IEEE TRANSACTIONS ON
VERY LARGE SCALE INTEGRATION (VLSI) SYSTEMS

IEEE Xplore® Digital Library
http://ieeexplore.ieee.org/Xplore

Algorithmic Optimization of Thermal and Power Management for Heterogeneous Mobile Platforms

Ganapati Bhat, Gaurav Singla, Ali K. Unver and Umit Y. Ogras

***Abstract*—State-of-the-art mobile platforms are powered by heterogeneous system-on-chips that integrate multiple CPU cores, a GPU, and many specialized processors. Competitive performance on these platforms comes at the expense of increased power density due to their small form factor. Consequently, the skin temperature, which can degrade the experience, becomes a limiting factor. Since using a fan is not a viable solution for hand-held devices, there is a strong need for dynamic thermal and power management (DTPM) algorithms that can regulate temperature with minimal performance impact. This paper presents a DTPM algorithm which uses a practical temperature prediction methodology based on system identification. The proposed algorithm dynamically computes a power budget using the predicted temperature. This budget is used to throttle the frequency and number of cores to avoid temperature violations with minimal impact on the system performance. Our experimental measurements on two different octa-core big.LITTLE processors and common Android applications demonstrate that the proposed technique predicts the temperature with less than 5% error across all benchmarks. Using this prediction, the proposed DTPM algorithm successfully regulates the maximum temperature and decreases the temperature violations by one order of magnitude, while also reducing the *total* power consumption on average by 7% compared to the default solution.**

***Index Terms*—Dynamic power management, thermal management, heterogeneous computing, multi-processor systems-on-chip, multicore architectures.**

I. Introduction

THE phenomenal growth in the demand for mobile devices has been pushing the limits of multiprocessor system-on-chips, (MPSoC), which power majority of the mobile devices in use today. The number and capacity of the CPU cores increase at a steady rate, while the degree of heterogeneity grows with the use of asymmetric cores and accelerators such as GPU and audio/video processors. Larger computational power increases the power dissipation of the device. This, in turn, intensifies the skin temperature and reduces the battery life. Indeed, recent results have shown that the skin temperature is a performance limiter in mobile devices [1, 2], while rapid changes in the temperature degrades reliability [3].

A variety of design and runtime approaches have been developed to maximize the performance during high activity periods, and minimize the power consumption when there is little activity. For instance, idle power management determines the number of active cores, while dynamic voltage-frequency scaling (DVFS) controls the operating frequency of active resources to match the system performance to the application requirements [4, 5, 6]. Newly emerged big.LITTLE architectures work in tandem with these techniques by combining high performance (big) and energy efficient (little) clusters [7]. The big cores are utilized when high performance is needed, whereas the little cores are used during low activity periods. A recent example of this architecture is the Samsung Exynos chip, which integrates four A15 (big) and four A7 (little) cores. Our measurements on this chip show $10\times$ performance and $30\times$ power consumption range between the lowest power and the highest performance configurations. In addition, moderate to high activity workloads easily raise the temperature beyond acceptable levels, as shown in Figure 1. Our experimental platform [8] employs a fan to address this problem. However, using a fan is *not feasible* for mobile platforms, such as smartphones and tablets. Likewise, passively waiting for thermal violations, and reactively throttling the cores degrades the performance as well as reliability by causing large temperature variations [9]. As a result, there is a strong need for DTPM approaches for heterogeneous architectures to effectively regulate temperature with minimal performance impact.

This paper presents a predictive DTPM algorithm for heterogeneous mobile platforms. The first step of the proposed solution is a broadly applicable methodology for generating power and thermal models for heterogeneous mobile platforms. This methodology starts with basic principles, and then generates mathematical models that enable accurate power/thermal predictions tailored to the mobile platform of interest. We validate the proposed models empirically on two different MPSoCs, using a variety of benchmarks. The second step is a novel runtime technique to periodically compute the power budget, and throttle the processing cores to avoid temperature violations with minimal impact on performance. The major knobs offered by big.LITTLE architectures to meet the power budget are CPU cluster (big or little), the number of active cores, the operating frequency (hence voltage) of the CPU and GPU cores, and the state of active accelerators, such as audio and image processors. Since the use of accelerators is largely governed by the application code and compiler, the proposed algorithm uses the rest of the knobs to constrain the temperature with minimal performance impact.

The novel contributions of this paper are as follows:

- A methodology for generating power and thermal models for heterogeneous MPSoCs, and experimental validation *using two different* commercial big.LITTLE platforms [8],

Ganapati Bhat and Umit Y. Ogras are with the School of Electrical, Computer and Energy Engineering, Arizona State University, Tempe, AZ. E-mail: {gmbhat,umit}@asu.edu
Gaurav Singla was with the School of Electrical, Computer and Energy Engineering, Arizona State University, Tempe, AZ and is currently at ARM Holdings. E-mail: grsingla@asu.edu
Ali K. Unver is with the Assembly & Test Technology Development, Intel Corporation, Chandler, AZ. E-mail: Ali.k.unver@intel.com
Manuscript received May 2017

Digital Object Identifier: 10.1109/TVLSI.2017.2770163

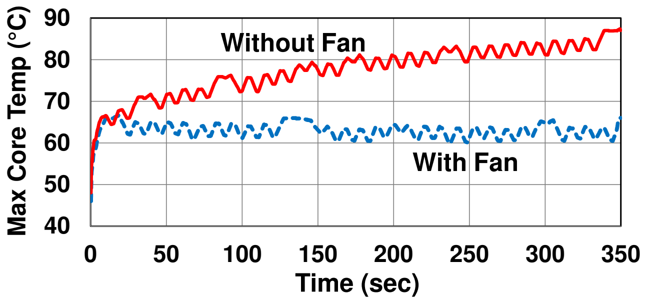


Fig. 1: Maximum core temperature with and without the fan.

- An algorithmic approach for dynamically computing the power budget using temperature prediction,
- A DTPM algorithm based on the performance gradient and power budget prediction,
- Exhaustive experimental evaluation on two commercial MPSoCs. Our experiments demonstrate that the proposed DTPM algorithm successfully regulates the maximum temperature and decreases the variation by one order of magnitude, while also reducing the *total* power consumption on average by 7% compared to the default solution.

The rest of this paper is organized as follows. The related work is reviewed in Section II. An overview of the proposed DTPM technique is provided in Section III. Power and thermal model generation methodologies are presented in Section IV. Thermal prediction and DTPM algorithm based on thermal prediction are described in Section V. Extensive experimental evaluation using Samsung Exynos 5410 and 5422 octa-core chips running a wide range of benchmarks is presented in Section VI. Finally, the conclusions and future research directions are discussed in Section VII.

II. Related Work

Many hardware and software-based thermal modeling techniques have been proposed for modern processors due to increased power densities and reliability implications. When the junction or skin temperature exceeds a safe operating point, the cores need to be throttled to reduce the temperature by lowering the power consumption. Therefore, the lack of a DTPM can lead to reduced performance due to throttling [10]. Similarly, different choices of DTPM have a significant impact on performance [11, 12, 13]. In particular, poor performance of reactive approaches has led researchers to develop compact thermal models [9, 14, 15, 16] and thermal prediction techniques [17, 18, 19, 20]. The work in [21] considers future temperature as linear extrapolation of its previous values. The thermal model we present in this work is similar to these approaches in using a linear time invariant system to predict temperature. However, instead of relying on material and design parameters to find the model coefficients, we use actual power, temperature measurements and system identification tools to find the parameters of the model. Utilization of temperature sensors and power meters [22, 23] makes our approach feasible and accurate.

Temperature is a strong function of power consumption. Therefore, accurate models of both dynamic and static power consumption play a critical role in temperature prediction. As technology is scaling, reduction in threshold voltage, channel length, and gate oxide thickness increase the leakage power component [24, 25]. The work in [26] proposes an approach to control the fan speed for cooling of data center servers by considering the dependence between leakage power and temperature. However, we deal with mobile platforms wherein fan is not an option. Simulators in [27, 28] assume a constant ratio between leakage and dynamic power. This assumption is not accurate since dynamic and leakage power's dependence on frequency, supply voltage, and temperature is different. We demonstrate in Section IV-A that leakage power is sensitive to temperature, while dynamic power is sensitive to frequency of operation and voltage.

Most of the prior work on thermal management has been done for homogeneous architectures, where all the cores possess similar architecture, power consumption, and performance abilities [29]. Heterogeneous processors increase thermal management complexity, as multiple resources have to be taken into consideration and the temperature of all these resources depend on each other [30]. For instance, 3-D multicore architectures are taken into consideration in [31, 32, 33]. An optimal control algorithm for fractal workloads running on multicore architectures with multiple voltage-frequency islands is proposed in [34]. A hierarchical power management technique for asymmetric processors is presented in [5], where the authors try to optimize the energy/performance trade-off under thermal design power constraints. Dynamic thermal management algorithms using task migration are explored in [35, 36]. CPU-GPU control frameworks for reducing the energy consumption and optimally allocating the power budget are presented in [37] and [38], respectively. Similarly, a control theoretic CPU-GPU thermal management algorithm to improve gaming performance is presented in [39]. However, this study considers a single temperature for the CPU, GPU, and the system, respectively. Another class of thermal management algorithms target maintaining a sustained performance under thermal limitations. For example, the work in [10] proposes a closed loop QoS control policy to minimize the thermal impact of extending the sustainability of desired QoS levels.

Unlike these studies, our approach calculates a precise power budget based on thermal prediction. The resources of heterogeneous architecture are utilized to distribute this power budget and control thermal violations. The algorithm targets heterogeneous platforms, but can also be used by other architectures. While most of the thermal management techniques are implemented and validated in a simulation environment, we demonstrate our technique on a commercial big.LITTLE platform [8]. The used platform offers new capabilities such as big/little clusters, and detailed power and temperature sensors. Since developing simulation models for new processors is obstructed by the difficulties in finding the exact floorplan, heat sink information, and parameter values, researchers are usually limited to few examples such as simple XScale core and Alpha processor [18, 40]. We plan to open our power and thermal models to the public in order to enable research on emerging heterogeneous platforms.

