A Calibration Based Thermal Modeling Technique for Complex Multicore Systems

Devendra Rai and Lothar Thiele
Computer Engineering and Networks Laboratory, ETH Zurich, 8092 Zurich, Switzerland.
Email: firstname.lastname@tik.ee.ethz.ch

***Abstract*—A calibration based method to construct fast and accurate thermal models of the state-of-the-art multicore systems is presented. Such models are usually required during Design Space Exploration (DSE) exercises to evaluate various task-to-core mapping, associated scheduling and processor speed-scaling options for their overall impact on the system temperature. Current approaches require modeling the thermal characteristics of the target processor using numerical simulators, which assume accurate information about several critical parameters (e.g., the processor floorplan). Such parameters are not readily available, forcing the system designers to use time and cost intensive, and possibly error-prone techniques such as using heat maps for reverse-engineering such parameters. Additionally, advanced power and temperature management algorithms commonly found in the state-of-the-art processors must also be accurately modeled. This paper proposes a calibration based method for constructing the complete system thermal model of a target processor *without* requiring any hard-to-get information such as the detailed processor floorplan or system power traces. Taking an example of a sufficiently complex Intel Xeon 8-core processor, we show that our approach yields an accurate thermal model, which is also lightweight both in terms of memory and compute requirements to be practically feasible for DSE over current processors.**

I. Introduction

Recent technology scaling has made processors more vulnerable to experiencing dangerously high temperatures, especially when executing demanding applications, see [1]. Constant exposure to high temperatures and/or rapid changes in the temperature of the processor are known to reduce its reliability, see [2]. In the short term, unchecked temperature rise may result in a degraded system performance as the processor attempts to control its temperature by lowering the operating frequency, or in the worst case, by shutting down. Such unforeseen performance penalties may be unacceptable in some use cases, such as streaming applications with tight throughput constraints. Runtime techniques are available which may help in keeping the temperature of the processor within safe bounds. However, these techniques are *reactive* in nature, and either rely on Dynamic Voltage and Frequency Scaling (DVFS) which imposes unforeseen penalties on the system performance, or task migration, making such techniques not particularly suited to embedded systems, specifically when operating under tight performance constraints (e.g., throughput or latency), see [3], [4]. Another approach is to perform Design Space Exploration (DSE) in order to compute such task-to-core mappings, associated task schedules and processor speed scaling options (collectively called *bindings*) which ensure that the temperature of the processor always stays within a specified safe bound, see [5], [6]. This technique is preferable from the system performance standpoint, but requires an *accurate and computationally efficient* thermal model of the target processor.

The state-of-the-art for thermal modeling of a processor is to use a simulator such as Hotspot, see [7]. In principle, it is possible to satisfactorily model any processor using such a simulator, given necessary information such as the detailed processor floorplan, power model at the micro-architectural level, cooling solution used, and power and/or temperature management algorithms implemented in the processor. Such details are rarely available, especially for current processors, forcing system designers to use time and cost intensive approaches in order to reverse engineer these critical parameters to be used with the thermal simulator. One common approach is to use heat maps captured from the silicon layer of the given processor as it executes instructions, and use heuristic algorithms to back-estimate the detailed power model to be used with the Hotspot simulator, see [8]. However, several critical details (e.g., processor floorplan, on-chip power and/or temperature management algorithms) must still be guessed. Furthermore, since the technique requires peeling off the packaging of the processor, the accuracy of the approach is not clear with the similar processor which retains its original packaging. The error in the estimated temperature trace due to incorrectly estimating one or more such parameters may be significant, see Figure 3, wherein a naïve model computes the overall trace of a core by simple superposition, ignoring the effect of the on-chip power and temperature management algorithm. Additionally, since Hotspot is a numerical simulator, it may be too slow to be used in any practical DSE exercise especially involving state-of-the-art processors with multiple cores. Thus, several approaches have been reported which trade off simulation accuracy for speed. One such approach is to use event-counters as a proxy for power dissipation at the core level, see [9]. The associated temperature trace may then be computed using a simple polynomial in the selected event counter. The reported approach is simple, fast, and is able to compute correct steady-state temperatures, but shows significant inaccuracy in estimating the transients, specifically at the task-scheduling boundaries. Furthermore, the accuracy of the approach was only measured against a modified Hotspot simulator. Another approach is the use of lumped thermal models which assume the entire processor as a point source of heat, and are therefore computationally very efficient, but may be highly inaccurate, see [10].

In another approach, Rai *et. al.* propose to make up for not having knowledge of the processor floorplan or detailed power model by assuming that an application exercises the processor in a well defined manner consistent across invocations and inputs (i.e., embedded applications), resulting in a so-called *thermal fingerprint* unique to each application, see [11]. Similar to the impulse response for a linear system, the thermal fingerprint completely defines the thermal model of the system *together* with the given application. The performance of the approach in terms of accuracy and speed was demonstrated against the Hotspot thermal model, but it remains unclear whether it performs as well with state-of-the-art complex computer systems.

In contrast, the approach proposed in this paper uses temperature sensors on-board the given state-of-the-art processor for constructing and validating the thermal model. It is assumed that the sensors are reliable and are placed at a location such that the readouts from the sensors provide the most representative temperature at that location.

