# A Generic Processor Temperature Estimation Method

Baver Ozceylan*[1], Boudewijn R. Haverkort[2], Maurits de Graaf[3], Marco E. T. Gerards[1]

[1] University of Twente, Enschede, the Netherlands
[2] Tilburg University, Tilburg, the Netherlands
[3] Thales Nederland B.V., Huizen, the Netherlands

* Corresponding Author: b.ozceylan@utwente.nl

*Abstract*

*Most modern mobile embedded devices have the ability to increase their computational power typically at the cost of increased heat dissipation. This may result in temperatures above the design limit, especially if active cooling is inapplicable. Thus, it is necessary to consider processor temperature while scheduling tasks. This means estimating the change in temperature due to changed workload is crucial for high performance mobile embedded devices. To address this challenge, we first introduce a model to estimate the temperature and classify the system dependent model parameters. Then, to determine these parameters, we develop a new method, which can be applied on any mobile embedded device. The only requirement for our new method is learning the device characteristics by processing a certain task while recording the temperature with built-in sensors. Our results show that our method can achieve high accuracy within a short testing period.*

## 1 Introduction

Nowadays, mobile embedded devices are used in many areas and the processing power requirements of applications running on these devices have grown significantly over the last few years. To satisfy this demand, the majority of mobile devices embody multiprocessor system-on-chips (SoCs), which integrate high-performance CPUs and accelerators, such as GPUs and audio processors. However, the downside of having such a high computational power is that it can cause high heat dissipation so that the chip temperature can elevate above the design limit. This is harmful since such a high temperature damages electronic components and decreases device lifetime. Moreover, a high temperature causes an increase in leakage power [1], which reduces energy efficiency. Although an active cooling mechanism, e.g., a fan, can be used to avoid overheating, some devices, such as smart-phones and military equipment, do not allow active cooling due to additional energy consumption, spatial or electromechanical limitations. In such cases, passive cooling mechanisms, based on throttling (slowing down) the processor, must be used. The most commonly used methods for throttling are *idle time scheduling*, which determines the percentage of active resource, and *dynamic voltage and frequency scaling* (DVFS), which controls the operating frequency of active resources [2].The adapted policies for throttling in modern mobile devices are mostly reactive techniques, which throttle the CPU only if the temperature is above the critical point. A reactive technique only considers the current situation. Hence, it allows maximum CPU utilization to provide a high QoS if the temperature is below the critical point. However, if the processing time is long enough such that the temperature reaches the critical point, it triggers the throttling mechanism and leads to an unplanned performance loss. This issue may also cause unnecessarily high power consumption if an application does not have strict QoS requirements because, as stated in [1], an increase in the CPU temperature increases the leakage power, which is the static power consumed by the CPU, i.e., independent of the CPU activities. Therefore, to achieve an optimal performance, a proactive approach needs to be considered, taking account the future. A proactive technique allows QoS-centric thermal management, i.e., dynamic control of the heat dissipation, in order to achieve sustainable performance. In [3], the authors run a racing game application under reactive and proactive policies to show the difference between these two techniques. According to their results, the reactive policy achieves high QoS initially. However, thermal throttling causes a significant QoS reduction over time since the temperature of the device quickly reaches the critical value. On the other hand, the proactive method uses just enough resources to meet the QoS requirement, which means it does not utilize more resources than necessary. Thus, it can achieve a longer sustainable QoS. Additionally, it can be observed from their example that the proactive policy makes the device run at lower temperature values on average, which implies a lower leakage power consumption.

The challenge is that proactive techniques need a model of the dynamic thermal behaviour of the system in order to satisfy QoS requirements without exceeding temperature limits [4,5]. In other words, thermal modelling and temperature estimation is crucial for taking measures before the temperature reaches critical levels. The authors of [4] and [6] propose models that depend on precalculated system parameters. However, this approach is not suitable for every system. For example, in [4], a furnace is used to characterise the leakage power. Although this method can achieve high accuracy, it is not applicable for systems that are already in use. On the other hand, in [6], the proposed method to determine the model parameters needs to measure ambient temperature periodically. However, not every device has the ability to sense the ambient temperature. In contrast, *we propose a new method for determining the dynamic temperature behaviour of an arbitrary system using only the target processor's built-in temperature sensor.*

This paper addresses the problem of estimating processor temperatures in a generic setting. In Section 2, we introduce a model to estimate the temperature. The model is created by analysing our experimental results and exercising the models in the literature. In Section 3, we define and solve an optimization problem to find optimal values for the system dependent model parameters. Then, we propose a method by using this solution to determine the parameters. In Section 4, we show experimental results to validate our generic processor temperature estimation method. Section 5 concludes the paper.

