# *Chapter 9*
# Runtime thermal management of many-core systems

*Anup Das*[1] *and Akash Kumar*[2]

Many-core systems are widely deployed for embedded and high-performance computing. With matured technological advances in the deep submicron technology nodes, it is now possible to integrate 100s to 1,000s of cores per system-on-chip die. However, growing core count results in thermal hot spots and large temperature gradients, which significantly impact system reliability, performance, cost and leakage power. A primary design optimization objective for many-core systems is, therefore, to efficiently manage thermal overheads of applications while satisfying their performance requirements. This improves system reliability leading to higher mean-time-to-failure.

This chapter presents approaches to runtime thermal management for embedded systems looking from two perspectives—management in a conventional many-core system and that designed in evolving three-dimensional (3D) integrated circuit (IC) technology. Specifically, this chapter first introduces the design of a runtime manager for a many-core system, which uses workload statistics from the processing cores to efficiently allocate resources in order to alleviate thermal emergencies. The impact of workload uncertainty is characterized using machine-learning techniques. Next, this chapter presents a fast thermal-aware approach for mapping throughput-constrained streaming applications on 3D many-core systems. In this approach, to avoid slow thermal simulations for every candidate mapping, a thermal model of the 3D IC is used to derive an on-chip power distribution (PD) that minimizes the temperature before the actual mapping is done. This distribution is then used in a resource allocation algorithm to derive a mapping that meets the throughput constraint while approaching the target PD and minimizing energy consumption. This way, in contrast to most existing approaches, a mapping can be derived in the order of minutes.

## 9.1 Thermal management of many-core embedded systems

Multimedia applications, such as video encoding and decoding, are characterized by different execution phases, which are defined as a group of consecutive frames.

[1]Department of Electrical and Computer Engineering, Drexel University, USA
[2]Institute of Computer Engineering, Department of Computer Science, Technische Universität Dresden, Germany

The average workload of the frames comprising a phase (inter) varies significantly across the different phases; however, the workload variation within each phase (intra) is relatively low. Proactive power and thermal management involves predicting these dynamic workloads a priori to determine the most appropriate frequency for every phase such that performance constraint is satisfied while minimizing the power consumption, which in turn leads to a reduction of average temperature [1–6]. Studies have been conducted recently to use machine learning to determine the minimum frequency through continuous feedback from the hardware performance monitoring unit (PMU) [7–15]. These approaches suffer from the following limitations.

First, some of the practical aspects of many-core systems are ignored in the existing works. Specifically, the CPU cycle count for a frame, obtained by reading the PMU registers at runtime, is assumed to be a true indicator of the frame workload. However, as we show in this chapter, the PMU register readings contain a certain amount of uncertainty, influenced by factors such as cache contention, dynamic random access memory (DRAM) access, that can have a significant impact on thermal management. This uncertainty is difficult to estimate at runtime due to the unpredictability associated with these factors, especially for many-core systems with a realistic assumption of concurrently executing routine applications. Thus, although workload estimation based on CPU cycle count leads to efficient thermal management using dynamic voltage/frequency scaling (DVFS), a significant improvement is possible by estimating the uncertainty as show in this chapter. Second, the existing approaches do not consider voltage and frequency switching overhead, which is significant in modern many-core systems. Last, the classical workload prediction-based power/thermal minimization techniques work in an ad hoc manner by predicting the workload and deciding the frequency based on this predicted workload. On the other hand, workload history-based statistical classification approaches determine the probability that a sudden spike (positive or negative) in the workload is due to a change in the phase of the workload that needs to be processed at a different frequency. Thus, instead of acting instantaneously, the classifier evaluates the probability distribution of the different classes based on the workload change, and the most probable frequency is selected such that the scaling leads to thermal improvement for future workloads. However, these classifiers require characterization using training data, i.e., a supervised learning approach. Modern many-core operating systems, such as Linux and Android, also support dynamic frequency scaling during application execution. The default and the most popular *ondemand* power governor [16] uses the current workload to determine the voltage-frequency value to process the future workload (a reactive approach). As we show in this chapter, thermal improvement using the *ondemand* power governor can be outperformed using a naive predictive heuristic.

