Modular Modelling of an Embedded Mobile CPU-GPU Chip for Feature Estimation

Oussama Djedidi, Mohand Arab Djeziri, Nacer K. M'Sirdi and Aziz Naamene
Laboratoire des Sciences de l'Information et des Systèmes (LSIS), Aix-Marseille University, Marseille, France

Keywords: Embedded Chips Central Processing Units, Graphics Processing Units, Modelling Simulation.

Abstract: This paper deals with the modelling of a CPU-GPU chip embedded in an Android phone. The model is used for the estimation of variables that characterise the operating state of System on Chip (SoC). The proposed model is built to demonstrate the causal relationships between the variables, through its interconnected structure of subsystems. This structure allows the extension of other components or the easy exchange of subsystems in the case of a change in components or operating mode. The model developed here requires no additional instrumentation—other than the one present on the phone—which facilitates its implementation. It is used for the estimation of the state of the system and can also be used for monitoring and behaviour prediction. The model is validated and the results are promising for further implementation.

1 INTRODUCTION

Modelling of Central Processing Units (CPU) and Graphics Processing Units (GPU) chips is done for a multitude of purposes. The first and foremost is to ensure performance reliability which can be done either by selecting the algorithms to be implemented and predicting their performance (M'Sirdi et al., 2016; Williams et al., 2009; Meng and Skadron, 2011), or running background programs in parallel that are able to predict the performance output of some components (Ardalani et al., 2015), or even by thoroughly studying the component itself (Kim et al., 2012a). However, in this work, we are more interested in the monitoring of the operating state of the chip. Hence, we focus our modelling on the physical variables i.e. frequency, voltage, power consumption, and temperatures in the SoC.

Reducing power consumption and improving battery life in mobile phones is a very active research area. Thus, several models estimating the power consumption of embedded CPU-GPU chips were developed. For instance, Zhang *et al.* propose a model called *"PowerBooter"* to the estimate power consumption of a smartphone through its battery sensors (Zhang et al., 2010). Another study used system calls to estimate the power consumptions (Pathak et al., 2011), while others used polynomial and regression models in a series of works to estimate the power consumption of smartphone components including the CPU and the GPU (Minyong Kim et al., 2012; Kim et al., 2012b; Kim and Chung, 2013; Kim et al., 2015). These studies offer simple modelling techniques and also show a clear dependence between frequency and power consumption.

Furthermore, several other notable studies focused on the power consumption of just the GPU. For instance, Leng *et al.* estimate the power consumption, in a discrete GPU, as a sum of three components, the dynamic power (power consumed by the GPU while running computations), the leakage power (essentially linked to transistor leakage currents in the architecture of the chip) and idle power (assumed constant) (Leng et al., 2013). Additionally, in their work, Adhinarayanan *et al.* present a GPU power estimator based on multiple regression techniques, that uses performance counters and temperature to deliver accurate power estimation at runtime (Adhinarayanan et al., 2016).

As Temperature is an important variable in the functioning and life cycle of CPU and GPU, several models were developed in the literature to study its profile and behaviour, such as the model developed by Hong *et al.*, which is a switched system with two first order models, one for the rise in temperature and the second for cooling. The time constants of these models were identified experimentally, and static gains are presented as a function of the maximum temperature of the system, power consumption, and memory intensity (Hong and Kim, 2010).

Djedidi, O., Djeziri, M., M'Sirdi, N. and Naamane, A.
Modular Modelling of an Embedded Mobile CPU-GPU Chip for Feature Estimation.
DOI: 10.5220/0006470803380345
In *Proceedings of the 14th International Conference on Informatics in Control, Automation and Robotics (ICINCO 2017) - Volume 1*, pages 338-345
ISBN: 978-989-758-263-9

Amongst the works mentioned above, there are fine grained models of the CPU-GPU chips, like the model proposed by Kim *et al.* which models performance as a function of of memory warps and models temperature as a function of memory intensity (Kim et al., 2012a). Nonetheless, such models use variables which are neither measurable nor accessible to read, such as the memory intensity for instance or are specific to certain GPU brands like the warps.

