency).

The controller must choose an appropriate trade-off when we cannot achieve both desirable performance and power concurrently. Unfortunately, classical MIMOs fix control weights at design time, and thus *cannot* perform *runtime* tradeoffs that require changing output priorities. Even with constant *reference values*, i.e., desired output values, unpredictable disturbances (e.g., changing workload and operating conditions) may cause the reference values to become unachievable. It is also plausible for the reference values themselves to change dynamically at runtime with system state and operating conditions (e.g., a thermal event).

Let us now consider a more complex scenario: a multi-threaded application running on Linux, executing on a mobile processor, where the system needs to track both the performance (FPS) and power simultaneously. Figure 2 shows the 2 × 2 MIMO model for this system with operating frequency and the number of active cores as control inputs, and FPS and power as measured outputs.

[3]We interchangeably use the terms (*measured output* and *sensor*), as well as the terms (*control input* and *actuator*), as shown in Figure 1.

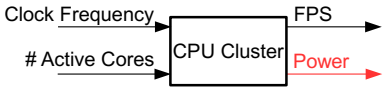


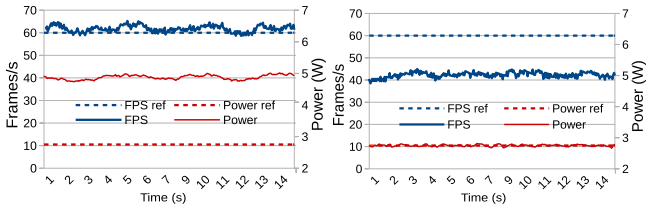
Figure 2. 2 × 2 MIMO model for a quad-core ARM cluster.

Both the FPS and power reference values are trackable individually, but not jointly. We implement and compare two different MIMO controllers in Linux to show the effect of competing objectives. One controller prioritizes FPS, and the other prioritizes power. Figure 3 shows the power and performance (in FPS) achieved by each MIMO controller using typical reference values for a mobile device: 60 FPS and 5 Watts. The application is x264, and the mobile processor consists of an ARM Cortex-A15 quad-core cluster. Each MIMO controller is designed with a different Q matrix to prioritize either FPS or power: Figure 3a's controller favors FPS over power by a ratio of 30:1 (i.e., only 1% deviation from the FPS reference is acceptable for a 30% deviation from the power reference), while Figure 3b uses a ratio of 1:30. We observe that neither controller is able to manage changing system goals. Thus, there is a need for a supervisor to autonomously orchestrate the system while considering the significance of competing objectives, user requirements, and operating conditions.

The use of supervisory control presents at least three additional advantages over conventional controllers. First, fully-distributed MIMO or SISO controllers *cannot* address system-wide goals such as power capping. Second, conventional controllers *cannot* model actuation effects that require system-wide perspective, such as task migration. Third, classical control theory *cannot* address problems requiring optimization (e.g., minimizing an objective function) alone [49, 66].

2.2 Scalability Issue 1: System Identification Complexity

Using a single MIMO controller for coordinated management of a large system comprised of several parallel subsystems (e.g., multi-core systems) is often not possible. The first step in controller design is to construct the dynamic system model by using either analytical models or black-box system identification methods. Constructing analytical models for complex structures such as processors is very challenging [36, 67] and is often performed for simple first-order SISO systems. Quantifying the effect of frequency scaling on measured power consumption is one example [55]. It is more practical to use statistical or black-box methods based on *System Identification Theory* [52, 53] for isolating the deterministic and stochastic



(a) FPS-oriented controller. **(b)** Power-oriented controller.

Figure 3. x264 running on a quad-core cluster controlled by 2 × 2 MIMOs with different output priorities.

components of the system and building the model for complex systems. An *Uncertainty Factor* is added to the model and *Robustness Analysis* [75] is performed to guarantee that the controller will correctly work with this level of uncertainty.

Each control input should have an impact on all measured outputs to properly identify a target system. This is not always the case in multi-core systems. Figure 4 (left) shows a 4×4 MIMO for a generic multi-core system. Actuators 1 and 4 affect the entire system, while Actuators 2 and 3 are limited to different specific subsystems. Similarly, Sensors 1 and 4 measure system-wide metrics, while Sensors 2 and 3 measure metrics at the subsystem level. This is problematic in designing a MIMO, because we must identify the system as a black box without any knowledge of subsystems. Multi-core systems often contain actuators and sensors with varying granularity in this manner. For example, consider the Exynos platform [30] that contains eight cores divided into two clusters: DVFS frequency settings and power sensors are applied at the cluster level, while performance counters are deployed per-core, which requires a 10×10 MIMO (Figure 4, right). An HMP has the additional property of incorporating non-uniform cores, which means system-wide actuators can have greatly different effects on subsystems.

The number of inputs and their subsystem scope has a significant impact on the difficulty of identifying an accurate system model. Figure 5 shows the modeled vs. observed system behavior of the power output for two MIMOs: the **left** plot is for a 2×2 controller (system in Figure 2) with per-cluster inputs and outputs; the **right** plot is for a 10×10 controller (system in Figure 4), showing significant deviation in accuracy. Section 5.2 quantifies and discusses system scalability in detail. We conclude that a single MIMO for controlling a multi-core system is not practical; instead, we propose using *multiple coordinated MIMOs* to control a multi-core platform.

2.3 Scalability Issue 2: Unmanageable State Space

Prior MIMO controller coordination in computer systems [66] is limited to small, simple systems, such as a single-core processor, and does not scale well. The complexity of a MIMO controller grows exponentially with the number of inputs and outputs due to the size of the state space.

During the design of an LQG controller, we must generate coefficient matrices A, B, C, D (Equations 1,2) in order to characterize the system [75]. Whenever the controller is invoked, the matrix multiplication operations in Equations 1 and 2 are executed. The largest of these matrices is A, whose dimensions are determined by $\#inputs + order$ and $\#outputs + order$. In a discrete controller, the *order* of a controller model determines how observed output history is stored in the model, and directly impacts both the controller size

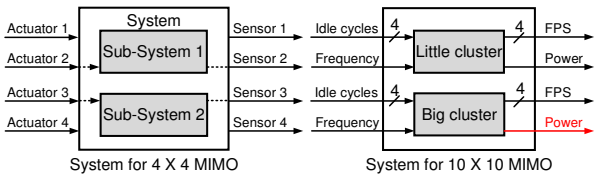


Figure 4. System for 4×4 (left) and 10×10 (right) MIMOs. The 10×10 has 8 per-core idle cycle insertion + 2 per-cluster frequency inputs, and 8 per-core FPS + 2 per-cluster power outputs.

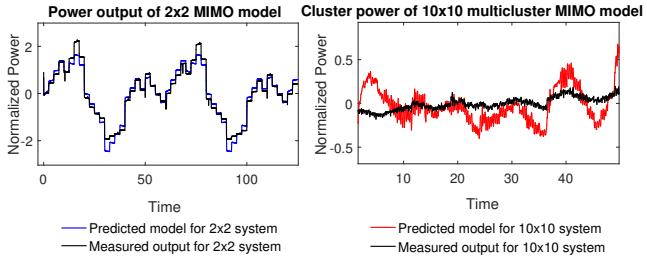


Figure 5. The accuracy of identified system models for a 2×2 MIMO (Figure 2) compared to a 10×10 MIMO (Figure 4). Shown is a single cluster-wide measured output for power, normalized around mean values.

and complexity. For a 2×2 MIMO, these matrices are up to 4×4 for a second-order model. An LQG controller's order must be larger than its number of outputs to work efficiently [36].

The coefficient matrices grow as we increase the order of the model. For instance, consider the fourth-order model used by Pothukuchi et al. [66], resulting in a maximum matrix size 6×6. Assume a fourth-order model for the remainder of this example. If we add one more actuator (e.g., reorder buffer size) to the control system inputs, the matrices grow to 7×6 for just a single-core processor [66]. Now, if we use the same technique to design a single MIMO for a multi-core processor, the size of the controller will grow to unmanageable sizes, complicating the controller design through system identification, and impacting the size and computational complexity of the controller implementation. To manage the same objectives for a dual-core system, our 2×2 MIMO would turn into a 4×4 MIMO (duplicate control inputs and measured outputs for each core). This would require matrices of size 8×8, and even larger for a multi-cluster HMP (Figure 4). Figure 6 shows the number of operations required each time the LQG controller in Equation 1 is invoked for different numbers of cores and orders. The number of multiply and add operations required for matrix multiplication grows exponentially along with the number of cores (i.e., number of inputs and outputs). The order becomes insignificant once $\#cores >> order$. Designing a single controller for runtime management of a many-core processor is therefore infeasible.

By deploying multiple MIMOs to manage a multi-core platform, we risk the same pitfalls as any uncoordinated management scheme. We must therefore provide coordination of the MIMOs. In order to

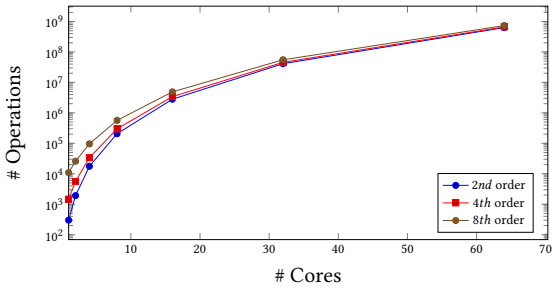


Figure 6. The total number of multiply-add operations required for matrix multiplication of a MIMO for different orders and core counts (input/output sizes).

preserve the benefits of MIMO control, we choose to deploy a local SISO/MIMO for each subsystem to manage identical objectives. This local SISO/MIMO is designed for its specific subsystem (e.g., core or cluster). This does not solve the issue of actuators and sensors applied and observed at different granularities in the platform. To manage hierarchy and system-wide inputs and outputs, we need either a supervisor on top of our local controllers, or a nested controller approach. A nested controller is *not* scalable, and is unable to manage the multiple modes of operation we may encounter due to conflicting objectives [66]. Therefore, we propose a *system-wide supervisor* to coordinate and control objectives within and between each local controller.

