Heterogeneity Exploration for Peak Temperature Reduction on Multi-Core Platforms

Tianyi Wang[1], Ming Fan[1], Gang Quan[1], and Shangping Ren[2]
[1]Department of Electrical&Computer Engineering, Florida International University, Miami, FL USA
[2]Department of Computer Science, Illinois Institute of Technology, Chicago, IL USA
E-mail[1]: {tiawang, mfan001, gaquan}@fiu.edu
E-mail[2]: ren@iit.edu

***Abstract*—As IC technology continues to evolve and more transistors are integrated into a single chip, high chip temperature due to high power density not only increases packaging/cooling cost, but also severely degrades reliability and the performance of computing systems. In the meantime, as transistor feature size continues to shrink, it becomes difficult to precisely control the manufacturing process. The manufacturing variations can cause significant differences from core to core and chip to chip. We believe that the heterogeneity due to manufacturing variations, if handled properly, can in fact improve the design objectives of real-time applications. In this paper, we study the problem on how to reduce the peak temperature of a real-time application by judiciously mirroring the physical architecture of an individual device to the logical architecture where the application was initially designed upon. We develop three computationally efficient algorithms for deploying applications to individual devices. Our simulation study has clearly shown that, by taking advantage of the uniqueness of each individual physical chip, the proposed approaches significantly reduce the peak temperature. The experiments also show that these approaches are efficient and have low operational cost.**

***Keywords*—peak temperature; manufacturing variations; topology virtualization; nominal design; multi-core**

I. Introduction

With billions of transistors integrated on a single chip to further drive the pace of multi-core design, high peak temperature has increasingly become a critical issue in computer system design. High chip temperature not only increases packaging/cooling cost (estimated at 1-3 dollar per watt [1]) but also significantly degrades system performance and reliability. A $10°C$ to $15°C$ increase of operation temperature can reduce the lifetime of a chip by half [2], [3]. Moreover, high chip temperature dramatically increases leakage power dissipation. The leakage power dissipation of a chip can be tripled when temperature increases from $45°C$ to $110°C$ according to [4], which in turn will further elevate temperature. Temperature constraint is becoming the first-class design concern for digital CMOS ICs.

In the meantime, as transistors become increasingly smaller, manufacturing variations become more and more substantial. As the chip's feature size continues to shrink, to the level below the wavelength of light used to print them, it becomes very difficult to precisely control the manufacturing process [5], [6]. ITRS [7] in 2008 predicted that circuit variability will increase from 48% to 66% in the next ten years. When temperature changes from $20°C$ to $60°C$, as much as 10% variation in dynamic power and 14x variation in leakage power are measured for an ARM Cortex M3 processor [8]. As such, designers can no longer assume their systems designed based on the nominal design parameters can always yield the expected performance. For real-time systems, the design based on the worst case scenarios become unacceptable, as the worst cases can be orders of magnitude different from the nominal values [9].

Many approaches have been proposed to deal with manufacturing variation problems. Significant achievements have been made on layout techniques and other device technologies by adding built-in sensors or redundant devices [6], [10], [11]. However, it becomes increasingly challenging as transistor size scales towards dimensions close to or below 10 nm [12]. Besides extensive work on layout and device level, there are increasing research efforts to address the manufacturing variation problem from architecture and system level. For example, performance binning technique is proposed to cluster chips with similar performance [13], [14]. However, this approach cannot deal with manufacturing variations of different cores within the same chip. Another popular approach is to adopt the statistical approach in the design optimization process. As an example, Wang et al. [15] proposed a task mapping and scheduling algorithm to maximize the performance yield rate (i.e., the probability that a processor can meet the desired performance of a given application) by statistically taking both variations of cores and physical links into account. These statistical approaches try to optimize results in a probabilistic manner. When deploying the design to each individual processor, the designs can be either too optimistic or too pessimistic due to different performance variations. One recent work [16] exploited process variations in Dark-Silicon homogeneous chip multi-processors, however they only considered the frequency variations and ignored leakage variations between cores.

There is another interesting approach proposed to address the manufacturing variation problem. This approach, so called *topology virtualization* [17], calls for judiciously mirroring the physical architecture of an individual device to the logic architecture of an application when the application is deployed (installed/initiated) to the device. Figure 1 illustrates this approach.

This work is supported in part by NSF under projects CNS-0969013, CNS-0917021, CNS-1018108, CNS-1018731, and CNS-0746643.

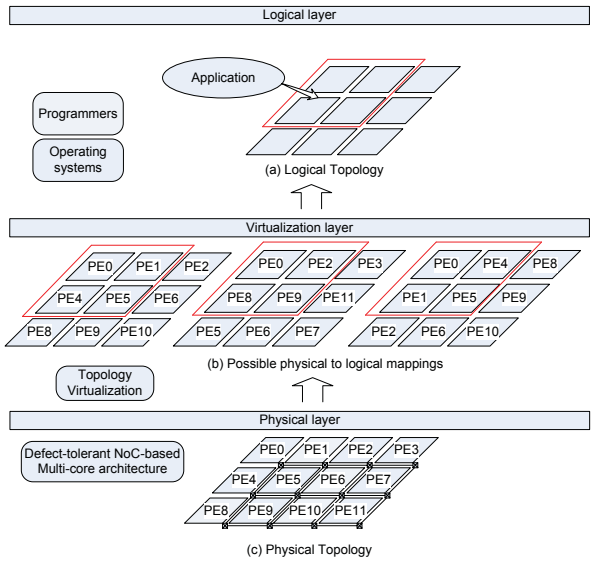

Fig. 1. The topology virtualization framework. (a) Target application designed based on the nominal parameters of a 3×3 logical topology; (b) Configure the practical topology of an individual processor differently to mirror the logical topology and optimize the performance of the target application; (3) The physical topology, a 3×4 mesh, for the processor.

Assume an application is developed based on a 3×3 logical homogeneous multi-core mesh architecture. The physical chip, on the other hand, is not necessarily the same 3×3 architecture. To fight for manufacture variation problem, it has been a common practice in industry to add redundant resources (e.g. processing cores) so that the entire chip can still work even if some cores are faulty [10], [6]. Assume the physical architecture of a chip is a mesh of 3×4 as shown in Figure 1. Note that, due to the manufacturing variation problem, the performance of all these cores are not necessarily homogeneous. Moreover, the design based on the logical architecture does not necessarily utilize each processor in exactly the same way. Opportunity is thus presented to optimize the performance of original nominal design by mirroring the physical architecture of each individual chip to the logical architecture differently as shown in Figure 1. There are a number of distinct advantages to this approach. First, compared with the statistical approach, this approach can exploit the unique characteristic of each individual device and better optimize the system performance. Second, the architecture changes can be managed by the operating system or lower level software such as BIOS, which is totally transparent to the application software.