Moreover, because of the complexity of the system and physical phenomena, inputs of the models constructed by machine learning are not the subject of a formal proof of the interaction between the inputs and estimated outputs. The choice of model inputs is often the result of observations and experimental tests in various operating conditions. These models are often built to give developers feedback on on performance, and power consumption, which will be used for program optimisation. The temperature estimation is, generally, used to set temperature thresholds for thermal throttling to keep the component out of harm's way.

The model proposed in this paper is built in a modular structure, and is composed of a set of interconnected subsystems, where the control components (subsystems for frequency and voltage) are clearly distinguished from the operating ones (power and temperature), allowing thus the model to adapt to changes in the system operating modes. The subsystems are inspired by those presented in the studies above and the algorithms given by manufacturers. The developed model is then used to estimate a set of variables characterising the operating state of the CPU-GPU system. Our main goal behind the development of this model is to use its generated estimation to monitor the state of the chip.

This paper is organised as follows. The targeted system is described is Section 2. In Section 3, we describe the data aquisition, then the modelling method and the models of each subsystem are detailed in Section 4. In Section 5, the model is validated by comparing the real and estimated outputs, then, the results are discussed. Finally, a conclusion is given in Section 6.

2 TARGETED SYSTEM DESCRIPTION

The electronic devices market proposes different CPU and GPU architectures. Each family of architecture comes with its own advantages and—naturally—complexities. One of the main goals of this work is to construct a general dynamic model for embedded CPU-GPU chips. Thus, for testing and undertaking experimental validation, the choice of a System on Chip (SoC) to be studied is firstly based on the software it runs.

For this study, we settled on an Android$^{\mathrm{TM}}$ phone. Android$^{\mathrm{TM}}$ phones are very popular, and with its programming framework, aoolications are effortlessly transferable to other devices. Moreover, Android is built on top of the popular open source operating system (OS) *Linux*. The availability of the code for this OS makes some of the needed parameters accessible for reading. Furthermore, the system's terminal, via Android Debugging Bridge (ADB), gives access to all system files, some of which are used for variables reading.

The smartphone we used in this study runs on Android$^{\mathrm{TM}}$ Marshmallow 6.0, and is equipped with a SoC harbouring a quad-core processor with variable frequencies—through frequency and voltage scaling—ranging between 300 MHz–2.45 GHz. It also sports an OpenGL$^{\mathrm{TM}}$ ES 3.0 capable GPU with frequencies ranging between 200 MHz–578 MHz. The SoC also contains the systems 2 GB RAM.

3 VARIABLES AND DATA ACQUISITION

To analyse and model the dynamics of the CPU-GPU chip, one must initially recover the relevant readings characterising the operating state of the system. For our modelling purposes, the needed readings are the loads of both the CPU and the GPU, the working frequencies, temperatures of the cores and other components, the voltage of each core and the power delivered by the battery. All of these variables–except for the temperature–have very fast variations. Therefore, the quality of the results depends directly on the chosen sampling period. Hence, we choose the minimum period necessary to follow the changes of the fast changing variable: the frequency. Frequency is evaluated by by frequency Governor (see paragraph 4.1). Hence, the minimum period for this program to reevaluate the load and change of the frequency accordingly was set as sampling period (in this case study, $T_s = 20\,\mathrm{ms}$).

Since we do not wish to hinder the normal behaviour of the system while reading the variables, we have written a lightweight application for data acquisition. However, during periods of high loads, the system will give priority to the user interface and system operations over background services (like our app), which sometimes leads to the scheduling of our application excecution being pushed back, and thus the chosen sampling period T_s being not respected. Fig. 1

shows the evolution of the sampling periods during a reading experience where some sampling values are well beyond the chosen period T_s and even reaching tens of seconds.