3 Background on Supervisory Control Theory

Supervisory control utilizes modular decomposition to mitigate the complexity of control problems, enabling automatic control of many individual controllers or control loops. Supervisory control theory (SCT) [73] benefits from formal synthesis methods to define principal control properties for *controllability* and *observability*. The emphasis on formal methods in addition to *modularity* leads to *hierarchical consistency* and *non-conflicting* properties.

3.1 Scalability via Supervisory Control

SCT solves complex synthesis problems by breaking them into small-scale sub-problems, known as modular synthesis. The results of modular synthesis characterize the conditions under which decomposition is effective. In particular, results identify whether a valid decomposition exists. A decomposition is valid if the solutions to sub-problems combine to solve the original problem, and the resulting composite supervisors are *non-blocking* and *minimally restrictive*. Decomposition also adds robustness to the design because nonlinearities in the supervisor do *not* directly affect the system dynamics.

Figure 7 illustrates how a supervisory control structure can hierarchically manage control loops. As shown in the figure, supervision is vertically decomposed into tasks performed at different levels of abstraction [84]. The supervisory controller is designed to control the high-level *plant model* P_{hi}, which represents an abstraction of the system. The *plant* is the pre-existing system that does *not* (without the aid of a controller or a supervisor) meet the given specifications. Information channel Inf_{hi} provides information about the updates in the high-level model to the supervisory controller, and the supervisory controller uses the Con_{hi} channel to control this model. However, due to the fact that P_{hi} is an abstract model, the controlling channel Con_{hi} is only a **virtual** channel. In other words, the control decisions of the supervisory controller will be implemented by controlling the low-level controller(s) C_{lo} through commands transmitted via the communication channel Com_{hi_lo}. Consequently, the low-level controller(s) C_{lo} can control one or multiple subsystems using the Con_{lo} channel and gather information via the observation channel Inf_{lo}. The changes in the low-level plant P_{lo} can trigger updates in the state of the high-level model P_{hi} through the information channel Inf_{lo_hi}. These updates would reflect the results of low-level controller C_{lo}'s controlling actions. The scheme of Figure 7 describes the division of supervision into *high-level management* and *low-level operational supervision*. Virtual control exercised via the Con_{hi} high-level control channel can be implemented via Com_{hi_lo} to adaptively coordinate the low-level controllers, for example by adjusting their operating modes according to the system goal. The important requirement of this hierarchical control scheme is *control consistency* and *hierarchical consistency* between the high-level model and the low-level plant, as defined in the standard Ramadge-Wonham control mechanism [84]. For a detailed description of SCT, we refer the reader to [5, 73, 74, 84].

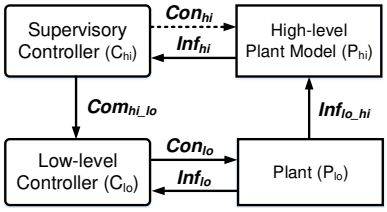


Figure 7. Scalability via Supervisory Control Structure.

3.2 Autonomy via Supervisory Control

Supervisory controllers are preferable to *adaptive (self-tuning) controllers* for complex system control due to their ability to integrate **logic** with **continuous dynamics**. Specifically, supervisory control has two key properties: i) rapid adaptation in response to abrupt changes in management policy [37], and ii) 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 (e.g., by upgrading the firmware or OS), rendering online learning-based self-tuning methods, e.g., least-squares estimation [3], unnecessary.

Figure 8 depicts the two mechanisms that enable SCT-based management via low-level controllers: **gain scheduling** and **dynamic references**. Gain scheduling is a nonlinear control technique that uses a set of linear controllers predesigned for different operating regions. Gain scheduling enables the appropriate linear controller based on runtime observations [51]. Scheduling is implemented by switching between sets of control parameters, i.e., $A_1 \rightarrow A_2$, $B_1 \rightarrow B_2$, $C_1 \rightarrow C_2$, and $D_1 \rightarrow D_2$ in Equations 1 and 2. In this case, the *controller gains* are the values of the control parameters A, B, C, and D. Gains are useful to change objectives at runtime in response to abrupt and sudden changes in management policy. In LQG controllers, this is done by changing priorities of outputs using the Q and R matrices (Section 2.1). This is what we call the Hierarchical Control structure, in which local controllers solve specified tasks while the higher-level supervisory controller coordinates the global objective

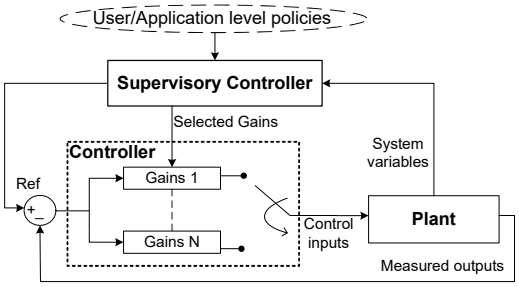


Figure 8. Autonomy via gain scheduling in SCT.

function. In this structure, the supervisory controller receives information from the plant (e.g., the presence of a thermal emergency) or the user/application (e.g., new QoS reference value), and steers the system towards the desired policy using its design logic and high-level model. Thanks to its top-level perspective, the supervisor can update reference values for each low-level controller to either optimize for a certain goal (e.g., getting to the optimum energy-efficient point) or manage resource allocation (e.g., allocating power budget to different cores).

4 SPECTR: On-chip Resource Management

We present SPECTR's supervisory control architecture (Section 4.1), describe an experimental case study demonstrating the design and verification of SPECTR on the Exynos HMP platform (Section 4.2), and outline SPECTR's control synthesis process (Section 4.3).

4.1 Hierarchical System Architecture

Figure 9 depicts a high-level view of SPECTR for many-core system 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, 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 [74][37][61]) to track the system state and provide the supervisor with guidelines. We illustrate the steps for designing a supervisory controller using the following experimental case study in which SCT is deployed on a real HMP platform, and we then outline the entire design flow from modeling of the high-level plant to generating the supervisory controller.

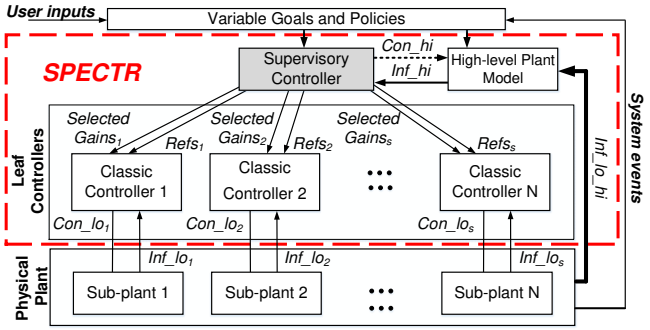


Figure 9. SPECTR overview.

4.2 Experimental Case Study

Figure 10 shows an overview of our experimental setup. We target the Exynos platform [35], which contains an HMP with two quad-core clusters: the **Big** core cluster provides high-performance out-of-order cores, while the **Little** core cluster provides low-power in-order cores. Memory is shared across all cores, so application threads can transparently execute on any core in any cluster. We consider a typical mobile scenario in which a single foreground application (the *QoS application*) is running concurrently with many background applications (the *Non-QoS applications*). This mimics a typical mobile use-case in which gaming or media processing is performed in the foreground in conjunction with background email or social media syncs.

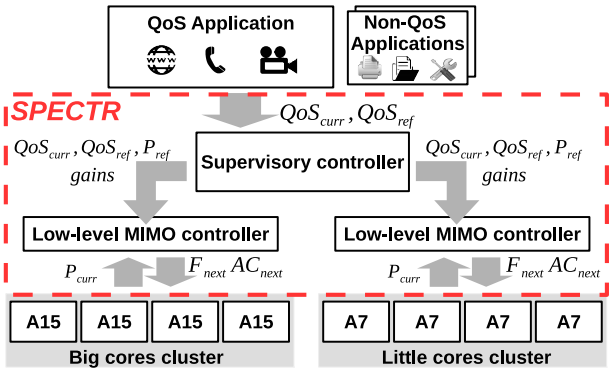


Figure 10. SPECTR implementation on the Exynos HMP with two heterogeneous quad-core clusters. Representing a typical mobile scenario with a single foreground application running concurrently with many background applications.

The system goals are twofold: i) meet the QoS requirement of the foreground application while minimizing its energy consumption; and ii) ensure the total system power always remains below the Thermal Design Power (TDP).

The **subsystems** are the two heterogeneous quad-core (*Big* and *Little*) clusters. Each cluster has two actuators: one actuator to set the operating frequency (F_{next}) and associated voltage of the cluster; and one to set the number of active cores (AC_{next}) on the cluster. We measure the power consumption (P_{curr}) of each cluster, and simultaneously monitor the QoS performance (QoS_{curr}) of the designated application to compare it to the required QoS (QoS_{ref}).[4]

Supervisory control commands guide the **low-level MIMO controllers** in Figure 10 to determine the number of active cores and the core operating frequency within each cluster.

Supervisory control minimizes the system-wide power consumption while maintaining QoS. In our scenario, the QoS application runs only on the Big cluster, and the supervisor determines whether and how to adjust the cluster's power budget based on QoS measurements.