Consequently, if the temperature traces computed using our thermal model agree with the readouts from the sensors, then the validation is taken to be correct and complete. A direct comparison of our results against a corresponding Hotspot thermal model (which does not yet exist) is infeasible due to significant time and cost challenges involved in accurately constructing such a model, as already pointed out.

Following the presented discussion, the problem addressed in this paper is summarized as follows:

Problem Statement: Given a processor P *with a set of on-chip temperature sensors* S *and a set of embedded applications* A*, construct an accurate and fast thermal model* M *of the processor* P *without requiring access to any hard-to-get parameters such as detailed power trace(s) or the floorplan of the processor.*

The model M is said to be accurate if the error in the estimated traces always remains less than the noise of the temperature sensors in the processor P; and is said to be fast if the temperature traces for *all* cores, each with tens of thousands of data points can be estimated in the order of seconds.

Contribution: The paper makes two contributions to the state-of-the-art: (i) an approach to construct an accurate and computationally efficient thermal model of the target processor without requiring any hard-to-get information such as the processor floorplan, detailed power traces or the details of the power and/or temperature management algorithms implemented in the processor, and (ii) validation of the model by extensive experiments on a state-of-the-art multicore processor, i.e., the Intel Xeon 8-core processor.

II. Overview of the Approach

Construction of the thermal model requires (i) access to the multiprocessor P, (ii) the set of embedded applications A which will be executed on P, and (iii) a special set of bindings to be used for calibration experiments. See Figure 1. During the calibration phase, the applications from the set A are executed as per specifically designed bindings on the processor and the resulting temperature traces collected. We exploit the observation that embedded applications exercise the processor in a well defined manner consistent across invocations and inputs, and estimate a transfer function between the application bindings and the observed *changes* in the temperature of the processor P. Thus, in contrast to current approaches wherein constructing a thermal model requires access to hard-to-get parameters, the proposed calibration experiments are designed to extract *all* the necessary parameters required by the thermal model. During the DSE phase, the system designer may explore *arbitrary* bindings of the applications in A until a solution satisfying the given set of temperature and performance constraints is discovered. It must be pointed out that though the same application set A is used during model construction (i.e., calibration) and DSE, the bindings used in these two phases may be completely unrelated.

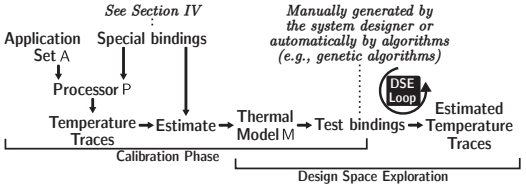

Fig. 1. Overview of the approach.

III. Setup and Notation

We consider a chip multiprocessor P with a set of C cores. Each core can operate at set of discrete frequencies in the set F. The cores may be heterogeneous, i.e., a given core may belong to one of the types given in the set $\mathsf{V} = \{GPU, FPU, RISC, ...\}$. A set of embedded applications A are available for execution on the processor P. During the construction of the thermal model, on-chip temperature sensors are sampled periodically with a period t_s. Therefore, the construction of the model and subsequent estimation of temperature traces is done for discrete time instants $t \in \mathbb{Z}^{\geq 0}$. The temperature trace sampled from a sensor is denoted as a tuple $\tau = \langle \tau_0, \tau_1, \cdots \rangle \in \mathsf{T}$ where τ_i is the temperature at the time instant $t = i$. A given application $a \in \mathsf{A}$ may or may not execute at a given time instant t, which is denoted by the associated utilization trace, $u = \langle u_0, u_1, \cdots \rangle \in \mathsf{U}$, with $u_i \in \{0, 1\}$, and $u_i = 0$ indicates that the application does not execute at time instant $t = i$. A core executing an application at a given time instant is considered to be active at that time instant, else, it is considered to be inactive. Usually, U is generated from a scheduling algorithm (e.g., round-robin). We assume that a set S of temperature sensors are available on the chip. A binding $b = \langle a \in \mathsf{A}, u \in \mathsf{U}, c \in \mathsf{C}, f \in \mathsf{F} \rangle \in \mathsf{B}$ indicates that an application a executes according to a utilization trace u on a core c which operates at a frequency f. Given a binding b, the helper function $\mathbf{a} : \mathsf{B} \rightarrow \mathsf{A}$ returns the application, the function $\mathbf{c} : \mathsf{B} \rightarrow \mathsf{C}$ returns the core, the function $\mathbf{f} : \mathsf{B} \rightarrow \mathsf{F}$ returns the operating frequency of the core, and the function $\mathbf{v} : \mathsf{B} \rightarrow \mathsf{V}$ returns the core-type. The function $\nu : \mathsf{C} \rightarrow \mathsf{V}$ provides the core-type for a given core. The i^{th} binding from the set B is denoted as b_i. The function $\mathbf{d} : \mathsf{C} \times \mathsf{C} \rightarrow \mathbb{R}^{\geq 0}$ computes the Euclidean distance between two cores.

IV. Constructing the Thermal Model

Overview: The overall procedure consists of a calibration phase with two parts: (i) construction of a thermal model M and a function $\mathbf{g}$ which is used to compute the change in the temperature of a core c due to *exactly* one application $a \in \mathsf{A}$ executing on the processor P with the binding $b \in \mathsf{B}$, and (ii) construction of the function $\mathbf{h}$ which together with M and $\mathbf{g}$ is used to compute the overall change in the temperature of a core $c \in \mathsf{C}$ due to multiple applications executing on the processor, each with its own binding. The function $\mathbf{h}$ also captures the thermal effect of the power and/or temperature management algorithms implemented in the processor.

