Accuracy-Aware Power Management for Many-Core Systems Running Error-Resilient Applications

Anil Kanduri, *Senior Member, IEEE*, Mohammad-Hashem Haghbayan, *Senior Member, IEEE*, Amir M. Rahmani, *Member, IEEE*, Pasi Liljeberg, *Member, IEEE*, Axel Jantsch, *Member, IEEE*, Hannu Tenhunen, *Member, IEEE*, and Nikil Dutt, *Fellow, IEEE*

***Abstract*— Power capping techniques based on dynamic voltage and frequency scaling (DVFS) and power gating (PG) are oriented toward power actuation, compromising on performance and energy. Inherent error resilience of emerging application domains, such as Internet-of-Things (IoT) and machine learning, provides opportunities for energy and performance gains. Leveraging accuracy-performance tradeoffs in such applications, we propose approximation (APPX) as another knob for close-looped power management, to complement power knobs with performance and energy gains. We design a power management framework, APPEND+, that can switch between accurate and approximate modes of execution subject to system throughput requirements. APPEND+ considers the sensitivity of the application to error to make disciplined alteration between levels of APPX such that performance is maximized while error is minimized. We implement a power management scheme that uses APPX, DVFS, and PG knobs hierarchically. We evaluated our proposed approach over machine learning and signal processing applications along with two case studies on IoT—early warning score system and fall detection. APPEND+ yields 1.9× higher throughput, improved latency up to five times, better performance per energy, and dark silicon mitigation compared with the state-of-the-art power management techniques over a set of applications ranging from high to no error resilience.**

***Index Terms*— Approximate computing, dark silicon, Internet-of-Things (IoT), power management, runtime mapping.**

I. Introduction

EMERGING application domains, such as Internet-of-Things (IoT), cyber–physical systems (CPSs), big data analytics, and so on, are compute intensive and power hungry [1]. Transistor scaling supported building denser chips that provide higher compute intensity to meet performance requirements of these applications, while keeping the power density constant. With transistor scaling reaching its physical limit, operating voltage approaches its threshold and cannot be further scaled down gracefully with transistor scaling [2]. This leads to rise in power density and subsequently thermal violation. Performance surges of emerging application domains, smaller chip areas, and limited cooling solutions contribute to high power densities and frequent thermal violations, potentially damaging the chip's functionality. To avoid thermal violations, the chip has to function within dissipatble (safe) limits of power. This forces a section of chip to be powered off temporally—this inactive portion is termed as dark silicon [3]. Dark silicon phenomena reduce performance, energy efficiency, and utilization of on-chip resources [2].

Power capping techniques are used to restrict power consumption of the chip to a fixed and safer limit, typically a design time estimate called thermal design power (TDP), beyond which thermal violations may occur [4]. Dynamic power capping and management techniques typically function in an observe-decide-act loop, observe instantaneous power consumption and temperature accumulation, decide on power actuation, and act on the decisions through power knobs [5]. Dynamic voltage and frequency scaling (DVFS), power gating (PG) [6], near threshold computing [7], and adaptive scheduling [8] are widely used knobs for power management. Combinatorial actuation of different power knobs can honor thermal and power constraints, although performance and/or energy gains can be minimal [9]. DVFS knob would be limited as the voltage approaches its threshold and cannot be scaled down any further and also suffers with increase in leakage power. PG knob addresses the issue of static power and is not limited as DVFS. However, the reduction in static power comes at the expense of performance, since only fewer cores are simultaneously powered up. Both DVFS and PG are triggered as a reaction to power violations, which might work for instantaneous power reduction in short term. Despite power capping benefits, they do not offer any substantial gains on performance or energy efficiency.

Approximate computing is emerging as an alternative for dark silicon mitigation, by trading off accuracy for performance and energy gains. Applications from several domains are inherently error resilient, based on their nature of computation and/or input data. For example, algorithms used in multimedia signal processing, machine learning, numerical methods, and so on can be iterative or NP-hard, making them

Manuscript received August 16, 2016; revised December 5, 2016 and February 20, 2017; accepted March 21, 2017. Date of publication April 28, 2017; date of current version September 25, 2017. This work was supported by the Marie Curie Actions of the European Union's H2020 Programme. *(Corresponding author: Anil Kanduri.)*

A. Kanduri, M.-H. Haghbayan, and P. Liljeberg are with the University of Turku, 20500 Turku, Finland (e-mail: spakan@utu.fi; mohhag@utu.fi; pakrli@utu.fi).

A. M. Rahmani is with the University of California Irvine, Irvine, CA 92617 USA, and also with TU Wien, 1040 Vienna, Austria (e-mail: amirr1@uci.edu).

A. Jantsch is with TU Wien, 1040 Vienna, Austria (e-mail: axel.jantsch@tuwien.ac.at).

H. Tenhunen is with the University of Turku, 20500 Turku, Finland, and also with the KTH Royal Institute of Technology, 16440 Stockholm, Sweden (e-mail: hannu@kth.se).

N. Dutt is with the University of California Irvine, Irvine, CA 92617 USA (e-mail: dutt@uci.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TVLSI.2017.2694388

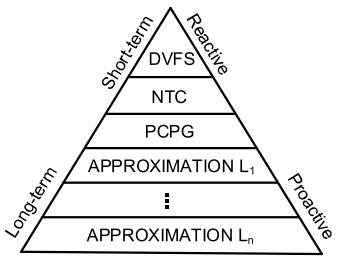

Fig. 1. Power management knobs.

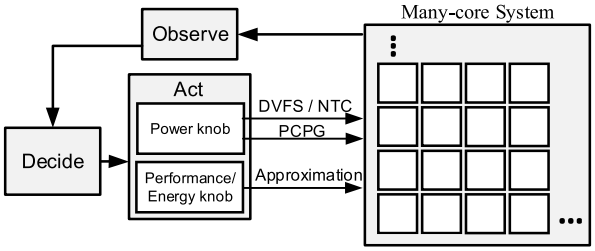

Fig. 2. Approximation knob.

tolerant to inaccurate computations. Similarly, IoT systems deal with continuous sensory data originating from analog or noisy sources, where relaxing certain computations has less effect on eventual result. Approximation (APPX) leverages inherent error resilience of such applications to reduce computational workload and increase energy efficiency.

Alongside soft computing applications, several IoT applications rely on a smart gateway for edge and/or cloud-based processing [10]. With several sensor nodes mapped to a single edge or cloud processor, performance and power challenges continue to effect IoT applications indirectly. Traditional power knobs alone would not suffice to ensure power capping while maintaining higher performance and energy efficiency. To fill the performance and energy gap of power knobs, we propose APPX as another knob for power capping and management. We couple the short-term reactive power knobs, DVFS and PG, with the long-term proactive energy and performance knob, APPX, in a hierarchical manner for power capping, while ensuring performance and energy gains, at the expense of accuracy. Fig. 1 shows power knobs classified in the order of their temporal effect and overhead.

In this paper, which is a major extension of our recent work published in [11], we propose an APPX-enabled power management framework APPEND+ that uses DVFS and PG knobs primarily for power capping and APPX knob for performance. Fig. 2 shows the top-level view of APPX as another knob for power management along with conventional power knobs. We design a power manager that makes decisions on the actuation of DVFS and PG knobs in case of power violation and APPX knob in case of throughput violation. The key idea of this paper is to switch the mode of execution of an application from accurate to approximate upon performance requirements with APPX knob invocation. In our previous work, we switch from accurate to approximate mode of execution among approximable applications, subject to system requirements [11]. At times, this strategy either over compensates for performance surges by approximating beyond the requirement, or falls short of meeting the performance requirement. We fill this gap using *sensitivity metric* for each application at each level of APPX that is used to make mode switching decisions. We use a set of variable accuracy implementations of an approximable task, with each approximate task identified by its sensitivity metric, which is represented as the performance gained per error induced. To enable selection of suitable candidates for mode switching, we prune this set to choose a candidate task for replacing the accurate task that offers maximal performance gain within minimal error. We present a run-time mapping and mode switching algorithm for replacing accurate tasks with approximate tasks from the set of variable accuracy implementations. Our contributions based on our prior work [11] are as follows:

1) APPX knob for closed loop power and performance management;
2) a classification algorithm for identifying approximable tasks that maximizes performance by pruning application space based on sensitivity metric;
3) a run-time mapping and mode switching technique for replacing accurate tasks with approximate tasks;
4) a power management framework APPEND+ that uses DVFS, PG, and APPX hierarchically for power capping and throughput improvement;
5) a case study of IoT applications, fall detection and early warning score (EWS), to evaluate APPEND+.

II. Related Work

A. Power Capping

Power capping techniques monitor the power consumption and actuate power knobs in a closed loop, in case of power consumption exceeding TDP. A PID controller-based power capping is presented in [12], where knob settings are actuated as per normalized gain of PID. Vega *et al.* [13] propose a power capping algorithm using DVFS, per-core PG (PCPG), and core folding, with all power knobs tightly coupled. They suggest that combinatorial usage of different power knobs is effective for system level power capping decisions. Cochran *et al.* [14] have used thread packing, i.e., allocation of threads per core as a power knob along with adaptive DVFS. PGCapping was presented in [6] that uses PCPG and DVFS in a hierarchical way for power capping and life time balancing. Kapadia and Pasricha [8] have used degree-of-parallelism as a knob for power management and to improve system reliability. Application mapping, i.e., spatial alignment of active cores for improving power budget and thus power capping limit, is proposed in [15] and [16]. A multiobjective power capping approach is presented in [15] and [16], which uses the combination of DVFS and PCPG based on network and workload characteristics. Chen *et al.* [18] have proposed using resource allocation at data center level as another knob for power actuation. They use history-based prediction for potential workload to determine CPU resource allocation. While all the above-mentioned techniques use TDP as upper bound, Pagani *et al.* [4] have proposed an adaptive way of setting the upper bound on power consumption, thermal safe power (TSP), as a function of spatial alignment of active

components. All these techniques focus exclusively on power capping and combinatorial usage of power knobs, but do not consider their implications on performance.

