MatEx: Efficient Transient and Peak Temperature Computation for Compact Thermal Models

Santiago Pagani*, Jian-Jia Chen†, Muhammad Shafique*, and Jörg Henkel*

*Chair for Embedded Systems (CES)
Karlsruhe Institute of Technology (KIT), Germany

†Department of Informatics
TU Dortmund University, Germany

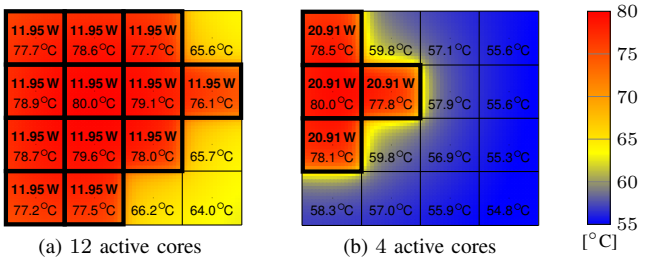
Corresponding Author: santiago.pagani@kit.edu

***Abstract*—In many core systems, run-time scheduling decisions, such as task migration, core activations/deactivations, voltage/frequency scaling, etc., are typically used to optimize the resource usages. Such run-time decisions change the power consumption, which can in turn result in transient temperatures much higher than any steady-state scenarios. Therefore, to be thermally safe, it is important to evaluate the transient peaks before making resource management decisions. This paper presents a method for computing these transient peaks in just a few milliseconds, which is suited for *run-time* usage. This technique works for any compact thermal model consisting in a system of first-order differential equations, for example, RC thermal networks. Instead of using regular numerical methods, our algorithm is based on analytically solving the differential equations using matrix exponentials and linear algebra. This results in a mathematical expression which can easily be analyzed and differentiated to compute the maximum transient temperatures. Moreover, our method can also be used to efficiently compute all transient temperatures for any given time resolution without accuracy losses. We implement our solution as an open-source tool called MatEx. Our experimental evaluations show that the execution time of MatEx for peak temperature computation can be bounded to no more than 2.5 ms for systems with 76 thermal nodes, and to no more than 26.6 ms for systems with 268 thermal nodes, which is three orders of magnitude faster than the state-of-the-art for the same settings.**

I. Introduction

Managing and dealing with temperature is a major issue in computing systems. In order to avoid undesired high temperatures, chips are generally provided with Dynamic Thermal Management (DTM) techniques [8]. When a part of a chip heats up above a predefined threshold temperature, DTM is triggered such that the temperature is reduced. For such a purpose, DTM techniques can, for example, power down parts of the chip, gate the clock of cores, or reduce the supply voltage and frequencies. Although naturally DTM is necessary because it guaranties that chips are not damaged due to high temperatures, frequent triggers of aggressive DTM techniques may degrade the overall performance of the system.

There are several power budgeting and thermal management techniques in the literature (e.g., [5], [14]) which are derived/formulated for the steady-state temperatures, that is, the stable temperatures achieved when the system runs during a long enough time without changes in power. The problem with such approaches is that when there are power changes (e.g., due to mapping/scheduling decisions), the transient temperatures might exceed the corresponding steady states. When these steady-state temperatures are high enough, such a transient behavior might trigger DTM, thus reducing the system's performance. More importantly, if the transient temperatures grow faster than the speed in which DTM can react to them, chips can be seriously damaged. Such transient effects are visible in the following motivational example.



(a) 12 active cores (b) 4 active cores

Fig. 1: Motivational example: Two different steady-states. Active cores are boxed in black. Top numbers represent the power consumptions of active cores and bottom numbers represent the temperatures in the center of each core. Detailed temperatures are shown according to the color bar.

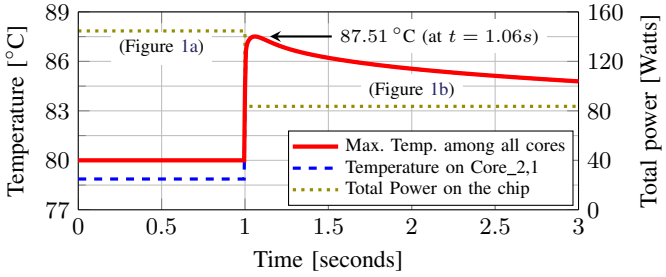


Fig. 2: Motivational example: Transient temperatures. During $t = [0s, 1s]$ there are 12 active cores according to Figure 1a. During $t = [1s, 3s]$, we activate 4 cores according to Figure 1b. The highest transient temperature happens on the core in the 2nd row and 1st column, that is, Core_2,1, at $t = 1.06s$.

Motivational Example: For simplicity in presentation, consider a many-core system with 16 cores[1] of size 3.2mm × 3.0mm, arranged in 4 rows and 4 columns. Assume a threshold temperature for triggering DTM of 80°C and a cooling solution from the default configuration of HotSpot [7]. After running simulations with HotSpot, Figure 1 shows the steady-state temperature distribution of two mappings with different numbers of active cores and power consumptions. For both cases the maximum steady-state temperature among all cores is 80°C, that is, DTM is not triggered in either steady state.