### 9.1.1 Uncertainty in workload estimation

Conventionally, the minimum frequency for an application is determined based on the CPU cycle count (henceforth, referred to as workload) read from the PMU registers. The underlying assumption is that the CPU cycle count corresponds to frame processing only. In modern many-core systems, there are a number of applications that continue to operate in the background. Some of these applications are user controlled such as web page rendering, email checking and virus scanning. There are also system-related applications that are routinely executed on the processing cores. Some of these applications are beyond the knowledge of the users and cannot be forcefully exited. As a result, the PMU register readings are not always a true indicator of the actual frame workload. This can be seen in a recent work by Das *et al.* [17].

To estimate the impact of workload uncertainty on the frequency value, let $\tilde{w}$ and $w$ denote the observed and the actual frame workloads, respectively, with $\tilde{w} = w + e$, where $e$ is the workload uncertainty. The observed and the required workload frequencies are related according to

$$\frac{f_{required}}{f_{observed}} = \frac{\tilde{w} - e}{\tilde{w}} \leq 1 \tag{9.1}$$

Clearly, estimating the uncertainty in the observed workload leads to further scope for energy improvement. Thus, the problem we are addressing is as follows. Given the workload obtained from the PMU registers at runtime, how to estimate the workload uncertainty ($e$), being agnostic of its probability distribution, such that the voltage-frequency value corresponding to the actual workload $w$ can be applied on the system. This is the objective of the next section.

## *9.1.2 Learning-based uncertainty characterization*

Statistical classification is the process of identifying a class (from a set of discrete classes) for a new observation, based on a training set of observations, whose class is known a priori [18]. In this work, we focus on a discriminative classifier—logistic regression applied to a multinomial variable [19]. This type of classifier predicts the probability distribution over a set of classes from a sample input to learn a direct mapping from the input sample to the output class. The logistic regression based classification is composed of two steps—modeling to estimate the probability distribution of the different classes for a given input, and parameter fitting to estimate the parameters of the logistic regression model. These are described next.

### 9.1.2.1 Multinomial logistic regression model

*Assumptions*

$\mathscr{A}_1$: There are $K$ discrete frequencies supported by the hardware. The incoming workload is assigned to one of these values, depending on which frequency results in the least energy consumption while satisfying the performance requirement. This is same as classifying into one of $K$ classes.

$\mathscr{A}_2$: The class of the next video frame is predicted based on the workloads of the $N$ previous frames. These are identified by $X = (x_1 \; x_2 \; \cdots \; x_N) \in \mathbb{R}^{1\times N}$, where $x_i$ is the workload of the $i$th previous frame.

$\mathscr{A}_3$: The workload class is denoted by the variable $y \in [1, 2, \ldots, K]$ and the logistic regression model is represented by the hypothesis $h_\theta$, with parameter $\theta \in \mathbb{R}^{(K-1)\times N}$.

It can be shown that for a given input *feature* set $X$, the logistic regression model outputs $h_\theta(X)$ is given by

$$h_\theta(X) = \begin{bmatrix} p_1 \\ p_2 \\ \vdots \\ p_{K-1} \end{bmatrix} = \left( \frac{e^{\left(\theta^{(1)^T} \cdot X\right)}}{\sum_{j=1}^{K} e^{\left(\theta^{(j)^T} \cdot X\right)}} \cdots \frac{e^{\left(\theta^{(K-1)^T} \cdot X\right)}}{\sum_{j=1}^{K} e^{\left(\theta^{(j)^T} \cdot X\right)}} \right) \tag{9.2}$$

The output class $y$ is given by

$$y = \underset{l}{\operatorname{argmax}} \{p_l \ \forall \ l \in [1, 2, \ldots, K]\} \tag{9.3}$$

### 9.1.2.2 Maximum likelihood estimation

We consider a training set of $M$ samples generated independently and identically. For each of these samples, the input *feature* $X$ and the output class $y$ are known a priori and the input–output pairs are identified as $(X^{(i)}, y^{(i)})\ \forall i \in [1, 2, \ldots, M]$. The maximum likelihood estimation is a technique to estimate the parameters ($\theta$ in our case) of a model by maximizing the likelihood of the joint probability distribution of the different observations. This is given by

$$\ell(\theta) = \ln(\mathscr{L}(\theta)) = \sum_{i=1}^{M} \sum_{l=1}^{K} \mathscr{I}(y^{(i)} = l) \cdot \ln \left( \frac{e^{\left(\theta^{(l)^T} \cdot X^{(i)}\right)}}{\sum_{j=1}^{K} e^{\left(\theta^{(j)^T} \cdot X^{(i)}\right)}} \right) \tag{9.4}$$

