Predictive Dynamic Thermal and Power Management for Heterogeneous Mobile Platforms

by

Gaurav Singla

A Thesis Presented in Partial Fulfillment
of the Requirement for the Degree
Master of Science

Approved April 2015 by the
Graduate Supervisory Committee:

Umit Y. Ogras, Chair
Bertan Bakkaloglu
Ali Unver

ARIZONA STATE UNIVERSITY

May 2015

ABSTRACT

Heterogeneous multiprocessor systems-on-chip (MPSoCs) powering mobile platforms integrate multiple asymmetric CPU cores, a GPU, and many specialized processors. When the MPSoC operates close to its peak performance, power dissipation easily increases the temperature, hence adversely impacts reliability. 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 abstract presents a DTPM algorithm based on a practical temperature prediction methodology using system identification. The DTPM algorithm dynamically computes a power budget using the predicted temperature, and controls the types and number of active processors as well as their frequencies. Experiments on an octa-core big.LITTLE processor and common Android apps demonstrate that the proposed technique predicts temperature within 3% accuracy, while the DTPM algorithm provides around 6$\times$ reduction in temperature variance, and as large as 16% reduction in *total platform power* compared to using a fan.

Dedicated to my family

ACKNOWLEDGEMENTS

Firstly, I would like to express my deepest gratitude and thank my advisor, Dr. Umit Ogras, who not only taught and motivated me to pursue research, but also helped me achieve certain level of confidence and maturity. Without his valuable time, support and guidance, I could not have finished this work. I would also like to thank Dr. Bertan Bakkaloglu and Dr. Ali Unver for taking out time and agreeing to be a part of my thesis defense committee.

I would like to thank Gurinderjit, Navyasree and Spurthi for helping conduct the experiments on Odroid platform which accelerated collection of the measurement data. Other students from eLAB: Ujjwal Gupta, Sankalp Jain and Harshad Navale's comments and suggestions not only accelerated but also improved the quality of my work. I would like to mention people from Intel corporation, Raid Ayoub and Mike for their financial and technical help throughout this work. Special thanks to David Kadjo from TAMU, for helping me at the software level and during the board bring-up stage.

I would like to pay respect and express love to my parents Rattan and Rajrani for their encouragement, sacrifices and endless support. My girlfriend Manbir and my friends Swanand and Nikita were a constant source of valuable feedback and always provided detailed help.

My master's term at Arizona State University was productive as well as enjoyable. Thanks to all the members of eLAB for this wonderful journey. Special thanks to Dr. Umit Ogras and Intel Corporation for financially assisting me during my master's study as well as for the conference trip. Finally,I would also like to thank Dr. Sule Ozev and Dr. Yu Cao for providing financial support by offering me teaching assistant position.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Figure Page

Chapter 1

INTRODUCTION

The abundance of logic and interconnect resources that can be integrated on a single chip pushes the limits of MPSoCs, which power the vast majority of mobile devices. Meanwhile, MPSoC design is driven by the persistent demand for faster and more powerful devices. On one hand, the number and capacity of the CPU cores increase at a steady rate. On the other hand, the degree of heterogeneity is growing with the inclusion of asymmetric cores and accelerators such as, GPU, video codecs, digital signal processors and display processing engine. The boost in computational power inevitably increases the power dissipation, which in turn reduces the battery lifetime and raises the chip temperature.

Recent results reveal that the skin temperature, hence the power consumption, is the performance limiter in mobile devices [23, 46]. Power and temperature have become the major constraints for throughput improvement of homogeneous as well as heterogeneous architectures. Furthermore, rapid changes in power and temperature also deteriorate reliability [4, 25]. Thermal regulation and stability are also as important as thermal and power control for a reliable system. Mean time to failure is also severely affected due to the thermal stress [37]. Effective control algorithms are need of the hour in order to maximize the performance while controlling both the power consumption and temperature of a MPSoC.

Competing requirements between performance and power consumption are addressed by a variety of design and run-time approaches that aim at maximizing performance during busy periods and minimizing power 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 [35, 36]. Newly emerged big.LITTLE processing

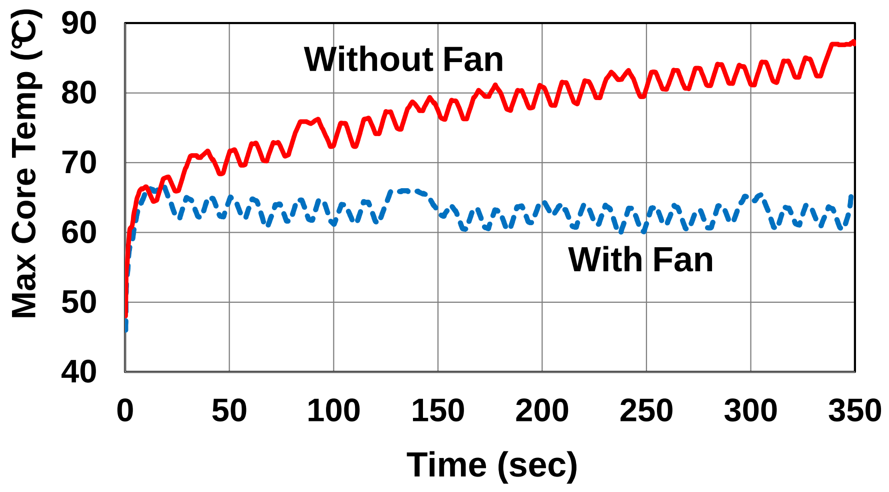

Figure 1.1: Maximum core temperature with and without the fan

works in tandem with these techniques by combining high performance (big) and energy efficient (little) clusters [14]. Big cores are utilized when high performance is needed, while little cores are used during low activity periods. A recent instance of this architecture is the Samsung Exynos 5410 chip, which hosts four A15 (big) and four A7 (little) cores as shown in Figure 1.2. Apart from the CPU, it comprises of GPU, memory and accelerators for video and jpeg. This processor is commercially being used in handheld devices.

Our measurements on this chip show 10× dynamic range in performance and 30× range in power consumption between the highest performance and lowest power configurations. Highest performance being the four big CPU cluster cores operating at highest possible frequency, while lowest power when 1 little CPU cluster core is operated at lowest possible frequency. Furthermore, moderate to high activity workloads demand big cores and raise the temperature easily beyond acceptable levels. The experimental platform we use, employs a fan to address this problem [1]. Figure 1.1 shows the temperature behavior of the hot spots in presence and absence of fan. Using a fan might lower and control the temperature, however fan is *not a viable option* for mobile platforms such as smartphones, where heterogeneous MPSoCs have widespread use. Technology scaling and emerging

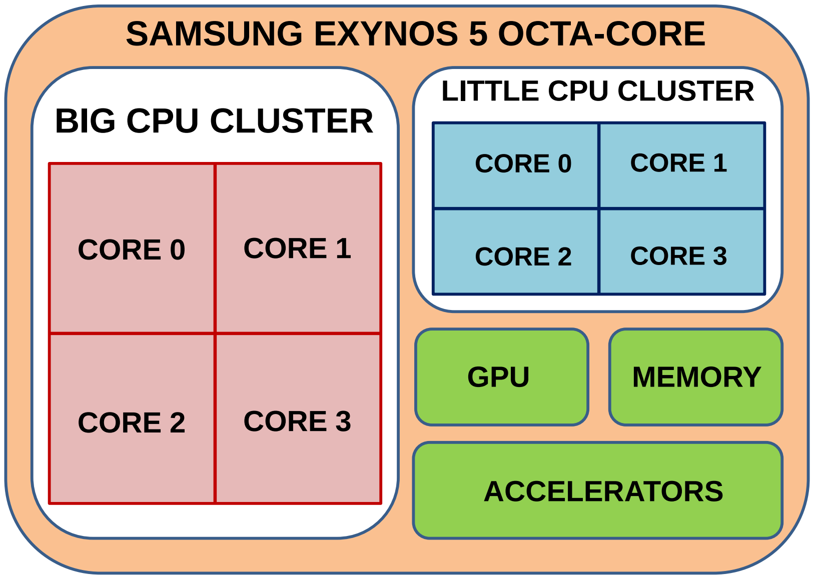

Figure 1.2: big-LITTLE architecture of Samsung Exynos 5 octa-core processor

techniques have reduced the thickness of mobile phones to such an extent that fan is not only difficult but almost impossible to be employed for cooling. Hence, there is a strong need for DTPM approaches for big.LITTLE architectures to effectively regulate temperature with minimal performance impact.

The major degrees of freedom offered by the big.LITTLE architectures are controlling the type of CPU cluster (big or little), number of active cores, operating frequency (hence voltage) of the cores, frequency of GPU, and set 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, we focus on rest of the knobs. Constraining the maximum frequency to limit the temperature, while passively waiting for thermal violations and reacting by throttling the cores, impairs performance as well as reliability by causing large temperature variations [43]. In contrast, predictive approaches can take advantage of rich set of dynamic configuration capabilities to manage temperature effectively [39].

In this thesis, we first present a broadly applicable methodology for generating power and thermal models for heterogeneous mobile platforms. This methodology starts from the first principles and generates mathematical models that enable *accurate power/thermal predictions tailored* to the mobile platform of interest. After empirically validating these models, we present a novel run-time technique to periodically compute the power budget that is *guaranteed* to keep the temperature within permissible limits. Finally, this power budget is used for determining the CPU cluster, number of active cores, and their frequencies to regulate temperature with minimal performance impact. The major contributions of this thesis are as follows:

- A methodology for generating power and thermal models for heterogeneous MP-SoCs, and experimental validation using one of the first commercial big.LITTLE architectures [1],
- A novel approach for dynamically computing run time dynamic power budget using temperature prediction, and implementing an effective DTPM algorithm based on this approach,
- Exhaustive experimental evaluation which demonstrates effective thermal regulation with $6\times$ smaller variance and as much as 16% reduction in *total platform power* across multiple benchmarks.

The rest of the thesis is organized as follows. Related work is presented in Chapter 2. Overview of the DTPM technique is explained in Chapter 3. Power and thermal model generation methodology and corresponding empirical validation appear in Chapter 4. Thermal prediction and DTPM algorithm based on thermal prediction are presented in Chapter 5. Finally, extensive experimental evaluation using Samsung Exynos 5410 octa-core chip and a wide range of benchmarks is presented in Chapter 6, while conclusions appear in Chapter 7.

Chapter 2

RELATED WORK

Thermal modeling and dynamic thermal management have received significant attention due to increased power densities and reliability implications of temperature. Before modern thermal management techniques, hardware approaches using liquid cooling and fans, where area and cost was not a limiting factor were utilized [13, 27].

One of the first works on dynamic thermal management is [5], where the authors explore the impact on performance due to different thermal management mechanisms. Researchers started with reactive approaches where frequency of each core is controlled once the safe temperature threshold has been surpassed [42]. This technique does not consider the temperature of the neighboring resources and fails to control the temperature gradients and hotspots. In particular, poor performance of reactive approaches led researchers to develop compact thermal models [20, 40, 43] and thermal prediction techniques [11, 39, 49]. In [24], authors consider future temperature as linear extrapolation of its previous values. Such techniques help in predicting future sample temperature values, which can then be used in pro-active thermal management methods. The thermal model presented in this thesis 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. Increasing the usage of temperature sensors and power meters [26, 32] make our approach feasible and accurate.