However, as seen in the transient simulations in Figure 2, the temperature on at least one core exceeds 80°C for several seconds, even reaching 87.51°C, when transitioning from the mapping in Figure 1a to that in Figure 1b. In fact, the temperature on Core_2,1 *increases to* 86.60°C *almost instantly*, potentially damaging the chip because DTM takes some time to become active. This transient behavior happens regardless

[1] *Out-of-order* Alpha 21264 cores in 22nm simulated with McPAT [12].

of the fact that the steady-state temperatures for both mappings do not exceed the threshold temperature of 80°C. *The cause behind this effect is that the* 8 *cores that are shut-down require some time to cool down, and they transfer heat to the other cores during this period.*

Objective: In this paper we present a *fast* and *accurate* method to compute the peaks in transient temperatures for *run-time* usage. In this way, the system can estimate whether a mapping/scheduling decision will exceed the threshold for triggering DTM. Our method works with any compact thermal model composed by a system of first-order differential equations, for example, RC thermal networks [7]. The proposed technique is based on matrix exponentials and linear algebra, allowing us to derive an analytical expression for computing the transient temperatures. The peaks in transient temperatures can be then easily computed by analyzing and differentiating such an equation, which is something impossible to do when solving the system of differential equations with regular numerical methods. Moreover, given that our method is based on an exact solution which is a function of time, it can also efficiently compute all transient temperatures for any given time resolution without accuracy losses.

Our Novel Contributions: For thermal models composed by a system of first-order differential equations:

- Based on matrix exponentials, we derive a fast polynomial-time algorithm that efficiently computes all transient temperatures from input power traces, for any given time resolution without accuracy losses.
- We derive a *fast* and *accurate* method to compute peaks in transient temperatures generated by changes of power consumptions inside a chip.

Open-Source Contributions: We implement our algorithms as an open-source tool called MatEx (from matrix exponentials), which is particularly useful for making thermally safe *run-time* mapping and scheduling decisions. MatEx is available for download at http://ces.itec.kit.edu/download.

II. Related Work

Significant work has been done for temperature estimation in integrated circuits. Most are limited to steady-state computations, e.g., [11], [15], [19], failing to detect transient peaks in temperature. However, there exist several techniques that are capable of dealing with transient thermal simulations, e.g., [3], [6], [7], [9], [16], [17], [18], [20].

The work in [18] presents a 3-D transient thermal simulator based on the Alternating Direction Implicit (ADI) method, whit linear time complexity and a linear memory requirement. However, different packaging components cannot be modeled with detailed temperature distribution information. In [9], authors propose coupling a gate-level logic simulator with an analytical solution. However, the model is only applicable for dynamic power consumption and the analytical solution is computationally exhausting for complex geometries with composite materials, limiting the approach only to dice geometries. In [3], authors present ESESC, a time-based sampling simulator that enables integrated power and temperature evaluations for multi-core systems. Nevertheless, temperature simulations cannot be easily separated from complete simulations, denying fast temperature prediction based on power consumption values. The work in [16] suggests a technique for temperature estimation based on application-specific calibrations using the available built-in sensors. The authors show that each application has a unique thermal signature, providing a fast method for temperature estimation by combining mapping and scheduling information. The limitation of [16] is that it assumes that an application consumes constant power as long as it is running.

Among all thermal simulators, HotSpot [7] is the most widely used. HotSpot constructs a compact thermal model based on the popular stacked-layer packaging scheme in modern Very Large-Scale Integration (VLSI) systems, resulting in an RC thermal network. In addition to modeling silicon and several packaging layers made of different materials, HotSpot includes a high-level on-chip interconnect self-heating power and thermal model such that the thermal impacts on interconnects can also be taken in consideration. For transient temperature computations, HotSpot solves the system of differential equations using a fourth-order *Runge-Kutta* numerical method, with an adaptive number of iterations.

There are a few techniques that currently compete with HotSpot [6], [17], [20]. Power Blurring [20] is a matrix convolution technique, in which the temperature distribution of the chip is regarded as a blurred image of the power map when it is blurred with a filter mask for the impulse response, obtained by using a Finite Element Analysis (FEA) tool such as ANSYS [2]. The thermal profile for a given power map is obtained by convolving it with the thermal mask. The work in [17] proposes a compact thermal modeling technique, which builds a composable model from detailed structures for each basic module using the finite difference method, then reduces the model complexity, and tries to merge the boundary nodes of modules to improve the reduction efficiency, leading to different space discretizations for the whole thermal system. By using the generalized integral transforms (GIT), the work in [6] presents an analytical thermal simulator to estimate the temperature distribution on a chip with a truncated set of spatial bases which only needs very small truncation points.

Although some of these tools are reasonably efficient, *none* of the considered techniques is suitable to *only* compute the peaks in temperature during the transient state. Hence, traditional tools must extract these peaks from extensive simulations for many time steps, taking several seconds to compute. Contrarily, the method presented in this paper can compute peaks in transient temperatures in just a few milliseconds, which is particularly useful for *run-time* mapping and scheduling decisions, and it can efficiently compute all transient temperatures for any time resolution without accuracy losses.