B. Approximation

Ansel *et al.* [19] have used variable accuracy implementations of the same algorithm, with language and compiler support to choose one among different implementations for exploring energy-accuracy tradeoffs. Baek and Chilimbi [20] have proposed APPX at software level with a choice between accurate and approximate versions of blocks of code using Green compiler. Hoffmann *et al.* [21] have proposed using energy-accuracy tradeoffs in context of power capping by translating static parameters of an application into dynamic knobs such as convergence for drop in accuracy. However, other APPXs at algorithmic level, such as logic simplification, cannot be translated into dynamic knobs. Escaping infinite loops and skipping iterations of bottleneck loops that consume longer execution time were proposed by Sidiroglou-Douskos *et al.* [22] as loop perforation. All these techniques explore ways to compute approximately for energy and performance gains, within acceptable quality. However, they do not use APPX for closed-loop power actuation.

C. IoT

In the context of IoT applications, the need for computational capacity at a sensor node level would not suffice. To meet real-time performance requirements, data collected over sensory nodes are processed at a smart gateway [10]. The gateway acts as an intermediate edge layer between the sensor front end and the cloud server back end [23]. Typical gateway can be a multicore platform, which still faces with power and energy consumption challenges [24]. Some of the IoT applications are concerned with actuation mechanism based on sensory data analysis, such as identifying sudden changes in input data. Such applications present with an opportunity to relax the accuracy of computation, which in turn can be used to conserve energy and accelerate the performance.

III. Approximation Knob

Approximate execution of an application provides variable performance and energy gains with the amount of error induced. We present the Pareto space of accuracy-performance tradeoffs with two example applications viz., sparse matrix multiplication and k-means clustering. We considered two $10\,000 \times 10\,000$ sparse matrices that are multiplied approximately by skipping inner most loop that performs multiplied accumulation. Fig. 3(a) shows normalized performance and energy gains with the number of inner most loops that are skipped (workload reduced) to reduce the number of computations. With 50% of workload reduced, performance and energy efficiency doubles. For k-means clustering, we use 10 000 random input data points to be classified into 50 clusters. We used relaxed convergence as the APPX, by early termination of the clustering algorithm with nonzero flips. Fig. 3(b) shows the gain in performance and energy for error induced by

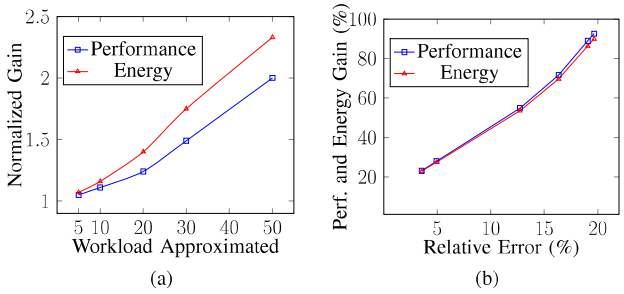

Fig. 3. Performance and energy gains with APPX. (a) Workload-performance tradeoffs for matrix multiplication. (b) Accuracy-performance tradeoffs for k-means clustering.

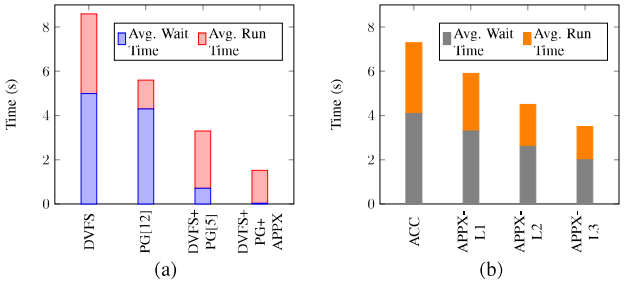

Fig. 4. Performance gains of different knobs for k-means clustering simulated on 16-core system. (a) Application service time for different knobs. (b) Application service time for different levels of APPX knob.

relaxing the convergence from 5%–10%. Relative performance gains increase as the error induced increases. Both these examples establish performance and energy gains with APPX. Specifically, they provide an insight on performance gains per error induced, which can be exploited while implementing the APPX knob.

We demonstrate the impact of different power knobs on the performance of many-core systems with applications dynamically entering and leaving the system using k-means clustering as an example. The performance of the system is determined by service time of an application, which is the sum of wait time, the time elapsed between application request and starting of the execution, and runtime, the time consumed in executing the application on chip [8]. Dynamic workload characteristics contribute to power violations, forcing actuation of power knobs. We simulate the application for four different power knobs viz., DVFS, PG, DVFS + PG (referred as MOC) [5], and DVFS + PCPG + APPX, using the experimental platform, detailed in Section VII. The APPX knob has k levels of APPX, where k is a parametrizable entity. In this case, we chose four levels of APPX.

The average service time for different knob combinations is shown in Fig. 4(a). In case of using DVFS and PG knobs [12], per-application runtime increases forcing incoming applications to wait longer, resulting in high service time. The combination of DVFS and PG has relatively better service time using the power management algorithm, as in [5] and [17]. With the APPX knob in combination with DVFS and PG, the service time is the lowest, indicating high performance and energy gain within the given power budget. The APPX knob

loads applications with relaxed accuracy that have lower workloads and thus low runtime. Consequently, more resources are available for incoming applications, improving the wait time and the overall service time. Fig. 4(b) shows application service time of APPX knob over different levels of APPX. The gain in performance with increasing level of APPX is trivial. Despite effective power capping and possibility of increasing the number of simultaneously active cores, performance still suffers with DVFS and PG when compared with that of APPX. Hence, we propose a hierarchical management for effective combination of these knobs to complement each other.

IV. Accuracy-Performance Tradeoffs: A Case for IoT

IoT applications deal with real world sensor data that are analog and involve noisy components. Performance requirements for IoT systems are usually high while energy budget is limited [1]. Collection and classification of raw data into meaningful clusters, filtering, computation for actuation decisions, and communicating external world are major functionalities of an IoT system. Every stage in this process has a variable tolerance to inaccurate computations, presenting opportunities for APPX. Leveraging this fine-grained error resilience, APPX can provide better performance-per-energy in IoT systems. We explore the possibilities of accuracy-performance tradeoffs in IoT domain using two case studies on health monitoring applications viz., EWS system and fall detection.

A. EWS System

An EWS system is used in health care for monitoring vital signs of a patient to proactively alert medical support. A method for EWS is proposed in [25] that uses three types of sensors—medical, environmental, and activity. Data collected from the sensors are preprocessed by using a Butterworth filter to remove noise components and false alarms. Sensory data from different sources are fused to extract useful details. Every physiological data sensed are allocated a score based on the range the sample belongs to and its implication on patient's health deterioration. The final score is calculated as a combination of scores of all the individual sensor nodes' data. When the final EWS exceeds a fixed threshold, a warning signal is transmitted seeking for medical attention. This system deals with heterogeneous data generated by different sensors and involves computations on unclassified, redundant, and noisy data. To extract meaningful insights, the EWS uses data acquisition, filtering, fusion, classification, and analysis, followed by computation and transmission. Although this system is mission critical, there are several intermediary stages where inaccurate computations can be tolerated. For example, heart rate measured is classified into one of the several ranges of heart rate data and a score is assigned according to the range it belongs to, but the exact value of heart rate is not used. Medical sensors trace several samples of data per second on an average, while a median of these data is good enough to represent all the samples. Relaxing such computations and data points that do not affect final EWS can enhance performance of the system within a lower energy budget.

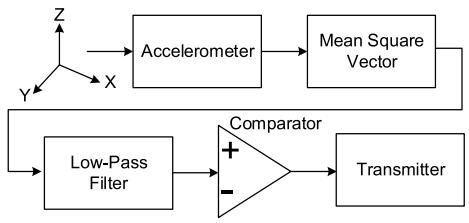

Fig. 5. Fall detection.

B. Fall Detection

Another example of IoT application that is data and compute intensive is fall detection [26]. Fall detection mechanism identifies whether a person using the wearable detector falls hazardously on the ground. Specifically, fall detection is used in the context of patient and elderly people monitoring, to bring attention and support upon a fall. Typical fall detection employs camera and gyroscopic or accelerometer sensors for identifying a fall with respect to inertial position. We use fall detection based on a accelerometer, as proposed in [26]. The accelerometer data in three dimensions are used to calculate signal magnitude vector as the square root of the sum of the squares of signal component in each axis. This is fed to a low pass filter to generate discrete signal of positioning. Unusual spikes in the filtered data when compared with a fixed threshold represent the possibility of a fall, which is then transmitted to support system infrastructure for further assistance. The fall detection mechanism is shown in Fig. 5. A major conundrum in fall detection is in identifying the abnormal spikes in positioning signal—whether to analyze the accelerometer data tightly coupled to the sensor or to transmit the data to a cloud computer. Analyzing the data in a simpler microcontroller has performance penalties while transmitting filtered data to a high-end cloud computer consumes more energy. Expensive floating point computations on high sampled accelerator data, such as multiplications, square root, and filtering, can be relaxed to gain performance. We have run the fall detector mechanism over three sample persons for 8 h of a day. Sensory data are collected at 100 samples/s and fall detection is executed at a gateway between sensor nodes and cloud server. These test cases show that accelerometer signals generate data that are usually redundant, indicating that skipping some of the sensory data samples would induce only a tolerable error. Reducing the sampling of sensor and filter length by half produced results that are similar to accurate computations. Relaxing accuracy of data analysis can thus improve fall detection's performance.

