Inter-Cluster Thread-to-Core Mapping and DVFS on Heterogeneous Multi-Cores — Source link

Basireddy Karunakar Reddy, Amit Kumar Singh, Dwaipayan Biswas, Geoff V. Merrett ...+1 more authors

Published on: 01 Jul 2018

Topics: Instructions per second, Multi-core processor, Workload, Efficient energy use and Instruction set

Inter-cluster Thread-to-core Mapping and DVFS on Heterogeneous Multi-cores

Basireddy Karunakar Reddy, Amit Kumar Singh, *Member, IEEE,* Dwaipayan Biswas, Geoff V. Merrett, *Member, IEEE,* and Bashir M. Al-Hashimi, *Fellow, IEEE*

Abstract—Heterogeneous multi-core platforms that contain different types of cores, organized as clusters, are emerging, e.g. ARM's big.LITTLE architecture. These platforms often need to deal with multiple applications, having different performance requirements, executing concurrently. This leads to generation of varying and mixed workloads (e.g. compute and memory intensive) due to resource sharing. Run-time management is required for adapting to such performance requirements and workload variabilities and to achieve energy efficiency. Moreover, the management becomes challenging when the applications are multi-threaded and the heterogeneity needs to be exploited. The existing run-time management approaches do not efficiently exploit cores situated in different clusters simultaneously (referred to as inter-cluster exploitation) and DVFS potential of cores, which is the aim of this paper. Such exploitation might help to satisfy the performance requirement while achieving energy savings at the same time. Therefore, in this paper, we propose a run-time management approach that first selects thread-to-core mapping based on the performance requirements and resource availability. Then, it applies online adaptation by adjusting the voltage-frequency (*V-f*) levels to achieve energy optimization, without trading-off application performance. For thread-to-core mapping, offline profiled results are used, which contain performance and energy characteristics of applications when executed on the heterogeneous platform by using different types of cores in various possible combinations. For an application, thread-to-core mapping process defines the number of used cores and their type, which are situated in different clusters. The online adaptation process classifies the inherent workload characteristics of concurrently executing applications, incurring a lower overhead than existing learning-based approaches as demonstrated in this paper. The classification of workload is performed using the metric Memory Reads Per Instruction (MRPI). The adaptation process pro-actively selects an appropriate *V-f* pair for a predicted workload. Subsequently, it monitors the workload prediction error and performance loss, quantified by instructions per second (IPS), and adjusts the chosen *V-f* to compensate. We validate the proposed run-time management approach on a hardware platform, the Odroid-XU3, with various combinations of multi-threaded applications from PARSEC and SPLASH benchmarks. Results show an average improvement in energy efficiency up to 33% compared to existing approaches while meeting the performance requirements.

Index Terms—Heterogeneous multi-cores, Multi-threaded applications, Run-time management, Performance, Energy consumption.

1 Introduction and Motivation

Heterogeneous multi-core architectures are computing alternatives for several application domains such as embedded [1] and cloud [2]. These architectures integrate several types of processing cores within a single chip. For example, ARM's big.LITTLE architecture contains two types of cores; big and LITTLE, where big cores are grouped into one cluster and LITTLE cores into another [3]. The big cluster has both higher cache capacity and computational power than the LITTLE one. In such architectures, distinct features of different types of cores can be exploited to meet end user requirements. These architectures are also equipped with dynamic voltage and frequency scaling (DVFS) capabilities that enable on-the-fly linear reduction of frequency (f) and voltage (V), yielding a cubic reduction in dynamic power consumption ($\propto V^2 f$). This facilitates to save energy if the power consumption is reduced enough to cover the extra time it takes to run the workload at a lower voltage-frequency (*V-f*).

• *B. K. Reddy, D. Biswas, G. V. Merrett, and B. M. Al-Hashimi are with the Department of Electronics and Computer Science, University of Southampton, U.K.; A. K. Singh is with School of Computer Science and Electronic Engineering, University of Essex, UK.*
E-mail: krb1g15@ecs.soton.ac.uk; a.k.singh@essex.ac.uk; db9g10@ecs.soton.ac.uk; gvm@ecs.soton.ac.uk; bmah@ecs.soton.ac.uk

Manuscript received December X, 2016; revised March X, 2017.

Modern systems equipped with heterogeneous multi-core chips need to deal with multiple applications running concurrently (at the same time) while achieving the desired levels of performance for each of them. Moreover, modern applications are multi-threaded (to exploit multi-core chips), which can be mapped onto different cores for parallel execution, and thus reducing the overall completion time.

Efficient run-time management of multi-threaded applications on heterogeneous multi-cores is of paramount importance to achieve energy efficiency and high performance requirements, that have been a key research focus for mobile and embedded systems [4]–[6]. In general, for each application, the management process first finds a thread-to-core mapping defining the number of used cores and their type, and then operating voltage-frequency levels of cores by looking the workload while satisfying the performance requirement. As part of these, the following experimental observations have been made, which form the motivation behind the proposed approach.

Observation 1: Fig. 1 shows the motivation to map an application on a heterogeneous multi-core architecture containing two types of cores, big (B) and LITTLE (L), where 4B and 4L cores are present. The horizontal axis shows various possible resource combinations. The vertical primary (left-hand side) and secondary (right-hand side) axes

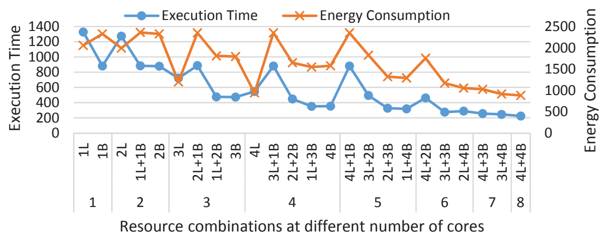

Fig. 1. Execution time (seconds) and energy consumption (J) values by executing the Blackscholes application (from PARSEC benchmark [9]) with various core combinations, including inter-cluster, on ARM's big.LITTLE architecture containing 4 big (B) and 4 LITTLE (L) cores.

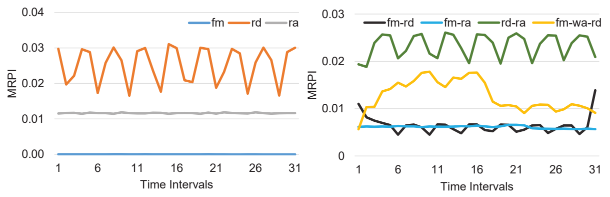

Fig. 2. Variation in MRPI for individual (left) and concurrent (right) execution of multiple applications.

show execution time and energy consumption, respectively, when executing at the various resource combinations. The application execution time scales well with number of cores and further it benefits from mapping onto big and LITTLE clusters at the same time (referred to as inter-cluster thread-to-core mapping). It can be seen that executing on 4L and 4B cores is beneficial in terms of execution time and energy consumption, and thus thread-to-core mapping should utilize the 4B and 4L cores.

Observation 2: Fig. 2 demonstrates the variations in workload when multiple applications are run in two different configurations: individually (left) and concurrently (right) on the A15 cluster of Odroid-XU3 platform [7]. Here, we consider three applications having different workloads from SPLASH: fmm (fm), radix (rd) and raytrace (ra), and their respective combinations fm-rd, fm-ra, rd-ra and fm-rd-ra. The metric considered to classify the workload is Memory Reads Per Instruction (MRPI=$\frac{\text{L2 cache read refills}}{\text{instructions retired}}$) as opposed to the more commonly used CPU cycles [8] for classifying the application workloads. Selection of MRPI is influenced by its relatively low overhead (two performance counters only) and high correlation with the memory intensiveness of an application. Furthermore, we experimentally verified its frequency agnostic behaviour as compared to other metrics, such as Memory Reads Per Cycle (MRPC), having maximal variations with respect to frequency. MRPI classifies the workload based on the degree of memory intensiveness. It can be observed from Fig. 2 that the different workload classes of the applications fm, rd and ra can clearly be classified as compute intensive, memory intensive, and mixed (compute and memory intensive), respectively, when run individually. However, it is completely different in the case of concurrent execution having greater workload variability due to applications' interference. Such classification can help to apply appropriate voltage-frequency levels to optimize energy while satisfying performance constraint.

There are existing approaches for run-time management of concurrently executing applications [6], [10]–[14]. However, the approaches of [10], [11] consider homogeneous multi-core architectures and thus cannot be applied to heterogeneous ones. Approaches proposed in [6], [13] do not exploit the inter-cluster thread-to-core mapping (*observation 1*) and run-time workload classification (*observation 2*) for performance-constrained applications, missing the opportunity for energy savings. In [12], application threads are mapped to more than one type of cores, but the approach heavily depends on off-line regression analysis of performance and energy for all possible thread-to-core mappings and *V-f* levels, which is non-scalable. Moreover, the *V-f* level is not adjusted during execution, i.e. *observation 2* is not exploited, which is beneficial for adapting to workload variations. In contrast, our approach exploits *observation 1* for thread-to-core mapping and *observation 2* for DVFS to achieve energy savings.

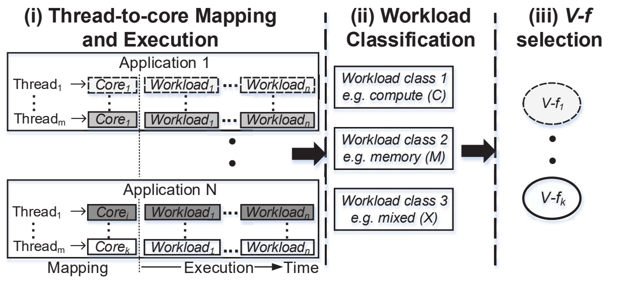
(i) Thread-to-core Mapping and Execution
(ii) Workload Classification
(iii) V-f selection
Application 1
Thread$_1$
Thread$_m$
Core$_1$
Workload$_1$
Workload$_n$
Application N
Core$_k$
Workload class 1 e.g. compute (C)
Workload class 2 e.g. memory (M)
Workload class 3 e.g. mixed (X)
V-f$_1$
V-f$_k$
Mapping
Execution
Time

Fig. 3. Key steps in runtime management of concurrent execution of multi-threaded applications on a heterogeneous multi-core architecture.