This paper makes the following main contributions:

- A general modelling framework for dynamic temperature estimation is presented and applied to Intel i7 and ARM Cortex-A53 processors.
- We propose a generic method that uses only the built-in temperature sensor of the target processor in order to determine the system dependent model parameters and prove its optimality.
- The proposed dynamic temperature estimation method achieves accurate results while requiring little time for determining the model parameters.

## 2 Thermal/Power Model

The temperature of a processor exponentially increases (or decreases) while executing a job (or idling). The change in temperature is determined by the processor's heat dissipation and it can be described with the following well-known relation between heat transfer and electrical phenomena [7]:

$$C\frac{dT(t)}{dt} = -GT(t) + P(t), \quad (1)$$

where $C$ and $G$ are thermal conductance and capacitance respectively. $P(t)$ and $T(t)$ represents respectively the power dissipation and the temperature at time $t$. Considering the time period wherein the power dissipation is constant, the temperature in this interval can be obtained by solving the first-order differential equation (1) as:

$$T(t) = T_0 + (T_\mathrm{F} - T_0)\left(1 - e^{\frac{-t}{\tau}}\right), \quad (2)$$

where $T_0$ and $T_\mathrm{F}$ are the temperature at $t = 0$ and final temperature, respectively. The $e^{-t/\tau}$ term leads to the exponential rise or fall that converges to $T_\mathrm{F}$. The power dissipation in the corresponding time period, which is constant, determines the $T_\mathrm{F}$ and the $C/G$ ratio determines τ, which is the thermal time constant.

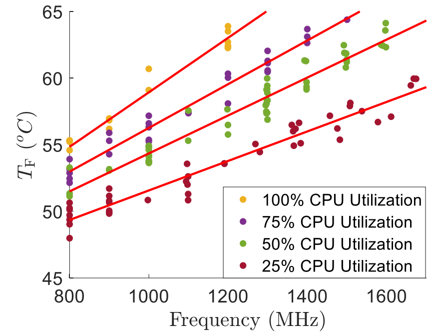



*Figure 1: Measurements taken from the Intel i7 processor*

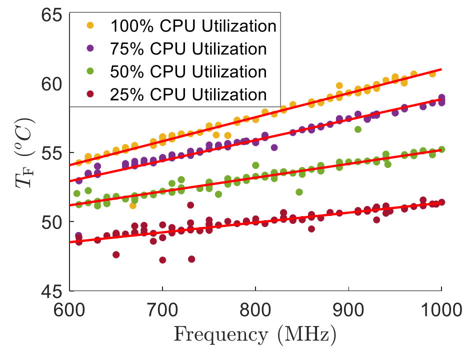



*Figure 2: Measurements taken from the ARM Cortex-A53 processor*

*Table 1: The coefficient matrices for the Intel and ARM systems obtained by minimizing RMSE.*

| **Processor** | **The coefficient matrix** | | | | **RMSE** |
|---|---|---|---|---|---|
| | $\gamma_{11}$ | $\gamma_{12}$ | $\gamma_{21}$ | $\gamma_{22}$ | |
| Intel i7 | 41.1 | 0.71 | 15.2 | 17.9 | 0.8570 |
| ARM Cortex-A53 | 45.1 | 1.05 | 3.36 | 14.1 | 0.6159 |

The frequency, *f*, and utilization, *u*, of the processor are jointly called the *performance parameters*, which influence the power dissipation. We have conducted several experiments to assess the relation between the performance parameters and $T_\mathrm{F}$. For these experiments, Intel i7 and ARM Cortex-A53 processors have been used with the same workload for a long period of time and the final temperature values have been recorded under different performance parameters. The results are shown in Fig. 1 and Fig. 2. In these figures, there are four different colours that correspond to four different CPU utilizations. For each of these, the system has been tested under various operating frequencies. The figures indicate that the relation between the frequency $f$ and the measured $T_\mathrm{F}$ is linear, which is indicated by regression lines. Therefore, $T_\mathrm{F}$ can be expressed as $T_\mathrm{F}(f) = \boldsymbol{\gamma}^T\boldsymbol{f}$, where $\boldsymbol{\gamma}$ is the coefficient vector and $\boldsymbol{f} = [1, f]^T$. This coefficient vector also depends on the CPU utilization, which, as it turns out, can be expressed as $\boldsymbol{\gamma} = \boldsymbol{\Gamma}^T\boldsymbol{u}$, where $\boldsymbol{\Gamma}$ is the coefficient matrix:

$$\boldsymbol{\Gamma} = \begin{bmatrix} \gamma_{11} & \gamma_{12} \\ \gamma_{21} & \gamma_{22} \end{bmatrix},$$

and $\boldsymbol{u} = [1, u]^T$. As a result, $T_F$ can be rewritten as $T_F(f,u) = \boldsymbol{u}^T \boldsymbol{\Gamma} \boldsymbol{f}$. Although, in this paper, we consider the same clock frequency for all different cores and the utilization as the total CPU utilization, this analysis can be easily extended using the same vector notation above.

