An Application- and Platform-agnostic Runtime Management Framework for Multicore Systems

Graeme M. Bragg[1], Charles Leech[1], Domenico Balsamo[1], James J. Davis[2], Eduardo Wachter[1], Geoff V. Merrett[1], George A. Constantinides[2] and Bashir M. Al-Hashimi[1]
[1]*School of Electronics and Computer Science, University of Southampton, SO17 1BJ, UK*
[2]*Department of Electrical and Electronic Engineering, Imperial College London, SW7 2AZ, UK*
gmb@ecs.soton.ac.uk, c.leech@soton.ac.uk, d.balsamo@soton.ac.uk, james.davis@imperial.ac.uk, e.weber-wachter@soton.ac.uk, gvm@ecs.soton.ac.uk, g.constantinides@imperial.ac.uk, bmah@ecs.soton.ac.uk

Keywords: Heterogeneous Systems, Runtime Management, Software Framework.

Abstract: Heterogeneous multiprocessor systems have increased in complexity to provide both high performance and energy efficiency for a diverse range of applications. This motivates the need for a standard framework that enables the management, at runtime, of software applications executing on these processors. This paper proposes the first fully application- and platform-agnostic framework for runtime management approaches that control and optimise software applications and hardware resources. This is achieved by separating the system into three distinct layers connected by an API and cross-layer constructs called knobs and monitors. The proposed framework also supports the management of applications that are executing concurrently on heterogeneous platforms. The operation of the proposed framework is experimentally validated using a basic runtime controller and two heterogeneous platforms, to show how it is application- and platform-agnostic and easy to use. Furthermore, the management of concurrently executing applications through the framework is demonstrated. Finally, two recently reported runtime management approaches are implemented to demonstrate how the framework enables their operation and comparison. The energy and latency overheads introduced by the framework have been quantified and an open-source implementation has been released[a].

---

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

1 INTRODUCTION

The management and control of hardware settings at runtime is crucial to the efficient execution of applications with varying performance requirements on embedded platforms. This has, however, become a non-trivial task for multi-core and heterogeneous embedded systems. In addition, applications have become increasingly dynamic in order to exploit the capabilities of these systems, with adjustable parameters that must be tuned to optimise their behaviour. As a result, the proactive optimisation of application performance and system energy efficiency is a key research challenge. Runtime management is a solution that enables optimisation of, and tradeoff between, quality, application throughput and energy with varying requirements.

One way in which this can be achieved is by the exposure and adaptation of tunable parameters from the application and platform through a consistent framework interface. However, the majority of current frameworks only provide a mechanism to monitor application performance, and do not allow for the simultaneous monitoring and control of hardware components and applications at runtime. Moreover, most existing frameworks do not support heterogeneous platforms, which contain processors with differing capabilities, or the management of concurrent applications.

This paper presents the first framework for fully application- and platform-agnostic runtime management that enables the simultaneous control and optimisation of software applications and hardware resources. This is achieved by separating systems into three distinct layers: application, runtime management and device. These layers are connected through cross-layer constructs called knobs and monitors, accessed through a novel application programming interface (API), which enable the flow of information between layers and the control and monitoring of

Table 1: Properties of state-of-the-art frameworks for runtime management of applications on multiprocessor systems.

Framework	Application–RTM				RTM–device		Monitor bounds	Hetero. platforms	Open source
	Knobs	Monitors	Non-temp. monitors	Multiple monitors	Knobs	Monitors			
Heartbeats (Hoffmann et al., 2010)	✗	✓	✗	✗	✗	✗	✗	✗	✓
PowerDial (Hoffmann et al., 2011)	✓	Heartbeats	✗	✗	✗	✗	✗	✗	✗
Heterogeneous Heartbeats (Fleming and Thomas, 2014)	✗	Heartbeats	✗	✗	✗	✓	✗	CPU+FPGA	✓
ARGO (Gadioli et al., 2015)	✓	✓	✓	✓	✗	✗	✗	✗	✗
AS-RTM (Paone et al., 2014)	✓	Heartbeats	✗	✓	✗	✗	✗	✗	✗
PTRADE (Hoffmann et al., 2013)	✗	Heartbeats	✗	✗	✓	✗	✗	✗	✗
DRM (Baldassari et al., 2017)	✗	Heartbeats	✗	✗	✗	✗	✗	✗	✗
BEEPS (Gaspar et al., 2015)	✗	Heartbeats	✗	✗	✓	✗	✗	✗	✗
Proposed	✓	✓	✓	✓	✓	✓	✓	✓	✓

runtime-tunable and -observable parameters. This reduces the design complexity by enabling the runtime management layer to provide a specific service to the applications, *e.g.* to meet a performance requirement, whilst meeting optimisation targets by controlling the hardware. The framework's novel features include:

- The ability to control and monitor applications and hardware simultaneously using a cross-layered approach.
- An API that provides a consistent way in which knobs and monitors are specified and monitored across applications and platforms.
- A mechanism to enable the management of concurrently executing applications and heterogeneous platforms.

Additionally, the framework enables the direct comparison of runtime management approaches and algorithms, which has not previously been possible, and simplifies runtime manager (RTM) development.