We believe that heterogeneity due to manufacturing variations, if explored properly, can in fact improve the design objective of a real-time application. In this paper, we study the problem of how to reduce the peak temperature by exploiting the architecture heterogeneity due to manufacturing variations. A few works is closely related to our approach proposed in this paper. When a processor has faulty cores and more redundant cores, Zhang et al. [18], [19] proposed several heuristics to replace faulty cores with redundant cores to improve the fabrication yield. They further extended their work to deal with performance variations by constructing sub-meshes using cores with similar performance [20]. Yue at el. [21], [22] improved upon Zhang's work [18], [19] by taking application characteristics into consideration, and intended to maintain the similar real-time performance after replacing faulty cores with redundant cores. A more recent work by Wang et al. [17] considered manufacturing variations on homogeneous multi-core platforms, proposed three re-mapping heuristics to maximize the throughput of task graph.

It is not difficult to see that the problem to optimize the performance of an application by mirroring the physical topology to the logical topology is an NP-hard problem [23]. While it is a common practice to use certain time-consuming techniques such as Genetic Algorithm (GA) [24], [25], Simulated Annealing Algorithm (SAA) [18], [19] to solve this problem, this is not viable for the *topology virtualization* approach. Since the physical to logical topology mapping is performed when deploying (installing or initiating) the application software on an individual device, the key to the success of this approach is to develop computationally efficient mapping methods that can effectively optimize the performance metrics for application software. To this end, we developed three physical to logical topology mapping heuristics to reduce the peak temperature of a processor. Our simulation study shows that the *topology virtualization* approach is very effective in reducing the peak temperature for a processor.

In what follows, we introduce the system, power and thermal models the research is based upon, and formulate the problem we are to address in Section II. In Section III, We discuss how to rapidly calculate the peak temperature for a periodic application . We then present three computationally efficient heuristics to minimize peak temperature in Section IV. Experimental results are discussed in Section V, and we conclude in Section VI.

II. Preliminary

In this section, we first introduce the system model, power and thermal models and then formulate the research problem we are to address in the paper.

A. System models

The multi-core platform considered in this paper consists of identical cores with traditional 2-D mesh architecture. Each core can run in one of r different operating modes. Each running mode is characterized by a tuple (v_k, f_k) $(1 \leq k \leq r)$, where v_k is the supply voltage and f_k is the working frequency for mode k, respectively. Note that due to manufacture variations, cores may have different maximum frequencies deviated from its nominal value. Therefore, cores that are running with the same supply voltage mode, their frequencies also can behave differently. In Figure 2 shows an example of two cores with their voltage-frequency modes compared to the nominal values (parameters are generated based on [6]).

Specifically, we assume the *physical architecture* (denoted as $A^p_{m\times n}$) is an $m \times n$ mesh, i.e.

$$A^p_{m\times n} = \{\mathcal{A}^p_{i,j}, i = 0,...,m-1; j = 0,...,n-1\}. \quad (1)$$

where $\mathcal{A}^p_{i,j}$ represents the core located at position (i, j) in the physical topology.

The target application is periodic with period of L. We assume the original *nominal design* of the application is based

Nominal Value (V, F)	Core 0	Core 1
(0.8, 0.8)	(0.8, 0.76)	(0.8, 0.82)
(0.9, 0.9)	(0.9, 0.87)	(0.9, 0.93)
(1.0, 1.0)	(1.0, 0.95)	(1.0, 1.04)
(1.1, 1.1)	(1.1, 1.03)	(1.1, 1.13)
(1.2, 1.2)	(1.2, 1.18)	(1.2, 1.25)

Fig. 2. Voltage-Frequency variation example between cores.

on a *logical architecture* (denoted as $A^l_{x\times y}$), which forms a standard $x \times y$ homogeneous mesh architecture, i.e.

$$A^l_{x\times y} = \{\mathcal{A}^l_{i,j}, i = 0,...,x-1; j = 0,...,y-1\}. \tag{2}$$

where $\mathcal{A}^l_{i,j}$ represents the core located at position (i, j) in the logical topology. The *nominal design* $\mathbf{S}(t)$ consists of a set of static, periodic voltage schedules, each of which, i.e. $\mathbf{S}_i(t)$, is applied to one logical core and dictates the change of its processing speed in each period. We assume that each schedule consists of a set of non-overlapping intervals with total length of L. Each interval has its own specified running modes. Let the voltage schedule on core i, denoted as $\mathbf{S}_i(t)$, consist of a set of interval $[t_0,t_1],...,[t_{q-1},t_q]$ such that $\bigcup_{q=1}^{s}[t_{q-1},t_q] = [0,\ L]$, and $[t_{q-1},t_q]\bigcap[t_{p-1},t_p] = \emptyset,$ if $q \neq p$. Also, let the running modes of interval i be $[v_i, f_i]$. We define the *utilization* of core i, denoted as U_i as

$$U_i = \frac{\sum_i (t_i - t_{i-1}) \times v_i}{v_{max} \times L}. \tag{3}$$

B. Power and thermal models

The total power dissipation of each core contains two parts: dynamic power and leakage power. We assume that the dynamic power is independent of temperature but sensitive to variation while the leakage power is sensitive to both. The total power dissipation of core i, denoted as P_i, is formulated as:

$$P_i = P_i^{dym} + P_i^{leak}, \tag{4}$$

$$P_i^{dym} = \gamma_{k_i} \cdot v_{k_i}^2 \cdot f_{k_i}, \tag{5}$$

$$P_i^{leak} = (\alpha_{k_i} + \beta_{k_i} \cdot T_i(t)) \cdot (v_{k_i} + \Delta^i_{leak}). \tag{6}$$

where α_{k_i}, β_{k_i} and γ_{k_i} are constants that depend on the mode k_i of core i. Δ^i_{leak} is a given constant that models the leakage power variations due to the impact of die-to-die (D2D) and within-die (WID) process variation [26].