The key steps in runtime management of concurrent execution of multiple applications on a heterogeneous multi-core system are summarized in Fig. 3. Considering the above two observations made on inter-cluster exploitation and workload classification, following four main challenges are associated with the run-time management (described subsequently):

(i) Efficient inter-cluster thread-to-core mapping for multiple applications (left-most part of Fig. 3).
(ii) Workload classification for concurrently executing applications based on the identified thread-to-core mapping (middle part of Fig. 3).
(iii) Identification of appropriate *V-f* level of cores for the associated workloads (right most part of Fig. 3).
(iv) Analysing concurrent applications' interference and taking appropriate measures to meet performance requirements.

(i) *Inter-cluster thread-to-core mapping* step needs to identify the number of used cores and their type for each application while meeting the performance requirement. In case of multiple applications, the mapping process has the challenge of allocating the right number and type of cores to each application from the available cores (4 LITTLE and 4 big cores for ARM's big.LITTLE architecture presented in the Odroid-XU3 [7]), such that their performance requirements are satisfied and energy consumption is minimized. The left-most part of Fig. 3 shows an example thread-to-core mapping for each application, where threads are allocated onto different types of cores (highlighted in various grey-scales).

(ii) *Workload classification* step needs to classify the workload within each cluster for the concurrently executing applications (shown under Execution in the left-most part of

Fig. 3) by taking the workload of each core into account. The classification could be into various classes such as compute intensive, memory intensive and mixed [15], which results from the varying ways in which they exercise the hardware. From a performance perspective, it is desirable to run a compute intensive application at a higher clock frequency as compared to memory intensive one such that high performance can be achieved. However, an appropriate metric needs to be identified in order to classify the workloads, using MRPI in our case. A high workload on the processing core means a low MRPI and vice versa. Classification at a given time interval plays a pivotal role for achieving the desired energy-performance trade-off, which is discussed in detail further in Section 4.

(iii) *Appropriate V-f identification* is required for the associated workloads such that the desired energy-performance trade-off can be achieved. Since multiple applications are executed concurrently, the *V-f* level needs to be identified by taking the performance requirements of all of them into account. Further, as different set of *V-f* levels are available for the cores situated into different clusters, e.g. big and LITTLE clusters in ARM's big.LITTLE architecture, it becomes challenging to identify the most suitable *V-f* levels for different clusters while respecting applications' performance constraints. The right most part of Fig. 3 shows an example demonstration of *V-f* assignment.

(iv) *Applications' interference* due to concurrent execution of applications may degrade their performance. In order to meet the performance requirements, the interference level should be analysed and then it should be used to take appropriate measures. The interference level can be measured as the joint performance degradation of applications when executing concurrently in comparison to individual executions. Clustered heterogeneous architectures such as ARM's big.LITTLE represent different amounts of interference on big and LITTLE cluster for the same workload due to different amounts of available memory for them and thus they need to be analysed separately. Thereafter, they need to be exploited to meet the performance requirements.

A close observation of the existing run-time management approaches indicates that they cannot address all the aforementioned challenges for executing multi-threaded applications on heterogeneous multi-core platforms (described in the Section 2). In order to overcome the limitations of the existing approaches towards addressing the above mentioned challenges, this paper makes the following contributions:

1) Offline analysis of individual applications for performance and energy consumption when mapped to various possible resource combinations on a given heterogeneous multi-core platform to obtain profiled data.
2) For concurrently executing applications, an online mapping strategy facilitated by sorted profile data, to compute the minimum energy consumption point, while satisfying the performance and resource constraints. For each application, the computed point defines thread-to-core mapping, and the platform is configured following the mapping to start the application execution.
3) For the chosen thread-to-core mappings of concurrently executing applications, an online energy optimization technique that first classifies their inherent workload characteristics and then pro-actively selects an appropriate voltage-frequency (*V-f*) pair according to predicted workload in order to minimize the switching transitions and energy.
4) Implementation and validation of both the offline and online steps on a real hardware platform, specifically Odroid-XU3 platform [7].

The offline analysis is performed by taking resource/core combinations from the 4 A15 (big) and 4 A7 (LITTLE) cores present on Odroid-XU3 platform [7]. The online mapping strategy chooses thread-to-core mappings such that total number of used cores does not exceed the available cores (4 A15 cores and 4 A7 cores). For online energy optimization, appropriate *V-f* for various workload classes is determined by performing offline design space exploration (DSE) that uses a custom program to generate a varying number of memory accesses. Subsequently, it monitors the workload prediction error and performance loss, quantified by instructions per second (IPS) at runtime and adjusts the chosen *V-f* to compensate. The proposed approach shifts heavy computations to offline and thus helps in reducing the runtime overheads compared to learning-based approaches [8], [16]. The proposed approach is validated on the Odroid-XU3 platform with the various combinations of applications from PARSEC and SPLASH benchmarks.

To the best of our knowledge, this is the first study on run-time management of concurrent multi-threaded applications on heterogeneous multi-core architecture where more types of cores are used by an application at the same time and *V-f* is adjusted at various time intervals during execution through workload selection, classification and prediction.

The rest of the paper is organized as follows. Section 2 presents related works. Section 3 introduces the system model describing the application, architecture and problem definition in more details. Section 4 describes various stages of the proposed methodology. Section 5 presents the experimental results and their analysis with chosen benchmark applications and hardware platform. Finally, Section 6 concludes the paper.

2 Related Work

There have been several works on offline optimization to achieve performance-energy trade-off points for multi-core systems by employing DVFS and/or task mapping [17]–[21]. However, these works have several drawbacks, such as they consider a single application at a time and thus cannot handle concurrent applications [17]–[20], cannot be applied for online optimization as they perform heavy time consuming computations, and most of them are not evaluated on real hardware platform [18], [20], [21]. Online optimization has also been considered to cater for dynamic workload scenarios in order to optimize energy consumption while respecting the timing constraint. For online optimization, either all the processing is performed at run-time or else the optimization is supported by offline analysed results.

For performing all the processing at run-time, several works have been reported [8], [15], [16], [22]–[24]. In [22], the online algorithm utilizes hardware performance monitoring counters (PMCs) to achieve energy savings without recompiling the applications. The authors of [23] present an accurate run-time prediction of execution time and a corresponding DVFS technique based on memory resource utilization. A similar approach, which is a hardware-specific implementation of the stall-based model, is proposed in [15]. In [24], an adaptive DVFS approach for FPGA-based video motion compensation engines using run-time measurements of the underlying hardware is introduced. In [8], online reinforcement learning based adaptive DVFS is performed to achieve energy savings. These approaches perform well for unknown applications to be executed at run-time, but lead to inefficient results as optimization decisions need to be taken quickly and offline analysis results are not used. Further, they are agnostic of concurrent workload variations and thus fail to adapt for concurrently executing multiple applications. Recently, there has been focus on online optimization facilitated by offline analysis results [6], [10]–[13], [25], [26]. Such approaches lead to better performance results than only online optimizations as they take advantage from both offline and online computations. In [10], thread-to-core mapping and DVFS is performed based on power constraint. Similarly, in [11], first thread-to-core mapping is obtained based on utilization and then DVFS is applied depending upon the surplus power. However, the approaches of [10], [11] target homogeneous multi-core architectures and thus cannot be applied to heterogeneous ones.

For heterogeneous multi-cores, recently some works have been reported that consider multi-threaded applications [6], [12], [13], [25], [26]. However, most of these approaches map application threads completely on one type of core situated within a cluster [25] [13] [26] [6]. This reduces the thread-to-core mapping complexity, but misses to benefit from the distribution of an application threads to multiple types of cores at a given moment of time. In [25], performance impact estimation (PIE) is used as a mechanism to predict which thread-to-core mapping is likely to provide the best performance in order to map the threads on the most appropriate core type. In a similar direction, some proposals have used workload memory intensity as an indicator to guide applications' thread-to-core mapping [27]–[31]. For a given platform containing two types of cores as big and LITTLE, such proposals map memory-intensive workloads on a LITTLE core and compute-intensive workloads on a big core. Similarly, in [13], at a given moment of time, all the threads of an application are mapped on one type of cores. The threads are moved from one core type to another when it beneficial by checking at a regular interval. However, DVFS is not exploited in [25] and [13], which can help to achieve further energy savings. In contrast, the approaches of [6], [12], [26] exploit DVFS, but they have several drawbacks. For example, in [26], design space is explored for a single application, which increases exponentially if concurrent applications have to be considered. In [6], each application is executed as single threaded and use only one type of core for it at a time. In [12], application threads are mapped to more than one type of cores, but the approach heavily depends on off-line regression analysis of performance and energy for all possible thread-to-core mappings and *V-f* settings, which is non-scalable. Additionally, *V-f* setting is not adjusted during execution, which is beneficial for adapting to workload variations.

In contrast to existing approaches, our approach considers concurrent execution of multiple applications, distributes threads on more types of cores at the same time (performs inter-cluster thread-to-core mapping), applies adaptive DVFS to save energy consumption and has been implemented in hardware.

3 System and Problem Formulation

This section describes the system architecture and applications considered in this work along with a detailed problem definition.