In order to obtain the two-by-two coefficient matrices for the Intel and ARM systems, we have minimized the *root-mean-square-error* (RMSE) between the measured and estimated $T_F$ values by using the normalized CPU frequency and utilization values for $f$ and $u$, respectively. The calculated values are shown in Table 1. The results show that the $\gamma_{12}$ element in the coefficient matrix has a minor effect since $u$ is between 0 and 1 and $\gamma_{12}$ is very low as compared to the range of $T_F$. If we fix $\gamma_{12}$ to zero as an approximation, the RMSE for the Intel and ARM systems slightly increase to 0.8618 and 0.6167 from the values in Table 1, respectively. Since the increase in the estimation error is very low compared to the range of $T_F$, we can ignore the effect of the $\gamma_{12}$ coefficient and fix it to zero. Therefore, we write $T_F(f,u) = \boldsymbol{u}^T \boldsymbol{\Gamma} \boldsymbol{f}$ as:

$$T_F(f,u) = \gamma_{11} + \gamma_{21} f + \gamma_{22} u f. \qquad (3)$$

According to (1) and (2), the expression in (3) is also the power model. In fact, this obtained model corresponds to the power models proposed in the literature [8]. The term $\gamma_{11} + \gamma_{21} f$ in (3) corresponds to the leakage power component, which is the static power dissipation caused by the leakage current, i.e., independent of the CPU activities. On the other hand, the term $\gamma_{22} u f$ corresponds to the dynamic power component, which depends on switching activities in the processor.

## 3 Parameter Estimation

The change in the temperature can be estimated by using the models defined in (2) and (3). The parameters in the models are $\gamma_{11}$, $\gamma_{21}$, $\gamma_{22}$ and $\tau$. These four parameters are system dependent, which means they may differ from one platform to another, even when these platforms use the same processor. Additionally, the $T_0$ term is the temperature while the processor is in idle state. Thus, $T_0$ can be found by recording the temperature while the system is idle. The aim, in this section, is to develop a generic method to determine the system dependent model parameters for a given system. We do so by collecting temperature data, while processing a known task under a chosen $\langle f, u \rangle$ performance parameter pair. This leads to a vector of $N$ observations, $\boldsymbol{Y} = [Y_1, Y_2, \ldots, Y_N]$, corresponding to a vector of points in time, $\boldsymbol{t} = [t_1, t_2, \ldots, t_N]$, respectively. At time $t_i$, the measured temperature value can be expressed as $Y_i = T_i + v_i$, where $T_i$ is the real temperature value at time $t_i$ and $v_i$ is the noise component. We assume that the noise components are independent from each other, identically distributed and the distribution is Gaussian. Thus, the observation data can be rewritten as $Y_i \sim N(T_i, \sigma_v^2)$ and the probability density function of $\boldsymbol{Y}$ can be expressed as follows:

$$\begin{aligned} P(\boldsymbol{Y}|\Theta, \boldsymbol{t}) &= P(Y_1|\Theta, t_1) P(Y_2|\Theta, t_2) .. P(Y_N|\Theta, t_N), \\ &= \prod_{i=1}^{N} \frac{exp\{-(Y_i - T_i)^2 / 2\sigma_v^2\}}{(2\pi\sigma_v^2)^{1/2}}, \end{aligned} \qquad (4)$$

where $\Theta$ is the set of model parameters. The expression in (4) is the likelihood function. The optimal values for the model parameters can be found by maximizing the likelihood function, which means finding the parameter values that make the observation data most probable. Maximizing the likelihood function is the same as maximizing the log-likelihood function, which can be expressed as follows:

$$\ell(\Theta; \boldsymbol{Y}|\boldsymbol{t}) = -\sum_{i=1}^{N} \frac{\left(Y_i - T_\Theta(t_i)\right)^2}{2\sigma_v^2} - \frac{N}{2} \log(2\pi\sigma_v^2), \qquad (5)$$

where $T_\Theta(t)$ is the estimation function that corresponds to the model defined in (2) and (3) by using $\Theta$ as the model parameters. If the parameter set $\widehat{\Theta}$ maximizes $\ell(\Theta; \boldsymbol{Y}|\boldsymbol{t})$, the model $T_{\widehat{\Theta}}(t)$ is optimal. Therefore, the optimization problem can be defined as follows:

$$\begin{aligned} \widehat{\Theta} &= \arg\max_{\Theta} \ell(\Theta; \boldsymbol{Y}|\boldsymbol{t}) \\ &= \arg\min_{\Theta} \sum_{i=1}^{N} \left(Y_i - T_\Theta(t_i)\right)^2. \end{aligned} \qquad (6)$$