### 9.1.2.3 Uncertainty interpretation

First we define two new terms—observed class and actual class. The observed class (denoted by $\tilde{y}$) is the class perceived at the output of the logistic regression model corresponding to input $X$ and includes the uncertainty. Let the variable $y$ denote the actual class as before. Using the basic principles of probability theory, the probability of the observed class is

$$P(\tilde{y} = i \mid X) = \sum_{r=1}^{K} P(\tilde{y} = i \mid y = r) \cdot P(y = r \mid X) = \sum_{r=1}^{K} \gamma_{i,r} \cdot p_r \tag{9.5}$$

where $\gamma_{i,r}$ is the probability that the actual class $r$ is flipped to the observed class $i$. Using this probability and the definition of the likelihood function, the log likelihood function is[1]

$$\ell(\theta, \gamma) = \sum_{i=1}^{M} \sum_{l=1}^{K} \mathscr{I}(\tilde{y}^{(i)} = l) \ln \left( \sum_{r=1}^{K} \gamma_{l,r} \cdot p_r \right) \tag{9.6}$$

Finally, the output of the hypothesis is modified as

$$h_\theta(X) = \left[ \left( \sum_{r=1}^{K} \gamma_{1,r} \cdot p_r \right) \cdots \left( \sum_{r=1}^{K} \gamma_{K-1,r} \cdot p_r \right) \right]^T \tag{9.7}$$

[1]The derivation steps are omitted for space limitation.

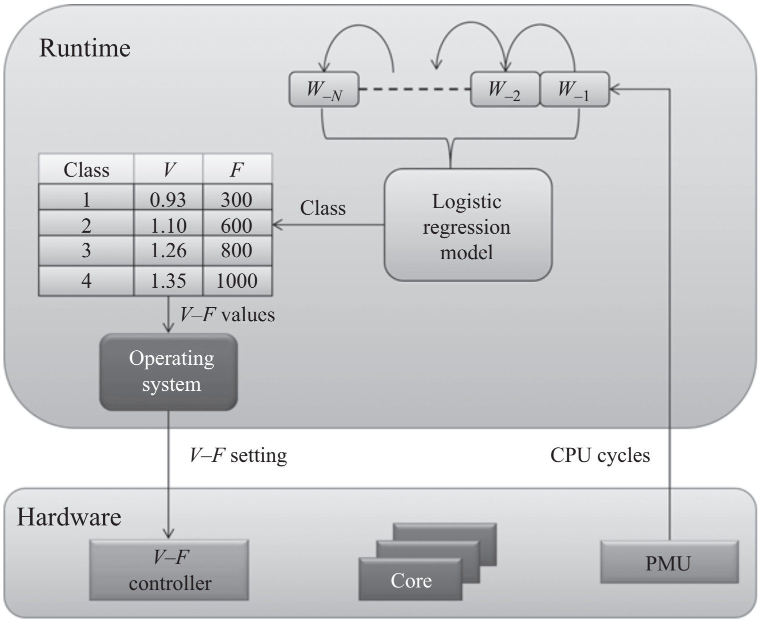
## *9.1.3 Overall design flow*

Figure 9.1 shows the adaptive thermal minimization methodology for a many-core system. An overview is provided on the interaction of the different blocks of this methodology.

***Application:*** Typically, multimedia applications are characterized with a performance constraint specified as *frames per second*, reciprocal of which gives the timing constraint for processing a frame. The application source code is annotated to include this timing requirement.

***Operating system:*** The operating system is responsible for coordinating the application execution on the hardware. After processing every frame of an application, the operating system stalls execution and triggers the classifier, which predicts the class for the next frame. This class is translated to a frequency value for the CPU cores. The operating system applies this frequency on the CPU cores using the `cpufreq` utility.