As technology is scaling, reduction in threshold voltage, channel length, and gate oxide thickness increases the leakage power component as explained in [7, 44]. Leakage power if not dominant has now become almost equal to the dynamic power in some cases like the Intel Pentium IV processors [15]. Most of the previous work addresses power and temperature separately, while we take into consideration their inter-dependencies using the power and thermal models. [51] considers leakage power and temperature dependence, using which the authors control the fan speed for cooling of data center servers, while we deal with mobile platforms where fan is not an option. Certain power simulators assume a constant ratio between leakage and dynamic power [28, 48]. This assumption is not accurate since dynamic and leakage power's dependence on frequency, supply voltage and temperature is different. We demonstrate in Section 4.1 that leakage power is sensitive to temperature while dynamic power is independent of temperature.

Thermal models are commonly employed for temperature control by voltage/frequency assignment and task scheduling/migration. For example, the work presented in [18, 21, 34] presents temperature control techniques for homogeneous multi-core systems through DVFS. Similarly, temperature aware task assignment and scheduling techniques are presented in [9, 19]. Basically, tasks are scheduled to the resources having low utilization and temperature. To enhance thermal control and management further, [8, 12] present task migration approach, where the tasks, which have been already scheduled to a particular resource are migrated to other resources in case of thermal violations. Researchers have even tried to implement different policies in conjunction with each other as proposed in [33]. Model predictive and optimal control theory have been recently employed for thermal management to achieve smooth control with minimal performance loss [45, 50]. In these papers, authors input the workload requirement for each core and then regulate frequencies to meet these requirements while satisfying the thermal constraints, but they use feedback control methods to obtain the frequency values instead of a heuristic approach. We implement our approach using DVFS and core control since the kernel of modern platforms already considers scheduling and migration techniques such as load balancer. Our thermal predictor and power models can be utilized by the above mentioned techniques.

Most of the above mentioned work is done for homogeneous architectures, where all the cores possess similar architecture, power consumption and performance abilities. Researchers in recent past started designing algorithms for more complex architectures. [10, 17, 22] takes into consideration 3d multi-core architectures. Due to scalability problems of centralized control, an agent-based thermal management technique is proposed in [2]. Finally, a hierarchical power management technique for asymmetric processors is presented in [35], where the authors try to optimize the energy/performance trade-off under thermal design power constraints. Authors in [41] present a simulator Qsilver for thermal management in GPU architectures. Heterogeneous processors increase thermal management complexity, as multiple resources are to be taken into consideration and the temperature of all resources depend on each other.

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. Our algorithm targets heterogeneous platforms but also can 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 [1]. The used platform offers new capabilities such as big/little clusters, and detailed power and temperature sensors. The platform employs a processor which is already available in mobile phones and tablets commercially. Since developing simulation models for new processors is obstructed by the difficulties in finding exact floorplan, heat sink information and parameter values, researchers are usually limited to few examples such as simple XScale core and Alpha processor [19, 39]. We plan to make our power and thermal models public to enable research on emerging heterogeneous platforms.

Chapter 3

OVERVIEW OF THE PROPOSED FRAMEWORK

State of the art mobile platforms are highly integrated closed systems where different hardware and software modules interact very tightly. Therefore, techniques targeting these platforms cannot be designed in isolation. Hence, all the models and algorithms presented in this work are incorporated with the existing software infrastructure, as outlined in Figure 3.1. Existing frequency and idle state governors, as well as the device specific drivers, *e.g.*, GPU driver, remain intact and feed their outputs to the proposed framework. For example, the ondemand governor [36] runs the default configuration and determines the operating frequency of each core. The governor activates at a specific period, checks the device utilizations, and makes changes to the configuration. Each component of the heterogeneous processor runs its own governor. Different governor or device specific optimizations implemented in dedicated drivers can work in coordination with the proposed framework.

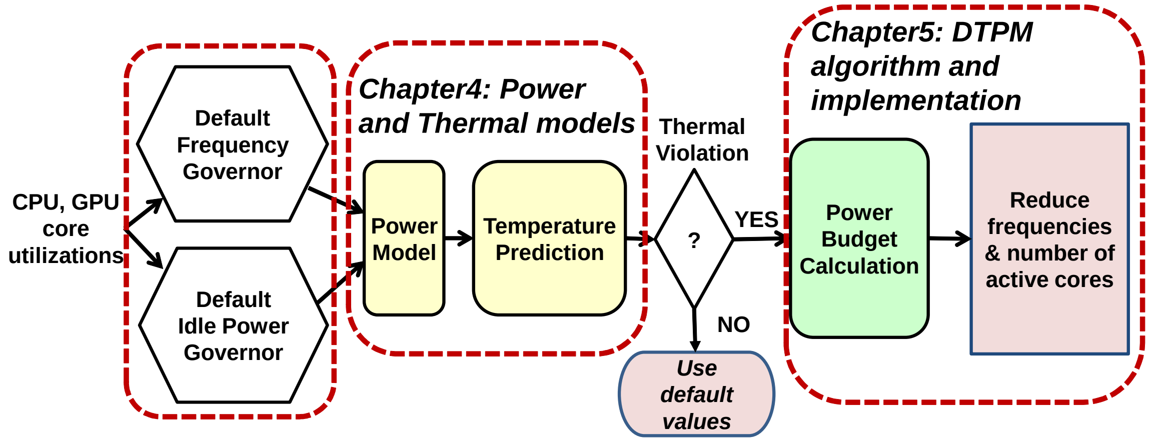
CPU, GPU core utilizations
Default Frequency Governor
Default Idle Power Governor
Chapter4: Power and Thermal models
Power Model
Temperature Prediction
Thermal Violation
?
YES
NO
Use default values
Chapter5: DTPM algorithm and implementation
Power Budget Calculation
Reduce frequencies & number of active cores

Figure 3.1: High level description of the DTPM algorithm. Sections detailing individual blocks are annotated

The default configuration of the processor runs ondemand or interactive as the default governor. The platform uses Linux version 3.4.76. The entire algorithm along with the models are implemented in the kernel. The operating system that has been used in Android 4.4.2 which perfectly emulates a mobile platform. Other operating system such as Ubuntu can also be ported to the system.

First we start with the power and thermal models which use the thermal and power sensor data as inputs. These are explained in detail in Chapter 4. We have implemented these models inside the Linux kernel, such that we can keep track of the power consumption and temperature values. The proposed power model uses the choice made by the default configuration to predict the power consumption before taking any action. The sensors provide the total power value which is then divided into leakage and dynamic components by the power model. Using this model, power values for a particular configuration is predicted. The power consumption predictions are then fed to the thermal model to predict the resulting temperature if these actions were taken. Thermal prediction is an important part of the framework, as it is the basis of our proactive approach.

Unless a thermal violation is predicted, the decisions made the default drivers such as the core and GPU frequencies, choice of big or little cluster and number of active cores, are affirmed. Thus, the proposed DTPM approach is non-intrusive when the temperature is within permissible levels. However, when a temperature violation is predicted, the proposed framework enters the proposed algorithm as is explained in Chapter 5. The first part of the algorithm is run time power budget computation. Power budget is the maximum value of power that can be consumed without violating thermal constraints. In order words, we start with the temperature constraint and work backwards to determine the maximum power consumption that can be tolerated. Then comes the second part of the algorithm which is final assignment of the configuration according to the calculated power budget. Here the power model is utilized to predict the configuration which will adhere to and satisfy the budget. The available budget is used to overwrite the set of active resources and their frequencies such that the temperature constraint violation can be prevented. These steps are detailed in the following sections as annotated in Figure 3.1.

Chapter 4

POWER AND THERMAL MODELING METHODOLOGY

Effective management of power and temperature depends critically on accurate analytical models that can be evaluated at run-time. Therefore, we start with presenting our modeling methodology that leverages thermal and power sensors [32]. In particular, power consumption of big CPU cluster P_{A15}, little CPU cluster P_{A7}, GPU P_{GPU} and memory P_{mem} are read using power sensors. If the power consumption of a target resource cannot be measured individually, it needs to be considered as a part of a bigger block whose power consumption can be measured. Likewise, temperature of each big core is read through temperature sensors. If a thermal hotspot of interest does not have a sensor, that point needs to be modeled as an unobservable node [40].

4.1 Power Modeling

The total power consumption can be expressed using the dynamic and leakage power as :

$$\begin{aligned} P_{total} &= P_{dynamic} + P_{leakage} \\ P_{total} &= \alpha C V_{dd}^2 f + V_{dd} I_{leakage} \qquad (4.1) \end{aligned}$$

where α and C are the activity factor and switching capacitance, respectively. Power models for major components of a mobile phone such as CPU, GPU, display, WiFi and battery exists in the literature [4, 6, 52]. Parameters such as supply voltage, operating frequency can be obtained by measurements or from the kernel source code. Therefore, we detail only our empirical approach to extract the leakage current and switching capacitance.

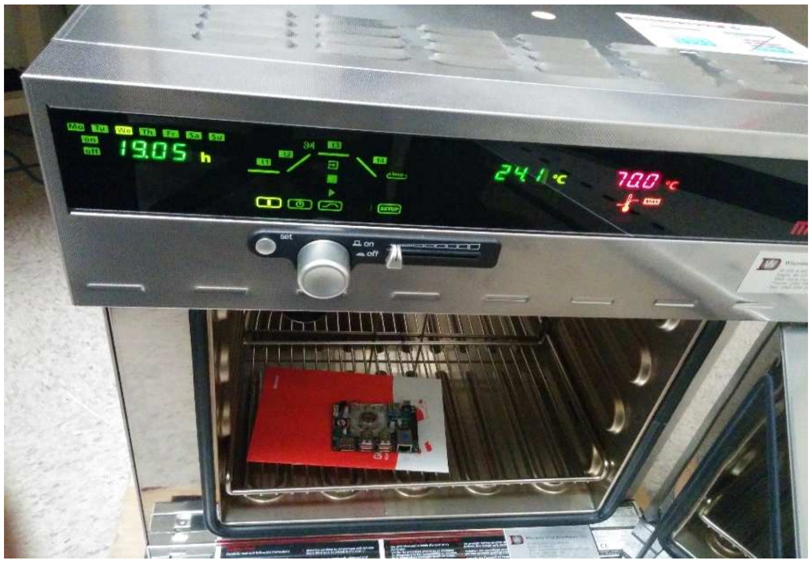

Figure 4.1: Temperature furnace used to model leakage power

4.1.1 Leakage Power Modeling

Temperature and leakage power have an internal loop with each other. They are interdependent, and increase in one causes the other also to increase. Most of the work done earlier does not consider this relationship while implementing power and thermal models. Leakage power varies exponentially with temperature. To model this behavior, we use the method described below:

Leakage Power Characterization: To model the dependence of leakage power on temperature, we used a temperature furnace as shown in Figure 4.1. The furnace helps in providing a constant ambient temperature value. We placed the target platform inside the furnace and swept the temperature from $40°C$ to $80°C$ in increments of $10°C$. During the tests, we used a light workload running only on the big cores with fixed f and V_{dd} such that the dynamic power did not increase the temperature. Light workload consumes low dynamic power and helps in maintaining the temperature at a constant value. Then, multiple power measurements were taken and this procedure was repeated for each power resource