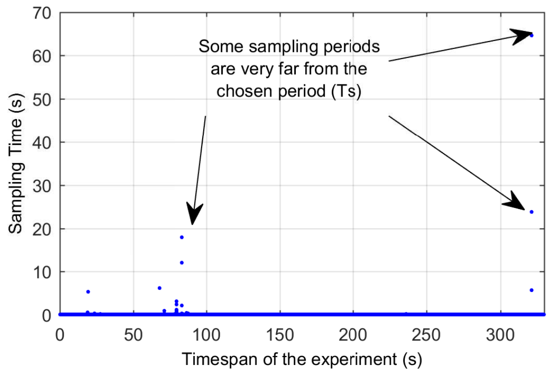

Figure 1: Sampling time variation over time.

The second difficulty in reading the variables is related to the management of instruction queues by the OS. Again, during periods of high the CPU loads, we came to notice that reading time—the time difference between the start of the reading process and its end—is sometimes very long, even reaching seconds (See Fig. 2), leading to the conclusion that the reading was interrupted. This renders the values taken during the said reading false (unsynchronized data, measurements delays and missing data).

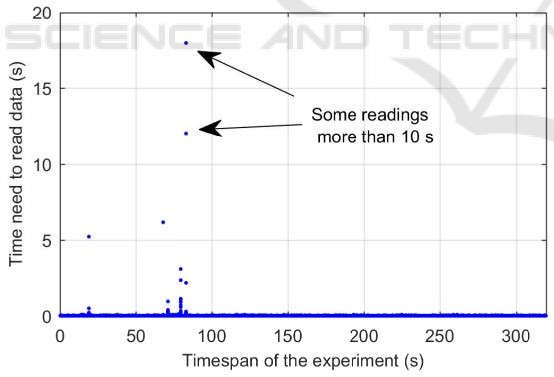

Figure 2: Reading time evolution and variation over time.

These two problems of data acquisition are taken into account and the data acquired during long sample times or interrupted readings will be automatically deleted from the database used in the next section for learning and model assessment.

4 SYSTEM MODELLING

The CPU-GPU chip is a complex system with variable structure, whose dynamics are nonlinear and not continuous. Hence, a gradual approach and a modular structure are adopted for its modelling as shown in Fig. 3. This modular approach provides for a gradual analysis and modelling each of the subsystems and account for the variable structure of the system. It also allows easy integration of subsystems in case of any changes. Fig. 3 shows a global vision of the interconnected and gradual modelling approach. In the following paragraphs, each of the subsystems will be detailed.

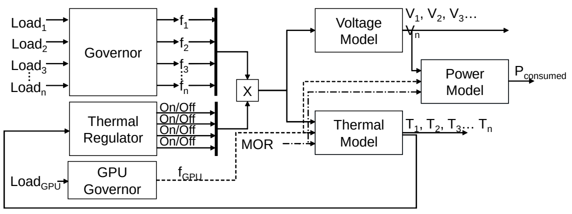

Figure 3: Diagram of the proposed model of the CPU-GPU system.

4.1 The Frequency Scaling Governor

Frequency scaling is carried out via *Governors*. On the studied system, several frequency governors exist. These governors calculate the frequency according to usage needs as well as several other factors (speed, power consumption). The governor we are modelling in this work is the *interactive* governor. However, thanks to the modular structure of the model, any governor can replace the one considered in this case study.

The *interactive* governor increases and decreases the frequency of each core as a function of the load and specific timers. When the CPU is back from the idle state, the governor starts a countdown timer with a predefined *timer_rate* value at the end of which, if the load exceeds a given value (*go_hispeed_load*), the governor calculates a new frequency for which the load will be equal or closest to *target_load* value.