In the remainder of this paper, a survey of existing frameworks is carried out to contrast the proposed framework against the state of the art. The proposed framework is experimentally validated with a range of applications and two different types of heterogeneous platform to demonstrate its application- and platform-agnostic properties and to illustrate its ease of use. The management of two concurrently-executing applications is then demonstrated. In addition, two recently reported runtime management approaches, the first based on performance counter-driven control and the second using reinforcement learning, are implemented with the framework to demonstrate how the framework enables their operation and comparison. Finally, the energy and latency overheads of the proposed framework are quantified. An open-source C++ implementation of the framework and API has also been released.[a]

2 Related Work

Various runtime management approaches exist in the literature for optimising system behaviour, whilst satisfying application requirements. These include dynamic voltage and frequency scaling (DVFS) (Das et al., 2014; Wang et al., 2017), per-core power gating (Rahmani et al., 2017), dynamic task mapping and thread migration (Reddy et al., 2017). While RTMs are typically designed to address general challenges, such as energy efficiency or thermal management, they are largely implemented on specific platforms or with specific classes of application, *e.g.* multimedia (Kim et al., 2017) or image processing (Yang et al., 2015).

In addition, benchmarks are typically used to assess relative performance and measure specific aspects of RTMs and hardware platforms. However, they do not typically expose application requirements (*e.g.* error or accuracy) in addition to performance and this can limit the range of optimisation opportunities of runtime management approaches. Furthermore, source code for RTMs is often not released, with limited detail on implementation reported, making reproduction of results a non-trivial task. This prevents the direct comparison of approaches, with several works relying on comparison via Linux governors (Singla et al., 2015; Reddy et al., 2017).

Runtime management can be enhanced by the exposure of dynamic knobs and monitors, which provide a mechanism to communicate with the application and platform. Specifically, knobs allow the tuning of hardware and application parameters by the RTM, while monitors enable the measurement of hardware properties and the observation of application behaviour, including the setting of performance targets by the application (Hoffmann et al., 2011; Fleming and Thomas, 2014; Gadioli et al., 2015; Leech et al., 2018b). In addition, knobs and monitors can been used to explore application-device tradeoffs, such as throughput-power (Hoffmann et al., 2013) and precision-throughput (Sui et al., 2016), and locate optimal operating points for applications (Vassiliadis et al., 2016). However, runtime management lacks portability unless these knobs and monitors are exposed through a consistent interface.

Several frameworks have been proposed in the past to address the challenge of providing such interfaces. Table 1 summarises their features. The most relevant framework is the Heartbeats API (Hoffmann et al., 2010), which provides a standardised interface for single or concurrent applications to communicate their current and target performance to external observers, such as an RTM. The Heartbeats API only allows applications to communicate their throughput (*i.e.* the heart rate), therefore it does not allow other types of parameters to be exposed, such as accuracy and error (classed as non-temporal monitors in column four of Table 1), and prevents tradeoffs between them. In addition, it does not extend this interface for monitoring or control of device parameters. Most of the frameworks reported in Table 1 are based on the Heartbeats concept and inherit its features, *e.g.* application monitors (column three).

In order to perform tradeoffs within a single application, multiple monitors of different types must be exposed, *e.g.* throughput and error. Column five of Table 1 shows that Heartbeats, and most of the frameworks that rely on it, do not support this functionality. In addition, for an application to meet its requirements, a target can be specified with the monitor. However, there is no indication as to whether the target is a maximisation or minimisation objective, as listed in column eight. As a result, these approaches do not allow fully application-agnostic behaviour.

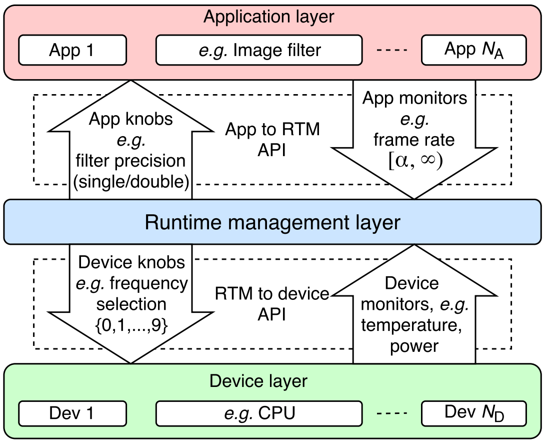

Figure 1: Cross-layer framework and API enabling communication between the application, runtime management and device layers using knobs and monitors. Examples are given for an image filter application on a CPU.

Columns six and seven show that current frameworks only provide partial abstraction of RTM to device communication, and do not include both knobs and monitors to control hardware components at runtime. Moreover, most existing works do not operate on heterogeneous platforms (column nine), which provide both high performance and energy efficiency by combining conventional 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. One framework supports a heterogeneous platform; however, it has been designed for a specific platform and introduces a hardware dependency in the process (Fleming and Thomas, 2014). This restricts the cross-platform capabilities of current frameworks, meaning that they do not allow current RTM approaches to be portable across multiple platforms.