A. The Thermal Model M

The thermal model is a set M of individual models $m \in \mathsf{M}$. A model m is derived such that it is possible to estimate the temperature trace at the distance d from an active core of type v executing an application a with the binding b. In particular, $m \in \mathsf{M}$ is an output-error (OE) model from the autoregressive moving average (ARMA) family of models, and has the form $m = \frac{B(z)}{F(z)}$ where $B(z)$ and $F(z)$ are polynomials in z^{-1}, the discrete time delay operator, see [12]. The function $\mathbf{m} : \mathsf{A} \times \mathsf{V} \times \mathbb{R}^{\geq 0} \times \mathsf{F} \rightarrow \mathsf{M}$ provides a model $m \in \mathsf{M}$ to compute the change in temperature at the distance d due to a core executing a binding b with an application $\mathbf{a}(b)$, on the core type $\mathbf{v}(b)$, and at frequency $\mathbf{f}(b)$. The construction of the thermal model requires designing a special utilization trace u^* together with an optimal set of calibration experiments as follows:

The Calibration Experiments: We exploit the observation that a given embedded application exercises the processor in a well defined manner consistent across invocations and inputs, resulting in temperature changes on the processor which is uniquely related to the given embedded application itself, also referred to as the *thermal fingerprint*, see [11]. Therefore, the purpose of the calibration experiment is to capture the thermal *impulse response* of each embedded application. Each experiment measures the change in temperature on the set

$\mathsf{C}' \subseteq \mathsf{C}$ of cores caused by exactly one application $a \in \mathsf{A}$ executing on the processor P with the associated binding b. Thus, a calibration experiment is determined by (i) the application a, (ii) the associated binding b, and (iii) the set of cores $\mathsf{C}' \subseteq \mathsf{C}$ from which the temperature traces are recorded. The observed changes are then captured into a thermal model $m \in \mathsf{M}$. In principle, a separate experiment is required for each unique combination of application, core-type, distance and core-frequency. In practice, the total number of calibration experiments are less due to two factors: (i) the influence of a hot core on the temperature of other cores reduces significantly with distance due to high silicon thermal resistivity, which in the presence of sensor noise may deteriorate the signal-to-noise (SNR) to the extent that constructing a model may be infeasible, and (ii) it may be possible to group applications based on similarities in their thermal impulse responses. However, we do not explore this possibility in the current work.

The Special Utilization Trace u^*: The purpose of this trace is to execute an application $a \in \mathsf{A}$ in a manner such that both the transient and steady-state thermal characteristics of the application can be accurately captured. Thus, the utilization trace consists of at least two distinct segments:

1) *Dynamics Segment*. This section of the calibration trace rapidly switches on and off the application (i.e., schedules the application in and out rapidly) in order to accurately model the resulting transient changes in temperature.
2) *Statics Segment*. This section of the calibration trace executes the application uninterrupted until all cores attain a steady state temperature. The application is then switched off until all cores again reach a steady state temperature. This part of the calibration trace is designed to model the steady state temperature change due to the application a.

The details are presented in Algorithm 1, in which an empty binding b is initialized, see line 1. Next, for every core type $v \in \mathsf{V}$, a *host core* $c^* = \mathbf{c}(b)$ is chosen to execute the application $\mathbf{a}(b)$ such that the resulting change in temperature over different distances can be observed in a single experiment, see line 4. The calibration experiment is performed for every unique core type $v \in \mathsf{V}$, frequency $f \in \mathsf{F}$, and application $a \in \mathsf{A}$, see lines 2-6. Excessive noise is removed from the observed temperature traces in order to avoid over-fitted models, see line 13. The selected denoising method must be edge-preserving in order to retain as much information on temperature transients as possible, see [14]. If there are multiples cores at a given distance d from $\mathbf{c}(a)$, then the average of the observations are taken, which also helps reduce the affects of noise, see line 20. Thereafter, a model is estimated from the given calibration trace and the observed temperature trace (line 21), see [12]. A model is separately computed for each discrete frequency $f \in \mathsf{F}$ since the thermal properties of the system may change considerably with frequency. In other words, it may not be possible to simply scale a given model derived at one frequency and use it at a different frequency. If this is not the case, an additional optimization step may be performed to remove redundant models.

The complexity of the assumed model may be varied by changing the number of poles (nf), the number of zeros (nb), and the initial discrete time delay (nk). The search for the best fitting model is an iterative procedure, gradually increasing the complexity of the model, see lines 28 - 43. The system designer may choose to restrict the maximum complexity of the temperature models by restricting the maximum number of poles, zeros and delay to `MAXPOLES`, `MAXZEROS` and `MAXDELAY` respectively. Between iterations, a new