The governor also takes into account sudden heavy loads by directly scaling up the frequency to *hispeed_freq* if the current frequency is below it for better reactiveness and to avoid unexpected CPU bottlenecks and sluggish performance. In addition, if the frequency of a core is greater or equal to *hispeed_freq*, the core must stay on the same frequency at least a period of *above_hispeed_delay*, before scaling up, and a period of *min_sample_time* (or *sampling_down_factor* if the current frequency is the maximum frequency) before scaling down. The last time constant, noted *timer_slack*, is an additional period of time that the core has to wait before shutting down if the load is equal to zero. The governor modelling algorithm is given in Fig. 4 and Fig. 5. This model is engineered from the original source code available in the code deposits of the manufacturer of the studied smartphone (Samsung, 2016).

```
procedure INTERACTIVE($Load, Time, currentFrequency$)
    define above_hispeed_delay, hispeed_freq, go_hispeed_load, min_sample_time, target_load, sampling_down_factor, timer_slack, timer_rate
    if $Initialisation = 0$ then
        $currentHiSpeedTimer \leftarrow 0$
        $currentTimers \leftarrow 0$
        $currentDownTimers \leftarrow 0$
        $currentTimersSlack \leftarrow 0$
        $oldTime \leftarrow Time$;
        $Initialisation \leftarrow 1$
    else
        $currentHiSpeedTimer \leftarrow currentHiSpeedTimer + (Time - oldTime)$
        $currentTimers \leftarrow currentTimers + (Time - oldTime)$
        $currentDownTimers \leftarrow currentDownTimers + (Time - oldTime)$
        $currentTimersSlack \leftarrow currentTimersSlack + (Time - oldTime)$
    end if
    if ($currentTimers \geq$ timer_rate) then
        if ($Load \geq$ go_hispeed_load) then
            if ($currentFrequency <$ hispeed_freq $\wedge$ $ChooseFreq(Load, currentFrequency) <$ hispeed_freq) then
                $newFrequency \leftarrow$ hispeed_freq
            else if ($ChooseFreq(Load, currentFrequency) >$ hispeed_freq) $\wedge$ ($currentHiSpeedTimer \geq$ above_hispeed_delay) then
                $newFrequency \leftarrow ChooseFreq(Load, currentFrequency)$
            else if ($ChooseFreq(Load, currentFrequency) < currentFrequency$) $\wedge$ ($currentDownTimers \geq$ min_sample_time) then
                $newFrequency \leftarrow ChooseFreq(Load, currentFrequency)$
            else if ($Load = 0$) $\wedge$ ($currentTimersSlack \geq$ timer_slack) then
                $newFrequency \leftarrow 0$
            end if
            $currentHiSpeedTimer \leftarrow 0$
            $currentDownTimers \leftarrow 0$
        end if
    else
        $newFrequency \leftarrow currentFrequency$
    end if
    return $newFrequency$
end procedure
```

Figure 4: The Interactive Governor algorithm (I—The general algorithm).

4.2 Voltage Model

Recovered readings show that, like frequency, voltage is discrete and varies in a set of well-defined values. Voltage readings are plotted against the frequencies, in Fig. 6, in order to see the relationship between

```
procedure CHOOSEFREQ($Load, currentFrequency$)
    define $target\_load$
    define $FREQUENCY\_TABLE$
    $freq = currentFrequency \times (Load/target\_load)$
    $TableSize \leftarrow 16$                                ▷ In our study case
    $Minimum \leftarrow freq$;
    $BestIndex \leftarrow 0$;
    while ($index < TableSize$) do
        if $|freq - FREQUENCY\_TABLE(index)| \leq Minimum$ then
            $BestIndex \leftarrow index$
            $Minimum \leftarrow |freq - FREQUENCY\_TABLE(index)|$
        else
            $index \leftarrow index + 1$
        end if
    end while
    return $FREQUENCY\_TABLE(BestIndex)$
end procedure
```

Figure 5: The Interactive Governor algorithm (II—The Choose frequency procedure).