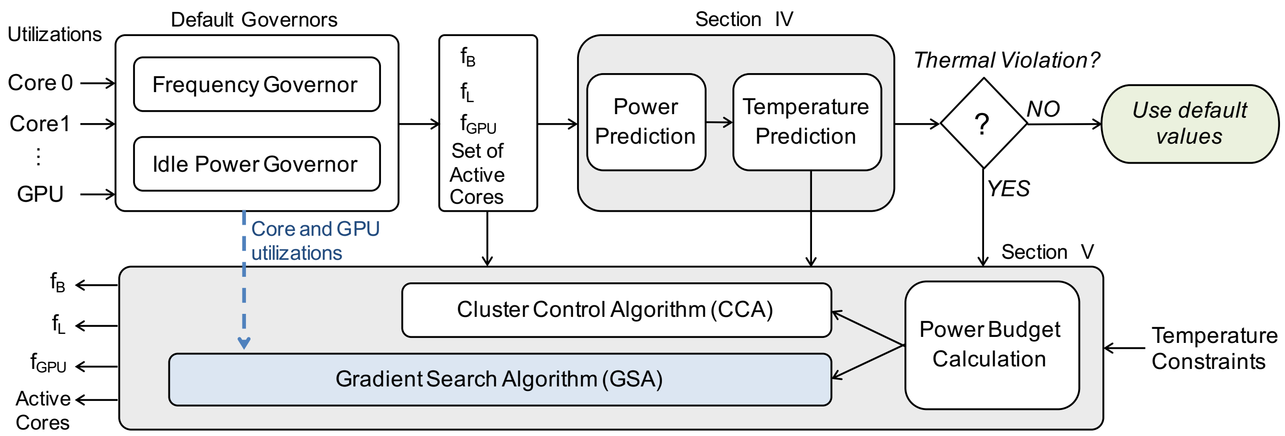


Fig. 2: High-level description of the DTPM algorithm. Sections where individual blocks are described in detail are annotated. Additional inputs required by the proposed gradient search algorithm are shown with dashed lines.

III. Overview of the Proposed Framework

The proposed DTPM framework is outlined in Figure 2. Modern mobile platforms are highly integrated closed systems with tight hardware and software module interactions. Therefore, techniques that target these platforms cannot be designed or evaluated in isolation. To this end, all the models and algorithms proposed in this paper are incorporated with the existing software infrastructure. As such, existing frequency and idle state governors, as well as the device specific drivers, remain intact.

In the proposed DTPM flow, the default policy, such as the on-demand governor [41], determines the operating frequency of each core. Before taking any action, the proposed power model uses these choices to predict the power consumption. Then, the power consumption predictions are fed to the thermal model to predict the temperature that would be reached if these actions were taken. Therefore, different governors or device specific optimizations implemented in dedicated drivers can work in coordination with the proposed framework. Unless a thermal violation is predicted, the decisions of the default drivers are affirmed based on the DTPM algorithm. Thus, the proposed DTPM approach is non-intrusive when the temperature is within permissible levels. However, when a temperature violation is predicted, the algorithm computes the maximum power consumption that does not result in a temperature violation. In general, there may be multiple configurations that can meet the power budget constraint. Since each feasible solution leads to a different loss in performance, it is critical to choose the configuration with the highest performance. We first present an overview of the cluster-based control presented in [42]. This algorithm starts with the temperature constraint and works backwards to determine the maximum power consumption that can be tolerated. It finds *only* the largest number of big cores and the big cluster frequency that satisfies the power budget, since the big cores have the highest performance impact. Second, we present a new algorithm, which performs a gradient search at runtime to find the feasible CPU configuration and GPU frequency with minimum performance loss. The newly proposed algorithm exploits more data, such as core and GPU utilizations, as shown by the dotted arrow in Figure 2. Finally, we overwrite the set of active resources and their frequencies such that the temperature constraint violation can be prevented.

IV. Power/Thermal Modeling Methodology

Effective power and temperature management depends critically on accurate analytical models that can be evaluated at runtime. Therefore, we start with presenting our modeling methodology that leverages thermal and power sensors. Our approach is applicable to arbitrary number of temperature and power consumption sensors. In our experimental evaluations, we employ thermal sensors to measure the temperature of each big core and the GPU. Similarly, current sensors are used to measure the power consumption of the big core cluster P_{A15}, little core cluster P_{A7}, GPU P_{GPU}, and memory P_{mem}.

A. Power Modeling

Power models have been discussed extensively in the literature [3] for major components, such as, CPU, GPU, display, and battery. Therefore, we only elaborate our empirical approach to extract the leakage current and switching capacitance. For a general discussion of processor power estimation techniques, we refer the reader to the survey presented in [43].

Let the number of resources whose frequency can be controlled at runtime be M, and the frequency of resource j be f_j, $1 \leq j \leq M$. The total power consumption of the j^{th} resource can be written as the sum of the dynamic and leakage power as:

$$\begin{aligned} P_{j,total} &= P_{j,dynamic} + P_{j,leakage} \\ P_{j,total} &= \alpha_j C_j V_j^2 f_j + V_j I_{j,leakage} \end{aligned} \quad (1)$$

where α_j is the activity factor, C_j is the switching capacitance, V_j is the operating voltage, and f_j is the operating frequency. Leakage current can be expressed as:

$$\begin{aligned} I_{j,leakage} &= A_s \frac{W}{L} \frac{kT_j^2}{q} e^{\frac{q(V_{GS}-V_{th})}{nkT_j}} + I_{j,gate} \\ I_{j,leakage} &= c_{j,1} T_j^2 e^{\frac{c_{j,2}}{T_j}} + I_{j,gate} \end{aligned} \quad (2)$$

TABLE I: The list of major parameters

Symbol	Description
M	Number of processing elements (resources) in the SoC
$P[k]$	$M \times 1$ array where $P_j[k],\ 1 \le j \le M$ denotes the power consumption of the j^{th} resource
f_j	The operating frequency of the j^{th} resource
$\vec{\mathbf{f}}$	The vector of frequencies $[f_1, f_2, \ldots, f_M]^T$
N	Number of thermal hotspots
$T[k]$	$N \times 1$ array where $T_i[k],\ 1 \le i \le N$ denotes the temperature of the i^{th} hotspot
$\widehat{T}[k+n]$	The predicted temperature in interval $k+n$
A_s, B_s	Parameters of the discrete time state-space system, that describe the thermal dynamics (Equation 5)
b_{ij}	Entries of B_s for $1 \le i \le N, 1 \le j \le M$
T_i^*	Maximum thermally safe temperature

where A_s is a technology dependent constant, L and W are channel length and width, k is the Boltzmann constant, T_j is the temperature, q is the electron charge, V_{GS} is the gate to source voltage, V_{th} is the threshold voltage, n is the sub-threshold swing coefficient, and $I_{j,gate}$ is the gate leakage current [44, 45]. In the second line of Equation 2, the technology and device parameters are condensed into generic parameters denoted by $c_{j,1}$ and $c_{j,2}$ for each of the resources. Next, we describe the step-by-step process to find the unknown parameters, assuming that dynamic power shows negligible variation with temperature. Our models can be used not only for DTPM as we demonstrate in this work, but also for power-temperature stability analysis [46].

Leakage Power Characterization: By using Equation 1 and Equation 2, the total power consumption is expressed as:

$$P_{j,total} = \alpha_j C_j V_j^2 f_j + V_j(c_{j,1} T_j^2 e^{\frac{c_{j,2}}{T_j}} + I_{j,gate}) \quad (3)$$

Equation 3 has four unknowns, $\alpha_j C_j$, $c_{j,1}$, $c_{j,2}$, and $I_{j,gate}$ for a given operating frequency, voltage, and temperature. This equation needs to be solved for each resource to get the values of the corresponding parameters. To estimate these parameters for our target platform, we place the target platform in a furnace as shown in Figure 3. Next, we set the temperature of the furnace to 40°C. During the characterization, we use a light workload running only on the j^{th} resource with fixed f_j and V_j such that the dynamic power does not increase the temperature. Therefore, the operating frequency f_j, voltage V_j and temperature T_j in Equation 3 are known. Then, we measure the total power consumption as a function of time, as illustrated in Figure 4. We repeat the same measurement by sweeping the temperature of the furnace in increments of 10°C up to 80°C. As a result, we obtain multiple sets of power consumption measurements and four unknowns. Finally, we employ nonlinear curve fitting to find the unknowns and model the leakage power.

The proposed analysis is repeated for each computing resource, such as the little cores and GPU, to find the unknowns in the leakage power model. We validate this approach on both Exynos 5410 and Exynos 5422 SoCs, as depicted in Figure 5 and Figure 6, respectively. Next, we discuss runtime computation of activity factor α and switching capacitance C.

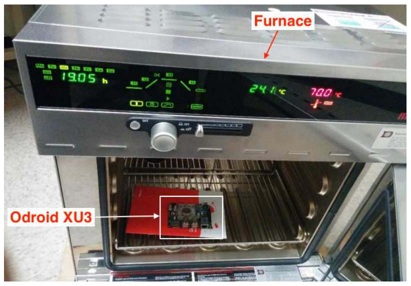


Fig. 3: Experimental setup for leakage power characterization.

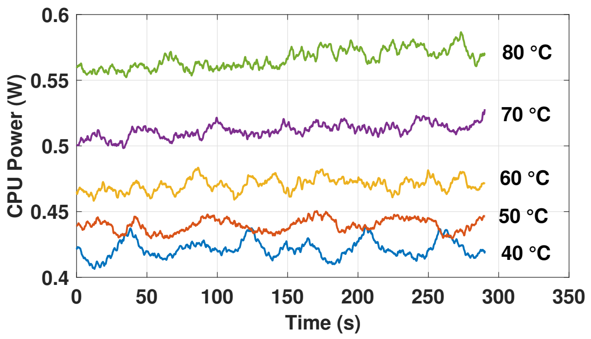


Fig. 4: Power measurements on the Odroid-XU3 platform at different temperatures with light dynamic activity.

Runtime computation of αC: We compute the activity factor αC every 100 ms at runtime in the Linux kernel. The steps in computing αC in the Linux kernel are as follows:

1) Read the registers that provide the current power consumption and temperature readings.
2) Compute the leakage power using the temperature and the leakage power model.
3) Subtract the leakage power from step 2 from the total power to get the dynamic power, as shown in Figure 7.
4) Divide the dynamic power in step 3 by V^2f to compute αC.

The second step of this calculation (i.e., the leakage power model) requires the technology dependent parameters of the platform. These parameters are extracted offline, as explained under leakage power characterization, while the rest of the operations are performed at runtime.

We use the proposed approach to predict the dynamic power consumption before any decision on the frequency is made. The accuracy of leakage and total power prediction is demonstrated in Figure 5 and Figure 6 for Exynos 5410 and Exynos 5422 SoCs, respectively. Our power model results in less than 5% prediction error on average as compared to measurements across a wide variety of workloads. Further experimental evaluation under common benchmarks is presented in Section VI-B.