Input: Applications A, Processor P
Output: Thermal Model M
Data: $t_s \leftarrow$ Discrete Sampling Interval;
Data: $T_{c,raw}, T_{d,obs}, d_{max} \leftarrow 0, d, H \leftarrow 0, u^*$; `// local variables`
1 $b \leftarrow \langle \phi, \phi, \phi, \phi \rangle$ `// Initial binding, all empty`
2 **foreach** *Core type* $v \in \mathsf{V}$ **do**
3 $d_{max} \leftarrow 0, H \leftarrow 0$;
4 Choose host core $\mathbf{c}(b) = c^* \mid \mathbf{d}(c^*, c_i) > \mathbf{d}(c_j, c_k), \nu(c^*) = v;\ c_i, c_j, c_k \in \mathsf{C}$;
5 **foreach** *application* $a \in \mathsf{A}$ **do**
6 **foreach** *frequency* $f \in \mathsf{F}$ **do**
7 Design the calibration trace $u^* \in \mathsf{U}$;
8 $T_{d,obs} \leftarrow 0, \forall d$;
9 Execute a according to binding $b = \langle a, u^*, c^*, f \rangle$; `//` $\mathbf{u}^*(b) = u^*, \mathbf{a}(b) = a$
10 **foreach** *core* $c \in \mathsf{C}$ **do**
11 $d \leftarrow \mathbf{d}(c, c^*)$;
12 $T_{c,raw} \leftarrow$ Observed temperature trace from core c ;
13 $T_{d,obs} \leftarrow T_{d,obs} +$ Denoise $(T_{c,raw})$;
14 $H[d] \leftarrow H[d] + 1$;
`// # cores at distance` d `from` c^*.
15 **if** $d > d_{max}$ **then**
16 $d_{max} \leftarrow d$;
17 **end**
18 **end**
19 **for** $d = 0 : d_{max}$ **do**
20 $T_{d,obs} \leftarrow T_{d,obs}/H[d]$;
`// Mean change at distance` d
21 $\mathbf{m}(a, v, d, f) =$`EstimateModel` $(T_{d,obs}, u^*, t_s)$;
`// Store the model;`
22 **end**
23 **end**
24 **end**
25 **end**
27 `Function` **EstimateModel**$(\hat{T}, u^*, t_s)$
`//` $\hat{T}$`: Observed temperature trace.` $\hat{U}$`: Calibration trace`
Data: fit $\leftarrow -1$, fit$' \leftarrow -1$;
`// fitness of the estimated model, Maximum: 100%.`
Data: [nb, nf, nk] $\leftarrow$ [2, 2, 1] ;
`// Initial order of model as vector [nb, nf, nk]`
`// nb-1: number of zeros in the model`
`// nf: number of poles in the model`
`// nk: discrete time delay (number of samples)`
Data: System Constraint: `MAXPOLES`, `MAXZEROS`, `MAXDELAY`;
Data: m: Computed temperature model of type $\frac{B(z)}{F(z)}$;
28 **for** *nk = 1:*`MAXDELAY` **do**
29 **for** *nb = 1:*`MAXZEROS` **do**
30 **for** *nf = 1:*`MAXPOLES` **do**
31 Compute model $m' = \frac{B(z)}{F(z)} \mid u^* \otimes m' + e \approx \hat{T}$ `// see [12]`
32 where e is the assumed error model;
`// e.g., zero mean white noise`
33 **if** $fit > fit'$ **then**
34 fit $\leftarrow$ `ComputeFit` $(\hat{T}, u^* \otimes m' + e)$;
35 fit$' \leftarrow fit$;
36 $m \leftarrow m'$;
`// the best model so far.`
37 **if** $fit = 100$ **then**
38 return m; `// perfect model.`
39 **end**
40 **end**
41 **end**
42 **end**
43 **end**
44 **return** m;
46 `Function` **ComputeFit**$(\hat{T}, \hat{T}^*)$
47 **return** $100 \times \left\{1 - \frac{\|\hat{T}^* - \hat{T}\|}{\|\hat{T} - \overline{\hat{T}}\|}\right\}$;
`//` `Normalized Root Mean Square Error estimate.` $\overline{\hat{T}}$`: Mean value of` $\hat{T}$`, see [13].`
49 `Function` **Denoise**(T_{raw})
`// Remove noise from the input trace using edge-preserving algorithms, e.g., Daubechies wavelets.`
50 **return** *Denoised temperature trace*;

Algorithm 1: Calibration based algorithm to compute the temperature model M.

model is judged better than a previously computed one if the fit value of the new model exceeds that of the old one, see line 33 and [13].

The Function $\mathbf{g}$*:* Given the thermal model M, the change in temperature on the core c due to *exactly one* binding b being executed on the processor can be computed using the function $\mathbf{g}$:

$$\tau = \mathbf{g}(b, \mathsf{M}, c) \stackrel{\text{def}}{=} \mathbf{m}(a, \mathbf{v}(b), \mathbf{d}(\mathbf{c}(b), c), \mathbf{f}(b)) \otimes \mathbf{u}(b) \quad (1)$$

Where $\otimes$ is the convolution operator and τ is the temperature trace. Notice that the function $\mathbf{g}$ is linear. Therefore, if a core dynamically switches between operating frequencies and tasks, separate temperature traces are computed for each task-frequency configuration and added point-wise to yield the final temperature trace.