V. Knob Actuation Scenarios

We primarily monitor power consumption, workload intensity, and sensitivity of applications to make knob actuation decisions. The threshold for power consumption is TDP. Power consumption exceeding TDP indicates a power violation. Furthermore, we also set another parameterizable threshold TDP_{th}, a metric that indicates possibility of a potential TDP violation, such that $0.66 \times TDP < TDP_{th} < TDP$. We use accumulated wait time (AWT) of application requests

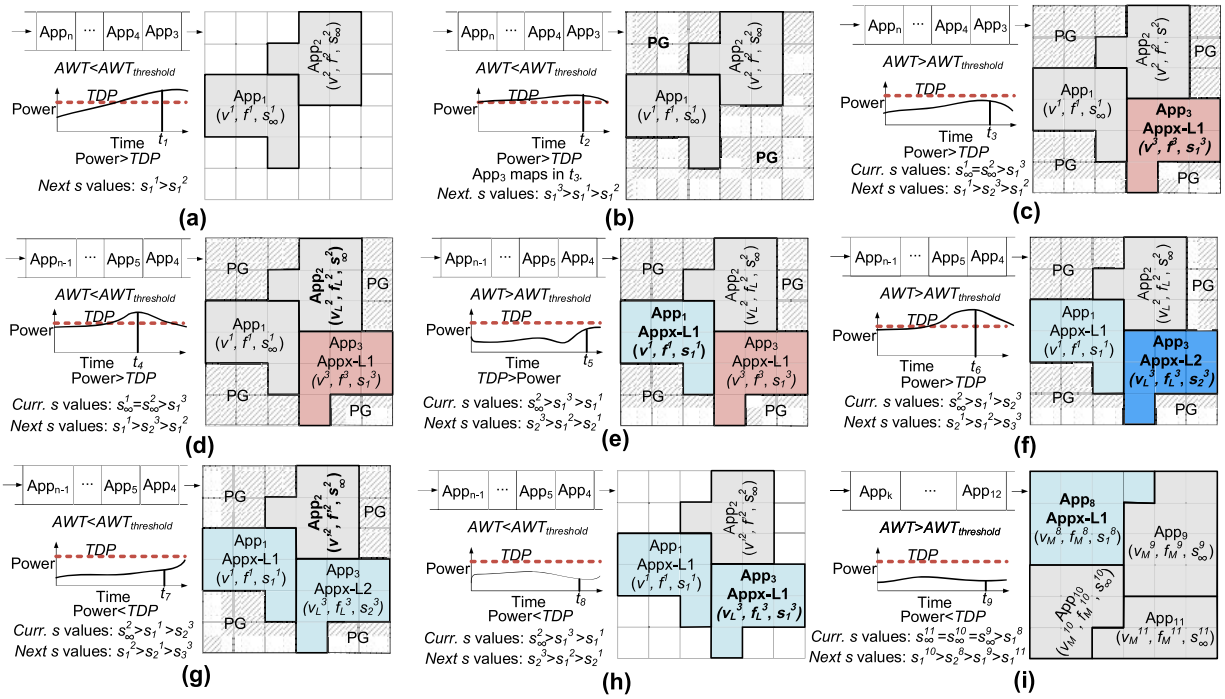

Fig. 6. Knob actuation scenarios. Each scenario shows applications running on a 36-core system along with power consumption and workload intensities.

made for monitoring the workload. For a set of applications $\text{App}_1, \text{App}_2, \ldots, \text{App}_N$ with wait times $w_1, w_2, \ldots, w_N$, AWT is given as

$$\text{AWT} = \left(\sum_{i=1}^{N} w_i\right)/N. \tag{1}$$

Longer application wait time indicates the higher workload intensity. A parametrized metric AWT_{th} is set as threshold for workload intensity. The power objective is to restrict the power consumption to TDP and throughput objective is to restrict the AWT to AWT_{th}. For the actuation of APPX knob, we use sensitivity metric of an application to choose an application and its corresponding level of APPX.

A. Application Sensitivity

The performance and energy gains varies for different applications over different levels of APPX. This presents a case where approximating one application might yield more performance gain than that of the others. We identify each application with a sensitivity metric as performance that could be gained per error induced. The motivation behind this is to choose an application that results in higher performance gain for the amount of error induced.

We define an application's sensitivity metric as

$$\text{Sensitivity} = \frac{\text{Perf}_i - \text{Perf}_{i-1}}{\text{Error}_i - \text{Error}_{i-1}} \tag{2}$$

where Perf_i and Error_i represent performance and error induced at ith level of APPX. Sensitivity of an application for any two given levels of accuracy would be high when the performance gained by lowering accuracy is high or when the accuracy loss in performance improvement is lower. Subject to application characteristics and input data, sensitivity of an application varies through different levels of APPX. Sensitivity metric of an application presents a wider Pareto-space of accuracy-performance tradeoffs that can be explored in choosing an application to be approximated and the level of APPX. Sensitivity metric identifies tasks that will result in the highest performance gain among the set of tasks currently running, and prioritizes these tasks as candidates for switching their mode of execution to approximate. This enables fine-grained control on APPX knob, appropriate choices for mode switching, and performance and energy gain at lower relative error, and limits the possibility of overcompensation with APPX. Details on sensitivity metric used for evaluation purpose are detailed in Section VII. We demonstrate possible scenarios that require knob(s) actuation under diverse power consumption and workload intensities. Fig. 6 summarizes these scenarios, representing power consumption, workload intensity, and knob actuations employed over a span of execution. Each scenario (a)–(i) shows power consumption with respect to TDP, applications waiting in the queue, applications that are mapped on the chip with their respective voltage and frequency levels, and application's sensitivity. Voltage and frequency levels of each mapped application $(\text{App}_1, \text{App}_2, \ldots, \text{App}_n)$ are represented as $[(v^1, f^1, s^1), (v^2, f^2, s^2), \ldots, (v^n, f^n, s^n)]$. The lowest and highest levels of voltage and frequency are (v_L, f_L) and (v_M, f_M), respectively. The corresponding lowest and highest levels of sensitivities are represented as s^1_∞ (lowest level is s_∞ as error is 0) and s^1_m. The sensitivity for each application at different levels of APPX is shown as s^n_i, where n is the application number and i is the level of APPX. Criteria for knob actuation are TDP violation (power > TDP) and high request rate of incoming applications ($AWT > \text{AWT}_{\text{th}}$).

1) *Scenario (a):* Two applications App_1 (v^1, f^1, s^1_∞) and App_2 (v^2, f^2, s^2_∞) are currently running in the

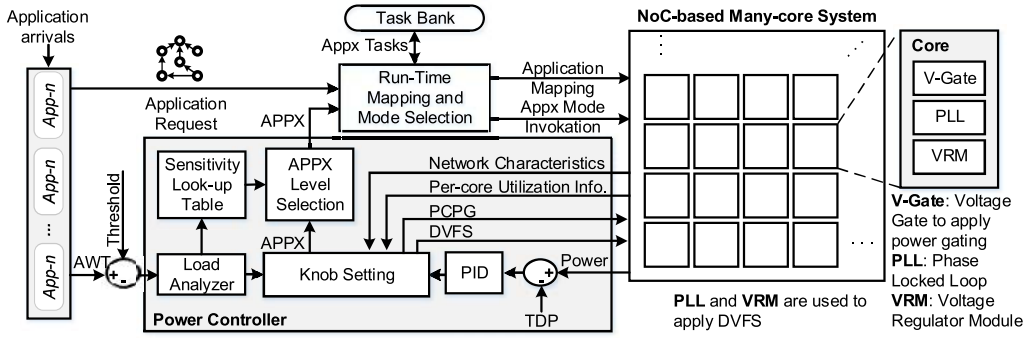

Fig. 7. Power management framework.

accurate mode of execution. At time t_1, a power violation (power > TDP) occurs, while the throughput is under control (AWT < AWT_{th}). DVFS knob is triggered over App_1 and App_2 to lower the power consumption within TDP.