First, we consider the case where the performance parameters $f$ and $u$ are constant for all $t_i$ ($i \in [1, N]$) in order to find a solution for the problem in (6). In this case, the model parameters are $\Theta = \langle T_F, \tau \rangle$ since (3) states that $T_F$ is constant when $f$ and $u$ are constant. In this case, the second-order derivative of the expression in (6) with respect to $T_F$ is always positive. Thus, the expression is convex with respect to $T_F$, which means that the solution can be computed in terms of $\tau$ by finding the point that makes the first-order derivative of the expression in (6) with respect to $T_F$ zero:

$$0 = \frac{\partial}{\partial T_F} \left[ \sum_{i=1}^{N} \left(Y_i - T_\Theta(t_i)\right)^2 \right]_{T_F = \hat{T}_F},$$

which can be rewritten as follows:

$$\hat{T}_F = \frac{\sum_{i=1}^{N} \left(Y_i - T_0 e^{\frac{-t_i}{\tau}}\right)\left(1 - e^{\frac{-t_i}{\tau}}\right)}{\sum_{i=1}^{N} \left(1 - e^{\frac{-t_i}{\tau}}\right)^2}, \qquad (7)$$

where $\hat{T}_F$ denotes the optimal value for the $T_F$ parameter. To simplify the expression in (7), we define the data set $\boldsymbol{Y}' = [Y_1', Y_2', \dots, Y_N']$, where $Y_i' = Y_i - T_0$ for all $i \in [1, N]$. After that, $\hat{T}_F$ can be expressed as follows:

$$\hat{T}_F(\tau, \boldsymbol{Y}') = T_0 + \frac{n(\tau, \boldsymbol{Y}')}{d(\tau, \boldsymbol{Y}')}, \qquad (8)$$

where $n(\tau, \boldsymbol{Y}')$ and $d(\tau, \boldsymbol{Y}')$ refer to the numerator and denominator, which are defined as follows:

$$n(\tau, \boldsymbol{Y}') := \sum_{i=1}^{N} Y_i' \left(1 - e^{\frac{-t_i}{\tau}}\right),$$
$$d(\tau, \boldsymbol{Y}') := \sum_{i=1}^{N} \left(1 - e^{\frac{-t_i}{\tau}}\right)^2.$$

Equation (8), in fact, finds the value that makes the weighted average of $\boldsymbol{Y}'$ equal to the weighted average of the estimated temperature values, where the weights are $1 - e^{-t_i/\tau}$ for all $i \in [1, N]$. Summarizing, when $f$ and $u$ are constant and $\tau$ is given, the solution for the problem in (6) is the $T_F$ value obtained by (8) with the given $\tau$. Thus, the model in (1) can be rewritten as follows:

$$T_\tau(t) = T_0 + \frac{n(\tau, \boldsymbol{Y}')}{d(\tau, \boldsymbol{Y}')}\left(1 - e^{\frac{-t}{\tau}}\right). \qquad (9)$$

According to the problem in (6), the optimal value for $\tau$, which we denote by $\hat{\tau}$, should satisfy the following condition:

$$0 = \frac{\partial}{\partial \tau}\left[\sum_{i=1}^{N}\left((Y_i' + T_0) - T_\tau(t_i)\right)^2\right]_{\tau=\hat{\tau}},$$

which can be rewritten as follows:

$$0 = \frac{n(\hat{\tau}, \boldsymbol{Y}')}{d^2(\hat{\tau}, \boldsymbol{Y}')}\big(2n'(\hat{\tau}, \boldsymbol{Y}')d(\hat{\tau}, \boldsymbol{Y}') - n(\hat{\tau}, \boldsymbol{Y}')d'(\hat{\tau}, \boldsymbol{Y}')\big), \qquad (10)$$

where the prime symbol represents the first-order derivative of a function with respect to $\tau$. Considering that $n(\tau, \boldsymbol{Y}')$ and $d(\tau, \boldsymbol{Y}')$ cannot be zero, we denote the right-hand side of the expression in (10) by $f_{\boldsymbol{Y}}(\tau)$ as:

$$f_{\boldsymbol{Y}}(\tau) = 2n'(\hat{\tau}, \boldsymbol{Y}')d(\hat{\tau}, \boldsymbol{Y}') - n(\hat{\tau}, \boldsymbol{Y}')d'(\hat{\tau}, \boldsymbol{Y}'), \qquad (11)$$