III. System Model

This paper focuses on compact thermal models that are composed by a system of first-order differential equations, relating the temperatures of different areas of the chip (thermal nodes) with their power consumptions and the ambient temperature, as shown in Equation (1). Considering the well-known duality between thermal and electrical circuits, one such a model is, for example, an RC thermal network like the one used in HotSpot [7]. For simplicity of presentation, the notation and descriptions used in this paper are namely those used in RC thermal networks. *Nevertheless, as long as the thermal model can be formulated as in Equation* (1)*, the same methods and conclusions apply for other models.*

An RC thermal network is composed by N thermal nodes that are interconnected through thermal conductances. Every thermal node is also associated with a thermal capacitance to consider the effects of the transient temperatures. The ambient temperature, defined as T_{amb}, is considered to be constant (there is no capacitance associated with it). The power

consumptions of active elements are heat sources. Thus, for an RC thermal network with N thermal nodes, we can build a system of N first-order differential equations, expressed as

$$\mathbf{A}\mathbf{T}' + \mathbf{B}\mathbf{T} = \mathbf{P} + T_{\text{amb}}\mathbf{G}, \tag{1}$$

where matrix $\mathbf{A} = [a_{i,j}]_{N\times N}$ contains the thermal capacitances, matrix $\mathbf{B} = [b_{i,j}]_{N\times N}$ contains the thermal conductances between vertical and lateral neighboring nodes, column vector $\mathbf{T} = [T_i(t)]_{N\times 1}$ represents the temperature on every thermal node, column vector $\mathbf{T}' = [T_i'(t)]_{N\times 1}$ holds the first order derivative of the temperature on every thermal node with respect to time, column vector $\mathbf{P} = [p_i]_{N\times 1}$ contains the power consumption on every thermal node, and column vector $\mathbf{G} = [g_i]_{N\times 1}$ contains the thermal conductance between every thermal node and the ambient temperature. When thermal node i is not in contact with T_{amb} the value of g_i is zero. Furthermore, Equation (1) can be rephrased as

$$\mathbf{T}' = \mathbf{C}\mathbf{T} + \mathbf{A}^{-1}\mathbf{P} + T_{\text{amb}}\mathbf{A}^{-1}\mathbf{G} \qquad \text{with } \mathbf{C} = -\mathbf{A}^{-1}\mathbf{B}. \tag{2}$$

Moreover, initial conditions are needed when solving any differential equation. Hence, we define vector $\mathbf{T}_{\text{init}} = [T_{\text{init}\,k}]_{N\times 1}$ which contains the initial temperatures on all nodes at $t = 0s$.

In the steady-state, Equation (1) becomes

$$\mathbf{B}\mathbf{T}_{\text{steady}} = \mathbf{P} + T_{\text{amb}}\mathbf{G} \quad \text{or} \quad \mathbf{T}_{\text{steady}} = \mathbf{B}^{-1}\mathbf{P} + T_{\text{amb}}\mathbf{B}^{-1}\mathbf{G},$$

where vector $\mathbf{T}_{\text{steady}} = \left[T_{\text{steady}_k}\right]_{N\times 1}$ contains the steady-state temperatures on all thermal nodes, and $\mathbf{B}^{-1} = \left[b^{-1}{}_{k,j}\right]_{N\times N}$ is the inverse of matrix $\mathbf{B}$. Moreover, only focusing on the steady-state temperature of node k, denoted T_{steady_k}, we have

$$T_{\text{steady}_k} = \sum_{j=1}^{N} b^{-1}{}_{k,j} \cdot p_j + T_{\text{amb}} \cdot \sum_{j=1}^{N} b^{-1}{}_{k,j} \cdot g_j. \tag{3}$$

IV. Problem Definition

For a given matrix $\mathbf{A}$, matrix $\mathbf{B}$, vector $\mathbf{G}$, and ambient temperature T_{amb}, the objective of this paper is to derive a fast and accurate method to efficiently compute the peak in the transient temperature on node k, that is, $T_k(t)$, after a change in the power consumption of one or more nodes, such that vector $\mathbf{P}$ is the new power vector after the change in power.

For this purpose, we first need to find an analytical solution for the system of first-order differential equations presented in Equation (2). In other words, we need to find a mathematical expression from which we can efficiently compute $T_k(t)$ for any thermal node k and time $t \geq 0$. This is done in Section V.

Then, from the results in Section V, we can find the time instant in which $T_k(t)$ is maximized, which we define as $t^{\uparrow k}$. Finally, we simply compute $T_k\left(t^{\uparrow k}\right)$ to obtain the value of the peak in the transient temperature of node k for the given power change. This is presented in Section VI.

V. Computing All Transient Temperatures

It has being well studied (e.g., [13]), that the system of first-order differential equations from Equation (2) can be analytically solved by using matrix exponentials. Therefore, assuming that when there is a change in power this happens at time $t = 0$ and the new power values are those of vector $\mathbf{P}$, applying the results from [13] we can re-write Equation (2) as a function of $\mathbf{T}_{\text{init}}$, $\mathbf{T}_{\text{steady}}$, and $\mathbf{C}$, resulting in

$$\mathbf{T} = \mathbf{T}_{\text{steady}} + e^{\mathbf{C}t}\left(\mathbf{T}_{\text{init}} - \mathbf{T}_{\text{steady}}\right), \tag{4}$$

where $e^{\mathbf{C}t} = \left[e^{\mathbf{C}t}{}_{i,j}\right]_{N\times N}$ is defined as a matrix exponential. Matrix $\mathbf{C}$ and matrix $\mathbf{B}$ are related to hardware and they are constant for a given thermal model. Hence, the variables in Equation (4) are t, $\mathbf{T}_{\text{init}}$, and $\mathbf{T}_{\text{steady}}$ (which in turn depend on $\mathbf{P}$ and T_{amb}). *Note that Equation (4) does not imply that the temperature on every node approaches its steady-state temperature exponentially, as $e^{\mathbf{C}t}$ is a matrix exponential and not a regular exponential function.*

