Application Control and Monitoring in Heterogeneous Multiprocessor Systems

Charles Leech, Graeme M. Bragg, Domenico Balsamo, Eduardo Wachter,
Geoff V. Merrett and Bashir M. Al-Hashimi
School of Electronics and Computer Science, University of Southampton, UK. Email: c.leech@soton.ac.uk

***Abstract*—Multiprocessor systems provide both high-performance and energy-efficient execution of applications on mobile and embedded systems under dynamic workload requirements, and can provide increased lifetime for devices in energy-constrained environments. However, their increasing complexity means that management at runtime has become a non-trivial task, especially in heterogeneous multiprocessor systems. In addition, there is no standardised mechanism to expose and manage the sources of control and monitoring from within applications and hardware resources at runtime. This paper presents an analysis of applications, platforms and runtime management approaches to motivate the need for a standardised framework that enables fully application- and platform-agnostic runtime management. The exposure of application controls and requirements through the presented framework is demonstrated with a stereo matching algorithm, including runtime management of multi-threading and frequency scaling on the 61-core Xeon Phi platform. In addition, the trading of application parameters, such as throughput and accuracy, is demonstrated within the framework using a runtime controller on the Odroid-XU3 platform. An open-source implementation of this framework has been released.** [1]

***Keywords*—Heterogeneous systems, application-awareness, runtime management, software frameworks.**

I. Introduction

Mobile and embedded systems have turned to multiprocessor system-on-chip (MPSoC) designs in order to increase platform performance. Initially, these systems were homogeneous architectures consisting of a group of identical cores. However, heterogeneous multiprocessor (HMP) systems have emerged as an alternative solution to meet higher performance requirements and support diversity amongst a range of next-generation applications. This is achieved by providing specialised hardware resources that target specific application behaviour and present a trade-off between power and performance [1]–[3]. As the computational power of these systems increases, a greater number of challenges can be addressed across a diverse range of fields. Applications from domains including computer vision and machine learning can now be deployed directly into embedded systems to provide high-quality and low-latency solutions to real-world challenges such as autonomous driving, healthcare monitoring, robotics and security [4]–[6].

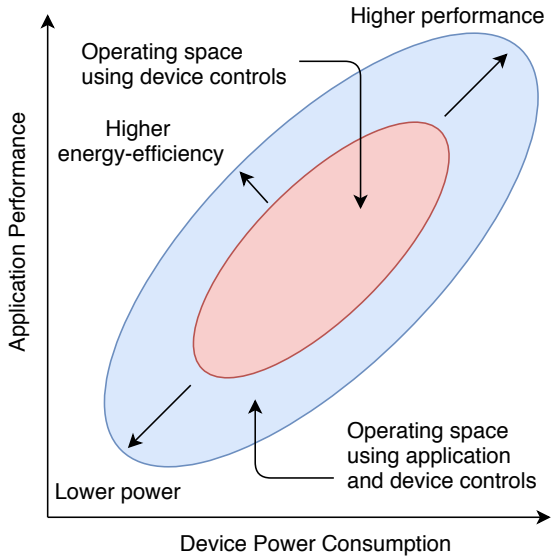


Fig. 1. The direct control and monitoring of applications at runtime expands the available operating space in HMPs to higher performance, lower power and more energy-efficient points.

However, energy-saving opportunities exist in how software applications currently execute on HMP platforms. HMPs introduce additional complexity and therefore management processes are required to maximise energy-efficiency under changing constraints. In addition, to exploit the capabilities of these systems, applications have becoming increasingly elaborate, with multiple programming models and libraries implemented to access the specialised hardware resources. This has created a range of adjustable parameters that must be tuned at design-time or runtime to optimise their behaviour. Furthermore, applications are being deploying in increasingly-constrained or dynamic environments, therefore they must be capable of adapting their behaviour to maintain system energy-efficiency. As a result, the proactive optimisation of application performance is a key research challenge, especially the management and control of applications at runtime on embedded HMP systems [7], [8].

One way in which this can be addressed is by the exposure and adaptation of tunable parameters from the application, in

This work was supported by the PRiME programme grant EP/K034448/1 (http://www.prime-project.org) and EPSRC grant EP/L000563/1.

Data supporting the results presented in this paper is available at https://doi.org/10.5258/SOTON/D0564.

[1]Available at: https://github.com/PRiME-project/PRiME-Framework

978-1-5386-3344-1/17/$31.00

order to control and monitor its behaviour at runtime. This is illustrated in Fig. 1 as the expansion from the red region to the blue region. The red region represents an operating space using only device controls, while the blue region represents the expanded operating space that is achieved through the addition of application controls. Additional control knobs have the potential to enable greater performance and power scaling as well as providing finer control over the exact operating point of the application. For example, introducing dynamic control of multi-threading can allow the application to be scaled down when or tuning the precision of arithmetic operations. Applying runtime management algorithms to optimise this operating space presents a solution that enables the optimisation and trade-off between computational quality, application throughput and energy efficiency for applications with dynamic requirements.