then any $\tau$ such that $f_{\boldsymbol{Y}}(\tau) = 0$ satisfies the condition in (10) and is optimal. In Fig. 3, the $f_{\boldsymbol{Y}}(\tau)$ curves are drawn for six different data sets, which are generated with the $T_F$ values indicated in the figure. For the results in Fig. 3, $f_{\boldsymbol{Y}}(\tau)$ is decreasing and convex for all $\tau \in [0, \hat{\tau}]$, where $\hat{\tau}$ is the solution and makes the $f_{\boldsymbol{Y}}(\tau)$ function equal to zero. Thus, we can use Newton's method to search the optimal $\tau$ starting from $\tau = 1$ since this method guarantees the operation within $[0, \hat{\tau}]$ due to convexity of the region. Algorithm 1 aims to find the $\tau$ value such that $f_{\boldsymbol{Y}}(\tau) = 0$, which is $\hat{\tau}$, by using Newton's method. The algorithm searches $\hat{\tau}$ by increasing $\tau$ iteratively starting from $\tau = 1$. In each iteration, the amount of increment is $-f_{\boldsymbol{Y}}(\tau)/f'_{\boldsymbol{Y}}(\tau)$, where $f'_{\boldsymbol{Y}}(\tau)$ is the first-order derivative of $f_{\boldsymbol{Y}}(\tau)$ with respect to $\tau$. The algorithm computes $f'_{\boldsymbol{Y}}(\tau)$ with the equation expressed in Line 3. The algorithm ends when $f_{\boldsymbol{Y}}(\tau)$ reaches a value close to zero, i.e., $\epsilon \geq f_{\boldsymbol{Y}}(\tau) \geq 0$, where $\epsilon$ determines the accuracy of the algorithm.

**Algorithm 1:** Finding $\hat{\tau}$

**Input:** $\boldsymbol{Y}, T_0$
**Output:** $\hat{\tau}$
**1:** $\boldsymbol{Y}' \leftarrow \boldsymbol{Y} - T_0$;
$\hat{\tau} \leftarrow 1$;
$f_{\boldsymbol{Y}}(\tau) \leftarrow 2n'(\hat{\tau}, \boldsymbol{Y}')d(\hat{\tau}, \boldsymbol{Y}') - n(\hat{\tau}, \boldsymbol{Y}')d'(\hat{\tau}, \boldsymbol{Y}')$;
**2:** **while** $f_{\boldsymbol{Y}}(\tau) > \epsilon$ **do**
**3:** $f'_{\boldsymbol{Y}}(\tau) \leftarrow 2n''(\hat{\tau}, \boldsymbol{Y}')d(\hat{\tau}, \boldsymbol{Y}') + n'(\hat{\tau}, \boldsymbol{Y}')d'(\hat{\tau}, \boldsymbol{Y}') - n(\hat{\tau}, \boldsymbol{Y}')d''(\hat{\tau}, \boldsymbol{Y}')$;
**4:** $\hat{\tau} \leftarrow \hat{\tau} - \frac{f_{\boldsymbol{Y}}(\tau)}{f'_{\boldsymbol{Y}}(\tau)}$;
**5:** $f_{\boldsymbol{Y}}(\tau) \leftarrow 2n'(\hat{\tau}, \boldsymbol{Y}')d(\hat{\tau}, \boldsymbol{Y}') - n(\hat{\tau}, \boldsymbol{Y}')d'(\hat{\tau}, \boldsymbol{Y}')$;
**6:** **End**

Algorithm 1 needs the first-order and second-order derivatives of $n(\tau, \boldsymbol{Y}')$ and $d(\tau, \boldsymbol{Y}')$ with respect to $\tau$ in order to compute $f_{\boldsymbol{Y}}(\tau)$ and $f'_{\boldsymbol{Y}}(\tau)$. These are given by:

$$n'(\tau, \boldsymbol{Y}') := \frac{-1}{\tau^2}\sum_{i=1}^{N} Y_i' t_i e^{\frac{-t_i}{\tau}},$$
$$d'(\tau, \boldsymbol{Y}') := \frac{-2}{\tau^2}\sum_{i=1}^{N}\left(1 - e^{\frac{-t_i}{\tau}}\right) t_i e^{\frac{-t_i}{\tau}},$$

$$n''(\tau, \boldsymbol{Y}') := \frac{-2}{\tau} n'(\tau, \boldsymbol{Y}') - \frac{1}{\tau^4}\sum_{i=1}^{N} Y_i' t_i^2 e^{\frac{-t_i}{\tau}},$$
$$d''(\tau, \boldsymbol{Y}') := \frac{-2}{\tau} d'(\tau, \boldsymbol{Y}') - \frac{2}{\tau^4}\sum_{i=1}^{N}(1 - 2e^{\frac{-t_i}{\tau}})\, t_i^2 e^{\frac{-t_i}{\tau}}.$$