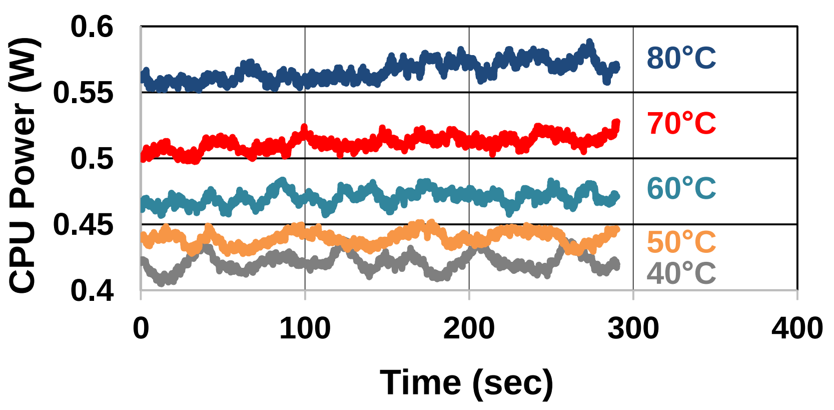

Figure 4.2: Total CPU power measurement data from the furnace

of the heterogeneous processor. The total power measured for different temperature values is shown in Figure 4.2. It is clear that since we maintained the dynamic power component constant, the increase in total power with temperature is due to the leakage component. In general, this analysis and procedure can be implemented for any platform to extract the corresponding leakage power model.

$$\begin{aligned} I_{leakage} &= A_s \frac{W}{L}(\frac{kT}{q}^2)e^{\frac{q(V_{GS}-V_{th})}{nkT}} + I_{gate} \\ I_{leakage} &= c_1 T^2 e^{\frac{c_2}{T}} + I_{gate} \end{aligned} \tag{4.2}$$

Equation 4.2 represents the leakage power equation, where A_s is a technology dependent constant, L and W are channel length and width, k is the Boltzmann constant, T is the temperature, q is the charge, V_{GS} is the gate to source voltage, V_{th} is the threshold voltage, n is the sub-threshold swing coefficient, and I_{gate} is the gate leakage current [29, 38]. These technology and parameters are then condensed into parameters denoted by c_1 and c_2. We employ non-linear fitting tool to find the unknown parameters c_1, c_2 and I_{gate} assuming that dynamic power shows negligible variation with temperature. Once values of these unknowns are obtained, we can model leakage power as a function of temperature as shown in Figure 4.3. It can be seen how leakage power varies exponentially with temperature.

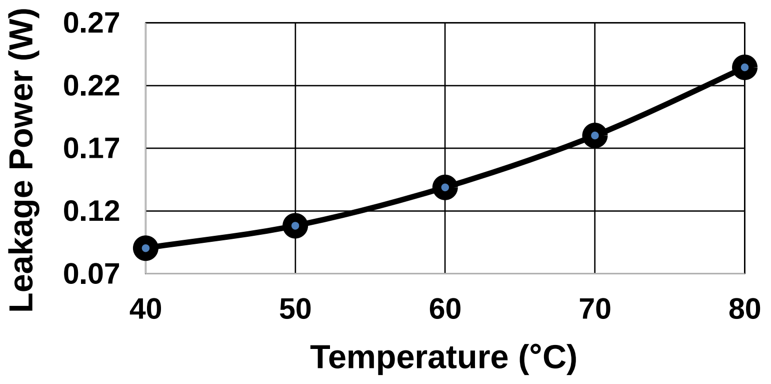

Figure 4.3: Leakage power variation with temperature

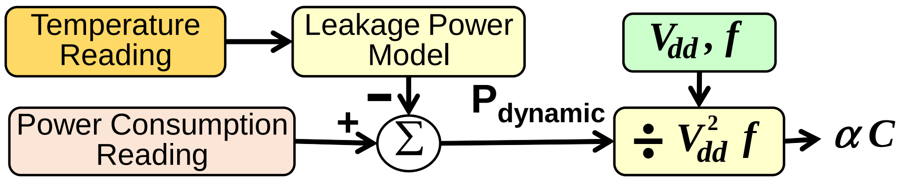

Figure 4.4: Run-time computations for computing the product of the activity factor and switching capacitance

4.1.2 Dynamic Power Modeling

Run-time computation of αC: At run-time, power/thermal sensors in the platform are used to measure the power consumption and temperature of the resource of interest. Then, the dynamic power consumption is found by subtracting the leakage power from the total power, as described in Figure 4.4. Finally, operating frequency and voltage at the time of the computation are used to extract the product of the activity factor and switching capacitance. This computation is continuously updated and an accurate reflection of activity factor is obtained at run-time. Then, this model is used to predict the dynamic power consumption before any decision on the frequency is made.

4.1.3 Power Model Validation

Figure 4.5 demonstrates the variation of leakage and dynamic power with temperature. As expected the dynamic power remains constant while the leakage power varies exponentially with temperature. The frequency used in this experiment is 1.6Ghz.

Figure 4.6 shows the variation of leakage and dynamic power with respect to frequency. As expected the dynamic power increases with frequency. Leakage power is product of the supply voltage and the leakage current. Leakage current does not vary with frequency but supply voltage does. Due to this relationship, there is a slight increase in leakage power with frequency. For this experiment, we tried to maintain the temperature constant.

Now we combine the dynamic and leakage power models to obtain the total predicted power. We compare the measured and the predicted power values in Figure 4.7.

4.2 Thermal Modeling

Due to several disadvantages of reactive approach, we have developed predictive thermal model. This enables us to predict the future temperature by taking into consideration temperature and power consumption values of all the neighboring components as well. Using the duality between the thermal and electrical networks, one can model the dynamics of the temperature using a state-space model [39, 47]. Suppose that there are N nodes

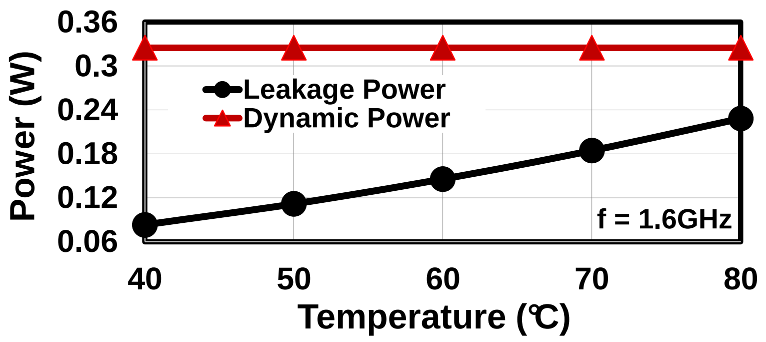

Figure 4.5: Leakage and Dynamic power variation with temperature

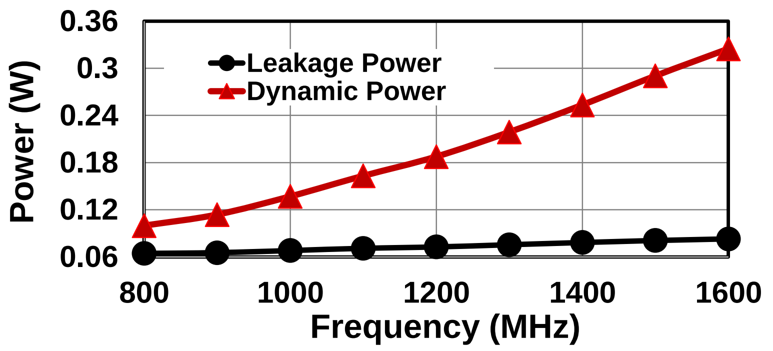

Figure 4.6: Leakage and Dynamic power variation with frequency

in the network, whose temperature and power consumption are given by $[T(t)]_{N\times1}$ and $[P(t)]_{N\times1}$, respectively. Then, $[T(t)]_{N\times1}$ can be expressed by the following differential equation [43]:

$$C_t\frac{dT}{dt} = -G_tT(t) + P(t) \tag{4.3}$$

where C_t and G_t are the thermal capacitance and conductance matrices. Since power/temperature measurement and control are performed periodically in OS kernels or firmware in practice, we discretize Equation 4.3 assuming a sampling period of T_s seconds:

$$\begin{aligned} T[k+1] &= (I - T_sC_t^{-1}G_t)T[k] + T_sC_t^{-1}P[k] \\ T[k+1] &= A_sT[k] + B_sP[k] \end{aligned} \tag{4.4}$$

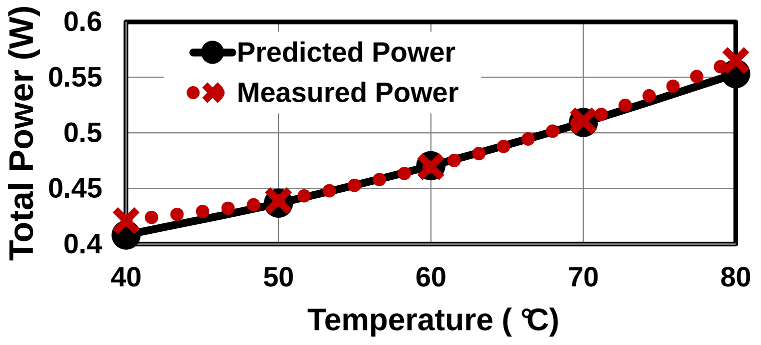

Figure 4.7: Power model validation

For our system, A is a 4x4 matrix which resembles the dependence of future core temperature on its previous value as well as on the neighboring cores while B is a 4x4 matrix which denotes the future core temperature dependence on the power resources in a particular platform. T matrix is comprised of 4 cores of the big cluster because those are the thermal hotspots and P matrix comprises of powers of the big CPU cluster, little CPU cluster, GPU and memory.

Finding the thermal conductance and capacitance matrices (A_s and B_s) using finite element simulations or a thermal modeling framework like Hotspot [43] would require detailed design information such as floorplan, heat sink geometry, and material properties, which are either not public or very hard to obtain. Furthermore, validating the thermal model would still require actual power and temperature measurements. Therefore, we start directly with actual measurements, and employ system identification to find C_t and G_t, as detailed next.

4.2.1 System Identification

The input to the difference equation is $P(k)$, which is the power consumption of the major resources. For instance, $P = [P_{A7}, P_{A15}, P_{GPU}, P_{mem}]^T$ for our system, where P_{A7} and P_{A15} correspond to the little and big core clusters, respectively. P_{GPU} corresponds to the GPU power consumption, whereas P_{mem} corresponds to the memory power. Thus even if the thermal hotspots mainly comprises of the big CPU cluster cores, its temperature dependence on other power resources is also taken into consideration. In order to obtain an accurate characterization, we controlled *each of these four sources separately* while keeping the others constant or at a minimum value. More precisely, we oscillated the frequency of big cores between the minimum and maximum values using a pseudo-random bit sequence (PRBS), and measured the temperature. The PRBS input is generated to cover a frequency spectrum, which is much broader than that excited by an arbitrary application.