3 Proposed Framework

To address the limitations of existing frameworks discussed in Section 2, a framework for application- and platform-agnostic runtime management of heterogeneous systems is presented. Figure 1 shows the proposed framework and how the three layers are connected by novel APIs (`App to RTM API` and `RTM to device API`). This provides consistent interfaces from an RTM to both hardware platforms and ap-

Table 2: Application-to-RTM and RTM-to-device API functions for the proposed framework.

Layer	Construct	Space	Identifier	Input(s)	Output(s)	Description
`app`	`knob`	`disc` / `cont`	`min`	`knob`, `min`	–	Update application knob's minimum allowed value
			`max`	`knob`, `max`	–	Update application knob's maximum allowed value
			`get`	`knob`	`value`	Pull application knob's current value
	`mon`		`min`	`mon`, `min`	–	Update application monitor's minimum desired value
			`max`	`mon`, `max`	–	Update application monitor's maximum desired value
			`weight`	`mon`, `weight`	–	Update application monitor's relative importance
			`set`	`mon`, `value`	–	Push application monitor's current value
`dev`	`knob`		`min`	`knob`	`min`	Pull device knob's minimum allowed value
			`max`	`knob`	`max`	Pull device knob's maximum allowed value
			`init`	`knob`	`init`	Pull device knob's initial (default) value
			`type`	`knob`	`type`	Pull device knob's type
			`set`	`knob`, `value`	–	Push device knob's current value
	`mon`		`type`	`mon`	`type`	Pull device monitor's type
			`get`	`mon`	`value`	Pull device monitor's current value and bounds

plications, which enables the design and implementation of application- and platform-agnostic runtime management approaches. As discussed in Section 2, application knobs expose tunable application parameters, *e.g.* filter precision, while monitors convey information about the behaviour of the applications, *e.g.* frame rate. Similarly, device knobs expose tunable device parameters while monitors convey information about the status of devices. Exposing knobs and monitors at both the application and device layer enable tradeoffs, *e.g.* performance-energy or accuracy-temperature, to be explored and exploited by the runtime management layer.

In addition, the proposed framework facilitates the comparison of existing RTMs as well as the management of concurrently-executing applications and heterogeneous platforms. The remainder of this section provides an overview of the technical concepts of the proposed framework and details of the novel API.

3.1 Framework Concepts

Structure: The separation of the system into the three distinct layers—*application*, *runtime management* and *device*—shown in Figure 1 reduces design complexity and provides flexibility during operation. The application layer comprises any number of software processes, while the device layer includes the hardware and its software drivers. The runtime management layer comprises an RTM responsible for the control and monitoring of the other two layers. This separation ensures portability and cross-compatibility; applications and device drivers only need to be written once to be used with any implemented RTM.

The framework can be viewed hierarchically "downwards" since, as far as knob and monitor control is concerned, applications are masters of the RTM. Applications make calls to the API, controlling the presence and configuration of each knob and monitor. Devices, meanwhile, are the RTM's slaves since they must respond to requests to set and get knob and monitor values, respectively. Thus, applications "pull" their knob settings from the RTM and "push" monitor updates, while device knobs are pushed from the RTM and monitor values pulled.

Communication: *Knobs* and *monitors*, shown in the dashed regions of Figure 1, facilitate communication between the layers. Bounds are attached to both knobs and monitors, in the form of *minima* and *maxima*, which allow applications and devices to inform an RTM of targets and constraints. Knob bounds represent a range of *allowed* values while monitor bounds represent a range of *desired* values, rather than a single target. An RTM's primary objective is to ensure that the monitor values of all applications and the device remain within their specified bounds. Beyond this, it is free to optimise any unbounded monitors in order to meet secondary objectives, *e.g.* to reduce power consumption. Minimal modification of applications is required to expose knobs and monitors through the framework.

The image filtering application shown in Figure 1 provides the option of selecting float or double precision for its numeric operations at runtime. This choice will be controlled by an RTM using an application knob with options $\{0,1\}$. If the same application requires a minimum throughput, *e.g.* expressed as a frame rate α, an application monitor with this bound can be provided. In this case, the application will periodically update the current frame rate so that the RTM can keep it within the range $[\alpha, \infty)$. On the hardware side, DVFS of the CPU is achieved via a device knob with options $\{0, 1, \cdots, 9\}$, enabling the RTM to switch between ten distinct voltage-frequency pairs. Finally, to enable thermal management by the RTM, a temperature sensor is exposed as a device monitor.

Weights: Individual applications may feature multiple performance objectives with differing priorities. For example, an application aware of both its throughput and accuracy may wish to prioritise the optimisation of one over the other. In the proposed framework, such priorities are expressed with a numeric *weight* attached to each monitor. These weights instruct the RTM to expend proportional effort in optimising each monitor's value. In a similar manner, application priority is indicated through attached weights such that, for example, a higher level of performance can be ensured by foreground processes.

Concurrency: Real-world systems commonly execute more than one application concurrently. Due to this, an RTM is required to carefully manage system resources so that each application can meet its performance targets. When considering concurrently executing applications, the framework provides a mechanism to identify and manage them simultaneously, enabling inter-application tradeoffs by the RTM.

Types: Knobs and monitors each have a *type* selectable from a discrete set of options, *e.g.* `TEMP` for a temperature monitor or `FREQ` for a frequency knob. This represents a compromise between complete agnosticism and the full provision of information. Providing "hints" to the RTM simplifies the process of determining the function of knobs and the properties represented by monitors, *e.g.* "lower power is better."

