Predictive Thermal Management for Energy-efficient Execution of Concurrent Applications on Heterogeneous Multi-cores

Eduardo Weber Wächter, Cédric de Bellefroid, Karunakar Reddy Basireddy, Amit Kumar Singh, *Member, IEEE*
Bashir Al-Hashimi, *Fellow, IEEE* and Geoff Merrett, *Member, IEEE*

***Abstract*—Current multi-core platforms contain different types of cores, organized in clusters (e.g., ARM's big.LITTLE). These platforms deal with concurrently executing applications, having varying workload profiles and performance requirements. Run-time management is imperative for adapting to such performance requirements, workload variabilities, and to increase energy and temperature efficiency. Temperature has also become a critical parameter since it affects reliability, power consumption and performance, and hence must be managed. This paper proposes an accurate temperature prediction scheme coupled with a run-time energy management approach to proactively avoid exceeding temperature thresholds while maintaining performance targets. Experiments show up to 20% energy savings while maintaining high temperature averages and peaks below the threshold. When compared to state-of-the-art temperature predictors, the present work predicts 35 % faster and reduces the mean absolute error from 3.25 °C to 1.15 °C for the evaluated applications scenarios.**

***Index Terms*—Runtime Management, Thermal Prediction, Multi-Cores.**

I. Introduction

Multi-core platforms that contain different types of cores, organised in clusters, are emerging; e.g. ARM's big.LITTLE architecture. These platforms often need to deal with variable workloads, generated by concurrently executing applications, having different performance requirements. Run-time management is imperative for adapting to such performance requirements and workload variabilities, and to increase energy and temperature efficiency [1]. Moreover, management becomes challenging when applications are multi-threaded and heterogeneity of the processing cores needs to be exploited (i.e. identifying the most appropriate cluster(s) for each application). The existing run-time management approaches exploit cores situated in different clusters simultaneously (referred to as inter-cluster exploitation) and Dynamic Voltage and Frequency Scaling (DVFS) potential of cores [1], [2], [3]. However, these approaches lack in providing an accurate temperature estimator. We postulate that such exploitation may help to satisfy performance requirements while simultaneously achieving energy savings and avoiding thermal peaks.

System on chip thermal management has become a critical subject. Its effect may vary from transient faults to long-term defects of the chip. To mitigate such effects, thermal hotspots, thermal gradients and thermal cycling need to be well managed. Thermal hotspots are high-temperatures at particular spatial locations on the chip. CPU or cache units are usual hotspots in a chip die hotspots. Thermal hotspots induce failures such as electromigration, stress migration and dielectric breakdown hybrid. Thermal gradients are the spatial variations of temperature across the die. As the die size of multi-core processors gets larger, thermal gradients increase the interconnect delays and consequently, inducing larger clock skews [4]. Communication between cores is thus negatively affected. Thermal cycling represents repeated temperature temporal variations in the die and reduces lifetime reliability [5].

To mitigate thermal hotspots, gradient and cycling issues, appropriate thermal management actions are needed to improve the performance of the system while reducing the power consumption and protecting the chip from damage. Most thermal management techniques focus on short-term performance, limiting the influence of temperature on the system performance. Performance should, however, be kept at an acceptable level for the running applications. Aforementioned thermal issues are faced using different techniques. *Clock-gating*, a reactive dynamic power and temperature management technique, reduces the power consumption and thus the temperature of the chip by shutting down parts of the circuit when the chip temperature is too high. DVFS can be used as a proactive or reactive technique, adapts the frequency to obtain the desired performance, limiting the power consumption as well as the temperature. Reactive DVFS reduces the frequency whenever the die temperature rises above a certain defined threshold. A task migration process moves tasks from hot cores to cool cores to avoid high-temperature peaks or thermal gradients across the die.

In this paper, we propose a run-time management approach coupled with an accurate temperature prediction scheme to comply with energy-performance requirements while keeping the temperature below a threshold. This way we focus on the long-term reliability while avoiding thermal hotspots and thermal cycling. We combine a power management algorithm with

Manuscript received August 24, 2018; revised November 3, 2018; accepted January 10, 2019. This work is 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/D0793 (https://doi.org/10.5258/SOTON/D0793).

E. W. Wachter, C. de Bellefroid, K. Basireddy, B. Al-Hashimi and G. Merrett are with the School of Electronics and Computer Science, University of Southampton, Southampton, UK (e-mail: e.weber-wachter@soton.ac.uk; cedb1g16@ecs.soton.ac.uk; kr.basireddy@soton.ac.uk; bmah@ecs.soton.ac.uk; gvm@ecs.soton.ac.uk)

A. K. Singh is with the University of Essex, Colchester, UK (e-mail: a.k.singh@essex.ac.uk)

a temperature prediction algorithm developed for heterogeneous architectures. The accurate temperature predictor helps to avoid high-temperature averages and peaks. To address the aforementioned challenges, this paper makes the following major contributions:

1) An accurate temperature prediction algorithm that can work for any frequency setting of the system.
2) A Runtime manager that proactively controls the frequency setting to keep the temperature below a configurable threshold value.

II. Experimental Design

This paper evaluates existing and the proposed methods on the heterogeneous MPSoC platform Odroid-XU3, composed of the Samsung Exynos 5422 SoC [6]. It contains four ARM Cortex-A15 (big) CPUs, four ARM Cortex-A7 (LITTLE) CPUs and six ARM Mali-T628 GPU cores. Such an architecture provides opportunities to exploit different designs as low power processing (LITTLE cores) and high-performance processing (big cores). The platform contains five temperature sensors enabling management decisions based on the current thermal state of the chip. The GPU and each one of the four big cores have temperature sensors.

The MPSoC provides DVFS at a per-cluster granularity. For the Cortex-A15 cluster, the frequency can be varied between 200 MHz and 2000MHz with a 100 MHz step, whereas for the Cortex-A7 cluster, it can be varied between 200 MHz and 1400 MHz with a step of 100 MHz. The frequency of the GPU cluster can be set at 177 MHz, 266 MHz, 350 MHz, 420 MHz, 480 MHz, 543 MHz and 600 MHz. It should be noted that we vary only frequency, but the firmware automatically adjusts the voltage based on preset pairs of voltage-frequency values. The SoC also has 2 GB LPDDR3 RAM, operating at 933 MHz. Memory system is not considered in the design space exploration while building the temperature predictor, as it only two levels of DVFS [7]: 400 and 800 MHz, which would affect severely the performance for the evaluated applications.

The Odroid-XU3 board allows hardware measurement of power consumption. It contains four real time current/voltage sensors for four separate power domains: big (A15) CPU cores, LITTLE (A7) CPU cores, GPU cores and DRAM. A power measurement circuit estimates the power as the product of voltage and current, i.e. P = I.V. The energy consumption is measured as the product of average power consumption and execution time. Since power is considered for all the domains, the energy consumption of all the software components (e.g., proposed predictor, OS, runtime manager, applications, etc.) executing within the chip are included.