3.1 System Architecture

The modern heterogeneous architectures contain different types of cores in varying number. Further, cores of the same type are grouped into clusters. One such architecture is considered for our work. We have taken a 28 nm Samsung Exynos 5422 chip hosted on the Odroid XU3 board [7], which is based on the ARM's big.LITTLE heterogeneous architecture and contains two clusters named big and LITTLE [32]. In addition, the chip contains a Mali-T628 GPU and 2GB DRAM LPDDR3. The big and LITTLE clusters contain high performance Cortex-A15 quad core processor and low power Cortex-A7 quad core processor, respectively. The board also contains four real time current/voltage sensors that facilitate measurement of power consumption (static and dynamic) on the four separate power domains: big (A15) cores, LITTLE (A7) cores, GPU and DRAM. The Odroid-XU3 board can run different flavors of Linux. It also supports core disabling and DVFS, helping in optimizing system operation in terms of performance and energy consumption. DVFS can be used to change *V-f* levels at a per-cluster granularity. For each power domain available for a cluster, the supply voltage and clock frequency can be adjusted to pre-set pairs of values. The Cortex-A15 quad core cluster has a range of frequencies between 200 MHz and 2000 MHz with a 100 MHz step, whereas the Cortex-A7 quad core cluster can adjust its frequencies between 200 MHz and 1400 MHz with a step of 100 MHz. The device firmware automatically adjusts the voltage for a selected frequency, therefore, adjusting *V-f* and frequency has interchangeably been used throughout the paper.

3.2 Applications

For multi-core systems, multi-threaded applications represent current and emerging workloads as they can used to evaluate concurrency and parallel processing. Examples of such applications are available in several benchmarks such as PARSEC [9] and SPLASH [33]. Applications from PARSEC and SPLASH benchmarks exhibit different memory behavior, data partitions and data sharing patterns from other benchmarks in common use. The memory behavior of some applications is presented in Fig. 2, which shows whether they are compute intensive, memory intensive or both compute and memory intensive while executing in various time intervals. Such a classification helps to take appropriate actions to perform required optimizations.

We have used applications from PARSEC and SPLASH benchmarks on the multi-core architecture of the Odroid-XU3 platform. For each considered application, the user can specify a performance requirement in terms of completion time of the application. Such performance requirements can be translated to throughput requirements for frame based applications like audio/video applications, where throughput is expressed as a frame rate to guarantee a good user experience. In a similar manner, the completion time requirement can also be translated to an instructions per second (IPS) requirement, as the total number of instructions in an application is known.

3.3 Problem Definition

For an application with R threads to be mapped onto a heterogeneous multi-core architecture with N clusters, i.e. N core types (C_1, C_2, C_3, ..., C_N), where each cluster contains l_i ($i = 1, ..., N$) cores, the possible number of thread-to-core mappings (TC_{map}) can be represented as,

$$TC_{map} = \begin{cases} \sum_{i=1}^{N} l_i + \prod_{i=1}^{N} l_i & R \geq \sum_{i=1}^{N} l_i \ \& \ R \geq l_i \\ R * N + R^N & R < \sum_{i=1}^{N} l_i \ \& \ R < l_i \end{cases} \quad (1)$$

For run-time power management, the modern cluster-based architectures support cluster-wide DVFS, where cores of the same type organized as a cluster are set to the same *V-f* level from a predefined set of *V-f* pairs [3]. For example, Odroid-XU3 [7], and Mediatek X20 [34] platforms employ such architecture. Let l_i be the number of cores of type C_i in a cluster E_i and nF_i be the number of available *V-f* levels. Then, to incorporate the *V-f* levels (nF_i) into thread-to-core mapping decisions, equation 1 will be modified as follows,

$$TC_{map\_VF} = \begin{cases} \sum_{i=1}^{N} l_i * nF_i + \prod_{i=1}^{N} l_i * nF_i & R \geq \sum_{i=1}^{N} l_i \ \& \ R \geq l_i \\ \sum_{i=1}^{N} R * nF_i + \prod_{i=1}^{N} R * nF_i & R < \sum_{i=1}^{N} l_i \ \& \ R < l_i \end{cases} \quad (2)$$

As it can be seen from equation 2, the initial design space is prohibitively large to explore during the application execution at different time intervals, and thus cannot be applied at runtime. In order to overcome this issue, the exploration of mapping can be fixed to the initial design space, and DVFS exploration can be carried at run-time during different time intervals by fixing the mapping from the initial design space. This helps to solve the thread-to-core mapping and DVFS problems orthogonally. We tackle the problem in the same manner, as defined below:

Given an active application or a set of active applications with performance constraints and a clustered heterogeneous multi-core architecture supporting DVFS

Find an efficient static thread-to-core mapping for each application at runtime and then apply DVFS during the application execution to minimize the energy consumption

subject to meeting performance requirement of each application without violating the resource constraints (number of available cores in a platform)

For a total of n applications, there are 2^n possible use-cases, where each use-case represent a set of active applications. Finding all the possible mappings for each use-case might not be possible within a limited time in case the number of applications and/or cores in each heterogeneous cluster increases. Therefore, the mappings can be explored for an individual application and used in conjunction for various use-cases at run-time, which also reduces the overhead to store the mappings. We employ the same measures.

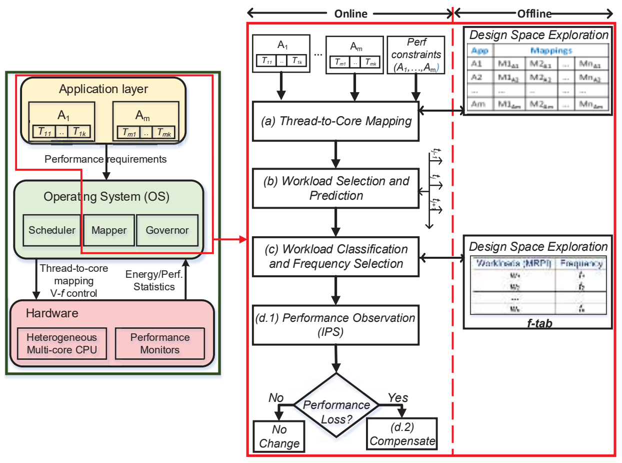

Fig. 4. Overview of a three-layer run-time management (left) and our contributions (right).

4 Proposed Run-time Management

A three-layer view of a typical run-time management is presented in Fig. 4 (left), where each layer interacts with the others to execute an application, as indicated by arrows. The top most layer is the application layer, which is composed of multiple applications having various workload classes. The middle layer is the operating system layer (e.g. iOS, Linux, etc.), which coordinates an application's execution on the hardware (bottom), consisting of multi-core processors. An overview of the proposed run-time management approach employed by the OS has been illustrated in Fig. 4 (right), which has the following stages:

(a) Thread-to-core mapping
(b) Workload selection and prediction
(c) Workload classification and frequency selection
(d) Performance observation and compensation

The novel aspects of proposed run-time management of concurrent execution of multiple applications are as follows:

- Run-time identification of energy efficient inter-cluster thread-to-core mapping that satisfies performance and resource constraints.
- Online selection and classification of concurrent workloads.
- A pro-active online DVFS technique using workload prediction, which takes performance degradation into account and adjusts the chosen *V-f* setting to compensate.

A detailed discussion of each stage is presented in the following sections.

4.1 Thread-to-Core Mapping

In order to meet the performance requirements of the applications to be run concurrently, and to minimize the energy consumption, an effective thread-to-core mapping is

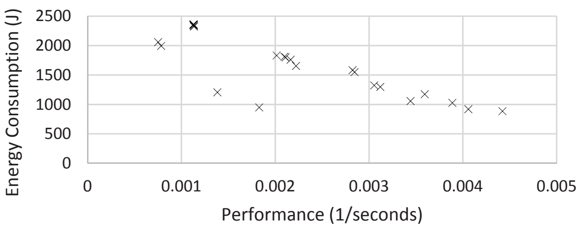

Fig. 5. Design points representing performance and energy trade-off points for $Blackscholes$ application.

important. This involves choosing an appropriate number of cores and their type for each application. Since there are several thread-to-core mapping options for each application, exploring the whole mapping space is time consuming. Therefore, at run-time, thread-to-core mapping is facilitated through an extensive offline analysis to guide application execution towards an energy efficient point. The offline analysis evaluates performance, and energy consumption for all the possible thread-to-core mappings at the maximum available frequency that helps to meet high performance requirements. At run-time, one of these mappings that leads to energy-efficiency while meeting performance and resource constraints is selected for each application. The following sections present a detailed discussion on offline analysis and run-time mapping selection.

4.1.1 Offline Analysis

For each available application, the offline analysis computes all the possible thread-to-core mappings and their performance and energy consumption on a given cluster-based heterogeneous architecture. For the considered applications, the number of threads is greater than the number of cores available on the chosen hardware platform (Odroid-XU3). Therefore, following equation 1, the total number of thread-to-core mappings for each application is 24 (TC_{map} =4+4+4*4). Fig. 1 presents an example analysis for the Blackscholes application that shows 24 mappings and their respective performance (1/execution time) and energy consumption.

The analysis results for each application are stored as design points that represent performance and energy trade-off points for all possible thread-to-core mappings at the maximum frequency. Each design point is represented as 4-tuple: (Prf, EC, n_L, n_b), where Prf, EC, n_L, and n_b denote performance, energy consumption, number of LITTLE cores, and number of big cores, respectively. These design points are sorted in descending order to quickly identify the points meeting a certain level of performance. This helps in minimizing the run-time overhead while choosing the minimum energy point for each performance-constrained application. Fig. 5 shows the design points for the $Blackscholes$ application corresponding to Fig. 1. Similarly, design points are stored for other applications as an outcome of the offline analysis.

4.1.2 Run-time Mapping Selection

For a set of active applications, the runtime mapping has to identify appropriate design points for each application such that the overall energy consumption is minimized without violating the performance and resource constraints (number of cores available in the platform). For example, in case of the Odroid-XU3 platform, the total number of used big and LITTLE cores should not exceed four of each.

