Self-Aware Power Management for Multi-core Microprocessors

Sai Manoj Pudukotai Dinakarrao[a]

[a]4400 University Drive, Fairfax, VA 22030 USA
Email: spudukot@gmu.edu

ARTICLE INFO

Keywords:
Self-awareness
Power management
Dynamic voltage and frequency scaling
Multi-core microprocessors

ABSTRACT

Power management is one of the significant challenges to be addressed in multi-/many-core microprocessors. Furthermore, the multi-core microprocessors experience unforeseen scenarios such as performance degradation over time, manufacturing defects, power, and thermal impacts with time. Traditional power management techniques, though efficient, is not designed to handle such unseen scenarios. Furthermore, the variation in performance requirements is one of the challenges faced in the era of machine learning. We propose a self-aware power management scheme for multi-core microprocessors in this work to address the above-mentioned issues. We perform application-level power management in this work to overcome the overheads imposed by core-level power management and system-level power management inefficiency. The power management unit employs a linear predictor for workload prediction to perform DVFS. On top of the power manager, the self-aware controller is hierarchically placed to monitor the system components' health and adapt the power manager's decision to meet the performance requirements and handle changes in system components' health. We evaluate the proposed self-aware power manager under externally provided high performance goals, and resource contention. A power saving of up to 16% compared to existing power management techniques, and 2.4× speedup with 25% additional power to satisfy high performance compared to power management without self-awareness for a microprocessor with up to 32-cores is achieved.

1. Introduction

In the era of machine learning and big-data processing, multi-core processors are one of the basic building blocks in the commercial as well as consumer electronic systems ranging from portable mobile phones to data-centers [30, 25]. Multi-core processors facilitate the processing of multi-threaded applications to enact better performance. Intrinsically, the multi-core processors encounter difficulties to meet the constrained resource demands such as power budget, especially under unpredictable runtime scenarios of varying workloads and performance requirements for different concurrently executing applications. This calls for an efficient resource management in multi-core microprocessors.

In addition to resource management, with the scaling down of VLSI technology nodes, multi-core systems face other challenges such as manufacturing defects, variation in system components' health (i.e., functional status due to aging, wear-out and so on), and variations in performance requirements with time [11, 48, 3, 2]. Here, the term 'system component' refers to units such as cores, memory units, voltage-frequency regulators, and sensors. As the components' health (functional status) or performance requirements often vary over time, design-time information is often not sufficient or accurate to tackle such unforeseen influences and adapt [1]. To counteract such unforeseen scenarios, higher-level of abstraction, learning [1] and awareness [11] are required.

Though, there exist multiple resources in multi-core processors, we consider 'power' as one of the pivotal resources that needs efficient management. The rationale for selecting power is that, meeting the power budget is one of the primary challenges faced in multi-core and many-core systems [21, 35]. For power management, dynamic voltage and frequency scaling (DVFS) [33, 41, 32, 52, 44, 49, 14, 29] is one of the proven effective techniques with adaptivity and power savings. DVFS refers to scaling down of voltage and frequency levels for under-utilized cores, thereby reducing the overall dynamic power. The workload characteristics such as temperature, power consumption, temperature, memory-access, and instructions- per-cycle (IPC) are considered for DVFS [33, 41, 52, 44]. The workload characteristic utilized for power management in this work is the power trace, as it reflects the overall power consumption. Performing DVFS at centralized level (system-level [10, 40]) is energy inefficient and hinders the power management efficiency for future systems with a plethora of cores. Similarly, DVFS at the lowest granular level (say per-core level [7, 8]) is highly energy efficient, but adds implementation overheads and design problems [15]. We perform an application-level DVFS in this work to overcome the implementation and design overheads of per-core level power management, and inefficiency by system-level power management.

The existing power management works though efficient, often consider the design-time goals, workload characteristics and very few techniques consider the information regarding the system components' health. They ignore the externally provided or modified goal information due to the system state changes. In a many-core multi-tenant systems, the external factors such as priority of other users or resource contentions highly influence the system state and needs to be considered during runtime. As a consequence of the above mentioned unforeseen challenges, the design-time goals of the system cannot address runtime modifications in the system. This leads to power management without considering the runtime information on goals or system state, re-

ORCID(s): 0000-0002-4417-2387 (S.M.P. Dinakarrao) sulting in an ineffective power management. As indicated in [11], the future systems are expected to operate with dynamically changing goals. This calls for a power management technique which adapts to the dynamically changing goals. *The dynamically changing goals is defined as follows: A modification in the existing goal (objective) or a selection among several goals that a power manager has to meet due to external input (goals) or change in the system's state during runtime.* We emphasize that the dynamic goals in this work does not correspond to the adaptation to the variations in the internal workload characteristics, as adapting to the workload variations is already considered by the underlying (traditional) power manager. Adaptation to dynamic goal is more towards selecting the most appropriate goal (such as power savings, high-performance, less utilization of a specific resource due its unavailability or contention) and adaptation to the changes in the system state or externally provided information during runtime in addition to the workload characteristics.

To meet the aforementioned demands, we propose a confluence of self-awareness and power management techniques in this work. Self-awareness is defined as the ability of a system to be aware of its own state as well as the state of its surrounding environment to adapt to variations [11, 23]. To achieve this, we propose integrating a self-aware controller with a power manager in multi-core system to monitor the components' health (state), performance requirements, the workload characteristics and utilize them to perform power management and meet the performance goals by adapting to runtime changes or requirements. Here, the workload characteristics refer to power trace, memory-access, temperature profile, and so on when executing an application. The information regarding the components' health can be derived based on the models or monitored through sensors.

The proposed self-aware power manager performs per-application DVFS in a multi-core microprocessor considering the workload characteristics, system components' health and the performance requirements. The workload characteristics and system components' health are obtained through the application log and the sensors, respectively. Further, to evaluate the role of self-awareness under dynamic conditions, we consider multiple scenarios: a) component's malfunction (degraded health) and resource contention and b) under externally provided dynamic goal(s). It has been observed that based on the health of the components, the self-aware power manager (SAPM) modifies the power management policy predicted by underlying (traditional) power manager. Similarly, to meet dynamically changing performance requirements, the power management policy chosen by the self-aware power manager is also seen to be impacted. The proposed self-aware controller is placed hierarchically on top of the (traditional) power manager to modify the policy to adapt to the unforeseen conditions such as variation in components' health, and the dynamic system goals (performance requirements).