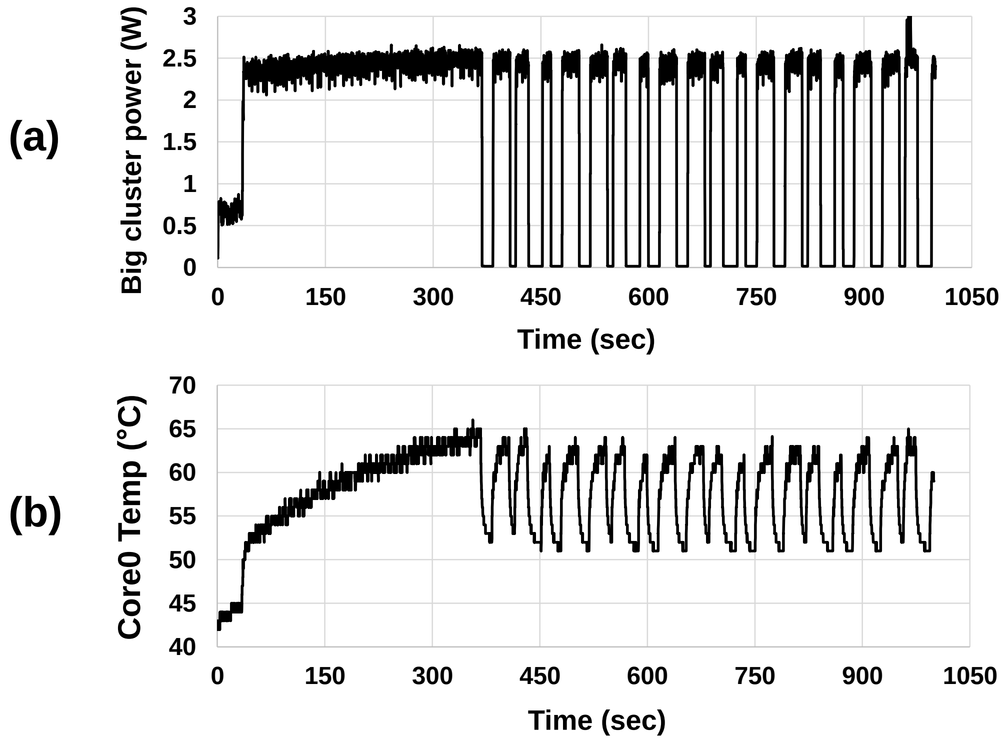

Figure 4.8: PRBS test signal for big cluster (a) Big cluster power (b) Core 0 temperature

The resulting spectrum is also large enough to capture temperature variations since the OS drivers that sample the temperature and implement the control algorithms are typically invoked around every $100\ ms$ to avoid computational overhead. Then, we recorded the input $P[k]$ and output $T[k]$ time series. Figure 4.8(a) shows the PRBS power test signal for the big cluster. The other power resources were minimum or constant at this point. Figure 4.8(b) demonstrates the corresponding temperature variation due to variation in power. Similar signals can be used for little cluster, GPU and memory.

Finally, we used the system identification toolbox of Matlab [30, 31] to find A_s and B_s in Equation 4.4. It is important to understand that all the parameters of matrices A_s and B_s cannot be modeled at once. Individual test signals for different power resources are applied and corresponding parameters are modeled. When all the power resources are completed, we obtain matrices A_s and B_s which then can be used to predict temperature.

Equation 4.4 not only describes how the temperature evolves but it also enables temperature prediction at an arbitrary number of time steps ahead. In particular, the temperature at time step $k + n$ can be derived as:

$$T[k+n] = A_s^n T[k] + B_s \sum_{i=0}^{n-1} A_s^i P[k+n-i-1] \tag{4.5}$$

This equation predicts the temperature at a future time step for a given power consumption trajectory. Before changing the frequency of a CPU core, its power consumption can be computed using Equation 4.1 and plugged to this equation to predict the resulting temperature.

4.2.2 Thermal Model Validation

We implemented this predictor in Linux kernel, and validated its accuracy by comparing against actual measurements. The validation for the thermal model is shown in Figure 4.9 for Blowfish benchmark. The prediction interval used is $1\ s$.

We observed that the average prediction error is less than 3% $(1°C)$ with a prediction interval of up to 1 second, while the error is within 7% $(2.5°C)$ for as long as 5 seconds for the Templerun gaming benchmark, as shown in Figure 4.10. Further evaluation using the complete set of benchmarks is presented in Chapter 6.

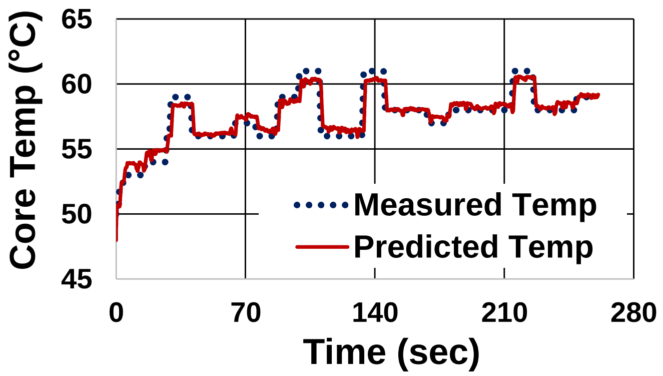

Figure 4.9: Thermal model validation for Blowfish benchmark with prediction interval of 1second

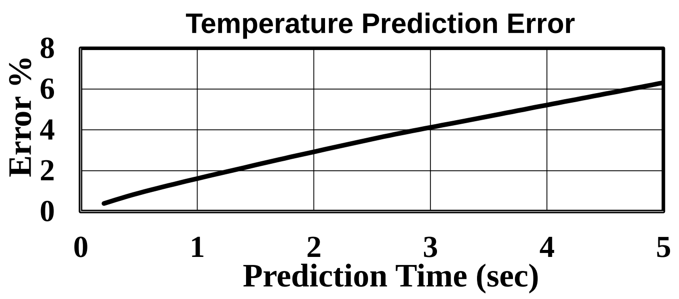

Figure 4.10: Average temperature prediction error for Templerun game

Both the power and thermal models are implemented in the kernel as a module. Other than the proposed technique the models can also be used by other thermal management techniques such as thermal aware scheduling and task migration. The described technique for thermal and power modeling can be used for other platforms as well.

Chapter 5

DYNAMIC THERMAL AND POWER MANAGEMENT

In the absence of a DTPM algorithm, the OS kernel wakes-up more processors and increases their frequencies as the workload intensifies. Consequently, increased power consumption elevates the temperature, which eventually results in a thermal violation. The proposed approach utilizes the power and thermal models introduced in Chapter 4 to dynamically compute a power budget, as outlined in Figure 5.1. Staying within this budget guarantees that no thermal violation will occur. Then, this power budget is used at run-time to limit the types, number and frequencies of the active resources. In this work, we use a prediction interval of “1s” since it is sufficient to control the temperature of our target platform. In general, accurate predictions up to “5s” can be made, as depicted in Figure 4.10.

5.1 Run-Time Power Budget Computation

Power budget is the maximum value of power that can be consumed by the processor without violating any constraints. Suppose the temperature constraint for each thermal hotspot is given by $[T_{constr}]_{N\times 1}$, where each entry gives the maximum permissible temper-

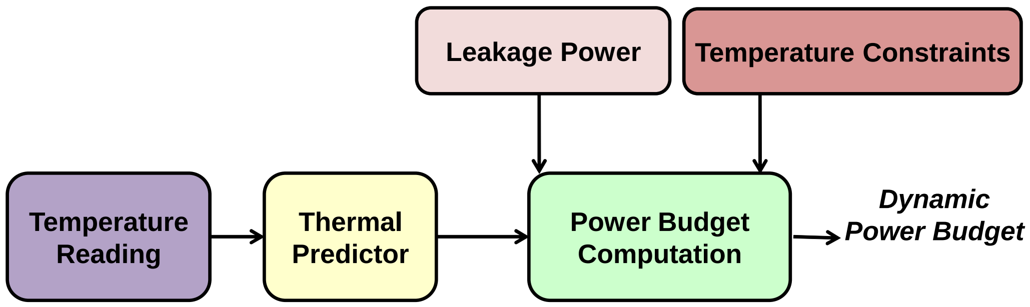

Figure 5.1: Temperature prediction and power budget computation

ature. Using Equation 4.4, we write

$$\begin{aligned} |T[k+1]| &\leq |T_{constr}| \\ |A_sT[k] + B_sP[k]| &\leq |T_{constr}| \end{aligned} \tag{5.1}$$

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

$$\begin{aligned} |A_sT[k] + B_sP[k]|_\infty &\leq T_{max} \\ max\;\{A_{s,i}T[k] + B_{s,i}P[k]\} &\leq T_{max} \quad 1 \leq i \leq N \end{aligned} \tag{5.2}$$

where $A_{s,i}$ and $B_{s,i}$ denote the i^{th} row of matrices A_s and B_s, respectively as shown by Equation 5.3 in detail.

$$\begin{bmatrix} T_{core0} \\ T_{core1} \\ T_{core2} \\ T_{core3} \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix} \begin{bmatrix} T_{core0} \\ T_{core1} \\ T_{core2} \\ T_{core3} \end{bmatrix} + \begin{bmatrix} b_{11} & b_{12} & b_{13} & b_{14} \\ b_{21} & b_{22} & b_{23} & b_{24} \\ b_{31} & b_{32} & b_{33} & b_{34} \\ b_{41} & b_{42} & b_{43} & b_{44} \end{bmatrix} \begin{bmatrix} P_{big} \\ P_{little} \\ P_{gpu} \\ P_{mem} \end{bmatrix} \tag{5.3}$$

Now, we convert the matrix inequality into a set of scalar inequalities, one for each thermal hotspot. Each row in A and B matrix corresponds to an equation for one core. Temperature constraints can be written as:

$$B_{s,i}P[k] \leq T_{max} - A_{s,i}T[k] \quad 1 \leq i \leq N \tag{5.4}$$

The right hand-side is known since we obtained A_s and B_s through system identification, and measure $T[k]$. We subtract the current temperature dependence. Consequently, run-time decisions are made such that $P(k)$ satisfies the power budget constraint given by Equation 5.4. The trigger value of the DTM algorithm can be varied for different systems while the algorithm remains the same. This equation has multiple possible answers and in order to achieve a unique solution, we solve it for equality such that the performance is maximized as shown in Equation 5.5. Here, instead of solving for all thermal hotspots we target the one with the maximum temperature and is most likely to violate constraints. Assuming core1 has the maximum temperature, we can write:

$$B_{s,1}P[k]_{total} = T_{max} - A_{s,1}T[k] \tag{5.5}$$

Thus now we obtain the total power budget. We calculate the dynamic power budget by subtracting the leakage power component from the total budget represented by Equation 5.6. Once we have the budget, we can finalize the configuration which adheres to it and control the temperature well within the limits. This budget will provide us with values for all power resources in the heterogeneous processor.

$$B_{s,1}P[k]_{dynamic} = T_{max} - A_{s,1}T[k] - P_{leakage} \tag{5.6}$$

5.2 DTPM Algorithm Implementation

The proposed algorithm is incorporated with the existing governors in the Linux kernel, as explained in Chapter 3 and illustrated in Figure 3.1. After calculating the power budget, we need to assign a configuration which will satisfy the budget and avoid violations. The performance also needs to be maximized while the temperature is being controlled. The power budget can be represented using the dynamic power equation as shown:

$$P_{budget} = \alpha C V_{dd}^2 f_{budget} \tag{5.7}$$

where f_{budget} is the frequency corresponding to the power budget. Parameters α and C have been already calculated and can be provided by the dynamic power model as explained in

Section 4.1. Since current V_{dd} is also known from measurements, f_{budget} is calculated using Equation 5.7.