The Function $\boldsymbol{h}$*:* The superposition function computes the final temperature trace for each core when multiple cores simultaneously execute a set of bindings B. Due to power and/or temperature management algorithms implemented in the processor, the final temperature trace τ for the core c may not be a simple superposition of temperature traces, also see Figure 3.

$$\tau \neq \sum_{b \in \mathsf{B}} \mathbf{g}(b, \mathsf{M}, c) \quad (2)$$

The exact thermal impact of such algorithms will vary between the processors. However, from the observations on the Intel Xeon processor, it is sufficient to use an application specific damping function $\mathbf{s} : \mathsf{A} \times \mathsf{V} \rightarrow \mathbb{R}^{>0}$ (details in the following paragraph). The final temperature trace for the core c is then computed by combining the *dominating* $T^d(t)$ and *non-dominating* components, $T^{nd}(t)$ as follows:

$$\tau = \mathbf{h}(\mathbf{g}(b_1, \mathsf{M}, c), \cdots \mathbf{g}(b_n, \mathsf{M}, c)) \stackrel{\text{def}}{=} T^d + T^{nd} \quad (3)$$

with:

$$\begin{aligned} \mathsf{B} &= \{b_1, \cdots, b_n\} \\ T^d(t) &= \mathbf{max}_{b \in \mathsf{B}}(\mathbf{g}(b, \mathsf{M}, c)(t)) \\ T^{nd}(t) &= \sum_{b \in \mathsf{B}'} (\mathbf{s}(\mathbf{a}(b), \mathbf{v}(b)) \times \mathbf{g}(b, \mathsf{M}, c)(t)) \\ B' &= B \setminus \left\{ b \mid \mathbf{g}(b, \mathsf{M}, c)(t) == T^d(t) \right\} \end{aligned}$$

where $\mathbf{g}(b, \mathsf{M}, c)(t)$ is the value of temperature trace $\mathbf{g}(b, \mathsf{M}, c)$ at time instant t, $\times$ is the scalar multiplication operator, and $+$ is the point-wise addition operator. The proposed structure of (3) is justified by the following observations: (i) the temperature of an active core c is largely determined by the application executing on itself, dominating the observed temperature changes for that core, (ii) if the core c is inactive, the temperature change on it cannot be lower than the maximum change in the temperature of c due to all other active cores acting individually, (iii) the flow of heat between a pair of cores is proportional to the temperature difference between them, and finally (iv) a certain power and temperature penalty is incurred when the first resource intensive application mapped to the processor invokes the on-board resource (e.g., power and temperature) management infrastructure. The penalty is amortized as more applications are mapped to the processor. This results in a large temperature change due to the first application, whereas further rise in temperature is relatively modest as more resource intensive applications are mapped to the processor. This behavior is captured by the damping function $\mathbf{s}$ (details follow).

The Damping Function $\mathbf{s}$*:* The damping function $\mathbf{s}(a, v)$ is also determined using a sequence of calibration experiments detailed in the Algorithm 2. The damping function is determined separately for each core type and application. The overall approach is to execute

Input: Applications A, Processor P, Thermal Model M
Output: Damping Ratios $\mathbf{s}(a, v), \forall a \in \mathsf{A}, \forall v \in \mathsf{V}$
Data: $f^* \leftarrow \max(\mathsf{F})$; `// Choose highest operating frequency`
Data: $t_s \leftarrow$ Discrete Sampling Interval;
Data: $\tau_{raw}, \tau_{base}, \tau_{obs}$: Variables;
1 $b \leftarrow \langle \phi, \phi, \phi, \phi \rangle$; `// Initial binding, all empty`
2 **foreach** *core type* $v \in \mathsf{V}$ **do**
3 Choose host core $c^* | \nu(c^*) = v$ s.t. c^* is at the center of distribution of cores of similar type;
4 **foreach** *application* $a \in \mathsf{A}$ **do**
5 $b = \langle a, u^*, c^*, f^* \rangle$; `// reuse` $\mathbf{u}^*$ `from Algorithm 1`
6 Execute the binding b;
7 $T_{raw} \leftarrow$ Observed temperature trace from core c^*;
8 $T_{base} \leftarrow$ `Denoise` (T_{raw}); `// Baseline`
9 $\mathsf{C}^* \leftarrow \{c \mid c \in \mathsf{C}, \nu(c) = v\}$;
`// Set of all cores of type` $v \in \mathsf{V}$ `including` c^*;
10 $\mathsf{B}' \leftarrow \{\langle a, u^*, c, f^* \rangle \mid c \in \mathsf{C}^*\}$;
`// A set of bindings for all cores` $c \in \mathsf{C}^*$
11 Execute the set of bindings B' ;
12 Execute the binding b; `// See line 6;`
13 $T_{raw} \leftarrow$ Observed temperature trace from core c^* ;
14 $T_{obs} \leftarrow$ `Denoise` (T_{raw}) ;
15 $\mathbf{s}(a, v) = \frac{T_{obs} - T_{base}}{\sum(\mathbf{g}(b_1, \mathsf{M}, c^*), \cdots, \mathbf{g}(b_n, \mathsf{M}, c^*))}$;
`//` $\{b_1, \cdots, b_n\} = \mathsf{B}'$
16 **end**
17 **end**
19 `Function` **Denoise**(T_{raw})
`// See Algorithm 1, line 49`
20 **return** *Denoised temperature trace*;

Algorithm 2: Algorithm to compute the damping function.

an application $a \in \mathsf{A}$ on a core type v located at the approximate center of the distribution of cores of the same type (line 2). A binding b is constructed in order to execute a at the highest possible operating frequency f^* with the utilization trace u^* developed earlier (line 5). Selection of the highest frequency improves the signal-to-noise (SNR) ratio permitting accurate estimates of the damping function. Next, the same application a is executed on *all* the cores of the similar type as per the same utilization trace u^* as before (line 10). The change in temperature on c^* is measured and compared against the expected change in temperature using the thermal model M yielding the damping ratio (line 15).