2) *Scenario (b):* App_1 and App_2 are now running at downscaled voltage and frequency levels (v^1, f^1, s^1_∞) and (v^2_L,f^2_L, s^2_∞) with DVFS invocation at t_1. Although power consumption is relatively lower than at t_1, power violation still persists. So, PG knob is invoked to power gate the remaining unoccupied cores (power gated cores are shaded). This would potentially violate throughput constraint, since subsequently arriving applications do not have any free cores to be mapped onto.
3) *Scenario (c):* Power consumption is below TDP and there is no power violation. Application request rate has increased and there is a throughput violation (AWT > AWT_{th}), due to power knobs triggered previously. APPX knob is invoked to counter throughput violation. App_3, arrived at t_3, has a higher sensitivity than App_1 and App_2, and hence is mapped in its approximate version [$\text{App}_3(v^3, f^3, s^3_1)$]. Intuitively, we are reducing the execution time of App_3 so that there are free cores for potentially incoming applications.
4) *Scenario (d):* At time t_4, the power consumption approaches TDP_{th}, indicating the possibility of power violation. DVFS knob is triggered to avoid potential power violation over App_2, which is chosen as candidate for voltage downscaling based on its network and compute characteristics [$\text{App}_2(v^2_L, f^2_L, s^2_\infty)$].
5) *Scenario (e):* At time t_4, power is under control, while throughput violation persists. App_3 is running in approximate mode at level-1. APPX knob is invoked again to address throughput violation, with a choice among switching App_2 to its next level, i.e., level-2 of APPX, or App_1 to level-1 of APPX. The sensitivity metric for App_1 at level-1 s^1_1 is higher than the other two applications at level-2 (s^2_2, s^3_2), hence App_1 is chosen to switch to level-1 of APPX [$\text{App}_1(v^1_L, f^1_L, s^1_1)$].
6) *Scenario (f):* At time t_5, power is under control, while request rate is still high. All the approximable applications are running in approximate mode at level-1. APPX knob is invoked again, with a choice among the applications for maximum performance gain. The sensitivity metric for App_3 at level-2 of APPX is higher ($s^3_2 > s^2_2 > s^1_2$), and is thus chosen to switch the level of APPX further to level-2 (shown in bold) [$\text{App}_3(v^3_L, f^3_L, s^3_2)$].
7) *Scenario (g):* At time t_5, power consumption is below TDP, so the DVFS knob is invoked to upscale the voltage and frequencies of some cores. Among the three applications, App_2 is chosen for upscaling, as it benefits the most based on its network and compute characteristics.
8) *Scenario (h):* At time t_8, power consumption is below TDP, and the application request rate is also below its threshold. DVFS knob is invoked to upscale voltage and frequencies of some more cores. Since App_2 is previously upscaled, App_1 is now chosen as the candidate that benefits from upscaling. Since the throughput constraint is maintained, APPX knob is invoked. App_3, which has a higher sensitivity, is chosen to switch a level up in accuracy, going into level-1 of APPX from level-2 [$\text{App}_3(v^3_L, f^3_L, s^3_1)$]. This invocation can be influenced by a user-defined parameter to upscale voltage instead of switching up the level of APPX.
9) *Scenario (i):* At time t_9, power consumption is well below TDP, allowing more power to be consumed safely. DVFS knob is invoked to upscale the voltage and frequencies of all active cores to their maximum values. The application request is still higher than threshold, despite voltage upscaling and hence APPX knob is invoked. App_8 has the highest sensitivity among running applications, hence it is chosen to switch mode of execution to approximate at level-1 [$\text{App}_8(v^8_M, f^8_M, s^8_1)$].

VI. System Design

We design our power management framework for NoC-based many-core systems supporting dynamic arrival of applications. Our power management framework monitors per-core power consumption and utilization, network intensity, incoming application request rate, and sensitivities of applications to error to actuate different knobs accordingly. The top-level view of our system architecture is shown in Fig. 7.

A. Application Modeling

We model individual computational blocks of an application as a task. Each task is identified by its compute intensity,

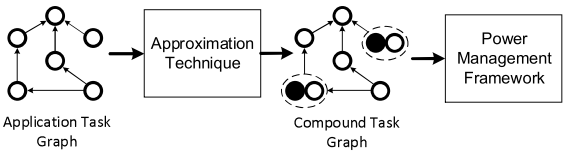

Fig. 8. Compound task graph—workflow.

communication volume with other tasks, and power consumption. Applications are modeled as directed graphs, with each node representing a task running concurrently with other tasks. Analysis of applications' power-performance characteristics and task graph formation is implemented as an off-line function. Incoming applications are classified as approximable and nonapproximable. Approximable applications have one or more tasks that can be replaced by their approximate versions. Such applications are modeled as compound task graphs, as shown in Fig. 8. We generate compound task graphs that include multiple versions of approximable tasks (in dotted lines), and the approximate tasks are shown in solid fill.

B. System Architecture

1) Runtime Mapping and Mode Selection: Incoming applications are queued in the application repository and make a request to be executed on the chip. The run-time mapping and mode selection unit (RMSU) responds to the application request by selecting free cores and maps the application in a task-per-core manner. In addition, if the application is approximable, RMSU buffers the approximate tasks from the compound task graph into the task bank. The task bank is implemented as a memory structure that holds pointers to the addresses of approximable tasks, to provide easier access to approximable tasks without a significant overhead. We use proactive application mapping *MapPro*, presented in [27]. Upon application mapping, RMSU sends the core allocation information to the power controller for potential actuation decisions. Servicing an application depends on the availability of free cores on the chip, which in turn depends on performance and power consumption of active cores. The number of outstanding application requests weighed with the time before they get serviced, and AWT (1) is sent to the power controller for knob actuation decisions.

2) Power Controller: Power controller is the central manager that monitors power consumption, incoming application request rate, and system metrics for power and performance knob actuation decisions. Every core on the chip is provided with power and processor utilization sensors. We use the combination of processor utilization, packet injection rate, and buffer utilization to prune the design space for the selection of candidates that are more suitable for voltage down/up scaling. Selection of appropriate candidates for employing the DVFS and PG knobs is elaborated in our previous work [5], [17].

Thus, we monitor power consumption, processor utilization, network congestion, and network intensity at runtime, forming the monitor phase of the power management framework. Actuation decisions of the power controller are based on parameters received from the monitor phase. We feed the difference between power consumption of the chip and TDP to a PID controller. Output of the PID controller is proportional to the difference between power consumption and TDP and determines voltage and frequency levels to be downscaled to avoid power violation. In case of power consumption being below TDP, voltage and frequency would be upscaled for better power utilization. Knob Setting block of the power controller receives the new voltage and frequency levels from the PID controller, along with processor utilization, buffer utilization, and packet injection rate from the monitor phase. Based on utilization and network parameters, knob setting block decides the cores to which voltage and frequency levels are to be updated. The PID controller's output is also used to decide the number of cores to be power gated. Both DVFS and PG actuations are applied to the chip, as shown in Fig. 7.

Load analyser compares application request rate, represented by AWT, and the threshold, AWT_{th}, to determine throughput violation ($AWT > AWT_{th}$). The *Knob Setting* uses this information and invokes APPX knob. Sensitivity metric over different levels of APPX for all the applications is summarized into a lookup table. Each application has k (parametrized) levels of APPX and sensitivities associated with each level. The level of APPX of an application that is currently running on the chip determines current sensitivity factor, while the ones that are preceding and succeeding are the previous and next sensitivity factors. The lookup table is pruned to find the application that has the highest sensitivity factor in its next level of APPX. For example, consider App_1 running at level-1 of APPX and app_2 running at level-2 of APPX. If APPX is to be invoked, the next level of APPX for app_1 is level-2 and for app_2 is level-3. So, the sensitivities of app_1 at level-2 (succeeding the current level-1) and app_2 at level-3 (succeeding the current level-2) are compared to find the application with highest next sensitivity value. The chosen application and corresponding level of APPX are forwarded to the RMSU. The RMSU retrieves the approximable tasks of the chosen application with the level specified by the APPX Level Selection from the task bank. The accurate task is then replaced with the approximate task retrieved from the task bank. For evaluation, we currently use four levels of APPX in increasing order of accuracy-performance tradeoffs. Alternatively, several fine-grained levels of accuracy tradeoffs could be used.

Mode switching: APPX knob invocation triggers switching the mode of execution of an approximable application. Depending on APPX knob setting, RMSU chooses the version of task to be included in the application mapping, while the other versions are buffered. With the invocation of APPX knob, there are two possible scenarios for mode switching: 1) mapping approximate task graphs and 2) switching mode of execution of applications currently running by task replacement. In the former case, the RMSU maps every incoming application in its approximate version by including the approximable tasks instead of accurate tasks, until the mode is switched back to accurate. The level of APPX is specified by the power manager. For applications that are currently running on the chip, power manager chooses the application(s) and the level of APPX to switch to. Based on these, RMSU identifies the corresponding approximate task specified by the power

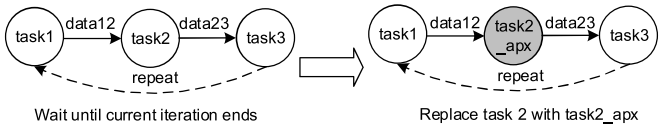

Fig. 9. Mode switching.

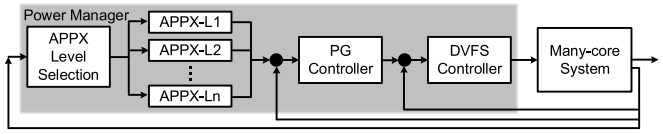

Fig. 10. Hierarchy of knobs.

manager from the task bank and replaces the accurate task with the approximate task. We modeled applications for evaluation as data-dependent concurrent tasks that execute periodically. The computational process repeats until the end of execution with a specified periodicity. Streaming and signal processing applications are good examples, which execute periodically over incoming samples of data, where it is possible to relax certain aspects of computation when new data arrive every period. In a similar manner, IoT applications work on real world sensory data, preprocessing, filtering, and computation over continuous intervals in a batch. Precisely, in these applications, the computational task remains the same while new data arrive after every interval. When the RMSU has to replace an accurate task with approximate, it lets the current iteration of accurate task's computation to finish execution. It waits until data from the accurate task are received at its destination end task (if any). Once the data transfer is completed, the RMSU loads the approximate task on to the chip, replacing the accurate task. Fig. 9 shows the process of task replacement during mode switching. The example has three tasks 1–3 out of which task2 is approximable. On invocation of APPX knob, the switching happens in the following sequence: 1) the RMSU finds the approximate task $task2\_appx$ from the task bank; 2) it waits until data from task2 (data23) is received at task3; 3) $task2\_appx$ is loaded by fetching the instruction stream into the cache (I-cache); and 4) after the data is received at task3, the execution of task2 will now start from new instruction stream of $task2\_appx$. Depending on the size of instruction cache used, instructions of task2 may require flushing, however, this is subject to hardware platform. Since the computational process of the application is periodic in nature, data are not changed with mode switching, and moving the data or flushing the data cache (D-cache) is not needed. The state of the application is hence preserved at the end of the period. The mode switching overhead is elaborated in Section VII.