There are several methods to solve matrix exponential $e^{\mathbf{C}t}$, including numerical methods that solve it for a given t, and analytical methods based on linear algebra in which t is a variable. Focusing on the latter, the work in [13] presents a solution to the matrix exponential as $e^{\mathbf{C}t} = \mathbf{V}\mathbf{D}\mathbf{V}^{-1}$. Here, $\mathbf{V} = [v_{i,j}]_{N\times N}$ represents a matrix containing the eigenvectors for matrix $\mathbf{C}$, matrix $\mathbf{V}^{-1} = [z_{i,j}]_{N\times N}$ is the inverse of matrix $\mathbf{V}$, and $\mathbf{D} = [d_{i,j}]_{N\times N}$ is a diagonal matrix such that $\mathbf{D} = \text{diag}\left(e^{\lambda_1\cdot t}, e^{\lambda_2\cdot t}, \ldots, e^{\lambda_N\cdot t}\right)$, where $\lambda_1, \lambda_2, \ldots, \lambda_N$ are the eigenvalues of matrix $\mathbf{C}$. The eigenvalues and eigenvectors of matrix $\mathbf{C}$ are computed by applying linear algebra [1], with a total time complexity of $O\left(N^3\right)$. Because this a physical and stable system in which all temperatures eventually reach their steady states, it holds that $\lambda_i \leq 0$ for all $i = 1, 2, \ldots, N$. *Note that for a given matrix $\mathbf{C}$ (that is, a given chip), the eigenvalues, matrix $\mathbf{V}$, matrix $\mathbf{V}^{-1}$, and $\mathbf{D}$ only need to be computed once.* In this way, the matrix exponential can be solved by computing every $e^{\mathbf{C}t}{}_{k,j}$ inside $e^{\mathbf{C}t}$ as

$$e^{\mathbf{C}t}{}_{k,j} = \sum_{i=1}^{N} v_{k,i} \cdot z_{i,j} \cdot e^{\lambda_i\cdot t}. \tag{5}$$

Applying these results for matrix exponentials from [13] to compute the transient temperatures, from Equations (4) and (5), the temperature on node k as a function of time becomes

$$T_k(t) = T_{\text{steady}_k} + \sum_{i=1}^{N} e^{\lambda_i\cdot t} \cdot v_{k,i} \cdot \sum_{j=1}^{N} z_{i,j}\left(T_{\text{init}\,j} - T_{\text{steady}_j}\right).$$

This computation can be achieved very efficiently by first building auxiliary matrix $\mathbf{H} = [H_{k,i}]_{N\times N}$, such that

$$H_{k,i} = v_{k,i} \cdot \sum_{j=1}^{N} z_{i,j}\left(T_{\text{init}\,j} - T_{\text{steady}_j}\right), \tag{6}$$

for all $k = 1, 2, \ldots, N$ and for all $i = 1, 2, \ldots, N$. Finally, for an ambient temperature T_{amb}, starting from initial temperatures $\mathbf{T}_{\text{init}}$ at $t = 0s$, and considering the steady-state temperatures $\mathbf{T}_{\text{steady}}$ after a power change to vector $\mathbf{P}$, the temperature on node k at time t, that is, $T_k(t)$, is computed as

$$T_k(t) = T_{\text{steady}_k} + \sum_{i=1}^{N} H_{k,i} \cdot e^{\lambda_i\cdot t}. \tag{7}$$

Note that, unlike the eigenvalues and eigenvectors which are only computed once for a given chip, *vector $\mathbf{T}_{steady}$ and matrix $\mathbf{H}$ are computed/build once for every change in power*, with a total time complexity of $O\left(N^2\right)$. Therefore, for a given t and with matrix $\mathbf{H}$ already built, the total time complexity for computing $T_k(t)$ for all $k = 1, 2, \ldots, N$ is $O\left(N^2\right)$.

There are additional *implementation improvements* that can reduce the execution time (not the complexity). For example, when building matrix $\mathbf{H}$ for every change in power, we can compute an auxiliary vector $\mathbf{T}_{\text{diff}} = \mathbf{T}_{\text{init}} - \mathbf{T}_{\text{steady}}$, allowing subtraction $T_{\text{init}\,j} - T_{\text{steady}_j}$ to be computed N times instead

Algorithm 1 Build auxiliary matrix $\mathbf{H}$

Input: Matrices $\mathbf{V}$ and $\mathbf{V}^{-1}$, eigenvalues, vectors $\mathbf{T}_{\text{init}}$ and $\mathbf{T}_{\text{steady}}$;
Output: Auxiliary matrix $\mathbf{H}$;
1: **for** $j = 1$ **to** N **do**
2: $\quad T_{\text{diff}\,j} \leftarrow T_{\text{init}\,j} - T_{\text{steady}\,j}$;
3: **end for**
4: **for** $i = 1$ **to** N **do**
5: $\quad \text{auxH}_i \leftarrow \sum_{j=1}^{N} z_{i,j} \cdot T_{\text{diff}\,j}$;
6: **end for**
7: **for** $k = 1$ **to** N **do**
8: $\quad$ **for** $i = 1$ **to** N **do**
9: $\quad\quad H_{k,i} \leftarrow v_{k,i} \cdot \text{auxH}_i$;
10: $\quad$ **end for**
11: **end for**
12: **return** Auxiliary matrix $\mathbf{H}$;