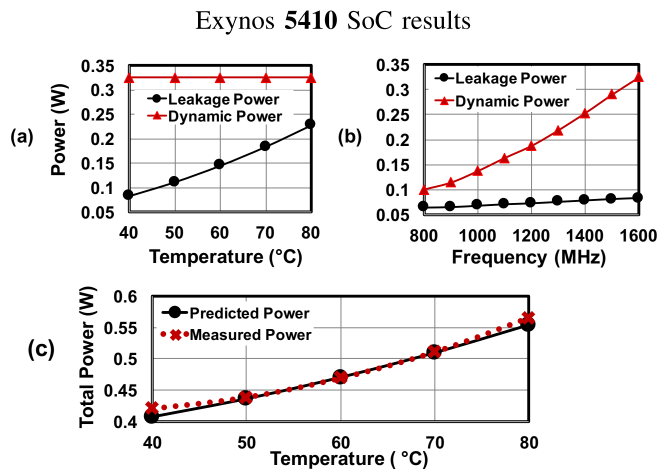


Fig. 5: Variation of leakage and power model with (a) temperature, and (b) frequency, (c) Predicted and measured power for Exynos *5410*.

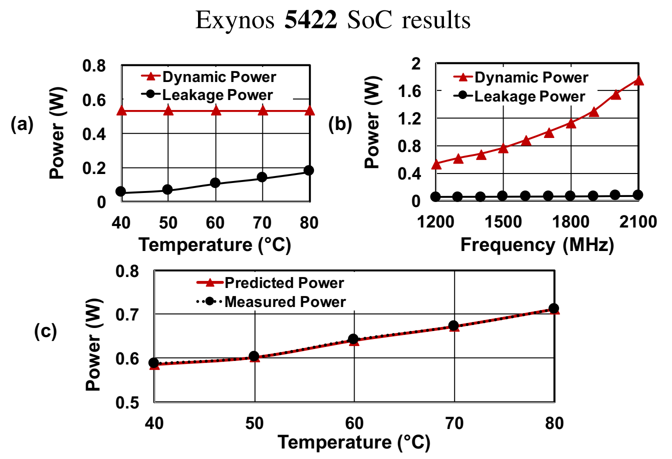


Fig. 6: Variation of leakage and power model with (a) temperature, and (b) frequency, (c) Predicted and measured power for Exynos *5422*.

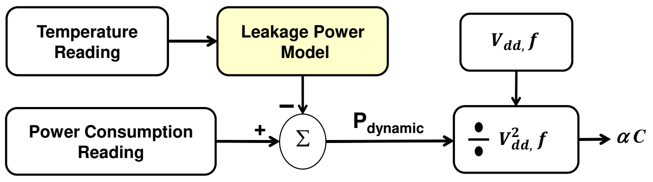


Fig. 7: Runtime procedure for computing the product of the activity factor and switching capacitance.

B. Thermal Modeling

Using the duality between the thermal and electrical networks, one can model the dynamics of the temperature using a state-space model [18]. Suppose that there are N nodes in the network, whose temperature and power consumption are given by $[T(t)]_{N\times 1}$ and $[P(t)]_{N\times 1}$, respectively. Then,

$$C_t \frac{dT}{dt} = -G_t T(t) + P(t) \tag{4}$$

where C_t and G_t are the thermal capacitance and conductance matrices [9]. Power/temperature measurements and control actions are performed periodically in OS kernels or firmware in practice. Hence, we discretize Equation 4 assuming a sampling period of T_s, and obtain the following equations:

$$\begin{aligned} T[k+1] &= (I - T_s C_t^{-1} G_t)T[k] + T_s C_t^{-1} P[k] \\ T[k+1] &= A_s T[k] + B_s P[k] \end{aligned} \tag{5}$$

Finding the thermal conductance and capacitance matrices using finite element simulations or a thermal modeling framework like Hotspot [9] would require detailed design information such as floorplan, heat sink geometry, and material properties, which are either not public or very difficult to obtain. Furthermore, validating the thermal model would still require actual power and temperature measurements. Therefore, we start directly with actual measurements, and then employ system identification to find C_t and G_t, as detailed next.

System Identification: The output of the state-space model given in Equation 5 is the temperature of the thermal hotspots, while the inputs are the corresponding power dissipations. In general, the system is not fully observable, i.e., we cannot measure the temperature at each node of the thermal network. In our mathematical formulation, the hotspots are the points whose temperature can be measured. This corresponds to temperatures of four big cores and the GPU on our experimental platform. In general, the hotspots should be chosen as the locations on the chip with the largest power density, since they are more likely to have higher temperatures. The chip designers need to make the temperature at these points observable by either using thermal sensors or analytical models that use software counters. Similarly, the power consumption of only a subset of the resources can be measured. For instance, in our system, the temperatures of the big cores and the GPU are observable, while the power consumption of the little and big core clusters, the GPU, and memory can be measured. Therefore, we perform system identification to characterize the temperature dynamics of the observable thermal hotspots in terms of $P[k]$. The resulting entries in A_s describe how the temperatures in time step k affect the temperature in the following interval. That is, the A_s matrix captures the coupling between different thermal hotspots in time. The entries in B_s describe how each power source affects the observable thermal hotspots. For example, the first column of B_s describes how the little cluster power consumption $P_L[k]$ (i.e., the first entry in $P[k]$) affects the temperature of each thermal hotspot.

In order to obtain an accurate characterization, we control each resource separately, except for memory, while keeping the remaining ones turned off or running at their minimum frequency using the powersave governor. For example, when profiling the big cores, we run a single little core at the lowest frequency and turn off the GPU using the dynamic hotplug mechanism in the Linux kernel. Memory is excluded since the target platform does not provide memory frequency control. With this setup, we oscillate the frequency of the active cluster between its minimum and maximum values using a pseudorandom bit sequence (PRBS). The PRBS input is generated to cover a frequency spectrum, which is broader than that excited by an arbitrary application. During each experiment, we record the input power series $P[k]$ and the output temperature $T[k]$. The data recorded using the PRBS sequence for each resource is used to estimate the column corresponding to the respective resource in the B_s matrix. To further improve the accuracy of modeling A_s, we perform one more experiment. We first apply a heavy dynamic activity to drive all the temperature hotspots close to their maximum safe values. Then, we turn off all the cores to pull the power dissipation $P[k]$ close to zero, and let the system cool down. The power and temperature measurements during the cool down period reveal the natural response of the system. Finally, we employ the system identification toolbox of Matlab, and our input/output measurements to find A_s and B_s in Equation 5. The mean squared error in the system identification is less than 9%, which indicates a good fit to the measurements. The technique is scalable and can be implemented for any system.

C. Thermal Prediction and Validation

The dynamics of the temperature as a function of the power consumption is governed by the state-space system in Equation 5. We can use this state-space equation to predict the temperature at an arbitrary number of time steps ahead. More precisely, we derive the temperature at time step $k+n$ as:

$$\widehat{T}[k+n] = A_s^n T[k] + \sum_{i=0}^{n-1} A_s^i B_s P[k+n-i-1] \quad (6)$$

Before changing the frequency of a core, we predict the power consumption at the new frequency using Equation 1. Then, we plug the power consumption to Equation 6 to predict the resulting temperature. We confirmed that all the eigenvalues of A_s are within the unit circle, as expected for a practical system. Hence, bounded power consumption inputs $P[k]$ will lead to bounded output. Since the future dynamic activity is unknown, we assume that the frequency stays constant while evaluating Equation 6. However, we account for the increase in the leakage power using the predicted temperature at each interval and our leakage model. This equation is implemented in Linux kernel, and its accuracy is validated by comparing the temperature predictions with actual measurements. More precisely, we predict four big core temperatures and the GPU temperature using the proposed thermal model *at every control interval*, i.e., every 100 ms. Therefore, the predictions are periodically corrected leading to the accurate results. If the maximum of these predictions exceeds the temperature constraint, then the proposed DTPM algorithm is triggered in that interval. Therefore, we are interested in the error between the predicted *maximum temperature* and the measured maximum temperature. Figure 8 shows that the prediction error is less than 3% (1°C) up to 1 second, while the error is within 7% (2.5°C) for as long as 5 seconds when running the Vellamo benchmark, as shown in Figure 8. The temperature prediction error grows with the time horizon primarily due to increased difference between the actual and predicted power consumption. Further evaluation using the complete set of benchmarks is presented in Section VI.

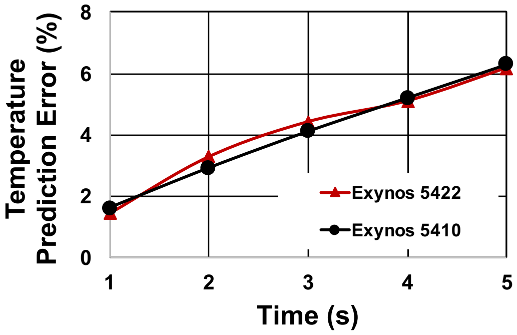


Fig. 8: The error in predicting the maximum temperature for the Vellamo benchmark.

V. Dynamic Thermal and Power Management

Existing governors implemented in commercial mobile systems increase the operating frequencies and the number of active cores to meet the demands of active applications. Higher performance, in turn, increases the power dissipation and generates thermal hotspots. Then, the resulting thermal violations are avoided by throttling the cores at the expense of performance loss. Since throttling penalizes the performance significantly, avoiding thermal violations proactively improves the performance by using the thermal headroom more efficiently [10, 42]. The DTPM algorithm presented in this section achieves this objective using the power and thermal models presented in Section IV. First, it dynamically computes the power budget that ensures no thermal violation will happen. Then, this power budget is used at runtime to limit the types, number, and frequencies of active resources.