Algorithm 1 Run-time thread-to-core mapping selection

```
Input: $CA_{Apps}$, $Apps_{Prfr}$, $DP$
Output: $Map$ for each application
 1: for each application $A_m$ do
 2:     Choose points $D_{A_m}$ ($\in DP$) such that $Prf > Am_{Prfr}$;
 3: end for
 4: for each combination point $CP$ (from $CA_{Apps}$) do
 5:     Compute energy consumption of $CP$ as $Energy[CP]$ (Equation 3);
 6:     Compute total number of used cores of different types (Equation 4);
 7:     Add $CP$ with its $Energy[CP]$ and $C_{i\_UsedCores}[CP]$ in set $CPS$;
 8: end for
 9: From $CPS$, select the point having minimum energy consumption ($minEnergy[CP]$) and satisfying resource constraint (i.e., $C_{i\_UsedCores}[CP] < available\ C_{i\_Cores}$);
10: For the $minEnergy[CP]$, return number of used cores and their types for each application as $Map$;
```

Algorithm 1 describes the run-time mapping selection algorithm. The algorithm takes concurrently active applications (CA_{Apps}), their performance requirements ($Apps_{Prfr}$) and design points (generated in the previous step, $DP = D_1, ..., D_m$) as input and provides a static thread-to-core mapping Map in terms of number of used cores and their types as output for each application. It has been observed in [10] that allocating more number of threads than cores does not actually give any performance benefits. Moreover, by varying number of threads per core (1, 2, .., t) where the value of t can vary depending on the application and resource allocation), the design space becomes prohibitively large. Therefore, to reduce the mapping complexity, the number of threads are chosen the same as the number of cores. For each application, the algorithm first chooses performance requirement satisfying points from its design points. Since the points are stored in decreasing order of performance, the points are chosen as the first entry in the storage to the last entry meeting the performance requirement. Thus, a quick selection of points take place. Then, for each combination point CP (formed by considering one point from each application), energy consumption and used cores of type C_i are computed as follows.

$$Energy[CP] = \sum_{m=1}^{NrCApps} Energy_m \tag{3}$$

$$C_{i\_UsedCores}[CP] = \sum_{m=1}^{NrCApps} C_{i}\_cores_m \tag{4}$$

where, $Energy_m$ and $C_i\_cores_m$ are the energy consumption and used C_i type cores of application m, respectively. For $NrCApps$ active applications, a combination point contains one point from each application.

After above computations for different combination points, the point having minimum energy consumption (based on minimum value selection algorithm) and satisfying the resource constraint is chosen (line 9, Algorithm 1). Then, for this chosen point, the number of used cores and their types (n_L and n_b) for each application are returned as the thread-to-core mapping Map, which is further controlled by `sched_setaffinity` interface in the Linux scheduler. Our approach is generic, but one-time offline analysis is required when the application or platform changes. In case a new application needs to be executed and its offline analysis is not done, the best effort or online learning heuristics [12] can be employed to obtain the mapping, but achieved results might not be efficient.

Simultaneous execution of multiple applications may affect each other due to interference in the shared memory. Thus, their execution time might increase in comparison to the scenario when executed individually. The stretching in the execution time depends upon the compute and memory intensiveness of the applications, e.g. higher stretching is expected for memory intensive applications due to heavy memory accesses in the shared memory. The degraded performances of the applications are compensated by proper *V-f* selections during various time intervals during execution, which is described in the following sub sections.

4.2 Workload Selection and Prediction

V-f setting is a function of workload, at time t_j it can be represented as:

$$V\text{-}f = \begin{cases} f\,(w_j) & \text{Individual workload} \\ f\,(w_{1j}, w_{2j}, ..., w_{Rj}) & \text{Concurrent workloads} \end{cases} \quad (5)$$

where, 1 to *R* represent threads of the application(s). Moreover, in a cluster based architecture, the *V-f* of an individual cluster will be set by considering the workloads of all the cores within a cluster which can be represented as:

$$V\text{-}f_{E_i} = f(W_{E_i}) \quad (6)$$

It is important to note that, for concurrent execution of multi-threaded applications, the *V-f* setting of each cluster should be chosen in such a way that all the applications meet their performance requirements. Furthermore, these applications generate varying and mixed workloads due to resource sharing (e.g. L2 cache and memory) showing intra and inter workload variations during their execution (see Fig. 2). Therefore, we need to select a representative *V-f* pair to guide the further stages in achieving energy efficiency without performance loss. Pseudo code of the proposed online DVFS is given in Algorithm 2.

Assume that there are *R* concurrently executing threads of application(s) on cluster E_i, and w_{rj} is the workload of a thread *r* for time interval $t_{j-1} \rightarrow t_j$. There will be *R* different workloads at every time interval of execution. The workload is quantified by the MRPI, where a low value represents a high load on the core and vice versa. If there are multiple workloads running within a cluster, choosing a *V-f* setting based on an average or single workload may lead to violation of performance requirements for some of the applications. For example, setting the *V-f* according to the high MRPI (memory-intensive) workload may hurt the performance of the compute-intensive workload, as memory-intensive workload can be run at lower frequencies than compute-intensive workload. Therefore, in a cluster-based DVFS supporting architectures, *V-f* level of the cores within

Algorithm 2 Proposed online DVFS approach

```
Input: Application scenario, $T_s$, f-tab and len
Output: V-f pair for each cluster
1: Initialisation: predicted workload ($P_w$)=0, $c_l$=$c_b$=0;
2: PMUINITIALIZE()
3: $f_{cur}$ = cpufreq_get_frequency (core#)
4: while (1) do
5:     compute new IPS value ($IPS_n$) for each application
6:     wait for $T_s$                    /*DVFS interval*/
7:     /*Set V-f level of A15-cluster*/
8:     if ($*c_b \neq len$) then
9:         actual workload($A_w$) = $Pmc\_data\_A15()$
10:        prediction error ($P_e$) = $A_w - P_w$
11:        $q = 0$
12:        FIND_SET_VF($A_w, P_w, P_e, c_b, core\#$)
13:        $q = 1$
14:    else
15:        wait for $T_s * len$           /*Adaptive sampling*/
16:        $*c_b = 0$
17:    end if
18:    /*Set V-f level of A7-cluster*/
19:    if ($*c_l \neq len$) then
20:        actual workload ($A_w$) = $Pmc\_data\_A7()$
21:        prediction error ($P_e$) = $A_w - P_w$
22:        FIND_SET_VF($A_w, P_w, P_e, c_l, core\#$)
23:    else
24:        wait for $T_s * len$           /*Adaptive sampling*/
25:        $*c_l = 0$
26:    end if
27: end while
28: function FIND_SET_VF($A_w, P_w, P_e, c, core\#$)
29:    $P_w$ = EWMA($P_w, A_w, P_e, \&c$)
30:    classify $P_w$, compute $\delta$ and get $f_n$ from $f - tab$
31:    if $q == 0$ then
32:        for each cluster do
33:            $Perf\_loss = ((IPS_{req} - IPS_n)/IPS_{req})$
34:        end for
35:        $\lambda = Perf\_loss * 100$
36:    end if
37:    if ($\lambda > x\%$) then
38:        $f_n = f_n + \lambda * f_{max}$
39:    end if
40:    if ($f_{new} \neq f_{cur}$) then
41:        cpufreq_set_frequency (core#, $f_n$)
42:        $f_{cur}$=$f_{new}$
43:        *c−−
44:    else
45:        *c++
46:    end if
47: end function
48: PMUTERMINATE()
```

a cluster should be set based on the most compute-intensive (minimum MRPI) workload running on those cores. To meet each application's performance requirement, the *V-f* setting for cluster E_i for the time interval $t_{j-1} \rightarrow t_j$ (considering single *V-f* domain for whole cluster) is set by the following workload (line 9 and 20 in Algorithm 2):

$$W_{E_{ji}} = min\,(w_{1ji}, w_{2ji}, w_{3ji}, w_{4ji}, ..., w_{Rji}) \quad (7)$$

Concurrent execution of multiple applications create contention/interference for shared resource, which impacts the performance of an individual application. Furthermore, the memory access latency experienced by each application, calculated at run-time based on the average memory-intensiveness of the running applications, is used to mini-

TABLE 1
Design time analysis of workload classes (MRPI range) and corresponding frequencies

A15				A7	
MRPI Range	Frequency (MHz)	MRPI Range	Frequency (MHz)	MRPI Range	Frequency (MHz)
>0.036	1000	(0.018,0.021]	1600	>0.055	900
(0.033,0.036]	1100	(0.015,0.018]	1700	(0.05,0.055]	1000
(0.03,0.033]	1200	(0.012,0.015]	1800	(0.044,0.05]	1100
(0.027,0.03]	1300	(0.009,0.012]	1900	(0.04,0.044]	1200
(0.024,0.027]	1400	<0.009	2000	(0.032,0.04]	1300
(0.021,0.024]	1500			<0.032	1400

mize the wasted cycles/switching activity by scaling down the frequency. We represent this latency as δ amount of increase in $W_{E_{ji}}$. Therefore, equation 7 can be modified as:

$$W_{E_{ji}} = min\ (w_{1ji}, w_{2ji}, w_{3ji}, w_{4ji}, ..., w_{Rji}) + \delta_{ji} \quad (8)$$

The value of δ is computed from average MRPI of the cores within a cluster. If all the running applications are memory-intensive, then the value of δ will be high due to increased memory traffic. We experimentally verified that, δ increases the application execution time from 1.08% to 3.80%, when multiple applications are executed concurrently. Based on the above observation, the δ value is set to 4.5% of average MRPI.

Proactive control of *V-f* is of utmost importance for online energy minimization [8]. Therefore, the future workload needs to be predicted at t_j to set the *V-f* pair for the interval $t_j \rightarrow t_{j+1}$. An exponential weighted moving average (EWMA) filter [35] is used to predict workload p_{j+1} during the interval $t_j \rightarrow t_{j+1}$ (line 29, Algorithm 2):

$$p_{j+1} = \gamma * a_j + (1 - \gamma) * p_j \quad (9)$$

where γ, p_j and a_j are the smoothing factor, predicted and actual workloads respectively during the interval $t_{j-1} \rightarrow t_j$. It is to be noted that $W_{E_{ji}}$, computed from equation 8 represents the actual workload a_j. To minimize miss-predictions caused by dynamic variations in the workload, the predicted workload of the interval $t_{j-1} \rightarrow t_j$ is compared against the actual workload measured from hardware PMCs (line 10 and 21 in Algorithm 2). Subsequently, computed prediction error P_e (the difference between actual and predicted workloads) is used to improve the workload prediction for $t_j \rightarrow t_{j+1}$. The effectiveness of proactive *V-f* control depends on the accuracy of workload prediction, hence an evaluation is provided in Section 5.