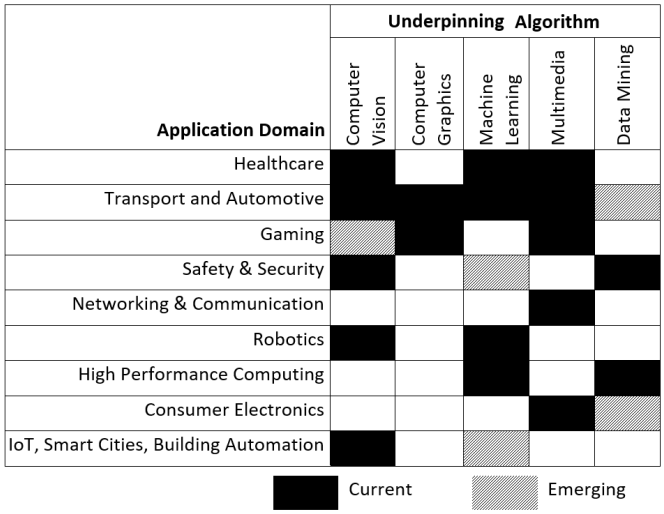
This approach can be generalised through a standardised software framework and the use of an application programming interface (API), to enable the exposure of control and monitoring sources from any application. However, current frameworks do not allow the simultaneous monitoring and control of both hardware components and applications at runtime [9]–[13]. Moreover, most existing frameworks do not inherently support HMP platforms or support the management of concurrent applications. Therefore, there exists no holistic approach to exposing sources of control and monitoring within applications in a standard way, which is extensible to any application-management scenario.

This paper presents the first fully application- and platform-agnostic framework for runtime management approaches that control and optimise software applications and hardware resources. The novel contributions of this work are:

1) analysis of emerging applications for HMP systems, runtime management approaches and existing frameworks to motivate the need for a standardised framework;
2) design and implementation of an application- and platform-agnostic framework for application management in HMP systems;
3) evaluation of how the exposure of sources of control and monitoring increases energy-efficiency and enables the trade-off of application requirements.

The presented framework is demonstrated with a stereo matching application, a process for depth estimation, on the 61 core Intel Xeon Phi co-processor. This highlights how application control and monitoring can enable greater power and performance scaling. A run-time management approach, which achieves a required performance threshold whilst minimising power consumption, is used to control multi-threading in the application and perform frequency scaling of the platform. In addition, a runtime controller is developed, to demonstrate that application parameters, such as throughput and accuracy, can be traded through the framework to optimise overall quality of service and minimise power consumption. This experimentation uses the Odroid-XU3 HMP platform. An open-source C++ implementation of the framework and its API has also been released.[1]

TABLE I
APPLICATION DOMAINS AND UNDERPINNING ALGORITHMS FOR HMP.



The remainder of the paper is organised as follows: Section II presents analysis of applications and benchmarks for HMPs. This highlights the importance of application-aware runtime management approaches, which are presented in Section III, along with existing frameworks that enable runtime management. The proposed framework for application control and monitoring is presented in Section IV. Experimental results of how application control and monitoring can increase system energy-efficiency, as well as a demonstration of trade-offs between application requirements, are presented in Section V. Finally, Section VI concludes the paper.

II. HMP Applications and Platforms

When considering a wide range of applications, it can be observed that each application domain relies on similar underlying methods or algorithms. For example, applications for automotive driver assistance, healthcare monitoring based on distributed webcams, or robotic sensing using stereo cameras are all built upon computer vision algorithms. In a similar manner, applications for object detection, image classification, speech recognition, or pedestrian detection and traffic in autonomous driving are all built upon machine learning methods such as neural networks. These applications contain similar patterns of computation and communication based on the similar data structures and algorithms that they employ.

Table I shows the current and emerging algorithms that underpin particular application domains. These algorithms have different resource requirements that cannot be satisfied with homogeneous multi-core processors. For this reason, heterogeneous multi-core platforms, which contain processors with differing capabilities, can offer significant advantage in terms of performance when facing complex and dynamic applications.

Heterogeneous multiprocessor platforms can be classified into two broad categories: *performance* and *functional* hetero-

geneity. The first relates to cores with the same instruction-set architecture (ISA) but different power-performance characteristics. An available HMP system that relies on this is the ARM big.LITTLE, which incorporates high-performance out-of-order ARM Cortex-A15 cores and low-power in-order ARM Cortex-A7 cores. Functional heterogeneity relates to cores with different characteristics such as general-purpose CPUs, graphics processing units (GPUs), digital signal processors (DSPs), field-programmable gate arrays (FPGAs) and various hardware accelerators. For example, the 61-core Intel Xeon Phi co-processor can be integrated into a functional heterogeneous system as an accelerator. The Samsung Exynos 5422 is an example of combining both types of heterogeneity into a single HMP system-on-chip (SoC), with two quad-core performance-heterogeneous CPU clusters and a Mali-T624 GPU, which is currently used in modern smartphones to increase both the energy-efficiency and performance capabilities of a system through functional heterogeneity.

Both categories require the management and control of hardware settings such that emerging applications can exploit the capabilities of these systems at runtime, with adjustable parameters that can be tuned to optimised their behaviour. Moreover, due to the differences in ISA, the management of functionally-heterogeneous systems requires advanced parallel programming models such as CUDA or OpenCL, so that applications can be executed across heterogeneous processing elements. As a result, the proactive optimisation of application performance for system energy-efficiency is a key research challenge. Run time management is a solution to this challenge that enables optimization of, and trade-off between, application throughput, computation quality and energy efficiency with varying requirements.