Algorithm 2 Compute transient temperatures at time t

Input: Auxiliary matrix $\mathbf{H}$, and time of interest t;
Output: Temperatures $T_k(t)$ for all $k = 1, 2, \ldots, N$;
1: **for** $i = 1$ **to** N **do**
2: $\quad \text{auxExp}_i \leftarrow e^{\lambda_i \cdot t}$;
3: **end for**
4: **for** $k = 1$ **to** N **do**
5: $\quad T_k(t) \leftarrow T_{\text{steady}\,k} + \sum_{i=1}^{N} \text{auxExp}_i \cdot H_{k,i}$;
6: **end for**
7: **return** $T_k(t)$ for all $k = 1, 2, \ldots, N$;

of N^2 times. Similarly, for a given t, we can compute an auxiliary vector $\left\{e^{\lambda_1 \cdot t}, e^{\lambda_2 \cdot t}, \ldots, e^{\lambda_N \cdot t}\right\}$, which allows this exponentiation to be computed only N times instead of N^2 times. A pseudo-code to build auxiliary matrix $\mathbf{H}$ is presented in Algorithm 1, and a pseudo-code to compute $T_k(t)$ for every k is presented in Algorithm 2.

VI. Computing Peaks in Transient Temperatures

In order to compute the peaks in the transient temperatures, we need to find the time instant $t^{\uparrow k}$ at which the temperature in node k is maximized. Because $T_k(t)$ results in a different expression for every k, the value of $t^{\uparrow k}$ is different for every node. Once $t^{\uparrow k}$ is known, we can compute $T_k\left(t^{\uparrow k}\right)$ to obtain the value of the peak in the transient temperature of node k. Since $T_k(t)$ depends on power vector $\mathbf{P}$, time $t^{\uparrow k}$ needs to be computed for every change in power.

Given that $T_k(t)$ is a summation of decaying exponential functions, we know that $T_k(t)$ is continuous and differentiable with respect to t for $t \geq 0$. We define the first-order derivative of $T_k(t)$ as $T'_k(t)$, which from Equation (7) is expressed as

$$T'_k(t) = \sum_{i=1}^{N} \lambda_i \cdot H_{k,i} \cdot e^{\lambda_i \cdot t}.$$

For example, Figure 3 shows the first-order derivative of the temperature on Core_2,1 for the simulations in Figure 2. As seen in Figure 3, when there is a peak in the temperature of node k, the slope of $T_k(t)$ changes sign and $T'_k(t)$ is equal to zero. Hence, a method to find a transient peak on node k is to solve $T'_k(t) = 0$. Given that $T_k(t)$ eventually reaches its steady-state temperature $T_{\text{steady}\,k}$, there is always at least one solution for $T'_k(t) = 0$ when $t \to \infty$. There can be several solutions with $T'_k(t) = 0$, and every solution in which $t \geq 0$ is a potential peak in the temperature on node k. A naive approach is to find all solutions and test the one that results in the highest temperature. Nevertheless, since $T_k(t)$ is a summation of decaying exponential functions, from control theory [10] we know that for any stable system in which all poles are less than or equal to zero, the maximum temperature (or overshoot in control theory [10]) happens always at the first peak. Applying the Laplace transform to Equation (7), we have

$$T_k(S) = \frac{T_{\text{steady}\,k}}{S} + \sum_{i=1}^{N} \frac{H_{k,i}}{S - \lambda_i},$$

where $T_k(S)$ is the Laplace transform of $T_k(t)$. Given that $\lambda_i \leq 0$ for all $i = 1, 2, \ldots, N$, all poles in $T_k(S)$ are less than or equal to zero, and hence the above conclusion holds. Thus, the only solution of interest for $T'_k(t) = 0$ is the closest solution to zero such that $t \geq 0$, and this solution is $t^{\uparrow k}$.

Unfortunately, equation $T'_k(t) = 0$ cannot be solved analytically. One approach for solving such an expression is to apply the Newton-Raphson method [4], for which we need the second-order derivative of $T_k(t)$, that is, $T''_k(t)$, expressed as

$$T''_k(t) = \sum_{i=1}^{N} \lambda_i^2 \cdot H_{k,i} \cdot e^{\lambda_i \cdot t}.$$

Then, starting from an initial guess for $t^{\uparrow k}$, defined as $t_0^{\uparrow k}$, the value of $t^{\uparrow k}$ is approximated iteratively. The value of $t^{\uparrow k}$ for the n-th iteration, defined as $t_n^{\uparrow k}$, is computed from $t_{n-1}^{\uparrow k}$. Particularly, $t_n^{\uparrow k}$ is computed as

$$t_n^{\uparrow k} = t_{n-1}^{\uparrow k} - \frac{T'_k\left(t_{n-1}^{\uparrow k}\right)}{T''_k\left(t_{n-1}^{\uparrow k}\right)} = t_{n-1}^{\uparrow k} - \frac{\sum_{i=1}^{N} \lambda_i \cdot H_{k,i} \cdot e^{\lambda_i \cdot t_{n-1}^{\uparrow k}}}{\sum_{i=1}^{N} \lambda_i^2 \cdot H_{k,i} \cdot e^{\lambda_i \cdot t_{n-1}^{\uparrow k}}}.$$

As initial guess we use $t_0^{\uparrow k} = 0$, such that the method converges to the first transient peak. Finally, $t^{\uparrow k}$ is set to $t_W^{\uparrow k}$, where W is a constant indicating the total number of iterations used when applying the Newton-Raphson method, and $T_k\left(t^{\uparrow k}\right)$ is computed through Equation (7). A pseudo-code to compute the peak temperatures on all nodes is presented in Algorithm 3.