4.3 Workload Classification and Frequency Selection

It is essential to classify the predicted workload for identifying an appropriate *V-f* pair for meeting the performance requirements and optimizing the energy. We use hardware PMCs for periodically getting information regarding architectural parameters during application execution (line 9 and 29, Algorithm 2). The modified performance monitoring tool `perfmon` [36] (enabled access to the A15- and A7-clusters) is used for accessing the PMCs, initialized and terminated through `PMUINITIALIZE` and `PMUTERMINATE` (line 2 and 48, Algorithm 2).

Classification of the workload as compute-intensive or memory-intensive, depends on the instruction mix during the time interval T_s. For example, if there is a large proportion of load/store instructions causing frequent cache misses, then the workload can be classified as memory-intensive. Furthermore, when there are frequent branch miss-predictions and lower-level cache misses (e.g. L1 in Odroid-XU3), the number of instructions executed and MRPI could be low. However, the penalty (measured in cycles) will remain intact no matter what the frequency is, because a branch miss-prediction involves only in-core operations [37]. Therefore, the workload will be treated as compute-intensive and the highest frequency is selected to minimize performance loss. On the other hand, if the processing core is idle or running only background processes, the number of instructions and MRPI may be low. However, this will not come under the compute-intensive case and the operating frequency can be set to a minimum value to minimize the power consumption. Here, the unused core is said to be idle, i.e. no application thread is executing on that core. The idle cores can be identified from the resource combination achieved by the thread-to-core mapping, which decides number of cores and their type allocated to each application (see Table 4). For example, in an octa-core (c_0, ..., c_7) platform, if thread-to-core mapping allocates four cores (c_0, .., c_3) to an application, the remaining four cores (c_4, ..., c_7) will be idle. The status of the core (used/idle) can be maintained in a shared location.

Workload classes are predetermined by observing the variation in execution time through a custom program, which generates a varying number of L2-cache misses by performing memory accesses on a large array. Subsequently, the experiment is repeated ten times across all available frequencies (200 MHz - 2000 MHz) on an Odroid-XU3 platform [7]. The A15-core is an out of order core which takes advantage of memory level parallelism such that part of an L2 cache miss latency overlaps with other independent L2 cache misses. Furthermore, the A15-cluster has greater L2-cache capacity compared to the A7-cluster. The influence of these factors is seen during exploration, and taken into account while choosing the MRPI range and its corresponding frequency. The classified workloads and corresponding optimal *V-f* settings obtained from *f-tab* are given in Table 1, where various MRPI ranges are mapped to frequency through DSE, and used at run-time to set the operating frequency to a desired value through the utility `cpufreq-set`, thereby minimizing run-time overheads (lines 41, Algorithm 2).

The range of MRPI values having little (<1%) or no effect on execution time for the same frequency are grouped into a single class. Application execution intervals with a large

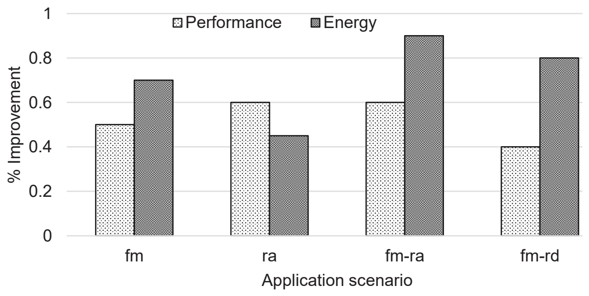

Fig. 6. Effect of adaptive sampling on energy and performance for various application scenarios.

MRPI are actually memory-intensive workloads, so they can be run at lower frequencies to save energy as higher frequencies will simply result in stalls while waiting for data from memory. This motivates us to assign a low frequency to large MRPI workloads, and it is decided by the speed of memory (~933 MHz in our case).

4.4 Performance Observation and Compensation

It is important to evaluate performance during execution to ensure all applications meet their performance requirements. Moreover, dynamic resource availability in a multi-core platform may impact the run-time performance of the applications. Therefore, we use instructions per second (IPS) as a metric for quantifying the run-time performance of each application for every elapsed time interval T_s. The performance loss is calculated once (lines 31-35, Algorithm 2) by comparing the achieved IPS (IPS_n) on each cluster with their required IPS (IPS_{req}) at every time interval. If the maximum performance loss λ% of all the currently running applications during the interval $t_{j-1} \rightarrow t_j$ is significant, the selected *V-f* (f_n) is increased by λ*f_{max} (lines 32 - 39, Algorithm 2) for subsequent time interval ($t_j \rightarrow t_{j+1}$) to compensate. Furthermore, the frequency is modified only when the performance loss (λ) is significant to minimize the overheads associated with DVFS [38]. We experimentally verified and set the value of λ to 1% by taking the variations in PMC data into account [39]. It is worth noting that setting any core's *V-f* to a new value within a cluster is sufficient to change the *V-f* of remaining cores belonging to the same cluster.

4.5 Adaptive Sampling

A smaller time interval (T_s, difference between t_{j-1} and t_j), for which a value of *V-f* is computed and set, increases the run-time overhead and degrades overall performance. Therefore, the value of T_s is experimentally chosen to be 200 ms so as to minimize the overhead on application performance considering the overheads associated with the PMC data collection, subsequent processing, system reliability and DVFS [38]. Furthermore, it can be observed from Fig. 2 that not every combination of applications exhibits large variation in workload during the execution, for example fm and fm-ra. This negligible variations will have no influence on the *V-f* setting. Therefore, to further reduce the run-time overheads, time period is adjusted according to the application workload variations.

To accomplish this we use counters c_b and c_l for tracking the workload variations on A15- and A7-clusters, respectively. These counters get incremented when the workload at t_j is significantly different (MRPI range) than that of the workload at t_{j-1} (lines 40-46, Algorithm 2). When c_b or c_l is equal to a configurable parameter *len*, the run-time adaptation on A15- or A7-cluster (PMC data collection and subsequent processing) is paused for *len**T_s period (lines 15 and 24, Algorithm 2). We evaluated the effect of adaptive sampling for the application workload combinations fm, ra, fm-ra and fm-rd (see Fig. 2), where an average improvement of 0.9% and 0.6% in energy and performance are observed respectively when adaptive sampling is enabled, as shown in Fig. 6.

TABLE 2
Selected applications from PARSEC [9] and SPLASH [33] benchmarks

Benchmark	App Name	Abbreviation
PARSEC	blackscholes	bl
	bodytrack	bo
	swaptions	sw
	freqmine	fr
	vips	vi
SPLASH	water-spatial	wa
	raytrace	ra
	fmm	fm

5 Experimental Validation

5.1 Experimental Setup

The proposed run-time management approach for energy optimization is extensively validated on an Odroid-XU3 platform running a modified Ubuntu Linux Kernel 3.10.96 with a number of combinations of applications from PARSEC [9] and SPLASH [33] benchmarks. The details of the Odroid-XU3 platform are already provided in Section 3.1. We selected applications from PARSEC and SPLASH, based on variations in MRPI values. Table 2 lists the considered applications. The applications are taken in various combinations to make sets of simultaneously active applications. To have better predictability and to ensure that each application meets its performance requirement, the system is not overloaded, i.e. no two applications share the cores. This allows scheduler not to delay the application execution. If applications arrive at different times, the later arrived ones can be mapped by taking the available resources and current status of the existing applications, computed as the remaining time to complete them. If existing applications are going to complete soon, the freed resources by them can be considered to decide the mapping of the current application, otherwise it should be decided based on the current available resources. This also avoids the overhead of data transfer for existing applications as their mapping is not disturbed. Energy consumption is calculated as a product of average power consumption (dynamic and static) and execution time. This includes both the core and memory energy consumption of all the software components (proposed algorithms (Algorithm 1 and 2), profiled data, OS, applications, etc.). The proposed run-time management approach is compared against various approaches, given in Table 5.1, to show energy savings while satisfying the performance constraints. As part of these, the state-of-the-art solution for the run-time resource management of the big.LITTLE, Heterogeneous Multi-Processing (HMP) scheduler [40] with various DVFS governors (ondemand, performance, conservative and interactive) is considered. HMP is a patch to the

TABLE 3
Approaches considered for comparison

Reference	Approach	Abbreviation
[12]	Exhaustive Search-based	ES
[25], [28]–[31]	Workload Memory Intensity based thread-to-core mapping	WMI
[40], [41]	HMP + Ondemand	HMPO
[40], [41]	HMP + Performance	HMPP
[40], [41]	HMP + Conservative	HMPC
[40], [41]	HMP + Interactive	HMPI
proposed	Inter-cluster Thread-to-core Mapping and DVFS	ITMD

standard scheduler in the Linux kernel which dynamically dispatches threads to big and LITTLE cluster according to their characteristics.

Furthermore, a mapping approach, which allocates the application's threads onto only one type of core(s) based on memory intensiveness [25], [28]–[31] is considered for the comparison. As concurrent execution of multi-threaded applications is not taken into account by the above approaches, following changes are made for a fair comparison.

- In case of single-application scenario, a memory intensive application's threads are mapped onto LITTLE cluster.
- In multiple-application scenario, applications are sorted based on their memory intensiveness and then one with the high memory intensity is mapped onto little cores and remaining applications are allocated onto big cluster with equal number of cores.

The proposed approach is also compared against a recently published exhaustive search-based (ES) approach [12]. As part of this, we used the thread-to-core mappings produced by our approach and varied the frequencies (247 design points; 200 MHz - 1400 MHz on LITTLE-cluster and 200 MHz - 2000 MHz on big-cluster) for different application scenarios. Then, selected the configuration (number of cores and their frequencies), having the lowest energy consumption while satisfying the performance requirements. Stamoulis and Marculescu [14] presented a process variation- and workload-aware thread-to-core mapping approach on heterogeneous multi-core systems. However, we could not consider this approach for the direct comparison with the proposed approach for the following reasons. It is proposed for maximizing the throughput under both performance and power constraints, while our approach minimizes the energy consumption under performance constraints. Moreover, the system architecture is different than the one (cluster-based architecture) used in this paper.