After finding $\hat{\tau}$, we want to determine the $\gamma_{11}$, $\gamma_{21}$ and $\gamma_{22}$ parameters. We have shown that when the performance parameters, which are $f$ and $u$, are constant, (8) gives the $T_F$ that solves the problem in (6) as $\hat{T}_F(\hat{\tau}, \boldsymbol{Y}')$. The problem is, now, finding $\gamma_{11}$, $\gamma_{21}$ and $\gamma_{22}$ that makes $T_{\mathrm{F}}(f,u)$ in (3) equal to $\hat{T}_F(\hat{\tau}, \boldsymbol{Y}')$ in (8) as follows:

$$\gamma_{11} + \gamma_{21} f + \gamma_{22} u f = T_0 + \frac{n(\hat{\tau}, \boldsymbol{Y}')}{d(\hat{\tau}, \boldsymbol{Y}')}. \qquad (12)$$

The equality in (12) indicates a linear problem. Thus, we need three different $\langle f, u, \boldsymbol{Y} \rangle$ sets to find a unique solution for the problem since there are three unknowns, $\boldsymbol{\gamma} = [\gamma_{11}, \gamma_{21}, \gamma_{22}]^T$. Considering three data sets, which are denoted by $\boldsymbol{Y}^{(1)}$, $\boldsymbol{Y}^{(2)}$ and $\boldsymbol{Y}^{(3)}$, the solution can be found by solving:

$$\begin{bmatrix} \hat{T}_F(\hat{\tau}, \boldsymbol{Y}'^{(1)}) \\ \hat{T}_F(\hat{\tau}, \boldsymbol{Y}'^{(2)}) \\ \hat{T}_F(\hat{\tau}, \boldsymbol{Y}'^{(3)}) \end{bmatrix} = \begin{bmatrix} 1 & f^{(1)} & f^{(1)} u^{(1)} \\ 1 & f^{(2)} & f^{(2)} u^{(2)} \\ 1 & f^{(3)} & f^{(3)} u^{(3)} \end{bmatrix} \boldsymbol{\gamma}, \qquad (13)$$

where $f^{(i)}$ and $u^{(i)}$ refer to the normalized clock frequency and CPU utilization corresponds to $\boldsymbol{Y}^{(i)}$.

At this point, we have Algorithm 1 to find the optimal $\tau$ parameter. As it can be observed in Fig. 3, the data set obtained for higher $T_{\mathrm{F}}$ values results in a faster decay of the $f_{\boldsymbol{Y}}(\tau)$ function, which increases the accuracy of the algorithm and decreases the number of iterations. For this reason, the first experiment to obtain the $\boldsymbol{Y}^{(1)}$ data set is under the highest clock frequency and the highest CPU utilization, i.e., $f = 1$ and $u = 1$. After finding $\hat{\tau}$, we solve (13) to obtain $\gamma_{11}$, $\gamma_{21}$ and $\gamma_{22}$. To do so, we need to conduct two more experiments with different $\langle f, u \rangle$ pairs. To simplify the operation in (13) the performance parameters are chosen as $\langle 1, u_L \rangle$ and $\langle f_L, 1 \rangle$ for the second and third experiments, which means for the $\boldsymbol{Y}^{(2)}$ and $\boldsymbol{Y}^{(3)}$ data sets, where $f_L$ and $u_L$ refers to low frequency and utilization values respectively. By solving (13) with the data sets $\boldsymbol{Y}^{(1)}$, $\boldsymbol{Y}^{(2)}$ and $\boldsymbol{Y}^{(3)}$ obtained with $\langle f, u \rangle$ performance parameter pairs as $\langle 1,1 \rangle$, $\langle 1, u_L \rangle$ and $\langle f_L, 1 \rangle$, $\gamma_{11}$, $\gamma_{21}$ and $\gamma_{22}$ parameters can be expressed as follows:

$$\gamma_{11} = T_0 + \frac{1}{1 - f_L} \left( \frac{n(\hat{\tau}, \boldsymbol{Y}'^{(1)})}{d(\hat{\tau}, \boldsymbol{Y}'^{(2)})} - f_L \frac{n(\hat{\tau}, \boldsymbol{Y}'^{(2)})}{d(\hat{\tau}, \boldsymbol{Y}'^{(2)})} \right), \qquad (15a)$$