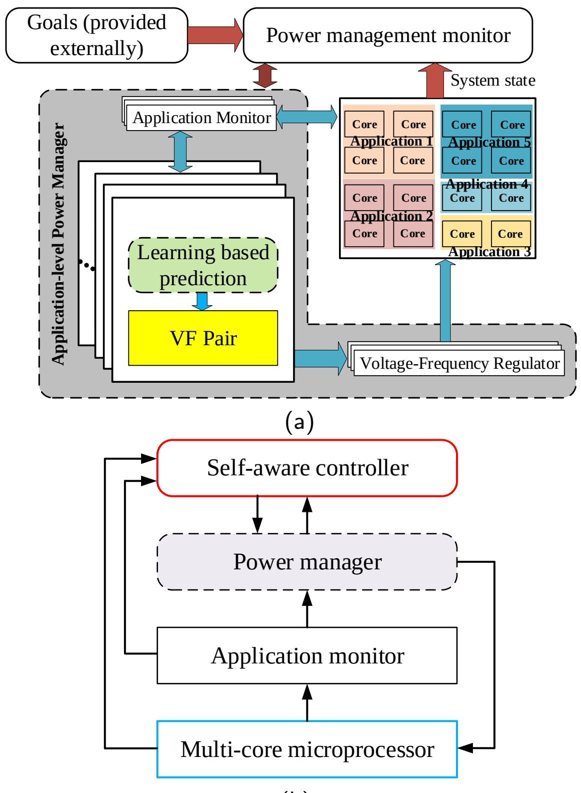

Figure 1: System architecture of multi-core microprocessor with adaptive power manager

Objectives and Contributions

In this work, we introduce a two-level self-aware adaptive power manager (power management monitor and application level power manager) that meets the design-time goals as well as adapt to the dynamically changing goals, as shown in Figure 1. At the design-time, power manager is equipped with per-application dynamic voltage and frequency scaling (DVFS) [41] for power management. A per-application level power management alleviates the associated overheads and inaccuracies caused by per-core and system-level power management schemes [7, 10]. Application-level DVFS is performed by predicting the workload characteristics (power consumption) with online learning. The power management monitor i.e., self-aware controller (highest-level in the proposed adaptive power management technique) triggers the verification of the system's state and modification in the power management methodology when an externally provided goal (performance requirement) or change in the system's state during runtime is detected. This is explained as follows: when there is any high-priority goal provided externally or change in system's state is detected, the power management monitor first tries to meet the externally provided high priority goal or modify the power management policy predicted by the traditional power manager to ensure that the current system's state is suitable to perform the power management efficiently with existing resources that are in good state (health). The major contribution of this work is the design of an adaptive self-aware power manager that not only performs power management, but also adapts its goals and power management policy depending on the state of the system or any externally provided goals. The objective and the contributions of this work are outlined below.

- **Objective:** To perform power management and adapt the power management methodology to the dynamically changing goals arising due to various consequences such as change in system's state, and externally provided goals.
- **Contribution:** A two-level power manager with DVFS performed at per-application level and the (self-aware) power management monitor to monitor the system's state and externally provided goals. The power management policy obtained by employing traditional DVFS is modified by the self-aware power manager in case of external goals or change in system's state so as to adapt to the dynamically changing goals.
- The proposed self-aware power manager is adaptive in terms of specifying goals and adapting the power management policy depending on the components' health and the externally provided performance requirements.

At a very high-level, the difference between the existing power managers and the proposed adaptive power manager is explained with an example. Consider a scenario where a multi-core system is running an application and provided with high-performance requirement during runtime. The traditional power manager irrespective of externally provided requirement, performs power management and achieves a power-performance trade-off. However, the proposed adaptive self-aware power manager considers the externally provided goals and prioritizes the high-performance over the power saving and adapts the working accordingly. Similar behavior is expected in the case of change in system components' health i.e., if a component is malfunctioning, the power management policy is modified to ensure that the defective components are isolated and not allocated a task. An in-depth distinction between proposed and existing works is presented in Section 4.3.

The remainder of this paper is structured as follows. Existing works on power management and self-awareness is outlined in Section 2. The architecture of the proposed self-aware power management system is illustrated in Section 3. Section 4 describes the proposed self-aware power management methodology. Section 5 discusses the evaluation of our approach under different goals such as power management, switching activity reduction and multi-objective optimization. The conclusions are drawn in Section 6.

2. State-of-the-Art

We outline some of the related power management techniques and self-aware frameworks here.

Recent power management techniques utilize learning and predicting the workloads with a *centralized predictor* to perform DVFS. Linear regression [44, 38] is one of the widely used techniques to predict the workloads, which fits a polynomial to the measured or observed values and extrapolate the future values. A linear regression is adopted in [38] to estimate the workload characteristics and perform DVFS. A gradient descent method based updating frequency by learning the workload using linear regression considering the observations from the performance counters, power sensors and measured latencies is proposed in [52]. A space-time multiplexing (STM) using less number of power converters is proposed for power management with auto-regressive moving average (ARIMA) for predicting the workloads and a singular value decomposition for clustering and voltage-frequency level assignment is proposed in [32]. In [9], linear regression is utilized to predict the memory accesses per cycle and CPU cycles per instruction (CPI). Based on the ratio of the predicted CPI with on-chip access and overall CPI, frequency scaling is performed for power management. It needs to be noted that the complexity and achieved accuracy varies with the applied regression technique and the application. In addition to the linear regression or simple predictors [27] and game theory-based solutions [43], machine learning based predictors such as Bayesian predictors [46, 20], Q-learning [31, 51, 50] are also employed to predict the workload characteristics and perform power management [28, 45]. A Bayesian workload predictor with classification and policy generation is proposed in [20] to perform power management. It uses dynamic programming with cost function as the objective for power management. A model-free reinforcement learning for dynamic power management with a Bayesian predictor for workload estimation is proposed in [46]. Based on the predicted workload and using reinforcement learning, power management is carried out. In a similar manner, a Q-learning based approach is proposed that considers the clock frequency, CPU utilization rate for the current task as states, and the tunes the frequency and voltage as actions [19]. Deep learning based workload prediction in the context of adaptive power scaling is proposed in [44]. Recent work [27] performs online learning based power management technique by considering the workload characteristics. It is clearly evident that regression or ML technique in one form or other is widely considered to estimate the workloads to perform power management.

The shortcomings of the existing works can be outlined as follows. In addition to complexity and performance trade-offs, often the power management is performed under the assumption that that the system is deployed under an ideal environment with no malfunctioning of the system components, which is not true in practical scenarios. Additionally, the objective or goal to achieve in most of the existing works are determined during design time and is fixed.

To overcome the shortcomings, self-awareness in the context of power and resource management is introduced. Some of the systematic and notable approaches proposed in projects are SEEC [17], HAMSoC [13], and CPSoC [39].

SEEC frame work has been applied for power management [18], and managing of multiple objectives [16], in which the desired goals and the current system state is mentioned as heartbeat rate of heartbeat APIs [16]. The monitoring-deciding-execution loop is thoroughly elaborated in SEEC while learning, and history mechanisms are not emphasized or not used at all. Learning and history information aids in combating unforeseen scenarios.

In HAMSoC [13], a hierarchical agent based monitoring is employed with self-awareness. As a demonstrative example of self-aware framework, a power management in a multi-core SoC is carried out in [13]. Here, the performance and power attributes are monitored and controlled to optimize power. The key advantage of this work is that the framework is fairly general and multiple system properties could be monitored.

Similarly, a cyber-physical SoC (CPSoC) equipped with sensors for monitoring the system state, self-awareness with the features of communication between different layers of system stack is proposed in [39]. The adaptation in CPSoC happens within or across the layers. The cooperative and hierarchical loops are used to translate user goals into system goals i.e., combination of traditional and virtual sensing enabled self-aware adaptation loops. The self-awareness models in CPSoC did not consider malicious attacks, functional design errors and non-functional aberrations, and show a lot of room for growth in its self-awareness capabilities.