To show the effectiveness of the proposed methodology compared to various existing approaches in terms of energy savings and performance, single and multiple-application scenarios are considered for the validation. Moreover, the validation of the workload prediction is also presented.

5.2 Energy Savings and Performance Comparison

5.2.1 Energy Savings

Table 4 presents the resource combinations achieved by the proposed mapping approach at run-time for various application scenarios. The mapping approach takes the individual application performance requirements into account, and

TABLE 4
Resource combination achieved by our mapping approach at run-time for different application scenarios.

App scenario	App combination	Resource combination
single	bl	4L+4B
	bo	4L+4B
	sw	4L+4B
	fr	4L
	wa	4L+4B
	ra	3L+4B
double	bl-bo	2L+2B : 2L+2B
	bl-sw	4B : 4L
	fr-sw	4L : 4B
	wa-bo	2L+2B : 2L+2B
	wa-bo	4L+3B : 1B
	wa-ra	4L+3B : 1B
triple	bl-bo-sw	3B : 1B : 4L
	bl-bo-fr	3B : 1B : 4L
	sw-bo-fr	4L : 1B : 3B
	bl-sw-fr	3B : 1B : 4L
	wa-ra-fm	3L+2B : 1B : 1L+1B
	wa-ra-vi	2L+2B : 1B : 2L+1B

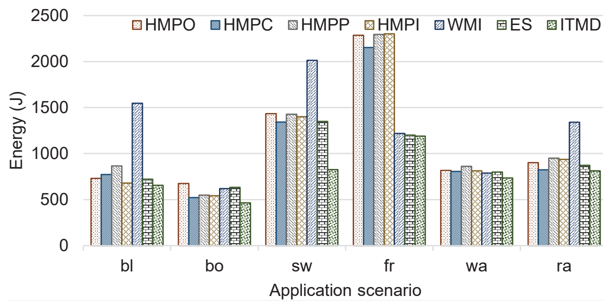

Fig. 7. Comparison of proposed approach with reported approaches for single active application.

chooses the points that minimize total energy consumption from the sorted profiled data, without violating the resource constraints. As discussed earlier, the selected thread-to-core mapping is not altered during the application execution.

In single-application scenario, there is only one active application. Fig. 7 shows the comparison of the adopted approach with existing techniques in terms of energy consumption. First, an energy efficient thread-to-core mapping is determined to satisfy the given performance requirement and resource availability. The experimental observation shows that, for most applications our thread-to-core mapping approach tends to choose all available cores, except for *fr* and *ra* (see single-application scenario in Table 4), as it is the energy efficient point. Afterwards, the proposed online DVFS approach takes control of the frequency scaling to minimize the wasted cycles in case of memory-intensive workloads. It periodically samples the PMCs data and uses a proactive *V-f* setting strategy using the workload prediction. From Fig. 7, it can be observed that the proposed method ITMD outperforms all existing approaches which used HMP scheduler for thread-to-core mapping with various Linux governors for DVFS and *WMI*. Except for *fr* and *ra*, energy savings are mostly due to DVFS as both HMP and proposed approach have the similar thread-to-core mapping. The higher energy savings in case of *fr* are because of mapping threads to power efficient A7 (L) cores, which is the same as that of WMI. It also has long execution that benefits from periodic DVFS. On an average, proposed approach achieves, 25%, 20%, 27%, 22%, and 33% energy

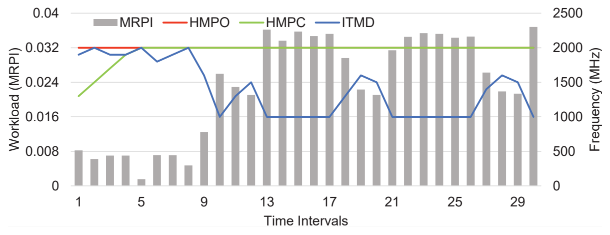

Fig. 8. MRPI and frequency at different time intervals of the application *fr* execution for various approaches.

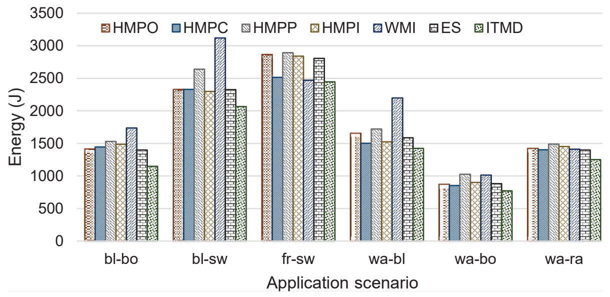

Fig. 9. Comparison of proposed approach with reported approaches for two active applications.

savings while meeting performance requirements compared to *HMPO, HMPC, HMPP, HMPI* and *WMI*, respectively. Furthermore, as our approach (ITMD) applies online DVFS at regular intervals, it provides better energy savings (17%) than the exhaustive search-based approach (ES).

Moreover, Fig. 8 shows the adaptiveness of the proposed online DVFS technique to workload variations for the application *fr*. A high MRPI leads to scaling down the frequency, thereby *ITMD* approach minimizes the power consumption, whereas *HMPO* and *HMPC* runs at max frequency. This is due to the fact that whilst the application is memory intensive, it places a high load on the processor cores as far as the load measured by the kernel is concerned. Therefore, these select the highest frequency even if it does not offer improvement in performance.

In case of multiple-application scenario, at a given moment two or more active applications will be contending for resources to meet their requirements. Such scenarios can be observed in a mobile phone where user tries to run more applications at the same time, e.g., mp3-decoder to listen to music and jpeg-decoder to view an image. A set of two applications from Table 2 are considered to stress on effectiveness of the adopted approach in choosing resources and *V-f* pair for minimizing the energy consumption while meeting each application performance requirement. Due to limited resource availability and contention, the energy savings are comparatively less than the single-application scenario. Fig. 9 gives the energy consumption for various approaches. On an average, proposed approach achieves 13%, 14%, 10%, 20%, 15%, and 23% energy savings while meeting performance requirements compared to *ES, HMPO, HMPC, HMPP, HMPI* and *WMI*, respectively. Moreover, chosen resource combinations are presented in row *two* of Table 4.

To further validate the ability of the proposed approach to adapt to concurrent execution of multiple applications, three-application scenario is also considered. Increase in number of active applications will lead to reduced solution

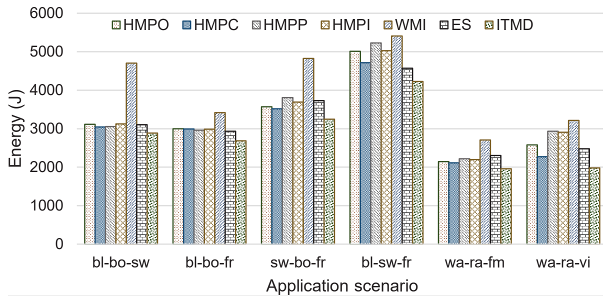

Fig. 10. Comparison of proposed approach with reported approaches for three active applications.

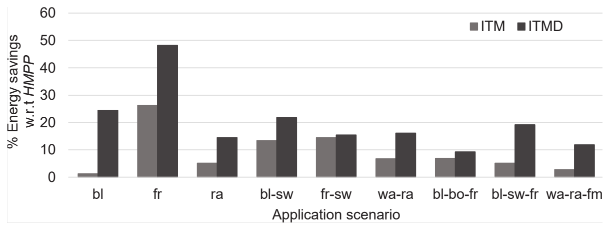

Fig. 11. Percentage of energy savings achieved by proposed *ITM* and *ITMD* respectively.

space for choosing an energy efficient thread-to-core mapping. As mentioned before, it is caused by the resource constraints (see Table 4 for resource combination) and increased contention due to concurrent workloads and demand for meeting their requirements. Furthermore, the online DVFS will have a little choice to scale down the frequency as it has to satisfy the performance requirement of different dynamic workloads (e.g. compute and memory). This results in decreased energy savings compared to single and two-application scenarios. Fig. 10 presents the comparison of adopted methodology with various previous techniques. On an average, proposed technique achieves 11%, 12%, 9%, 16%, 14%, and 30% energy savings while meeting performance requirements compared to *EC, HMPO, HMPC, HMPP, HMPI* and *WMI*, respectively.

The four and more applications scenario seems to be not feasible because of high resource contention, leading to not meeting given requirements (some applications were terminated by *out of memory (OOM) killer* daemon when multiple memory intensive applications are run). It is explained further in the following section. On an average the adopted approach achieves energy savings up to 33% compared to existing techniques.

5.2.2 *Breakdown of Energy Savings by Our Mapping and DVFS Approaches*

Individual contribution of the thread-to-core mapping (*ITM*) and online DVFS in energy savings is computed by disabling and enabling DVFS respectively. Further, percentage energy savings are calculated by comparing against the *HMPP*, as shown in Fig. 11 for different application scenarios. On an average *ITM* achieves energy savings of 9% w.r.t *HMMP*. Further, when proposed online DVFS is applied on top of *ITM* (*ITMD*), an extra 11% of energy savings is obtained. It can be observed from Fig. 8 that, the workload varies over time, for example from low MRPI to high MRPI (at 8^{th} and 10^{th} time intervals). As the online DVFS is applied at regular intervals, the proposed approach

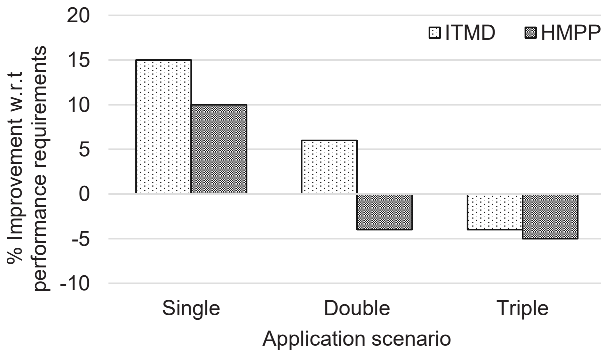

Fig. 12. Performance improvement/degradation of the adopted approach and *HMPP*.

exploits these variations to achieve energy efficiency even for a compute-intensive application.