There is the possibility that $T_k(t)$ is decreasing or increasing, that is, there is no peak in $T_k(t)$ and its maximum value happens at $T_k(0)$ or $T_{\text{steady}\,k}$, respectively. Thus, the actual maximum temperature to consider is

$$\max\left\{T_k\left(t^{\uparrow k}\right), T_k(0), T_{\text{steady}\,k}\right\}.$$

Moreover, if there is a new change in power at time t_{change}, the temperature on node k might never reach $T_{\text{steady}\,k}$. For such cases, the maximum temperature to consider is

$$\max\left\{T_k\left(t^{\uparrow k}\right), T_k(0), T_k\left(t_{\text{change}}\right)\right\},$$

and $T_{\text{init}\,k}$ for the next computation is set to $T_k\left(t_{\text{change}}\right)$.

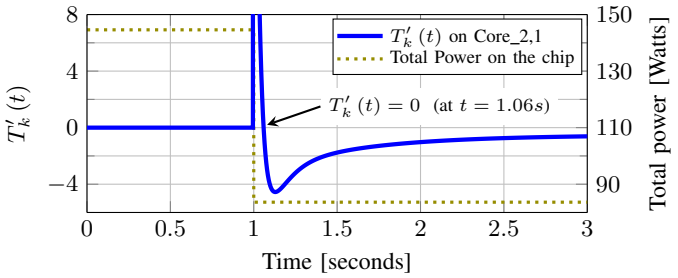


Fig. 3: First-order derivative of the temperature on Core_2,1 for the transient simulations in Figure 2. After the change in power, $T'_k(t)$ is zero at $t = 1.06s$ and when $t \to \infty$.

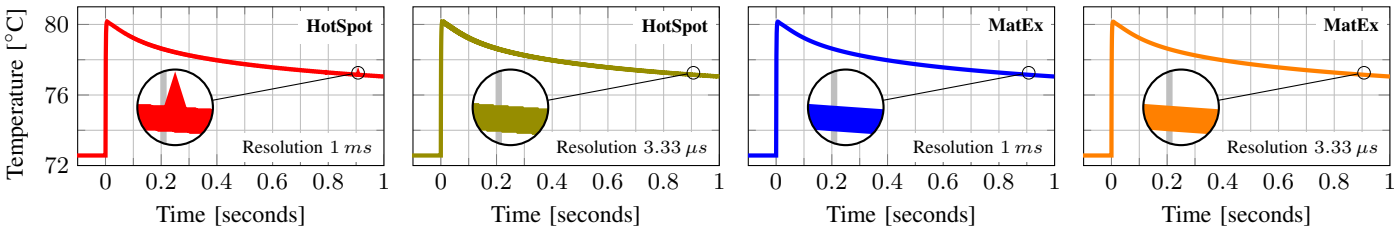


Fig. 4: Simulation results with all transient temperatures for the case of 64 cores and a single change in power.

Algorithm 3 Peak temperature computation

Input: Auxiliary matrix $\mathbf{H}$, and total number of iterations W;
Output: Peak temperatures $T_k\left(t^{\uparrow k}\right)$ for all $k = 1, 2, \ldots, N$;
1: **for** $k = 1$ **to** N **do**
2: $t^{\uparrow k} \leftarrow 0$;
3: **for** $n = 1$ **to** W **do**
4: $t^{\uparrow k} \leftarrow t^{\uparrow k} - \frac{\sum_{i=1}^{N} \lambda_i \cdot H_{k,i} \cdot e^{\lambda_i \cdot t^{\uparrow k}}}{\sum_{i=1}^{N} \lambda_i^2 \cdot H_{k,i} \cdot e^{\lambda_i \cdot t^{\uparrow k}}}$;
5: **end for**
6: **end for**
7: **return** $T_k\left(t^{\uparrow k}\right)$ for all $k = 1, 2, \ldots, N$;

Algorithms 1, 2, and 3 are implemented as an open-source tool called MatEx (from matrix exponentials), available for download at http://ces.itec.kit.edu/download.

VII. Evaluations

In this section, we evaluate the accuracy and the performance of MatEx, compared to the widely-adopted HotSpot [7].

A. Setup

The compact thermal model, that is, the values of matrix $\mathbf{A}$, matrix $\mathbf{B}$, and vector $\mathbf{G}$, are computed through HotSpot and given to MatEx as inputs. As MatEx is not tied to HotSpot, *any other* compact modeling tool can be used to derive the RC thermal network. We use HotSpot in our evaluations mainly to compare the accuracy of MatEx with a widely-used tool.

Inside MatEx, Algorithm 2 is used to compute all transient temperatures for a given time resolution, and Algorithm 3 is used to compute the peaks in transient temperatures generated by the changes of power inside the chip. Therefore, we conduct two separate experiments, one to test each algorithm. When testing Algorithm 2, we need to specify the time resolution, that is, the time steps between every transient computation. Particularly, we consider two resolutions: $3.333\,\mu s$ and $1\,ms$, and these same time resolutions are used for the simulations with HotSpot. When evaluating Algorithm 3 we consider 20 iterations for the Newton-Rhapson method. The length for the power trace is set to $1\,s$. Given that MatEx needs to build auxiliary matrix $\mathbf{H}$ for every change in power, a higher number of power changes results in longer execution time for MatEx. Hence, to account for two different cases, the power traces are generated randomly considering 1 power change and 100 power changes[2]. With respect to the number of cores in the chip, we consider 4 different cases in which the chip has 16 cores, 32 cores, 48 cores, and 64 cores. For these numbers of cores, HotSpot generates RC thermal networks with 76 nodes, 140 nodes, 204 nodes, and 268 nodes, respectively.