The existing frameworks are undoubtedly efficient and considers the system state to perform the power management. However, if one can observe the existing frameworks, they are built in a hierarchical manner with complex control loops throughout the stack, leading to solving an NP-hard problem. Despite of its own merits, it nearly requires modification or redesign of the whole system stack, additional communication links and protocols. However, neither goal management, nor history or learning mechanisms have been explored.

The primary similarities and differences compared to the existing self-aware frameworks are:

- Similar to the existing self-aware frameworks, proposed self-aware controller considers and monitors the health of the system components to perform power management, DVFS and meet the system requirements.
- Compared to the existing self-aware frameworks, the proposed self-aware power manager does not need any special APIs, and performs learning of the workload characteristics (power trace) to process.
- The proposed self-aware power manager makes use of lightweight predictor to mitigate additional overheads.
- In comparison to other power management works, the proposed work utilizes self-awareness to combat and adapt to unforeseen physical faults or failures.

In this work, we propose a self-aware power manager where the self-aware controller is separated and placed on top of the power management block to provide the similar functional features as proposed in other self-aware frameworks. The advantages can be listed as follows: modular design makes it less complex, no need to modify the system stack, tested and debug well, self-aware block could be embedded with existing architectures with little or no modifications.

3. System Architecture

3.1. System Architecture

The architecture of the proposed self-aware power manager for a multi-core microprocessor is depicted in Figure 1. The system can run single- or multi-threaded applications with each thread on one core. The proposed self-aware power manager has two-levels: application-level power manager and power management monitor (also referred as self-aware monitor). The application-level power manager comprises of an application monitor unit, learning-based application workload predictor followed by a voltage-frequency pair (VF pair) recommending unit. The *Application monitor* unit observes the workload characteristics at per-application granularity. Per-application granularity circumvents the overhead concerns. The *learning-based predictor* learns and predicts the workload characteristics. Based on the predicted workload characteristics, the *VF pair recommender* decides the appropriate VF levels. The workload indicates the characteristics of the application running on a core. As aforementioned, we consider power trace as the workload characteristic for an application. The implementation of the proposed self-aware power manager on top of traditional power manager is depicted in Figure 1(b).

The power management monitor i.e., self-aware monitor is placed hierarchically on top of the application-level power manager. The *power management monitor* monitors the system status, and the goals added externally during runtime. In the event of provision of external goals, the power management monitor prioritizes the goals and alters the power management policy in order to meet the new goals. Similarly, when the state of the system components change, the goals are modified and provided to the application-level power manager. As such, in case of change in the goals, the adaptive power manager reformulates the objectives, constraints and performs the power management accordingly. The system components are the components in the hardware layer such as sensors, voltage-frequency regulators (in this case), and PLLs. The following assumption is made regarding the system: for the sake of experimentation, we assign a binary value for the system components' health i.e., working or malfunctioning. However, one can utilize mathematical models and replace the utilized binary health model, depending on the requirement.

3.2. System Model

3.2.1. Application Model

We denote the application set as $A = \{a_1, a_2, \cdots, a_m\}$. Here a_i represents i-th application. Figure 1(a) represents a snapshot of multi-core system with multiple applications deployed. An application can be single- or multi-threaded, and each core executes one thread at a time. From the application perspective, we assume that the applications are multi-threaded without data dependencies i.e., can be executed in parallel, similar to [33, 36, 37]. Also, at any given time, the total number of executed threads are smaller or equivalent to number of cores. In Figure 1, different shades on processor cores represent different applications running on them. The distribution of applications is not uniform i.e., different applications can run on different number of cores, depending on the number of threads.

3.2.2. Hardware and Power Model

We focus on the components of the system in the context of self-awareness. The components we are concerned in this work are the components in the hardware layer such as sensors, actuators (Phase-Locked Loops (PLLs) and voltage regulators). Each of these components can be internal or external components of a given computing core, with only cores shown in Figure 1 for the purpose of conciseness. The components are represented as $C = \{c_1, c_2, ..., c_n\}$. In this work, the health of component c_i is represented by $h_i \in \{0, 1\}$ with 0 representing malfunction or not functioning and 1 represents functioning well. The set of components used for power management is $\mathcal{C}, \mathcal{C} \subseteq C$. Design time goals (such as standby mode when no application is running) are denoted as D and runtime goals as $\mathcal{R}$. The runtime goals (such as providing best performance depending on the application type) can have higher priority than design time goals. It needs to be noted that the health of the components can also be represented as a function of time, provided exact model exists, which is out of scope of this work.

3.2.3. Power Model

The total power consumption of a core comprises of static and dynamic power. The dynamic power consumption of an application running at a frequency f is modeled as $\frac{1}{2}\mathbf{C}V^2f$. The static power is dominantly due to leakage power and varies exponentially with threshold voltage. The dynamic power consumption is due to the application dependent switching activities in the core. The total power consumption [5, 12] when operating at voltage V and frequency f is modeled as below:

$$P(V, f) = P_{static} + P_{Dynamic} = I_0 e^{\frac{-V_{th}}{\eta V_T}} V + \alpha \mathbf{C} V^2 f \quad (1)$$

Here I_0 and η are technology parameters; V_T is the thermal voltage; V_{th} is the threshold voltage; α represents the switching activity factors and C is the average capacitance. To obtain per-application power or energy trace, we sum the power traces of the cores on which the application is executing.

3.3. Problem Statement

In the multi-core and many-core systems, the resource constrained embedded systems have to be intelligent enough to monitor their functionality and utilization, and service the goals to ensure efficient processing and resource utilization. Furthermore, as the systems have also goals provided during runtime or change of system state (health) is quite feasible, adapting to the goals at runtime is needed for an efficient power utilization and system performance. Thus, the problem is formulated as follows.

Problem Statement: For an effective power management, the power manager has to consider the runtime system state, workload characteristics and external goals (if any), and adapt itself to the runtime conditions and dynamically changing goals such as high-performance requirements or resource contention.

4. Self-aware Power Management

In this section, we describe the functionality of self-aware power manager. First, we present the application-level power manager, followed by the power management monitor using two different scenarios: a) in case of externally provided goals, and b) when the state of components' change.

4.1. Application-level Power Manager

Algorithm 1 Utilized Application-level Power Management ($PM()$)

Require: Workloads ($W = \{W_1, W_2, ...\}$), goals $\mathcal{G}$, constraints $\mathcal{C}$, Components (Voltage-Frequency (VF) levels) ($C = \{c_1, c_2, ..., c_q\}$)
Ensure: Power management policy (R_{policy}) and used components (C_{used})

```
1:  Workload for application $i$: $W_i = \{x_i(1), x_i(2), ...x_i(n)\}$;
2:  for $j = p$ to $n$ do                                  ▷ Perform prediction
3:      $\widehat{x_i(j)} = \sum_{k=1}^{p} b_k * x_i(j-k)$;
4:      $\sum_i^p b_i R(j-i) = -R(j)$;                      ▷ Find $a_i$
5:      for $k = 1$ to $q$ do
6:          $d(\widehat{x_i(j)}, s_k) = (\widehat{x_i(j)} - s_k)^2$;   ▷ $s_k$ corresponds to power provided with component $c_k$
7:          if $d(\widehat{x_i(j)}, s_k) = min$ then
8:              $W_i(j) \leftarrow c_k$;
9:          else
10:             $k = k + 1$;
11:         end if
12:     end for
13: end for
```

The power management policy generation by the application-level power manager (i.e., function $PM()$) is as follows. Here, the power management policy (R_{policy}) refers to the DVFS settings. For power management, we employ learning based DVFS. The DVFS in this work is performed with the predictor, followed by mapping the predicted power values to the voltage and frequency. It needs to be noted that the application-level power management technique is utilized to overcome the overhead concerns. However, this power management technique can be replaced with other techniques as well [34]. The application-level power manager is programmed at OS-level and is an independent module compared to the power management monitor unit. As such, the employed power management technique is replaceable with other variants.