We use the RC network to model the thermal behavior of a multi-core platform, same to that in [27], [28]. Let C_i and R_{ij} denote the thermal capacitance (in $Watt/^\circ C$) of core i and thermal resistance (in $J/^\circ C$) between core i and j, respectively. The thermal behavior of the i^{th} core can be formulated as

$$C_i \cdot \frac{dT_i(t)}{dt} + \frac{T_i(t)}{R_{ii}} + \sum_{j\neq i} \frac{T_i(t) - T_j(t)}{R_{ij}} = P_i(t) \tag{7}$$

Incorporating equation (4) in the above equation, we have

$$C_i \cdot \frac{dT_i(t)}{dt} + G_{ii} \cdot T_i(t) + \sum_{j\neq i} G_{ij} \cdot T_j(t) = \Psi_i \tag{8}$$

where

$$G_{ij} = \begin{cases} \sum_{j=1}^{m} \frac{1}{R_{ij}} - \beta_{k_i} \cdot v_{k_i} - \beta_{k_i}\Delta^i_{leak} & \text{if } i = j \\ \frac{-1}{R_{ij}} & \text{otherwise} \end{cases} \tag{9}$$

and

$$\Psi_i = \alpha_{k_i} \cdot v_{k_i} + \alpha_{k_i}\Delta^i_{leak} + \gamma_{k_i} \cdot v_{k_i}^2 \cdot f_{k_i} \tag{10}$$

Let $\mathbf{T}_{amb}$ denote the ambient temperature. We thus have

$$\mathbf{C}\frac{d\mathbf{T}(t)}{dt} + \mathbf{G}\big(\mathbf{T}(t) - \mathbf{T}_{amb}\big) = \Psi \tag{11}$$

where $\mathbf{C}$ and $\mathbf{G}$ are $m \times m$ matrices.

$$\mathbf{C} = \begin{bmatrix} C_1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & C_m \end{bmatrix} \quad \mathbf{G} = \begin{bmatrix} G_{11} & \cdots & G_{1m} \\ \vdots & \ddots & \vdots \\ G_{m1} & \cdots & G_{mm} \end{bmatrix} \tag{12}$$

and $\mathbf{T}(t)$ and Ψ are $m \times 1$ vectors

$$\mathbf{T(t)} = \begin{bmatrix} T_1(t) \\ \vdots \\ T_m(t) \end{bmatrix} \quad \Psi = \begin{bmatrix} \Psi_1 \\ \vdots \\ \Psi_m \end{bmatrix} \tag{13}$$

C. Problem formulation

With all models discussed above, our problem is to minimize peak temperature while guaranteeing timing constraints. It is worth pointing out that the nominal design **S(t)** is based on manufacture-variation-free scenario, hence, when we apply the nominal design to each individual chip which may be affected by manufacture variations, we may not be able to guarantee timing constraints if no appropriate actions are taken. We formally define the research problem below.

Problem 1: Given

- a physical topology of a multi-core platform $A^p_{m\times n}$, r different processor modes and leakage variation Δ_{leak} for each core;
- a logical topology $A^l_{x\times y}$;
- a nominal design **S(t)**,

determine the physical to logical topology mapping such that the chip peak temperature of the chip is minimized when running **S(t)** on $A^p_{m\times n}$ and all timing constraints are also met.

III. Temperature dynamics formulation

Our goal is to minimize the peak temperature when running the *nominal design* on the practical processor. To this end, it is necessary that we can quickly identify the exact peak temperature when running a periodic schedule. In what follows, we introduce a method to quickly calculate the peak temperature for a periodic voltage schedule on multi-core platforms.

Consider an interval $[t_{q-1},t_q]$ and assume the supply voltages or working frequencies of all cores remain the same within the interval. Let κ_q represent the specific running modes of all cores in interval $[t_{q-1},t_q]$. Then based on equation (11), we have

$$\left.\frac{d\mathbf{T}(t)}{dt}\right|_{t\in[t_{q-1},t_q]} = \mathbf{A}_{\kappa_q}\big(\mathbf{T}(t) - T_{amb}\big) + \mathbf{B}_{\kappa_q} \tag{14}$$

where $\mathbf{A}_{\kappa_q} = -\mathbf{C}^{-1}\mathbf{G}_{\kappa_q}$ and $\mathbf{B}_{\kappa_q} = \mathbf{C}^{-1}\Psi_{\kappa_q}$. Since $\mathbf{A}_{\kappa_q}$ and $\mathbf{B}_{\kappa_q}$ are constant, equation (14) is simply a first-order constant coefficient ordinary differential equations (ODEs) with the following solution:

$$\mathbf{T}(t_q) = e^{\mathbf{A}_{\kappa_q}\Delta t_q}\big(\mathbf{T}(t_{q-1}) - T_{amb}\big) + \mathbf{A}_{\kappa_q}^{-1}(e^{\mathbf{A}_{\kappa_q}\Delta t_q} - I)\mathbf{B}_{\kappa_q} + T_{amb} \tag{15}$$

where $\Delta t_q = t_q - t_{q-1}$. Therefore, given a state interval, its ending temperature can be determined by the starting temperature $\mathbf{T}(t_{q-1})$ and the corresponding interval mode κ_q.

With equation (15), given a periodic schedule $\mathbf{S}(t)$ and the initial temperature $\mathbf{T}(0)$, we can calculate the temperature at any time instant by tracing temperature from one interval to another. However, it can be computational costly to trace the temperature until it reaches the steady state. It is also desirable to calculate the stable temperature by setting $\frac{d\mathbf{T}(t)}{dt} = 0$. This works if all cores run at a constant speed schedule, but does not work anymore for a periodic schedule with different running modes. In what follows, we present a fast method to identify the peak temperature for a periodic schedule $\mathbf{S}(t)$.

Let us first consider the temperature variation at the end of each period, i.e. $t = nL$. Let the scheduling points of $\mathbf{S}(t)$ in the first period be $t_0, t_1, ..., t_{s-1}$, respectively. We assume that the running modes for all cores remain unchanged between two neighboring scheduling points. Similarly, let the *corresponding* scheduling points in the second period be $t'_0, t'_1, ..., t'_{s-1}$, respectively. Note that $t_0 = 0, t'_0 = t_s = L$ and $t'_s = 2L$. According to equation (15), at time t_1 and t'_1, we have

$$\mathbf{T}(t_1) = e^{\mathbf{A}_{\kappa_1}\Delta t_1}\big(\mathbf{T}(t_0) - T_{amb}\big) + \mathbf{A}_{\kappa_1}^{-1}(e^{\mathbf{A}_{\kappa_1}\Delta t_1} - \mathbf{I})\mathbf{B}_{\kappa_1} + T_{amb} \tag{16}$$

$$\mathbf{T}(t'_1) = e^{\mathbf{A}_{\kappa_1}\Delta t'_1}\big(\mathbf{T}(t'_0) - T_{amb}\big) + \mathbf{A}_{\kappa_1}^{-1}(e^{\mathbf{A}_{\kappa_1}\Delta t'_1} - \mathbf{I})\mathbf{B}_{\kappa_1} + T_{amb} \tag{17}$$