C. Power Management Algorithm

We employ the DVFS and PG knobs synergistically with APPX in a hierarchical way, as shown in Fig. 10. Triggering and tuning of these knobs together for power capping and performance maintenance is handled by power management algorithm, as listed in Algorithm 1. We define three epochs e_1, e_2, and e_3 for actuation of APPX, PG, and DVFS knobs, respectively, such that $e_1 > e_2 > e_3$. At every epoch e_1, application request rate is monitored. The difference between AWT and its threshold is used to choose settings for APPX knob. Settings for the APPX knob viz., application and level of APPX, are determined by level selection algorithm, as listed in Algorithm 2. The extent of throughput violation (ΔT) determines the mode of incoming applications. If ($\Delta T > 1$), it reflects a steeper request rate of applications. Then, the mode is set to in, indicating to the RMSU that newly incoming applications are to be mapped in their approximate mode at level-1. If ($1 > \Delta T > 0$), the mode is set to run, i.e., to switch the execution mode of currently running applications only. In this case, based on sensitivity factors of currently

Algorithm 1 Power Management Algorithm

Inputs: P: Power Consumption, AWT: Accumulated Wait Time;
Outputs: $APPX$, $PCPG$, $DVFS$, $mode$: Knob actuations and Mode switching commands for RMSU;
Constants: TDP: Power budget, AWT_{th}: *AWT* Threshold, $e1$, $e2$, $e3$: Knob actuation epochs, $SFLT$: Sensitivity factor look-up table
Global Variables: $currSF[\,]$, $nextSF[\,]$, $prevSF[\,]$: Sensitivity factor vectors of current, next and previous levels of approximation of running applications; $cores_{un}$, $cores_{gate}$, $cores_{pref}$: Cores - Unoccupied, power gated and preferable for DVFS.
Variables: App_R, App_X, App_Y: Applications - currently running, approximable, running in approximate mode; SF: Sensitivity factor;
Body:

```
for epoch = e1 do
    ΔT = AWT − AWT_th;
    (app, level, mode) = appxChoose(ΔT, currSF);
    APPX(app, level, mode);
    for epoch = e2 do
        ΔP = P − TDP;
        if ΔP ≥ 0 then
            PCPG_gate(cores_un);
        else
            PCPG_un(cores_gate);
        for epoch = e3 do
            ΔP = TDP − P;
            if ΔP ≥ 0 then
                DVFS_down(cores_pref);
            else
                DVFS_up(cores_pref);
```

Algorithm 2 APPX Level Calculation Function [$appxChoose()$]

Inputs: ΔT, $currSF(SF[\,])$;
Outputs: app: Application chosen for approximation, $level$: level of approximation, $mode$: Mode for incoming applications;
Body:

```
if ΔT ≥ 0 then
    if ΔT ≥ 1 then
        mode = in;
    else
        mode = run;
    SV[ ] ← nextSF(SFLT[ ]);
    sort(SV[ ]);
    (app, level) = max(SV[ ]);
else
    SV[ ] ← prevSF(SFLT[ ]);
    sort(SV[ ]);
    (app, level) = min(SV[ ]);
return (app, level, mode);
```

TABLE I
APPLICATIONS' ENERGY-ACCURACY TRADEOFFS

Application		Norm. Perf.				Norm. Energy				%Error			
		Level-1	Level-2	Level-3	Level-4	Level-1	Level-2	Level-3	Level-4	Level-1	Level-2	Level-3	Level-4
Machine Learning	Linear Regression	1.01	1.09	1.15	1.20	1	1.08	1.15	1.2	0.01	0.05	0.08	0.1
	K-means	1.23	1.54	1.71	1.92	1.22	1.53	1.69	1.89	3.53	12.74	16.34	19.66
	K-NN	1.08	1.1	1.15	1.57	1.11	1.13	1.18	1.75	1.13	2.1	5.85	28.8
Signal Processing	FFT	1.01	1.01	1.05	1.07	1.04	1.06	1.08	1.11	2.17	4.14	7.6	13.34
	LPF	1.1	2.0	3.34	10.5	1.2	2.0	3.5	9.33	15.72	16.6	19.35	30.09
	Vector Multi-plication	1.07	1.18	1.25	1.68	1.06	1.15	1.2	1.6	8.53	12.39	18.2	30.0
IoT	EWS	1.11	1.15	1.33	1.49	1.22	1.29	1.37	1.57	10.4	14.5	17.49	27.72
	Fall Detection	1.1	1.39	1.7	2.06	1.1	1.4	1.75	2.33	9.18	11.38	14.1	22.9

Algorithm 3 Mode Switching Algorithm

Inputs: app: Application chosen for mode switching, $level$: Level of approximation, $mode$: Mode of mapping a new application
Outputs: Map: Mapping configuration of incoming application;
Variable: $newApp$: Incoming application, t^{level}: Approximable task and its level of approximation;
Body:

```
if mode = in then
    if newApp then
        for t^a ∈ newApp do
            TaskBank.push(t^a);
            switch(t^1, t);
            Map(newApp);
if mode = run then
    wait until current iteration of t finishes;
    switch(TaskBank→ app.t^level, app.t);
```

running applications, the sensitivity factor vector for the next level of APPX is looked up from the sensitivity factor lookup table (SFLT). This vector is sorted to find the application with highest sensitivity among running applications. The chosen application and level of APPX are returned to the power manager for invocation of APPX knob. In case the throughput is not violated ($\Delta T < 0$), the APPX level of a chosen application is shifted up, to a relatively accurate level. The sensitivity factors of preceding level of APPX of currently running applications are looked up from SFLT. These are sorted to find the application that is least sensitive to error. This application is chosen for the invocation of APPX knob, and the application and its level of APPX are returned to the power manager. APPX knob is invoked by the power manager by sending the knob settings received from the level selection algorithm to the RMSU. This is presented in a listing in Algorithm 3. When the mode is in, new incoming application's tasks are buffered into the task bank. The new application is mapped in its approximate mode at level-1. When the mode is set to run, the task and its level of APPX specified by the power manager are retrieved from the task bank. Accurate version of this task is replaced by the approximate task. Epochs e_2 and e_3 are smaller than e_1 and are concerning power violations. Power violations are monitored at epoch e_2. If power consumption exceeds TDP, PCPG knob is actuated by PG cores that are currently unoccupied on the chip. Conversely, when power consumption is below TDP, cores that are previously power gated are powered up. At epoch e_3, power violations are addressed by DVFS knob. Cores that benefit relatively higher from DVFS actuation are the preferable cores. DVFS knob is actuated over these preferable cores. Similar to the PCPG knob, voltage and frequency levels of preferable cores are upscaled when power consumption is below TDP. The actuation of PCPG and DVFS knobs is based on our prior work on multiobjective power management framework [5], [17].

VII. EVALUATION

In this section, we assess the efficiency of our APPX-enabled power management approaches APPEND+ and APPEND, with and without considering sensitivity of application. We compare our approach against the state-of-the-art dynamic power management/capping techniques PG [12] which is based on PCPG, and MOC [5] which is based on per-core DVFS and PCPG.

A. Application Setup

For evaluation purpose, we choose interdisciplinary error-resilient application domains of machine learning and signal processing. Furthermore, we selected two applications from IoT domain, given the nature of input data and computations involved. The applications used for the evaluation of APPEND+ are presented in Table I. The chosen machine learning applications are data-triggered on-line learning techniques that fall under classification and estimation. They are interdisciplinary, specifically with IoT-based applications, being used in recognition, mining, synthesis, and automation that are performance and energy demanding. These workloads are based on the iterative methods of computation, meaning that the accuracy of result converges toward an optimal solution with a more number of iterations. Since an accurate solution may not exist and lower convergence could still offer an acceptable result, they become candidates for APPX. For evaluation purpose, we choose four levels of APPX, level-1 through level-4. We normalize the performance and energy gains of approximate tasks from level-1 to level-4 against their accurate versions. Table I shows the normalized gain in performance and energy, and relative error induced with APPX for different applications and levels of APPX. The applications tested are error resilient in general. We chose APPXs that result in soft errors, ensuring there are no critical errors or exceptions. For linear regression, we use training data of 1 million data samples and a test it over 1000 samples.