The prediction of workload is performed with linear regression, whose weights are updated based on the prediction error. This is performed in the application-level power manager (Figure 1).

$$\widehat{x_i(n)} = \sum_{j=1}^{p} b_j x_i(n-j) + \gamma$$
$$\sum_{i=1}^{p} b_i R(j-i) = -R(j) \tag{2}$$

Here, b_j, $j = 1, 2, ..., p$ are the coefficients, p representing the order of prediction i.e., number of previous samples considered for the prediction; $x_i(n)$ represents the workload (power trace) at the time-instant n for application i; γ is the prediction error; and the predicted value is indicated as $\widehat{x_i(n)}$. The $R(j)$ is the autocorrelation for the signal. The application-level power trace is obtained by summing up the power consumption by individual threads of the application, if running on multiple cores.

Further, based on the demanded power, and the power that can be provided with the existing VF settings of the voltage frequency regulator. The mapping of predicted power and VF levels is performed based on the closest distance to the formed clusters (center of each cluster represent the amount of power that could be provided when using the VF settings with which cluster is associated), as in Line 5-11 of Algorithm 1. In Algorithm 1, the s_k denotes the power provided the component c_k whose voltage and frequency levels are (v_k, f_k). If the performance requirements are not met, the frequency is scaled up accordingly, though not presented in Algorithm 1.

4.2. Self-aware Power Management

The proposed self-aware power manager follows the basic principle of observe-decide-act. However, during the decide phase, it also performs the analysis of observed data. We explain the principle using two different scenarios.

4.2.1. Case 1: Externally Provided Runtime Goals

The working principle of the proposed adaptive power manager that adapts to the dynamically changing runtime goals is described in two phases.

Observe: Monitor Dynamically Changing Goals

The *power management monitor* keeps track of (externally) provided runtime goals. For instance, in this work, the runtime goals are provided into the system through a C program. Initially, the system is provided with a header file where the primary requirements such as performance are provided, and the newly provided file overwrites the values, leading to a change in performance requirement during runtime. In similar manner, other goals are introduced into the system. In addition, the power management monitor also looks for the changes in the system state (as in Line 1 of Algorithm 2). As the systems (microprocessors) are deployed in large networks or data centers, the resources are shared between different systems [26]. As such, in this work, the system state information considered is the resource contention. Whenever a resource is contended, the power management monitor unit levies the constraint of not or minimally employing the contended resource for power management or to the existing goal. The availability of i-th component is indicated by variable h_i, $h_i \in \{0, 1\}$, where 1 indicates that the component is available. The state of components is indicated by $H = \{h_1, h_2, ..., h_k\}$, where k indicates the components available for the current system. However, this work is not limited for resource contention and can be extended with other state information changes such as malfunctioning of a component, and aging with time, which will also result in similar modifications in the goal and constraints of the power manager. For this case study, we assume that all the components are in working state.

The per-application power manager has the application monitor unit that tracks the applications' workload characteristics W, ($W = \{W_1, W_2, ...\}$ where W_i represents the workload characteristics for application i) such as power trace, memory access and so forth (as in Line 1 of Algorithm 2). In this work, power trace of an application is the workload characteristic considered for power management.

Algorithm 2 Self-aware Power Management in a Multi-core Microprocessor under Runtime Provided Goals

```
Require: Design-time goals $\mathcal{D}$, design-time constraints $C_\mathcal{D}$, modified design- or runtime goals $\mathcal{R}$, modified constraints $C_\mathcal{R}$, Applications ($W = \{W_1, W_2, ...\}$), system components $C = \{c_1, c_2, ..., c_k\}$
Ensure: Meet system goals (design-time and runtime) even under dynamically changing goals
 1: Obtain $H = \{h_1, h_2, ..., h_k\}$ for $C$    ▷ Monitor the status of system components
 2: Monitor characteristics for application $i$: $W_i = \{x_i(1), x_i(2), ...x_i(n)\}$;
 3: if $\mathcal{R} == \{\emptyset\}$ then ▷ When there is no modification in goals or change in system state
 4:     $[R_{policy}, C_{used}] \leftarrow PM(W, \mathcal{D}, C_\mathcal{D}, C)$    ▷ Generate power management policy, as in Algorithm 1
 5: else if $\mathcal{R} \neq \{\emptyset\} \&\& P_r(\mathcal{R}) > P_r(\mathcal{D})$ then    ▷ If externally specified runtime goals have higher priority
 6:     $[R_{policy}, C_{used}] \leftarrow PM(W, \mathcal{R}, C_\mathcal{R}, C)$    ▷ Generate the policy to meet the runtime goals
 7: else if $\mathcal{R} \neq \{\emptyset\} \&\& P_r(\mathcal{R}) \leq P_r(\mathcal{D})$ then    ▷ If runtime goals are externally specified
 8:     $[R_{policy}, C_{used}] \leftarrow PM(W, \mathcal{D} \cup \mathcal{R}, C_\mathcal{D} \cup C_\mathcal{R}, C)$    ▷ Merge the two goals and generate the corresponding policy that meets the goals
 9: end if
10: $enforce(R_{policy})$    ▷ Enforce the recommended policy
```

Decide-and-Act: Adaptation to Changing Goals

The adaptation to dynamically changing goals and system state with the proposed adaptive power manager is as follows. If there exists no change in system state or no goals externally i.e., power management monitor unit does not observe any changes, the system performs the application-level power management described in Algorithm 1 (as in Line 3-4 of Algorithm 2). In case of externally provided goals with higher priority, the application monitor unit replaces the design-time goal with the external goal and performs the power management according to the new goal. In case of equal priority, the externally provided goals are integrated with the existing design-time goals and power management is carried out by the application-level power manager (as in Line 5-8 of Algorithm 2). For lower priority, the execution could be delayed (though not performed in this work). In case of change in system state, the constraints and goals are modified by the power management monitor unit as described previously and the application-level power management is carried out. In case of orthogonal goals that conflict each other such as high performance and low battery utilization (say), techniques like multi-objective optimization is employed, though not presented here for brevity.

As the externally provided goal (requirement) could be anything such as high-performance, we denote with a more generic term $\mathcal{R}$. In this work, a demonstration with high-performance is performance as external goal is provided, as such $PM(W, \mathcal{R}, C_{\mathcal{R}}, C)$ corresponds to maximizing the throughput through frequency scaling. As provided in the comments of Line 11, $enforce()$ represents running the determined policy (R_{policy}) i.e., execute the application with the determined VF levels. The utilized per-application level power management ($PM()$) is detailed in Algorithm 1.

4.2.2. *Case 2: Self-awareness when a Component Fails*

The proposed self-aware power manager follows an observe-decide-act cycle for resource management even when a component fails, as described below.