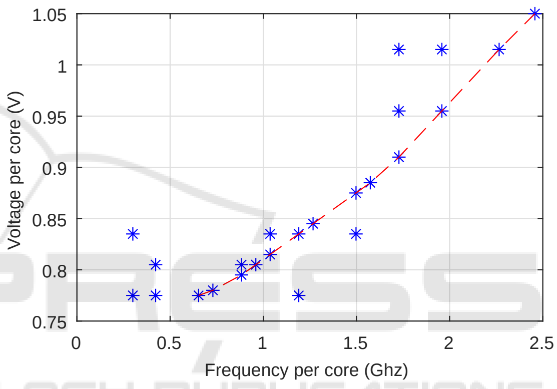

Figure 6: Voltage values drawn against frequency value with the almost linear trend in red.

these two variables. It shows an almost linear trend in the middle where 98% of the values are concentrated, hinting that a fixed voltage value is associated with each frequency value. However, Fig. 6 also shows that some frequencies are linked to multiple voltage values, this is due to the fact that the core needs to increase its voltage before scaling up the frequency, and decrease the frequency before lowering the voltage.

Furthermore, by analyzing the measurements, we find 15 frequency values (plus a zero frequency for a turned-off core) against 14 voltage values, leading to the belief that some frequencies share the same voltage value. Thus, to better investigate the relationship between the frequencies and voltages, a histogram of voltage values for each frequency is constructed. Fig. 7 shows the histogram of voltage values for $f = 300\,\text{MHz}$ indicating that the voltage associated with this frequency value is clearly $V_{300MHz} = 0.775\,\text{V}$. All other voltage values are obtained following the same fashion.

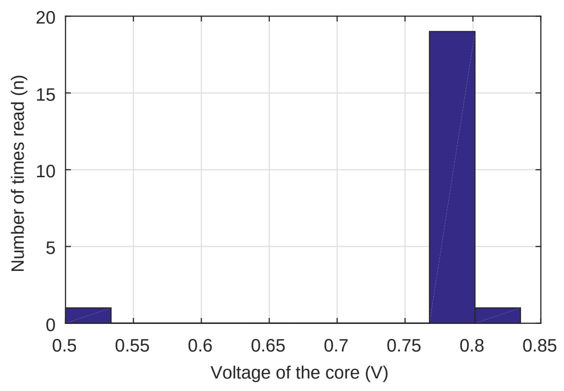

Figure 7: Histogram of voltage values for $f = 300\,\mathrm{MHz}$.

4.3 The Thermal Regulator

Thermal management is a very important task, especially during boot and system startup periods where the temperature can become excessively high. The algorithms used during startup are generally fixed and not accessible to the programmer. However, in the user space, SoC manufacturers generally implement programmable algorithms for programmers to use. In the smartphone we are studying, the SoC manufacturer implemented three types of regulators from which the programmer can choose to manage the temperature of the SoC. The implemented regulators are Proportional Integral Derivative (PID), Single Step, and finally, the one used by the manufacturer of the studied smartphone a *Monitor*.

This algorithm samples the temperature every *sampling_ms*. If the temperature of the core is higher than a predefined *thresholds*, the core is shutdown. Once the temperature of the said core drops below *thresholds_clr*, it can be turned on again.

4.4 Power Model

In the relevant literature, some works, such as the one presented by Wang *et al.*, use battery measurements to estimate the power consumed by applications (Wang et al., 2013). Others track kernel queries to determine power consumption (Pathak et al., 2011). Other techniques, like the one presented in (Kim et al., 2012b; Kim and Chung, 2013; Kim et al., 2015) involve the construction of polynomial models to estimate the power consumption as a function of the frequency and of the load. In this case study, we use battery readings. However, the obtained value of the current supplied by the battery is a constant, and the value of voltage varies in a set of fixed values which results in staircase-like power output signal, and makes any convergence of polynomial and auto-regressive models (ARX, ARMA, ...) impossible. Additionally, in order to minimise the influence of the other components of the smartphone, all communication and secondary peripherals (WiFi, screen, cameras ...) were disabled and assumed to consume a static constant amount of power in that state.