Subtract equation (16) from (17), and simplify the result by applying $\Delta t'_1 = \Delta t_1$, $t_0 = 0$ and $t'_0 = L$, we get

$$\mathbf{T}(t'_1) - \mathbf{T}(t_1) = e^{\mathbf{A}_{\kappa_1}\Delta t_1}(\mathbf{T}(L) - \mathbf{T}(0))$$

Similarly, we can derive that

$$\begin{aligned} \mathbf{T}(t'_2) - \mathbf{T}(t_2) &= e^{\mathbf{A}_{\kappa_2}\Delta t_2}e^{\mathbf{A}_{\kappa_1}\Delta t_1}(\mathbf{T}(L) - \mathbf{T}(0)) \\ &... \\ \mathbf{T}(t'_s) - \mathbf{T}(t_s) &= e^{\mathbf{A}_{\kappa_s}\Delta t_s}...e^{\mathbf{A}_{\kappa_1}\Delta t_1}(\mathbf{T}(L) - \mathbf{T}(0)) \end{aligned} \tag{18}$$

Since $t_s = L$, $t'_s = 2L$, and let $K = e^{\mathbf{A}_{\kappa_s}\Delta t_s}...e^{\mathbf{A}_{\kappa_1}\Delta t_1}$, equation (18) can be rewritten as

$$\mathbf{T}(2L) - \mathbf{T}(L) = \mathbf{K}(\mathbf{T}(L) - \mathbf{T}(0)) \tag{19}$$

Similarly, we have

$$\mathbf{T}(xL) - \mathbf{T}((x-1)L) = \mathbf{K}^{x-1}(\mathbf{T}(L) - \mathbf{T}(0)) \tag{20}$$

where $x = 1, 2, ..., n$. Sum up these n equations, we get

$$\mathbf{T}(nL) = \mathbf{T}(0) + (\sum_{x=1}^{n}\mathbf{K}^{x-1})(\mathbf{T}(L) - \mathbf{T}(0)) \tag{21}$$

In the above, $\{\mathbf{K}^{x-1} | x = 1, 2, ..., n\}$ forms a matrix geometric series. If $(\mathbf{I} - \mathbf{K})$ is invertible, then we have

$$\mathbf{T}(nL) = \mathbf{T}(0) + (\mathbf{I} - \mathbf{K})^{-1}(\mathbf{I} - \mathbf{K}^n)(\mathbf{T}(L) - \mathbf{T}(0)) \tag{22}$$

Similarly, for any time instant $t = nL + t_q$, we can get that

$$\mathbf{T}(nL + t_q) = \mathbf{T}(t_q) + \mathbf{K}_q(\mathbf{I} - \mathbf{K})^{-1}(\mathbf{I} - \mathbf{K}^n)(\mathbf{T}(L) - \mathbf{T}(0)) \tag{23}$$

where $\mathbf{K}_q = e^{\mathbf{A}_{\kappa_q}\Delta t_q} \cdot e^{\mathbf{A}_{\kappa_{q-1}}\Delta t_{q-1}}...e^{\mathbf{A}_{\kappa_1}\Delta t_1}$. Equation (23) can be used to quickly calculate the temperature at $t = nL + t_q$, where $n \geq 1$ and $t_q \in [0, L]$. Moreover, let $n \to \infty$ in equation (23), we can quickly identify the steady-state temperature corresponding to t_q as

$$\mathbf{T}_{ss}(t_q) = \mathbf{T}(t_q) + \mathbf{K}_q(\mathbf{I} - \mathbf{K})^{-1}(\mathbf{T}(L) - \mathbf{T}(0)) \tag{24}$$

From equation (24), given a periodic schedule $\mathbf{S}(t)$, we can formulate the system steady-state temperature with information of the first period directly, which is much more efficient than to keep track of temperature variations based on equation (15).

IV. Physical to logical mapping heuristics

Before we introduce our mapping heuristics, we want to first guarantee that the timing constraints are met after re-mapping. We make one assumption that the highest frequency in $\mathbf{S}(t)$ can always be no greater than the maximum frequency supported by the core on which it is mapped. Under this assumption, we adjust the core's voltage-frequency mode such that the timing constraint satisfaction can be guaranteed. Specifically, we have two solutions for each interval if the current running mode cannot satisfy its nominal design parameter, 1) we change it to the next neighbor running mode or 2) we use the two neighboring speeds alternatively until the timing constraints are met.

Now we present three mapping approaches to solve Problem 1 as defined above. It is not difficult to prove that Problem 1 is in fact NP-hard. As mentioned before, while common approaches such as GA or SAA are commonly used to guide mapping decisions during the design phases, they are not applicable in *topology virtualization* approach due to their high timing complexities. Since mapping decisions must be made when installing or initiating the application, the key to the success of this approach is the computation efficiency of the mapping algorithms and their effectiveness. In what follows, we develop three heuristics and study their effectiveness.

A. A simple utilization/leakage matching heuristic

Our goal is to map the physical topology to the logical topology such that an existing nominal design can be improved in terms of peak temperature in the presence of core heterogeneity. As we discussed in Section II-B, leakage variation is one of the key factors that affects temperature. An intuitive approach is therefore to match the logical core with the largest utilization to the least leaky physical core. The rationale behind this approach is that, when the larger utilization schedule is assigned to less leaky core, the less heat it generates. For example, consider two cores with identical voltage schedule (i.e., same utilization). The heat contributed by dynamic power is the same for both cores, but the one that is more leaky will generate more heat due to leakage power and therefore higher temperature. The algorithm is presented in Algorithm 1.

Algorithm 1 sorts the logical cores based on the assigned utilizations and the physical cores based on their leakage variations. Then, a physical core is mapped one by one to a

Algorithm 1 A simple heuristic to match high utilization logical core to low leaky physical core.

1: $\mathcal{M} = \emptyset$; // $\mathcal{M}$ is the mapping solution space
2: LC = The sorted list of $\mathcal{A}^l_{i,j} \in A^l_{x\times y}, i = 0,...,x-1; j = 0,...,y-1$ by their utilizations based on nominal design in decreasing order;
3: LP = The sorted list of $\mathcal{A}^p_{i,j} \in A^p_{m\times n}$ by Δ^{ij}_{leak}, $i = 0,...,m-1; j = 0,...,n-1$ in increasing order;
4: **for** $i = 0$ to $sizeof(LC)-1$ // for each logical core in the sorted list **do**
5: **if** The utilization assigned to $LC(i) > 0$ **then**
6: $\mathcal{M} = \mathcal{M} + \{LC(i) \rightarrow LP(i)\}$;
7: **end if**
8: **end for**

logical core according to these two lists. The complexity of the algorithm mainly comes from sorting of the two lists. We assume the physical mesh is larger than the logical mesh. Therefore, the complexity of Algorithm 1 is $O((m\times n)log(m\times n))$.