Through empirical analysis, it is observed that big cluster has the maximum performance and moving to the little cluster might have biggest performance impact. So priority is to run the application in the big CPU cluster as far as possible. Let f_{min} and f_{max} be the minimum and maximum frequency values for the big cluster. First it is checked whether the frequency assigned by Equation 5.7 is within the big cluster frequency range as shown:

$$f_{min} \leq f_{budget} \leq f_{max} \tag{5.8}$$

If Equation 5.8 is satisfied, we assign f_{budget} to big cluster. Then we again compute the temperature in the next interval and observe decrease in temperature. In case f_{budget} is not constrained within the big cluster frequency range, then before moving to the little CPU cluster, we explore the option of turning off a core in the big cluster. If the power budget cannot be met with the current number of active cores, then the hottest core is put to sleep, and the tasks running on this core are migrated to the other cores by the kernel. Some applications tend to be scheduled such that they utilize a particular core and increase its temperature more than the other cores. We check whether this is the case, using Equation 5.9 as shown.

$$T_{hot} - T_i \geq \Delta \quad 1 \leq i \leq N \tag{5.9}$$

where, T_{hot} is the temperature of the hottest core, N is the total number of cores in the big cluster and Δ is the maximum temperature difference value allowed which is calculated empirically. If Equation 5.9 is true, the hottest core is turned off.

Similar to CPU, frequency value is set for GPU and other power resources if available. In case of other power resources, we have to deal with choice of frequency only unlike

CPU, where the decisions are complex. To summarize, the proposed algorithm first finds the maximum feasible frequencies under the available power budget. Finally, when the power budget is so small that it cannot be satisfied even with three big cores running at minimum possible frequency, then all the active tasks are migrated to the little cluster and big cores are put to sleep. Moving to the little cluster and reducing the GPU frequency (if GPU is active) are used as the last resort, since they have the biggest performance impact and migrating across clusters has a larger overhead based on our empirical evaluations.

Chapter 6

EXPERIMENTAL SETUP AND EVALUATION

The proposed algorithm is implemented in the Linux kernel version 3.4.76. In this chapter, we evaluate our approach by running multiple benchmarks.

6.1 Experimental Setup and Methodology

The setup includes the development platform, display, power sensors (internal powers), thermal sensors and power meter (total platform power) as shown in Figure 6.1. The internal sensor values are logged periodically and used in the kernel for implementing the algorithm.

6.1.1 Development Platform

The proposed framework is evaluated using the Odroid-XU+E platform [1] powered by Samsung Exynos 5410 processor. This MPSoC is a single ISA heterogeneous processor

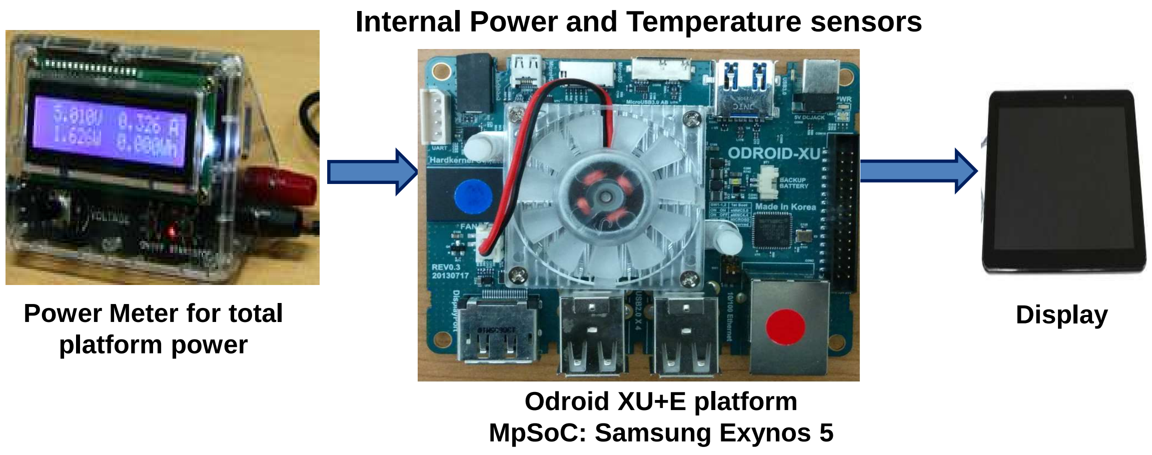
Internal Power and Temperature sensors
Power Meter for total platform power
Display
Odroid XU+E platform
MpSoC: Samsung Exynos 5

Figure 6.1: Experimental setup

which uses the ARM big.Little architecture [3]. It integrates 2 types of CPUs on the same SoC. The 2 types of CPUs are the big CPU cluster (4 ARM A15 cores) and the little CPU cluster (4 ARM A7 cores). Apart from that, it is also composed of a GPU, audio and video encoders/decoders and other basic components. The Odroid platform can activate only the big or the little cluster at a given time. The processor supports DVFS capabilities where the frequencies can be regulated in order to meet the workload requirements. The CPU clusters are symmetric i.e. all cores in the same CPU cluster need to be operated at the same frequency. Each core in the same cluster cannot run a different frequency value. There are nine discrete frequency levels in the big cluster as shown in Table 6.1 while there are eight frequency levels in the little CPU cluster as shown in Table 6.2. This processor is used commercially in mobile phones and tablets running Android operating system. In order to support images,games and videos, the GPU also supports DVFS and has 5 discrete frequency levels as shown in Table 6.3.

Table 6.1: Frequency Table for the big CPU cluster

Frequency (MHz)
800
900
1000
1100
1200
1300
1400
1500
1600

Table 6.2: Frequency Table for the little CPU cluster

Frequency (MHz)
500
600
700
800
900
1000
1100
1200

Table 6.3: Frequency table for GPU

Frequency (MHz)
177
266
350
480
533

6.1.2 Data Measurement

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 built-in temperature sensors located on each big core which are the thermal hotspots. Thus the temperature value for each big core can be measured. A UNIX script was prepared and used to log the sensor values for validation. The logged data was saved in form of tables as .CSV format. In the kernel, these sensor values were periodically measured and used to implement the thermal and power models. Apart from the sensor values we used `time` command in order to measure the execution time. Performance is measured as the amount of time taken to execute a particular application. *All the results reported in this thesis are direct measurements on this platform. Hence, the implementation overheads are included in the results.*

6.1.3 Benchmarks

We used 15 benchmarks, 11 from the Mi-Bench embedded benchmark suite [16], 3 frequently used game and video applications and one self written multi-threaded matrix multiplication code, which is mainly used during debugging. Mi-Bench is an embedded benchmark suite available for researchers. Since our focus is mobile phones and tablets, we included common mobile games and video applications. We adhere to a realistic setup i.e. when an application runs on a mobile phone, multiple background processes also load the processor. Likewise, while running each benchmark all background processes were allowed to run. Even if a benchmark is single threaded, there are many active threads in the system since the benchmarks run along with Android operating system stack and all other kernel background processes. Therefore, multiple cores were active during the experiments and this number varied dynamically. Finally, the games and video benchmarks utilized GPU, while the other benchmarks were CPU intensive. While running games and video applications, we executed matrix-multiplication benchmark in background to overload the CPU. The benchmarks and their relevant properties are summarized in Table 6.4. The benchmarks are also categorized according to their comparative CPU power consumption as low, medium and high. High being the benchmarks which consume more power than others.

Table 6.4: Benchmarks used in the experiments

Types	Benchmarks	Category
Security	Blowfish, Sha	Low, Medium
Network	Dijkstra, Patricia,	Low, Medium
Computational	Basicmath, Matrix Multiplication	High
	Bitcount, Qsort	Medium
Telecomm	CRC32, GSM, FFT	Low, Medium, High
Consumer	JPEG	Medium
Games	Angry-Birds, Temple-run	High
Video	Youtube	Low

A brief summary of all the benchmarks is as follows:

Audio/Video and Games: Common Android games like Templerun and Angry-Birds were used to emulate mobile phones and tablets. Apart from that audio/video application Youtube which is also a common mobile application was used.

CPU Intensive: Basicmath benchmark involves cubic function solving, integer square root and angle conversions from degrees to radians. Bit-count counts the number of bits in an array of integers while Quick-sort sorts a large array of strings into ascending order. A multi-threaded matrix multiplication application was developed to observe the behavior for multi-threaded benchmarks. Apart form mobile phones, these benchmarks are common embedded applications used in automotive and industrial scenarios.

JPEG encode/decode: JPEG is a consumer devices test benchmark. It is a standard compression image format used in cases when some data loss is acceptable. We took multiple images and then encoded as well as decoded them which is common in mobile phones.

Network and Security: The Security category includes several common algorithms for data encryption, decryption and hashing. Patricia is used to test the network capabilities of embedded processors. Other than Patricia, this category includes benchmarks like Blowfish, Sha, CRC32 and Dijkstra.

Telecommunications: With the explosive growth of the Internet, many portable consumer devices are integrating wireless communication. These benchmarks consist of voice encoding and decoding algorithms, frequency analysis and a check sum algorithm. FFT performs a Fast Fourier Transform and its inverse transform on an array of data. The Global Standard for Mobile (GSM) communications is the standard for voice encoding and decoding in Europe and many countries. The input data is small and large speech samples.

6.2 Experimental Configurations

We execute all the benchmarks in multiple configurations which are described below:

Default configuration (With fan) : We ran the benchmarks first with the default configuration of the target platform which uses a fan. The fan is activated when maximum core temperature exceeds $57°C$. Then, the fan speed is increased to 50% and 100% when the temperature passes $63°C$ and $68°C$, respectively. We emphasize that using a fan is *not feasible* when this chip is used in a smartphone or tablet, which is the case for Samsung Galaxy S4. Therefore, we evaluated two more solutions besides the proposed DTPM technique.

Without fan : We disabled the fan and re-ran all the benchmarks. Since the fan is not activated when the workload is low, we observe little or no changes for light activity. However, temperature increases quickly for high loads and keeps on increasing continuously. To avoid physical damage to the device, we limited the run time to a few minutes for these workloads. We also implemented a heuristic thermal management algorithm which mimics the fan control algorithm. Instead of increasing the fan speed, this heuristic throttles the frequency by 18% and 25% when the temperature passes $63°C$ and $68°C$, respectively.

Proposed DTPM algorithm : After compiling the whole kernel with our modifications, we flashed it to the device. The kernel function implementing our models is called periodically whenever the CPU frequency driver is executed (once every 100ms). We first ran the modified kernel with the power models and thermal predictor without taking any real action to assess the power and performance overhead. We did not observe any noticeable change in power and performance due to our models.

6.3 Experimental Evaluation

6.3.1 Temperature Prediction Accuracy

We ran each benchmark and predicted the temperature $T[k+10]$ at every control interval $T[k]$. The temperature one second (10 control intervals) ahead of time is predicted and the predictions are compared to the measured values at the end of each experiment. Figure 6.2 shows that the average prediction error is less than 3% ($1°C$) and it never exceeds 4% ($1.4°C$). One second prediction window is selected since 10 control intervals are sufficient to regulate the temperature. We also validated that for prediction windows as large as "5s" the prediction error increases moderately, as depicted in Figure 4.10.