III. Motivational Example

Current state-of-the-art runtime management approaches present a way of dynamically changing voltage and frequency (DVFS) to avoid power consumption or temperature from surpassing a given requirement. Some of these proposals present throttling on frequency [8] while trying to comply with these requirements. Runtime management (RTM) normally takes into account only performance and energy. In [9], the authors propose a RTM approach that first selects thread-to-core mapping based on performance requirements and resource availability. Then it classifies the workload using the metric Memory Reads Per Instruction (MRPI). Finally, it decides the appropriate V-f pair for the predicted workload. This approach does not take into account the temperature, leading to temperature violations and performance losses due to frequency throttling by the linux kernel. In this paper, the approach in [9] is used as a study of a state-of-the art RTM approach to include the temperature predictor and avoid temperature going beyond the threshold.

Figure 1 presents frequency and temperature measurements for the big core when the Linux Ondemand governor is controlling the execution of `blackscholes` application from the PARSEC benchmark [10]. The application was run by allocating all cores (big and LITTLE) available on the Odroid-XU3. It is shown that multiple times after 100 seconds, Linux needs to reduce the frequency after the temperature has reached above 95 C. These variations in the frequency may lead to thermal cycling. In [11], the authors evaluate the thermal behaviour for mobile gaming devices. It shows that the GPU frequency is reduced but is not coordinated with the CPU frequency adjustment. The net effect is that the temperature continues to rise even after throttling CPU frequency due to thermal inertia. This phenomenon occurs because the temperature of a device is influenced by its current frequency and its past frequency values.

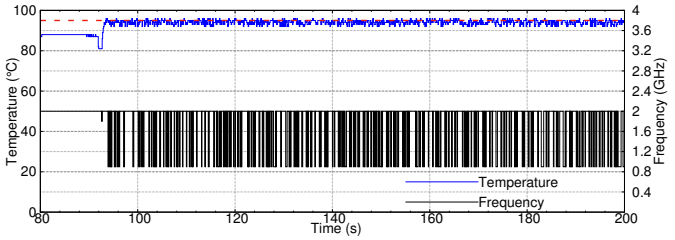

Fig. 1: Linux ondemand governor Frequency setting over time.

Even RTM which provide better energy-performance tradeoffs presents the same behaviour. Figure 2 presents the same scenario but with a state-of-the-art RTM [8] controlling the system. This RTM does not take into account the temperature, which also leads to Linux overriding operation to throttle the frequency when the temperature exceeds the 95 C threshold.

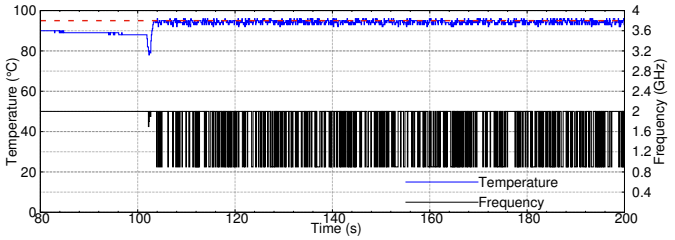

Fig. 2: Frequency over time with a State-of-the-art RTM.

This throttling could be avoided using less abrupt changes to current frequency changing algorithms. One possible solution is shown in Figure 3 where the performance requirements are met, but there is less changing of frequency and no throttling. This can be achieved by employing proactive management where a temperature predictor can be used to set

the frequencies while staying below thermal threshold. With less frequency throttling, we aim to also reduce the average temperature of the chip and increase performance as well.

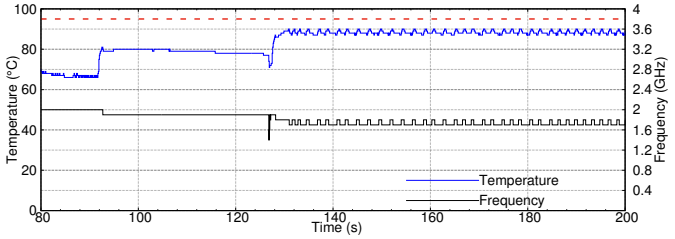

Fig. 3: Frequency setting over time by avoiding throttling.

IV. State-of-the-Art

A. Temperature prediction

Firstly we introduce the reactive and the proactive approaches used for temperature management. **Reactive** methods focus on reducing the die temperatures based on the current temperatures. Most of those techniques involve shutting down or slowing down cores after the die temperature rises above a defined threshold. The time between two temperature checks is usually short to avoid to temperature exceeding the limit. Some examples of reactive approaches are implemented in the Linux kernel. If the CPU temperature goes above the threshold, Linux will throttle the frequency, as demonstrated in the previous section, largely impacting application performance. On the other hand, **proactive** methods usually involve the prediction of future die/core temperatures to adjust the workloads or frequencies before exceeding a defined threshold. The computation of the predicted temperature increases the performance overhead of proactive methods when comparing to reactive ones. However, they run less frequently than reactive methods.

In [12], Coskun et al. propose an example of proactive methods using an auto-regressive moving average (ARMA) model to predict future temperatures. This ARMA model is one example of regression models employed to temperature prediction. The Authors in [13] extend an ARMA model considering with exogenous inputs (ARMAX model). The exogenous inputs are, in this case, the average power trend of the running applications. In [14], Ge et al. claim that their neural network approach to predict future temperature performs, on average, 79% better than the ARMAX model while limiting the maximum prediction error to 2.5 C. This maximum prediction error is however difficult to compare with the ARMA model as this model takes some time to adapt to the changing environments (ambient temperature, running applications etc.).

Prakash et al. [11] estimate the temperature of the CPU and GPU separately for cooperative CPU-GPU thermal management on chip. Their estimator uses the actual temperature sensors of both the CPU and GPU as well as the cores utilisation to set the frequency setting for the next time interval. Sigla et al. present a predictor using power sensors to predict the next power consumption based on the following frequency setting is developed in [15]. Their technique uses a leakage power model of the ARM big.LITTLE architecture on Odroid-XU3 to test its predictor and Dynamic Power and Frequency Management technique. An Extension of this work has also been published in [16].

In [17] the Authors propose a power management strategy for mobile games. The approach saves 1.9% of energy compared to the Android default governor for the evaluated scenarios in the Odroid XU3 board. The work uses the frame rate as a metric to evaluate the workload predictors and applying a thread to core mapping. The power management is employed to minimize the operating frequency while keeping the frames-per-second (FPS) constraint.

Two works [18], [19] also propose power-temperature analysis for many- and multiprocessor systems. Pagani et al. [18] presents power budget concept, called Thermal Safe Power (TSP), which is an abstraction that provides power constraints as a function of the number of simultaneously active cores. Executing cores at any power consumption below TSP ensures that thermal management actions are not triggered. The Authors shows simulations for platform models with 72 heterogeneous cores which provides offline and online TSP computation for a particular mapping of active cores and ambient temperature. The simulations allows to obtain safe power and power density constraints for the worst cases, allowing system designers to estimate mapping decisions and the amount of dark silicon.

[19] presents a power-temperature stability and safety analysis technique. The approach is based on a formula to compute the stable fixed point and thermally-safe power consumption at runtime. Hardware measurements on a XU3 board with Android OS show that can predict the stable fixed point with an average error of 2.6%.

B. Run Time Management

Runtime management represents an essential paradigm in tackling these challenges by enabling optimisation and trade-offs between computational quality, application throughput, system reliability and energy efficiency. An increasing number of runtime management algorithms are being employed to control and optimise the execution of applications on heterogeneous embedded systems. Mainly online optimisation has been considered to cater for dynamic workload scenarios to optimise energy consumption while respecting the timing constraint. For online optimisation, either all the processing is performed at run-time or else the optimisation is supported by offline characterisation.