[2]We intentionally use synthetic traces in order to have two very distinctive numbers of power changes. Where we to use power traces from real applications, the effects due to the number of power changes would be harder to observe, given that they would be imposed by the application and architecture.

B. Results

Figure 4 presents the experimental results comparing the accuracy of both solutions, for the case of 64 cores and a single change in power[3]. Given that HotSpot uses the fourth-order *Runge-Kutta* method to solve the system of differential equations, Figure 4 shows (zooming) that there are some accuracy losses when the time resolution decreases. Specifically, by taking $3.333\,\mu s$ as reference (most accurate case for HotSpot), we observe some jitter in HotSpot's transient output when the time resolution is $1\,ms$. This happens because for each new step, the *Runge-Kutta* method computes a difference from the previous result, and a smaller resolution introduces more error. Contrarily, *the accuracy of MatEx is entirely independent on the time resolution used for computing all transient temperatures*. This comes from the fact that MatEx computes the temperature through Equation (7) for any given time t, and the value of the previous temperature computation is not necessary. Therefore, when running full transient simulations, *the user can adapt the time resolution in MatEx for any specific needs, without additional considerations about accuracy losses*.

Figure 5 and Figure 6 present the execution time results of HotSpot and MatEx (Algorithm 1 and Algorithm 2), respectively, for the experiment that computes all transient temperatures, for all the cases described in Section VII-A. For each tool, a higher time resolution results in longer execution times. In HotSpot, execution time is only affected by the time resolution, and not by the number of power changes. In MatEx, aside from the effect of the time resolution, the number of power changes can also slightly increase the execution time. In our experiment, for a resolution of $1\,ms$ Algorithm 2 is executed 1000 times, while Algorithm 1 is executed 100 times (100 power changes) or only once (1 power change), thus the effect of the number of power changes is noticeable. However, for a resolution of $3.333\,\mu s$ Algorithm 2 is executed $3 \cdot 10^5$ times, masking the execution time of Algorithm 1. The most important fact is that *MatEx is always faster than HotSpot for the same time resolution*. Specifically, *MatEx is on average* 40 *times faster than HotSpot for all the evaluated cases, being up to* 100 *times faster for the case with* 64 *cores, one power change, and time resolution of* $1\,ms$. This enables MatEx to be used not only for peak temperature estimations, but also to replace HotSpot for general transient temperature computation.

Figure 7 presents the execution time results of MatEx (Algorithm 1 and Algorithm 3) for the experiment that only computes the peaks in the transient temperatures, for all the cases described in Section VII-A. Because HotSpot (or any other numerical method) is unable to analyze the peaks in temperature without simulating the entire trace, the execution time of HotSpot for this experiment is the same as when computing all transient temperatures and these values are

[3]Other results are omitted due to space constraints. Nevertheless, similar observations apply to all other cases.

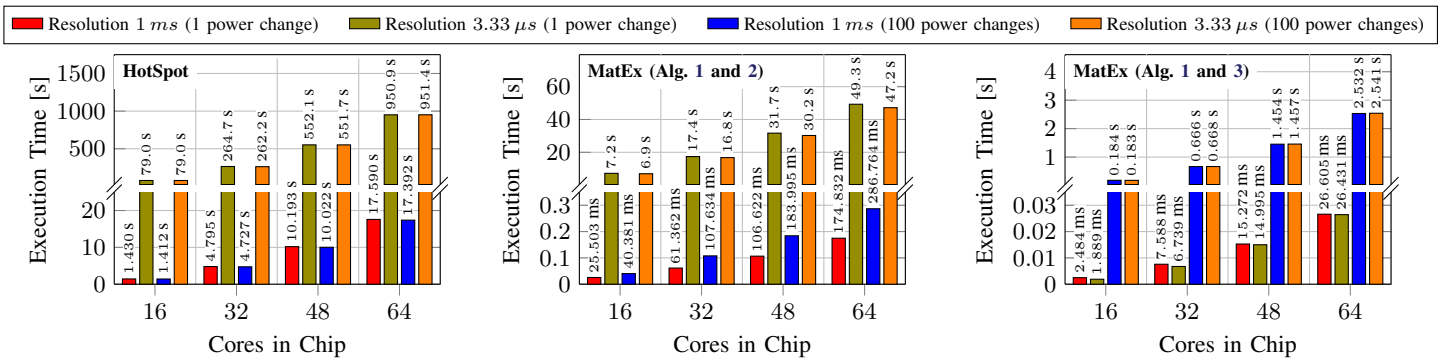


Fig. 5: Execution time for computing *all* transient temperatures and the *peaks* in the transient temperatures *using HotSpot*.

Fig. 6: Execution time for computing *all* transient temperatures *using MatEx* (Algorithm 1 and Algorithm 2).

Fig. 7: Execution time for computing the *peaks* in the transient temperatures *using MatEx* (Algorithm 1 and Algorithm 3).

already presented in Figure 5 (HotSpot was trivially modified to keep track of the highest transient temperature incurring in negligible overhead). On the other hand, MatEx (Algorithm 3) uses the Newton-Raphson method to compute the time instants of the peaks in temperature. Particularly, for a system with 16 cores, Figure 7 shows that the execution time of MatEx when having *one power change is only up to* $2.5\,ms$, which is suited for *run-time* scheduling decisions. The results also show that for peak temperature computation the number of power changes has a linear impact in the execution time of Algorithm 3. Thus, when we have 100 power changes the execution time of Algorithm 3 is around 100 times longer than when having a single power change.