Gain scheduling is used to switch the priority objective of the low-level controllers. We define two sets of gains for this case-study: 1) *QoS-based* gains are tuned to ensure that the QoS application can meet the performance reference value, and 2) *Power-based* gains are tuned to limit the power consumption while possibly sacrificing

[4] The Exynos platform provides only per-cluster power sensors and DVFS; hence our use of cluster-level sensors and actuators.

some performance if the system is exceeding the power budget threshold.

4.3 Supervisor Synthesis Process

The **supervisory controller** is responsible for coordinating the low-level controllers shown in Figure 10. The supervisory control synthesis, illustrated on Figure 11, follows five steps [5]:

1. Develop high-level *Plant Model* (P) as a discrete-event dynamic system.
2. Develop *Intended Behavior Specification* of the plant (S_P) (i.e., desired control behavior).
3. Perform *Synthesis* of the Supervisor (S) from the plant model and behavioral specifications.
4. Perform *Nonblocking Property Checks* to remove any logical/blocking conflicts.
5. Perform *Controllability Property Checks* to ensure that the supervisor meets controllability properties.

In Sections 4.3.1-4.3.3, we discuss each step of modeling, specification, synthesis and verification of the supervisory controller. All steps are automated by the Supremica SCT tool-set [1]. For ease of visualization, we show the automaton generated by Supremica in each step. We integrate the two goals described in Section 4.2 for the system in Figure 10. We ensure autonomy of the system to meet the QoS requirements while the total power remains within the defined boundaries by using gain scheduling.

4.3.1 Plant Model

Any physical plant G can be described using an infinite number of attributes, while the plant model P can capture only a finite number of attributes. Therefore, we begin by capturing the platform's most relevant characteristics (*power consumption* and *QoS* in our study) to build a plant model. Given the formal underpinnings of SCT, we exploit automata theory [41] to automatically generate the plant model from simpler models of its constituent subsystems (i.e., subplants).

Now, consider an automaton A defined as a 5-tuple $A = \langle Q_A, \Sigma_A, \delta_A, i_A, M_A \rangle$, where Q_A is the set of states, Σ_A is the set of events consumed by A, $\delta_A : Q_A \times \Sigma_A \rightarrow Q_A$ is the state transition function, i_A is the initial state and M_A is the set of final states. The *synchronous composition* of two automata A and B, A||B, is then defined as [58]:

$$A \parallel B = \langle Q_A \times Q_B, \Sigma_A \cup \Sigma_B, \delta, i_A \cdot i_B, M_A \times M_B \rangle, \text{with}$$

$$Q_A \times Q_B = \{q_A \cdot q_B | q_A \in Q_a, q_B \in Q_B\}$$

$$\delta(q_A \cdot q_B, e) = \begin{cases} \delta_A(q_A, e) \cdot \delta_B(q_B, e) & \text{if } \delta_A(q_A, e) \text{ and } \delta_B(q_B, e) \text{defined} \\ \delta_A(q_A, e) \cdot q_B & \text{if} \delta_A(q_A, e) \text{ defined and } e \notin \Sigma_B \\ q_A \cdot \delta_B(q_B, e) & \text{if } e \notin \Sigma_A \text{ and } \delta_B(q_B, e) \text{ defined} \\ \text{undefined} & \text{otherwise} \end{cases}$$

Synchronous composition (operator ||) synchronizes the operations of two automata such that common events are synchronized but private events are **not** affected by the other automaton. This preserves the main characteristics of each automaton while including their interactions that affect the whole plant.

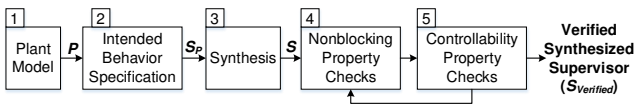


Figure 11. Synthesis process for a Supervisory Controller

Figure 12a shows two simple examples for the Big cluster automata (examplifying two of many possible ways to define our systems control solution). In states *S1* and *S2* of the top automaton, we prioritize QoS: the power reference is updated to meet the QoS reference in a power-efficient manner. Upon detection of a power budget violation, a *critical* signal is generated. The signal results in a transition to the *SwitchGains* state where power-driven gains replace the performance-driven gains. This updates the low-level controller's priority objective from QoS to power. The supervisor also has the opportunity to enforce a reduced power reference have depending on the severity of the situation (*S0* in bottom automaton of Figure 12a). Once the power of the Big cluster returns to a safe region, gains are switched back to prioritize QoS. We can make suitable plant models in a similar manner for the Little cluster and its interaction with the whole system. Figure 12b shows the synchronous composition of the two Big cluster plant models and specifies all possible interactions for these two automata. In this model, all states are accessible and all events are accepted.[5] However, such complete freedom might not be desirable for the system. We now describe the *specification* that restricts this model to fit the intended behaviour of the system.

4.3.2 Intended Behavior Specification

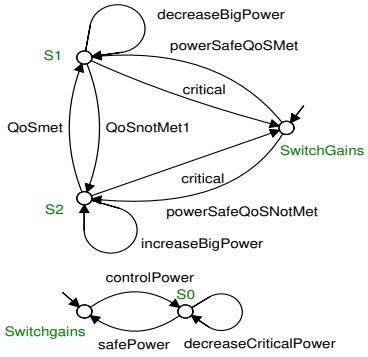
While the plant model sets the physical boundaries for *all* possible actions, the specification defines the accepted (i.e., ideal) and forbidden states through *restrictions* on the behavior of the plant model. These restrictions are then transformed into a formal description for the synthesis process.

Figure 12c shows a sample specification for the Big cluster in our case study. The plant model shown in Figure 12b has no limitations on exceeding the power budget; our specification prevents exceeding the power budget for no more than three control intervals (i.e., *Threshold* state is a forbidden state[6]). Similarly, we can limit the chip power consumption using a specification that restricts the sum of the power budgets of both clusters to be below a safe threshold defined by thermal design power (TDP). In our case study, we use a three-band (i.e., uncapping threshold, capping target and above capping threshold) algorithm similar to [90] for making power capping decisions. While we are below the first threshold (uncapping threshold), controllers focus on meeting their QoS requirements. When we exceed this threshold, gain scheduling ensures that we remain in the capping target region.

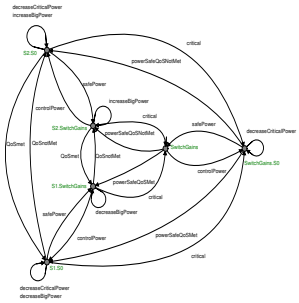
4.3.3 Synthesis

Once we have a plant model and a formal specification of intended behavior, a synthesis algorithm is guaranteed to generate a correct controller [27]. Hence, a correct plant P and specification S_P are crucial to synthesize a supervisory controller S such that the closed-loop system fulfills the specification S_P. Figure 12d shows an example supervisor that was automatically synthesized for the Exynos platform using the Supremica tool, given as input the plant model and the intended behavioral specification capturing desired outcomes and restricting undesired behavior (e.g., Figure 12c). Note

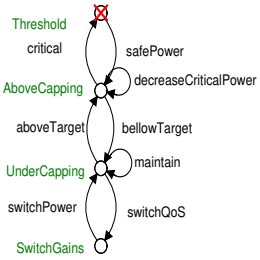
[5] Accepted states are shown with solid dark circles.
[6] A red cross identifies a forbidden state.



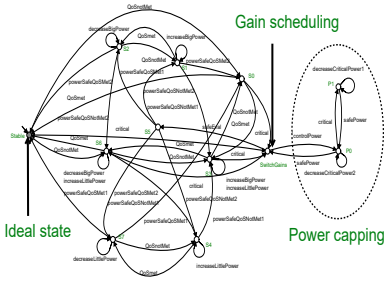
(a) Models for Big cluster: Top: QoS management, Bottom: Power Capping.



(b) Automatically generated Big cluster plant P using the || operator on two models described in Fig. 12a.



(c) Example of intended behavior Specification S_P. Red cross indicates forbidden state.



(d) Synthesized supervisor S generated from plant model and specification using the Supremica tool [1]. Checked for non-blocking and controllability properties.

Figure 12. Supervisor Synthesis Process. Figures 12b and 12d are automatically generated by the SCT tool, and the state details are *not* important.

that the models built for plant P and the intended behavior specification S_P are design artifacts, and only the final synthesized and verified supervisor $S_{verified}$ is implemented in the system. We now describe the verification of additional properties for ensuring correctness of the entire supervisory controller.

4.3.4 Non-blocking and Controllability Property Checks

We must ensure that the synthesized supervisor is both **non-blocking** and **controllable**. The non-blocking property guarantees that some accepted states (e.g., the ideal state shown in Figure 12d) can *always* be reached, so that at least one of the tasks can always be completed. On the other hand, the controllability property guarantees that the supervisor can always keep the plant within the boundaries set by the specification. In our example, there is one accepted (i.e., ideal) state that satisfies the QoS requirement while maintaining the power consumption under the limit. The SCT tool ensures that in the generated supervisor (Figure 12d) there is a path to this accepted state from *every* other *valid* state. In addition, the plant model is pruned by the specification to make it adhere to desired behavior. The closed-loop system will *never* reach a state such that an uncontrollable event causes it to violate the specification. These two properties are provided by two different algorithms: the trimming algorithm [27] provides the non-blocking property, and the extension algorithm [37] provides the controllability property. However, these two algorithms interfere with each other, with trimming possibly impairing controllability, and vice versa. Therefore, the two algorithms must be run successively and iteratively, until they return the same result.

Uncontrollable states. The search for the largest controllable sub-automaton of the specification begins with identifying the uncontrollable states. Subsequently, any state that reaches an uncontrollable state via an uncontrollable event is identified. This forms the basis for the algorithms that construct a controller given a specification and a plant.