Before starting the modelling process, it is necessary to determine the model inputs. In the case of the CPU, the power consumption is often given by the relation (Adam Kerin, 2013) :

$$P_{CPU} = f \times V^2 \times C \tag{1}$$

where f denotes the frequency, V the voltage and C the electric charge stored in the CPU, which is relatively constant. Thus, the CPU power becomes a function of the frequency and voltage. It was shown in the previous subsection that the voltage is itself a function of frequency. For the GPU, voltage measurements are not available in this case study, thus its frequency is used as input of the model (Kim et al., 2015). The last input is the Memory Occupation Rate (MOR)—the ratio of the occupied memory to the full memory—which will help include the memory power consumption in the model, since it is a part of the SoC.

The model developed to estimate power consumption is a neural network with two layers. A hidden layer whose activation function is a sigmoid, and containing 8 neurons, and an output layer containing a single neuron with a linear transfer function.

4.5 Temperature Model

Heat transfer and temperature modelling have been studied quite extensively in the relevant literature. However, in this work, we will not focus on the mechanics of heat generation and transfer since its aim is to estimate temperature for monitoring and diagnosis purposes. Thus we focused our attention on finding variables affecting it i.e. correlations.

Temperature dynamics are different from those of the other studied variables; for one it does not range in a specific set of values. Furthermore, it does not depend only on the inputs, but also on its own previous values. It is directly correlated with the frequency of the CPU and the GPU. However, as shown in (Hong and Kim, 2010), we note that the correlation in the measurements between the recorded temperature and power consumption is imperceptible, which was confirmed by our own results. Therefore, the considered inputs of the model of the SoC temperature are the frequencies and the MOR.

The temperature readings (Fig. 11) show two main trends for which we should account. The first is the rise (warming) and fall (cooling) of temperature occurring over relatively long periods of time, and the second is the high-frequency small temperature changes. To better represent these dynamics, we chose to use an autoregressive–moving-average (ARMAX) model. ARMAX models use the regression of inputs and previous outputs, along with the moving average to simulate or predict the current output:

$$\begin{aligned} y(k) = {} & P_1 y(k-1) + \ldots + P_n y(k-n) \\ & + Q_1 u(k-1) + \ldots + Q_m u(k-m) \\ & + e(k) + H_1 e(k-1) + \ldots + H_r e(k-r) \end{aligned} \qquad (2)$$

Equation (2) is the linear difference equation of an ARMAX (n,m,r) (orders of the model), with $y(k)$ being the output to compute, u the exogenous (X) variable or system input, and e is the moving average (MA) variable ((Fung et al., 2003)). In the case of this work, temperature T_{SoC} is the output $y(k)$ to be estimated, and the system input is $u(k) = [f_1, \ldots, f_4, f_{GPU}, MOR]$. The parameters P, Q, and H are constants evaluated by iterative search algorithms.

5 EXPERIMENTAL AND MODEL VALIDATION

The experimental results presented in this section are obtained through the application then compared with estimations made by the model. The evolutions of the measured frequencies of one core, and the frequencies estimated by the model are given in Fig. 8. The plots are nearly identical, with a slight delay at the instants of frequency changes. The maximum delay recorded $\tau_f = 0.2\,$s. This result validates the frequency model.

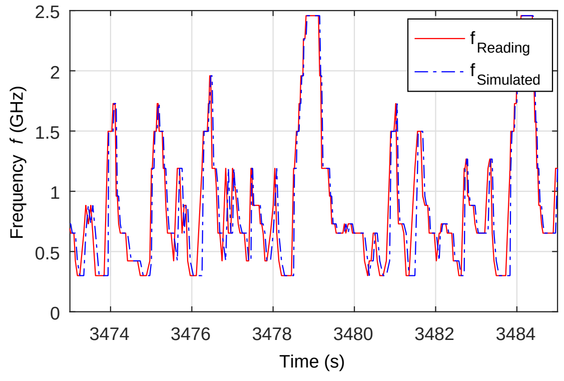

Figure 8: Frequency model estimations vs system readings.

Fig. 9 shows the evolution of the measured voltage of one core, compared to the estimated one. As for frequencies, the plots are again nearly identical, with a slight delay at the instants of voltage changes. The maximum delay recorded $\tau_V = 0.22\,$s. Thus, the voltage model is, also, validated.