For performing all the processing at run-time, several works have been reported [20], [21], [22], [23], [24], [25]. In [20], the online algorithm utilizes hardware performance monitoring counters (PMCs) to achieve energy savings without recompiling the applications. The authors of [21] present an accurate run-time prediction of execution time and a corresponding DVFS technique based on memory resource utilisation. A similar approach, which is a hardware-specific implementation of the stall-based model, is proposed in [22]. In [23], an adaptive DVFS approach for FPGA-based video motion compensation engines using run-time measurements of the underlying hardware is introduced. In [25], 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 optimisation 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 [26], [27], [28], [29], [30], [31], [32]. Such approaches lead to better performance results than only online optimizations as they take advantage from both offline and online computations. In [26], thread-to-core mapping and DVFS are performed based on power constraint. Similarly, in [27], first thread-to-core mapping is obtained based on utilisation and then DVFS is applied depending upon the surplus power. However, the approaches of [26], [27] target homogeneous multi-core architectures and thus cannot be applied to heterogeneous ones.

The state-of-the-art shows some implementations of run-time managers using temperature predictors [15] or the current chip temperature [11]. These works show good improvements in energy and/or temperature efficiency. All these temperature predictors were evaluated for a training set (explained in details on next section); however the approaches lack accuracy, showing high average prediction errors (up to 5 C or 4%) , which may be improved by an accurate temperature predictor. Also, a few implementations [16] are implemented as kernel modules rather than a standalone library, which impact portability, e.g. to use the predictor in other runtime manager.

V. Proposed Methodology

An overview of the proposed methodology for temperature prediction and its integration into the runtime manager is illustrated in Figure 4. First, a training data set to classify the best temperature prediction regression model is created offline. This training set is composed of measurements of the system when executing applications from the PARSEC benchmark on both the big and LITTLE clusters of the chip. During this step, we log the temperature, frequency and power consumption for the memory and big cluster. Offline data is collected at a rate of 1 Hz and later used for 1 Hz temperature prediction at runtime, predicting the temperature over the next second of execution. The frequency was changed randomly every 500 ms to evaluate all the operating points of the platform. This way we focus more on the platform behavior rather than the application behavior.

When applying this approach, we noticed that leaving the fan disabled limits the testing ranges and capabilities. In particular, we were only able to use lower frequency levels of the big cluster as higher frequencies lead to reaching the temperature threshold quickly. By default the Linux kernel usually starts the fan when the temperature rises above the 65 C temperature threshold. Leaving the fan with this setting adds a non-linearity which is undesired. Therefore, the fan is turned on to make sure the predictor does not have any other non-linear behavior which would increase the temperature prediction error. We endorse that the same methodology could be applied for the system with the fan always off, but it would limit the system to operate only with lower frequencies. In this case the training set would generate different regression coefficients, explained in next section.

This training set is then compared with the different regression model outputs, explained in the next sections. The regression model that provides the least error on predicting temperature is then used to feed the runtime manager. Finally, the runtime manager can take into account the next interval when setting the frequency and mapping of the tasks. One advantage of the predictor is that it is totally decoupled from the RTM and thus can be used with other RTMs.

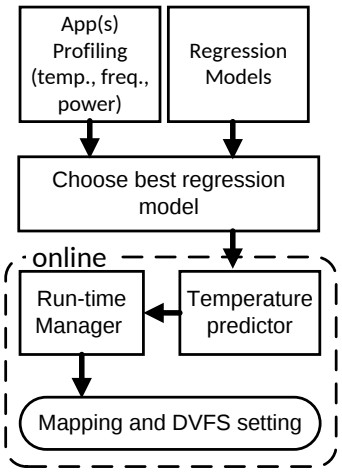

Fig. 4: Methodology to choose temperature regression model and how it is used in runtime management.

A. Temperature Predictor

This section outlines the proposed temperature predictor. We first list and describe the assumptions that are followed throughout the paper.

Assumption 1: The LITTLE cluster does not influence the global temperature. Since there are no temperature sensors available specifically targeting the LITTLE cluster, the only measurement available is the cluster's power consumption. To measure the impact of the LITTLE cluster on the big cluster temperature, we executed the PARSEC benchmark on the 4 LITTLE cores only, then measure the temperature on the big cluster. The maximum temperature achieved on the big cluster was 42 C. When executing the same benchmark on the big cluster, the minimum temperature on the big cluster is 42 C while the maximum is 95 C. Also, the frequency setting of the LITTLE cluster will not be modified by the temperature management algorithm developed in Section V-B.

Assumption 2: The GPU is not used by the running applications. The GPU frequency is set to the lowest possible frequency, and its power consumption is constant during system operation. To take into account the GPU management, applications should be written with environments such as OpenCL or OpenGL to enable the design space exploration. Also, the RTM can be able to deal with load balancing between the CPU and GPU to target energy/performance or temperature trade-offs.

Considering the above assumptions, Equation 1 outlines the regression model of the proposed temperature predictor.

$$T(t) = constant + \quad T(t \quad 1) + \quad T(t \quad 2) + \quad P_t^{big} + \quad P_{t \quad 1}^{mem} \quad (1)$$

The predictor uses the two past temperature measurements ($T(t \quad 1)$ and $T(t \quad 2)$) as well as the future power consumption estimation for the big cores (P_t^{big}) and the memory (P_t^{mem}). Considering that the memory operates at a constant frequency, the value of $P_{t \quad 1}^{mem}$ is the last value measured from the on-chip power sensor. The value of P_t^{big} is estimated using the following equation 2 [15],

$$P_t^{big} = (\quad C)_{t \quad 1}^{big} : f_t^{big} : (V_t^{big})^2 + V_t^{big} I_{leakage;t \quad 1}^{big} \quad (2)$$

Where and C are the activity factor and switching capacitance, f is the operating frequency, V is the voltage and $I_{leakage}^{big}$ corresponds to the leakage current, computed using the Equation 3,

$$I_{leakage}^{big} = \quad _1 T^2 e^{\frac{\beta_2}{T}} + I_{gate} \quad (3)$$

The parameters , , , , , $_1$ and $_2$ represent regression coefficients. The chip has first been put in an oven to retrieve measurements of the power sensors with temperature $T \ 2$ [40; 90] and a constant frequency f. During that process, is kept approximately constant using constant workload. The leakage power model has been computed using simulated annealing on MATLAB. Simulated annealing gave really good result to find $_1$ and $_2$ with a starting temperature of a thousand, a cooling rate of 10 ^{6} and five rounds per temperature. Figure 5 shows the power estimation of each cluster with I_{gate} and $_i$ coefficients computed using the simulated annealing.

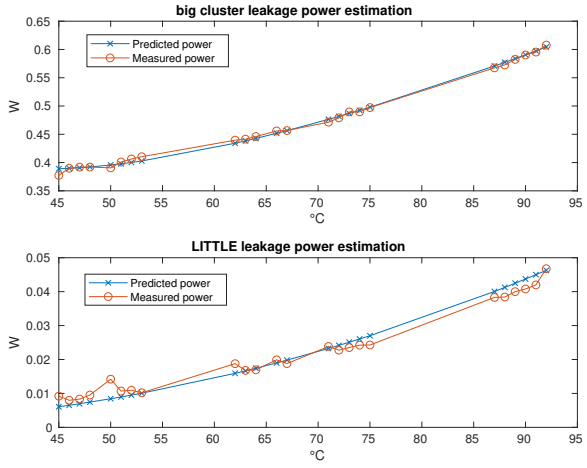

Fig. 5: Results from leakage coefficient computation and simulation. The predicted leakage power graph is close to the measured power of each cluster.