Spaces: All knob and monitor values are expressed in standardised, unitless formats to maintain application and device agnosticism. The proposed framework allows discrete- and continuous-valued versions of each knob and monitor so that appropriate optimisation processes can be used by the RTM. These *spaces* enable the translation of application-specific information into agnostic sets, as shown in Figure 1 for the ranges of the knobs and monitors. Discrete versions use signed integer values while their continuous counterparts operate using floating-point data.

Adaptability: In order 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 instead. Applications are allowed to be attached to and detached from the framework at any point during runtime. This capability is in contrast to existing frameworks, most of which assume a constant application set, contrary to the typical use of many embedded systems.

3.2 API Specification

The proposed framework is realised through novel API calls that connect the system layers of Figure 1 and enable the exposure of knobs and monitors between them in a consistent manner across applications and hardware platforms. Table 2 illustrates how the API functions are split into application (`app`) and device (`dev`) categories, with subcategories for knob (`knob`) and monitor (`mon`) interaction. Discrete- (`disc`) and continuous-valued (`cont`) versions exist across the API to indicate knob and monitor typology.

The RTM must be made aware of the allowable and desired values for knobs and monitors, respectively, in order to ensure that its optimisations have positive effects. For knobs, functions `app_knob_(disc|cont)_(min|max)()` facilitate this, letting the application indicate the range in which values can be chosen. Conversely, monitor functions `app_mon_(disc|cont)_(min|max|weight)()` allow the setting of RTM objectives, with `*_min()` and `*_max()` functions indicating desired lower and upper bounds. Where an application requires only a maximum or minimum bound, the other end of the range can be left unbounded using `(DISC|CONT)_MIN` or `(DISC|CONT)_MAX`. Intra-application weighting values between 0.0 and `CONT_MAX` can be used to indicate relative monitor importance to the RTM using `*_weight()` functions, guiding its optimisations. All of these settings can be updated during application execution if required. Functions `app_knob_(disc|cont)_get()` and `app_mon_(disc|cont)_set()` are used by the application to get the current value of a knob from the RTM and set a value for a monitor to the RTM, respectively. The timing of these actions is application-controlled.

Device-layer knobs and monitors are exposed and updated via the RTM-to-device API functions, as shown in the lower half of Table 2. Functions `dev_knob_(disc|cont)_(min|max)()` are equivalent to their application-layer counterparts, setting ranges of valid values. Additional functions `dev_knob_(disc|cont)_(type|init)()` return the type of the knob or its initial value, *i.e.* that from which the RTM starts its exploration. Type-related functions return values from defined sets and are called by the RTM using `dev_mon_(disc|cont)_type()`. The RTM uses functions `dev_knob_(disc|cont)_set()` and `dev_mon_(disc|cont)_get()` for setting device knob values and accessing monitor values and bounds from the device at runtime.

4 Evaluation

In order to demonstrate the capabilities of the framework and validate its operation, a series of experiments have been carried out. Illustrative RTMs were used where appropriate to demonstrate specific concepts. The experimental setup is discussed in Section 4.1, after which the framework's basic operation and ease of use are exemplified in Section 4.2. Application agnosticism is shown throughout this section while platform agnosticism is demonstrated in Section 4.3 with the same application-RTM pair executing on two different heterogeneous platforms. Support for concurrent applications is shown in Section 4.4, with two different applications executing on one platform The ability of the framework to enable direct comparison of RTMs is shown in Section 4.5 with two recently reported runtime management approaches. Finally, framework overheads are analysed in Section 4.6.

4.1 Experimental Setup

Two heterogeneous embedded platforms were used to demonstrate the proposed framework. The Odroid-XU3 development board, containing an ARM big.LITTLE architecture with two quad-core CPU clusters and a GPU, was used to demonstrate the ease of use of the framework, the direct comparison of RTMs and to assess overheads. The platform contains five temperature sensors to monitor the CPU and GPU and four power sensors to monitor each CPU cluster, the GPU and memory. Each of these was exposed to the framework as a device monitor. Three device knobs were exposed to provide DVFS for each CPU cluster and the GPU. Table 3 summarises the knobs and monitors of the Odroid-XU3.

A Cyclone V SoC Development Kit was used to demonstrate platform-agnostic operation of the framework. This platform includes a heterogeneous CPU-FPGA system-on-chip containing two ARM CPUs and FPGA fabric. Using OpenCL, applications can execute on either the CPUs or the FPGA.

Four different applications from the numerical and multimedia domains were used to demonstrate the application-agnostic properties of the framework.

Table 3: Device-level knobs and monitors for Odroid-XU3.