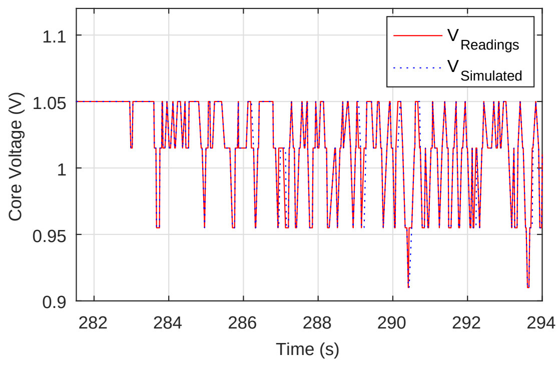

Figure 9: Voltage model estimations vs system readings.

The measured power delivered by the battery compared to the estimated power by the neural network model are given in Fig. 10. This result shows the good accuracy of the model during the slow variations and static phase. However, there is also noticeable noise during the phase of rapid changes, especially between $t = 50\,$s and $t = 60\,$s. The Mean Absolute Error (MAE) recorded is 0.0083 W over 2×10^5 samples, with a maximum error of 6.60%, while the Mean Squared Error (MSE) is 2.3896×10^{-4}. Thus, the power model is validated.

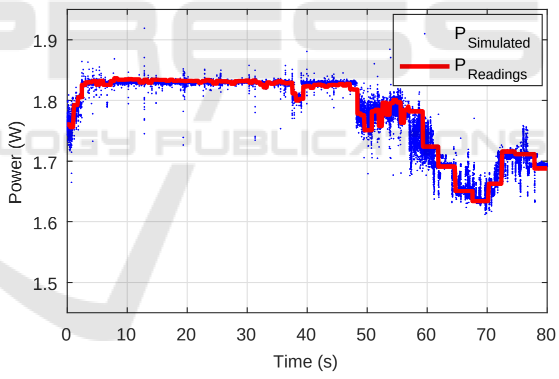

Figure 10: Power model estimations vs system readings.

Fig. 11 shows the evolution of the estimated temperature compared with the measured temperature. The model accurately follows the dynamics of the measured temperature during temperature change (heating and cooling), and also during phases with weak temperature changes. Fig. 11 also shows that the model takes into account the initial conditions of temperature. It has an MAE of 0.9947 °C, with a one time maximum of 8.14%, and an MSE of 1.8584. Hence, the temperature model is also validated.

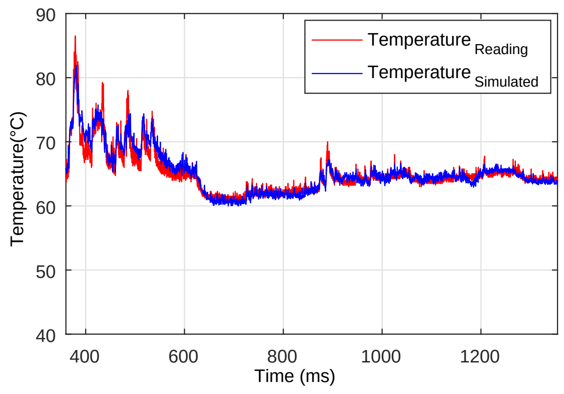

Figure 11: Temperature model estimations vs system readings.

6 CONCLUSION

A data acquisition and estimation systems have been developed for a CPU-GPU embedded chip. Measurements are acquired on the fly for operational state estimation.

The estimation model developed is validated experimentally. The parameter and variable estimation is structured as an interconnected system with variable structure. The modularity of the estimation system is easily adaptable to changes in the system structure and its operation modes.

In future works, the next step after developing the model is to use it to monitor the operating state and drifts in characteristics of the chip.

ACKNOWLEDGEMENT

This paper is a part of the *MMCD* project supported and funded by the BPI, to whom we address our thanks along with the *FUI 19* project partners : INRIA, IRTS, and Nolam ES.