One way of improving the results of the predictor is to use previous prediction errors to estimate the future values. The Autoregressive Moving Average model (ARMA) uses this approach and should, therefore, improve the quality of the prediction. Equation 4 shows the estimated prediction error ($\tilde{}(t)$) model. The mean represented as , represents the actual error, and $\tilde{}$ the predicted error and $_1$ and $_2$ are the values previously calculated.

$$Error\ Prediction : \tilde{}(t) = \quad + \quad _1 : (t \quad 1) + \quad _2 : (t \quad 2) \quad (4)$$

Figure 6 presents a comparison between the predicted and actual measured temperature for different applications and temperature ranges. Also, it shows the evolution of prediction error over time.

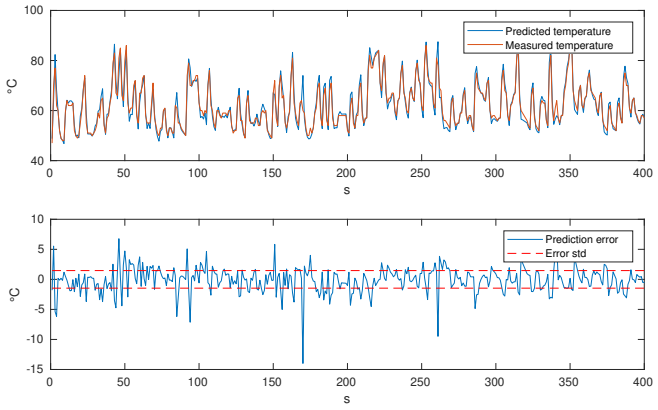

Fig. 6: Temperature measurements for the predicted temperature versus the actual readings and the error between them.

The estimator developed estimates the future temperature with a low average error at runtime but leaves different errors based on the operating frequency, e.g. the average error is different if the big cluster is executing at 1.7 or 2 GHz.

Error Correction Algorithm

The error difference between the frequencies is due to the willingness of the temperature estimator to act the same for the whole frequency set. To solve this issue, we propose an error correction algorithm that uses different error coefficients for each possible frequency setting of the big cluster. Algorithm 1 computes the associated error after each temperature prediction iteration. The Algorithm uses an `error_correction` table to store the error for each frequency. The error is calculated by the difference between the temperature prediction and the actual temperature measurement (Line 3) and then stored in the position of the `error_correction` table (Line 4). This error is then used to predict the next interval temperature (Line 6) taking into account the last error for that frequency. The same temperature prediction model could be applied for the error correction, but this would lead to a longer execution time and more memory to store the values for each one of the frequencies. Therefore, the decision is to trade-off accuracy for a lower execution time, and then providing a lower processing overhead.

Algorithm 1 Error correction algorithm

```
1: while true do
2:     temp = measure_temperature();
3:     error = temp  pred_temp;
4:     error_correction[frequency] = error;
5:     [pred_temp; frequency] = temp_estimator();
6:     pred_temp = pred_temp + error_correction[frequency];
7:     sleep(TIME);
8: end while
```

B. Dynamic Temperature Management

A Dynamic Temperature Management (DTM) is used to limit the chip temperature based on the proposed temperature estimator developed in the previous section. This section describes the developed DTM algorithm. The temperature is managed proactively by the proposed DTM while the Linux kernel uses a reactive method by default with the lowest temperature threshold being 95 degrees Celsius. Reactive temperature management needs to be running faster than proactive methods to avoid high-temperature violations since it decides after the temperature reaches a given temperature. This may result in performance losses as cores are shutdown to reach lower temperatures.

The **goal** of a DTM algorithm on heterogeneous architectures is to determine a maximum frequency setting for each cluster separately to avoid temperature violations. This algorithm is analysed based on its error rate, mean absolute error (MAE) and the performances losses or gains. The DTM algorithm may also reduce the power consumption. For example, the Dynamic Temperature Management developed in [15] is not only used for temperature purposes but also to limit the power consumption of the cluster.

The DTM algorithm developed determines the maximum frequency based on the temperature estimator from the previous section. It applies DVFS based on the maximum frequency for each cluster during a certain time interval. Algorithm 2 outlines the developed DTM. The DTM algorithm predicts the temperature for the highest frequency and reduces this frequency until the prediction stays below the defined threshold. This enables the temperature to stay below the threshold while keeping the maximum performances.

Algorithm 2 Dynamic Temperature Management

```
1: while true do
2:     frequency = MAX_FREQUENCY
3:     pred_temp = predict_temperature(frequency)
4:     while pred_temp > THRESHOLD do
5:         frequency = frequency - -
6:         pred_temp = predict_temperature(frequency)
7:     end while
8:     set_frequency(frequency)
9:     sleep(TIME)
10: end while
```

C. Predictive Dynamic Thermal and Power Management

The Predictive Dynamic Thermal and Power Management (PDTPM) for Heterogeneous Mobile Platforms developed uses [8], a Dynamic Power Management algorithm for heterogeneous architectures that applies DVFS to the different clusters. This method takes advantage of the frequency of the memory read and write instructions to adapt the CPU frequency settings and consequently reduce the energy consumption. The approach combines application mapping and DVFS to reduce the energy consumption.

It starts by applying a thread-to-core mapping of the different applications depending on their memory intensiveness. It then applies DVFS to reduce the energy consumption. The proposed PDTPM algorithm (Algorithm 3) combines the DTM algorithm (previous Section) with the power management based on memory reads per instruction (**MRPI**).

Algorithm 3 Predictive Dynamic Thermal and Power Management (PDTPM) algorithm.

```
1: interval_count = DTM_INTERVAL;
2: while true do
3:     %%% Temperature management %%%
4:     if interval_count == 0 then
5:         max_big_freq = DTM_big(big_freq);
6:         interval_count = DTM_INTERVAL;
7:     end if
8:     %%% Power management %%%
9:     new_little_freq = mrpi_little(little_freq);
10:    new_big_freq = mrpi_big(big_freq);
11:    if new_big_freq > max_big_freq then
12:        repredict_temperature();
13:        new_big_freq=max_big_freq;
14:    end if
15:    %%% Frequency set %%%
16:    set_frequencies(big, new_big_freq);
17:    set_frequencies(little, new_little_freq);
18:    interval_count- -;
19:    sleep(100ms)
20: end while
```

The DVFS algorithm is executed 10 times every second while the DTM should predict the temperature and choose the maximum frequency settings every second. The $interval\_count$ variable has been introduced for this purpose (in this case, $DTM\_INTERVAL$ should be initialised with 9). The DTM algorithm only runs when this variable is equal to 0 (line 4). It then resets this variable to its maximum value, which is equal to the number of time the DPM algorithm should run before another temperature prediction is made (see $DTM\_interval$ constant). The function $DTM\_big()$ predicts the temperature for the next interval and returns the maximum frequency settings that can be applied to the cores for the next time interval to avoid temperature threshold violations.