$$\gamma_{21} = \frac{1}{1 - u_L} \left( \frac{n(\hat{\tau}, \boldsymbol{Y}'^{(2)})}{d(\hat{\tau}, \boldsymbol{Y}'^{(2)})} - u_L \frac{n(\hat{\tau}, \boldsymbol{Y}'^{(3)})}{d(\hat{\tau}, \boldsymbol{Y}'^{(3)})} \right) - \frac{1}{1 - f_L} \left( \frac{n(\hat{\tau}, \boldsymbol{Y}'^{(1)})}{d(\hat{\tau}, \boldsymbol{Y}'^{(1)})} - f_L \frac{n(\hat{\tau}, \boldsymbol{Y}'^{(2)})}{d(\hat{\tau}, \boldsymbol{Y}'^{(2)})} \right), \qquad (15b)$$

$$\gamma_{22} = \frac{1}{1 - u_L} \left( \frac{n(\hat{\tau}, \boldsymbol{Y}'^{(3)})}{d(\hat{\tau}, \boldsymbol{Y}'^{(3)})} - \frac{n(\hat{\tau}, \boldsymbol{Y}'^{(2)})}{d(\hat{\tau}, \boldsymbol{Y}'^{(2)})} \right), \qquad (15c)$$

An important observation is that $T_0$ only appears in (15a), which means it only affects the $\gamma_{11}$ parameter. Since $T_0$ is the only term in the model that depends on the environment, we can argue that environmental changes only affect the $\gamma_{11}$ parameter. Therefore, to make the system adaptable to changing environment, we only need to update the $\gamma_{11}$ parameter.

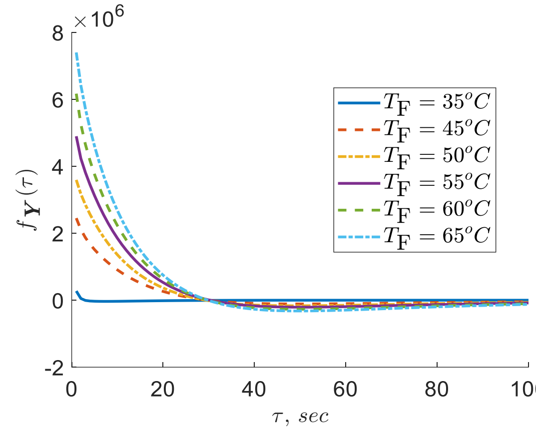



*Figure 3: $\boldsymbol{f_Y(\tau)}$ with different data sets*

As a result, our method has four steps:

1) Measure the temperature while the system is idle at various time instances and determine $T_0$ by averaging the measurements.
2) Execute a task and record the temperature data over time by using built-in temperature sensor in the processor for different $\langle f, u \rangle$ performance parameter pairs as $\langle 1,1 \rangle$, $\langle 1, u_L \rangle$ and $\langle f_L, 1 \rangle$.
3) Calculate the optimal value for $\tau$ parameter by applying Algorithm 1 to the data set obtained with the $\langle 1,1 \rangle$ parameter pair.
4) Find the optimal value for $\gamma_{11}$, $\gamma_{21}$ and $\gamma_{22}$ parameters by using (15a), (15b) and (15c).

Then, the change in temperature can be estimated in time by using (2) and (3) with the found parameter values.

## 4 Model Validation

We have validated the proposed temperature estimation method on Intel i7 and ARM Cortex-A53 processors. First, we have applied the method of Section 3 to determine the model parameters for each processor. Then, we have conducted 500 experiments with randomly chosen operating frequency. During each experiment, the processor is loaded with a task that can only use a certain amount of CPU resource, i.e., the CPU utilization, which is also chosen randomly. To compare the real CPU temperature and estimation results, we have recorded the temperature during the experiments by using the built-in sensors in the processors. The proposed parameter estimation method requires three data sets. The number of data points in these data sets affects the accuracy of the model. Fig. 4 shows the RMSE results between the estimated and measured values with respect to the number of data points used in the proposed method. The results show that the RMSE can be reduced to $0.9^o C$ for the Intel and ARM processors with forty data points for each of the three data sets. In this example, the time between each measurement is three seconds. Hence, the proposed model requires only two minutes for each