REFERENCES

Adam Kerin (2013). Power vs. Performance Management of the CPU.

Adhinarayanan, V., Subramaniam, B., and Feng, W.-c. (2016). Online Power Estimation of Graphics Processing Units. In *IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, number May, Colombia.

Ardalani, N., Lestourgeon, C., Sankaralingam, K., and Zhu, X. (2015). Cross-architecture performance prediction (XAPP) using CPU code to predict GPU performance. *Proceedings of the 48th International Symposium on Microarchitecture - MICRO-48*, pages 725–737.

Fung, E. H., Wong, Y., Ho, H., and Mignolet, M. P. (2003). Modelling and prediction of machining errors using ARMAX and NARMAX structures. *Applied Mathematical Modelling*, 27(8):611–627.

Hong, S. and Kim, H. (2010). An integrated GPU power and performance model. In *ACM SIGARCH Computer Architecture News*, volume 38 of {*ISCA*} *'10*, page 280, New York, NY, USA. ACM.

Kim, H., Vuduc, R., Baghsorkhi, S., Choi, J., and Hwu, W.-m. (2012a). *Performance Analysis and Tuning for General Purpose Graphics Processing Units (GPGPU)*, volume 7. Morgan & Claypool publishers.

Kim, M. and Chung, S. W. (2013). Accurate GPU power estimation for mobile device power profiling. *Digest of Technical Papers - IEEE International Conference on Consumer Electronics*, pages 183–184.

Kim, M., Kong, J., and Chung, S. W. (2012b). Enhancing online power estimation accuracy for smartphones. *IEEE Transactions on Consumer Electronics*, 58(2):333–339.

Kim, Y. G., Kim, M., Kim, J. M., Sung, M., and Chung, S. W. (2015). A novel GPU power model for accurate smartphone power breakdown. *ETRI Journal*, 37(1):157–164.

Leng, J., Hetherington, T., ElTantawy, A., Gilani, S., Kim, N. S., Aamodt, T. M., and Reddi, V. J. (2013). GPUWattch: Enabling Energy Optimizations in GPGPUs. *Proceedings of the 40th Annual International Symposium on Computer Architecture - ISCA '13*, 41:487.

Meng, J. and Skadron, K. (2011). A performance study for iterative stencil loops on GPUs with ghost zone optimizations. *International Journal of Parallel Programming*, 39(1):115–142.

Minyong Kim, Joonho Kong, and Sung Woo Chung (2012). An online power estimation technique for multi-core smartphones with advanced display components. In *2012 IEEE International Conference on Consumer Electronics (ICCE)*, pages 666–667. IEEE.

M'Sirdi, S., Godard, W., and Pantel, M. (2016). A Multi-Core Interference-Aware Schedulability Test for IMA Systems, as a Guide for SW/HW Integration. In *8th European Congress on Embedded Real Time Software and Systems (ERTS 2016)*, TOULOUSE, France.

Pathak, A., Hu, Y. C., Zhang, M., Bahl, P., and Wang, Y.-M. (2011). Fine-Grained Power Modeling for Smartphones Using System Call Tracing. *Proceedings of the sixth conference on Computer systems EuroSys 11*, page 153.

Samsung (2016). Samsung Opensource Release Center.

Wang, C., Yan, F., Guo, Y., and Chen, X. (2013). Power estimation for mobile applications with profile-driven battery traces. *Int Symp on Low Power Electronics and Design*, pages 120–125.

Williams, S., Waterman, A., and Patterson, D. (2009). Roofline: An Insight Visual Performance Model for Multicore Architectures. *Communications of the ACM*, 52(4):65.

Zhang, L., Tiwana, B., Qian, Z., Wang, Z., Dick, R. P., Mao, Z. M., and Yang, L. (2010). Accurate online power estimation and automatic battery behavior based power model generation for smartphones. In *IEEE/ACM/IFIP int conf Hardware/software codesign and system synthesis - CODES/ISSS '10*, page 105, New York, New York, USA. ACM Press.