***Hardware:*** The hardware consists of processing cores with a PMU to record performance statistics. Of the different performance statistics available, we focus on CPU cycle count. After processing every frame, the PMU readings are collected using the `perfmon` utility. Subsequently, the readings are reset to allow recording for the next frame. Finally, before the start of the next frame, the frequency value set by the operating system is first converted to a corresponding CPU clock divider setting and is then written into appropriate CPU registers. The frequency is scaled to execute the next frame.



| Class | *V* | *F* |
|---|---|---|
| 1 | 0.93 | 300 |
| 2 | 1.10 | 600 |
| 3 | 1.26 | 800 |
| 4 | 1.35 | 1000 |

*Figure 9.1 Design methodology*

## *9.1.4 Early evaluation of the approach*

The proposed run-time approach is evaluated on Texas Instrument's PandaBoard featuring ARM A9 cores and Intel quad-core system running Linux.

### 9.1.4.1 Impact of workload uncertainty: H.264 case study

To signify the impact of the workload uncertainty on the thermal behavior, an experiment is conducted using two 30 s video sequences—"ducks" and "sta_launch" [20]. The H.264 decoder application is restricted to execute on only one core using the `cpu-affinity` feature of the operating system. Further, we let all other routine applications execute freely on any cores. The videos are decoded ten times each and the CPU cycles consumed for each run is recorded. The CPU cycles count are normalized with respect to the maximum obtained for the ten readings. These results are shown in Figure 9.2 corresponding to the label *single core*. The minimum, maximum, the median value and the 80% distribution (as box) of the CPU cycles count for the runs are shown as box plots for both the videos. Furthermore, the percentage difference between minimum and the maximum CPU cycles count of the ten runs (referred to as the variation) is reported on top of each box. Next, the same experiment is repeated by allowing the H.264 decoder application to run on two and four cores of the system. These results are also plotted in Figure 9.2.

As seen from the figure, when the H.264 decoder runs on single core, there is a variation of 3.5% and 6.7% in CPU cycles count for the two videos, respectively. It is to be noted that, even though the H.264 application executes on one core, some of the threads from other routine applications are also scheduled on this core by the operating system. Thus, there is an amount of uncertainty in the observed CPU cycle count. Hence, the voltage-frequency value obtained based on this observed CPU cycle count