TABLE II
APPLICATION SENSITIVITY

Application		Sensitivity			
		Level-1	Level-2	Level-3	Level-4
Machine Learning	Linear Regression	10.1	0.09	0.13	0.23
	K-means	6.51	0.28	0.44	0.64
	K-NN	0.07	0.06	0.06	0.43
Signal Processing	FFT	0.01	0.01	0.05	0.07
	LPF	0.6	5.5	2.9	1.84
	Vector Multi-plication	0.24	1.59	0.31	0.83
IoT	EWS	0.95	0.26	0.58	0.3
	Fall Detection	1.02	0.86	0.72	0.14

We use loop perforation to skip 5% to 15% of computations on input training data. The error is calculated as relative to accurate regression. For k-means clustering and k-nearest neighbors, we use relaxed convergence. We compromise on the number of flips, coverage of neighbors, and training data sets, respectively, for these applications. We set the limits of relaxation on convergence from 3% to 10% for four levels of APPX. For the fast Fourier transform, we approximate the computation involving exponential functions with relaxed memorization and storing the twiddle factors in lower precision. We compute the complex exponential for one iteration and reuse it for subsequent samples, despite the inputs to exponential function not being the same. For a low-pass filter, we use a Blackman window with 50 coefficients. We reduce the number of coefficients up to 5 for relaxed execution. We use sparse vector multiplication because of its broader usage and application in several other fields. For this application, we simplify the logic of product calculation that replaces accumulated multiplications of rows and columns with a single multiplication of means of a row and column. For IoT case study on EWS, we reduce the number of samples of heart rate sensor and compromise data fusion. We run the accurate and approximate versions on data sets collected from three different subjects. For fall detection, we reduce the sampling rate of an accelerometer and the number of filter coefficients, and simplify the logic of magnitude vector computation. We use real-time accelerator data collected from a subject wearing the fall detector, over different physical activities of walking, sprinting, and resting.

We used the normalized performance gain and relative error induced upon mode switching for each application over four levels of APPX. We calculated the error sensitivity as gain in performance per error, as described in (2). Table II shows the sensitivity metrics for the applications used over different levels of APPX. It should be noted that normalized gain and sensitivity with increasing level of APPX are distinct. The sensitivity metric represents the amount of performance gained per amount accuracy lost by moving a level of APPX further. For example, linear regression has sensitivity of 10.1 at level-1, while the sensitivity at level-2, 0.09, is much lower than the previous level. This intuitively means that changing the level of APPX for this application from level-1 to level-1 either has a lower performance gain or higher error penalty or both. Similarly, k-means at level-1 has much higher sensitivity than that of the other levels. The normalized gain at level-1 for k-means is 1.23, however, the error, 0.01, (see Table I) is extremely small, making the sensitivity high. APPEND+ considers the sensitivity metric of all the applications currently running on the chip to make decisions on which application and which level of APPX are to be chosen for APPX knob actuation. APPEND is oblivious to the sensitivity metric and chooses applications in a naive manner and corresponding level of APPX in a sequentially increasing order.

B. Simulation Environment

Applications are modeled as task graphs, as described in Section VI. We implement each application such that one task is allocated one core on interval-core-based Sniper simulator, annexed with McPAT for modeling power [28]. We used Gainestown architecture that has Nehalem-like processing elements (PEs) with 32 kB of instruction and data caches. We model the application into concurrent tasks, preserving data flow nature. We use loop perforation and relaxed convergence in case of approximate tasks. We extract execution time, average power, and energy consumption per task. We normalize these values as compute factor metric for each node in the task graph, along with the amount of data flow as the communication volume between tasks. The task graph for each application is thus a directed network of nodes that holds execution time, communication volume, average static, and dynamic power consumption for accurate and approximate tasks. The performance and power values extracted from these simulations provide the relative performance and power gains of a task when the execution is switched to approximate from accurate. We use this ratio to model the power and throughput gains while simulating, so that the APPEND+ framework can be adaptive for all hardware platforms irrespective of architecture. The Sniper simulator provides PEs with other variants of microarchitecture. The relative performance gains for approximate versions over the accurate versions may hold good over different hardware platforms, unless the architecture is highly customized. We use our in-house cycle accurate simulator implemented in SystemC to evaluate the proposed power management framework. We extended Noxim [29] NoC simulator using its network infrastructure for interconnects. The power characteristics of PEs are modeled based on metrics extracted from McPAT and Lumos [30]. Lumos is an analytical framework that quantifies power-performance characteristics with technology node scaling for many-core systems. We used Lumos for physical scaling parameters, voltage scaling, and TDP metric for different network sizes. We added the support for dynamic arrival and servicing of applications through the run-time mapping unit. The mapping unit receives commands from power controller, implemented as a software module. The test bed is a rectangular network with X-Y routing. The $tile_{(0,0)}$ of the mesh acts as the central manager that is responsible for keeping track of mapping information. The network size is 12×12 and the chip area is 138 mm^2. For the first node selection in the run-time mapping process, we use MapPro [27] method. For the DVFS purpose, we use 15 VF levels with voltage in the range of 0.8–1.2 V. The frequency of the on-chip communication network (e.g., routers) is set to the maximum level (similar to [12] and [5]). The TDP value is

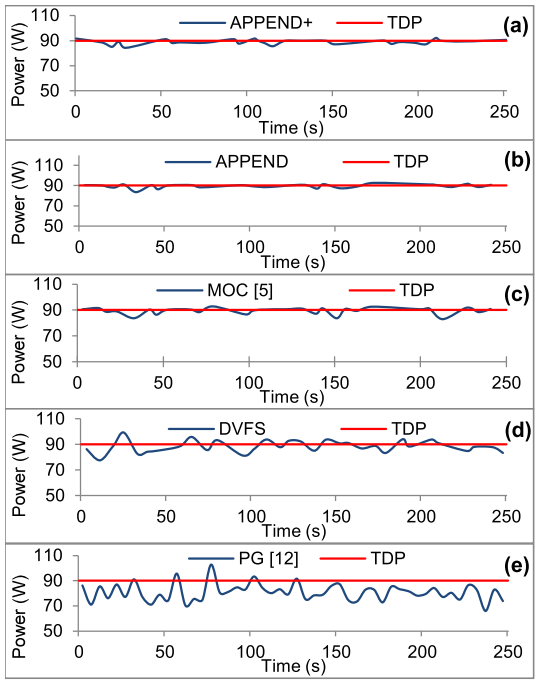

Fig. 11. Instantaneous power consumption of different knobs (a–e) on a 144-core system. (a) APPEND+, (b) APPEND, (c) MOC, and (d) DVFS, (e) PG. Power is capped at TDP.

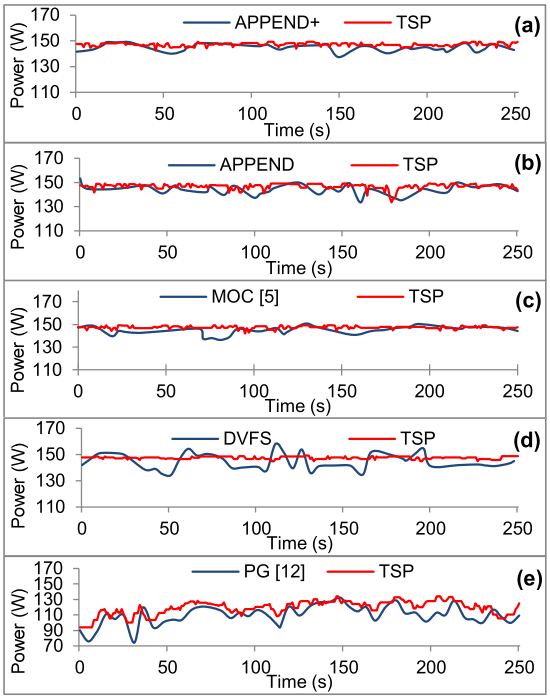

Fig. 12. Instantaneous power consumption of different knobs (a–e) on a 144-core system. (a) APPEND+, (b) APPEND, (c) MOC, (d) DVFS, and (e) PG. Power is capped at TSP calculated at run-time.

set to 90 W, calculated based on the chip's power density. We also evaluate our approach for power capping under a variable power budget, TSP [4]. TSP is calculated as a function of simultaneously active cores, which vary at runtime based on application arrival and mapping. TSP provides a relatively higher power budget than the conservative design time estimate of TDP. We estimate TSP online and use it as the upper bound on power budget for evaluating APPEND+. We implemented the APPEND+ technique over integrated simulation framework, as summarized earlier. It is possible to implement the same as an operating system level policy, provided the hardware platform supports sensing and actuation of power.

C. Evaluation Metrics and Results

For evaluation purposes, we simulate the system over a period in which 200 applications are serviced. The evaluation metrics are as follows.

1) Power consumption: Power consumption of the system over the period of execution, honoring TDP by capping the power.
2) AWT: Accumulated value of wait time of applications before the application request is serviced.
3) Throughput: Time consumed to service 200 applications.

Our prerequisite goal is to cap the power consumption such that TDP constraint is honored throughout the period of execution. Fig. 11 shows the power consumption of DVFS, PG, MOC, APPEND, and APPEND+, along with TDP constraint, over the execution time for servicing 200 applications. TDP violation is more frequent with PG and DVFS knobs, while TDP is honored for most of the execution period with MOC, APPEND, and APPEND+. Fig. 12 shows the power consumption of the system with TSP as the upper bound on power.