Non-blocking. The supervisory controller is non-blocking if the closed-loop system is always able to reach some **marked** state (i.e., *Ideal* state shown in Figure 12d). In order to find a lean non-blocking supervisor, we must find the set of accessible states. It is desirable to find the largest possible sub-automaton that has this property.

5 Experimental Evaluation

We compare SPECTR with three alternative resource managers. The first two managers use two uncoordinated 2×2 MIMOs, one for each cluster: *MM-Pow* uses power-oriented gains, and *MM-Perf* uses performance-oriented gains. These fixed MIMO controllers act as representatives of a state-of-the-art solution, as presented in [66], one prioritizing power and the other prioritizing performance. The third manager consists of a single full-system controller (*FS*): a system-wide 4×2 MIMO with individual control inputs for each cluster. *FS* uses power-oriented gains and its measured outputs are chip power and QoS. This single system-wide MIMO acts as a representative for [93], maximizing performance under a power cap.

We analyze an execution scenario that consists of three different phases of execution:

1. *Safe Phase*: In this phase, only the QoS application executes (with an achievable QoS reference within the TDP). The goal is to meet QoS and minimize power consumption.
2. *Emergency Phase*: In this phase, the QoS reference remains the same as that in the Safe Phase while the power envelope is reduced (emulating a thermal emergency). The goal is to adapt to the change in reference power while maintaining QoS (if possible).
3. *Workload Disturbance Phase*: In this phase, the power envelope returns to TDP and background tasks are added (to induce interference from other tasks). The goal is to meet the QoS reference value without exceeding the power envelope.

This execution scenario with three different phases allows us to evaluate how SPECTR compares with state-of-the-art resource managers when facing workload variation and system-wide changes in state (e.g., thermal emergency) and goals.

Evaluated resource manager configurations. We generate stable low-level controllers for each resource manager using the

Matlab System Identification Toolbox [59].[7] We use the Control Effort Cost matrix (R) to prioritize changing clock frequency over number of cores at a ratio of 2:1, as frequency is a finer-grained and lower-overhead actuator than core count. We generate training data by executing an in-house microbenchmark and varying control inputs in the format of a staircase test (i.e., a sine wave), both with single-input variation and all-input variation. The micro-benchmark consists of a sequence of independent multiply-accumulate operations performed over both sequentially and randomly accessed memory locations, thus yielding various levels of instruction-level and memory-level parallelism. The range of exercised behavior resembles or exceeds the variation we expect to see in typical mobile workloads, which is the target application domain of our case studies.

Experimental setup. We perform our evaluations on the ARM big.LITTLE [2] based Exynos SoC (ODROID-XU3 board [35]) as described in our case study (Figure 10). We implement a Linux userspace daemon process that invokes the low-level controllers every 50ms. When evaluating SPECTR, the daemon invokes the supervisor every 100ms. We use ARM's Performance Monitor Unit (PMU) and per-cluster power sensors for the performance and power measurements required by the resource managers. The userspace daemon also implements the Heartbeats API [39] monitor to measure QoS. By periodically issuing *heartbeats*, the application informs the system about its current performance. The user provides a performance reference value using the Heartbeats API.

To evaluate the resource managers, we use the following benchmarks from the PARSEC benchmark suite [6] as QoS applications (i.e., the applications that issue heartbeats to the controller): `x264`, `bodytrack`, `canneal`, and `streamcluster`. The selected applications consist of the most CPU-bound along with the most cache-bound PARSEC benchmarks, providing varied responses to change in resource allocation. Speedups from 3.2X (`streamcluster`) to 4.5X (`x264`) are observed with the maximum resource allocation values compared to the minimum. We also use one of four machine-learning workloads as our QoS application: `k-means`, `KNN`, `least squares`, and `linear regression`. These four workloads provide a wide range of data-intensive use cases. For all experiments, each QoS application uses four threads. The background (non-QoS) tasks used in the third execution phase are single-threaded microbenchmarks, and have no runtime restrictions, i.e., the Linux scheduler can freely migrate them between and within clusters.

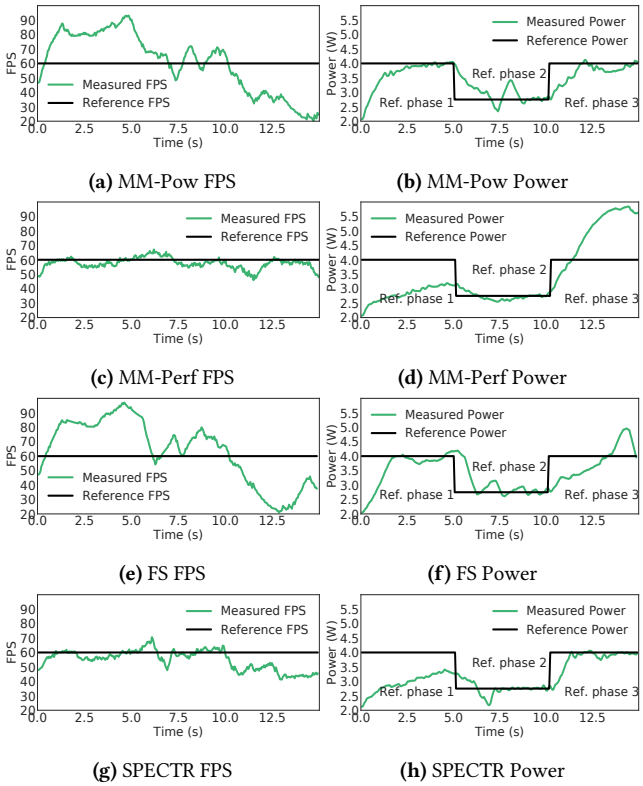
5.1 Comparison of Resource Managers

For brevity, we focus our discussion on the `x264` benchmark results. Other results are summarized at the end of this section. We use heartbeats to measure the frames per second (FPS) as our QoS metric. Figure 13 shows the measured FPS and power for `x264` with respect to their reference values over the course of execution for all of the resource management controllers.

5.1.1 x264 Benchmark

To show the energy efficiency of SPECTR, we study the Safe Phase. The Safe Phase consists of the first 5 seconds of execution during which only the QoS application executes on the Big cluster. In this phase, all controllers are able to achieve the FPS reference value

[7] We generate the models with a stability focus. All systems are stable according to Robust Stability Analysis. We use Uncertainty Guardbands of 50% for QoS and 30% for power, as in [66].



(a) MM-Pow FPS **(b)** MM-Pow Power

(c) MM-Perf FPS **(d)** MM-Perf Power

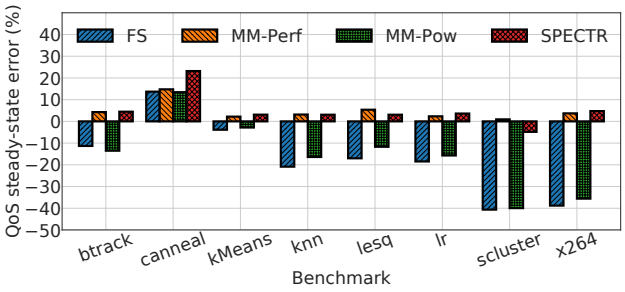
(e) FS FPS **(f)** FS Power

(g) SPECTR FPS **(h)** SPECTR Power

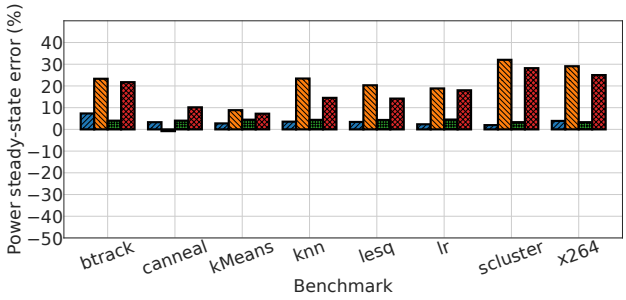
Figure 13. Measured FPS and Power of all four resource managers for three Phases of 5 seconds each, for the `x264` benchmark.

within the power envelope. Figures 14a and 14b show the average steady-state error (%) of QoS and power respectively for each resource manager in Phase 1. Steady-state error is used to define *accuracy* in feedback control systems [36]. Steady-state error values are calculated as $reference - measured\ output$. Negative values indicate that the power/QoS **exceeds** the reference value, positive values indicate power savings or failure to meet QoS. We make two key observations. First, both MM-Perf and SPECTR reduce power consumption by 25% (Fig. 14b) while maintaining FPS within 10% (Fig. 14a) of the reference value. The MM-Perf controller operates efficiently because the reference FPS value is achievable within the TDP threshold. The SPECTR controller similarly operates efficiently: it is able to recognize that the FPS is achievable within TDP and, as a result, lower the reference power. Second, the FS and MM-Pow controllers unnecessarily exceed the reference FPS value and, as a result, consume excessive power. This is because these controllers prioritize meeting the power reference value, consuming the entire available power budget to maximize performance.