III. Existing Runtime Management Approaches and Frameworks

Over the past few years, different runtime management approaches have been proposed to optimize system behaviour while satisfying application requirements. These include dynamic voltage and frequency scaling (DVFS) [2], per-core power gating [14], dynamic task mapping and thread migration [3]. Runtime managers (RTMs) typically rely on the exposure of dynamic knobs and monitors, which provide a mechanism to communicate with the application and platform. Knobs allow the tuning of application and hardware parameters by the RTM, while monitors enable the observation of application behaviour (i.e. application requirements) and the measurement of hardware properties by the RTM [4], [9], [10], [15]. In addition, knobs and monitors can been used to explore application-device trade-offs, such as throughput-power [11] and precision-throughput [7], and locate optimal operating points for applications [16].

While RTMs are typically designed to address general challenges, such as energy-efficiency or thermal management in embedded systems, they are largely implemented on specific platforms or with specific classes of application, *e.g.*, multimedia [17] or image processing [5]. Hence, they cannot be easily ported to different applications and platforms. Benchmarks are typically used to assess relative performance and measure specific aspects of RTMs and hardware platforms. A benchmark is a set of programs whose execution results provide the evidence of the computational performance of a platform. A large number of benchmark suites have been implemented over the past few years to measure different aspects and evaluate the performance of these systems. Most target uni-processor systems (e.g. MiBench), or are only commercial available (e.g., MultiBench by EEMBC). Some notable benchmarks for multiprocessor-based systems are also available for both performance-heterogeneous systems such as ParMiBench and PARSEC, and functionally-heterogeneous systems such as Rodinia and Parboil which rely on on CUDA and/or OpenCL to execute on GPUs.

However, benchmarks do not typically expose application requirements (i.e. error, accuracy or certainty) in addition to performance through a mechanism such as knobs and monitors. This limits the range of optimization opportunities of run-time management approaches. As a result, exposing more than just application performance gives more optimisation opportunities to the RTM. One way in which this can be achieved is through the exposure and adaptation of tunable parameters, from the application to the RTM, through a standardised framework interface.

Several frameworks have been proposed in the literature to address the challenge of providing cross-layer communication with knobs and monitors [10], [11], [15], [18]. The most relevant framework is the Heartbeats API [12], which provides a standardised interface for applications to communicate their current and target performance to external observers, such as an RTM. However, the Heartbeats API only allows applications to communicate their throughput (*i.e.* the heart rate), and it does not allow other types of parameters to be exposed, such as accuracy and error, and prevents trade-offs between them.

Most of the frameworks are based on the Heartbeats concept and inherit its features, *e.g.* application monitors [9], [11], [13], [15], [18], [19]. In order to perform trade-offs within a single application, multiple monitors of different types must be exposed, *e.g.*, throughput, accuracy or error rate, but many frameworks do not support this functionality. In addition, for an application to meet its requirements, a target can be specified with the monitor. A small number of frameworks support this feature but they do not indicate as to whether the target is a maximisation or minimisation objective [9], [11], [13], [15], [19]. Furthermore, only some frameworks support concurrent applications, which is a common scenario in real-world systems [10], [13], [18], [19]. As a result, these approaches cannot be considered broadly applicable because they do not allow fully application-agnostic behaviour.

Current frameworks provide partial abstraction of RTM to device communication, but do not include both knobs and monitors to control hardware components at runtime [9], [12], [13], [19]. Moreover, most existing works do not operate on heterogeneous platforms, which provide both high-performance and energy-efficiency by combining conventional

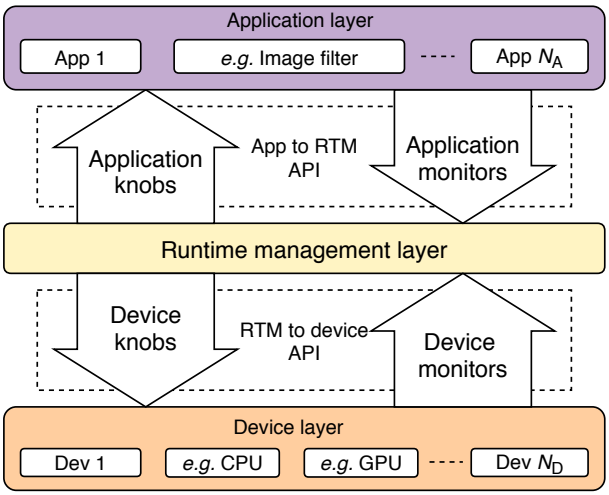


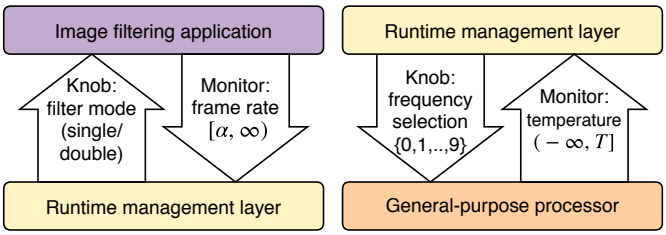
Fig. 2. Cross-layer framework and API enabling communication between the application, runtime management and device layers using knobs and monitors. Examples are given for an image filter application on an HMP system.

CPUs with other accelerators. These platforms typically increase the scalability of parallel applications and systems, and therefore they need to be managed by a framework that supports device-agnostic control. The only approach that is based on a heterogeneous platform introduces a hardware dependency in the process [9]. This restricts the cross-platform capabilities of current frameworks, meaning that they do not allow current RTM approaches to be portable across multiple platforms. Finally, many of the current frameworks are not made available, preventing their use and the extension of their features by the research community.

IV. Application- and Platform-Agnostic Framework

Section III presented a discussion of the limitations of existing frameworks. To address these limitations, this section presents an application- and platform-agnostic framework that supports heterogeneous multiprocessor systems. This framework enables the design and implementation of standardised runtime management approaches by providing unified interfaces to both applications and hardware platforms.