Const.	Space	Type	For	No.
`knob`	`disc`	`FREQ`	LITTLE cluster	1
	`disc`	`FREQ`	big cluster	1
	`disc`	`FREQ`	GPU	1
`mon`	`cont`	`POW`	Clusters, RAM, GPU, SoC	5
	`cont`	`TEMP`	big cores	4
	`cont`	`TEMP`	GPU	1
	`disc`	`PMC`	LITTLE cores	16
	`disc`	`PMC`	big cores	24

Listing 1: RTM code for agnostic control and monitoring of application and device knobs and monitors.

```
1  void rtm::control_loop(){
2   while(1){
3    temp_mon = dev_api.mon_cont_get(temp_mons[2]);
4    if(apps.size()){
5     app_perf = app_mons_cont[0];
6     if(app_perf.val < app_perf.min){
7      if(freq_knob.val < freq_knob.max){
8       freq_knob.val++;
9       dev_api.knob_disc_set(freq_knob, freq_knob.
         val);
10    }}
11    else if(temp_mon.val > temp_mon.max){
12     freq_knob.val--;
13     dev_api.knob_disc_set(freq_knob, freq_knob.
         val);
14 }}}}
```

4.2 Agnostic Runtime Management

A basic controller was implemented within the runtime management layer to illustrate the use of knobs and monitors for maintaining an application performance target while optimising a given device monitor. Listing 1 shows the code for the controller, which ensures that the value of the application performance monitor remains within its bounds. This is achieved by adjusting the device frequency knob in order to avoid violations of the monitor bounds `app_perf.min` and `app_perf.max` (lines 6 – 9). The optimisation of device temperature (line 11) is the secondary objective and is achieved by decrementing the frequency knob (line 12), trading off excess application performance (lines 12–13).

The behaviour of this controller is shown in Figure 2 while running a numerical benchmarking application (Whetstone). This benchmark performs numerical functions using integer and floating-point arithmetic. Its performance is measured in thousands

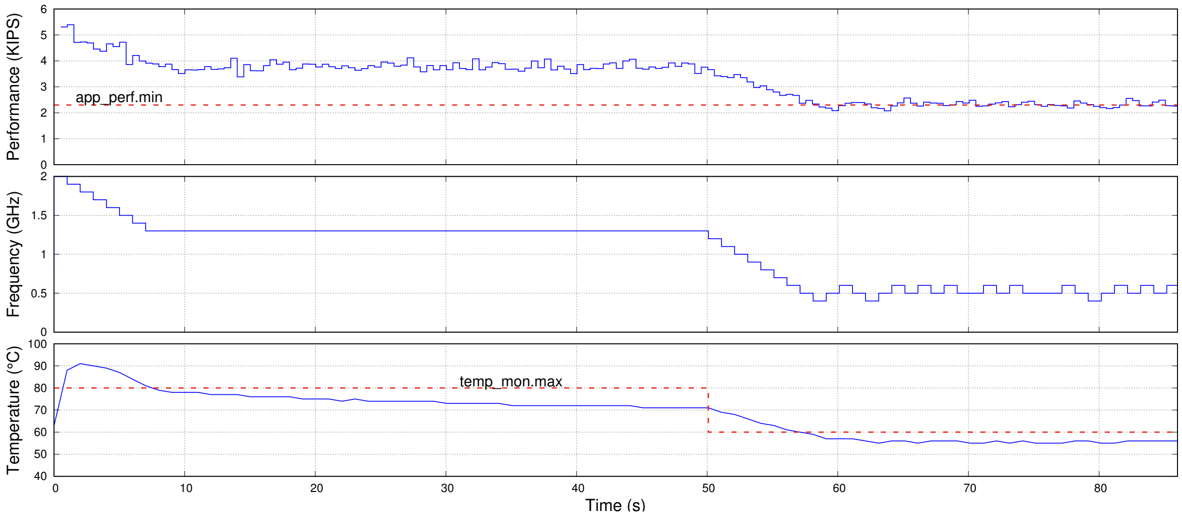

Figure 2: Device temperature optimisation under application performance constraints using the controller RTM, including dynamic adjustment of the temperature threshold from 80 to 60°C.

of Whetstone instructions per second (KIPS), which is exposed as a continuous monitor with bounds of $[2.30, \infty)$. Initially, the controller set the device frequency to maximum and observed the device temperature. As the temperature increased above the maximum threshold specified by `temp_mon.max` (80°C), the controller reduced the frequency until the temperature was below the threshold whilst ensuring that the application performance was higher than `app_perf.min`. After 50 seconds, the platform reduced its temperature threshold to 60°C and the RTM reduced the frequency in response until the updated monitor bound was satisfied while still meeting the application throughput requirement.

This experiment demonstrates the basic operation of the framework and illustrates the dynamic nature of its knobs and monitors. The controller is application- and platform-agnostic as it could operate, without modification, with any application that exposes a performance monitor and any platform that exposes a frequency knob and temperature monitor.

4.3 Platform Agnosticism

The portability of RTMs and applications implemented within the framework is demonstrated in Figure 3, which shows the design-space exploration (DSE) of the same application across two heterogeneous platforms using the same RTM code. A Jacobi iterative solver was used as a case-study application.

The Jacobi method solves the system of N linear equations $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$, where $\boldsymbol{A}$ is an $N \times N$ matrix and $\boldsymbol{x}$ and $\boldsymbol{b}$ are $N \times 1$ column vectors. If $\boldsymbol{A}$ is decomposed into diagonal and remainder components $\boldsymbol{D}$ and $\boldsymbol{R}$, under suitable conditions $\boldsymbol{x}$ can be computed iteratively, with later iterations containing more accurate results. The application can operate a tradeoff between the speed of calculation (solves per second) and the accuracy of the result (mean squared error) by adjusting the number of iterations performed and the precision of the data type.