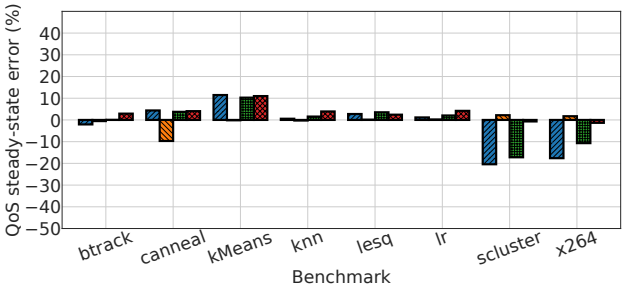
To show SPECTR's ability to adapt to a sudden change in operating constraints, we study the Emergency Phase. The Emergency Phase of execution emulates a thermal emergency, during which, the TDP is lowered to ensure that the system operates in a safe state. This occurs during the second 5-second period of execution in Figure 13. We observe that all controllers are able to react to the change in power reference value and maintain QoS. However, compared to the other controllers, FS has a sluggish reaction (Figure 13f) to the change in power reference, despite the fact that it



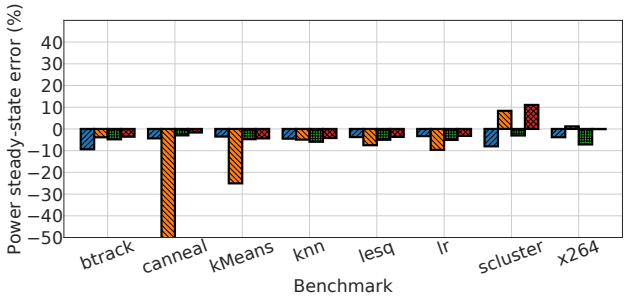
(a) QoS steady-state error in Phase 1.



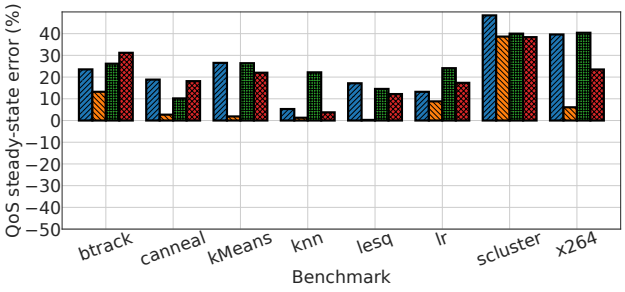
(b) Power steady-state error in Phase 1.



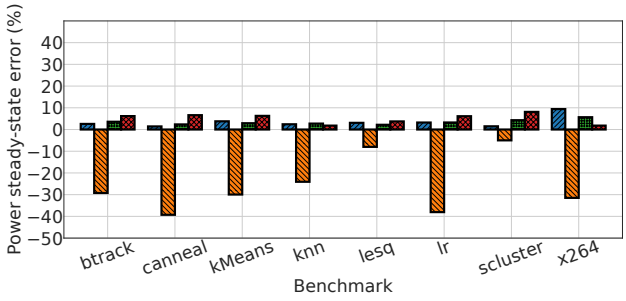
(c) QoS steady-state error in Phase 2.



(d) Power steady-state error in Phase 2.



(e) QoS steady-state error in Phase 3.



(f) Power steady-state error in Phase 3.

Figure 14. Steady-state error for all benchmarks, grouped by phase. A negative value indicates the amount of power/QoS **exceeding** the reference value (bad), a positive value indicates the amount of power saved (good) or QoS degradation (bad).

is designed to prioritize tracking the power output. *Settling time* is a property used to quantify responsiveness of feedback control systems [36]. *Settling time* is the time it takes to reach sufficiently close to the steady-state value after the reference values are set. The average settling time for the power output of FS is 2.07 seconds, while SPECTR has an average settling time of 1.28 seconds. The larger size of the state-space ($x(t)$ matrix in Equation 1 and 2) and the higher number of control inputs in the 4×2 FS compared to those of 2×2 controllers in SPECTR is the reason for the slow settling time of FS. This is also the reason why SISO controllers are generally faster that MIMOs [36].

To show SPECTR's ability to adapt to workload disturbance and changing system goals, we study the Workload Disturbance Phase. The Workload Disturbance Phase occurs in seconds 10-15 of execution in Figure 13. In this phase, 1) the QoS reference value and the power envelope return to the same values as in Phase 1, and 2) we introduce disturbance in the form of background tasks. As a result of the workload disturbance, the QoS reference is *not* achievable within the TDP. We make two observations regarding the steady-state error in Figures 14e and 14f. First, SPECTR behaves similarly to MM-Pow, even though in Phase 1 it behaved similarly to MM-Perf. The SPECTR supervisor is able to recognize the change in execution scenario and constraints, and adapt its priorities appropriately. In this case, SPECTR achieves much higher FPS than all controllers except MM-Perf (Fig. 14e), while obeying the TDP limit (Fig. 14f). Second, both FS and MM-Pow operate at the TDP limit, but achieve a significantly lower FPS than the reference value. MM-Perf comes within ~5% of the reference FPS (Fig. 14e) while exceeding the TDP by more than 30% (Fig. 14f), which is undesirable.

5.1.2 Other Benchmarks

We perform the same experiments for PARSEC benchmarks `bodytrack`, `canneal`, `streamcluster`, as well as machine-learning benchmarks `k-means`, `KNN`, `least squares`, and `linear regression`. For these workloads, we use the generic *heartbeat rate* (HB) directly as the QoS metric, as FPS is not an appropriate metric. Figures 14a, 14c, and 14e show the average steady-state error (%) of QoS for Phases

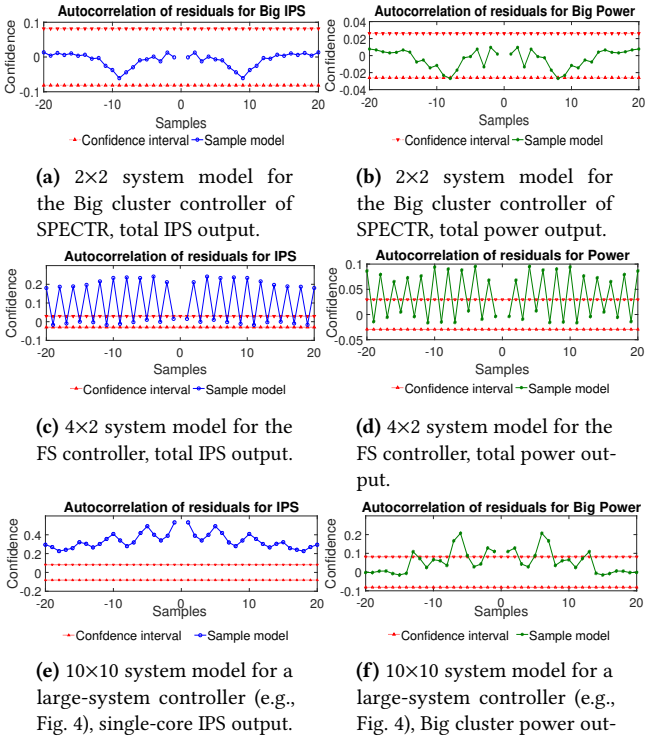
1, 2, and 3 respectively. Figures 14b, 14d, and 14f show the average steady-state error (%) of power for Phases 1, 2, and 3 respectively. We summarize the observations for the additional experiments with respect to x264 for the three phases. In the Safe Phase, the behavior of bodytrack, streamcluster, k-means, KNN, least squares, and linear regression is similar to that of x264 (Figures 14a and 14b). canneal follows the same pattern with respect to power as all other benchmarks (Fig. 14b). canneal's QoS steady-state error is the only difference in behavior we observe in Phase 1. None of the managers are able to meet the QoS reference value for canneal in Phase 1 (Fig. 14a). This is due to the fact that the phase of canneal captured in the experiment primarily consists of serialized input processing, so the number of idle cores has reduced affect on QoS. In the Emergency Phase, our observations from x264 hold for nearly all benchmarks regarding response to change in power reference value, achieving less than 10% power steady-state error (Fig. 14d). The only exceptions are canneal and k-means: the MM-Perf manager is unable to react to change in TDP for canneal and k-means. The MM-Perf manager lacks a supervisory coordinator and prioritizes performance, and was unable to find a configuration for canneal and k-means that satisfied the QoS reference value within TDP. In the Workload Disturbance Phase, SPECTR, FS, and MM-Pow all achieve near-reference power (Fig. 14f). As expected, MM-Perf violates the TDP in all cases, but always achieves the highest QoS (Fig. 14e).

We conclude that SPECTR is effective at (1) efficiently meeting multiple system objectives when it is possible to do so, (2) appropriately balancing multiple conflicting objectives, and (3) quickly responding to sudden and unpredictable changes in constraints due to workload or system state.

5.2 Scalability Evaluation

To evaluate the scalability of SPECTR with respect to single or nested MIMO solutions, we compare the identified models for controlled systems of different sizes. After an estimated system dynamics is produced using system identification techniques, it is cross-validated using different data sets. The common practice is to assess the model by analyzing *residual* auto-correlation [67]. Residual is the stochastic component (e.g., disturbance, noise, etc.) of the system output, which is not supposed to be included in the model. When validating the model, the model output is compared to noisy system outputs. Therefore we expect the residual to be pure noise. To verify this, the residual is analyzed for correlation. If there is *no* correlation between the residual and itself or any inputs, the model is accurate enough. *Confidence* can be used to specify a range. A confidence level is the probability with which the true output will fall into a range called a *confidence interval*. The confidence interval provides a range of values that is likely to contain the population parameter of interest [64]. A confidence level of 99% results in a confidence interval that spans three standard deviations. In our case, a higher confidence level means more confidence in where the true output will lie, and a model output within the confidence interval indicates that the deterministic component of the model output will be near the true output.

Figure 15 compares the autocorrelation of residuals for instructions per second (IPS) and power of three systems: 1) 2×2 Big cluster MIMO used in SPECTR, 2) 4×2 MM-Pow, and 3) 10×10 controller that represents a large system (Figure 4). The 2×2 controller for the Little cluster shows similar behavior to the 2×2 controller for



(a) 2×2 system model for the Big cluster controller of SPECTR, total IPS output.

(b) 2×2 system model for the Big cluster controller of SPECTR, total power output.

(c) 4×2 system model for the FS controller, total IPS output.

(d) 4×2 system model for the FS controller, total power output.

(e) 10×10 system model for a large-system controller (e.g., Fig. 4), single-core IPS output.