A. Runtime Power Budget Computation

Let the maximum thermally safe temperature be given by T_{constr}. Since each thermal hotspot needs to stay below this value, the temperature constraint for each thermal hotspot can be written as $[T_{constr}]_{N\times 1}$. Using Equation 5, we can express the temperature constraints as:

$$\begin{aligned} |\widehat{T}[k+1]| &\leq |T_{constr}| \\ |A_s T[k] + B_s P[k]| &\leq |T_{constr}| \end{aligned} \quad (7)$$

where the $|\cdot|$ represents the norm operation. Since thermal control algorithms typically use the maximum temperature, we employ the L_∞ norm and denote $|T_{constr}|_\infty = T_{max}$ to re-write the temperature constraint as:

$$\begin{aligned} |A_s T[k] + B_s P[k]|_\infty &\leq T_{max} \\ max\ \{A_{s,i} T[k] + B_{s,i} P[k]\} &\leq T_{max} \quad 1 \leq i \leq N \end{aligned} \quad (8)$$

where $A_{s,i}$ and $B_{s,i}$ denote the i^{th} row of matrices A_s and B_s, respectively. Hence, we convert the matrix inequality into a set of scalar inequalities, one for each thermal hotspot. Temperature constraints can be written as:

$$B_{s,i} P[k] \leq T_{max} - A_{s,i} T[k] \quad 1 \leq i \leq N \quad (9)$$

The right hand-side is known since we obtained A_s and B_s through system identification and measured $T[k]$. We call this value the thermal headroom for the i^{th} hotspot, and denote it as $T_i^* = T_{max} - A_{s,i}T[k]$ for $1 \leq i \leq N$. The left hand side can be further expanded as:

$$B_{s,i}P[k] = \sum_{j=1}^{M} b_{ij}P_j[k] \leq T_i^* \tag{10}$$

where $P_j[k]$ is the power consumption of the j^{th} resource, and b_{ij}'s are the entries of B_s, as summarized in Table I. For example, $P_1[k]$–$P_4[k]$ in our system are the power consumption of the little cluster, big cluster, memory, and GPU, respectively. Equation 10 has multiple solutions. In order to obtain a unique solution, we propose to maximize the performance while maintaining the temperature below the maximum temperature constraint. The run time algorithms proposed to achieve this objective is described next.

B. Predictive DTPM Algorithm using Gradient Search

Accurate temperature predictions can be used to design proactive algorithms that can prevent thermal violations. We presented a particular example that uses power budget given by Equation 10 in [42]. In what follows, we overview this algorithm and discuss its limitations. Then, we present a novel predictive algorithm based on gradient search.

CPU Cluster Oriented Algorithm (CCA) [42] This algorithm first checks whether the core configuration and frequency chosen by the default governors can lead to a thermal violation. If a violation is detected, the algorithm uses the inequality given by Equation 10 to determine how much reduction in the power consumption is required to meet the budget. Since $P[k]$ is a vector where each entry corresponds to the power consumption of different sources, there are many different ways of decreasing the frequencies to satisfy this inequality. Furthermore, each alternative solution has a different performance impact as a function of the workload. Hence, the general solution of this matrix inequality is nontrivial. The algorithm presented in [42] finds the largest big cluster frequency that satisfies the power budget without reducing the frequency of the other resources. Hence, we refer to this as the CPU cluster oriented algorithm. If the lowest possible big cluster frequency (f_{low}) leads to a thermal violation, then it resorts to reducing the GPU frequency. Finally, CCA moves the application to the little cluster and turns off the big cluster as the last resort.

CCA relies on the empirical observation that the big CPU cluster has the largest impact on thermal hotspots. Since the big cluster may also have larger impact on performance, a more general solution should also consider performance. Similarly, CCA is quite effective when only one CPU cluster can be active at any given time, as in Samsung Exynos 5410. In general, both clusters and other processing elements may be active at the same time. Next, we present a gradient search algorithm (GSA) that overcomes these limitations.

*1) **Problem Formulation:*** Using the frequency f_j and utilization ρ_j for each resource $1 \leq j \leq M$, we can approximate the objective function as the execution or response time of an application as:

$$O(\vec{\mathbf{f}}) = \sum_{j=1}^{M} \frac{\rho_j}{f_j} + t_0 \tag{11}$$

where the parameters $\rho_j \geq 0$ represent the weighted effect of f_j on the execution time, and t_0 accounts for the frequency independent fraction [47]. This equation helps us quantify the relative importance of a certain computing resource on performance. We need to minimize this objective function under the power budget constraints given in Equation 10. The inequality in Equation 10 is written in terms of power consumption, while the objective function is written in terms of frequency. Hence, we re-write this inequality by substituting Equation 1 into Equation 10 as:

$$\sum_{j=1}^{M} b_{ij}P_j[k] \leq T_i^*,\ 1 \leq i \leq N \tag{12}$$
$$\sum_{j=1}^{M} b_{ij}\left[\alpha C_j V_j^2[k] f_j[k] + V_j[k] I_{j,leakage}\right] \leq T_i^*,\ 1 \leq i \leq N$$

The voltage V_j is typically a linear function of the operating frequency [47, 48]. Therefore, we approximate the voltage as $V_j[k] = \beta f_j[k] + \gamma$. By substituting this relation to Equation 12, the power consumption of the j^{th} resource can be expressed as a cubic function of the frequency:

$$\begin{aligned} P_j[k] &= \alpha_j C_j(\beta f_j[k] + \gamma)^2 f_j[k] + \\ &\quad (\beta f_j[k] + \gamma)(c_{j,1}T_j^2 e^{\frac{c_{j,2}}{T_j}} + I_{j,gate}) \\ &= \alpha_j C_j \beta^2 f_j[k]^3 + 2\alpha_j C_j \beta\gamma f_j[k]^2 + \\ &\quad (\alpha_j C_j \gamma^2 + \beta c_{j,1} T_j^2 e^{\frac{c_{j,2}}{T_j}} + \beta I_{j,gate}) f_j[k] + \\ &\quad \gamma(I_{j,gate} + c_{j,1}T_j^2 e^{\frac{c_{j,2}}{T_j}}) \end{aligned} \tag{13}$$

For notational convenience, we define *non-negative* coefficients $m_{0j} - m_{3j}$:

$$\begin{aligned} m_{3j} &\triangleq \alpha_j C_j \beta^2, \quad m_{2j} \triangleq 2\alpha_j C_j \beta\gamma \\ m_{1j} &\triangleq (\alpha_j C_j \gamma^2 + \beta c_{j,1} T_j^2 e^{\frac{c_{j,2}}{T_j}} + \beta I_{j,gate}), \\ m_{0j} &\triangleq \gamma(I_{j,gate} + c_{j,1}T_j^2 e^{\frac{c_{j,2}}{T_j}}) \end{aligned} \tag{14}$$

With this simplification, the constraint can be re-written as:

$$\sum_{j=1}^{M} b_{ij}\left[m_{3j}f_j[k]^3 + m_{2j}f_j[k]^2 + m_{1j}f_j[k] + m_{0j}\right] \leq T_i^* \tag{15}$$

for $1 \leq i \leq N$. Thus, using Equations 11 and 15, the optimization problem can be formulated as:

$$\begin{aligned} &minimize \quad O(\vec{\mathbf{f}}) = \sum_{j=1}^{M} \frac{\rho_j}{f_j} + t_0 \\ &subject\ to \quad \sum_{j=1}^{M} b_{ij}(m_{3j}f_j^3 + m_{2j}f_j^2 + m_{1j}f_j + m_{0j}) \leq T_i^* \\ &\qquad\qquad\ \ for \quad 1 \leq i \leq N \end{aligned} \tag{16}$$

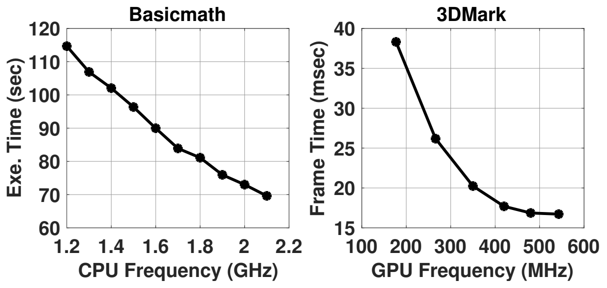


Fig. 9: Platform data demonstrating the performance as a function of the operating frequency for CPU and GPU bound benchmarks. Basicmath is a CPU bound benchmark and 3DMark is a GPU bound benchmark.

where the time index k is dropped, since this problem is solved at each control interval.

*2) **Solution Approach**:* The gradient of the objective function can be found as $\nabla(\vec{\mathbf{f}}) = [-\rho_1/f_1^2, -\rho_2/f_2^2, \ldots, -\rho_M/f_M^2]^T$. By taking the second order derivative with respect to the frequencies, the Hessian matrix $\mathbf{H}$ of the objective function $O(\vec{\mathbf{f}})$, can be obtained as:

$$\mathbf{H}_{i,j} = \begin{cases} \frac{2\rho_j}{f_j^3} & i = j, 1 \le i, j \le M, \\ 0 & \text{otherwise.} \end{cases} \tag{17}$$

We can see that all the nonzero entries of the Hessian are positive as $\rho_j \ge 0$ and $f_j > 0$, $1 \le j \le M$. Using Sylvester's criterion, we can see that the Hessian matrix of the objective function is positive definite [49]. Therefore, the objective function $O(\vec{\mathbf{f}})$ is a convex function of the operating frequencies. For illustration, Figure 9 shows the execution time and frame processing time for a CPU and a GPU bound application, respectively. We observe that the both are convex functions of the operating frequency.

Equation 16 is an inequality constrained convex optimization problem [50]. We know that it needs to be solved at runtime, *only if* the inequality constraint is violated. When this happens, the operating frequencies need to be decreased such that the power consumption reduces until the thermal constraints are satisfied. This, however, will also reduce the performance, i.e., lead to an increase in the objective function $O(\vec{\mathbf{f}})$. One could find the optimum set of frequencies by performing an exhaustive depth first search for all available frequency levels. However, performing an exhaustive search in the OS kernel at each iteration of the CPU frequency governor is prohibitive. The convexity of $O(\vec{\mathbf{f}})$ reveals that the performance loss will be minimized, if the frequencies are decreased along the direction that maximizes the gradient $\nabla(\vec{\mathbf{f}})$. Therefore, we propose the computationally efficient gradient search algorithm described next.