B. Limitations to the Accuracy of the Thermal Model

The accuracy of the thermal model and therefore the temperature traces may be limited due to the following factors: (i) the noise in the temperature sensors used during the calibration phase, (ii) if the processor has a limited set of temperature sensors (e.g., less number of sensors than the number of cores) then there will be a few cores for which the true estimate of the temperature will not be available, also restricting the accuracy of the resulting thermal model, (iii) if the designer chooses to trade-off the complexity of the model for speed, then the resulting model may not faithfully reproduce temperature traces, especially transients, and (iv) two temperature sensors located at an equal distance d from an active core c may not sense the same change in temperature as a temperature hotspot on c may not be symmetrically located. Algorithm 1 does not consider such asymmetry in order to keep the procedure and the model simple. However, should it be necessary, removing this inaccuracy from the model is straightforward.

V. Experiments and Results

Hardware Environment: Though the proposed approach for constructing the thermal model is versatile and independent of any specific hardware architecture, we demonstrate that the our approach can be successfully used in constructing the thermal model of a state-of-the-art multicore processor featuring power and temperature

management algorithms. Therefore, a commercially available blade server consisting of two Intel Xeon E5-2690 8-core processors sharing 128GB RAM was selected. Though the selected processor is not specifically designed for embedded applications, it is representative for the trend in using multicore and multiprocessor architectures in high-performance embedded applications. In the available system, cores 0-7 belong to processor with ID 1, whereas cores 8-15 belong to the processor with ID 2. For all practical purposes, the processors may be considered to be thermally isolated from each other. The automatic on-demand frequency increase (i.e., the *Intel turbo boost*) was disabled in order to prevent any unplanned operating frequency changes. All cores within *each* processor can operate only at a *single* common frequency in the range 1.2GHz - 2.9GHz. Precise control over the system fans is not possible, therefore, in order to avoid any errors due to unforeseen variation fan speeds, all fans operate at a constant full speed of 15,000 RPM. The processor features 1 temperature sensor per core with a resolution of 1^oC and error of $\pm 1^oC$.

Software Environment: The operating system (OS) is a preemptible Linux kernel 3.11, contained to core 15 using `cpusets`, see [15]. All non-essential kernel services are stopped in order to prevent any unexpected variations in temperature or resource availability during the experiments. A dedicated observer on core 13 utilizes a modified `coretemp` module for reading temperature sensors every 1 ms, which represents the common tick interval in modern systems, see [16]. Each application executes with real-time priority according to a precomputed binding. All algorithms are implemented in Matlab (R2013b) and the system identification toolbox is used to construct the thermal model (line 31, Algorithm 1). A mix of eight applications `CPUBurn-in`, `H.263 Encoder`, `Basicmath`, `RSA Decoder`, `Bitcount`, `ADPCM`, `Quicksort` and `FFT` were carefully selected in order to represent a wide variety of embedded applications, see [17]–[20].

The Results: The validity of the proposed approach is justified by the following results: (i) *accuracy:* The error in the temperature traces estimated using the thermal model M is less than 2^oC (the overall error in the temperature sensors) for all tested bindings, including those in which a core switches between applications and frequencies at runtime, (ii) *use case:* An example of temperature aware DSE in order to reduce the thermal cycles experienced by a given core, and (iii) *efficiency:* Memory and computational complexity of computing temperature traces at the DSE stage showing the resource efficiency of the developed thermal model.

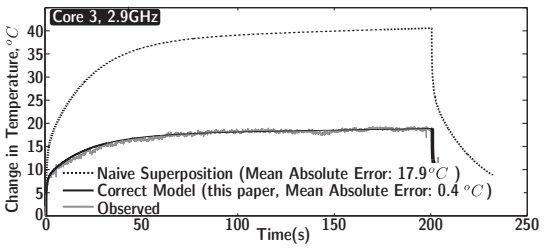

Fig. 3. Long term accuracy of the thermal model when all cores execute a different application from the set A.

Accuracy: The number of calibration experiments were restricted such that $d \leftarrow \max(d, 3)$ for each combination of a, v and f without any significant loss of accuracy owing to sensor noise and thermal resistivity of silicon, see Section IV-A and Algorithm 1. Three types of experiments were carried out: (i) *multiple applications:* 23 experiments were designed, out of which the first 10 experiments required all the cores in the range 0-7 to execute a mutually unique application from the set A. For the next 13 experiments, a randomly chosen subset of cores execute randomly selected applications. Applications do not share cores. The associated utilization traces were constructed such that the continuous execution time of the corresponding task was a random variable in the range 100-180,000 ms. The experiments were repeated for processor frequencies in the set {2.9GHz, 2.4GHz, 1.2GHz}. (ii) *dynamic task and frequency switching:* a core multitasks between applications and simultaneously cycles between operating frequencies in the set {2.9GHz, 2.4GHz, 1.2GHz}. (iii) *Long Range:* Cores 0-7 executes a randomly drawn application from the set A for 200s. Due to space constraints, detailed temperature trace comparisons are provided for a subset of results, whereas a full summary is presented in Figure 4. The mean absolute error (MAE) in the temperature traces computed using the model remains less than 2^oC, the uncertainty level in the temperature sensors, even when the temperature traces have significant transients, see Figure 2. Notice that the continuous execution time of tasks in Figure 2 is restricted in order to present sufficient visual details on the accuracy of the thermal model in estimating temperature transients. As a consequence, the dynamic range of the observed temperature traces is less than 10^oC. Finally, the error in the estimated temperature trace when each core (0-7) executes a randomly drawn application from the set A for hundreds of seconds remains small ($< 1^oC$), see Figure 3. For reference, the error due to a (naïve) lumped thermal model is also shown. The correlation between errors across frequencies is due to an error in estimated thermal model for the `BASICMATH` application due to the asymmetrically located temperature hotspot, as discussed in Section IV-B, see Figure 4. The temperature traces estimated using the model are accurate even when a core dynamically changes between frequency and tasks, with the observed mean absolute error being 1.3^oC, see Figure 6.