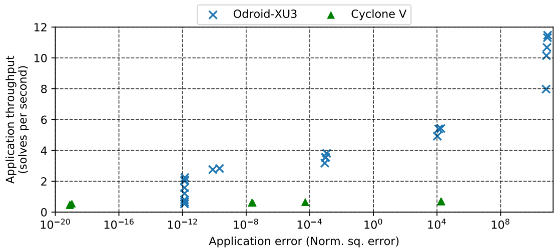

Figure 3: Design-space exploration of the Jacobi application across the Odroid-XU3 and Cyclone V devices.

Throughput and accuracy were exposed as monitors while iterations to perform and precision were exposed as knobs. The DSE extended to application execution on the heterogeneous components of both platforms, including the GPU on the Odroid and the FPGA on the Cyclone V, in addition to the CPUs. Points in Figure 3 show the resultant throughput and error for each combination of knob values, with blue crosses for the Odroid and green triangles for the Cyclone V. This experiment demonstrates that the same application and RTM code can be used on any platform supported within the proposed framework.

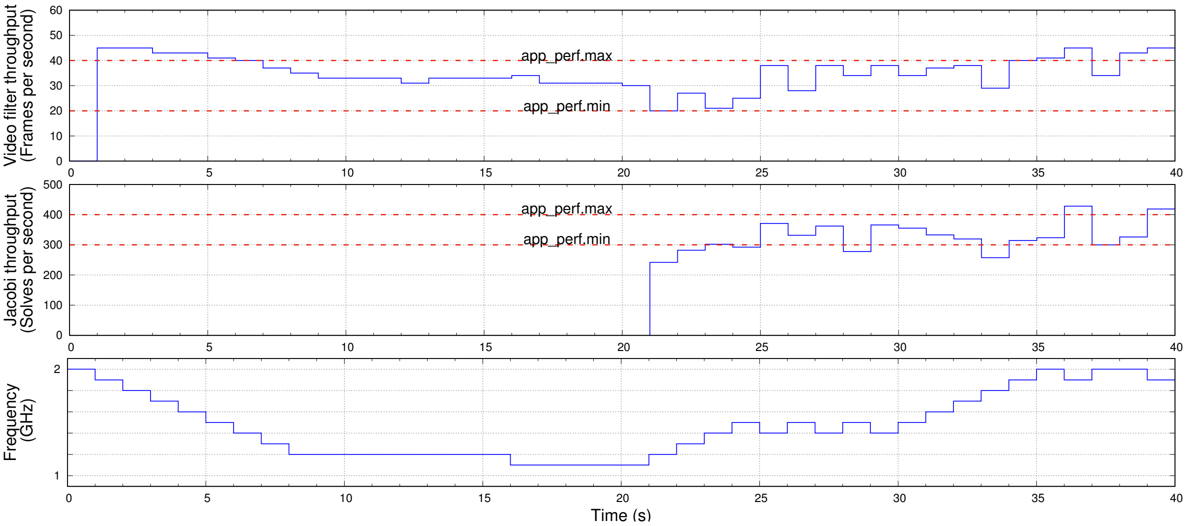

Figure 4: Runtime management of the throughput of two concurrently-executing applications through the framework. The Jacobi application begins execution at 21 seconds and the device frequency is adjusted to compensate.

4.4 Concurrency Management

This subsection demonstrates how the framework supports the management of concurrently executing applications. A runtime control algorithm was implemented with a target of keeping the throughput monitor of each application within its bounds, `app_perf.min` and `app_perf.max`, while minimising device frequency. The behaviour of this controller is shown in Figure 4, where the execution of two applications is indicated by their throughput over time. The top plot shows a video filtering application and the middle plot shows the Jacobi iterative solver.

Initially, the video filter application was the only application executing. As a result, the runtime controller adjusted the CPU frequency to meet the application throughput bounds at the lowest frequency possible. The Jacobi application began its execution after 21 seconds, shortly after which the RTM observed that its throughput was below the desired minimum bound. The throughput of the video filter also decreased due to competition for device resources. To compensate, the controller increased the CPU frequency such that the throughput of both applications returned to within their bounds.

4.5 Comparison of RTM approaches

To demonstrate the framework's optimisation and comparative capabilities, two state-of-the-art runtime management approaches were implemented within the proposed framework. The first approach, RTM-A (Reddy et al., 2017), aims to optimise power consumption by monitoring hardware performance counters to identify opportunities where CPU frequency can be reduced without impacting application performance. The second approach, RTM-B (Maeda-Nunez et al., 2015), employs reinforcement learning to predict the frequency that should be selected to meet an application performance target based on previous application behaviour. RTM-A was originally evaluated on the Odroid-XU3 platform using standard benchmarks with a reported mean energy saving of 25% compared to the Linux Ondemand governor. RTM-B was evaluated on the BeagleBoard-xM platform using a video decoder application with a reported mean reduction in energy consumption of 30% when compared to the Ondemand governor.