*3) **Gradient Search Algorithm**:* The GSA algorithm starts with predicting the temperature at a future time step $k+n$ using the current temperature vector $T[k]$. If the predicted temperature is greater than the threshold T_{max} for any hotspot, the algorithm intervenes to prevent the violation, as outlined in the pseudo-code shown Figure 10 (line 3). The gradient search starts with $\vec{\mathbf{f}}$ initialized to the values determined by the default governor. Let $f_j[k]$ be the current frequency of resource j, and δ_{fj} be the difference between $f_j[k]$ and the next lower frequency level. For example, if the current frequency is 1.3 GHz and the set of supported frequencies are $[1.2\text{ GHz}, 1.3\text{ GHz}, \ldots, 2\text{ GHz}]$, then $\delta_{fj} = 100$ MHz. The algorithm computes the projected performance loss of reducing the frequency of each resource, as follows:

```
Require: $Active\ cores, frequency, temperature$
 1: procedure GRADIENT SEARCH ALGORITHM
 2:     $\widehat{T}[k+n] \leftarrow$ Predict Temperature after n control intervals
 3:     if $\widehat{T}[k+n] \ge T_{max}$ for any thermal hotspot then
 4:         Evaluate the Constraint given by Equation 10
 5:         if Constraint is violated then
 6:             while Constraint is violated do OR $f_j \ne f_{low}\ \exists j$
 7:                 Evaluate $\vec{\Delta}[O(\vec{\mathbf{f}_k})]$ given by Equation 18
 8:                 if More than one resource has $\rho_j > 0$ then
 9:                     Find $j$ s.t. $\vec{\Delta}_j[O(\vec{\mathbf{f}_k})] = min(|\vec{\Delta}_j[O(\vec{\mathbf{f}_k})]|), \forall j$
10:                     Reduce frequency of resource $j$ by $\delta_{fj}$
11:                     Predict the $P[k+1]$ using the new $\vec{\mathbf{f}_k}$
12:                     $\widehat{T}[k+n] \leftarrow$ Predicted temperature after n intervals
13:                     Evaluate the Constraint given by Equation 10
14:                 else
15:                     Compute the power budget for active resource using
                        Equation 10
16:                     Solve for frequency that meets the calculated power
                        budget
17:                     break
18:                 end if
19:             end while
20:         end if
21:         if Thermal Constraint is violated AND $f_j = f_{low} \forall j$ then
22:             Turn off the hottest resource
23:         end if
24:         Apply frequency changes to the system
25:         return
26:     else
27:         Use default policy
28:     end if
29: end procedure
```

Fig. 10: Pseudo-code for the second DTPM algorithm based on **gradient** search.

$$\begin{aligned} \Delta_j[O(\vec{\mathbf{f}_k})] &= O(\vec{\mathbf{f}_k})\Big|_{f_j=f_j[k]} - O(\vec{\mathbf{f}_k})\Big|_{f_j=f_j[k]-\delta_{fj}} \\ &= \frac{\rho_j}{f_j[k]} - \frac{\rho_j}{f_j[k]-\delta_{fj}} \qquad 1 \le j \le M \\ &= -\rho_j \frac{\delta_{fj}}{f_j[k](f_j[k]-\delta_{fj})} \end{aligned} \tag{18}$$

After computing $\Delta_j[O(\vec{\mathbf{f}_k})]$ for each resource, we construct vector $\vec{\Delta}O(\vec{\mathbf{f}_k}) = [\Delta_1[O(\vec{\mathbf{f}_k})], \ldots, \Delta_M[O(\vec{\mathbf{f}_k})]]$. $\vec{\Delta}O(\mathbf{f}_k)$ is simply the gradient $\nabla(\vec{\mathbf{f}})$ evaluated at $\mathbf{f}_k$, which takes the discrete frequency values supported by the corresponding resource. In our implementation, ρ_j is the utilization of each resource, since the utilization is an indicator of relative importance of each resource and it is easy to profile in the kernel without any overhead. Once the gradient at the current operating point $\vec{\Delta}O(\mathbf{f}_k)$ is found, the algorithm checks if multiple resources are active or not. That is, we check if only one resource has $\rho_j > 0$. In such cases, we first calculate the power budget for the active resource using Equation 10. We can use Equation 10 to calculate the power budget since only one of the resources is active. Once the power budget is calculated, we solve for the frequency that meets the power budget. This avoids unnecessary gradient search. Otherwise, the algorithm finds the resource that has the minimum $|\vec{\Delta}_j O(f_j)|$. Since this implies the smallest performance penalty, the frequency of the corresponding resource is marked for reduction by one level, i.e., from $f_j[k]$ to $f_j[k] - \delta_{fj}$ (lines 8-10 in Figure 10). Using the new frequency level, we predict the new power consumption vector. Then, we use the projected power consumption to predict the temperature at a future time step $k + n$. If the new prediction still violates the temperature constraint, we re-iterate the loop between lines 8-19 in Figure 10. This continues until either the temperature constraint is met or each core reaches its minimum frequency. If the latter is true, there may be still a thermal violation after n control intervals. This is evaluated in lines 21-23 in Figure 10. If we still predict a thermal violation, the condition in line 22 will put the hottest core to sleep and the tasks running on it are migrated. Finally, we assign the new frequency levels and exit the algorithm.

VI. Experimental Evaluation

A. Experimental Setup and Methodology

1) Experimental Platforms: The proposed framework is evaluated using the Odroid-XU3 platform [8] powered by Samsung Exynos 5422 MPSoC. The Odroid-XU3 is a single ISA heterogeneous big.LITTLE processor composed of a big core cluster (4 A15 cores), little cluster (4 A7 cores), a GPU, and other basic components. Unlike the previous generation big.LITTLE architectures which were used in our previous study [42], Odroid-XU3 can run both the big and little clusters at the same time. This new capability allows scheduling the background tasks of the OS to the little cores, and running the intensive applications on big cores when needed. Built-in power sensors measure the power consumption of big core cluster, little core cluster, GPU, and memory separately, while external power meters enable logging the total platform power. The platform also provides temperature sensors located on each big core and GPU, which are used as the thermal hotspots in our evaluation.

2) Benchmarks: We employ 18 benchmarks to validate the proposed approach; 11 from the Mi-Bench embedded benchmark suite [51], 3 benchmarks from the PARSEC suite [52], 3 frequently used graphics benchmarks (3DMark Ice Storm Extreme, GLBench ALU, GLBench trex), and one custom matrix multiplication code, which is mainly used during debugging. The benchmarks and their relevant properties are summarized in Table II. PARSEC suite is selected to evaluate our algorithm on multi-threaded workloads. We run each PARSEC benchmark with four threads (i.e., the number of A15 cores in the system). It is also important to note that benchmarks run along with Android operating system and all other kernel background processes. Hence, there are many active threads in the system, even when running a single threaded benchmark. Therefore, multiple cores were active during the experiments and this number varied dynamically. Finally, the games and video benchmarks utilize the GPU more heavily, while the other benchmarks are CPU intensive.

TABLE II: Benchmarks used in the experiments.

Type	Benchmarks
Security	Blowfish, Sha
Network	Dijkstra, Patricia
Computational	Basicmath, Matrix Multiplication, Bitcount, Qsort, Blackscholes, Fluidanimate
Telecom	CRC32, GSM, FFT
Consumer	JPEG
Streaming	Streamcluster
Browser	Vellamo
Graphics	GLBench, 3DMark (GT1, GT2, Physics tests)

3) Comparisons: We compare the proposed gradient search algorithm (i.e., GSA) against the CPU Cluster Oriented Algorithm, (i.e., CCA) [42] and the default configuration without the fan, which is described next.

Default configuration without fan – This configuration uses the Linux kernel (3.10.9) that comes with the device. We disable the fan on the development board, since it *not feasible* when this chip is used in a smartphone or tablet. We note that the default intelligent power allocation algorithm (IPA), which is integrated into the Linux kernel [53] and the default CPU/GPU governors, are still active in this configuration. Disabling the fan has no impact during light activity, but the temperature increases more quickly for high loads, as shown in Figure 13. We use this configuration as a baseline for our performance, power consumption, and energy savings comparisons.

4) Implementation Details and Overhead: All the power and temperature models and the proposed algorithms are incorporated with the existing governors in the Linux 3.10.9 kernel, as illustrated in Figure 2. Our implementation consists of two major components. The power models presented in Section IV-A and the temperature prediction model (Equation 6) described in Section IV-C are implemented as custom functions that can be called by the frequency governors in the kernel. Likewise, the algorithm that controls the set of active cores and their frequencies is implemented as a patch in the cpufreq driver subsystem. After compiling the kernel with our modifications, we flash it to the device. If a temperature violation is predicted, we run the proposed algorithm outlined in Figure 10. Otherwise, we do not intervene and apply the decision of the default governors. The proposed DTPM algorithm takes around 390 μs to predict the temperature and to determine the frequency levels. The kernel functions implementing our models are called periodically whenever the CPU frequency driver is executed (by default once every 100 ms). Since the proposed algorithm does not run unless there is a thermal violation, the overhead of our approach is less than 0.39%. All the results reported in this paper are direct measurements on this platform. Hence, the implementation overheads are accounted for in the reported results.

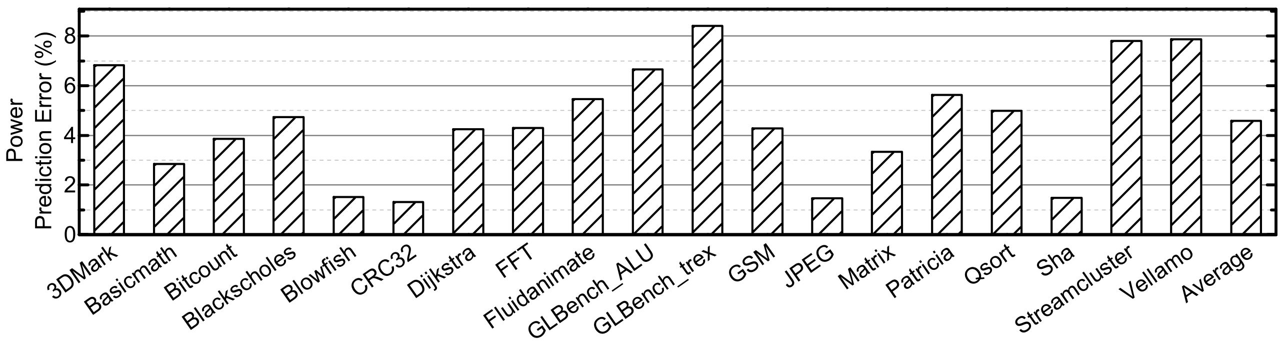


Fig. 11: Summary of power prediction error. The default on-demand governor changes the frequency dynamically during each run. Therefore, the power model is validated under dynamically varying frequency settings.

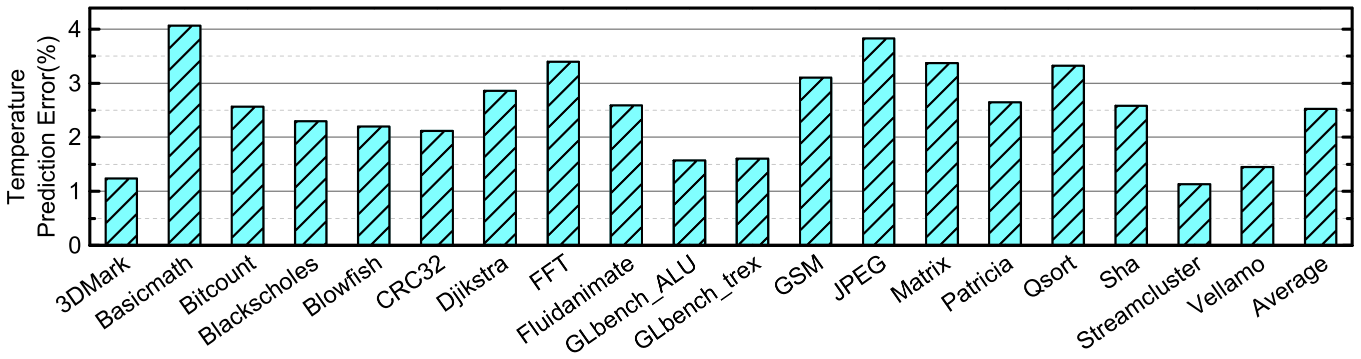


Fig. 12: Temperature prediction error for all the benchmarks.