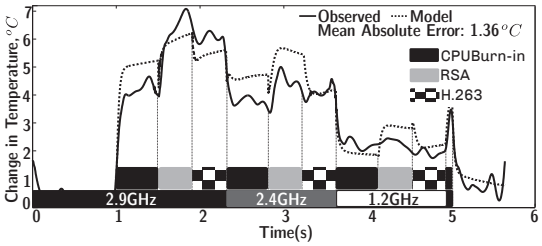

Fig. 6. Estimated vs. observed temperature for a core with dynamic frequency and application switching.

Use Case for Temperature Aware DSE: Reducing thermal cycles experienced by the processor may be one of the constraints which must be met by a feasible solution. Consider the case wherein three tasks execute on different cores with given schedules, but result in significant thermal cycles, see Figure 5(A-C). This may be common when the tasks are periodic in nature, and their average utilization of the given core is less than 100%. A temperature-aware DSE can be used to explore better bindings which can significantly decrease thermal cycles experienced by the cores in the processor, see Figure 5 (D). Notice that the total amount of work done by each application in Figures 5 (A-C) is exactly the same as in Figure 5 (D).

Memory and Computational Costs for DSE: The memory and computational requirements for estimating the temperature traces directly depend on the complexity of the assumed thermal model. For the complexity of [nb = 5, nf = 5, nk = 2] used in this work, the corresponding *fit* obtained was 88%. Each new temperature data point requires

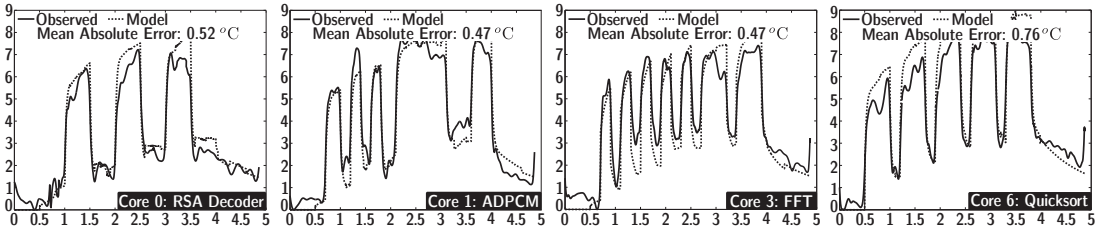

Fig. 2. Accuracy of the thermal model when *each* core executes an application according to a predefined binding. Only a subset of results shown. See Figure 4 for more details. Core frequency: 2.9GHz. Horizontal axis: time(s); Vertical axis: Change in temperature, ^{o}C.

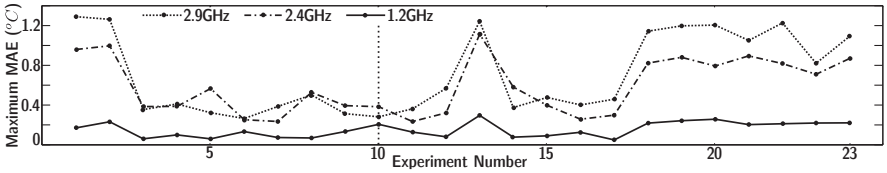

Fig. 4. Summary of 69 experiments. Each experiment from 1-10 requires all 8 cores to execute a mutually unique application. Each experiment from 11-23 selects a random set of cores and applications. Applications do not share cores. Correlation between errors across frequencies is due to an imperfect thermal model for the `BASICMATH` application.

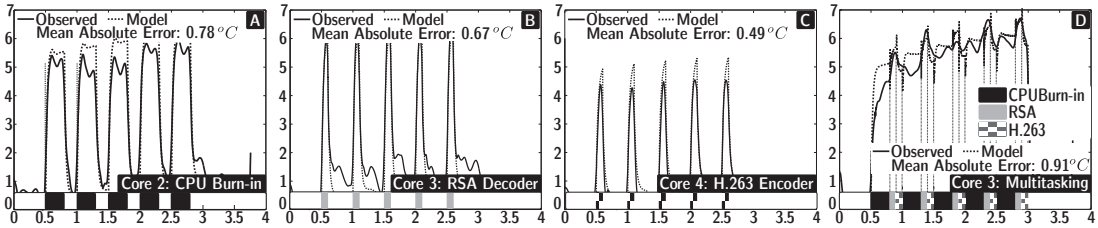

Fig. 5. Thermally-aware DSE: Figure A-C show thermal cycles experienced by the processor for a naive binding. Figure D shows an optimized binding which reduces thermal cycles. The total work done by the three tasks remains the same. Horizontal axis: time(s); Vertical axis: Change in temperature, ^{o}C.