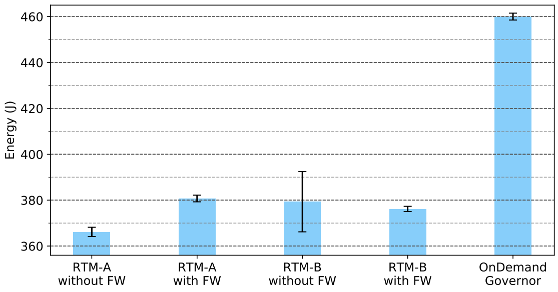

Figure 5: Mean total energy consumed by the Odroid-XU3 running the video decoder application under the control of each RTM, both with and without the framework (FW). The experiment was repeated 50 times for each RTM.

These two approaches lack portability and direct comparisons cannot be made due to the different platforms used for experimental validation. Implementation within the proposed framework allows them

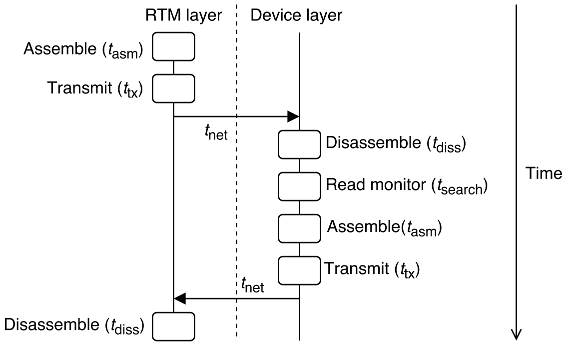

Figure 6: Breakdown of the sources of latency introduced by the framework for communication between the RTM and device layers.

to be directly compared, saving development time and improving the accuracy of the comparison. To demonstrate this, the RTMs were evaluated using an OpenCV video decoding application on the Odroid-XU3 platform. The application exposes a continuous monitor for the frame rate, with a minimum bound of 25 frames per second. The RTMs are directly compared in Figure 5, between bars two and four, showing that the application consumed a mean total energy of 381 J and 376 J under the control of RTM-A and RTM-B, respectively. Comparison with the Linux Ondemand governor (bar five) shows energy savings of 17.2% and 18.2%, respectively. This demonstrates that while RTM-B achieves a greater energy saving, it is less than reported in the literature for this specific application and platform pair.

4.6 Overheads

As with any abstraction, the framework introduces an energy overhead due to the additional computation required. This overhead can be estimated by comparing standalone versions of RTM-A and RTM-B against their implementations within the framework. Results of these experiments can be seen in Figure 5 for RTM-A (bars one and two) and for RTM-B (bars three and four). RTM-A required 19.6 J (5.48%) more energy, while RTM-B required only 15.2 J (4.23%) more energy, in the minimum case. The minimum case was used to minimise the impact of other running processes on the result. When compared to the Ondemand governor, the two RTMs still achieved significant savings regardless.

The framework also introduces latency overheads that limit RTM reaction rates. Figure 6 is a visualisation of the steps involved in reading a device monitor inside the framework, from which seven internal latency sources can be identified. t_{asm}, t_{tx} and t_{diss} are the times to assemble, transmit and disassemble a message used for conveying monitor information. t_{net} is the message-passing interface latency and t_{search} is the time to search for and read a monitor.

The latency related to each API call was measured and found to be 80–200 μs, with 40% attributed to cross-layer communication. For an RTM reading one device monitor and setting one device knob per update, this limits the update rate to 1.67 kHz.

5 Conclusions

This paper has presented a framework that enables application- and platform-agnostic runtime management of concurrently executing applications on heterogeneous multi-core systems. This is achieved by visualising a system as three distinct layers connected by dynamic knobs and monitors that allow a range of tunable parameters and observable metrics to be exposed. Framework operation with concurrent applications has been demonstrated. The framework enables the direct comparison of competing RTM approaches, which was not previously possible, and simplifies RTM development. It also introduces very modest energy and latency overheads that have limited impact on the operation and performance of RTMs. An open-source C++ implementation is available[a].

In addition to the experiments presented in this paper, the framework has been used to explore temperature variability of a heterogeneous platform for reliability modelling (Tenentes et al., 2017) and to demonstrate how application knobs and monitors can provide additional opportunities for system optimisation (Leech et al., 2018a). Research is ongoing to provide further validation of the framework and to integrate additional applications, devices and RTMs.

ACKNOWLEDGEMENTS

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 are openly available from the University of Southampton repository available at https://doi.org/10.5258/SOTON/D0565.

An open source implementation of the framework can be found at https://github.com/PRiME-project/PRiME-Framework.

The authors would like to thank Joshua M. Levine and James R. B. Bantock for their role in the initial development of the PRiME Framework methodology and API. The authors would like to acknowledge Mohammad Sadegh Dalvandi and Basireddy Karunakar Reddy for contributions to the experimental results and the development of runtime algorithms.

REFERENCES

Baldassari, A., Bolchini, C., and Miele, A. (2017). A Dynamic Reliability Management Framework for Heterogeneous Multicore Systems. In *IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems*.

Das, A., Shafik, R. A., Merrett, G. V., Al-Hashimi, B. M., Kumar, A., and Veeravalli, B. (2014). Reinforcement Learning-based Inter- and Intra-application Thermal Optimization for Lifetime Improvement of Multicore Systems. In *Design Automation Conference*.