Algorithm 1 is fast and intuitive, but it has several issues. First, Algorithm 1 does not take the heat transfers from neighboring cores into account. In general, high utilization schedule can result in lower peak temperature when executed on a less leaky core. However, if several such cores are very close to each other, allocating high utilization schedules to these cores can result in high chip temperature. Second, utilization defined in this paper is more related to the average power consumption. In fact, temperature varies more closely with power density rather than the average power consumption. In what follows, we develop two approaches to address these problems.

B. Hot-Cold Swapping

Algorithm 2 Hot-Cold swapping.

1: Initialize $\mathcal{M}$; // by initial mapping algorithm or Algorithm 1
2: **while** User defined stop criteria not satisfied **do**
3: Calculate $\mathbf{T} = \{T_1, T_2, ..., T_{m\times n}\}$, where T_i is the steady-state temperature of core i in $\mathcal{M}$;
4: $\mathcal{A}^l_{max}$ = The logical core with maximum temperature $T_{max} = max(\mathbf{T})$;
5: $\mathcal{A}^l_{min}$ = The logical core with minimum temperature $T_{min} = min(\mathbf{T})$;
6: //Swap the mapping between $\mathcal{A}^l_{max}$ and $\mathcal{A}^l_{min}$
7: $LC(\mathcal{A}^l_{max}) \rightarrow LP(\mathcal{A}^l_{min})$;
8: $LC(\mathcal{A}^l_{min}) \rightarrow LP(\mathcal{A}^l_{max})$;
9: Denote the new mapping as $\mathcal{M}'$;
10: Calculate **T'**; // steady-state temperature of new mapping $\mathcal{M}'$
11: **end while**

Given the *nominal design*, we can calculate the steady-state temperature for each core by the method we proposed in Section III, based on which we can easily calculate the peak temperature when temperature reaches the stable status. By calculating the peak temperature, this method avoids the pitfall in the previous method to identify the peak temperature based on the schedule utilization. Then *Hot-Cold Swapping* algorithm swaps the physical to logical topology between the hottest and coldest cores as shown in Algorithm 2. Similar to the principle for the "heat-and-run" heuristic [29], this method always exchanges the voltage schedules for the hottest/coldest cores, with the expectation that the heat across the chip can be spread and balanced in the entire chip until certain criteria (such as a pre-set peak temperature limit or loop counts) are reached. The complexity of Algorithm 2 is mainly from calculating the stable status temperature according to equation (24). Note that the dimension of matrix $\mathbf{A}_{\kappa_q}$ is $(x\times y)\times(x\times y)$. Since the complexity for the straightforward implementation of the matrix multiplication and inversion are both $O(n^3)$ for $n\times n$ matrices, the complexity for each iteration in Algorithm 2 is $O(s\times(x\times y)^3)$ where s is the number of scheduling points for $\mathbf{S}(t)$.

While the *Hot-Cold Swapping* algorithm is simple, it does not necessarily reduce the peak temperature when swapping the schedules on a pair of hot/cold cores each time. The problem for this is that this approach does not take the heat transfers into consideration. Consider a core with light workload but surrounded with high workload cores and thus becomes the hottest core. When changing the schedule of it with other cold cores of lower temperature but higher workload, the peak temperature can become even higher.

C. Enhanced Hot-Cold Swapping

To solve the problem for *Hot-Cold Swapping*, we develop an *Enhanced Hot-Cold Swapping* as shown in Algorithm 3. The major difference between the two algorithms is that, in our *enhanced hot-cold swapping* algorithm, we tentatively swap the hottest and coldest cores. The swapping is accepted only when the new peak temperature is lower than the original one. If the peak temperature of the new mapping is higher than the initial mapping, we search for the core with second minimum temperature and swap it with the hottest core, until we can find such a swapping that reduces the peak temperature or we have exhausted all the possibilities. In the worst case, there are $(m\times n)-1$ pairs of processor to be tested. Therefore, the complexity to run one iteration of Algorithm 3 is $O(s\times(m\times n)\times(x\times y)^3)$, where s is the number of scheduling points for $\mathbf{S}(t)$, $(m\times n)$ is the matrix size for the physical topology, and $(x\times y)$ is the matrix size for the logical topology. By ensuring the peak temperature non-increasing, *Enhanced Hot-Cold Swapping* heuristic can be more effective in guiding the search process to identify the good physical to logical topology mapping. In the next section, we use experiments to evaluate the performance of these algorithms.

V. Experimental results

In this section, we perform three sets of experiments. First, we compare the peak temperature differences between with and without heterogeneity-aware approaches. Second, we study the performances of three approaches presented in Section IV and compare our algorithms with *nominal design* and the optimal solution *exhaustive search* which enumerates all the possibilities. The last experiment is to compare the computation costs between different algorithms.

A. Experimental setup

In our simulation study, the multi-core platform consists a 2-D 3×3 mesh. We adopt the processor model from [30], each

Algorithm 3 Enhanced Hot-Cold swapping.

Initialize $\mathcal{M}$; // by initial mapping algorithm or Algorithm 1
$\Lambda = 0$;
Calculate $T = \{T_1, T_2, ..., T_{\mathcal{P}}\}$, where T_i is the steady-state temperature of core i in $\mathcal{M}$;
$T^* = T$;
while $\Lambda < \varepsilon$ // user defined threshold **do**
  $\mathcal{A}^l_{max}$ = The logical core with maximum temperature $T^*_{max} = max\{T^*\}$;
  $\mathcal{A}^l_{min}$ = The logical core with minimum temperature $T^*_{min} = min\{T^*\}$;
  //Swap the mapping between $\mathcal{A}^l_{max}$ and $\mathcal{A}^l_{min}$
  $LC(\mathcal{A}^l_{max}) \rightarrow LP(\mathcal{A}^l_{min})$;
  $LC(\mathcal{A}^l_{min}) \rightarrow LP(\mathcal{A}^l_{max})$;
  Denote the new mapping as $\mathcal{M}'$;
  Calculate T'; // steady-state temperature of new mapping $\mathcal{M}'$
  if $(max\{T'\} \geq max\{T^*\})$ **then**
    $T^* = T^* - \{T^*_{min}\}$;
  else
    if $(T^* == \{T^*_{max}\})$ **then**
      break;
    else
      $\mathcal{M} = \mathcal{M}'$;
      $\Lambda = \frac{max\{T\} - max\{T'\}}{max\{T\}}$;
    end if
  end if
end while