5.2.3 Performance

As discussed earlier, performance requirements are defined for each application. The proposed approach always tries to meet the performance requirement of each application, i.e. finishing the execution within the stipulated time. To validate the adaptability of the proposed approach to the performance requirements, the achieved performance is compared against the given performance requirement, computed as percentage improvement, for all the application scenarios. The average percentage improvement in each scenario is presented in Fig. 12 in comparison with *HMPP* (performance requirements-unaware), as it maximizes the performance. The figure shows that *ITMD* always outperforms *HMPP* even when there is a high contention due to more active applications (e.g. three-application scenario). Moreover, in some cases, the adopted technique achieves up to 15% improvement over given performance requirements, whereas *HMPP* achieves 10% improvement. Additionally, the following observation can be made from Fig. 12. As the number of active applications increases, meeting high performance requirements is not feasible (see three-application scenario in Fig. 12) due to resource constraints and interference. Therefore, choosing a low performance requirement or a platform with more resources may guarantee meeting the requirements while running higher number of active applications.

To further substantiate the need for performance requirements-aware approaches, we recorded the number of violations by disabling the performance requirements-aware property of the proposed approach, resembling the technique presented in [12]. As a result, the mapping algorithm produces thread-to-core mappings that minimize the total energy consumption, which may not satisfy the performance constraints. For single application scenario, the average percentage of performance requirement-violating mappings are nearly zero. This is because, using all the cores (4L and 4B) leads to minimum energy and better performance for all the applications (except for *fr* (4L)). In case of multiple applications executing concurrently, performance violations are significantly high. The average percentage of performance requirement-violating mappings are 98.2% and 99.6% for two- and three-application scenarios, respectively.

5.3 Workload Prediction

The accuracy of the predicted workload as compared to the actual workload of the prior time intervals depends on the smoothing factor γ (9). The optimal value of γ was experimentally obtained by sweeping it between 0.1 and 1, and observing the corresponding workload miss-predictions (under/over) for various application workloads. A value of 0.6 is used for all the experiments as it resulted in relatively accurate workload prediction. Fig. 13 shows the actual and predicted values for three different application scenarios along with the percentage root mean square error (that is up to 2.4%). The figure shows that the prediction slightly goes up with the number of active applications, which is due to increased dynamic workload variations. To improve the accuracy in such cases, we will look into better workload prediction techniques in the future.

5.4 Overheads of the Proposed Approach

5.4.1 Run-time Overhead

The run-time overhead of the adopted approach includes time for finding thread-core-mapping (T_{map}) and *V-f* pair ($T_{V\text{-}f}$), which can be represented as,

$$T_o = T_{map} + T_{V\text{-}f} \tag{10}$$

$$T_{V\text{-}f} = \frac{T_{ex} - r * len * T_s}{T_s} * [VfS_o + PMC_o + Proc_o] \tag{11}$$

where T_o, T_{ex}, r, VfS_o, PMC_o, and $Proc_o$, represent total overhead, execution time, number of times the adaptation is paused, overheads associated with *V-f* switching, PMC collection and remaining processing steps (involving len and T_s, shown in Algorithm 2), respectively. T_{map} depends on the implementation of the Algorithm 1, in our case it is up to 1.6 ms (averaged over various application scenarios). Moreover, $T_{V\text{-}f}$ is about 300 μs, which is 0.15% of T_s (200 ms). Fig. 14 illustrates the total run-time overhead, computed as percentage of total execution time, for eight application scenarios. The run-time overhead for application scenario *bl-bo-fr*, having a long execution time of 1053 sec is ~0.17%, which is very minimal. Whereas, commonly used learning-based approaches have significant overheads (up to 216 sec for learning and 1 sec for subsequent stages) for a single-application scenario [16]), which gets further aggravated by dynamic workload variations causing frequent re-learning. Therefore, the scalability of such approaches in comparison to the proposed technique is limited for multi-core platforms executing multiple multi-threaded applications concurrently.

5.4.2 Offline Analysis Overhead

As discussed earlier, the profiled data of each application contains performance (1/execution time), energy consumption, number of big, and number of LITTLE cores for each design point. The total number of design points for each application is 24, which results in a small storage overhead of 770 bytes. The energy overhead due to storing of profiled data is already included in the energy consumption values reported in the previous sections.

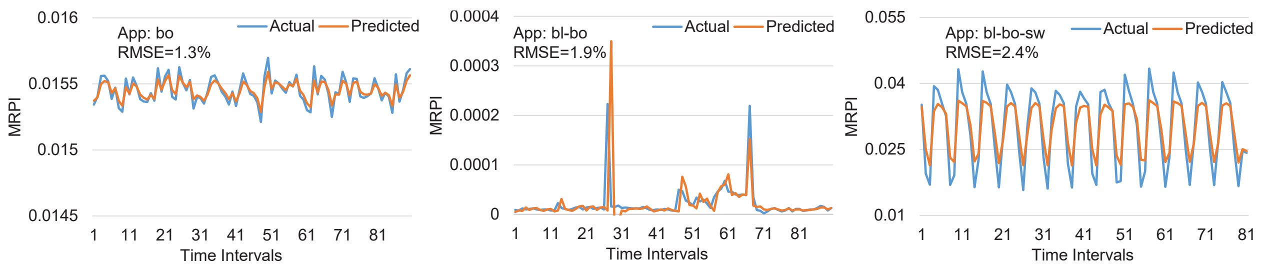

Fig. 13. Workload prediction using EWMA for three different application scenarios - one, two and three active applications (left to right).

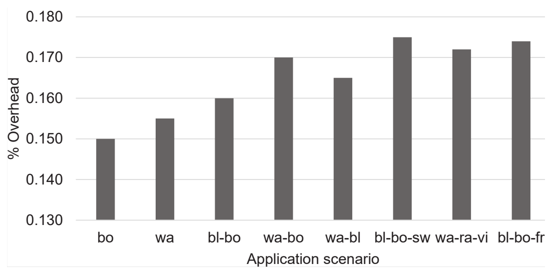

Fig. 14. Run-time overhead of the proposed approach.

6 Conclusions

We proposed a run-time management methodology for concurrently executing multi-threaded applications on a heterogeneous multi-core system. It uses the knowledge from design time analysis for efficient thread-to-core mapping and workload classification through MRPI to make run-time decisions. Furthermore, it also employs workload selection and prediction techniques for pro-active *V-f* control and online performance observation and compensation to adapt to the dynamic variations. Validation on Odroid-XU3 platform for various application scenarios shows an average improvement up to 33% in energy consumption compared to the existing approaches while achieving up to 15% performance improvement over given performance requirements. The advances reported in this paper are important contributions towards the development of future energy efficient, feature rich embedded systems with heterogeneous many-cores. In future, we will look into per-core DVFS techniques which allow to control the *V-f* level of each core separately.

Acknowledgement

This work was supported in parts by the EPSRC Grant EP/L000563/1 and the PRiME Programme Grant EP/K034448/1 (www.prime-project.org). Experimental data used in this paper can be found at DOI:10.5258/SOTON/D0249 (https://doi.org/10.5258/SOTON/D0249).

References

[1] A. K. Singh, M. Shafique, A. Kumar, and J. Henkel, “Mapping on multi/many-core systems: survey of current and emerging trends,” in *Proceedings of the 50th Annual Design Automation Conference*. ACM, 2013, p. 1.

[2] B. Khemka, R. Friese, S. Pasricha, A. A. Maciejewski, H. J. Siegel, G. A. Koenig, S. Powers, M. Hilton, R. Rambharos, and S. Poole, “Utility maximizing dynamic resource management in an oversubscribed energy-constrained heterogeneous computing system,” *Sustainable Computing: Informatics and Systems*, vol. 5, pp. 14–30, 2015.

[3] P. Greenhalgh, “big.LITTLE processing with ARM cortex-a15 & cortex-a7,” *ARM White paper*, pp. 1–8, 2011.

[4] D. N. Truong, W. H. Cheng, T. Mohsenin, Z. Yu, A. T. Jacobson, G. Landge, M. J. Meeuwsen, C. Watnik, A. T. Tran, Z. Xiao *et al.*, “A 167-processor computational platform in 65 nm CMOS,” *IEEE Journal of Solid-State Circuits*, vol. 44, no. 4, pp. 1130–1144, 2009.

[5] A. Prakash, H. Amrouch, M. Shafique, T. Mitra, and J. Henkel, “Improving mobile gaming performance through cooperative CPU-GPU thermal management,” in *Design Automation Conference (DAC), 2016 53nd ACM/EDAC/IEEE*. IEEE, 2016, pp. 1–6.

[6] B. Donyanavard, T. Mück, S. Sarma, and N. Dutt, “SPARTA: runtime task allocation for energy efficient heterogeneous many-cores,” in *Proceedings of the Eleventh IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis*. ACM, 2016, p. 27.

[7] “Odroid-XU3,” www.hardkernel.com/main/products.

[8] R. A. Shafik, S. Yang, A. Das, L. A. Maeda-Nunez, G. V. Merrett, and B. M. Al-Hashimi, “Learning transfer-based adaptive energy minimization in embedded systems,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 35, no. 6, pp. 877–890, 2016.

[9] C. Bienia and K. Li, “Parsec 2.0: A new benchmark suite for chip-multiprocessors,” in *Proceedings of the 5th Annual Workshop on Modeling, Benchmarking and Simulation*, vol. 2011, 2009.