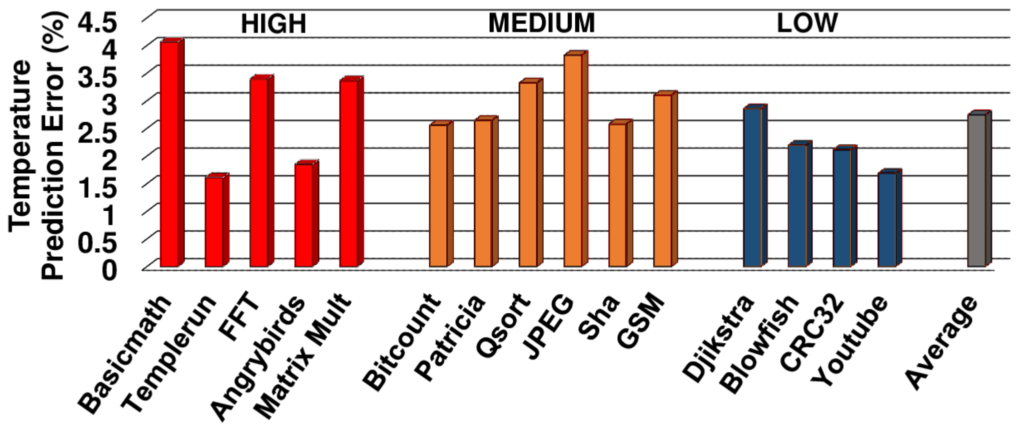

Figure 6.2: Temperature prediction error for all the benchmarks

6.3.2 Temperature Control and Stability

The objective of the DTPM algorithm is to ensure that the temperature is regulated successfully without using a fan. To provide a fair comparison with the default configuration, we used a temperature constraint of $63°C$ which is used in the fan control algorithm. We validated that the proposed algorithm can regulate the temperature for all of the benchmarks with minimal performance impact. As representative examples, the results for Templerun and Basicmath benchmarks are shown in Figure 6.3 and Figure 6.4, respectively. First, we observe that the proposed DTPM algorithm successfully limits the temperature to the specified constraint, which is easily violated without the fan. Furthermore, the temperature variation is significantly smaller than the default solution with fan, without using a fan, and the heuristic algorithm, which is not shown for clarity. More precisely, we observe as high as $6\times$ reduction in variance for both of the benchmarks, as summarized in Figure 6.5. Superior and smoother operation is achieved since the performance is throttled *only if* a thermal violation is predicted, and *only as much as needed* with the help of precise power budgeting.

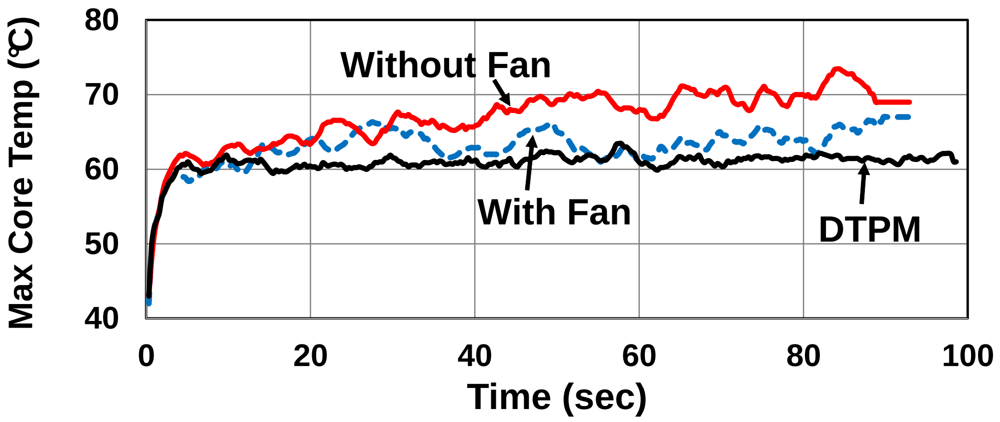

Figure 6.3: Temperature control for Templerun benchmark

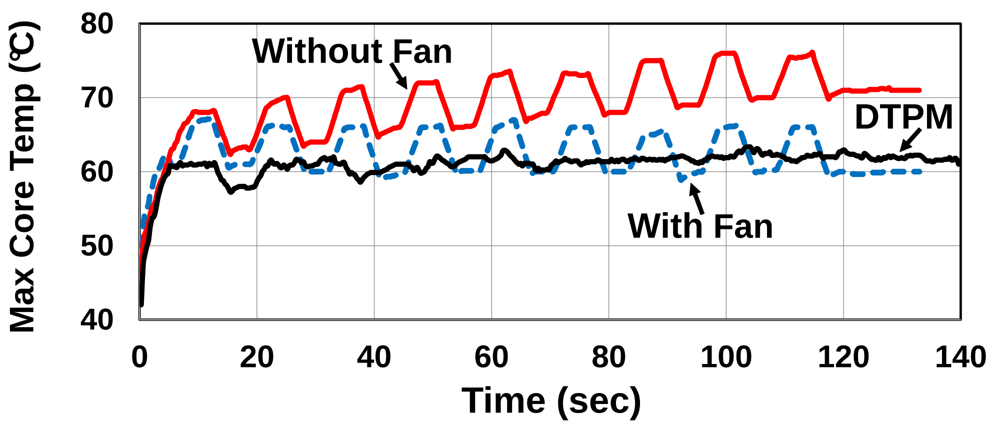

Figure 6.4: Temperature control for Basicmath benchmark

6.3.3 Power and Performance Evaluation

The proposed DTPM algorithm demotes the frequency and number of active cores only if the default values exceed the power budget. When the computation load is light, the temperature barely reaches the maximum constraint. Therefore, the proposed algorithm rarely interferes with the system and results in almost no change in frequencies of the resources.

Figure 6.6 shows the frequency and temperature variation for one of the low activity benchmarks Dijkstra for the default configuration and DTPM algorithm. Now it can be seen

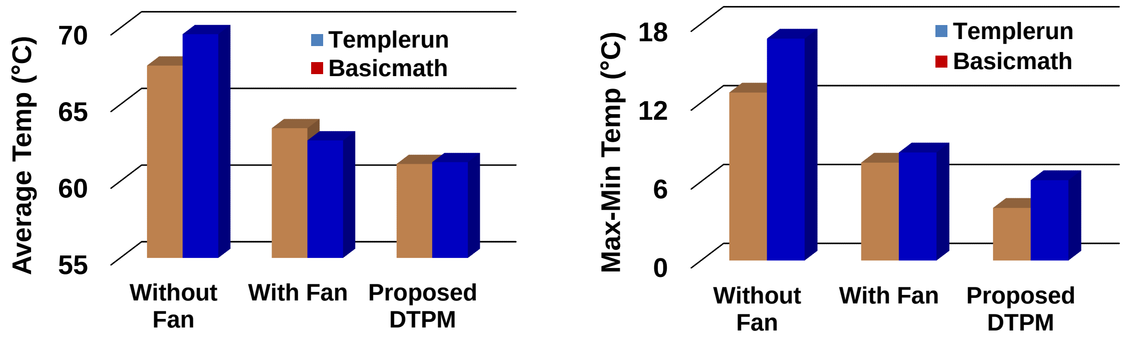

Figure 6.5: Thermal stability comparison for Templerun and Basicmath benchmark

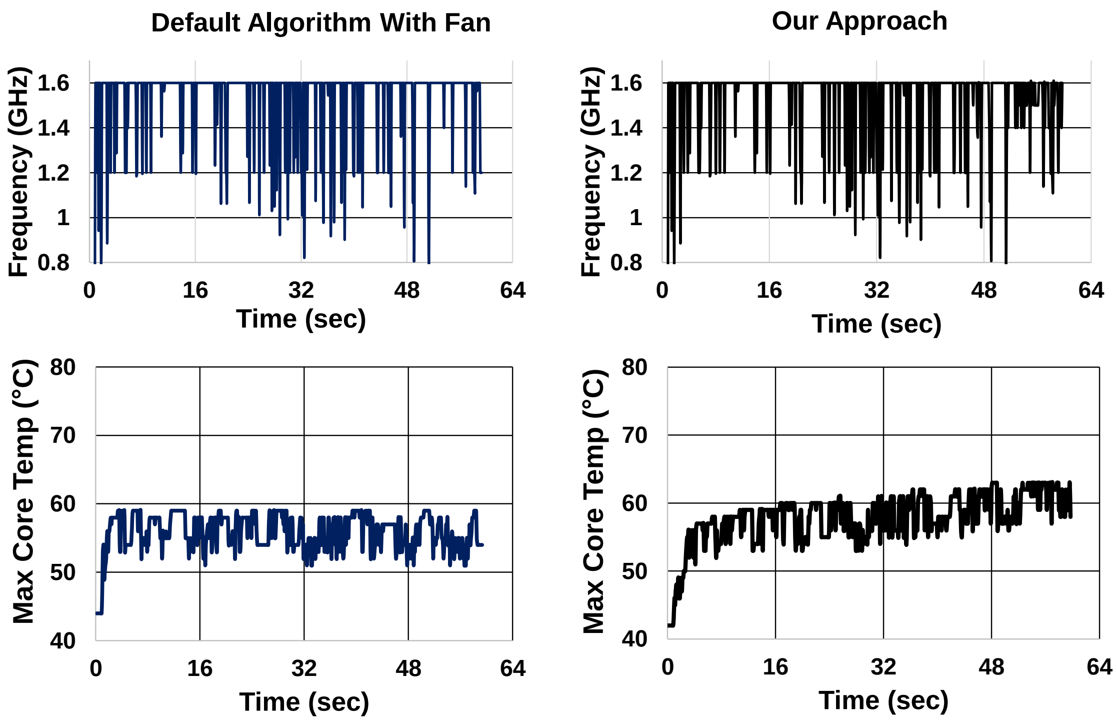

Figure 6.6: Frequency and temperature variation for Dijkstra benchmark while running the default configuration

that even when DTPM algorithm is executed there is not much need of thermal throttling and hence both the frequency variations are alike. But avoiding the fan, even if it is rarely active, results in around 3% platform power savings which corresponds to about 0.2 W.

As the computational load increases, the number of active cores and their frequencies increase. Hence, both the core and fan power consumption increase. Consequently, the power savings obtained using the proposed approach become more significant. Figure 6.7 shows how the frequency and temperature changes for one of the medium benchmarks Patricia when running the default and the proposed DTPM algorithm. In this case, it can be observed that how the frequency is throttled as calculated by the DTPM algorithm. We achieve 8% power savings for medium activity benchmarks on average.