B. Power Consumption and Temperature Prediction Accuracy

We start the experimental study by evaluating the accuracy of our power model, since it is an integral part of the proposed DTPM algorithm. The power prediction error for each benchmark is listed in Figure 11. The relative absolute error is less than 5% for most of the benchmarks, and the average error is only 4.7%. We observe a slightly higher error for several benchmarks including 3DMark, GLBench_trex, and Vellamo. Closer inspection of these benchmarks reveals that these benchmarks exhibit rapid changes in dynamic activity at certain intervals. Our model over- or undershoots during these intervals, while maintaining less than 5% relative error most of the remaining time. In particular, median relative absolute error ranges from 2.7% to 3.1% even for these benchmarks.

To evaluate the accuracy of our temperature model, we ran each benchmark and predicted the temperature $T[k+10]$ at every control interval $T[k]$. That is, the temperature one second (10 control intervals) ahead of time is predicted using a sliding window, and the predictions are compared to the measured values at the end of each experiment. Figure 12 shows that the average prediction error is less than 3% (1°C), and it never exceeds 5% (2°C). A one-second prediction window is selected since 10 control intervals are sufficient to regulate the temperature. We also note that prediction windows as large as 5 seconds do not increase the processing time, while the prediction error increases moderately, as depicted in Figure 8.

C. Temperature Control and Stability

The objective of the DTPM algorithm is to ensure that the temperature is regulated successfully without a fan. The proposed algorithm can regulate the temperature for all of the benchmarks with minimal performance impact. To illustrate the temperature control of demanding GPU and CPU-bound applications, we provide the temperature profile as a function of time for 3DMark and Fluidanimate benchmarks in Figures 13 and 14, respectively. Figure 13 clearly shows that the temperature profile of CCA and GSA are the same as the default algorithm before there is any temperature violation. When the temperature exceeds 78°C at around t =85s, the GSA algorithm starts throttling the GPU frequency. As the zoomed graph clearly illustrates, the GSA algorithm regulates the temperature successfully. In contrast, the default algorithm and CCA lead to higher temperatures, since they do not control the GPU frequency effectively. We also note that the temperature fluctuates as the 3DMark benchmark transitions from the GT1 test to GT2 test (around $t=216$ s) and GT2 test to physics test (around $t=261$ s). Since the GPU utilization reaches 100% during the heavy phases of GT1 and GT2, the GSA algorithm allows higher frequency during these times. Similarly, Figure 14 shows that the default algorithm leads to temperature violation a couple of seconds after launching the application. In contrast, the CCA and GSA algorithms successfully maintain the temperature within the specified constraint with less than 10% impact on the performance.

Effect of GPU Temperature Control: We emphasize that

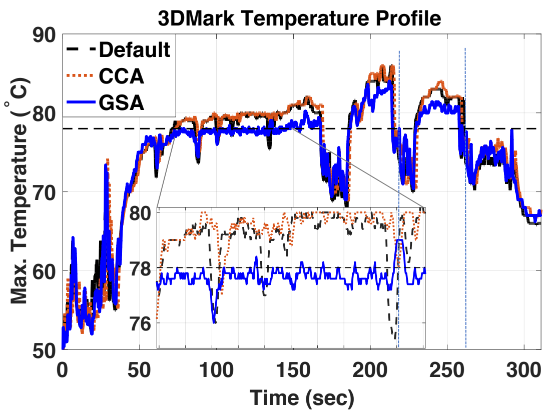


Fig. 13: Temperature control for the 3DMark benchmark.

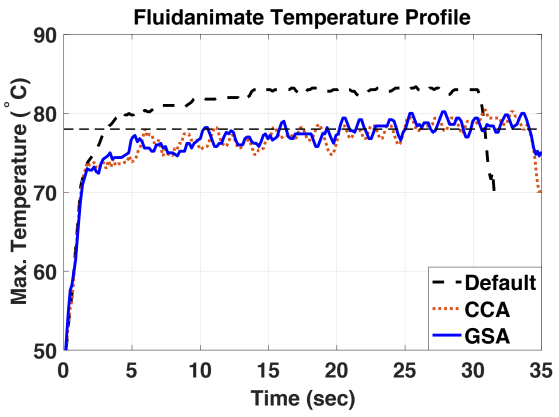


Fig. 14: Temperature profile for Fluidanimate benchmark.

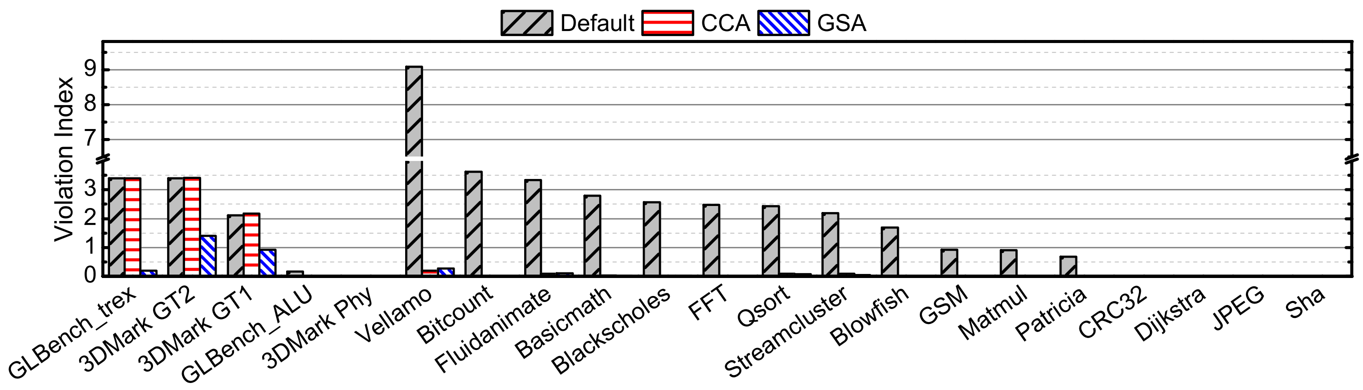


Fig. 15: Violation index for all the benchmarks.

the CCA algorithm proposed in [42] does not provide an effective control of the GPU frequency, unlike the proposed GSA algorithm. Therefore, CCA can violate the maximum chip temperature constraint, when a GPU intensive application runs. Hence, the primary advantage of GSA over CCA is the ability to handle both CPU and GPU-bound applications.

D. Performance, Power and Energy Consumption Evaluation

Temperature control, the primary objective of the proposed DTPM algorithm, is achieved by throttling the cores and turning them off when necessary. Therefore, it is imperative to study the effectiveness in preventing temperature violations and impact on performance, power consumption, and total energy. In order to obtain accurate evaluations, we performed five separate runs, and confirmed the consistency of measurements. The performance of CPU and GPU applications are measured by execution time and frames rate, respectively. During evaluations, we use a temperature constraint of 68°C for the single threaded applications, Vellamo, GLBench_ALU, and GLBench_trex. We use a higher threshold (78°C) for multi-threaded benchmarks, 3DMark and FFT, since they have very high computational requirements and tend to push the temperature higher.

We first analyze the ability of GSA to minimize the temperature violations, since this also helps in understanding the impact on performance and power consumption. To quantify the number of temperature violations, we define the set of all control intervals with a temperature violation as $\mathcal{S}_\mathcal{V}$. We need to minimize both the number of violations and the difference with the temperature constraint T_{constr}. Therefore, we define the violation index as:

$$\text{Violation Index} = \frac{\sum_{j \in \mathcal{S}_\mathcal{V}} \left(|T_j - T_{constr}|\right)}{N_v} \tag{19}$$

where N_v is the number of temperature violations observed with the default configuration. Equation 19 penalizes larger violations more heavily and normalizes the violation index with respect to the default configuration. Figure 15 shows the violation indices first for the GPU benchmarks followed by the CPU-bound benchmarks in decreasing order. We observe that the CCA algorithm performs similar to the default configuration for GPU benchmarks, since it does not control the GPU frequency effectively. In contrast, GSA decreases the temperature violations significantly. We note that both CCA and GSA are able to capture the temperature profile accurately as they use the same thermal model. GSA is able to reduce the temperature violations, since it is able to control both CPU and GPU frequencies effectively. We also observe that both GSA and CCA are very effective in avoiding temperature violations of the remaining set of benchmarks. Finally, the last four light benchmarks do not lead to noticeable amount

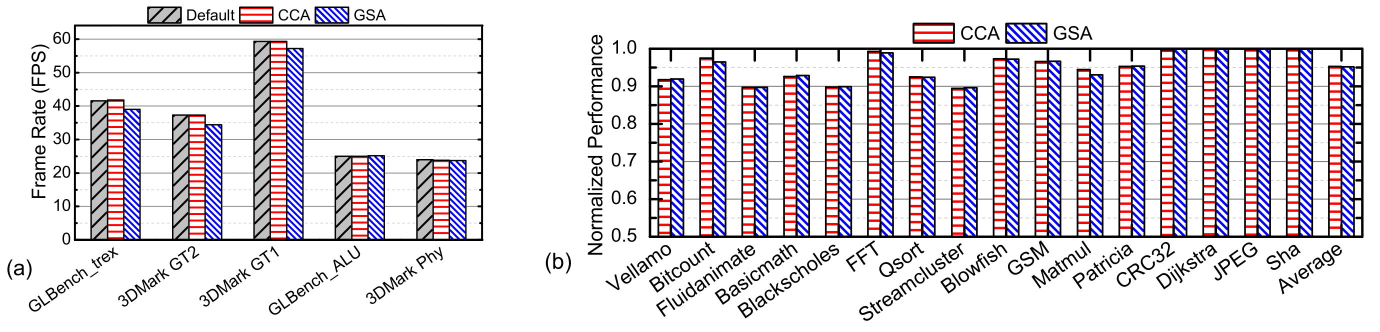


Fig. 16: (a) Frame rates for GPU benchmarks. (b) Normalized performance of CCA and GSA for CPU-heavy benchmarks. Performance is measured by the execution time for CPU benchmarks and performance score for Vellamo.