(f) 10×10 system model for a large-system controller (e.g., Fig. 4), Big cluster power output.

Figure 15. Autocorrelation of residuals for identified system models of different sized MIMO controllers. We show a single performance and power output for each modeled system across multiple sample inputs.

the Big cluster in Figure 15a. MM-Perf controller shows similar behavior to MM-Pow in Figure 15c.

The two main properties desired while checking the autocorrelation of residuals are for the controller to: 1) stay inside the confidence interval, and 2) avoid sharp peaks and drops. While the 2×2 controller stays within the confidence interval for IPS and power (Figure 15a,15b), the 4×2 controller exhibits sharp peaks that *violate* the confidence interval for multiple sample inputs (Figure 15c,15d). The controller for the large 10×10 system has difficulty staying within the confidence interval, especially for IPS (Figure 15e,15f). Controllers for large MIMO systems with more complex behavior are not only slower in terms of settling time, but also often infeasible to design due to the lack of a sufficiently accurate system dynamics model.

We conclude that SPECTR supports scalability for resource management that classical controllers do not. Classical controllers cannot accurately model large systems. SPECTR solves this issue by deploying many simple controllers for decomposed subsystems, and coordinating them with a high-level supervisor.

5.3 Overhead Evaluation

To show the overhead of the low-level MIMO controllers, we study their execution time. We measure the MIMO controller execution time to be 2.5*ms*, on average, over 30 seconds. The MIMO controller is invoked every 50*ms* resulting in a 5% overhead, which is experienced by all evaluated controllers. We measure the runtime of

the supervisor to be 30μs, which is negligible even with respect to the MIMO controller execution time. The supervisor is invoked less frequently than the MIMO controllers (2× the period in our case), executes in parallel to the workload and MIMO controllers, and simply evaluates the system state in order to determine if the MIMO controller gains need changing. State changes that result in interventions on the low-level controllers occur only due to system-wide changes in the state (e.g., thermal emergency) or goals (e.g., change in performance reference value or execution mode), which are infrequent. When the supervisor needs to change the MIMO gains, it simply points the coefficient matrices to a different set of stored values. In our case study, we have two sets of gains (QoS and power oriented) that are generated when the controllers are designed and stored during system initialization. Changing the coefficient arrays at runtime takes effect immediately, and has no additional overhead.

To show the overhead of SPECTR's supervisory controller, we compare the total execution time of identical workloads with and without SPECTR. With respect to the preemption overhead due to globally managing resources, Linux's HMP scheduler typically maps SCT threads to a core on the low-power Little cluster. Therefore, the SCT threads are executed without preempting the QoS application, which always executes on the Big cluster. We verify the overall impact of the control system overhead by running the benchmarks on two different systems: i) a vanilla Linux setup[8] and ii) vanilla Linux with SPECTR running in the background. For (ii), SPECTR controllers perform all the required computations but do *not* change the system knobs (thus only the SPECTR overhead affects the system). When comparing the QoS of the applications across multiple runs, we verify a negligible average difference of 0.1% between the two systems.

We conclude that the benefits of SPECTR come at a negligible performance overhead.

6 Systematic Design Flow of SPECTR

Figure 16 presents SPECTR's design flow to streamline the process for HMP architects to build supervisory controllers for new platforms and resource types.

The top part of Figure 16 illustrates the design process of the Supervisory Controller. In **Step 1**, we need to define the high-level goals (e.g., power capping, QoS) for coordination and resource management for the entire system. In **Step 2**, we create a plant model for our system to generate the supervisory controller. For small systems, this can be done in a single step by describing all possible variations of the system, but more complex systems can be modeled via the modular decomposition of the system. This enables specification of each subsystem as an individual sub-plant in a formal manner, with individual sub-plants combined later to automatically generate the full plant model. These sub-plants can be also broken down into smaller elements for ease of modeling. This step is crucial because the scope of sensors and actuators have to be determined to ensure that i) the system is properly identifiable and ii) subsystem controllers, to be designed in later steps, are lightweight. The former condition is satisfied if the *coefficient of determination*, also known as R^2, is greater or equal than 80% [36] (a rule of thumb in control theory), while the latter can be examined by considering the number of control inputs, number of measured outputs, and the order of the model. In **Step 3**, we describe the desired system behavior using a specification that restricts the plant model. Preventing the system from exceeding its power budget can be an example of a specification. The supervisory synthesis process in the third step checks the sanity of the supervisory controller based on sub-plant models and the verified specification. SCT tools use a formal methodology to analyze properties of the discrete-event system and also generate the supervisor. In **Step 4**, we ensure that the whole system behaves according to the given specification. In this work, we use the Supremica tool [1] to synthesize and verify the supervisory controller.

The bottom part of Figure 16 illustrates the design process of the low-level (i.e., leaf) controllers. Each individual subsystem can have a different number of control inputs and measured outputs. Various types of controllers (e.g., SISO, MIMO) can be used for leaf controllers. In this work, we focus on the design of a MIMO controller for each leaf controller. In **Step 5**, during the low-level controller design, after defining each subsystem's inputs and outputs, we gather experimental data to perform black-box system identification [53][26] to extract the dynamics of the system (e.g., state-space model, transfer function). Any system identification toolbox, such as the ones available in MATLAB [59] or GNU Octave [22], can be used for this purpose. MATLAB System Identification toolbox also recommends a suitable order for the system. In **Step 6**, we specify priorities associated with each <goal, condition> pair by feeding MATLAB's Control System toolbox with the relative weight of inputs and outputs that are represented by Q and R matrices. For instance, designers can emphasize the relative importance of total power over IPS of one core as a <goal, condition> pair. Similarly, other <goal, condition> pairs prioritizing different objectives can defined and added.

In **Step 7**, using MATLAB, we generate a set of MIMO controllers for each <goal, condition> pair where gains of each controller are stored in the subsystem controllers to be used for gain scheduling at run-time. The goal of **Step 8** is to ensure that the controller is stable for all the uncertainties whose maximum sustained impact is bounded by a designer-specified margin. Finally, in **Step 9**, Matlab's Simulink tool can be used as a hybrid simulation environment to integrate and verify the full-fledged control system. If the overall response of the system is acceptable, we generate the target code for implementation and verification on the real platform. Otherwise, as low-level controllers are already verified, we go back to the supervisory control design and update the specification of the supervisor in order to enhance the overall control behavior.

7 Related Work

To the best of our knowledge, there has been no prior work in applying SCT to handle the scalability and autonomy issues facing resource managers for (heterogeneous) multi-core systems. We believe SPECTR is the first effort in exploiting SCT to provide scalability and autonomy for on-chip HMP resource management.

Resource management approaches in the literature can be classified into five main classes: Optimization [29, 31, 34, 57, 65, 81, 83], Machine Learning [7, 16, 21, 32, 43], Model-based Heuristics [4, 7, 9, 12, 13, 15, 17, 19–21, 23, 24, 46, 54, 62, 78–80, 87–89, 92, 93], Rule-based Heuristics [18, 28, 44], and Control Theory [25, 33, 38, 40, 47, 48, 55, 56, 60, 63, 68, 70, 71, 76, 91]. Pothukuchi

[8] Ubuntu 16.04.2 LTS and Linux kernel 3.10.105 (https://dn.odroid.com/5422/ODROID-XU3/Ubuntu/).

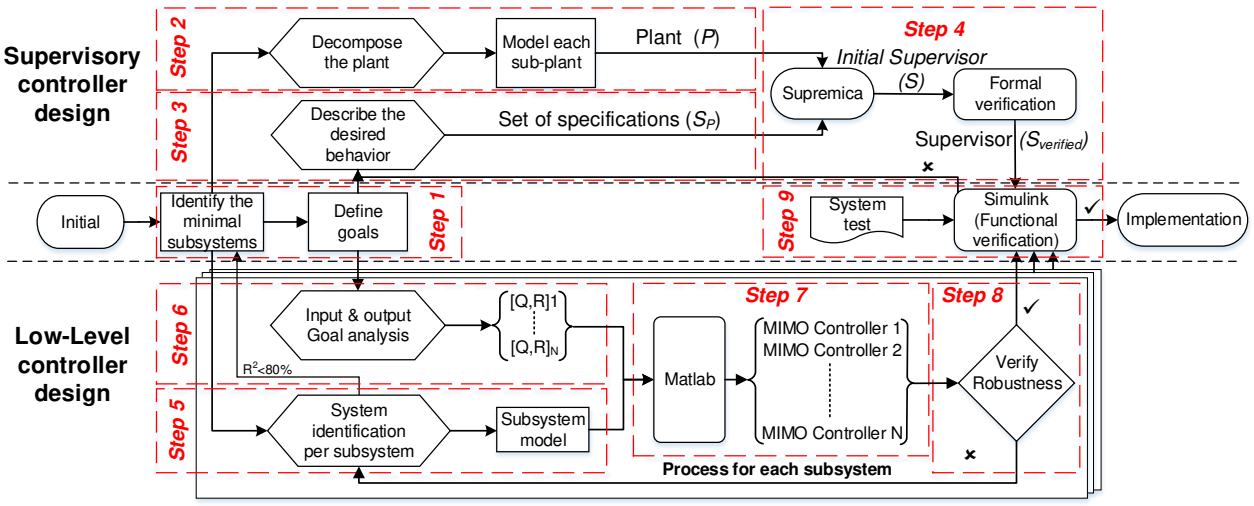


Figure 16. SPECTR design flow. Low-level controller design process is repeated for every subsystem.

et al. [66] discuss the shortcomings of ad-hoc and heuristic-based approaches in addressing some of the attributes, such as lack of guarantees, and the need for exhaustive training and close-to-reality models. In addition, there have been efforts to enable coordinated management in computer systems in various ways [7, 10, 11, 13–15, 21, 23, 24, 45, 50, 69, 77, 82, 85, 86, 90]. They coordinate and control multiple goals and actuators in a non-conflicting manner by adding an ad-hoc component to a controller or hierarchical loops.