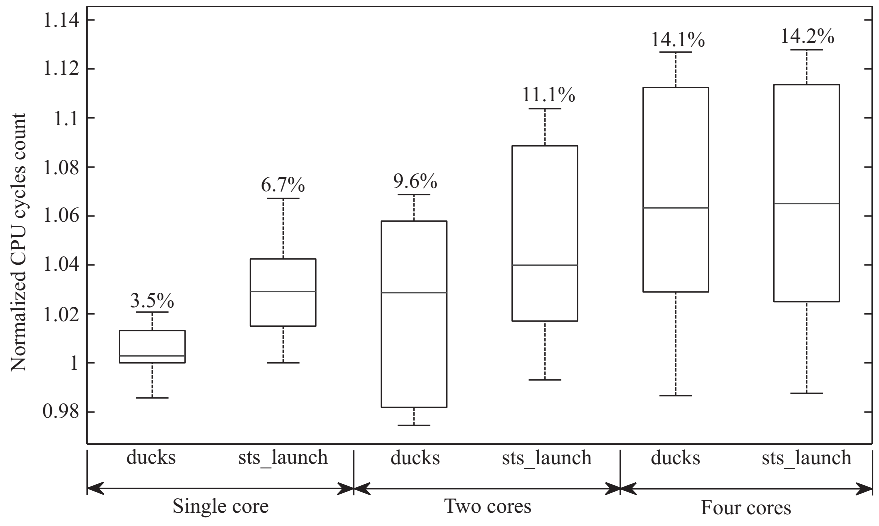


*Figure 9.2 Variation in CPU cycles count due to workload uncertainty*

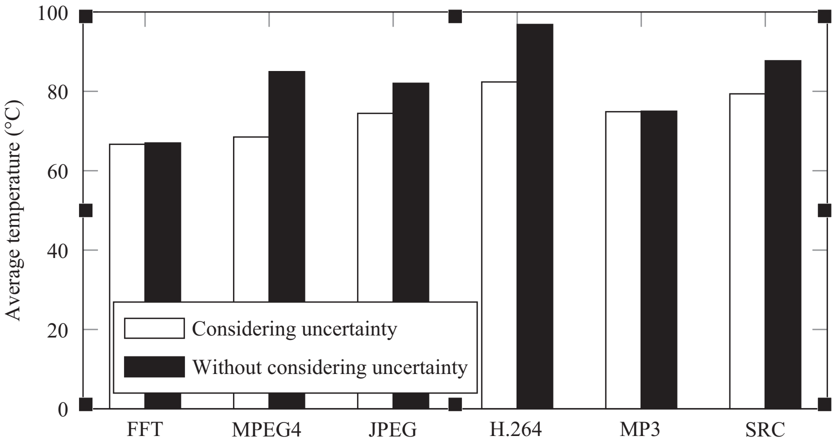


*Figure 9.3 Thermal improvement using our proposed approach*

is not optimal, implying that there is a performance slack that enables the operating system to schedule more threads on this core. When the H.264 application is allowed to execute on two cores, the percentage variation for the two videos increases to 9.6% and 11.1%, respectively. This is because, as the application uses more cores, there are more cache conflicts due to other background threads, increasing the workload uncertainty. Finally, when all the four cores are used by the H.264 decoder, the variation increases to 14%. It can thus be concluded that, workload uncertainty can result in as high as 14% variation in the observed CPU cycles count, clearly motivating our approach to model the uncertainty and mitigate it using the runtime manager.

#### 9.1.4.2 Thermal improvement considering workload uncertainty

Figure 9.3 reports the thermal improvement obtained by considering workload uncertainty. Results are shown for six applications. For some applications such as FFT and MP3, the average temperature are similar, while for others, there are significant difference in the temperature by considering workload uncertainty. Thermal reduction considering uncertainty is between 1 °C and 20 °C. These results clearly show the potential of thermal improvement considering workload uncertainty and allocation of frequency based on this uncertainty estimation.

This concludes our proposal for thermal management for conventional 2D many-core systems. In the remainder of the chapter, we present heuristics for thermal management of systems in 3D IC technology.

## 9.2 Thermal management of 3D many-core systems

3D ICs provide interesting possibilities to implement promising many-core systems in advanced technology nodes. In a 3D IC, multiple layers of logic (or memory) are stacked vertically; these layers are interconnected using vertical interconnect accesses, commonly referred as through silicon vias (TSVs). Figure 9.4 illustrates a 3D IC with two layers and two TSVs. Stacking multiple layers of processing and/or memory elements into a 3D IC structure can significantly reduce the die size and average interconnect wire length. This in turn can reduce communication delays and interconnect energy, while increasing the interconnect flexibility [21,22]. However, stacking active layers increases the power density, which can cause serious thermal problems, affecting both the performance and reliability of a system.

In this section, we introduce an integrated thermal-aware approach for mapping streaming applications on a 3D many-core system. A streaming application, such as video decoding, typically has a throughput requirement, which possibly cannot be guaranteed in combination with unexpected thermal emergencies in a 3D IC. The goal of the proposed approach is to map and schedule multiple applications, satisfying their throughput requirements, communication and storage constraints, while minimizing the peak-temperature and temperature gradients across the chip. Since simulating the thermal process with a high temporal resolution is a time-consuming activity, the approach described in this work is split into two main steps. In the first step, thermal characteristics of the 3D IC are extracted from a simple but flexible model of the physical chip. Then, the extracted profile is passed to the actual resource allocation algorithm, which does not require iterative temperature simulations. This causes runtime of the total flow to be in the order of minutes, in contrast to most existing thermal-aware mapping approaches.

The many-core system is assumed to contain a 3D mesh of (homogeneous or heterogeneous) processing tiles. Since networks-on-chip (NoCs) are widely regarded