(nb+nf) multiplications, 1 division and (nb+nf) additions, with the associated storage requirements for (nb+nf) coefficients and (nk+nb−1) data points for the delay operator. Therefore, it can be concluded that the memory and compute costs incurred in using the thermal model during DSE with state-of-the-art processors is insignificant.

VI. Conclusions

This paper presented a calibration based technique to construct a fast and accurate thermal model for a state-of-the-art multicore processor, without requiring any auxiliary information such as the processor floorplan, or detailed power traces, making our technique applicable to any processor. Extensive experiments on an Intel Xeon 8-core processor with 8 applications validate both the accuracy and the speed of our technique. Specifically, we showed that the error in the estimated temperature traces using the proposed technique is within the noise envelope of the on-chip temperature sensors used during the calibration phase, for all tested bindings, including those in which all cores of the processor execute a different application. We also presented a use-case in which the thermal model is used for DSE to discover thermally optimal bindings allowing the processor to run without any danger of experiencing thermally induced faults or unforeseen performance losses.

References

[1] J. Kong, S. W. Chung, and K. Skadron, "Recent Thermal Management Techniques for Microprocessors," *ACM Comput. Surv.*, vol. 44, no. 3, pp. 13:1–13:42, Jun. 2012.

[2] J. Srinivasan, S. Adve, P. Bose, and J. Rivers, "The impact of technology scaling on lifetime reliability," in *Dependable Systems and Networks, 2004 International Conference on*, June 2004, pp. 177–186.

[3] S. Herbert and D. Marculescu, "Analysis of dynamic voltage/frequency scaling in chip-multiprocessors," in *ACM/IEEE International Symposium on Low Power Electronics and Design (ISLPED)*, Aug 2007, pp. 38–43.

[4] Y. Wang, K. Ma, and X. Wang, "Temperature-constrained Power Control for Chip Multiprocessors with Online Model Estimation," *SIGARCH Comput. Archit. News*, vol. 37, no. 3, pp. 314–324, Jun. 2009.

[5] S. Murali, A. Mutapcic, D. Atienza, R. Gupta, S. Boyd, and G. De Micheli, "Temperature-aware processor frequency assignment for MPSoCs using convex optimization," in *Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, 2007, pp. 111–116.

[6] G. Paci, P. Marchal, F. Poletti, and L. Benini, "Exploring "Temperature-aware" Design in Low-power MPSoCs," in *Proceedings of the Conference on Design, Automation and Test in Europe: Proceedings*, ser. DATE '06, 2006, pp. 838–843.

[7] W. Huang, M. R. Stan, K. Skadron, K. Sankaranarayanan, S. Ghosh, and S. Velusam, "Compact thermal modeling for temperature-aware design," in *Proceedings of the 41st Annual Design Automation Conference*, ser. DAC, 2004, pp. 878–883.

[8] F. J. Mesa-Martinez, J. Nayfach-Battilana, and J. Renau, "Power Model Validation Through Thermal Measurements," in *Proceedings of the 34th Annual International Symposium on Computer Architecture*, ser. ISCA, 2007, pp. 302–311.

[9] S. W. Chung and K. Skadron, "Using On-Chip Event Counters For High-Resolution, Real-Time Temperature Measurement," in *Thermal and Thermomechanical Phenomena in Electronics Systems, ITHERM*, 2006, pp. 114–120.

[10] D. Rai, H. Yang, I. Bacivarov, J.-J. Chen, and L. Thiele, "Worst-case temperature analysis for real-time systems," in *Design, Automation Test in Europe Conference Exhibition (DATE), 2011*, March 2011, pp. 1–6.

[11] D. Rai, H. Yang, I. Bacivarov, and L. Thiele, "Power Agnostic Technique for Efficient Temperature Estimation of Multicore Embedded Systems," in *International Conference on Compilers, Architectures and Synthesis for Embedded Systems*, ser. CASES '12, 2012, pp. 61–70.

[12] L. Dugard and I. D. Landau, "Recursive Output Error Identification Algorithms Theory and Evaluation," *Automatica*, vol. 16, no. 5, pp. 443–462.

[13] http://www.mathworks.ch/ch/help/ident/ref/goodnessoffit.html.

[14] D. Lazzaro and L. Montefusco, "Edge-preserving wavelet thresholding for image denoising," *Journal of Computational and Applied Mathematics*, vol. 210, pp. 222–231, 2007.

[15] https://www.kernel.org/doc/Documentation/cgroups/cpusets.txt.

[16] https://wiki.edubuntu.org/CoreTemp.

[17] http://www.cpuburnin.com/.

[18] C. Lee, M. Potkonjak, and W. Mangione-Smith, "MediaBench: a tool for evaluating and synthesizing multimedia and communications systems," in *EEE/ACM International Symposium on Microarchitecture.*, Dec 1997, pp. 330–335.

[19] https://polarssl.org/.

[20] 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 *Proceedings of the Workload Characterization.*, ser. WWC '01, 2001, pp. 3–14.