Observe the Health of System

In a traditional system with a power manager, the basis to perform the power management is the application's workload characteristics W, ($W = \{W_1, W_2, ...\}$ where W_i represents the workload characteristics for application i) such as power trace, memory access and so forth. Whereas, the self-aware system with power manager monitors the health of the system components H along with the applications' workload characteristics W. The health of the system components ($H = \{h_1, h_2, ..., h_k\}$ for k components) is useful to asses the system condition (self-assessment) and to adapt the variations in the component's health. This lays the basis for self-awareness, given in Line 1-2 of Algorithm 3. The health of the voltage regulators and the PLLs are considered in this work. The self-aware controller observes the policy recommended by the power management unit. The policy (R_{policy}) denotes the DVFS settings in our demonstrative case study.

Algorithm 3 Self-Aware Power Management

```
Require: Design time goals $\mathcal{D}$, design time constraints $C_{\mathcal{D}}$, explicit runtime goals $\mathcal{R}$, runtime constraints $C_{\mathcal{R}}$, Applications ($W = \{W_1, W_2, ...\}$), system components $C = \{c_1, c_2, ..., c_k\}$
Ensure: Meet system goals (design time and runtime) even under unforeseen scenarios
 1: Obtain $H = \{h_1, h_2, ..., h_k\}$ for $C$    ▷ Monitor the health of system components
 2: Monitor application $i$ characteristics: $W_i = \{x_i(1), x_i(2), ...x_i(n)\}$;
 3: if $\mathcal{R} == \{\emptyset\}$ then    ▷ When there is no runtime goals
 4:     $[R_{policy}, C_{used}] \leftarrow PM(W, \mathcal{D}, C_{\mathcal{D}}, C)$    ▷ Generate power management policy
 5: else if $\mathcal{R} \neq \{\emptyset\} \&\& P_r(\mathcal{R}) > P_r(\mathcal{D})$ then    ▷ Runtime specified goals have higher priority
 6:     $[R_{policy}, C_{used}] \leftarrow PM(W, \mathcal{R}, C_{\mathcal{R}}, C)$    ▷ Generate the policy to meet the runtime goals
 7: else if $\mathcal{R} \neq \{\emptyset\} \&\& P_r(\mathcal{R}) \leq P_r(\mathcal{D})$ then    ▷ If runtime and design time goals have same priority
 8:     $[R_{policy}, C_{used}] \leftarrow PM(W, \mathcal{D} \cup \mathcal{R}, C_{\mathcal{D}} \cup C_{\mathcal{R}}, C)$    ▷ Merge the two goals and generate the corresponding policy
 9: end if
10: if $h_i == 1 \ \forall \ c_i \in C_{used}$ then    ▷ If all the utilized components are healthy
11:     $enforce(R_{policy})$ ▷ Enforce the recommended policy
12: else
13:     $C = C - \{c_a\}$, $where \ h_a = 0 \ for \ c_a$    ▷ When the needed components are faulty
14:     Go to Step 1    ▷ Regenerate the policy without considering the faulty components
15: end if
```

The policy generation is performed in the OS layer and enforced in hardware layer, in this work.

Decide and Act: Adapt to changed System's State

In the self-aware system the 'decide' has been used for two different purposes, as explained below. The self-aware unit monitors for the explicitly specified runtime goals, referred as runtime goals. If there are no runtime goals, the power manager (here) generates the power management policy (described in Line 3-4 of Algorithm 3). When the runtime goals are specified, the self-aware manager looks for the priority of meeting the runtime goals and follows Algorithm 2. If no external goals are provided, the system generates the power management policy, as defined in Algorithm 1. Once the policies are generated (R_{policy}) along with the required components (C_{used}), the self-aware controller validates the health of components required (C_{used}). If the components required are in functional state (healthy), the self-aware controller enforces the decision and executes it, Line 10-11 of Algorithm 3. If the required components are faulty or not available, the self-aware controller lets the

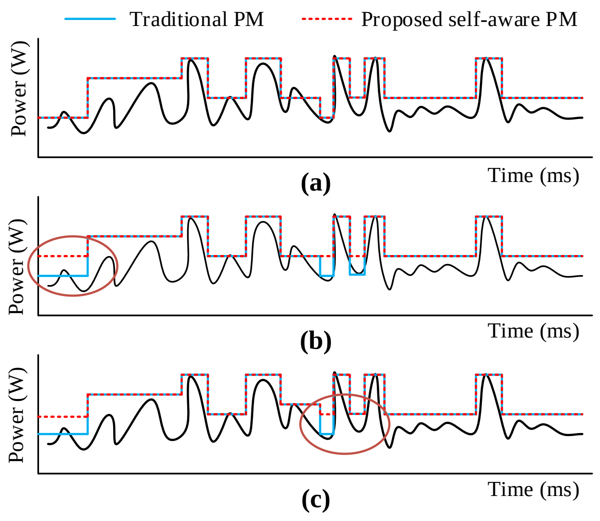

Figure 2: Behavior of traditional power manager (Traditional PM) and proposed self-aware power manager (Proposed self-aware PM) under: (a) normal conditions; (b) high performance requirements; (c) state of component changes

power manager to regenerate the policy without considering the unhealthy components, Line 13-14 of Algorithm 3. As aforementioned, the component's health can be multi-level rather than binary.

4.3. Differences w.r.t. Existing works

The differences compared to some of the existing power management works [33, 41, 52, 32, 44, 11] are:

- The proposed adaptive power manager considers the application workload characteristics and runtime system state information, whereas most of the existing works use the application workload characteristics and few works also use the design-time system state information or estimated models for system state. However, this does not reflect the real system state information at runtime.
- Unlike existing works, the dynamic goals (in this work) are the goals that are externally provided to the system externally or design-time goals adapted due to the change in the system state.
- In contrast to the existing power management works that considers solely workload characteristics for power management, the proposed work considers the external goals as well as internal changes (workload characteristics and system state information).

Figure 2 illustrates the difference between the policies generated by the traditional and proposed self-aware power manager in different situations. When there exist no externally provided goals and state of all the components are functional, the policy from the traditional and proposed self-aware power manager are same, as shown in Figure 2(a). Consider a scenario where a high performance is provided externally (higher than performance constraint provided during design time), than the self-aware power manager overwrites the policy provided by the traditional power manager to meet the performance constraints. In Figure 2(b), if one can observe closely, the lowest VF level is not chosen by the self-aware power manager, instead a higher VF level is chosen in order to meet the performance constraints. In similar fashion, Figure 2(c) depicts the policy from the traditional and self-aware power manager when the state of one of the voltage regulator changes. In this example, the voltage regulator that provides lowest VF settings is either malfunctioning or not available, hence the self-aware power manager chooses second voltage regulator whenever the lowest level is predicted by the traditional power manager, rather the the third regulator which provides lowest voltage-level. It needs to be noted that in this example we consider three voltage levels exist.