The second part (lines 9 and 10) applies DVFS based on the MRPI algorithm. It returns the frequencies settings for the next MRPI interval ($\frac{1}{10}$ of the DTM interval). If the frequency computed by the MRPI exceeds the max_big_freq (lines 11

to 14), the algorithm predicts the temperature for the next DTM interval (line 12 - The DTM_big function is called inside the $repredict\_temperature()$ function) and resets the frequency of the big cluster to the maximum allowed by the DTM algorithm (line 13).

Lastly, the algorithm (lines 15 to 19) sets the computed frequencies to both clusters, updates the interval counter and sleeps until the next interval.

VI. Results

This section is divided into two parts. First, we evaluate and compare the proposed regression model with the state-of-the-art, and we show the impact of the improvements made on this model. Then we present the results of the runtime management using the regression to predict the temperature. The temperature measurements have been collected at room temperature.

Validation of the proposed temperature predictor and methodology is done on the Odroid-XU3, see section IV for more details of the experimental setup. PARSEC [10] and SPLASH [33] applications are used to compare the results of the proposed PDTPM algorithm with different approaches. The chosen mapping for the validation and comparison is taken from a state-of-the-art approach [8].

TABLE I: Selected applications from PARSEC [10] and SPLASH [33] benchmarks including its performance requirements (execution time) in seconds.

PARSEC			SPLASH		
App Name	Abbr.	Req. (s)	App Name	Abbr.	Req. (s)
blackscholes	bl	500	water-spatial	wa	357.1
bodytrack	bo	416.7	raytrace	ra	454.5
swaptions	sw	1052.6	fmm	fm	9.5
freqmine	fr	909.1	MIBENCH		
vips	vi	416.7	basicmath	bm	0.826
streamcluster	st	588.2	crc32	crc	5.263
fluidanimate	fl	476.2	blowfish	bw	5.55

Table I lists the applications used for the validation. These applications will be tested against the new PDTPM approach and then compared to a series of Mapping-DPM tools of the Linux kernel and the Inter-cluster Thread-to-core Mapping and DVFS (ITMD). The ITMD approach proposes a mapping of tasks and the MRPI metric used to execute DVFS. The Linux kernel uses the Heterogeneous Multi-Processing (HMP) [34] scheduler to map the task on the different clusters. The temperature threshold of the PDTPM approach is set to 90 C while the Linux reactive temperature limit is 95 C. This is required to avoid temperature peaks to reach temperatures more than 95 C, the same as the Linux reactive control. The list of considered approaches is listed in Table II.

TABLE II: Approaches considered for comparison purposes with the additional proposed temperature approach.

Reference	Approach	Abbreviation
[34], [35]	HMP + Ondemand	HMPO
[34], [36]	HMP + Performance	HMPP
[34], [36]	HMP + Conservative	HMPC
[34], [36]	HMP + Interactive	HMPI
[8]	Inter-cluster Thread-to-core Mapping and DVFS	ITMD
[15]	CPU Cluster-Oriented Algorithm	CCA
[16]	Gradient Search Algorithm	GSA
proposed	PDTPM	PDTPM

A. Regression Model Evaluation

Table III outlines the results on the training set for the regression with and without the error prediction. MAE represents the mean absolute error, AE_{max} is the maximum absolute error and AE_{std} the standard deviation. The **AIC** is an estimator of the relative quality of a set of statistical models for a given set of data, i.e. AIC estimates the quality of each model, relative to each of the other models. A model with a lower AIC provides a better estimator. Thus, AIC provides a means for model selection. The mean absolute error on the training data set is 1.21 C without the error prediction and drops to 1.13 C when using it.

TABLE III: Comparison of regression with and without the error prediction. MAE, AE_{std} and AE_{max} are presented in degrees Celsius.

	Interval [s]	MAE	AE_{std}	AE_{max}	AIC
w/o Error Pred.	1.0	1.21	1.32	18,91	34222
with Error Pred.	1.0	1.13	1.31	16,91	33222

Table IV shows the error generated when the frequency error correction algorithm is used in combination with the proposed temperature predictor. This table outlines the reduction of the mean absolute error when using the error correction algorithm. The average error drops by more than one degree, from 2.48 C to 1.19 C, while the dependence of the error on the frequency drops by 0.3 C. This algorithm is not only useful to reduce the error difference between the frequencies, but it also lowers a lot the mean absolute error, by an average of approximately 50%.

TABLE IV: Online comparison of the proposed model with and without the error correction algorithm. MAE, AE_{std} and AE_{max} are presented in degrees celsius.

	Interval [s]	MAE	AE_{std}	AE_{max}
w/o Error Correc.	1.0	2.48	2.05	28.58
with Error Correc.	1.0	1.19	2.3	29.05

The prediction error is analysed by running a series of multi-threaded PARSEC applications on the big and the LITTLE cluster together for each different temperature predictor described in the Section V-B earlier in this paper. The final temperature prediction model developed performs better for each different temperature threshold of the DTM.

Figure 7 gives the results from the different predictors with different temperature threshold values. The bars show the mean absolute error (MAE) between the actual temperature and that estimated by the predictor, while the blue lines show the standard deviation for each predictor. It is important for the temperature predictor to maintain similar results and error for different ranges of temperature. The proposed predictor gives better error averages (53% better than the version without the error correction and 64% better than a state-of-the-art temperature predictor [15] with a temperature threshold of 90 C) while keeping the error standard deviation within the same range. The error standard deviation for other temperature thresholds might grow further alongside higher thresholds. This is because error correction introduces an instability when the workload is changing. This error correction algorithm needs time to adapt to the changing environments.

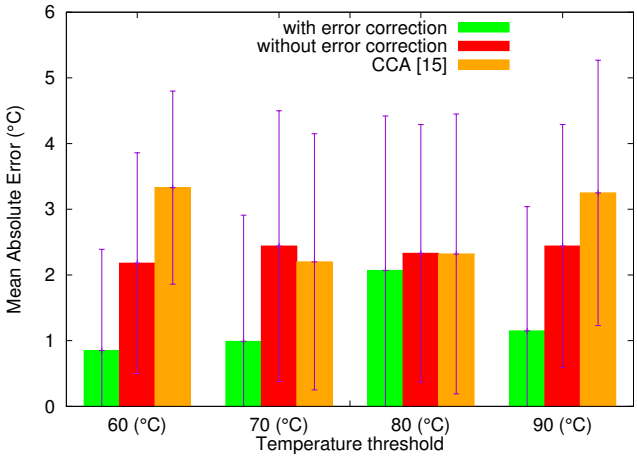

Fig. 7: Online comparison of different predictors using the DTM with a temperature threshold of 60, 70, 80 and 90 C.

Figure 8 presents a comparison of the proposed temperature predictor with the approach in [16] for a set of applications. The proposed predictor shows less error for most of the applications and also all the errors are lower than 2%.