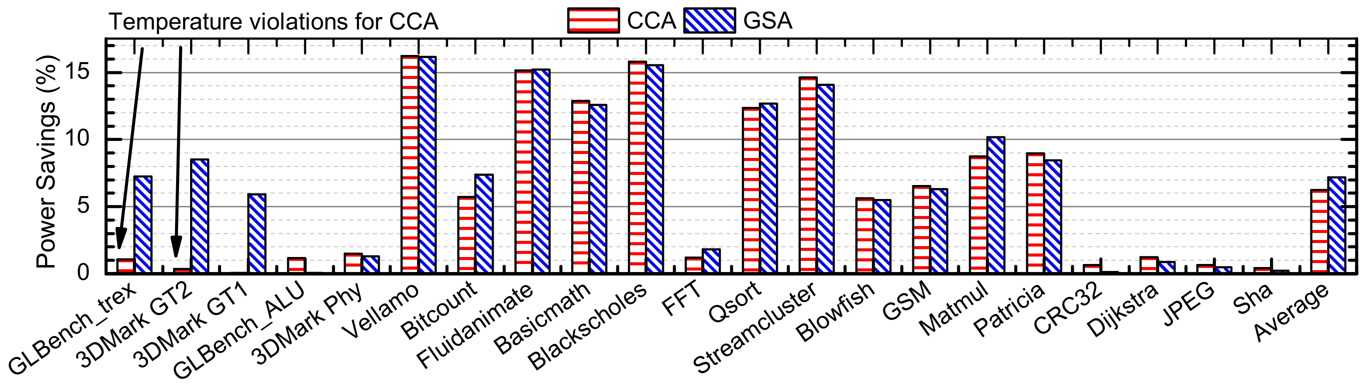


Fig. 17: Power savings achieved by CCA and GSA with respect the default configuration (these results ***do not*** include the savings due to disabling the fan).

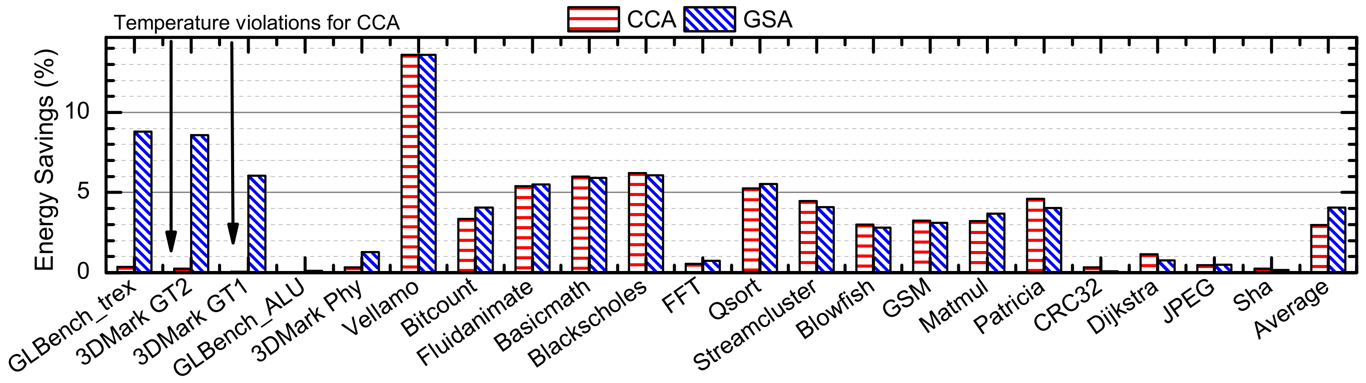


Fig. 18: Energy savings achieved by CCA and GSA with respect the default configuration (these results ***do not*** include the savings due to disabling the fan).

of violations even with the default configuration. They are included to demonstrate that the proposed algorithm does not interfere with the performance, when there are no temperature violations. Indeed, Figure 16 shows that GSA and CCA do not cause any performance penalty for these four benchmarks.

We analyze the performance impact of GPU and CPU benchmarks in Figure 16 in terms of frame rate (measured in FPS) and normalized execution time, respectively. The default configuration and CCA both achieve 41 FPS for GLBench_trex benchmark, which is expected since both have similar amounts of temperature violations. Although GSA reduces the number of temperature violations by 5×, it still achieves 39 FPS, which is hard to distinguish visually from the default configuration. Similarly, the frame rate achieved by GSA is within 3 FPS of the default configuration for the 3DMark tests, although the number of temperature violations are decreased significantly, as shown in Figure 15. We observe a performance degradation for more intensive benchmarks, such as Vellamo and Blackscholes. However, the performance impact is always less than 9%, and both the median and mean performance loss are 5%. We also observe that GSA and CCA have similar performance when there is little or no GPU activity, since the gradient search reduces to throttling the CPU frequency.

The biggest advantage of the proposed GSA algorithm is its ability to control both GPU and CPU temperature hotspots with minimal performance impact, and without using a fan. When we enable the fan in our platform, a power consumption overhead of 400-800 mW is observed. This corresponds on average to 20-30% of total (big cluster, little cluster, GPU and memory) power consumption for the benchmarks used in this work. In addition to this, throttling the performance leads to *further* reduction in the power consumption. In particular, we observe greater power and energy savings for those benchmarks that experience larger performance impact, as summarized in Figure 17 and Figure 18. For example, GSA leads to 17% less power consumption than the default configuration while running the Vellamo application. On average, GSA achieves 7% power and 4% energy savings, in addition to eliminating the fan power. *CCA produces similar results, but it is not able to stabilize the temperature for GPU heavy applications.*

VII. Conclusion

In this paper, we presented a feasible temperature prediction methodology and a DTPM algorithm for heterogeneous MP-SoCs. We demonstrated that the proposed technique predicts the temperature with less than 5% error across all benchmarks. Using the temperature predictions, the proposed algorithmic optimization approach computes a precise power budget at runtime. Then, this budget is used to throttle the frequency and number of cores with minimal impact on system performance. A thorough experimental evaluation shows that the proposed approach not only eliminates the need for a fan, which is not a viable choice for mobile devices, but also provides significant thermal and reliability advantages. In particular, the proposed DTPM algorithm successfully regulates the maximum temperature and decreases the temperature violations by one order of magnitude, while also reducing the *total* power consumption on average by 7% compared to the default solution.

Acknowledgment

This work was supported partially by Strategic CAD Labs, Intel Corporation, Semiconductor Research Corporation (SRC) task 2721.001, and National Science Foundation under Grant No. CNS-1526562. The authors would like to thank Tanvir Mustofa of Arizona State University for helping with the experiments, and Michael Kishinevsky and Raid Ayoub of Intel Corporation for feedback and guidance.

References

[1] Q. Xie, J. Kim, Y. Wang, D. Shin, N. Chang, and M. Pedram, "Dynamic Thermal Management in Mobile Devices Considering the Thermal Coupling Between Battery and Application Processor," in *Proc. of Int. Conf. on Comput.-Aided Design*, 2013, pp. 242–247.

[2] D. Kadjo, U. Y. Ogras, R. Ayoub, M. Kishinevsky, and P. Gratz, "Towards Platform Level Power Management in Mobile Systems," in *Proc. of System-on-Chip Conf.*, 2014, pp. 146–151.

[3] D. Brooks, R. P. Dick, R. Joseph, and L. Shang, "Power, Thermal, and Reliability Modeling in Nanometer-Scale Microprocessors," *IEEE Micro*, vol. 27, no. 3, pp. 49–62, 2007.

[4] L. Benini and G. DeMicheli, *Dynamic Power Management: Design Techniques and CAD Tools*. Springer Science & Business Media, 2012.

[5] T. S. Muthukaruppan, M. Pricopi, V. Venkataramani, T. Mitra, and S. Vishin, "Hierarchical Power Management for Asymmetric Multi-Core in Dark Silicon Era," in *Proc. of Design Autom. Conf.*, 2013, p. 174.

[6] R. Z. Ayoub *et al.*, "OS-Level Power Minimization Under Tight Performance Constraints in General Purpose Systems," in *Proc. of the Int. Symp. on Low-power Electron. and Design*, 2011, pp. 321–326.

[7] P. Greenhalgh, "Big.LITTLE Processing With Arm Cortex-A15 & Cortex-A7," *ARM White Paper*, 2011.

[8] Odroid Platforms. ODROID − XU + E and ODROID − XU3. http://www.hardkernel.com/main/main.php, Accessed 10/24/2016.

[9] K. Skadron *et al.*, "Temperature-Aware Microarchitecture: Modeling and Implementation," *ACM Trans. Archit. Code Optim.*, vol. 1, no. 1, pp. 94–125, 2004.

[10] O. Sahin, P. T. Varghese, and A. K. Coskun, "Just Enough is More: Achieving Sustainable Performance in Mobile Devices under Thermal Limitations," in *Proc. of the IEEE/ACM Intl. Conf. on Computer-Aided Design*, 2015, pp. 839–846.

[11] D. Brooks and M. Martonosi, "Dynamic Thermal Management for High-Performance Microprocessors," in *Int. Symp. on High-Perf. Comput. Archit.*, 2001, pp. 171–182.

[12] J. Donald and M. Martonosi, "Techniques for Multicore Thermal Management: Classification and New Exploration," in *ACM SIGARCH Computer Architecture News*, vol. 34, no. 2, 2006, pp. 78–88.

[13] C. Isci, A. Buyuktosunoglu, C.-Y. Cher, P. Bose, and M. Martonosi, "An Analysis of Efficient Multi-Core Global Power Management Policies: Maximizing Performance for a Given Power Budget," in *Proc. of Int. Symp. on Microarchit.*, 2006, pp. 347–358.

[14] S. Sharifi and T. S. Rosing, "Accurate Direct and Indirect On-Chip Temperature Sensing for Efficient Dynamic Thermal Management," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 29(10), pp. 1586–1599, 2010.

[15] W. Huang *et al.*, "HotSpot: A Compact Thermal Modeling Methodology for Early-Stage VLSI Design," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 14, no. 5, pp. 501–513, 2006.

[16] D. Shin, S. W. Chung, E.-Y. Chung, and N. Chang, "Energy-Optimal Dynamic Thermal Management: Computation and Cooling Power Co-Optimization," *IEEE Trans. Ind. Informat.*, vol. 6, no. 3, pp. 340–351, 2010.