Fig. 2 shows how the framework is created by separating the system into three distinct layers, *i.e.* application, runtime management and device. This reduces the design complexity by enabling the runtime management layer to provide a specific service to the applications (upper layer), *e.g.*, to meet a performance requirement, whilst meeting optimisation targets by controlling the hardware resources (lower layer). These layers are connected through an API and cross-layer constructs called knobs and monitors (arrows in Fig. 2), which enable the flow of information between the layers and the control and monitoring of runtime-tunable and -observable parameters. Specifically, the application layer comprises any number of software processes, while the device layer includes the hardware and its software drivers. The runtime management layer



(a) Application to RTM interactions. (b) RTM to device interactions.

Fig. 3. Cross-layer knob and monitor use within the presented framework.

comprises an RTM responsible for the control and monitoring of the other two layers. The separation between the layers ensures portability and cross-compatibility; applications and device drivers only need to be written once to be used with any implemented RTM.

Shown as arrows in the dashed regions in Fig. 2, and in the two examples in Fig. 3, knobs and monitors enable the communication of information between the layers of the framework. Knobs allow the runtime configuration of application and device parameters, while monitors enable the measurement of hardware properties and the observation of application behaviour at runtime. Bounds are attached to both knobs and monitors, in the form of *minima* and *maxima*, which allow applications and devices to inform the runtime management process of targets and constraints. Bounds on knobs represent a range of *allowed* values while bounds on monitors represent a range of *desired* values. The monitor's value is acceptable anywhere between these two limits.

An example of application knobs and monitors is shown in Fig. 3a with an image filter application that has the option to select float or double precision for its numeric calculations at runtime. In the framework, this choice is exposed as an application knob with options $\{0, 1\}$ and controlled by the RTM. If this application has differing constraints, *e.g.* minimum latency, expressed as a time α, an application monitor with this minimum bound could be given to the RTM. In this case, the application periodically updates the current latency so that the RTM can keep this within the range $(0, \alpha]$. On the hardware side, the example of a general-purpose processor is considered in Fig. 3b within the device layer. DVFS of the CPU is achieved using a device knob with options $\{0, 1, \ldots, 9\}$, enabling the RTM to switch between ten distinct voltage-frequency pairs. Finally, to enable thermal management by the RTM, a temperature sensor is considered as an illustrative device monitor.

The combination of cross-layer knobs and monitors provides a mechanism to enable optimisation between applications and the device. The RTM's primary objective is to ensure that the true values of all application monitors remain within the specified bounds. Beyond this, it is free to optimise any unbounded monitors in either the application or device layer to meet any secondary objectives, for example to reduce power consumption. All knobs and monitors are expressed in a standardised, unit-less format to maintain application

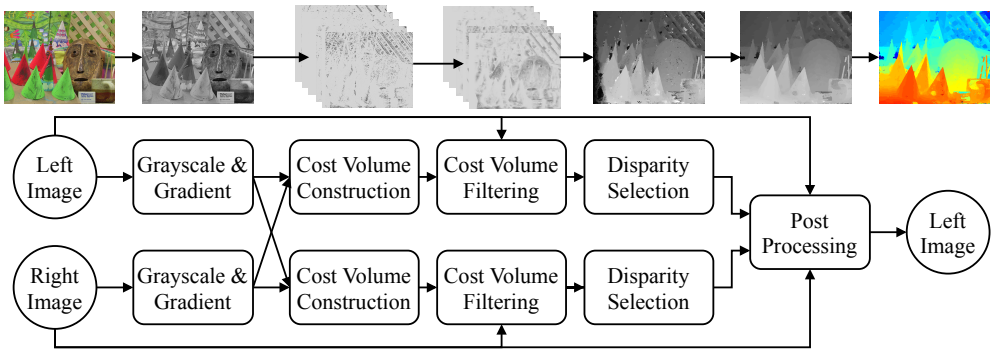


Fig. 4. The stereo matching process composed of a series of image processing stages, in order to estimate the depth of a scene from a stereoscopic image pair [4].

and device agnosticism. For example, DVFS intervals for the Odroid-XU3 (introduced in Section V-B) are translated by the device layer from MHz to a consecutive integer set as: $\{200, 300, \cdots, 2000\} \rightarrow \{0, 1, \cdots, 18\}$ before being exposed through the framework.

To provide maximal flexibility, all bounds and weights are adjustable at runtime, and no restrictions are placed on when update to these can occur. Most commonly, applications create their knobs and monitors before being executed, however no limitation is imposed on such events occurring partway through application execution. Applications can also be attached to and detached from the framework at any time, using a specific set of registration and de-registration API functions. This capability is in contrast to existing frameworks, most of which assume a constant application set, contrary to the typical use of many commercial embedded systems.

A. Application Knobs and Monitors

Integrating monitors into applications allows them to report their requirements and acceptable ranges of operation. This provides information to runtime management software about how changes to a system configuration affects the behaviour of applications. This also enables the system to be tuned holistically to reach a more energy-efficient system-wide configuration that also meets application requirements. Furthermore, exposing tunable parameters from within applications as knobs provides direct control over the application. Knobs extend the operating space of an application into additional dimensions, such as regulating the number of simultaneous threads. Knobs and monitors also enable the adaptation of the application in the presence of dynamic system constraints and tuning the application at runtime avoids a dependency on design-time information. This results in a more accurate optimisation given real-time empiric knowledge of the behaviour of the application and platform.