When activity of the benchmarks increases even further, the rise in temperature occurs

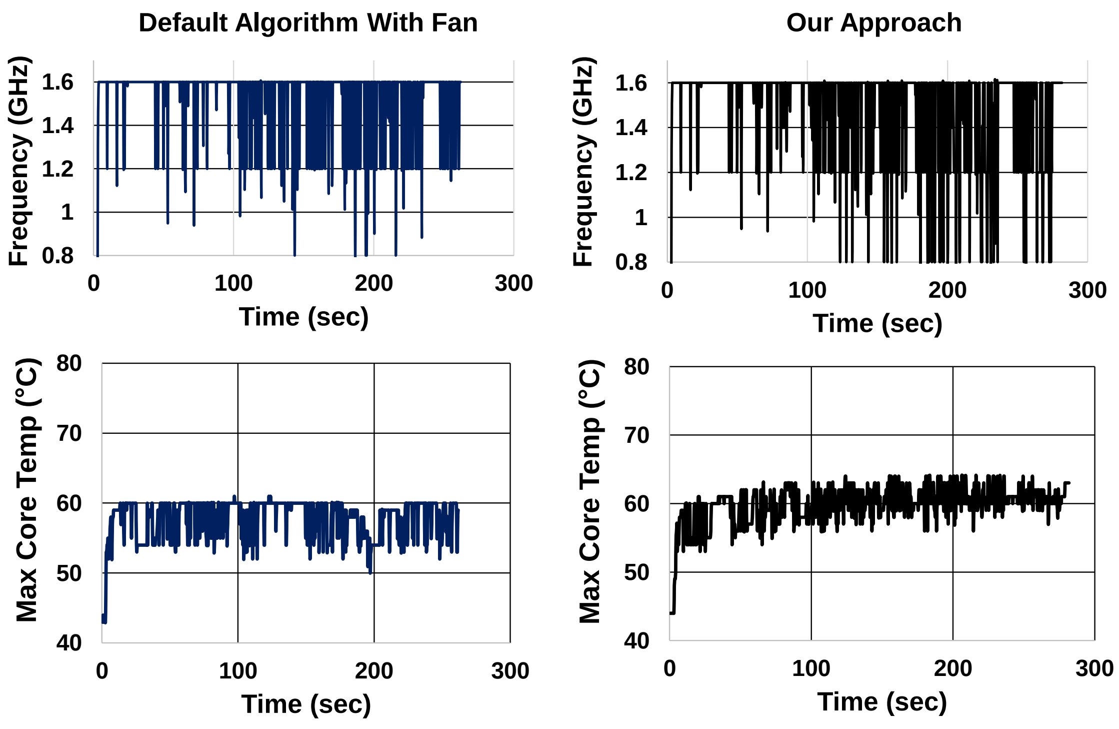

Figure 6.7: Frequency and temperature variation for Patricia benchmark while running the default configuration

frequently. Whenever the temperature rises, the frequency and configuration is calculated and set by the drivers in the kernel. Figure 6.8 shows the frequency and temperature variation for one of the high activity benchmarks(Matrix-Multiplication) for the default configuration and DTPM algorithm. A fan was employed while running the default configuration, so we do not see reduction in frequencies due to rise in temperature.The marked regions in Figure 6.8 indicate thermal throttling due to increase in temperature. The variation in frequency can be clearly distinguished in this case. On average, 14% savings for high activity benchmarks are observed. It is important to note that these savings are significant since they are at the platform level. For example, 14% savings corresponds to 0.7 W savings, which would increase the lifetime of a typical smartphone battery by around 25% from 2h to 2h30m under continuous use.

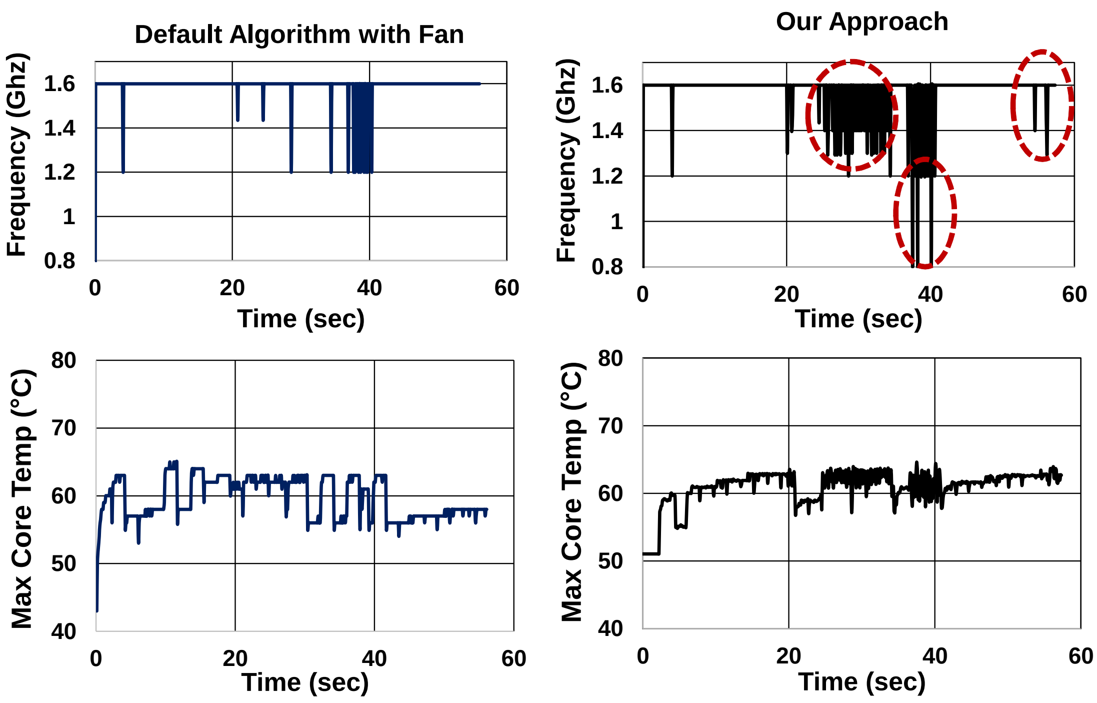

Figure 6.8: Frequency and temperature variation for Matrix Multiplication benchmark while running default configuration

Minimum Performance Degradation: For any embedded system the maximum performance can be achieved in presence of an active cooling component as the system will run at maximum possible frequency. When we remove the fan, the temperature has to be controlled using a DTPM algorithm. Thus we compare our algorithm with the best case which actually is not a feasible solution in mobile systems. We also implemented reactive heuristic approaches which lowers the frequency after temperature threshold is reached. The reactive DTPM algorithm that mimics the fan control results in around 20% loss in performance measured by execution time. Despite significant power savings, the performance loss is only 3.3% on average, while it is less than 1% for low activity benchmarks. The performance loss hardly reaches 5% even for the most demanding applications. The power and performance results have been summarized in Figure 6.9. Similar results are shown for multi-threaded benchmarks in Figure 6.10

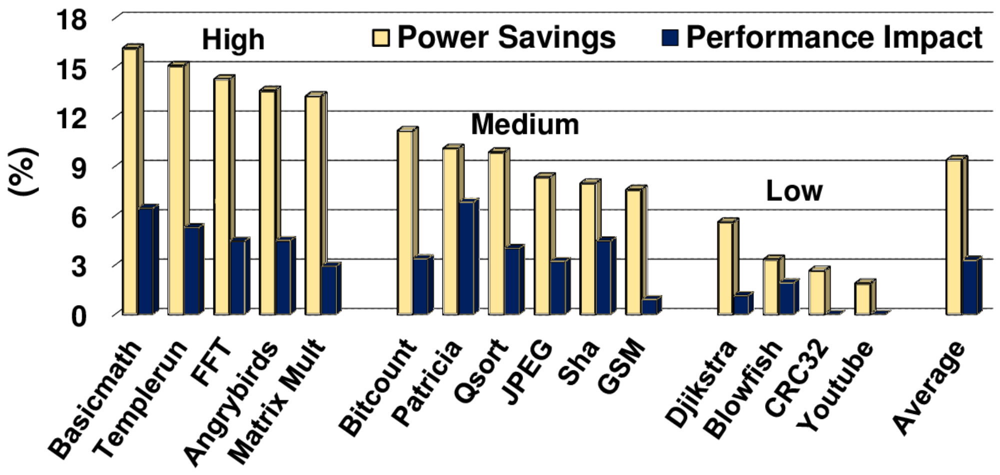

Figure 6.9: Power savings and performance loss summary

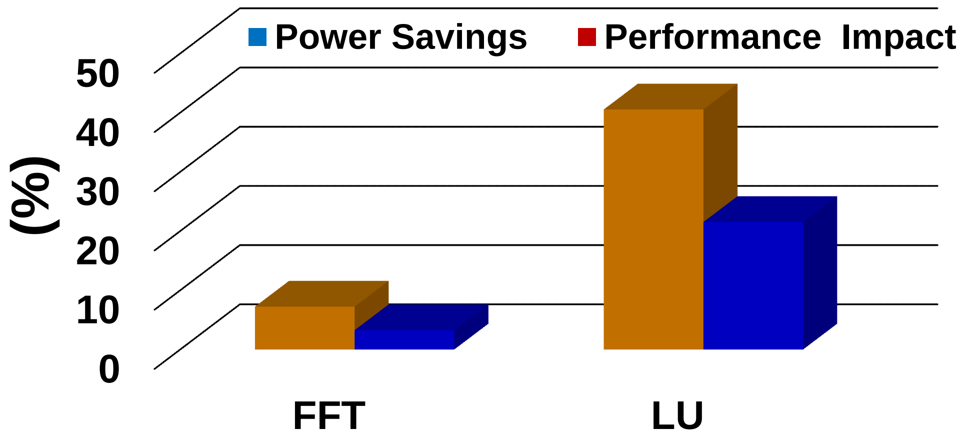

Figure 6.10: Power savings and performance loss summary for multi-threaded benchmarks

Chapter 7

CONCLUSION AND FUTURE WORK

In this thesis, we presented a practical temperature prediction methodology and a DTPM algorithm for heterogeneous MPSoCs. The proposed approach calculates a precise power budget based on temperature predictions at run-time. Then, this budget is used to control the type of cores (big or little), number of active cores and frequency of the cores. 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 power, thermal, and reliability advantages. In particular, it regulates the temperature more effectively than the default configuration which uses a fan, and on average offers 10% platform power savings with 3.3% loss in performance.

7.1 Future Work

In this work we focused on the CPU. Some benchmarks and applications utilize the GPU more than CPU and hence it becomes necessary to throttle GPU as well. In future, we plan to extend our power budget approach to make use of the heterogeneous processor in true sense. The power budget distribution among components is shown in Figure 7.1. The power distribution problem is a np-hard problem and we need to solve it dynamically while maximizing performance.