Table 1
Overview of System Configuration

Item	Description	Value
Microprocessor core	Frequency (Max)	2.66GHz
	Voltage (Max.)	1.2V
	Technology node	22nm
	L1-I cache	32KB
	L1-D cache	32KB
	L2 cache	256KB
L3-Cache	8MB	

5. Simulation Results

5.1. Simulation Settings

The proposed self-aware power management scheme is implemented in SniperSim multi-core simulator [6]. Nehlam microachitecture based (22nm) core models are used for the simulations. In simulations, we use four voltage-frequency levels for power management, that are supported by standard Nehlam microachitecture based cores: (1.2V, 2.66GHz), (1.1V, 1.8GHz), (1.0V, 1.5GHz) and (0.9V, 1.0GHz). A switching time for the voltage-frequency regulator (10μs) is considered in the simulations [42]. The reported power and timing overhead includes switching power and time. In order to evaluate the self-aware power manager, simulations are run with Parsec [4] and SPLASH-2 [47] benchmarks. The predictor is of order 4, chosen based on accuracy vs complexity analysis.

5.2. Tool Flow

The tool flow to perform self-aware power management in Snipersim multi-core simulator is presented in Figure 3. The Snipersim simulator is initialized with the configurations such as number of cores, microarchitecture, technology node, VF levels and so on. For an unbiased and better evaluation of self-aware power management, applications are assigned to the cores in a random manner. Next, the workload statistics at application level granularity are monitored from the simulator and provided to the machine learning based predictor. Based on the predicted workload(s), the corresponding voltage and frequency levels are assigned to the core(s) that are running the corresponding applications.

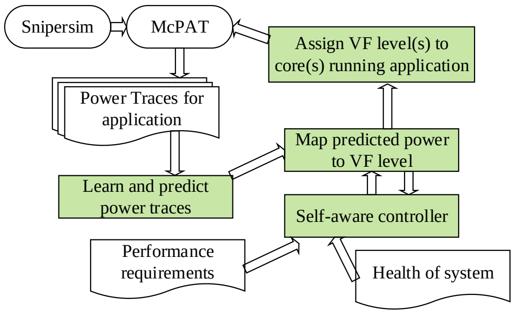
Snipersim
McPAT
Assign VF level(s) to core(s) running application
Power Traces for application
Map predicted power to VF level
Learn and predict power traces
Self-aware controller
Performance requirements
Health of system

Figure 3: Experimental tool flow to perform self-aware power management in SniperSim

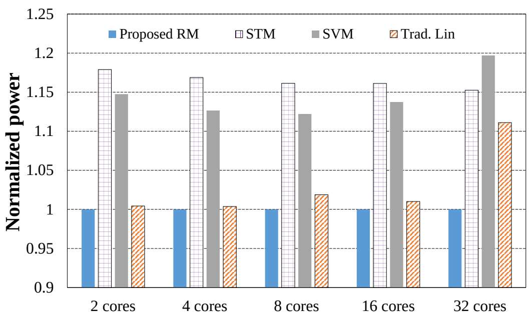

Figure 4: Power savings with the proposed application-level power manager under performance constraints

Lastly, McPAT [24] is used to obtain the power consumption statistics of the microprocessor. To perform non-DVFS, the workloads can be run on similar settings without invoking the DVFS.

To implement the proposed technique in real-world systems, the existing power manager needs to be enhanced and an interface to communicate with the external interrupt signals need to be provisioned. The external goals will be provided to the system in the form of interrupts. The priority of the requests (maskable or unmaskable) depends on the initiating device.

5.3. Evaluation of Self-Aware Controller

In this work, the components' health and performance requirements are provided manually. However, in the real environment, this can be provided with the aid of the sensors and requirements from user. The performance of the self-aware power manager is evaluated under different conditions.

5.3.1. Power Management

For evaluating the effectiveness of application-level power management (in Section 4.1), the system is provided with no external goals and the system state is kept unchanged during runtime. The proposed application-level power management is compared against some of the recent works such as [32, 53, 52]. The power savings is shown in Figure 4. The X-axis represents the number of cores and Y-axis represents normalized power consumption. The application-level power management (denoted as 'Proposed RM') is compared against

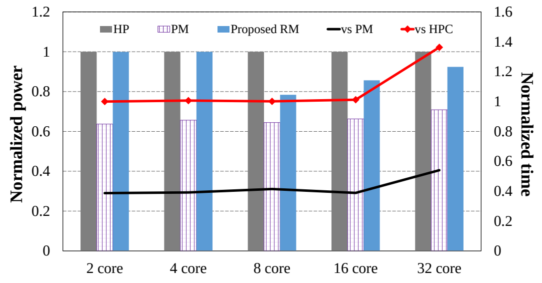

Figure 5: Power savings and runtime with adaptive power manager under externally provided goal (high-performance)

works similar to DVFS with workload prediction ('Trad. Lin') [52], space-time multiplexing based DVFS ('STM') [32], and SVM based workload prediction, and classification DVFS ('SVM') [53].

The rationale for comparing with [32, 53, 52] is as follows. In [32], ARMA based workload prediction and singular value decomposition (SVD) based VF level assignment with space-time multiplexing (STM) is performed, which is similar to the utilized power management. A linear regression with offline learning based workload prediction and VF level assignment is proposed in [52]. In works like [39], linear regression is employed for power management. These works are reproduced with minor modifications for fair comparison.

Traditional linear regression based power management (similar to [52]) performs similar to the utilized application-level power management for small number of cores. However, for the system with large number of cores (such as 16 or 32), the proposed power management outperforms. This is observed due to the variation in the granularity of monitoring.Furthermore, the complexity increases with the increase in number of cores and the accuracy of the model as well decreases. For small cores, the system-level and application-level traces are similar, but has variations with increase in the number of cores. For a 32-core microprocessor, the difference between proposed power management and the linear regression based power management is around 12%. The application-level power manager achieves 16% and 14% higher power savings on average compared to space-time multiplexing and SVM based power management respectively for a microprocessor up to 32-cores. In terms of overhead, the proposed application-level power management has nearly 0.97× runtime compared to the linear regression based power management; and 1.05× runtime compared to STM based implementation with up to 32 cores (not represented in Figure 4).

5.3.2. High-Performance as Externally Provided Goal

To evaluate the adaptation to dynamically changing goals, the system is externally provided during runtime with the requirement to achieve high-performance. The power management monitor unit detects the externally provided goal and modifies the goals and constraints of the power management (which application-level power manager is using) to

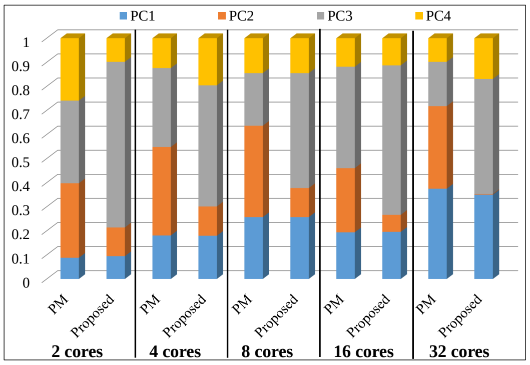

Figure 6: Allocation of tasks to Voltage Frequency regulators

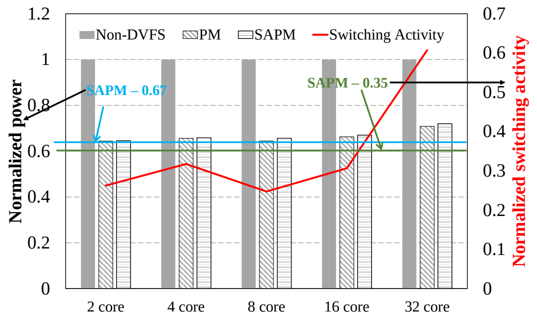

Figure 7: Power savings and switching activity with self-aware power manager (SAPM) under optimized switching activity, high performance and power constraints

the high-performance, as in algorithm 2. This high-performance achievement can be seen in terms of overall runtime of applications.