Fleming, S. T. and Thomas, D. B. (2014). Heterogeneous Heartbeats: A Framework for Dynamic Management of Autonomous SoCs. In *International Conference on Field-Programmable Logic and Applications*.

Gadioli, D., Palermo, G., and Silvano, C. (2015). Application Autotuning to Support Runtime Adaptivity in Multicore Architectures. In *International Conference on Embedded Computer Systems: Architectures, Modeling and Simulation*.

Gaspar, F., Taniça, L., Tomás, P., Ilic, A., and Sousa, L. (2015). A Framework for Application-guided Task Management on Heterogeneous Embedded Systems. *ACM Transactions on Architecture and Code Optimization*, 12(4).

Hoffmann, H., Eastep, J., Santambrogio, M. D., Miller, J. E., and Agarwal, A. (2010). Application Heartbeats: A Generic Interface for Specifying Program Performance and Goals in Autonomous Computing Environments. In *International Conference on Autonomic Computing*.

Hoffmann, H., Maggio, M., Santambrogio, M. D., Leva, A., and Agarwal, A. (2013). A Generalized Software Framework for Accurate and Efficient Management of Performance Goals. In *International Conference on Embedded Software*.

Hoffmann, H., Sidiroglou, S., Carbin, M., Misailovic, S., Agarwal, A., and Rinard, M. (2011). Dynamic Knobs for Responsive Power-aware Computing. In *International Conference on Architectural Support for Programming Languages and Operating Systems*.

Kim, Y. G., Kim, M., and Chung, S. W. (2017). Enhancing Energy Efficiency of Multimedia Applications in Heterogeneous Mobile Multi-core Processors. *IEEE Transactions on Computers*, 66(11).

Leech, C., Bragg, G. M., Balsamo, D., Wachter, E., Merrett, G. V., and Al-Hashimi, B. M. (2018a). Application Control and Monitoring in Heterogeneous Multiprocessor Systems. In *International Symposium on Reconfigurable Communication-centric Systems-on-Chip*.

Leech, C., Kumar, C., Acharyya, A., Yang, S., Merrett, G. V., and Al-Hashimi, B. M. (2018b). Runtime performance and power optimization of parallel disparity estimation on many-core platforms. *ACM Transactions on Embedded Computing Systems*, 17(2).

Maeda-Nunez, L. A., Das, A. K., Shafik, R. A., Merrett, G. V., and Al-Hashimi, B. (2015). PoGo: An Application-specific Adaptive Energy Minimisation Approach for Embedded Systems. In *HiPEAC Workshop on Energy Efficiency with Heterogenous Computing*.

Paone, E., Gadioli, D., Palermo, G., Zaccaria, V., and Silvano, C. (2014). Evaluating Orthogonality between Application Auto-tuning and Run-time Resource Management for Adaptive OpenCL Applications. In *International Conference on Application-specific Systems, Architectures and Processors*.

Rahmani, A. M., Haghbayan, M. H., Miele, A., Liljeberg, P., Jantsch, A., and Tenhunen, H. (2017). Reliability-aware runtime power management for many-core systems in the dark silicon era. *IEEE Transactions on Very Large Scale Integration Systems*, 25(2).

Reddy, B. K., Singh, A. K., Biswas, D., Merrett, G. V., and Al-Hashimi, B. M. (2017). Inter-cluster Thread-to-core Mapping and DVFS on Heterogeneous Multicores. *IEEE Transactions on Multi-scale Computing Systems*, PP(99):1–1.

Singla, G., Kaur, G., Unver, A. K., and Ogras, U. Y. (2015). Predictive dynamic thermal and power management for heterogeneous mobile platforms. In *Design, Automation Test in Europe*.

Sui, X., Lenharth, A., Fussell, D. S., and Pingali, K. (2016). Proactive Control of Approximate Programs. In *International Conference on Architectural Support for Programming Languages and Operating Systems*.

Tenentes, V., Leech, C., Bragg, G. M., Merrett, G., Al-Hashimi, B. M., Amrouch, H., Henkel, J., and Das, S. (2017). Hardware and Software Innovations in Energy-efficient System-reliability Monitoring. In *IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems*.

Vassiliadis, V., Chalios, C., Parasyris, K., Antonopoulos, C. D., Lalis, S., Bellas, N., Vandierendonck, H., and Nikolopoulos, D. S. (2016). Exploiting Significance of Computations for Energy-constrained Approximate Computing. *International Journal of Parallel Programming*, 44(5).

Wang, Z., Tian, Z., Xu, J., Maeda, R. K. V., Li, H., Yang, P., Wang, Z., Duong, L. H. K., Wang, Z., and Chen, X. (2017). Modular Reinforcement Learning for Self-adaptive Energy Efficiency Optimization in Multicore System. In *Asia and South Pacific Design Automation Conference*.

Yang, S., Shafik, R. A., Merrett, G. V., Stott, E., Levine, J. M., Davis, J., and Al-Hashimi, B. M. (2015). Adaptive Energy Minimization of Embedded Heterogeneous Systems using Regression-based Learning. In *International Workshop on Power and Timing Modeling, Optimization and Simulation*.