B. Application Monitor Weighting and Tradeoffs

Individual applications may have multiple performance requirements and these may have differing priorities. For example, an application that is aware of both its throughput and accuracy may wish to prioritise the optimisation of one over the other. These priorities may also be dynamic and vary at runtime in response to external stimuli. In the proposed framework, this priority is expressed with a numeric weight attached to each monitor. These weights instruct the RTM to expend proportional effort in optimising each monitor's value. Application monitor weights can be used to influence RTM behaviour in two different ways. Firstly, the weight determines which monitor to prioritise for optimisation. Other monitors can either be kept within their bounds or not optimised if they are below a given weight.

Alternatively, the weights can be used to proportionally set a target between the bounds of each application monitor while taking into account whether the monitor is a maximisation objective, *i.e.* throughput, or a minimisation objective, *i.e.* error. The RTM then attempts to reach the weighted targets for each of the monitors.

In addition, applications themselves can be assigned a weighting value. In a multi-tasking scenario, applications operating as foreground processes must ensure a higher level of consistency, to meet the tighter response times demanded by a user or dependent system, therefore these applications should be assigned a higher weight than those in the background. In contrast to existing frameworks, all bounds and weights are adjustable at runtime in the presented framework, to provide maximal flexibility and ensure that no restrictions are placed on when these updates can occur. A demonstration of application monitor weighting through the framework is presented in V-B using a trade-off between the throughput and accuracy of an application.

V. Experimental Results

In this section, evaluation of the application control and monitoring features of the framework are presented. The ability of the framework to expose sources of control and monitoring within applications is shown in Section V-A with a stereo matching application. The reduction in energy consumption on a multi-core platform is demonstrated using a runtime management approach that uses linear modelling to predict the power consumption of the platform and performance of the application.

The weighting of application monitors is demonstrated in Section V-B with the Jacobi iterative method. This application expose two application monitors to the RTM, throughput and error, providing a trade-off opportunity. The application is a computational kernel used for solving systems of linear equations, commonly found embedded in real-world applications.

A. Energy Saving Through Application Knobs and Monitors

This section uses a stereo matching algorithm to demonstrate the use of knobs and monitors to control application behaviour and monitor bounding to guide optimisation at runtime. Stereo matching performs depth estimation and 3D sensing and is used across many embedded applications including person counting and tracking [20], autonomous navigation [21] and mobile robotics [22]. A stereoscopic pair of images is provided as an input and a depth or disparity map is returned that encodes the real-world depth of each pixel. Fig. 4 shows how the stereo matching process is composed of a series of parallel image processing kernels. A software implementation of the application is used that supports the dynamic adaptation of the level of multi-threading used by each kernel [4]. Parallelism control is exposed through the framework as the `Threads` knob (line 1 of Table II). The performance of the application, measured in frames-per-second (FPS), is exposed through the framework as a monitor to the runtime manager. This monitor has a minimum bound on its value, shown in Table II as $\mathbb{R} \in [0.4, \infty)$. No maximum performance bound is specified, therefore it is set to ∞ to signify that it remains unbounded.

The Intel Xeon Phi 7120P platform is used to analyse the advantages of exposing knobs and monitors from applications such as stereo matching. The Xeon Phi has 61-cores, which share a common voltage island and support frequency scaling between 619 and 1240 MHz in nine fixed steps, with a corresponding voltage ranging from 0.995 V to 1.06 V. Frequency control is exposed as a device knob, as shown in Table II, and is made platform-agnostic by scaling the frequencies to consecutive integer values through the framework.

The runtime manager ensures that the performance monitor value remains within its bounds and that power consumption is optimised if the performance requirement changes. This is achieved using Multiple Linear Regression (MLR) models to predict the power and performance from a given set of knob values. The MLR models are built at runtime using training samples collected from real-time execution of the application. Once built, the RTM uses the models to make predictions and conduct performance and power scaling at runtime in response to changes in the performance requirement.

Fig. 5 shows a subset of the points which make up the power-performance operating space of the stereo matching application when executing on the Xeon Phi. Each point represents a unique combination of frequency and number of available cores. The MLR models convert this discrete operating space into a continuous model and enable interpolation between these sample points, down to the single core granularity. The combination of device and application knobs, to control both DVFS and multi-threading, allows the application to scale across a wider range of power consumptions, from 10 to 60 W, and deliver a wider performance range, from 0.06 to 0.97 FPS, than would be possible through DVFS and device knobs alone. The operating points of the application under DVFS control alone are shown in Fig. 5 with the points connected by a line (assuming all the cores are used).

TABLE II
APPLICATION- AND DEVICE-LEVEL KNOBS AND MONITORS FOR THE STEREO MATCHING APPLICATION AND XEON PHI PLATFORM.