Pothukuchi and Torrellas [67] present guidelines for designing formal MIMO controllers that tune processor architectural parameters to enhance coordination, and demonstrate coordinated management of multiple goals for **unicore** processors [66]. However, MIMO control lacks scalability and autonomy in handling complex control problems. The concept of SCT has been used in many fields [8, 27, 42].

8 Conclusion

We develop SPECTR, a hierarchical supervisory control mechanism for resource management in heterogeneous many-core systems. SPECTR combines the strengths of classic control theory with state-of-the-art heuristic approaches to efficiently manage complex systems with multiple goals in a hierarchical manner. SPECTR leverages formal Supervisory Control Theoretic techniques, such as gain scheduling, to achieve autonomy for individual distributed controllers and scalability for the entire system, while satisfying higher-level system goals. We demonstrate the effectiveness of SPECTR via a real system implementation on the Exynos platform that consists of a heterogeneous multi-core processor. Our evaluations show that SPECTR successfully coordinates conflicting objectives to achieve efficient execution of a dynamic QoS workload within a power budget, while state-of-the-art alternatives are unable to do so. We conclude that SPECTR is a promising approach to handle the complexity and scalability of managing the resources of emerging heterogeneous many-core systems in the face of dynamically changing runtime goals such as QoS requirements, power budgets, and thermal limits.

Acknowledgments

We acknowledge financial support from the following: NSF Grant CCF-1704859; and the Marie Curie Actions of the European Union's H2020 Programme.

References

[1] K. Akesson, M. Fabian, H. Flordal, and R. Malik, "Supremica - An integrated environment for verification, synthesis and simulation of discrete event systems," in *WODES 2006.*
[2] ARM, "big.LITTLE Technology: The Future of Mobile," Tech. Rep., 2013. [Online]. Available: https://www.arm.com/files/pdf/big_LITTLE_Technology_the_Futue_of_Mobile.pdf
[3] K. J. Astrom and B. Wittenmark, *Adaptive Control.* Addison-Wesley, 1995.
[4] A. Bartolini, M. Cacciari, A. Tilli, and L. Benini, "A distributed and self-calibrating model-predictive controller for energy and thermal management of high-performance multicores," in *DATE, 2011.*
[5] M. W. Bertil A. Brandin and B. Benhabib, "Discrete Event System Supervisory Control Applied to the Management of Manufacturing Workcells," in *Computer-Aided Production Engineering, C. Venkatesh and J.A. McGeough, eds. (Amsterdam: Elsevier), 1991.*
[6] C. Bienia, "Benchmarking modern multiprocessors," Ph.D. dissertation, Princeton University, 2011.
[7] R. Bitirgen, E. Ipek, and J. F. Martinez, "Coordinated Management of Multiple Interacting Resources in Chip Multiprocessors: A Machine Learning Approach," in *MICRO, 2008.*
[8] J. Buerger and M. Cannon, "Nonlinear MPC for supervisory control of hybrid electric vehicles," in *ECC, 2016.*
[9] K. K. Chang, A. G. Yağlıkçı, S. Ghose, A. Agrawal, N. Chatterjee, A. Kashyap, D. Lee, M. O'Connor, H. Hassan, and O. Mutlu, "Understanding Reduced-Voltage Operation in Modern DRAM Devices: Experimental Characterization, Analysis, and Mechanisms," *Proc. ACM Meas. Anal. Comput. Syst.*, 2017.
[10] S. Choi and D. Yeung, "Learning-Based SMT Processor Resource Distribution via Hill-Climbing," in *ISCA, 2006.*
[11] R. Cochran, C. Hankendi, A. K. Coskun, and S. Reda, "Pack & Cap: Adaptive DVFS and Thread Packing Under Power Caps," in *MICRO, 2011.*
[12] R. Das, R. Ausavarungnirun, O. Mutlu, A. Kumar, and M. Azimi, "Application-to-core mapping policies to reduce memory system interference in multi-core systems," in *HPCA, 2013.*
[13] R. Das, O. Mutlu, T. Moscibroda, and C. R. Das, "Application-aware prioritization mechanisms for on-chip networks," in *MICRO, 2009.*
[14] R. Das, O. Mutlu, T. Moscibroda, and C. R. Das, "AéRgia: Exploiting Packet Latency Slack in On-chip Networks," in *ISCA, 2010.*
[15] H. David, C. Fallin, E. Gorbatov, U. R. Hanebutte, and O. Mutlu, "Memory power management via dynamic voltage/frequency scaling," in *ICAC, 2011.*
[16] C. Delimitrou and C. Kozyrakis, "Quasar: Resource-efficient and QoS-aware Cluster Management," in *ASPLOS, 2014.*

[17] Q. Deng, D. Meisner, A. Bhattacharjee, T. F. Wenisch, and R. Bianchini, "CoScale: Coordinating CPU and Memory System DVFS in Server Systems," in *MICRO, 2012.*

[18] A. S. Dhodapkar and J. E. Smith, "Managing Multi-configuration Hardware via Dynamic Working Set Analysis," in *ISCA, 2002.*

[19] B. Donyanavard, T. Mück, S. Sarma, and N. Dutt, "Sparta: Runtime task allocation for energy efficient heterogeneous many-cores," in *CODES, 2016.*

[20] C. Dubach, T. M. Jones, and E. V. Bonilla, "Dynamic Microarchitectural Adaptation Using Machine Learning," in *TACO, 2013.*

[21] C. Dubach, T. M. Jones, E. V. Bonilla, and M. F. P. O'Boyle, "A Predictive Model for Dynamic Microarchitectural Adaptivity Control," in *MICRO, 2010.*

[22] J. Eaton, D. Bateman, S. Hauberg, and R. Wehbring, *GNU Octave version 3.8.1 manual: a high-level interactive language for numerical computations.* CreateSpace Independent Publishing Platform, 2014.

[23] E. Ebrahimi, C. J. Lee, O. Mutlu, and Y. N. Patt, "Prefetch-aware shared-resource management for multi-core systems," in *ISCA, 2011.*

[24] E. Ebrahimi, C. J. Lee, O. Mutlu, and Y. N. Patt, "Fairness via source throttling: A configurable and high-performance fairness substrate for multi-core memory systems," in *ASPLOS, 2010.*

[25] E. Ebrahimi, O. Mutlu, C. J. Lee, and Y. N. Patt, "Coordinated Control of Multiple Prefetchers in Multi-core Systems," in *MICRO, 2009.*

[26] L. P. Eric Walter, *Identification of Parametric Models from experimental results.* Springer, 1997.

[27] M. Fabian and A. Hellgren, *Desco — a Tool for Education and Control of Discrete Event Systems.* Springer, 2000.

[28] S. Fan, S. M. Zahedi, and B. C. Lee, "The Computational Sprinting Game," in *ASPLOS, 2016.*

[29] X. Fu, K. Kabir, and X. Wang, "Cache-Aware Utilization Control for Energy Efficiency in Multi-Core Real-Time Systems," in *ECRTS, 2011.*

[30] P. Greenhalgh, "Big. little processing with arm cortex-a15 & cortex-a7," in *ARM White paper, 2011.*

[31] U. Gupta, R. Ayoub, M. Kishinevsky, D. Kadjo, N. Soundararajan, U. Tursun, and U. Ogras, "Dynamic Power Budgeting for Mobile Systems Running Graphics Workloads," in *TMSCS, 2017.*

[32] U. Gupta, J. Campbell, U. Y. Ogras, R. Ayoub, M. Kishinevsky, F. Paterna, and S. Gumussoy, "Adaptive performance prediction for integrated GPUs," in *ICCAD, 2016.*

[33] M. H. Haghbayan, A. Miele, A. M. Rahmani, P. Liljeberg, and H. Tenhunen, "Performance/Reliability-Aware Resource Management for Many-Cores in Dark Silicon Era," *IEEE Transactions on Computers*, 2017.

[34] V. Hanumaiah, D. Desai, B. Gaudette, C.-J. Wu, and S. Vrudhula, "STEAM: A Smart Temperature and Energy Aware Multicore Controller," in *TECS, 2014.*

[35] Hardkernel, "ODROID-XU," Tech. Rep. [Online]. Available: http://www.hardkernel.com/main/main.php

[36] J. L. Hellerstein, Y. Diao, S. Parekh, and D. M. Tilbury, *Feedback Control of Computing Systems.* John Wiley & Sons, 2004.

[37] J. P. Hespanha, "Tutorial on supervisory control," in *Lecture Notes for the workshop Control using Logic and Switching for the 40th Conference on Decision and Control, 2011.*

[38] H. Hoffmann, "Coadapt: Predictable behavior for accuracy-aware applications running on power-aware systems," in *ECRTS, 2014.*

[39] H. Hoffmann, M. Maggio, M. D. Santambrogio, A. Leva, and A. Agarwal, "A generalized software framework for accurate and efficient management of performance goals," in *EMSOFT, 2013.*

[40] H. Hoffmann, S. Sidiroglou, M. Carbin, S. Misailovic, A. Agarwal, and M. Rinard, "Dynamic Knobs for Responsive Power-aware Computing," in *ASPLOS, 2011.*

[41] J. E. Hopcroft and J. D. Ullman, *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley, 1979.