[17] I. Yeo, C. C. Liu, and E. J. Kim, "Predictive Dynamic Thermal Management for Multicore Systems," in *Proc. of Design Autom. Conf.*, 2008, pp. 734–739.

[18] S. Sharifi, D. Krishnaswamy, and T. S. Rosing, "PROMETHEUS: A Proactive Method for Thermal Management of Heterogeneous MP-SoCs," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, pp. 1110–1123, 2013.

[19] A. K. Coskun, T. S. Rosing, and K. C. Gross, "Utilizing Predictors for Efficient Thermal Management in Multiprocessor SoCs," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 28, no. 10, pp. 1503–1516, 2009.

[20] R. Cochran and S. Reda, "Thermal Prediction and Adaptive Control Through Workload Phase Detection," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 18, no. 1, p. 7, 2013.

[21] O. Khan and S. Kundu, "Hardware/Software Co-Design Architecture for Thermal Management of Chip Multiprocessors," in *Proc. of the Conf. on Design, Autom. and Test in Europe*, 2009, pp. 952–957.

[22] R. McGowen, "Adaptive Designs for Power and Thermal Optimization," in *Proc. of Int. Conf. on Comput.-Aided Design*, 2005, pp. 118–121.

[23] T. Lee, M. Johnson, and M. Crowley, "Temperature Sensor Integral With Microprocessor and Methods of Using Same," Oct. 5 1999, US Patent 5,961,215. [Online]. Available: http://www.google.com/patents/US5961215

[24] Y. Taur and T. H. Ning, *Fundamentals of Modern VLSI Devices*. Cambridge University Press, 2009.

[25] A. P. Chandrakasan, W. J. Bowhill, and F. Fox, *Design of High-Performance Microprocessor Circuits*. Wiley-IEEE press, 2000.

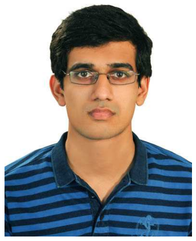
[26] M. Zapater *et al.*, "Leakage-Aware Cooling Management for Improving Server Energy Efficiency," *IEEE Trans. Parallel Distrib. Syst.*, vol. 26, no. 10, pp. 2764–2777, 2015.

[27] W. Liao, J. M. Basile, and L. He, "Leakage Power Modeling and Reduction With Data Retention," in *Proc. of Int. Conf. on Comput.-Aided Design*, 2002, pp. 714–719.

[28] W. Ye, V. Narayanan, M. Kandemir, and M. J. Irwin, "The Design and Use of Simplepower: A Cycle-Accurate Energy Estimation Tool," in *Proc. of Design Autom. Conf.*, 2000, pp. 340–345.

[29] R. David, P. Bogdan, R. Marculescu, and U. Ogras, "Dynamic Power Management of Voltage-frequency Island Partitioned Networks-on-Chip using Intel's Single-chip Cloud Computer," in *Proc. of the Int. Symp.*

on Networks on Chip*, 2011, pp. 257–258.
[30] S. Sharifi, A. K. Coskun, and T. S. Rosing, "Hybrid Dynamic Energy and Thermal Management in Heterogeneous Embedded Multiprocessor SoCs," in *Proc. of Asia and South Pacific Design Autom. Conf.*, 2010, pp. 873–878.
[31] A. K. Coskun, J. L. Ayala, D. Atienza, T. S. Rosing, and Y. Leblebici, "Dynamic Thermal Management in 3D Multicore Architectures," in *Proc. of the Conf. on Design, Autom. and Test in Europe*, 2009, pp. 1410–1415.
[32] F. Hameed, M. Faruque, and J. Henkel, "Dynamic Thermal Management in 3D Multi-Core Architecture Through Run-Time Adaptation," in *Proc. of the Conf. on Design, Autom. and Test in Europe*, 2011, pp. 1–6.
[33] D.-C. Juan, S. Garg, and D. Marculescu, "Statistical Thermal Evaluation and Mitigation Techniques for 3D Chip-Multiprocessors in the Presence of Process Variations," in *Proc. of the Conf. on Design, Autom. and Test in Europe*, 2011, pp. 1–6.
[34] P. Bogdan, R. Marculescu, S. Jain, and R. T. Gavila, "An Optimal Control Approach to Power Management for Multi-Voltage and Frequency Islands Multiprocessor Platforms under Highly Variable Workloads," in *Proc. of the Int. Symp. on Networks on Chip*, 2012, pp. 35–42.
[35] Z. Wang, S. Ranka, and P. Mishra, "Efficient Task Partitioning and Scheduling for Thermal Management in Multicore Processors," in *Proc. of Int. Symp. on Quality Electronic Design*, 2015.
[36] H. Wang *et al.*, "Hierarchical Dynamic Thermal Management Method for High-Performance Many-Core Microprocessors," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 22, no. 1, pp. 1:1–1:21, 2016.
[37] W.-M. Chen, S.-W. Cheng, P.-C. Hsiu, and T.-W. Kuo, "A User-Centric CPU-GPU Governing Framework for 3D Games on Mobile Devices," in *Proc. of Int. Conf. on Comput.-Aided Design*, 2015, pp. 224–231.
[38] U. Gupta *et al.*, "Dynamic Power Budgeting for Mobile Systems Running Graphics Workloads," *IEEE Trans. Multi-Scale Comput. Syst.*, 2017.
[39] A. Prakash, H. Amrouch, M. Shafique, T. Mitra, and J. Henkel, "Improving Mobile Gaming Performance Through Cooperative CPU-GPU Thermal Management," in *Proc. of Design Autom. Conf.*, 2016, p. 47.
[40] V. Hanumaiah, S. Vrudhula, and K. S. Chatha, "Performance Optimal Online DVFS and Task Migration Techniques for Thermally Constrained Multi-Core Processors," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 30, no. 11, pp. 1677–1690, 2011.
[41] V. Pallipadi and A. Starikovskiy, "The Ondemand Governor," in *Proc. of the Linux Symp.*, vol. 2, 2006, pp. 215–230.
[42] G. Singla, G. Kaur, A. K. Unver, and U. Y. Ogras, "Predictive Dynamic Thermal and Power Management for Heterogeneous Mobile Platforms," in *Proc. of the Design, Automa. & Test in Europe Conf. & Exhibition*, 2015, pp. 960–965.
[43] H. Sultan, G. Ananthanarayanan, and S. R. Sarangi, "Processor Power Estimation Techniques: A Survey," *Int. J. High Perf. Syst. Archit.*, vol. 5, no. 2, pp. 93–114, 2014.
[44] Y. Liu, R. P. Dick, L. Shang, and H. Yang, "Accurate Temperature-Dependent Integrated Circuit Leakage Power Estimation is Easy," in *Proc. of the Conf. on Design, Autom. and Test in Europe*, 2007, pp. 1526–1531.
[45] K. Roy, S. Mukhopadhyay, and H. Mahmoodi-Meimand, "Leakage Current Mechanisms and Leakage Reduction Techniques in Deep-Submicrometer CMOS Circuits," *Proc. IEEE*, vol. 91, no. 2, pp. 305–327, 2003.
[46] 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, 2017.
[47] U. Gupta, S. Korrapati, N. Matturu, and U. Y. Ogras, "A Generic Energy Optimization Framework for Heterogeneous Platforms Using Scaling Models," *Microprocessors and Microsystems*, vol. 40, pp. 74–87, 2016.
[48] J. Zhuo and C. Chakrabarti, "Energy-Efficient Dynamic Task Scheduling Algorithms for DVS Systems," *ACM Trans. Embedd. Comput. Syst.*, vol. 7, no. 2, p. 17, 2008.
[49] G. T. Gilbert, "Positive Definite Matrices and Sylvester's Criterion," *The American Mathematical Monthly*, vol. 98, no. 1, pp. 44–46, 1991.
[50] S. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge Univ. Press, 2004.
[51] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, and R. B. Brown, "Mibench: A Free, Commercially Representative Embedded Benchmark Suite," in *Proc. of Int. Symp. on Workload Characterization*, 2001, pp. 3–14.
[52] C. Bienia, S. Kumar, J. P. Singh, and K. Li, "The PARSEC Benchmark Suite: Characterization and Architectural Implications," in *Proc. of Int. Conf. on Parallel Archit. and Compilation Techniques*, 2008, pp. 72–81.
[53] Hardkernel. ODROID-XU3 Linux Kernel Source Tree. https://github.com/hardkernel/linux/tree/odroidxu3-3.10.y-android. Accessed 02/19/2017.



Ganapati Bhat received his B.Tech degree in Electronics and Communication from Indian School of Mines, Dhanbad, India in 2012. From 2012-2014 he worked as a software engineer at Samsung Research and Development Institute, Bangalore, India. He is currently a PhD student in Computer Engineering at the department of Electrical, Computer and Energy engineering, Arizona State University. His research interests include energy optimization in computing systems, dynamic thermal and power management, energy management for wearable systems, and hardware architectures for emerging communication protocols.

Gaurav Singla received his B.E degree in Electronics and Telecommunication from Maharashtra Institute of Technology, Pune, India in 2011. From 2011-2013 he worked as a design engineer at Tata Technologies, Pune, India. He completed his Masters degree in Electrical engineering from Arizona State University(2013-2015). He is currently working as a Graduate Design Engineer and is a part of the memory design team at ARM. His research interests include memory design techniques and dynamic thermal and power management for multicore architectures.

Ali Unver is a Senior Test R & D Engineer at Intel Corporation in Chandler, AZ. He has a book published in 2009, called Observation Based PDE Models for Stochastic Production Systems. He received an M.S. degree in Electrical Engineering in 2003, and a PhD degree in Applied Mathematics in 2008 both from Arizona State University. He also worked as a postdoctoral researcher at UCLA Institute for Pure and Applied Mathematics in 2009. Dr. Unver is married with three children.

Umit Y. Ogras received his Ph.D. degree in Electrical and Computer Engineering from Carnegie Mellon University, Pittsburgh, PA, in 2007. From 2008 to 2013, he worked as a Research Scientist at the Strategic CAD Laboratories, Intel Corporation. He is currently an Assistant Professor at the School of Electrical, Computer and Energy Engineering. Recognitions Dr. Ogras has received include Strategic CAD Labs Research Award, 2012 IEEE Donald O. Pederson Transactions on CAD Best Paper Award, 2011 IEEE VLSI Transactions Best Paper Award and 2008 EDAA Outstanding PhD. Dissertation Award. His research interests include digital system design, embedded systems, multicore architecture and electronic design automation with particular emphasis on multiprocessor systems-on-chip (MPSoC).