Layer	Name	Construct	Allowed/target values
Stereo Matching	Threads	`knob`	$\mathbb{N} \in [1, \infty)$
	Performance	`mon`	$\mathbb{R} \in [0.4, \infty)$
Xeon Phi	CPU Frequency	`knob`	$\mathbb{N}\{0, 1, ..., 9\}$
	CPU Power	`mon`	$\mathbb{R} \in [0, \texttt{MAX\_POW})$

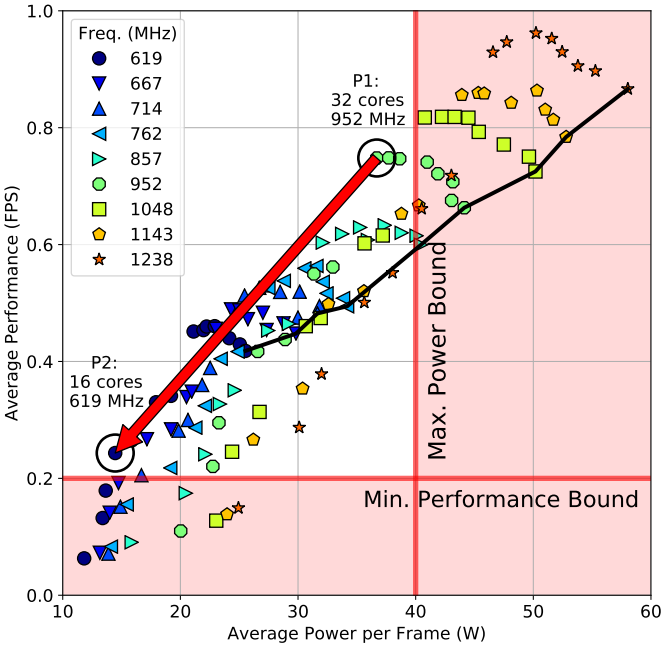


Fig. 5. Bounded operating space of the stereo matching application on the Intel Xeon Phi. Points joined by the black line indicate the operating range of using DVFS only with the task using all cores. Other points indicate unique combinations of frequency and core mapping.

Red shaded regions in Fig. 5 illustrate the process of applying bounds to monitors through the framework, with the application specifying a minimum performance of 0.2 FPS and the user indicating a maximum power consumption of 40 W. The framework allows these bounds to be adjusted at runtime. The RTM traverses the Pareto-optimal frontier of the application's operating space, whenever the bounds of the performance and power monitors change, in order to adjust the application and device knobs as required. For example, the arrow in Fig. 5 demonstrates a scaling operation that could be performed by the RTM in order to move the application to a more energy-efficient operating point in response to changes in the performance monitor bound. Starting from a high-

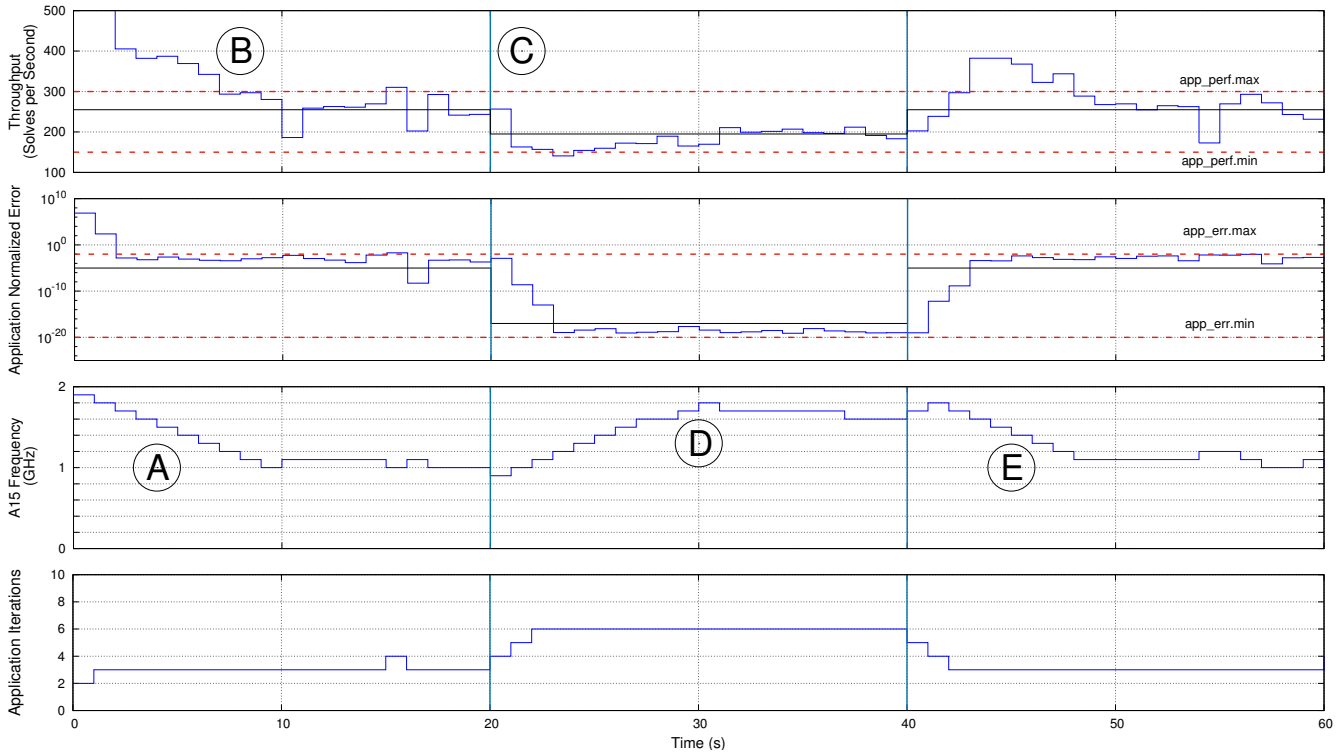


Fig. 6. Demonstration of application monitor weights for performance and error being used to influence runtime management of the Jacobi application. Vertical lines show when the monitor weights are updated by the application and black lines indicate the new target values.

performance high-power point at P1 (32 cores at 952 MHz), the frequency and the number of threads are decreased in order to transition to P2 (16 cores at 619 MHz).