[42] Q. Hui, W. Qiao, and C. Peng, "Neuromorphic-computing-based feedback control: A cognitive supervisory control framework," in *CDC, 2016.*

[43] E. Ipek, O. Mutlu, J. F. Martínez, and R. Caruana, "Self-Optimizing Memory Controllers: A Reinforcement Learning Approach," in *ISCA, 2008.*

[44] C. Isci, A. Buyuktosunoglu, C.-Y. Cher, P. Bose, and M. Martonosi, "An Analysis of Efficient Multi-Core Global Power Management Policies: Maximizing Performance for a Given Power Budget," in *MICRO, 2006.*

[45] P. Juang, Q. Wu, L.-S. Peh, M. Martonosi, and D. W. Clark, "Coordinated, distributed, formal energy management of chip multiprocessors," in *ISLPED, 2005.*

[46] H. Jung, P. Rong, and M. Pedram, "Stochastic modeling of a thermally-managed multi-core system," in *DAC, 2008.*

[47] D. Kadjo, R. Ayoub, M. Kishinevsky, and P. V. Gratz, "A Control-theoretic Approach for Energy Efficient CPU-GPU Subsystem in Mobile Platforms," in *DAC, 2015.*

[48] A. Kanduri, M. H. Haghbayan, A. M. Rahmani, P. Liljeberg, A. Jantsch, N. Dutt, and H. Tenhunen, "Approximation knob: Power Capping meets energy efficiency," in *ICCAD, 2016.*

[49] C. Karamanolis, M. Karlsson, and X. Zhu, "Designing Controllable Computer Systems," in *HoTOS, 2005.*

[50] C. J. Lee, V. Narasiman, E. Ebrahimi, O. Mutlu, and Y. N. Patt, "DRAM-aware last-level cache writeback: Reducing write-caused interference in memory systems," UT Austin, Tech. Rep., 2010.

[51] D. Leith and W. Leithead, "Survey of gain-scheduling analysis and design," in *International Journal of Control, 2000.*

[52] L. Ljung, "Black-box models from input-output measurements," in *I2MTC, 2001.*

[53] L. Ljung, *System Identification: Theory for the User.* Prentice Hall PTR, 1999.

[54] D. Lo, T. Song, and G. E. Suh, "Prediction-guided Performance-energy Trade-off for Interactive Applications," in *MICRO, 2015.*

[55] K. Ma, X. Li, M. Chen, and X. Wang, "Scalable power control for many-core architectures running multi-threaded applications," in *ISCA, 2011.*

[56] M. Maggio, H. Hoffmann, M. D. Santambrogio, A. Agarwal, and A. Leva, "Controlling software applications via resource allocation within the heartbeats framework," in *CDC, 2010.*

[57] D. Mahajan, A. Yazdanbakhsh, J. Park, B. Thwaites, and H. Esmaeilzadeh, "Towards Statistical Guarantees in Controlling Quality Tradeoffs for Approximate Acceleration," in *ISCA, 2016.*

[58] F. Maraninchi, "Operational and Compositional Semantics of Synchronous Automaton Compositions ," in *CONCUR, 1992*, aug.

[59] MathWorks, "System Identification Toolbox," Tech. Rep., 2017. [Online]. Available: https://www.mathworks.com/products/sysid.html

[60] A. K. Mishra, S. Srikantaiah, M. Kandemir, and C. R. Das, "CPM in CMPs: Coordinated Power Management in Chip-Multiprocessors," in *SC, 2010.*

[61] S. Morse, *Control using logic-based switching.* Springer, 1997.

[62] T. S. Muthukaruppan, A. Pathania, and T. Mitra, "Price Theory Based Power Management for Heterogeneous Multi-cores," in *ASPLOS, 2014.*

[63] T. S. Muthukaruppan, M. Pricopi, V. Venkataramani, T. Mitra, and S. Vishin, "Hierarchical Power Management for Asymmetric Multi-core in Dark Silicon Era," in *DAC, 2013.*

[64] NIST, "Engineering Statistics Handbook," Tech. Rep. [Online]. Available: http://www.itl.nist.gov/div898/handbook/index.htm

[65] P. Petrica, A. M. Izraelevitz, D. H. Albonesi, and C. A. Shoemaker, "Flicker: A Dynamically Adaptive Architecture for Power Limited Multicore Systems," in *ISCA, 2013.*

[66] R. P. Pothukuchi, A. Ansari, P. Voulgaris, and J. Torrellas, "Using Multiple Input, Multiple Output Formal Control to Maximize Resource Efficiency in Architectures," in *ISCA, 2016.*

[67] R. P. Pothukuchi and J. Torrellas, "A Guide to Design MIMO Controllers for Architectures," in *http://iacoma.cs.uiuc.edu/iacoma-papers/mimoTR.pdf.*

[68] Q. Wu, P. Juang, M. Martonosi, D. W. Clark, "Formal Online Methods for Voltage/Frequency Control in Multiple Clock Domain Microprocessors," in *ASPLOS, 2004.*

[69] R. Raghavendra, P. Ranganathan, V. Talwar, Z. Wang, and X. Zhu, "No "Power" Struggles: Coordinated Multi-level Power Management for the Data Center," in *ISCA, 2008.*

[70] A. M. Rahmani, M. H. Haghbayan, A. Kanduri, A. Y. Weldezion, P. Liljeberg, J. Plosila, A. Jantsch, and H. Tenhunen, "Dynamic power management for many-core platforms in the dark silicon era: A multi-objective control approach," in *ISLPED, 2015.*

[71] A. M. Rahmani, M. H. Haghbayan, A. Miele, P. Liljeberg, A. Jantsch, and H. Tenhunen, "Reliability-Aware Runtime Power Management for Many-Core Systems in the Dark Silicon Era," in *TVLSI, 2017.*

[72] A. M. Rahmani, A. Jantsch, and N. Dutt, "HDGM: Hierarchical Dynamic Goal Management for Many-Core Resource Allocation," in *ESL, 2017.*

[73] P. J. Ramadge and W. M. Wonham, "The control of discrete event systems," in *Proceedings of the IEEE, 1989.*

[74] M. H. Safanov, *Focusing on the knowable: Controller invalidation and learning.* Springer, 1997.

[75] S. Skogestad and I. Postlethwaite, *Multivariable Feedback Control: Analysis and Design.* John Wiley & Sons, 2005.

[76] S. Srikantaiah, M. Kandemir, and Q. Wang, "SHARP control: Controlled shared cache management in chip multiprocessors," in *MICRO, 2009.*

[77] J. Stuecheli, D. Kaseridis, D. Daly, H. C. Hunter, and L. K. John, "The virtual write queue: Coordinating dram and last-level cache policies," in *ISCA, 2010.*

[78] B. Su, J. Gu, L. Shen, W. Huang, J. L. Greathouse, and Z. Wang, "PPEP: Online Performance, Power, and Energy Prediction Framework and DVFS Space Exploration," in *MICRO, 2014.*

[79] L. Subramanian, V. Seshadri, Y. Kim, B. Jaiyen, and O. Mutlu, "Mise: Providing performance predictability and improving fairness in shared main memory systems," in *HPCA, 2013.*

[80] L. Subramanian, V. Seshadri, A. Ghosh, S. Khan, and O. Mutlu, "The Application Slowdown Model: Quantifying and Controlling the Impact of Inter-application Interference at Shared Caches and Main Memory," in *MICRO, 2015.*

[81] X. Sui, A. Lenharth, D. S. Fussell, and K. Pingali, "Proactive Control of Approximate Programs," in *ASPLOS, 2016.*

[82] P. Tembey, A. Gavrilovska, and K. Schwan, "A Case for Coordinated Resource Management in Heterogeneous Multicore Platforms," in *ISCA, 2012.*

[83] R. Teodorescu and J. Torrellas, "Variation-Aware Application Scheduling and Power Management for Chip Multiprocessors," in *ISCA, 2008.*

[84] J. Thistle, "Supervisory control of discrete event systems," in *Mathematical and Computer Modelling, 1996.*

[85] V. Vardhan, W. Yuan, A. F. Harris, S. V. Adve, R. Kravets, K. Nahrstedt, D. Sachs, and D. Jones, "GRACE-2: integrating fine-grained application adaptation with

global adaptation for saving energy," in *IJES, 2009*.
[86] A. Vega, A. Buyuktosunoglu, H. Hanson, P. Bose, and S. Ramani, "Crank It Up or Dial It Down: Coordinated Multiprocessor Frequency and Folding Control," in *ISCA, 2013*.
[87] X. Wang and J. F. Martínez, "ReBudget: Trading Off Efficiency vs. Fairness in Market-Based Multicore Resource Allocation via Runtime Budget Reassignment," in *ASPLOS, 2016*.
[88] X. Wang, K. Ma, and Y. Wang, "Adaptive Power Control with Online Model Estimation for Chip Multiprocessors," in *TPDS, 2011*.
[89] Y. Wang, K. Ma, and X. Wang, "Temperature-constrained Power Control for Chip Multiprocessors with Online Model Estimation," in *ISCA, 2009*.
[90] Q. Wu, Q. Deng, L. Ganesh, C.-H. Hsu, Y. Jin, S. Kumar, B. Li, J. Meza, and Y. J. Song, "Dynamo: Facebook's Data Center-Wide Power Management System," in *ISCA, 2016*.
[91] Q. Wu, P. Juang, M. Martonosi, L.-S. Peh, and D. W. Clark, "Formal control techniques for power-performance management, 2005," in *IEEE Micro, 2005*.
[92] K. Yan, X. Zhang, J. Tan, and X. Fu, "Redefining QoS and customizing the power management policy to satisfy individual mobile users," in *MICRO, 2016*.
[93] H. Zhang and H. Hoffmann, "Maximizing Performance Under a Power Cap: A Comparison of Hardware, Software, and Hybrid Techniques," in *ASPLOS, 2016*.