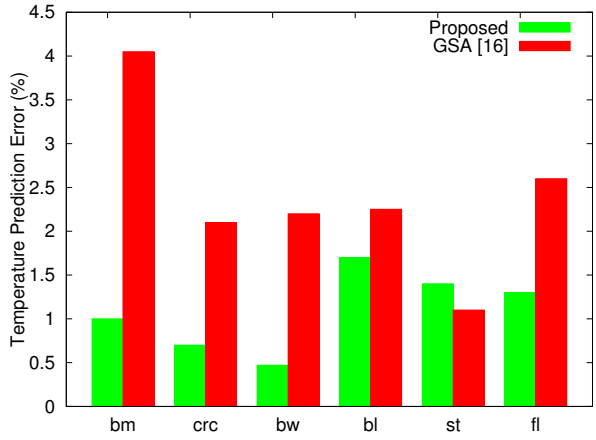

Fig. 8: Comparison of the proposed temperature predictor with the GSA precictor [16] for different applications.

B. Runtime Manager Evaluation

Table V lists the different application scenarios with their respective core allocations amongst the big and/or LITTLE cluster. These scenarios are then launched separately on the Odroid-XU3. The performances and energy consumption are measured and then compared.

TABLE V: Application scenarios and resource combinations determined by [8] mapping approach.

single	double	triple
bl (4L+4B)	bl-bo (2L+2B : 2L+2B)	bl-bo-sw (3B : 1B : 4L)
bo (4L+4B)	bl-sw (4B : 4L)	bl-bo-fr (3B : 1B : 4L)
sw (4L+4B)	fr-sw (4L : 4B)	sw-bo-fr (4L : 1B : 3B)
fr (4L)	wa-bl (2L+2B : 2L+2B)	bl-sw-fr (3B : 1B : 4L)
wa (4L+4B)	wa-bo (4L+3B : 1B)	wa-ra-vi (2L+2B : 1B
ra (3L+4B)	wa-ra (4L+3B : 1B)	: 2L+1B)

Figure 9 outlines an overview of the energy and execution time savings obtained by the proposed approach compared to the approaches detailed in Table V. It computes the improvements of each scenario for all approaches and then presents the average improvement for each approach. Figure 9 (a), (b) and (c) present results for scenarios with single, double and triple applications, respectively. Later, Figures 11 , 12 and 13 provide details for each of the applications compared to the Linux performance governor (HMPP).

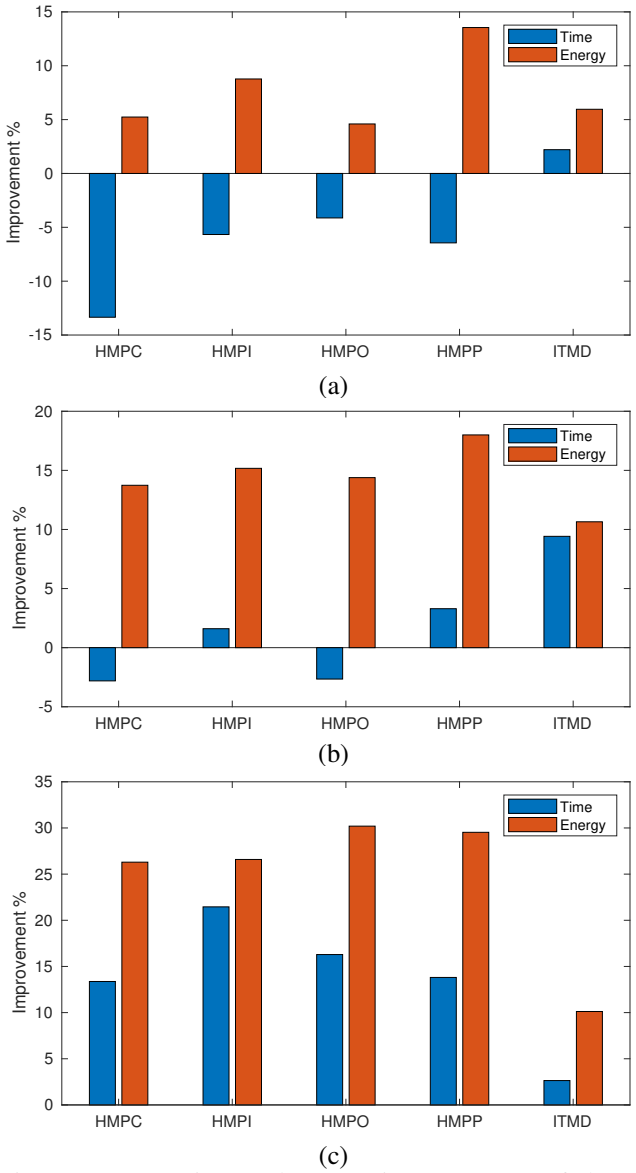

Fig. 9: Average time and energy improvements of the proposed PDTPM compared to existing approaches. Figure shows results for single (a), double (b) and triple (c) application scenarios, respectively.

Energy

Figure 9 (a) shows **single application** scenarios compared to the Linux HMP and governors. The proposed PDTPM performs better on average by 5 to 10% regarding energy consumption than the Linux governors, while increasing 5 to 10% on the execution time. Therefore showing a simple trade-off rather than real improvements. It is interesting to note that

for single application scenarios, the ITMD approach consumes a little more than the HMP+conservative Linux approach, less than 2% on average. This means that most of the PDTPM energy savings for single application scenarios are due to the temperature management algorithm, which improves energy savings in any of the single application cases. The results also show that the performance governor is affected since maintaining the highest frequency leads to more temperature threshold violation and thus frequency throttling is more likely to occur when comparing to the conservative governor, for example.

Figure 9 (b) outlines the **double application** scenarios. The proposed approach improves the energy consumption, by more than 10% on average than any other Linux HMP-governor association considered. It saves 14% of energy compared to the Linux conservative governor, the low-energy governor, and more than 17% compared the Linux, performance governor. Part of it is due to the mapping proposed by ITMD approach and its MRPI based DVFS management tool. However, more than 10% of the energy savings are due to PDTPM and especially to the temperature management algorithm added to the original ITMD.

Triple application scenarios shows that PDTPM improves more the energy savings made over the Linux HMP-governor associations (see figure 9(c)). It reaches an average of 28% of energy savings improvements over the different governors and more than 25% on the conservative governor, the one focusing on low energy consumption. These improvements are partly due to the added temperature management algorithm. The PDTPM raise the energy savings by 10% on average compared to the power management in ITMD alone while the other 15% is due to the mapping and power management.

The energy savings made on the different application scenarios by the temperature management evidently increases with the number of applications running at the same time on the cores. This is due to the high workload and consequently, high temperatures induced on the cores. The temperature management part of the DPTM limits the frequency to even lower frequency settings than for single application scenarios. This results in a reduction of the energy consumption.

Unlike existing approaches, the proposed approach is aware of concurrent execution; therefore, in the case of multi-application scenario, there was more space for optimization in terms of choosing thread-to-core mapping and compensating for contention. Moreover, the temperature threshold violations by other approaches become more prominent when multiple-applications are executed concurrently (this leads to frequent scaling down of frequency). This has helped our proactive thermal manager (PDTPM) to improve performance by not aggressively scaling down the frequency. The above two cases lead to improved performance in the case of two and three application scenarios compared to the single-application scenario.

Figure 10 presents the results for the average power for the single, double and triple application scenarios. It shows that the proposed approach reduces the power consumption when compared to the Linux governors and ITMD.