Figure 5 compares the proposed adaptive power manager, traditional power manager and a system design with objective of high-performance. The performance of systems (power and runtime) with high-performance and low power are represented by 'HP' and 'PM', respectively. The proposed work is denoted by 'Proposed RM'. The bar graph represent the power consumption (normalized) and the lines represent the runtime (normalized). The comparison of runtime of the proposed work with low power and high-performance systems are denoted by 'vs PM' (black line) and 'vs HP' (red line), respectively.

The proposed adaptive manager performs similar to a high-performance system for small number of cores, and tries to optimize the power. For large number of cores (32 cores or so), the system trades the power to performance, and moreover the goal is provided during runtime so part of execution is already performed. Compared to a system with high-performance as design-time goal, the proposed power manager has nearly 9% power saving and 1.07× runtime. Compared to a system with optimization of power as design-time objective, the proposed system has 25% higher power consumption on average, but 2.4× faster on average.

5.3.3. *Resource Contention*

In large scale systems such as data centers, though the number of hardware resources are abundant, reusability and sharing of the resources is widely employed to efficiently utilize the existing resources. In such scenarios, it is highly possible that the resources demanded by one system could be in use by other systems, resulting in resource contention. This kind of resource contention is often bottlenecks in large processing units such as datacenters [22]. To verify the adaptation to dynamically changing goals with respect to the change in system state, the VF regulator 2 (VF 2) is set under contention. As such, the power management monitor unit modifies the constraints to not utilize the VF 2 for the application-level power management. We demonstrate the adaptability with VF regulator as one of the resources, but could be extended to other resources. Figure 6 shows the allocation of the tasks/applications to the VF regulators. Compared to a power manager with power minimization as the objective, the assignment of tasks to VF2 is reduced by nearly 68% with the proposed adaptive power manager. Contrarily, the distribution of tasks on other VF regulators lead to an increase of 3% power consumption. As the goal is introduced during runtime, the resource manager after adaptation shows some utilization of VF 2 in Figure 6.

5.3.4. *Power saving under Reduced Switching Activity and High Performance Constraints*

Lastly, we evaluate the proposed self-aware resource management (represented as SAPM) system under the constraints of lower power consumption with reduced VF regulator switching activity and high performance requirements, which is one of the commonly encountered bottlenecks in the multicore microprocessors. The performance in terms of power savings and switching activity is shown in Figure 7. A power saving of 33% is achieved on average compared to non-DVFS (non-DVFS) and is 1% lower power saving (saves power by reduced switching, but allocation of DVFS levels is improper) than traditional power management (denoted as PM) without self-awareness on average. However, the system has 65% lower switching activity and 4% lower runtime compared to power manager without self-awareness.

6. Conclusions

A self-aware power manager which monitors the health of the system components as well the workloads to perform power management is proposed in this work. The self-aware controller is placed hierarchically on top of the power manager and is independent to other components of the system. This hierarchically built system has the advantages of real-time power management and adaption to system goals at runtime and variations in the components' health. The utilized application-level power manager employs a linear regression based workload prediction for DVFS. The self-aware unit monitors and validates the decision made by the underlying traditional power manager and modifies the decision (power management policy) in order to meet the externally provided goals and/or change in system state. The proposed self-aware power manager is evaluated under various scenarios such as externally provided high performance requirement, change in system state (resource contention of voltage-frequency regulators), and both the previously mentioned objectives. The proposed self-aware power manager achieves up to 16% compared to existing power management techniques; and 2.4× faster to achieve high performance with additional 25% power consumption; and 65% lower switching activity and 4% lower runtime under both resource contention and high performance requirements on average compared to power management without self-awareness for microprocessor up to 32 cores.

References

[1] Bartolini, A., et.al., 2010. A virtual platform environment for exploring power, thermal and reliability management control strategies in high-performance multicores, in: Great Lakes Symp. on VLSI.

[2] Baumann, R.C., 2005. Radiation-induced soft errors in advanced semiconductor technologies. IEEE Transactions on Device and Materials Reliability 5, 305–316.

[3] Bernstein, J.B., et.al., 2006. Electronic circuit reliability modeling. Microelectronics Reliability 46, 1957 – 1979.

[4] Bienia, C., et.al., 2008. The PARSEC benchmark suite: Characterization and architectural implications, in: Int. Conf. on Parallel Architectures and Compilation Techniques.

[5] Brooks, D., Dick, R.P., Joseph, R., Shang, L., 2007. Power, thermal, and reliability modeling in nanometer-scale microprocessors. IEEE Micro 27, 49–62.

[6] Carlson, T.E., et.al., 2014. An evaluation of high-level mechanistic core models. ACM Trans. Archit. Code Optim. 11, 28:1–28:25.

[7] Chen, G., Huang, K., Knoll, A., 2014. Energy optimization for real-time multiprocessor system-on-chip with optimal DVFS and DPM combination. ACM Trans. Embed. Comput. Syst. 13, 111:1–111:21.

[8] Chen, J.J., Thiele, L., 2010. Energy-efficient scheduling on homogeneous multiprocessor platforms, in: ACM Symp. on Applied Computing.

[9] Choi, K., Soma, R., Pedram, M., 2005. Fine-grained dynamic voltage and frequency scaling for precise energy and performance tradeoff based on the ratio of off-chip access to on-chip computation times. IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems 24, 18–28.

[10] Devadas, V., Aydin, H., 2010. Coordinated power management of periodic real-time tasks on chip multiprocessors, in: Int. Conf. on Green Computing.

[11] Dutt, N., Jantsch, A., Sarma, S., 2016. Toward smart embedded systems: A self-aware system-on-chip (SoC) perspective. ACM Trans. Embed. Comput. Syst. 15, 22:1–22:27.

[12] Ejlali, A., Al-Hashimi, B.M., Eles, P., 2012. Low-energy standby-sparing for hard real-time systems. IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems 31, 329–342.

[13] Guang, L., et.al., 2011. HAMSoC: A monitoring-centric design approach for adaptive parallel computing, in: Autonomic Networking-on-Chip : Bio-inspired Specification, Development and Verification.

[14] Hantao, H., Manoj, P.D.S., Xu, D., Yu, H., Hao, Z., 2014. Reinforcement learning based self-adaptive voltage-swing adjustment of 2.5D I/Os for many-core microprocessor and memory communication, in: IEEE/ACM Int. Conf. on Computer-Aided Design (ICCAD).

[15] Herbert, S., Marculescu, D., 2007. Analysis of dynamic voltage/frequency scaling in chip-multiprocessors, in: Int. Symp. on Low Power Electronics and Design.

[16] Hoffmann, H., 2014. CoAdapt: Predictable behavior for accuracy-aware applications running on power-aware systems, in: Euromicro Conf. on Real-Time Systems.

[17] Hoffmann, H., et.al., 2010. Seec: A framework for self-aware computing. Online.

[18] Hoffmann, H., et.al., 2013. A generalized software framework for accurate and efficient management of performance goals, in: ACM Int. Conf. on Embedded Software.