The cost function shown in Equation 7.1 corresponds to the execution time which we need to minimize.

$$J(f_1, f_2 ... f_n) = \sum_{i=1}^{n} \frac{c_i}{f_i} \tag{7.1}$$

In this equation c_i is the performance parameter for each component of the heterogeneous processor. The cost function will be minimized subject to the constraint of power

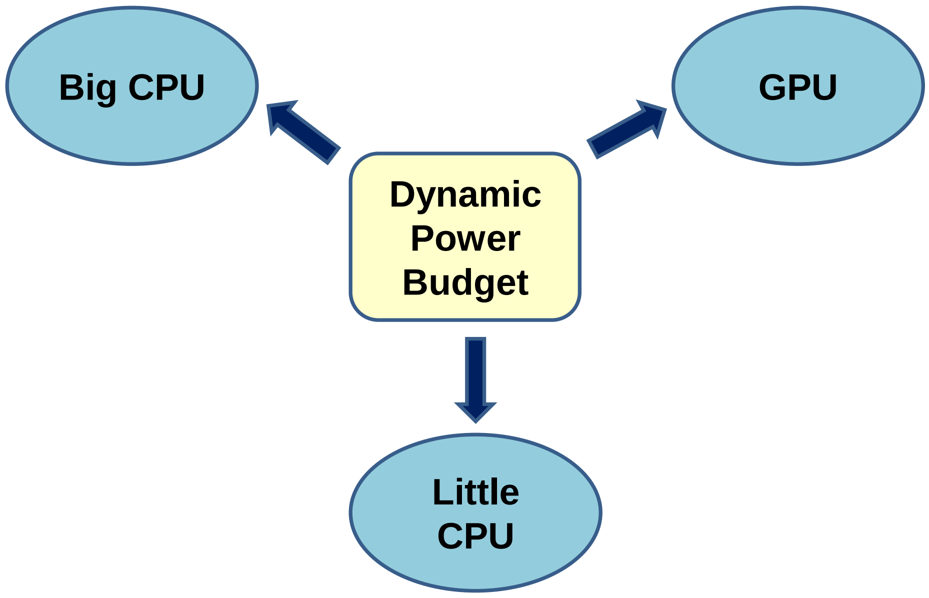

Figure 7.1: Power distribution in heterogeneous processor

budget as represented by Equation 7.2.

$$P(f_1, f_2 ... f_n) = \sum_{i=1}^{n} a_i {f_i}^3 \ \leq \ P_{budget} \tag{7.2}$$

Power distribution amongst the components is a difficult problem. To find an optimal frequency corresponding to each component such that the performance is maximized while satisfying the power budget further adds to the complexity. Branch and bound algorithm solves this problem theoretically, but is limited during implementation by the use of recursive function in the linux kernel source due to kernel stack issues. Hence we throttle the frequency of the components which has least affect on performance as follows in Equation 7.3.

$$\begin{aligned} &J(f_{CPU-1}, f_{GPU}) \quad if \quad \Delta J(f_{CPU-1}, f_{GPU}) \leq \Delta J(f_{CPU}, f_{GPU-1}) \\ &J(f_{CPU}, f_{GPU-1}) \quad if \quad \Delta J(f_{CPU}, f_{GPU-1}) \leq \Delta J(f_{CPU-1}, f_{GPU}) \end{aligned} \tag{7.3}$$

REFERENCES

[1] ODROID − XU + E. http://www.hardkernel.com/main/main.php.

[2] M. A. Al Faruque, J. Jahn, T. Ebi, and J. Henkel. Runtime thermal management using software agents for multi-and many-core architectures. *IEEE Design & Test of Computers*, 27(6)(6):58–68, 2010.

[3] ARM. big.little processing. http://www.arm.com/products/ processors/technologies/biglittleprocessing.php.

[4] D. Brooks, R. P. Dick, R. Joseph, and L. Shang. Power, thermal, and reliability modeling in nanometer-scale microprocessors. *IEEE Micro*, 27(3):49–62, 2007.

[5] D. Brooks and M. Martonosi. Dynamic thermal management for high-performance microprocessors. In *High-Performance Computer Architecture, 2001. HPCA. The Seventh International Symposium on*, pages 171–182. IEEE, 2001.

[6] A. Carroll and G. Heiser. An analysis of power consumption in a smartphone. In *USENIX annual technical conference*, pages 271–285, 2010.

[7] A. P. Chandrakasan, W. J. Bowhill, and F. Fox. *Design of high-performance microprocessor circuits*. Wiley-IEEE press, 2000.

[8] P. Chaparro, J. González, G. Magklis, C. Qiong, and A. González. Understanding the thermal implications of multi-core architectures. *Parallel and Distributed Systems, IEEE Transactions on*, 18(8):1055–1065, 2007.

[9] J. Choi et al. Thermal-aware task scheduling at the system software level. In *Proc. of Int. Symp. on Low Power Electron. and Design*, pages 213–218, 2007.

[10] A. K. Coskun, J. L. Ayala, D. Atienza, T. S. Rosing, and Y. Leblebici. Dynamic thermal management in 3d multicore architectures. In *Proc. of DATE*, pages 1410–1415, 2009.

[11] A. K. Coskun, T. S. Rosing, and K. C. Gross. Utilizing predictors for efficient thermal management in multiprocessor socs. *IEEE Trans. on CAD of Integrated Circuits and Syst.,*, 28(10):1503–1516, 2009.

[12] J. Donald and M. Martonosi. Techniques for multicore thermal management: Classification and new exploration. In *ACM SIGARCH Computer Architecture News*, volume 34, pages 78–88, 2006.

[13] S. Fok, W. Shen, and F. Tan. Cooling of portable hand-held electronic devices using phase change materials in finned heat sinks. *International Journal of Thermal Sciences*, 49(1):109–117, 2010.

[14] P. Greenhalgh. Big. little processing with arm cortex-a15 & cortex-a7. *ARM White Paper*, 2011.

[15] A. Grove. Changing vectors of moores law. In *Keynote speech, International Electron Devices Meeting*, 2002.

[16] 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*, pages 3–14, 2001.

[17] F. Hameed, M. Faruque, and J. Henkel. Dynamic thermal management in 3d multi-core architecture through run-time adaptation. In *Proc. of DATE*, pages 1–6, 2011.

[18] H. Hanson, S. W. Keckler, S. Ghiasi, K. Rajamani, F. Rawson, and J. Rubio. Thermal response to dvfs: Analysis with an intel pentium m. In *Proceedings of the 2007 international symposium on Low power electronics and design*, pages 219–224. ACM, 2007.

[19] V. Hanumaiah, S. Vrudhula, and K. S. Chatha. Performance optimal online dvfs and task migration techniques for thermally constrained multi-core processors. *IEEE Trans on CAD of Integrated Circuits and Syst.*, 30(11):1677–1690, 2011.

[20] W. Huang et al. Hotspot: A compact thermal modeling methodology for early-stage vlsi design. *IEEE Trans. on Very Large Scale Integration Syst.*, 14(5):501–513, 2006.

[21] C. J. Hughes, J. Srinivasan, and S. V. Adve. Saving energy with architectural and frequency adaptations for multimedia applications. In *Proceedings of the 34th annual ACM/IEEE international symposium on Microarchitecture*, pages 250–261. IEEE Computer Society, 2001.

[22] 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 DATE*, pages 1–6, 2011.

[23] D. Kadjo, U. Y. Ogras, R. Ayoub, M. Kishinevsky, and P. Gratz. Towards platform level power management in mobile systems. In *In Proc. of System-on-Chip Conf*, pages 146–151, 2014.

[24] O. Khan and S. Kundu. Hardware/software co-design architecture for thermal management of chip multiprocessors. In *Proceedings of the Conference on Design, Automation and Test in Europe*, pages 952–957. European Design and Automation Association, 2009.

[25] R. Kumar and V. Kursun. Impact of temperature fluctuations on circuit characteristics in 180nm and 65nm cmos technologies. In *Circuits and Systems, 2006. ISCAS 2006. Proceedings. 2006 IEEE International Symposium on*, pages 4–pp. IEEE, 2006.

[26] 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.

[27] T.-Y. T. Lee, B. Chambers, and K. Ramakrishna. Thermal management of handheld telecommunication products. *Electronics Cooling Magazine*, 4(2):30–33, 1998.

[28] W. Liao, J. M. Basile, and L. He. Leakage power modeling and reduction with data retention. In *Proceedings of the 2002 IEEE/ACM international conference on Computer-aided design*, pages 714–719. ACM, 2002.

[29] Y. Liu, R. P. Dick, L. Shang, and H. Yang. Accurate temperature-dependent integrated circuit leakage power estimation is easy. In *Proc. of DATE*, pages 1526–1531, 2007.

[30] L. Ljung. System identification toolbox. *The Matlab Users Guide*, 1988.

[31] L. Ljung. System identification toolbox for use with MATLAB. 2007.

[32] R. McGowen. Adaptive designs for power and thermal optimization. In *Proc. of ICCAD*, pages 118–121, 2005.

[33] R. Mukherjee and S. O. Memik. Physical aware frequency selection for dynamic thermal management in multi-core systems. In *Proc. of ICCAD*, pages 547–552, 2006.

[34] S. Murali et al. Temperature control of high-performance multi-core platforms using convex optimization. In *Proc. of DATE*, pages 110–115, 2008.

[35] 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 DAC*, page 174, 2013.

[36] V. Pallipadi and A. Starikovskiy. The ondemand governor. In *Proc. of the Linux Symp.*, volume 2, pages 215–230, 2006.

[37] M. Pedram and S. Nazarian. Thermal modeling, analysis, and management in vlsi circuits: Principles and methods. *Proceedings of the IEEE*, 94(8):1487–1501, 2006.

[38] K. Roy, S. Mukhopadhyay, and H. Mahmoodi-Meimand. Leakage current mechanisms and leakage reduction techniques in deep-submicrometer cmos circuits. *Proc. of the IEEE*, 91(2):305–327, 2003.

[39] S. Sharifi, D. Krishnaswamy, and T. S. Rosing. Prometheus: A proactive method for thermal management of heterogeneous mpsocs. *IEEE Trans. on CAD of Integrated Circuits and Syst.*, pages 1110–1123, 2013.

[40] S. Sharifi and T. S. Rosing. Accurate direct and indirect on-chip temperature sensing for efficient dynamic thermal management. *IEEE Trans. on CAD of Integrated Circuits and Syst.*, 29(10):1586–1599, 2010.

[41] J. W. Sheaffer, K. Skadron, and D. P. Luebke. Studying thermal management for graphics-processor architectures. In *Performance Analysis of Systems and Software, 2005. ISPASS 2005. IEEE International Symposium on*, pages 54–65. IEEE, 2005.

[42] K. Skadron et al. Temperature-aware microarchitecture. In *ACM SIGARCH Computer Architecture News*, volume 31, pages 2–13, 2003.

[43] K. Skadron et al. Temperature-aware microarchitecture: Modeling and implementation. *ACM Trans. on Arch. and Code Optimization*, 1(1):94–125, 2004.

[44] Y. Taur and T. H. Ning. *Fundamentals of modern VLSI devices*. Cambridge university press, 2009.

[45] Y. Wang, K. Ma, and X. Wang. Temperature-constrained power control for chip multiprocessors with online model estimation. In *ACM SIGARCH Comp. Arch. News*, volume 37, pages 314–324, 2009.

[46] 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 ICCAD*, pages 242–247, 2013.

[47] J. Yang et al. Dynamic thermal management through task scheduling. In *Proc. Int. Symp. on Perf. Analysis of Systems and Software.*, pages 191–201, 2008.

[48] W. Ye, N. Vijaykrishnan, M. Kandemir, and M. J. Irwin. The design and use of simplepower: a cycle-accurate energy estimation tool. In *Proceedings of the 37th Annual Design Automation Conference*, pages 340–345. ACM, 2000.

[49] I. Yeo, C. C. Liu, and E. J. Kim. Predictive dynamic thermal management for multicore systems. In *Proc. of DAC*, pages 734–739, 2008.

[50] F. Zanini, D. Atienza, L. Benini, and G. De Micheli. Multicore thermal management with model predictive control. In *European Conf. on Circuit Theory and Design*, pages 711–714, 2009.

[51] M. Zapater, O. Tuncer, J. L. Ayala, J. M. Moya, K. Vaidyanathan, K. Gross, and A. K. Coskun. Leakage-aware cooling management for improving server energy efficiency.

[52] L. Zhang et al. Accurate online power estimation and automatic battery behavior based power model generation for smartphones. In *Proc. of Int. Conf.on Hardware/-Software Codesign and System Synthesis*, pages 105–114, 2010.