[10] R. Cochran, C. Hankendi, A. K. Coskun, and S. Reda, “Pack & cap: adaptive DVFS and thread packing under power caps,” in *Proceedings of the 44th annual IEEE/ACM international symposium on microarchitecture*. ACM, 2011, pp. 175–185.

[11] H. Sasaki, S. Imamura, and K. Inoue, “Coordinated power-performance optimization in manycores,” in *Parallel Architectures and Compilation Techniques (PACT), 2013 22nd International Conference on*. IEEE, 2013, pp. 51–61.

[12] A. Aalsaud, R. Shafik, A. Rafiev, F. Xia, S. Yang, and A. Yakovlev, “Power-aware performance adaptation of concurrent applications in heterogeneous many-core systems,” in *Proceedings of the 2016 International Symposium on Low Power Electronics and Design*. ACM, 2016, pp. 368–373.

[13] J. Ma, G. Yan, Y. Han, and X. Li, “An analytical framework for estimating scale-out and scale-up power efficiency of heterogeneous manycores,” *IEEE Transactions on Computers*, vol. 65, no. 2, pp. 367–381, 2016.

[14] D. Stamoulis and D. Marculescu, “Can we guarantee performance requirements under workload and process variations?” in *Proceedings of the 2016 International Symposium on Low Power Electronics and Design*. ACM, 2016, pp. 308–313.

[15] V. Spiliopoulos, G. Keramidas, S. Kaxiras, and K. Efstathiou, “Power-performance adaptation in intel core i7,” 2011.

[16] A. K. Singh, C. Leech, K. R. Basireddy, B. M. Al-Hashimi, and G. V. Merrett, “Learning-based run-time power and energy management of multi/many-core systems: Current and future trends,” in *Journal of Low Power Electronics (JOLPE)*, 2017.

[17] C.-H. Hsu and U. Kremer, “Compiler-directed dynamic voltage scaling for memory-bound applications,” *Technical Report DCS-TR-498, Department of Computer Science, Rutgers University*, 2002.

[18] J. Luo and N. K. Jha, “Power-efficient scheduling for heterogeneous distributed real-time embedded systems,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 26, no. 6, pp. 1161–1170, 2007.

[19] M. Qiu and E. H.-M. Sha, “Cost minimization while satisfying hard/soft timing constraints for heterogeneous embedded systems,” *ACM Transactions on Design Automation of Electronic Systems*, vol. 14, no. 2, p. 25, 2009.

[20] L. K. Goh, B. Veeravalli, and S. Viswanathan, "Design of fast and efficient energy-aware gradient-based scheduling algorithms heterogeneous embedded multiprocessor systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 20, no. 1, pp. 1–12, 2009.

[21] K. Ma, X. Li, M. Chen, and X. Wang, "Scalable power control for many-core architectures running multi-threaded applications," in *ACM SIGARCH Computer Architecture News*, vol. 39, no. 3. ACM, 2011, pp. 449–460.

[22] A. Weissel and F. Bellosa, "Process cruise control: event-driven clock scaling for dynamic power management," in *Proceedings of the 2002 international conference on Compilers, architecture, and synthesis for embedded systems*. ACM, 2002, pp. 238–246.

[23] L. C. Singleton, C. Poellabauer, and K. Schwan, "Monitoring of cache miss rates for accurate dynamic voltage and frequency scaling," in *Electronic Imaging*. International Society for Optics and Photonics, 2005, pp. 121–125.

[24] A. Nabina and J. L. Nunez-Yanez, "Adaptive voltage scaling in a dynamically reconfigurable FPGA-based platform," *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, vol. 5, no. 4, p. 20, 2012.

[25] K. Van Craeynest, A. Jaleel, L. Eeckhout, P. Narvaez, and J. Emer, "Scheduling heterogeneous multi-cores through performance impact estimation (PIE)," in *ACM SIGARCH Computer Architecture News*, vol. 40, no. 3. IEEE Computer Society, 2012, pp. 213–224.

[26] E. Del Sozzo, G. C. Durelli, E. Trainiti, A. Miele, M. D. Santambrogio, and C. Bolchini, "Workload-aware power optimization strategy for asymmetric multiprocessors," in *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2016*. IEEE, 2016, pp. 531–534.

[27] S. Ghiasi, T. Keller, and F. Rawson, "Scheduling for heterogeneous processors in server systems," in *Proceedings of the 2nd conference on Computing frontiers*. ACM, 2005, pp. 199–210.

[28] M. Becchi and P. Crowley, "Dynamic thread assignment on heterogeneous multiprocessor architectures," in *Proceedings of the 3rd conference on Computing frontiers*. ACM, 2006, pp. 29–40.

[29] J. Chen and L. K. John, "Efficient program scheduling for heterogeneous multi-core processors," in *Proceedings of the 46th Annual Design Automation Conference*. ACM, 2009, pp. 927–930.

[30] D. Koufaty, D. Reddy, and S. Hahn, "Bias scheduling in heterogeneous multi-core architectures," in *Proceedings of the 5th European conference on Computer systems*. ACM, 2010, pp. 125–138.

[31] D. Shelepov, J. C. Saez Alcaide, S. Jeffery, A. Fedorova, N. Perez, Z. F. Huang, S. Blagodurov, and V. Kumar, "HASS: a scheduler for heterogeneous multicore systems," *ACM SIGOPS Operating Systems Review*, vol. 43, no. 2, pp. 66–75, 2009.

[32] "ARM big.LITTLE Technology," http://www.arm.com/.

[33] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta, "The SPLASH-2 programs: Characterization and methodological considerations," in *Computer Architecture, 1995. Proceedings., 22nd Annual International Symposium on*. IEEE, 1995, pp. 24–36.

[34] "Mediatek X20," http://www.96boards.org/product/mediatek-x20/.

[35] S. Sinha, J. Suh, B. Bakkaloglu, and Y. Cao, "Workload-aware neuromorphic design of the power controller," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 1, no. 3, pp. 381–390, 2011.

[36] S. Eranian, "Perfmon2: a flexible performance monitoring interface for linux," in *Proc. of Ottawa Linux Symposium*. Citeseer, 2006, pp. 269–288.

[37] G. Keramidas, V. Spiliopoulos, and S. Kaxiras, "Interval-based models for run-time dvfs orchestration in superscalar processors," in *Proceedings of the 7th ACM international conference on Computing frontiers*. ACM, 2010, pp. 287–296.

[38] S. Park, J. Park, D. Shin, Y. Wang, Q. Xie, M. Pedram, and N. Chang, "Accurate modeling of the delay and energy overhead of dynamic voltage and frequency scaling in modern microprocessors," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 32, no. 5, pp. 695–708, 2013.

[39] A. Das, A. Kumar, B. Veeravalli, R. Shafik, G. Merrett, and B. Al-Hashimi, "Workload uncertainty characterization and adaptive frequency scaling for energy minimization of embedded systems," in *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2015*. IEEE, 2015, pp. 43–48.

[40] K. Yu, D. Han, C. Youn, S. Hwang, and J. Lee, "Power-aware task scheduling for big. little mobile processor," in *SoC Design Conference (ISOCC), 2013 International*. IEEE, 2013, pp. 208–212.

[41] "Linux-governors." [Online]. Available: https://www.kernel.org/doc/Documentation/cpu-freq/governors.txt

Basireddy Karunakar Reddy received his M.Tech. degree in Microelectronics and VLSI from Indian Institute of Technology (IIT), Hyderabad, India in 2015. He is a Ph.D. student in Electronic and Electrical Engineering at the University of Southampton, UK. His current research interests include design-time and run-time optimization of performance and energy in multi-core heterogeneous systems.

Amit Kumar Singh (M09) received the B.Tech. degree in Electronics Engineering from Indian Institute of Technology (Indian School of Mines), Dhanbad, India, in 2006, and the Ph.D. degree from the School of Computer Engineering, Nanyang Technological University (NTU), Singapore, in 2013. He was with HCL Technologies, India for year and half before starting his PhD at NTU, Singapore, in 2008. He worked as a post-doctoral researcher at National University of Singapore (NUS) from 2012 to 2014, at University of York, UK from 2014 to 2016 and at University of Southampton, UK from 2016 to 2017. He is currently working as a Lecturer at University of Essex, UK. His current research interests include system level design-time and run-time optimizations of 2D and 3D multi-core systems with focus on performance, energy, temperature, and reliability. He has published over 50 papers in the above areas in leading international journals/conferences. Dr. Singh was the receipt of ISORC 2016 Best Paper Award, PDP 2015 Best Paper Award, HiPEAC Paper Award, and GLSVLSI 2014 Best Paper Candidate. He has served on the TPC of IEEE/ACM conferences like ISED, MES, NoCArc, ESTIMedia and DATE.

Dwaipayan Biswas Dwaipayan Biswas obtained his MSc in System on Chip (SoC) and PhD from the University of Southampton, UK in 2011 and 2015 respectively. He is presently a Research fellow working on embedded systems and biomedical signal processing at University of Southampton.

Geoff Merrett (GSM06-M09) received the B.Eng. degree (Hons.) in electronic engineering and the Ph.D. degree from the University of Southampton, Southampton, U.K., in 2004 and 2009, respectively.

He is currently an Associate Professor in electronic systems with the University of Southampton. His current research interests include low-power and energy harvesting aspects of embedded & mobile systems. He has published over 100 articles in journals/conferences in the above areas.

Dr. Merrett was the General Chair of the Energy Neutral Sensing Systems Workshop from 2013 to 2015. He is a fellow of the The Higher Education Academy.

Bashir M. Al-Hashimi (M99-SM01-F09) is an ARM Professor of Computer Engineering, Dean of the Faculty of Physical Sciences and Engineering, and the Co-Director of the ARM-ECS Research Centre, University of Southampton, Southampton, U.K. He has published over 380 technical papers. His current research interests include methods, algorithms, and design automation tools for low-power design and test of embedded computing systems. He has authored or co-authored five books and has graduated 35 Ph.D. students.