(a) Power/thermal parameters

$V_{dd}(v)$	α	β	γ
0.00	0.0	0.0	0.0
0.60	0.2734	0.1313	16
0.65	0.5764	0.1383	16
0.70	0.9606	0.1457	16
0.75	1.4508	0.1534	16
0.80	2.0804	0.1615	16
0.85	2.8944	0.1700	16
0.90	3.9538	0.1789	16
0.95	5.3415	0.1882	16
1.00	7.1701	0.1979	16
1.05	9.5926	0.2081	16
1.10	12.8179	0.2188	16
1.15	17.1306	0.2300	16
1.20	22.9195	0.2416	16
1.25	30.7152	0.2538	16
1.30	41.2430	0.2665	16

(b) HotSpot parameters

Parameter	*Value*
Total Cores	9 (3×3)
Area per Core	4 mm^2
Die Thickness	0.15
Heat Spreader Side	20 *mm*
Heat Sink Side	30 *mm*
Convection Resistance	0.1 *K/W*
Convection Capacitance	140 *J/K*
Ambient Temperature	30° *C*

TABLE I. EXPERIMENT PARAMETERS

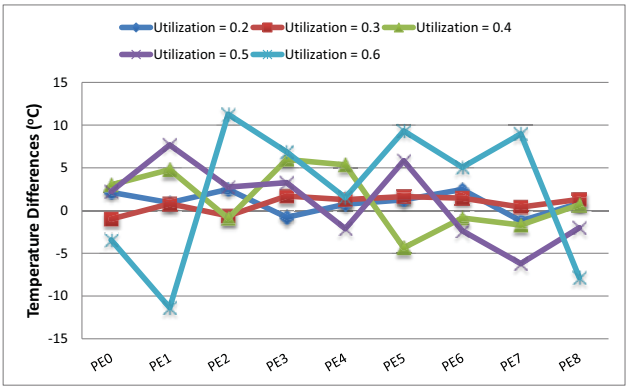

Fig. 3. Temperature comparisons between different utilizations

core supports 15 active modes with supply voltages ranging from $0.6V$ to $1.3V$ with step interval of $0.05V$ while the maximum frequency is generated as normal distribution and the frequencies of each running mode is calculated accordingly [11]. For each core, we generate a static, periodic voltage schedule. Specifically, we divide the first period into 50 state intervals equally, for each state interval we randomly select one voltage mode from $0.6V$ to $1.3V$. After generating the voltage schedule for each core within the first period, we repeat the same schedule pattern for the rest of periods. As we discussed in Section II-B, within each state interval the running mode of each core is constant.

We calculate the curve fitting parameters similar to the power model discussed in [31], [32] shown in Table I (a) and parameters from HotSpot-5.02 [33] shown in Table I (b) which we use for temperature calculation. Leakage variations are randomly generated as normal distribution: $\Delta_{leak} \sim N(\mu, \sigma)$, where $\mu = 0$ and $\sigma = 0.1 \times \bar{v}$ ($\bar{v}$ is the average voltage speed of $V_{dd}(v)$ in Table I (a)) based on [9], [8]. The ambient temperature is $T_{amb} = 30°C$.

All experiments are running on a Window XP/SP3 platform powered by Intel(R) Core(TM)2 Duo CPU @ 2.93GHz with 3.21 GB of RAM.

B. Temperature vs. leakage variation

In this experiment, we first study the need to take leakage power consumption variations into consideration for temperature calculations. Utilization of each core U_i is calculated by equation (3). For different utilizations, we compare temperature differences for each core with and without leakage variations. The leakage variations are generated according to section V-A and each test case is running for 20 rounds.

In Figure 3, we compare temperature differences on 5 different utilization settings $U_i \in [0.2, 0.3, 0.4, 0.5, 0.6], i = 1, 2, ..., \mathcal{P}$. X-axis represents core IDs while Y-axis represents temperature differences between the cases with and without leakage variations. As indicated from the figure, with the increase of utilization, the temperature variations for each core is also increasing. The larger the utilization is, the larger the discrepancy in temperature calculation. For example, the temperature difference is no more than $3°C$ when utilization is 0.2 while the temperature difference can be as large as more than $10°C$ when utilization is 0.6. Therefore, for a more accurate temperature calculation and peak temperature optimization, we need to take leakage variations into consideration.

C. Peak temperature minimization

Next, we study different approaches to optimize peak temperature. We randomly generate voltage schedule for each core with utilization $U_i \in \{[0.2, 0.3], [0.3, 0.4], [0.4, 0.5], [0.5, 0.6]\}, i = 1, 2, ..., \mathcal{P}$ and label as *nominal design*. The reason we limit core's utilization to $U_i = 0.6$ is to satisfy the peak temperature constraint $T_{max} = 95°C$ in steady state ($T_{max} = 95°C$ is the temperature threshold we choose). We assume that the logical architecture and physical architecture are of equal size, i.e., $x = m$ and $y = n$. When the physical topology size is larger than the logical topology size, it is not difficult to see that

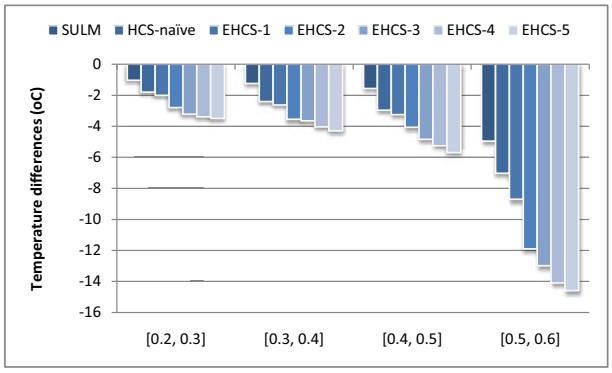

Fig. 4. Peak temperature reductions normalized to initial mapping

our heuristics can perform better simply because of the extra resources or optimization opportunities available. For each utilization setting, we generate 100 test cases and calculate the average temperature reduction.

We denote our three optimization methods as *SULM* (the *simple utilization/leakage matching* heuristic), *HCS-naive* (the *Hot-Cold Swapping* heuristic) and *EHCS* (the *Enhanced Hot-Cold Swapping* heuristic). If no temperature reduction can be made, *EHCS* will stop. *EHCS*-1 refers to our *Enhanced Hot-Cold swapping* algorithm with one iteration.