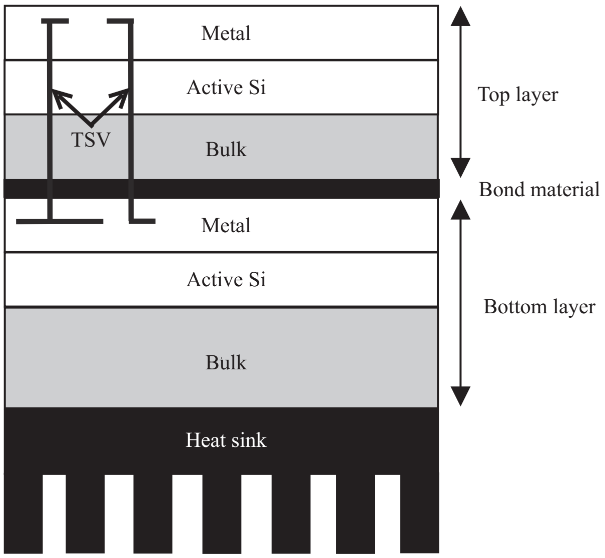


*Figure 9.4 Two stacked layers connected by TSVs [23]*

as the most promising communication architecture for many-core 3D architectures [22], a 3D NoC is assumed to provide the communication between these tiles. The streaming applications are modeled as synchronous dataflow graphs (SDFGs) [24]. Since there are multiple trade-offs involved, for example, between optimizing for energy consumption or peak temperature in the 3D IC [25], a set of parameters has been used to steer the optimization process by providing varying weights to the optimization criteria. This results in a flexible thermal-aware mapping flow. The thermal process resulting from the mapping is simulated using the *HotSpot* thermal simulator [26], which is extended to consider thermal effect of TSVs.

To evaluate performance, the proposed flow is used to map and schedule a set of synthetic benchmark applications as well as realistic multimedia streaming applications on a NoC-based 3D many-core systems. The performance evaluations show that compared to the load-balancing (LB) strategy, the peak temperature and energy consumption are reduced by 7% and 47%, respectively, while meeting all timing and storage constraints.

### *9.2.1 Recent advances on 3D thermal management*

Thermal-aware mapping and scheduling on 3D many-core systems is a well-studied topic. Multiple approaches have been proposed, which can be split into dynamic (runtime) and static (design-time) thermal-aware mapping techniques. Dynamic approaches measure or estimate instantaneous thermal distribution in the chip and initiate actions in order to minimize hotspots and thermal gradients (spatial, temporal or both). In [27], several dynamic mechanisms such as temperature-triggered DVFS, clock gating and hot-task migration are reviewed, and a runtime task assignment algorithm is proposed that takes the thermal history of cores into account. A thermal-aware OS-level scheduler for 3D many-core system is proposed in [28]. These methods share the goal of minimizing the peak temperature and thermal gradients without sacrificing performance too much. Constraints such as deadlines or memory requirements are, however, not taken into account, as well as the effect of inter-task communication. Recent studies have shown that NoCs can dissipate a substantial part of the power budget, and the dissipation depends on the traffic [29]. Therefore, ignoring the interconnect thermal contribution can lead to underestimation of temperature, leading to system failures before they are anticipated. For 2D ICs, a thermal-aware task assignment and scheduling technique for real-time applications is proposed in [30], which is based on mixed integer linear programming. However, the techniques for 2D ICs cannot simply be applied to 3D ICs due to significantly different thermal behavior of 3D ICs. Skadron *et al.* [26] developed the HotSpot thermal simulator to evaluate the steady state and dynamic temperature distribution in ICs.

Static mapping approaches aim at finding a thermal-aware mapping at design time, by using a model of the physical chip, or by using general knowledge about the thermal behavior of 3D ICs. In [29], both temperature and communication load are considered, and a genetic algorithm is used to generate static mappings. Cheng *et al.* [25] show that a trade-off exists when minimizing energy usage as well as peak temperature and use a combination of heuristics, simulated annealing and a greedy algorithm to find optimal static mappings. However, application constraints such as throughput requirements are not taken into account. The authors of [31] propose an approach to find optimal mappings in a thermal sense for applications with deadlines. First, a power balancing algorithm is used to find an initial mapping. The initial mapping is then iteratively improved by simulating the temperature distribution and migrating tasks. Communication between tasks as well as memory constraints are not taken into account. Thiele *et al.* [32] argue that being able to guarantee the maximum peak temperature in a many-core system and analyzing real-time applications early in the design process (i.e., analyzing at design-time) is important, since it removes the need for unpredictable runtime mechanisms. Toward this, the authors use formal methods in a tool to find optimal static mappings of SDFG-modeled applications on heterogeneous 2D many-core systems while guaranteeing performance and peak temperature. The communication overhead is taken into account, but the power dissipation of the NoC has not been considered. Since vertical communication in a 3D NoC can be considerably faster and energy efficient than horizontal communication [25], a straightforward extension of their methods to 3D many-core systems will not provide optimal mapping solutions. In contrast to above strategies, our approach performs thermal-aware mapping of throughput-constraint applications on 3D many-core systems while taking memory as well as communication constraints into account. Further, our approach considers the effect of TSVs on the temperature distribution and power dissipation and minimizes the energy consumption.