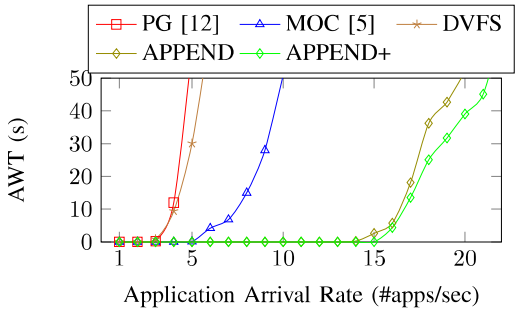

Fig. 13. Accumulated waiting time.

Unlike TDP, TSP varies at runtime offering a flexibility in power capping. DVFS knob violates the TSP limit and is not at efficient power capping. PG honors TSP constraint, however, the power consumption always remains lower than (but not closer to) TSP, indicating a lower utilization of available power budget. MOC, APPEND, and APPEND+ meet the power capping requirement and have a better power budget utilization. APPEND and APPEND+ maintain power consumption closest to TDP when compared with other knob combinations, reflecting better utilization of available power budget. This indicates mitigation of dark silicon and can be attributed to hierarchical usage of power knobs in APPEND+'s power controller. Moreover, we actuate power knobs, DVFS and PG, by monitoring power consumption over an epoch e_1 and trigger the APPX knob proactively over epoch e_2 with e_2 being five times longer than e_1. This eliminates possible random actuations or oscillations between different modes of execution. With better utilization, APPEND and APPEND+ are able to service applications faster, reducing the runtime

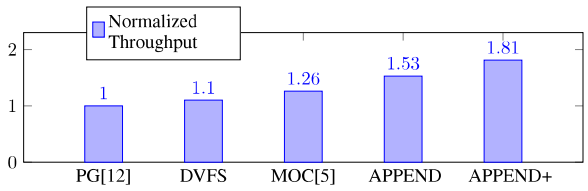

Fig. 14. Normalized throughput (TDP).

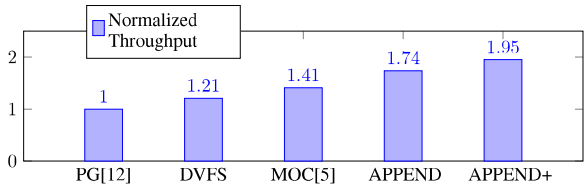

Fig. 15. Normalized throughput (TSP).

and consequently wait time of incoming applications. Fig. 13 shows the AWT for different power capping actuators over the period of execution. We present AWT as a function that is directly related to the rate of application requests made. Similar to power capping, APPEND+ has the best AWT, preceded by APPEND, MOC, PG, and DVFS. DVFS- and PG-based actuations have higher AWTs already when the application request rate reaches 3 per second. MOC has a relatively high AWT when application request rate is 5 per second. However, APPEND and APPEND+ have a near-zero AWT for as long as five times more than DVFS and PG and three times more than that of MOC. This demonstrates the ability of APPEND and APPEND+ to service applications faster despite high workloads, when compared with the other knobs. APPEND+ has AWT greater than zero when the application request rate reaches 15 per second. Also, AWT accumulation is more steeper in case of other knobs than that of APPEND and APPEND+, indicating a substantial rise in their wait times with high request rates. The minimal AWT and high service rates of APPEND and APPEND+ also results in high throughput and energy efficiency. Normalized gain in throughput for all knob combinations with TDP as upper bound and TSP as upper bound is shown in Figs. 14 and 15, respectively. APPEND+ followed by APPEND has the higher throughput, that is up to 1.9× better than PG and 1.4× better than MOC, showing a significant gain in performance and energy while power capping is strictly maintained. Employing APPX knob allows APPEND+ to minimize execution time of applications running on the chip. With applications leaving the system faster, more resources (cores) become available for incoming applications and reduce their wait time. APPEND benefits from AWT and throughput mutually improving each other. It is also to be noted that throughput gain of APPEND+ is relatively higher with TSP than that of TDP. The same trend can be seen with throughput of all the other knobs too, reflecting better utilization of power under TSP constraint. APPX knob can be implemented exclusively in software, making the APPEND+ framework scalable and adaptable across different hardware platforms. In the context of IoT applications, APPEND+ can be used both at the edge and cloud layers which can deliver real-time high performance at the front end. The sensitivity lookup table can be used only to store sensitivity metrics of applications that are currently running on the chip, limiting the size of the lookup table without affecting scalability.

TABLE III
APPLICATIONS—BEHAVIOR AND OVERHEAD

App	Instr.		Instr. Sim (M)		L1-I Accesses (M)		Overhead % (Norm.)
	Acc	Appx	Acc	Appx	Acc	Appx	
Linear Regression	1105	1150	93	75	9	7	1.2
K-Means	1018	1021	449	102	57	14	0.3
K-NN	1457	1513	140	105	37	24.7	0.3
FFT	1147	1159	251.4	249	37	36	0.8
LPF	721	721	197	18	19.9	19	0.1
Vector Multiplication	775	779	33.2	22.7	20.3	19.8	0.5
EWS	1017	1032	66.1	4.1	4.3	4.1	0.3
Fall Detection	795	801	41.9	20.05	3.4	2.23	0.1

1) Error and Overhead Analysis: Behavioral patterns of accurate (Acc) and approximate (Appx) versions of each application are shown in Table III. The approximate versions are at level-4 of APPX, to reflect the maximum overhead caused and maximum performance gained. The number of instructions of each application (Instructions), the number of instructions simulated [Instr. Sim (M) in million], the number of L1-instruction cache accesses [L1-I Accesses (M)] (in million), and normalized overhead (in %) are presented in Table III. These metrics are extracted from individual application simulations (accurate and approximate) on Sniper. A number of instructions are slightly higher for approximate tasks due to conditional branching involved. However, these instructions eventually result in reduced overall workload and hence improve performance. For each application, we used 1 million elements in training set in increasing steps of 100 000 data points per period. A number of simulated instructions depend on training and test data sets used, and are variable in case of different sizes of data used. With loop perforation and relaxed convergence, input data elements are skipped, resulting in fewer instructions required to be simulated. Switching execution from accurate to approximate version incurs some overhead due to monitoring and triggering the approximate version. For every approximate task, the switching of execution mode involves a conditional branching instruction(s). The overhead incurred during this transformation included in the approximate task's compute factor. For the applications we used, the normalized overhead penalty incurred in mode switching ranged between 0.3% up to 1.2%. This overhead is negligible when compared with the workload reduced by APPX and thus levies no significant performance penalty. In terms of power overhead, the task bank and sensitivity lookup tables used in APPEND+ framework are simple memory structures with fewer access during execution of an application. The power consumption of these components is insignificant compared with the total system power. Loading an approximate task involves moving new instruction stream to the instruction cache, with a possibility of increase in the number of DRAM accesses. However, this

depends on the number of application instructions and the size of L1-instruction cache. For instance, a larger application coupled with smaller L1-instruction cache presents a worst case scenario that would force the system to evict accurate task and fetch the approximate task from main memory. In our test bed, we used L1-I cache of 32 kB and all the applications have instructions up to as many as 1500. The worst case penalty in terms of communication for switching from accurate version of a task to the approximate version can be calculated as follows:

$$\text{penalty} = \frac{\text{size}_{\text{app}}}{\text{size}_{\text{pkt}}} \times (P_L + (n \times r_L) + \text{MC}_L + \text{DRAM}_L) \tag{3}$$

where size_{app} and size_{pkt} are the application and packet sizes, P_L is the packetizing latency, n is the number of hops from a core to nearest memory controller, r_L is the router channel latency, and MC_L and DRAM_L are the access latencies of memory controller and off-chip memory. We demonstrate the theoretical worst case overhead penalty of mode switching for a video encoding application run on Intel SCC as an example [31]. The experimental many-core platform Intel SCC has the off-chip memory, core, and network frequencies of 400, 533, and 800 MHz, respectively. The worst case mode switching penalty on SCC for the video encoding application of size 6 kB using the formula in (3) is 1.5 ms. For the same application, penalty in task migration is 10.6 ms, seven times more than the mode switching overhead, to move both instructions of 6 kB and data of 16 kB. Task migration overhead can still be higher when more data are to be moved, while mode switching needs no movement of data. It should be noted that these values are subjective to the platform on which they are executed, while the relative difference in overheads between mode switching and task migration might hold good.

VIII. Conclusion

In this paper, we proposed APPX as another knob for power management in many-core systems. We implemented APPX knob based on application's sensitivity to error such that performance gain is maximized within minimal error. We developed power managing schemes to combine DVFS and PG knobs with APPX knob to meet system requirements in power capping, performance, and energy efficiency. We presented a power management framework, APPEND+, that monitors chip's power and performance requirements at runtime and triggers different knob actuations accordingly. We evaluated APPEND+ against other state-of-the-art power management techniques, over machine learning, signal processing, and IoT applications. APPEND+ improves performance and energy efficiency with the APPX knob and sensitivity aware actuation of the APPX knob, while power capping is maintained with the combination of APPX, DVFS, and PG knobs.

References

[1] W. Arden *et al.*, *More-Than-Moore White Paper Version 2*, 2010, p. 14.

[2] A. M. Rahmani *et al.*, *Dark Side of Silicon*. Springer, 2016.

[3] H. Esmaeilzadeh, E. Blem, R. S. Amant, K. Sankaralingam, and D. Burger, "Dark silicon and the end of multicore scaling," in *Proc. ISCA*, 2011, pp. 365–376.

[4] S. Pagani *et al.*, "TSP: Thermal safe power: Efficient power budgeting for many-core systems in dark silicon," in *Proc. CODES+ISSS*, 2014, p. 10.