The mapping and PDTPM energy savings can be separated into three parts. The application mapping onto the cores try to limit the energy consumption by a mapping memory intensive application on the LITTLE core, sometimes trading performances against energy savings when performance requirements are still met. The MRPI based power management limits the frequency and consequently the energy consumption by adapting the frequency to the memory intensiveness of the applications. The temperature management limits the frequency avoid a certain threshold. This increases the energy savings as analysed earlier.

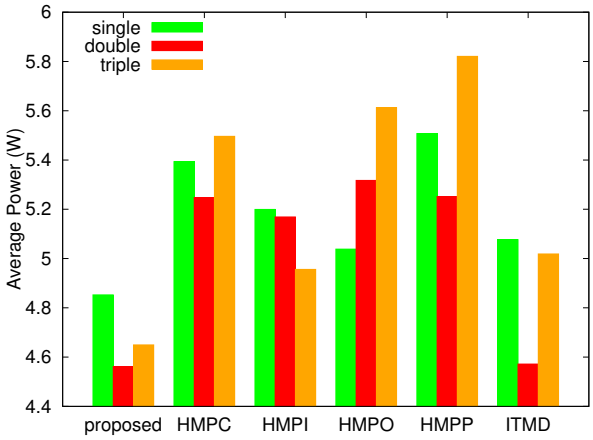

Fig. 10: Average power for the evaluated scenarios with single, double and triple applications.

Performance

Now we compare the PDTPM approach with the Linux Performance Governor for each scenario running one, two and three applications concurrently using the mapping of Table V. **Single application** scenarios show that PDTPM improves energy at the cost of performances. The improvements for single application scenarios are limited. For single application scenarios the PDTPM run faster than HMP-performance except for the PARSEC application *Freqmine* [10] as shown on figure 11. Energy savings of *Freqmine* are considerable at the cost of lower performances.

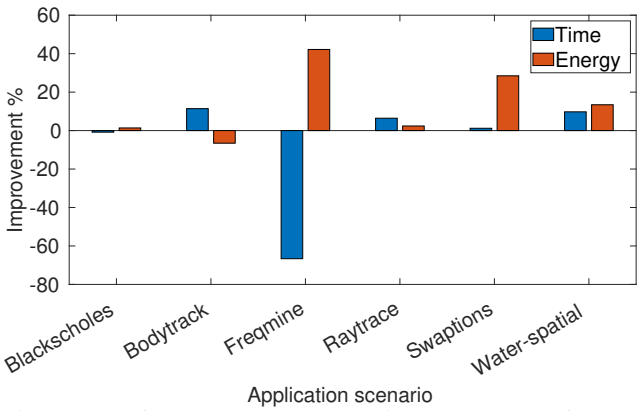

Fig. 11: Performance and energy improvements of the proposed PDTPM over Linux HMP + the performance governor.

In **double application** scenarios, the PDTPM improves the energy savings compared to the Linux HMP and governors, while the performance overheads of the proposed PDTPM are limited. The PDTPM encourages better performances for most of the double application scenarios compared to the different Linux HMP-governors associations. The addition of temperature management in proposed PDTPM improves the performances by almost 10%. Figure 12 shows that performances are usually better for every double application scenarios compared to HMPP, the Linux governor built for the highest performances.

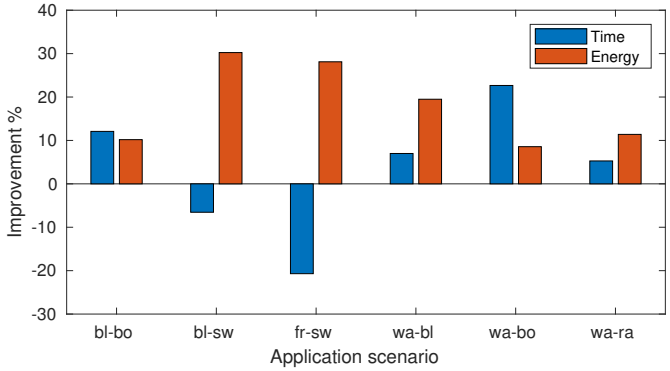

Fig. 12: Performance and energy improvements of the proposed PDTPM over Linux HMP + the performance governor for double application scenarios.

Triple application scenarios follow the results given by the two application scenarios. Figure 13 shows that performances and energy improve for all scenarios compared to the Linux governor. The energy savings are in the worst case, 10% when compared to the Linux governor.

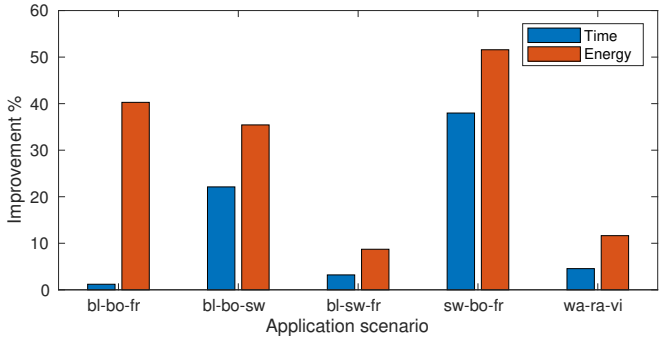

Fig. 13: Performance and energy improvements of the proposed PDTPM over Linux HMP + the performance governor for triple application scenarios.

Thermal Cycling

Managing the temperature variations is also important to avoid the reduction of lifetime reliability. A DTM algorithm may induce more temperature variation due to the frequency scaling that may change the frequency and thus the temperature at every time interval. This section compares the thermal cycling rates of the PDTPM for the different predictors and Linux governors.

Figure 14 shows the average of the temperature variations for the presented scenarios. Each bar represents the average temperature variation within one second. First, we measure how much the temperature decreases or increases compared to the previous second and then calculate the average for the sample. Figure 14 shows that the thermal cycling of the proposed model is almost equivalent to the one from [15] for every temperature thresholds. The difference in temperature between two measurements almost doubles at 90 C.

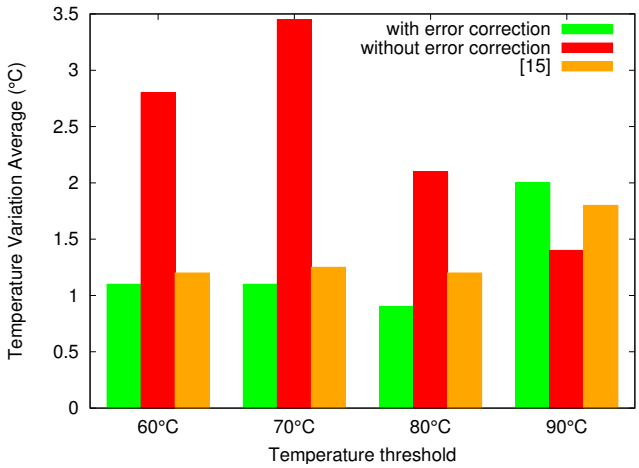

Fig. 14: Temperature variation average for the proposed associated with the different predictors. Thermal cycling is represented by the difference of temperature between two measurements of one second.

The PDTPM also increases thermal cycling compared to the Linux performance CPU governor [36]. Table VI shows that the difference between two temperature checks of the Linux performance governor is halved the value of the thermal cycling of the PDTPM. The Linux reactive thermal management model reduces thermal cycling. This is due to the smaller intervals between reactive measurements and frequency setting adjustments.

TABLE VI: Thermal cycling comparison of the PDTPM and the Linux performance governor.