In Figure 4, we compare peak temperature reductions between *SULM*, *HCS-naive*, and *EHCS*-1 to *EHCS*-5 with 4 different utilization settings ranging from $[0.2,0.3]$ to $[0.5,0.6]$. From the figure, the first conclusion we can get is that with the increase of utilization, all heuristics can get more temperature reductions. It is because the higher utilization we have, the more potential we may benefit from heterogeneity-aware algorithms which try to avoid putting higher utilizations on more leaky cores. Second, *SULM* performs the worst because it does not take heat transfers between neighbors into account. *HCS-naive* is better than *SULM* because it does consider heat transfers, but compared to *EHCS*, *HCS-naive* would always swap the physical to logical topology between the hottest and coldest cores regardless and it is possible that sometimes the peak temperature after swapping is higher than the *nominal design*, therefore hampering its performance. From *EHCS*-1 to *EHCS*-5, it indicates that the more iterations we run the better peak temperature reduction we can get.

One thing we need to note is that with utilization setting of $[0.5,0.6]$, *EHCS* can perform much better than previous settings. Higher utilization indicates that each core has larger possibility to run at high voltage modes that are more sensitive to core heterogeneity. Therefore, with heat transfers and leakage variations into account, *EHCS* can benefit more than *SULM* and *HCS-naive* from utilization increasing.

Another set of experiments is shown in Figure 5. We want to see how good *EHCS* algorithm is compared to the optimal solution *exhaustive search* which enumerates all the mapping possibilities. This time we generate each core's utilization $U_i \in [0.2,0.6], i = 1,2,...,\mathcal{P}$ for a more general purpose. We perform 100 test cases for each algorithm and calculate the average.

From the figure, *SULM* can get $5.86°C$ reduction, *HCS-naive* can get $8.79°C$ reduction. From *EHCS*-1 to *EHCS*-10 can get $10.16°C$ all the way to $14.09°C$ reduction, while exhaustive search can get $18.99°C$ reduction in average. Note that after *EHCS*-5 the reduction potential is not significant. Therefore, we can choose different iterations based on the timing and improvement we want to achieve. In general, compared with *exhaustive search*, *EHCS*-5 algorithm only performs less than $5°C$ of difference.

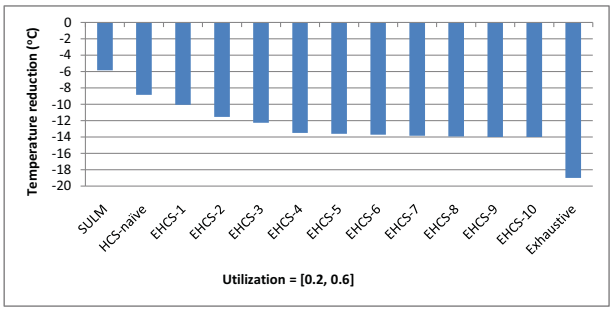

Fig. 5. Temperature reductions between *EHCS* and *Exhaustive search*

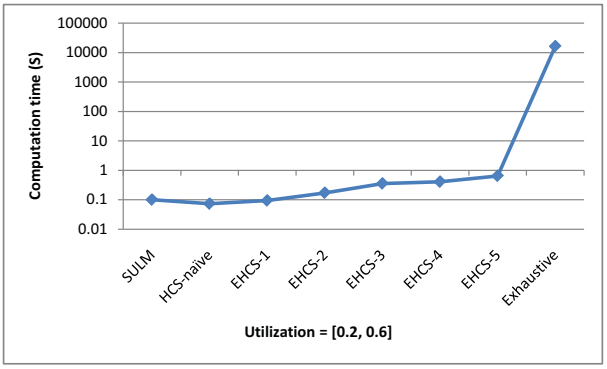

Fig. 6. Computation time differences between different approaches

D. Operational cost

As mentioned earlier, the computation efficiency plays a vital role in *topology virtualization*. We next study the computational cost for different approaches with utilization setting $U_i \in [0.2,0.6], i = 1,2,...,\mathcal{P}$. For simplicity we just compare from *EHCS*-1 to *EHCS*-5 to give a trend of how the timing complexity looks like. Figure 6 shows the time that each algorithm runs only one test case while *EHCS*-1 to *EHCS*-5 with different iterations from one to five. From the figure, it indicates that the timing complexity is linearized and it is what we expected since we need such a mapping heuristic that can perform fast and produce relatively good results. *EHCS*-5 which is the most time consuming among the five, takes about 0.64 seconds to finish. However, *exhaustive search* is very time consuming, the computation complexity is $O(\mathcal{P}!)$ which takes 4 to 5 hours to finish in our experiment.

VI. Conclusions

Temperature minimization problem is becoming more and more critical in computer system design. In the meantime, the increasing manufacturing variations for IC chips also raise the

concerns in the design of computing systems. We believe that the heterogeneity caused by manufacturing variations, if utilized appropriately, can in fact improve the design objectives of real-time applications. In this paper, we develop three heuristics to judiciously mirror the underlying physical architecture of an individual device to the logical architecture with the objective of peak temperature minimization. The proposed heuristics can achieve $14.09°C$ temperature reduction in average and less than $5°C$ of difference compared with *exhaustive search*. Overall, our proposed algorithm can be finished within 1 second (more than 10^4 times faster compared to *exhaustive search*) which is the key to the success of optimization problems through *topology virutalization*.

REFERENCES

[1] K. Skadron, M. Stan, W. Huang, S. Velusamy, K. Sankaranarayanan, and D. Tarjan, "Temperature-aware microarchitecture," in *Computer Architecture, 2003. Proceedings. 30th Annual International Symposium on*, 2003, pp. 2–13.

[2] L.-T. Yeh and R. C. Chu, *Thermal Management of Microelectronic Equipment: Heat Transfer Theory, Analysis Methods, and Design Practices*. New York, NY: ASME Press, 2002.

[3] JEDEC, *Failure mechanisms and models for semiconductor devices*, http://www.jedec.org, 2009.

[4] K. Chakraborty and S. Roy, "Rethinking threshold voltage assignment in 3d multicore designs," in *VLSI Design, 2010. VLSID '10. 23rd International Conference on*, 2010, pp. 375–380.

[5] S. Nassif, K. Bernstein, D. Frank, A. Gattiker, W. Haensch, B. Ji, E. Nowak, D. Pearson, and N. Rohrer, "High performance cmos variability in the 65nm regime and beyond," in *Electron Devices Meeting, 2007. IEDM 2007. IEEE International*, 2007, pp. 569–571.

[6] K. Kuhn, "Managing process variation in intel's 45nm cmos technology," *Intel Technical Journal*, vol. 12, no. 2, pp. 93–110, Jun. 2008.

[7] ITRS, *International Technology Roadmap for Semiconductors (2008 Edition)*. Austin, TX.: International SEMATECH, http://public.itrs.net/, 2008.

[8] L. Wanner, C. Apte, R. Balani, P. Gupta, and M. Srivastava, "Hardware variability-aware duty cycling for embedded sensors," pp. 1–1, 2012.