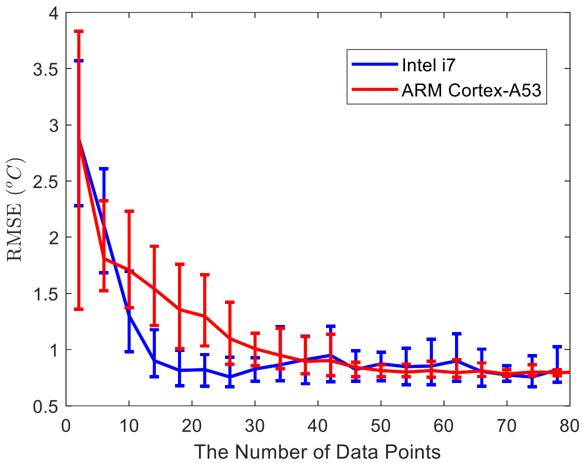



*Figure 4: The RMSE results for the Intel and ARM processors with respect to the number of data points*

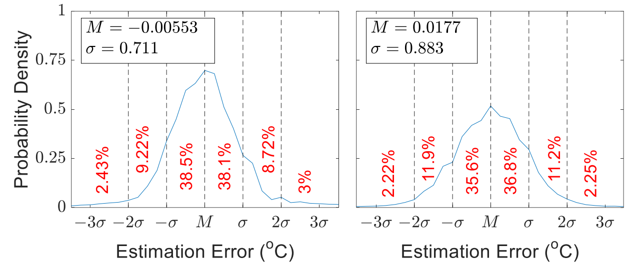
experiment to collect required data so that the RMSE can be reduced below $0.9°C$.

Fig. 5a and Fig. 5b show the probability density functions of the estimation error for the Intel and ARM processors, respectively, when the RMSE is reduced to $0.9°C$. The figures show that, with a confidence of 95%, the estimation error is below 1.4°C for the Intel processor and it is below 1.8°C for the ARM processor.

## 5 Conclusion

Overheating may decrease device lifetime by damaging vulnerable electronic components. Moreover, a processor consumes more energy while executing a task in higher temperatures. In some cases, active cooling mechanisms are not available and passive cooling mechanisms, based on throttling (slowing down) the processor, must be used. This paper proposes a new method to determine the dynamic temperature behavior of an arbitrary system based on a limited number of measurements, that can be quickly taken. This method can be used for scheduling purposes by predicting the temperature based on processor's operating frequency and utilization. This leads to a pro-active approach to cope with overheating problem and maximize performance and minimize energy consumption. The method only requires the data obtained from built-in temperature sensors while conducting three simple experiments, which means the experiments can be conducted without any extra hardware, so that the method is suitable for any device; we also show that the duration of these experiments does not have to be long and that the dynamic temperature behavior can be estimated with high accuracy. We have implemented and tested our method on processors of different architectures with good results.

Future work will extend the model for the systems that are able to run processor cores with different clock frequencies at




*(a) Intel i7* *(b) ARM Cortex-A53*

*Figure 5: The probability density functions of the estimation errors*

the same time. Furthermore, a scheduling method will be developed based on proposed model in this paper.

## Literature


[1] N. S. Kim, T. Austin, D. Blaauw, T. Mudge, J. S. Hu, M. J. Irwin, M. Kandemir, and V. Narayanan, "Leakage current: Moore," *IEEE Computer*, vol. 36, no. 12, Dec., pp 68-75, 2003.

[2] G. Singla, G. Kaur, A. K. Unver, and U. Y. Ogras, "Predictive dynamic thermal and power management for heterogeneous mobile platforms," In Proc. EDA Consortium Design, Automation & Test in Europe Conference & Exhibition '15, 2015, pp. 960-965.

[3] O. Sahin, and A. K. Coskun, "QScale: Thermally-efficient QoS management on heterogeneous mobile platforms," In Proc IEEE/ACM International Conference on Computer-Aided Design '16, 2016, pp. 1-8.

[4] G. Bhat, G. Singla, A. K. Unver, and U. Y. Ogras, "Algorithmic optimization of thermal and power management for heterogeneous mobile platforms," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 26, no. 3, March, pp. 544-557, 2018.

[5] O. Sahin, L. Thiele, and A. K. Coskun, "MAESTRO: Autonomous QoS management for mobile applications under thermal constraints," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 38, no. 8, Feb., pp. 1557-1570, 2019.

[6] Y. Lee, H. S. Chwa, K. G. Shin, and S. Wang, "Thermal-aware resource management for embedded real-time systems," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 11, Nov., pp 2857-2868, 2008.

[7] K. Skadron, M. R. Stan, K. Sankaranarayanan, W. Huang, S. Velusamy, and D. Tarjan, "Temperature-aware microarchitecture: Modeling and implementation," *ACM Transactions on Architecture and Code Optimization*, vol. 1, no. 1, March, pp. 94-125, 2004.

[8] H. Sultan, G. Ananthanarayanan, and S. R. Sarangi. "Processor power estimation techniques: Survey." International Journal of High Performance Systems Architecture, vol. 5, no. 2, May, pp. 93-114, 2014.