[19] Islam, F., Lin, M., 2015. A framework for learning based dvfs technique selection and frequency scaling for multi-core real-time systems, in: IEEE Int. Conf. on Embedded Software and Systems.

[20] Jung, H., Pedram, M., 2010. Supervised learning based power management for multicore processors. IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems 29, 1395–1408.

[21] Khdr, H., Pagani, S., Sousa, E., Lari, V., Pathania, A., Hannig, F., Shafique, M., Teich, J., Henkel, J., 2017. Power density-aware resource management for heterogeneous tiled multicores. IEEE Transactions on Computers 66, 488–501.

[22] Kim, Y., Sylvester, D., Blaauw, D., 2011. LC2: Limited contention level converter for robust wide-range voltage conversion, in: Symp. on VLSI Circuits.

[23] Lewis, P.R., et.al., 2011. A survey of self-awareness and its application in computing systems, in: IEEE Conf. on Self-Adaptive and Self-Organizing Systems Workshops.

[24] Li, S., et.al, 2009. McPAT: An integrated power, area, and timing modeling framework for multicore and manycore architectures, in: IEEE/ACM Int. Symp. on Microarchitecture.

[25] Lin, J., Zhu, S., Yu, Z., Xu, D., Manoj, P.D.S., Yu, H., 2015. A scalable and reconfigurable 2.5D integrated multicore processor on silicon interposer, in: IEEE Custom Integrated Circuits Conf.

[26] Liu, H., 2011. A measurement study of server utilization in public clouds, in: IEEE Int. Conf. on Dependable, Autonomic and Secure Computing.

[27] Manoj, P.D.S., Jantsch, A., Shafique, M., 2018. SmartDPM: Dynamic power management using machine learning for multi-core microprocessors. Journal of Low-Power Electronics 14.

[28] Manoj, P.D.S., Lin, J., Zhu, S., Yin, Y., Liu, X., Huang, X., Song, C., Zhang, W., Yan, M., Yu, Z., Yu, H., 2017. A scalable network-on-chip microprocessor with 2.5D integrated memory and accelerator. IEEE Transactions on Circuits and Systems I: Regular Papers 64, 1432–1443.

[29] Manoj, P.D.S., Wang, K., Yu, H., 2013. Peak power reduction and workload balancing by space-time multiplexing based demand-supply matching for 3D thousand-core microprocessor, in: ACM/EDAC/IEEE Design Automation Conf.

[30] Manoj, P.D.S., Yu, H., 2013. Cyber-physical management for heterogeneously integrated 3D thousand-core on-chip microprocessor, in: IEEE International Symposium on Circuits and Systems (ISCAS).

[31] Manoj, P.D.S., Yu, H., Huang, H., Xu, D., 2016. A Q-Learning based self-adaptive I/O communication for 2.5D integrated many-core microprocessor and memory. IEEE Trans. on Computers 65, 1185–1196.

[32] Manoj, P.D.S., Yu, H., Wang, K., 2015. 3D many-core microprocessor power management by space-time multiplexing based demand-supply matching. IEEE Trans. Computers 64, 3022–3036.

[33] Pagani, S., et.al., 2017. Energy efficiency for clustered heterogeneous multicores. IEEE Trans. on Parallel and Distributed Systems 28, 1315–1330.

[34] Pagani, S., Manoj, P.D.S., Jantsch, A., Henkel, J., 2018. Machine learning for power, energy, and thermal management on multi-core processors: A survey. IEEE Transactions on Computer Aided Systems of Integrated Circuits and Systems .

[35] Pathania, A., Pagani, S., Shafique, M., Henkel, J., 2015. Power management for mobile games on asymmetric multi-cores, in: IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED).

[36] Pathania, A., Venkataramani, V., Shafique, M., Mitra, T., Henkel, J., 2017. Defragmentation of tasks in many-core architecture. ACM Trans. Archit. Code Optim. 14.

[37] Rapp, M., Sagi, M., Pathania, A., Herkersdorf, A., Henkel, J., 2020. Power- and cache-aware task mapping with dynamic power budgeting for many-cores. IEEE Transactions on Computers 69, 1–13.

[38] Rountree, B., et.al., 2011. Practical performance prediction under dynamic voltage frequency scaling, in: Int. Green Computing Conference and Workshops.

[39] Sarma, S., et.al., 2014. On-chip self-awareness using cyberphysical-systems-on-chip (CPSoC), in: Int. Conf. on Hardware/Software Codesign and System Synthesis.

[40] Seo, E., Jeong, J., Park, S., Lee, J., 2008. Energy efficient scheduling of real-time tasks on multicore processors. IEEE Trans. on Parallel and Distributed Systems 19, 1540–1552.

[41] Shafique, M., Ivanov, A., Vogel, B., Henkel, J., 2016. Scalable power management for on-chip systems with malleable applications. IEEE Trans. on Computers 65, 3398–3412.

[42] Singhal, R., 2008. Inside Intel® core microarchitecture (nehalem), in: IEEE Hot Chips Symp.

[43] Somu Muthukaruppan, T., Pathania, A., Mitra, T., 2014. Price theory based power management for heterogeneous multi-cores, in: International Conference on Architectural Support for Programming Languages and Operating Systems.

[44] Tarsa, S.J., Kumar, A.P., Kung, H.T., 2014. Workload prediction for adaptive power scaling using deep learning, in: IEEE Int. Conf. on IC Design Technology.

[45] Tesauro, G., et.al., 2007. Managing power consumption and performance of computing systems using reinforcement learning, in: Int. Conf. on Neural Information Processing Systems.

[46] Wang, Y., Pedram, M., 2016. Model-free reinforcement learning and bayesian classification in system-level power management. IEEE Trans. on Computers 65, 3713–3726.

[47] Woo, S.C., et.al., 1995. The SPLASH-2 programs: Characterization and methodological considerations. SIGARCH Comput. Archit. News 23, 24–36.

[48] Wu, S.S., Wang, K., Manoj, P.D.S., Ho, T.Y., Yu, M., Yu, H., 2014. A thermal resilient integration of many-core microprocessors and main memory by 2.5D TSI I/Os, in: Design, Automation Test in Europe Conference Exhibition (DATE).

[49] Xu, D., Manoj, P.D.S., Huang, H., Yu, N., Yu, H., 2014. An energy-efficient 2.5D through-silicon interposer I/O with self-adaptive adjustment of output-voltage swing, in: IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED).

[50] Xu, D., Yu, N., Huang, H., Manoj, P.D.S., Yu, H., 2018. Q-learning based voltage-swing tuning and compensation for 2.5D memory-logic integration. IEEE Design and Test 35, 91–99.

[51] Xu, D., Yu, N., Manoj, P.D.S., Wang, K., Yu, H., Yu, M., 2015. A 2.5-D memory-logic integration with data-pattern-aware memory controller. IEEE Design Test 32, 1–10.

[52] Yang, S., et.al., 2015. Adaptive energy minimization of embedded heterogeneous systems using regression-based learning, in: Int. W. on Power and Timing Modeling, Optimization and Simulation.

[53] Zaman, M., et.al., 2015. Workload characterization and prediction: A pathway to reliable multi-core systems, in: IEEE Int. On-Line Testing Symp.