B. Application Monitor Weighting and Tradeoffs

This section experimentally demonstrates the use of application monitor weighting to guide runtime manager optimisation and achieve a trade-off between performance and accuracy for an application. The application considered is the Jacobi iterative method, a mathematical algorithm used to solve the system of N linear equations $\boldsymbol{Ax} = \boldsymbol{b}$, where $\boldsymbol{A}$ is an $N \times N$ matrix and $\boldsymbol{x}$ and $\boldsymbol{b}$ are $N \times 1$ column vectors. The algorithm is commonly embedded in real-world applications as a computational kernel, which can be used for the purpose of generating an approximation to a physical process. If $\boldsymbol{A}$ is decomposed into diagonal and remainder components $\boldsymbol{D}$ and $\boldsymbol{R}$, $\boldsymbol{x}$ can be computed iteratively via (1), also shown in an element-wise fashion in (2), where k is the iteration index. Therefore, (2) can be parallelised by computing each $x_i^{(k+1)}$ independently.

$$\boldsymbol{x}^{(k+1)} = \boldsymbol{D}^{-1}\left(\boldsymbol{b} - \boldsymbol{R}\boldsymbol{x}^{(k)}\right) \tag{1}$$

$$x_i^{(k+1)} = \frac{1}{a_{ii}}\left(b_i - \sum_{j \neq i} a_{ij} x_j^{(k)}\right) \tag{2}$$

The error in the approximation, after performing K iterations of (1), is established using $\epsilon = \|\boldsymbol{Ax} - \boldsymbol{b}\|$, which represents a single solution of the application. Tuning K operates a trade-off between accuracy and computational speed. The application exposes three knobs and two monitors, presented in the first section of Table III. In the context of this experiment, $\|\boldsymbol{Ax} - \boldsymbol{b}\|$ is captured as an error monitor, with maximum bound ϵ. A throughput monitor reports the time taken to complete K iterations of the algorithm. The application is implemented as an OpenCL kernel that can be executed on CPUs, GPUs and FPGA-based accelerators. Selection of the resource to use is controlled through the `device type` knob to the application.

The Odroid-XU3 development board is used as the device layer in this experiment. The SoC is an HMP based on the ARM big.LITTLE architecture, with two quad-core CPU clusters and a GPU. The lower half of Table III summarises the three device knobs are exposed through the framework to provide DVFS control of each CPU cluster and the GPU. The platform also contains sensors to directly monitor the power consumption of each CPU cluster, the GPU and memory subsystem. These are exposed through the framework as device monitors with the bounds shown in Table III.

An RTM controller has been implemented to demonstrate how monitor weights can be used to guide the trade-off of application metrics. The controller uses the application monitor weights to set a target value for each monitor within its bounds and attempts to meet these targets with the available application and device knobs. The optimisation of device power is the secondary objective of the controller. This is

TABLE III
APPLICATION- AND DEVICE-LEVEL KNOBS AND MONITORS FOR THE JACOBI APPLICATION AND ODROID-XU3 PLATFORM.

Layer	Name	Construct	Allowed/target values
Jacobi	Iterations	knob	$\mathbb{N} \in [1, \infty)$
	Data type	knob	{`float`, `double`}
	Device type	knob	{CPU, GPU/FPGA}
	Throughput	monitor	$\mathbb{R} \in [10, \infty)$
	Error	monitor	$\mathbb{R} \in (-\infty, 1e^{-12}]$
Odroid XU3	A7 cluster freq.	knob	$\mathbb{N}\{0, 1, \ldots, 12\}$
	A15 cluster freq.	knob	$\mathbb{N}\{0, 1, \ldots, 18\}$
	GPU freq.	knob	$\mathbb{N}\{0, 1, \ldots, 6\}$
	A15 cluster power	monitor	$\mathbb{R} \in [0, \texttt{MAX\_POW})$
	A7 cluster power	monitor	$\mathbb{R} \in [0, \texttt{MAX\_POW})$
	Memory power	monitor	$\mathbb{R} \in [0, \texttt{MAX\_POW})$
	GPU power	monitor	$\mathbb{R} \in [0, \texttt{MAX\_POW})$

achieved by decrementing the frequency knob whenever possible, trading-off excess application performance or accuracy.

Fig. 6 shows how the RTM uses the weights of the throughput and error monitors to synthesise a target value within the monitor's bound. Given the bounds specified by each monitor, the RTM chooses to use the A15 cluster frequency knob and the iterations knob to meet these target values. Initially, the RTM decreases frequency (label (A) in Fig. 6) and increases iterations to meet the monitor bounds and approach the weighted target values (B), while minimising power. During the first period, the throughput monitor has a higher relative weight, leading to a target value closer to the monitor's maximum bound. At 20s, the weightings are changed (C) and the RTM increases the iterations to meet the lower error target, calculated from its higher weight. This reduces the throughput and the frequency is increased to compensate (D). When the monitor weights are updated again at 40s, the RTM reduces the number of iterations and CPU frequency as required (E).