VIII. Conclusions

In this paper we have presented a new method to efficiently compute peaks in transient temperatures, as well as all transient temperatures for any given time resolution. This new method is based on matrix exponentials and we have integrated it in an open-source tool, called MatEx. Experimental evaluations were conducted to compare the efficiency and accuracy of MatEx with that of HotSpot. With respect to the computation of all transient temperatures, our results show that, while in HotSpot the time resolution affects the accuracy, in MatEx the resolution can be freely chosen without additional considerations about accuracy losses. Furthermore, the execution time of MatEx for the same time resolution is up to 100 times faster than HotSpot. For the computation of the peaks in temperature, the execution time of MatEx is just a few milliseconds for all the evaluated cases with only one change in power. This suggests that MatEx can be used for *run-time* computation of peaks in temperature for scheduling and task migration decisions. In this way, MatEx can efficiently help to prevent undesired triggers of DTM or damages to the chip when DTM would be unable to react fast enough.

Acknowledgements

This work was partly supported by the German Research Foundation (DFG) as part of the Transregional Collaborative Research Centre *Invasive Computing* [SFB/TR 89].

References

[1] "Eigen C++ template library (v3.2.2)," 2014, http://eigen.tuxfamily.org.

[2] ANSYS Inc., "ANSYS R15.0," 2014, http://www.ansys.com.

[3] E. K. Ardestani and J. Renau, "ESESC: A fast multicore simulator using time-based sampling," in *HPCA*, 2013, pp. 448–459.

[4] J. F. Bonnans, J. C. Gilbert, C. Lemaréchal, and C. A. Sagastizábal, *Numerical Optimization: Theoretical and Practical Aspects*, 2nd ed. Springer-Verlag, 2006.

[5] X. Hu, Y. Xu, J. Ma, G. Chen, Y. Hu, and Y. Xie, "Thermal-sustainable power budgeting for dynamic threading," in *DAC*, 2014, pp. 187:1–187:6.

[6] P.-Y. Huang and Y.-M. Lee, "Full-chip thermal analysis for the early design stage via generalized integral transforms," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 17, no. 5, pp. 613–626, May 2009.

[7] W. Huang, S. Ghosh, S. Velusamy, K. Sankaranarayanan, K. Skadron, and M. Stan, "HotSpot: a compact thermal modeling methodology for early-stage VLSI design," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 14, no. 5, pp. 501–513, May 2006.

[8] Intel Corporation, "Dual-core intel xeon processor 5100 series datasheet, revision 003," August 2007.

[9] S. Klab, A. Napieralski, and G. De Mey, "Logi-thermal simulation of digital CMOS ICs with emphasis on dynamic power dissipation," in *MIXDES*, 2009, pp. 361–365.

[10] B. Kuo and F. Golnaraghi, *Automatic Control Systems*, 8th ed. John Wiley & Sons, 2002.

[11] Y.-M. Lee, T.-H. Wu, P.-Y. Huang, and C.-P. Yang, "NUMANA: A hybrid numerical and analytical thermal simulator for 3-D ICs," in *DATE*, 2013, pp. 1379–1384.

[12] S. Li, J.-H. Ahn, R. Strong, J. Brockman, D. Tullsen, and N. Jouppi, "McPAT: An integrated power, area, and timing modeling framework for multicore and manycore architectures," in *MICRO*, 2009, pp. 469–480.

[13] H. M. Moya-Cessa and F. Soto-Eguibar, *Differential Equations: An Operational Approach*. Rinton Press, 2011.

[14] T. S. Muthukaruppan, M. Pricopi, V. Venkataramani, T. Mitra, and S. Vishin, "Hierarchical power management for asymmetric multi-core in dark silicon era," in *DAC*, 2013, pp. 174:1–174:9.

[15] H. Qian, H. Liang, C.-H. Chang, W. Zhang, and H. Yu, "Thermal simulator of 3D-IC with modeling of anisotropic TSV conductance and microchannel entrance effects," in *ASP-DAC*, 2013, pp. 485–490.

[16] D. Rai, H. Yang, I. Bacivarov, and L. Thiele, "Power agnostic technique for efficient temperature estimation of multicore embedded systems," in *CASES*, 2012, pp. 61–70.

[17] H. Wang, S. X.-D. Tan, D. Li, A. Gupta, and Y. Yuan, "Composable thermal modeling and simulation for architecture-level thermal designs of multicore microprocessors," *TODAES*, vol. 18, no. 2, pp. 28:1–28:27, Apr. 2013.

[18] T.-Y. Wang and C. Chen, "3-D Thermal-ADI: a linear-time chip level transient thermal simulator," *TCAD*, vol. 21, no. 12, pp. 1434–1445, Dec 2002.

[19] Y. Zhan and S. S. Sapatnekar, "Fast computation of the temperature distribution in VLSI chips using the discrete cosine transform and table look-up," in *ASP-DAC*, 2005, pp. 87–92.

[20] A. Ziabari, J.-H. Park, E. Ardestani, J. Renau, S.-M. Kang, and A. Shakouri, "Power blurring: Fast static and transient thermal analysis method for packaged integrated circuits and power devices," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 2014.