[9] S. Borkar, T. Karnik, S. Narendra, J. Tschanz, A. Keshavarzi, and V. De, "Parameter variations and impact on circuits and microarchitecture," in *Proceedings of the 40th annual Design Automation Conference*, 2003, pp. 338–342.

[10] J. Oliver, R. Amirtharajah, V. Akella, and F. T. Chong, "Credit-based dynamic reliability management using online wearout detection," in *Proceedings of the 5th conference on Computing frontiers*, 2008, pp. 139–148.

[11] K. Kuhn, "Cmos transistor scaling past 32nm and implications on variation," in *Advanced Semiconductor Manufacturing Conference (ASMC), 2010 IEEE/SEMI*, Jul. 2010, pp. 241 –246.

[12] Y. Taur, "Cmos design near the limit of scaling," *IBM Journal of Research and Development*, vol. 46, no. 2.3, pp. 213–222, Mar. 2002.

[13] J. Sartori, A. Pant, R. Kumar, and P. Gupta, "Variation-aware speed binning of multi-core processors," in *Quality Electronic Design (ISQED), 2010 11th International Symposium on*, Mar. 2010, pp. 307–314.

[14] Z. Mengying, O. Alex, and X. C. Jason, "Profit maximization through process variation aware high level synthesis with speed binning," in *Proceedings of the Conference on Design, Automation and Test in Europe*, ser. DATE '13. San Jose, CA, USA: EDA Consortium, 2013, pp. 176–181. [Online]. Available: http://dl.acm.org/citation.cfm?id=2485288.2485335

[15] F. Wang, Y. Chen, C. Nicopoulos, X. Wu, Y. Xie, and N. Vijaykrishnan, "Variation-aware task and communication mapping for mpsoc architecture," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 30, no. 2, pp. 295–307, Feb. 2011.

[16] B. Raghunathan, Y. Turakhia, S. Garg, and D. Marculescu, "Cherry-picking: exploiting process variations in dark-silicon homogeneous chip multi-processors," in *Proceedings of the Conference on Design, Automation and Test in Europe*, ser. DATE '13. San Jose, CA, USA: EDA Consortium, 2013, pp. 39–44. [Online]. Available: http://dl.acm.org/citation.cfm?id=2485288.2485301

[17] T. Wang, G. Quan, S. Ren, and M. Qiu, "Topology virtualization for throughput maximization on many-core platforms," in *Parallel and Distributed Systems (ICPADS), 2012 IEEE 18th International Conference on*, 2012, pp. 408–415.

[18] L. Zhang, Y. Han, Q. Xu, and X. Li, "Defect tolerance in homogeneous manycore processors using core-level redundancy with unified topology," in *Design, Automation and Test in Europe, 2008. DATE '08*, Mar. 2008, pp. 891–896.

[19] L. Zhang, Y. Han, Q. Xu, X. wei Li, and H. Li, "On topology reconfiguration for defect-tolerant noc-based homogeneous manycore systems," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 17, no. 9, pp. 1173–1186, Sep. 2009.

[20] L. Zhang, Y. Yu, J. Dong, Y. Han, S. Ren, and X. Li, "Performance-asymmetry-aware topology virtualization for defect-tolerant noc-based many-core processors," in *Design, Automation Test in Europe Conference Exhibition (DATE), 2010*, Mar. 2010, pp. 1566–1571.

[21] K. Yue, S. Ghalim, Z. Li, F. Lockom, S. Ren, L. Zhang, and X. Li, "A greedy approach to tolerate defect cores for multimedia applications," in *Embedded Systems for Real-Time Multimedia (ESTIMedia), 2011 9th IEEE Symposium on*, Oct. 2011, pp. 112–119.

[22] K. Yue, F. Lockom, Z. Li, S. Ghalim, S. Ren, L. Zhang, and X. Li, "Hungarian algorithm based virtualization to maintain application timing similarity for defect-tolerant noc," in *Design Automation Conference (ASP-DAC), 2012 17th Asia and South Pacific*, Feb. 2012, pp. 493–498.

[23] M. R. Garey and D. S. Johnson, *Computers and Intractability; A Guide to the Theory of NP-Completeness*. New York, NY, USA: W. H. Freeman & Co., 1990.

[24] T. Lei and S. Kumar, "A two-step genetic algorithm for mapping task graphs to a network on chip architecture," in *Digital System Design, 2003. Proceedings. Euromicro Symposium on*, 2003, pp. 180–187.

[25] S. Saha, Y. Lu, and J. Deogun, "Thermal-constrained energy-aware partitioning for heterogeneous multi-core multiprocessor real-time systems," in *Embedded and Real-Time Computing Systems and Applications (RTCSA), 2012 IEEE 18th International Conference on*, 2012, pp. 41–50.

[26] S. Garg and D. Marculescu, "System-level leakage variability mitigation for mpsoc platforms using body-bias islands," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 20, no. 12, pp. 2289–2301, 2012.

[27] S. Sharifi, R. Ayoub, and T. Rosing, "Tempomp: Integrated prediction and management of temperature in heterogeneous mpsocs," in *Design, Automation Test in Europe Conference Exhibition (DATE), 2012*, 2012, pp. 593–598.

[28] I. Ukhov, M. Bao, P. Eles, and Z. Peng, "Steady-state dynamic temperature analysis and reliability optimization for embedded multiprocessor systems," in *Design Automation Conference (DAC), 2012 49th ACM/EDAC/IEEE*, 2012, pp. 197–204.

[29] M. Gomaa, M. D. Powell, and T. N. Vijaykumar, "Heat-and-run: leveraging smt and cmp to manage power density through the operating system," in *Proceedings of the 11th international conference on Architectural support for programming languages and operating systems*, ser. ASPLOS XI. New York, NY, USA: ACM, 2004, pp. 260–270. [Online]. Available: http://doi.acm.org/10.1145/1024393.1024424

[30] W. Liao, L. He, and K. Lepak, "Temperature and supply voltage aware performance and power modeling at microarchitecture level," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 24, no. 7, pp. 1042 – 1053, 2005.

[31] G. Quan and Y. Zhang, "Leakage aware feasibility analysis for temperature-constrained hard real-time periodic tasks," *ECRTS*, pp. 207–216, 2009.

[32] G. Quan and V. Chaturvedi, "Feasibility analysis for temperature-constraint hard real-time periodic tasks," *Industrial Informatics, IEEE Transactions on*, vol. 6, no. 3, pp. 329–339, 2010.

[33] K. Skadron, M. Stan, W. Huang, S. Velusamy, K. Sankaranarayanan, and D. Tarjan, "Temperature-aware computer systems: opportunities and challenges," *IEEE Micro*, vol. 23, no. 6, pp. 52–61, 2003.