VI. CONCLUSIONS AND FUTURE WORK

This paper has demonstrated that HMP systems can benefit from a mechanism to expose and manage the sources of control and monitoring from within applications at runtime. Their increasing complexity means that management at runtime has become a non-trivial task and energy-saving opportunities exist in how software applications currently execute on HMP systems. To address this challenge, this paper has presented a framework that allows the direct control and monitoring of software applications and supports the management of concurrent applications and heterogeneous platforms. The runtime management of a stereo matching algorithm on a multi-core platform has shown that the exposure of application controls and requirements through the framework enables greater optimisation of energy-efficiency. In addition, the trading of application parameters using weights has been demonstrated on a HMP using a runtime controller that manages application and device knobs and monitors through the framework. An open-source implementation of the framework has been released.[1] Research is ongoing to provide further validation of the framework, including experimentation with concurrent application execution and the integration of additional real-world applications.

REFERENCES

[1] S. Pagani *et al.*, *Design Space Exploration and Run-Time Adaptation for Multicore Resource Management Under Performance and Power Constraints*. Springer Netherlands, 2017, ch. 10, pp. 301–332.

[2] A. Das *et al.*, "Reinforcement Learning-based Inter- and Intra-application Thermal Optimization for Lifetime Improvement of Multicore Systems," in *ACM/EDAC/IEEE Design Automation Conf.*, 2014.

[3] B. K. Reddy *et al.*, "Inter-cluster Thread-to-core Mapping and DVFS on Heterogeneous Multi-cores," *IEEE Trans. on Multi-Scale Comput. Syst.*, vol. PP, no. 99, pp. 1–1, 2017.

[4] C. Leech *et al.*, "Runtime performance and power optimization of parallel disparity estimation on many-core platforms," *ACM Trans. on Embedded Comput. Syst. (TECS)*, vol. 17, no. 2, p. 41, 2018.

[5] S. Yang *et al.*, "Adaptive Energy Minimization of Embedded Heterogeneous Systems using Regression-based Learning," in *Int'l Workshop on Power and Timing Modeling, Optim. and Sim.*, 2015.

[6] F. Gong *et al.*, "Cooperative DVFS for Energy-efficient HEVC Decoding on Embedded CPU-GPU Architecture," in *ACM/EDAC/IEEE Design Automation Conference*, 2017.

[7] X. Sui *et al.*, "Proactive Control of Approximate Programs," in *Int'l Conf. on Arch. Support for Prog. Lang. and Operating Syst.*, 2016.

[8] G. Singla *et al.*, "Predictive dynamic thermal and power management for heterogeneous mobile platforms," in *2015 Design, Automation Test in Europe Conf. Exhibition (DATE)*, 2015.

[9] S. T. Fleming and D. B. Thomas, "Heterogeneous Heartbeats: A Framework for Dynamic Management of Autonomous SoCs," in *Int'l Conf. on Field Prog. Logic and Appl.*, 2014.

[10] D. Gadioli *et al.*, "Application Autotuning to Support Runtime Adaptivity in Multicore Architectures," in *Int'l Conf. on Embedded Comput. Syst.: Architectures, Modeling, and Simulation*, 2015.

[11] H. Hoffmann *et al.*, "A Generalized Software Framework for Accurate and Efficient Management of Performance Goals," in *Int'l Conf. on Embedded Software*, 2013.

[12] H. Hoffmann *et al.*, "Application Heartbeats: A Generic Interface for Specifying Program Performance and Goals in Autonomous Computing Environments," in *Int'l Conf. on Autonomic Comput.*, 2010.

[13] A. Baldassari *et al.*, "A Dynamic Reliability Management Framework for Heterogeneous Multicore Systems," in *IEEE Int'l Symp. on Defect and Fault Tolerance in VLSI and Nanotechnology Syst.*, 2017.

[14] A. M. Rahmani *et al.*, "Reliability-Aware Runtime Power Management for Many-Core Systems in the Dark Silicon Era," *IEEE Trans. on VLSI Syst.*, vol. 25, no. 2, 2017.

[15] H. Hoffmann *et al.*, "Dynamic Knobs for Responsive Power-aware Computing," in *Int' Conf. on Arch. Supp. for Prog. Lang. and OS*, 2011.

[16] V. Vassiliadis *et al.*, "Exploiting Significance of Computations for Energy-constrained Approximate Computing," *Int'l J. of Parallel Prog..*, vol. 44, no. 5, 2016.

[17] Y. G. Kim *et al.*, "Enhancing Energy Efficiency of Multimedia Applications in Heterogeneous Mobile Multi-core Processors," *IEEE Trans. Comput.*, vol. 66, no. 11, 2017.

[18] E. Paone *et al.*, "Evaluating Orthogonality between Application Autotuning and Run-time Resource Management for Adaptive OpenCL Applications," in *IEEE Int'l Conf. on Appl.-specific Syst., Arch. and Proc.*, 2014.

[19] F. Gaspar *et al.*, "A Framework for Application-guided Task Management on Heterogeneous Embedded Systems," *ACM Trans. on Arch. and Code Optim.*, vol. 12, no. 4, 2015.

[20] A. Burbano *et al.*, "3D-sensing Distributed Embedded System for People Tracking and Counting," in *International Conference on Computational Science and Computational Intelligence (CSCI)*, Dec 2015, pp. 470–475.

[21] H. Oleynikova *et al.*, "Reactive Avoidance Using Embedded Stereo Vision for MAV Flight," in *IEEE International Conference on Robotics and Automation (ICRA)*, May 2015, pp. 50–56.

[22] S. Solak and E. D. Bolat, "Distance Estimation using Stereo Vision for Indoor Mobile Robot Applications," in *9th International Conference on Electrical and Electronics Engineering (ELECO)*, Nov 2015, pp. 685–688.