[5] A.-M. Rahmani *et al.*, "Dynamic power management for many-core platforms in the dark silicon era: A multi-objective control approach," in *Proc. ISLPED*, 2015, pp. 219–224.

[6] K. Ma and X. Wang, "PGCapping: Exploiting power gating for power capping and core lifetime balancing in CMPs," in *Proc. PACT*, 2012, pp. 13–22.

[7] L. Wang and K. Skadron, "Implications of the power wall: Dim cores and reconfigurable logic," *IEEE Micro*, vol. 33, no. 5, pp. 40–48, Sep./Oct. 2013.

[8] N. Kapadia and S. Pasricha, "VARSHA: Variation and reliability-aware application scheduling with adaptive parallelism in the dark-silicon era," in *Proc. DATE*, 2015, pp. 1060–1065.

[9] T. Komoda, S. Hayashi, T. Nakada, S. Miwa, and H. Nakamura, "Power capping of CPU-GPU heterogeneous systems through coordinating DVFS and task mapping," in *Proc. ICCD*, 2013, pp. 349–356.

[10] A.-M. Rahmani *et al.*, "Smart e-Health gateway: Bringing intelligence to Internet-of-Things based ubiquitous healthcare systems," in *Proc. CCNC*, 2015, pp. 826–834.

[11] A. Kanduri *et al.*, "Approximation knob: Power capping meets energy efficiency," in *Proc. ICCAD*, 2016, pp. 1–8.

[12] M.-H. Haghbayan *et al.*, "Dark silicon aware power management for manycore systems under dynamic workloads," in *Proc. ICCD*, 2014, pp. 509–512.

[13] 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 *Proc. MICRO*, 2013, pp. 210–221.

[14] R. Cochran, C. Hankendi, A. K. Coskun, and S. Reda, "Pack & Cap: Adaptive DVFS and thread packing under power caps," in *Proc. MICRO*, 2011, pp. 175–185.

[15] A. Kanduri, M.-H. Haghbayan, A.-M. Rahmani, P. Liljeberg, A. Jantsch, and H. Tenhunen, "Dark silicon aware runtime mapping for many-core systems: A patterning approach," in *Proc. ICCD*, 2015, pp. 573–580.

[16] M. Shafique, S. Garg, T. Mitra, S. Parameswaran, and J. Henkel, "Dark silicon as a challenge for hardware/software co-design," in *Proc. CODES+ISSS*, 2014, pp. 1–10.

[17] 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," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 25, no. 2, pp. 427–440, Feb. 2017.

[18] H. Chen, C. Hankendi, M. C. Caramanis, and A. K. Coskun, "Dynamic server power capping for enabling data center participation in power markets," in *Proc. ICCAD*, 2013, pp. 122–129.

[19] J. Ansel *et al.*, "PetaBricks: A language and compiler for algorithmic choice," *ACM SIGPLAN Notices*, vol. 44, no. 6, pp. 38–49, 2009.

[20] W. Baek and T. M. Chilimbi, "Green: A framework for supporting energy-conscious programming using controlled approximation," in *Proc. PLDI*, 2010, pp. 198–209.

[21] H. Hoffmann, S. Sidiroglou, M. Carbin, S. Misailovic, A. Agarwal, and M. Rinard, "Dynamic knobs for responsive power-aware computing," *ACM SIGPLAN Notices*, vol. 46, no. 3, pp. 199–212, 2012.

[22] S. Sidiroglou-Douskos, S. Misailovic, H. Hoffmann, and M. Rinard, "Managing performance vs. accuracy trade-offs with loop perforation," in *Proc. FSE*, 2011, pp. 124–134.

[23] T. N. Gia, M. Jiang, A.-M. Rahmani, T. Westerlund, P. Liljeberg, and H. Tenhunen, "Fog computing in healthcare Internet of Things: A case study on ECG feature extraction," in *Proc. CIT*, 2015, pp. 356–363.

[24] C. Tan, A. Kulkarni, V. Venkataramani, M. Karunaratne, T. Mitra, and L. S. Peh, "LOCUS: Low-power customizable many-core architecture for wearables," in *Proc. CASES*, 2016, p. 11.

[25] A. Anzanpour, A. M. Rahmani, P. Liljeberg, and H. Tenhunen, "Internet of Things enabled in-home health monitoring system using early warning score," in *Proc. MobiHealth*, 2015, pp. 174–177.

[26] A. Odunmbaku, A.-M. Rahmani, P. Liljeberg, and H. Tenhunen, "Elderly monitoring system with sleep and fall detector," in *Proc. HealthyIoT*, 2015, pp. 473–480.

[27] M. H. Haghbayan, A. Kanduri, A. M. Rahmani, P. Liljeberg, A. Jantsch, and H. Tenhunen, "MapPro: Proactive runtime mapping for dynamic workloads by quantifying ripple effect of applications on networks-on-chip," in *Proc. NOCS*, 2015, p. 26.

[28] T. E. Carlson, W. Heirmant, and L. Eeckhout, "Sniper: Exploring the level of abstraction for scalable and accurate parallel multi-core simulation," in *Proc. SC*, 2011, pp. 1–12.

[29] F. Fazzino, M. Palesi, and D. Patti. (2008). *Noxim: Network-on-Chip Simulator*. [Online]. Available: http://sourceforge. net/projects/noxim

[30] L. Wang and K. Skadron, "Dark vs. dim silicon and near-threshold computing extended results," Dept. Comput. Sci., Univ. Virginia, Charlottesville, VA, USA, Tech. Rep. CS-2013-01, 2012.
[31] S. Holmbacka, M. Fattah, W. Lund, A.-M. Rahmani, S. Lafond, and J. Lilius, "A task migration mechanism for distributed many-core operating systems," *J. Supercomput.*, vol. 68, no. 3, pp. 1141–1162, 2014.

Anil Kanduri (SM'14) received the B.Tech. degree in electronics and communications from Jawaharlal Nehru Technological University at Kakinada, Kakinada, India and the M.Sc. (Tech) degree in embedded computing from the University of Turku, Turku, Finland, where he is currently pursuing the Ph.D. degree with the Department of Information Technology.

His current research interests include energy efficient computer architectures, run-time management, and approximate computing.

Mohammad-Hashem Haghbayan (SM'14) received the B.A. degree in computer engineering from the Ferdowsi University of Mashhad, Mashhad, Iran, and the M.S. degree in computer architecture from the University of Tehran, Tehran, Iran. He is currently pursuing the Ph.D. degree with the University of Turku, Turku, Finland.

He has several years of experience working in industry as well as developing research tools before starting his Ph.D. His current research interests include high-performance energy-efficient architectures, power management techniques, and online/offline testing.

Amir M. Rahmani (M'08) received the M.Sc. degree from the University of Tehran, Tehran, Iran, in 2009, the Ph.D. degree from the University of Turku, Turku, Finland, in 2012, and the MBA degree from the Turku School of Economics, European Institute of Innovation and Technology ICT Labs, in 2014.

He is currently a Marie Curie Global Fellow with the University of California Irvine, Irvine, CA, USA, and with Technische Universität Wien, Vienna, Austria, and also an Adjunct Professor (Docent) in embedded parallel and distributed computing from the University of Turku. He has authored over 120 peer-reviewed publications.

Pasi Liljeberg (M'09) received the M.Sc. and Ph.D. degrees in electronics and information technology from the University of Turku, Turku, Finland, in 1999 and 2005, respectively.

He is currently a Full Professor in Embedded Computing Architectures with the Embedded Computer Systems Laboratory, University of Turku. His current research interests include parallel and distributed systems, Internet-of-Things, e-Health, embedded computing architecture, fog computing, 3-D multiprocessor system architectures, cyber-physical systems, and reconfigurable system design.

Axel Jantsch (M'97) received the Dipl.Ing. and Dr.Tech. degrees from the Technische Universität Wien (TU Wien), Vienna, Austria, in 1988 and 1992, respectively.

From 1997 to 2002, he was an Associate Professor with the KTH Royal Institute of Technology, Stockholm, Sweden, where he was also a Full Professor of Electronic Systems Design from 2002 to 2014. Since 2014, he has been a Professor with the Institute of Computer Technology, TU Wien. He has authored over 300 articles and one book in the areas of VLSI design and synthesis, HW/SW codesign and cosynthesis, networks-on-chip, and self-awareness in cyber-physical systems.

Hannu Tenhunen (M'83) received the Diploma degree from the Helsinki University of Technology, Espoo, Finland, in 1982, and the Ph.D. degree from Cornell University, Ithaca, NY, USA, in 1986.

In 1985, he joined the Signal Processing Laboratory, Tampere University of Technology, Tampere, Finland, as an Associate Professor and later served as a Professor and a Department Director. Since 1992, he has been a Professor with the KTH Royal Institute of Technology, Stockholm, Sweden, where he also served as a Dean. His current research interests include VLSI architectures and systems, especially network-on-chip systems.

Nikil Dutt (S'84–M'89–SM'96–F'08) received the Ph.D. degree in computer science from the University of Illinois at Urbana–Champaign, Champaign, IL, USA, in 1989.

He is currently a Chancellor's Professor of Computer Science with the Department of Electrical Engineering and Computer Science, University of California Irvine, Irvine, CA, USA. His current research interests include embedded systems, electronic design automation, computer architecture, systems software, formal methods, and brain-inspired computing.

Dr. Dutt is an ACM Distinguished Scientist. He received the IFIP Silver Core Award.