90°C	Interval [s]	ΔT=s (>85°C)
With error correction	1.0	2.02
Linux performance governor	1.0	1.09

Overheads

As outlined in the previous sections, the proposed PDTPM saves energy by different means, but it has an overhead for predicting the temperature for the next interval. This section evaluates the overhead of the temperature predictor only, not taking into account the power management and DVFS. We measure the overheads caused by the temperature prediction, comprising the lines 4 to 7 of Algorithm 3, during the execution of the all scenarios outlined before. The average overhead is 836.5 s with a standard deviation of 48.5 s. However, 70% of these overhead is spent reading the temperature sensor and the power of the memory and big cluster. Therefore, only the temperature prediction algorithm takes only around 250 s in average. Since the temperature prediction is executed at every 10th time the DVFS algorithm is executed, the overhead is minimal. To put these results in perspective, a Model

predictive control-based policy (MPC) proposed in [37] takes more than 4 ms, and it is applied every 10 ms. The best comparison is the proposal in [16], where it takes 390 s to predict the temperature and to determine the frequency levels. This means the proposed approach is 35% faster than [16].

VII. Conclusions

This paper firstly demonstrates that an accurate temperature predictor helps to limit the high-temperature averages and peaks. The proposed temperature model combines [8] and a regression model to reduce further the theoretical mean absolute error to 1.13 C. The addition of an error correction algorithm that uses different error predictions for each frequency setting improves further the accuracy at runtime. The overall result is a faster and more accurate temperature prediction when comparing with the latest state-of-the-art proposals [16], [15]. The second contribution is the temperature estimator developed to build a Dynamic Temperature Management algorithm. The DTM proves that temperature management already reduces the energy consumption by limiting the frequency. The third contribution combines a state-of-the-art power manager and the DTM algorithm. This combination gave great results for two and three application scenarios, improving up to 20% the energy savings compared to the power manager alone while limiting the performance overhead. Finally, the accurate predictor is decoupled from the run-time manager and may be easily included in a different approach.

The proposed prediction can be extended to estimate the temperature for the GPU since the Odroid-XU3 board provides temperature and current/voltage sensors for the GPU. Therefore we could also evaluate the division of workloads between the CPU clusters with the GPU, taking into account the energy/performance and temperature trade-offs. The proposed approach could be applied to single ISA heterogeneous multicore platforms. This could require: (i) power monitors or an accurate power model, and (ii) temperature monitors.

References

[1] A. M. Rahmani, M. H. Haghbayan, A. Miele, P. Liljeberg, A. Jantsch, and H. Tenhunen, "Reliability-aware runtime power management for many-core systems in the dark silicon era," *IEEE Trans. on VLSI Syst.*, vol. 25, no. 2, 2017.

[2] Y. G. Kim, M. Kim, and S. W. Chung, "Enhancing Energy Efficiency of Multimedia Applications in Heterogeneous Mobile Multi-core Processors," *IEEE Transactions on Computers*, vol. 66, no. 11, 2017.

[3] S. Yang, R. A. Shafik, G. V. Merrett, E. Stott, J. M. Levine, J. Davis, and B. M. Al-Hashimi, "Adaptive Energy Minimization of Embedded Heterogeneous Systems using Regression-based Learning," in *Int'l Workshop on Power and Timing Modeling, Optim. and Sim.*, 2015.

[4] A. K. Coskun, T. S. Rosing, and K. Whisnant, "Temperature aware task scheduling in MPSoCs," *EDAA*, 2007.

[5] A. Das, B. Al-Hashimi, and G. Merrett, "Adaptive and hierarchical runtime manager for energy-aware thermal management of embedded systems," *ACM Transaction on Embedded Computing Systems*, vol. 15, 2016.

[6] (2016) Exynos 5 octa (5422). [Online]. Available: www.samsung.com/exynos/

[7] A. Pathania, Q. Jiao, A. Prakash, and T. Mitra, "Integrated cpu-gpu power management for 3d mobile games," in *2014 51st ACM/EDAC/IEEE Design Automation Conference (DAC)*, June 2014, pp. 1–6.

[8] K. R. Basireddy, A. K. Singh, D. Biswas, G. V. Merrett, and B. M. Al-Hashimi, "Inter-cluster thread-to-core mapping and DVFS on heterogeneous multi-cores," *IEEE Transactions on Multi-Scale Computing Systems*, 2017.

[9] K. R. Basireddy, A. Singh, G. V. Merrett, and B. M. Al-Hashimi, "Itmd: run-time management of concurrent multi-threaded applications on heterogeneous multi-cores," in *Conference on Design, Automation and Test in Europe 2017 (DATE'17)*, January 2017.

[10] C. Bienia, S. Kumar, J. P. Singh, and K. Li, "The parsec benchmark suite: Characterisation and architectural implications," *Princeton University Technical Report TR-811-08*, 2008.

[11] A. Prakash, H. Amrouch, M. Shafique, T. Mitra, and J. Henkel, "Improving mobile gaming performance through cooperative CPU-GPU thermal management," *DAC '16 Design Automation Conference*, vol. 47, 2016.

[12] A. K. Coskun, T. S. Rosing, and K. Gross, "Utilizing predictors for efficient thermal management in multiprocessor socs," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 28, 2009.

[13] L.Ljung, *System Identification: Theory for the User (2nd Edition).* Upper Saddle River, NJ: Prentice-Hal PTR, 1999.

[14] Y. Ge, Q. Qiu, and Q. Wu, "A multi-agent framework for thermal aware task migration in many-core systems," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 20, 2010.

[15] G. Singla, G. Kaur, A. K. Unver, and U. Y. Ogras, "Predictive dynamic thermal and power management for heterogeneous mobile platforms," in *Design Automation and Test in Europe Conference*, 2015.

[16] G. Bhat, G. Singla, A. K. Unver, and U. Y. Ogras, "Algorithmic optimization of thermal and power management for heterogeneous mobile platforms," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 26, no. 3, pp. 544–557, March 2017.

[17] N. Peters, D. Füß, S. Park, and S. Chakraborty, "Frame-based and thread-based power management for mobile games on hmp platforms," in *2016 IEEE 34th International Conference on Computer Design (ICCD)*, Oct 2016, pp. 169–176.

[18] S. Pagani, H. Khdr, J. Chen, M. Shafique, M. Li, and J. Henkel, "Thermal safe power (tsp): Efficient power budgeting for heterogeneous manycore systems in dark silicon," *IEEE Transactions on Computers*, vol. 66, no. 1, pp. 147–162, Jan 2017.

[19] G. Bhat, S. Gumussoy, and U. Y. Ogras, "Power-temperature stability and safety analysis for multiprocessor systems," *ACM Trans. Embed. Comput. Syst.*, vol. 16, no. 5s, pp. 145:1–145:19, Sep. 2017. [Online]. Available: http://doi.acm.org/10.1145/3126567

[20] A. Weissel and F. Bellosa, "Process cruise control: event-driven clock scaling for dynamic power management," in *CASES '02 International conference on Compilers, architecture, and synthesis for embedded systems*, 2002, pp. 238–246.

[21] 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.

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

[23] 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.

[24] 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.

[25] 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.

[26] 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.

[27] 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.

[28] 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, 2012, pp. 213–224.