Next to thermal-aware mapping, thermal-aware floorplanning can also help to mitigate thermal problems. Thermal-aware floorplanning techniques have been developed for 2D ICs [33] as well as for 3D stacked ICs [34,35]. There are also some research studies on efficient floorplanning to mitigate thermal problems.

## *9.2.2 Preliminaries*

3D ICs have gained significant research attention in recent years. However, experimental platforms for 3D ICs are not yet available to research community. In the absence of real platforms, abstract models of 3D ICs are used for analysis and design space exploration. This section covers the application model, multiprocessor platform model and the 3D IC model.

### 9.2.2.1 Application model

To model streaming applications with a throughput constraint, synchronous dataflow graphs (SDFGs) [24] are used. In a SDFG, an application is modeled as a set of tasks, called *actors*, that communicate chunks of data with a predefined size, called *tokens*. An example SDFG that models an H.263 decoder is depicted in Figure 9.5. The nodes correspond to the actors and the edges represent data dependencies, referred to as connections, between the actors. The H.263 decoder is modeled with four actors *vld, iq, idct* and *mc* and four edges $d_1$, $d_2$, $d_3$ and $d_4$. An actor has fixed input and output rates on every connection. The input rate corresponds to the number of tokens that the actor consumes from the incoming connection when fired once. Similarly, the output rate defines the number of tokens that are produced on the outbound connection

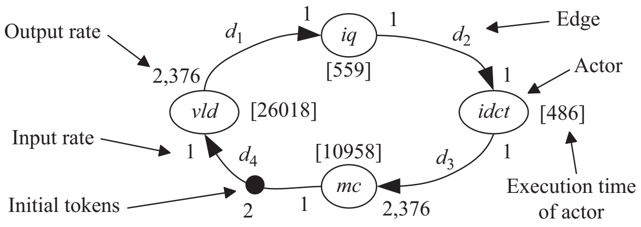


*Figure 9.5 SDFG of a H.263 decoder*

during one firing of the actor. An initial number of tokens might be available on the connection. An actor fires as soon as sufficient tokens are available at all its incoming connections, and enough buffer space is available to store the produced tokens. The size of a token may be different for every connection.

The input and output rates of the actors in an SDFG determine the relative frequency in which the actors can fire, which can be represented by a unique repetition vector. In the application model, the worst case execution times (in time units) and memory requirements (in bits) of all actors on all possible processing elements are specified. For example, an actor performing encoding may have worst case execution times of 10,000 time units on an ARM7 or 2,000 time units on dedicated encoder hardware. Specifying requirements for all possible mappings enables the use of heterogeneous architectures. For all connections, the size of the tokens (in bits) is specified, as well as the memory required when mapping the connection to memory, or the bandwidth required when mapping the connection to interconnect. If actors are fired as soon as they are ready to fire (self-timed execution), the execution pattern of a consistent, strongly connected SDFG is always periodic after an initial start-up period [36]. The time between two recurring states in the execution of an SDFG defines the throughput of the application. An application may have a throughput requirement, fixing the maximum time between two recurring states. The throughput of a SDFG may be calculated by simulating the execution until a recurrent state is found [36].

#### 9.2.2.2 Multiprocessor platform model

In this work, a regular 3D mesh of tiles connected by a NoC is considered, as depicted in Figure 9.6. Every tile contains at least a network interface (NI), connecting the tile to the interconnect network. Furthermore, a tile may contain a processing element ($P$) of some type $PT$ (processor type), for example, an ARM core and a memory. Different types of processors are possible, allowing the modeling of heterogeneous architectures. Such an architecture can be modeled by an architecture graph consisting of tiles and connections as defined below.

**Definition 9.1 (Tile).** *A tile $t$ is a 9-tuple $(pt, w, m, c, i, o, pa, pi, pm)$ with $pt \in PT$ the processor type, $w \in \mathbb{N}_0$ the TDMA timewheel size, $m \in \mathbb{N}_0$ the available memory (in bits), $c \in \mathbb{N}_0$ the maximum number of supported connections, $i, o \in \mathbb{N}_0$ the maximum i/o bandwidth (in bits), $pa, pi \in \mathbb{R}$ the active and idle power (in W) and $pm \in \mathbb{N}_0^3$ the position of the tile in the mesh.*

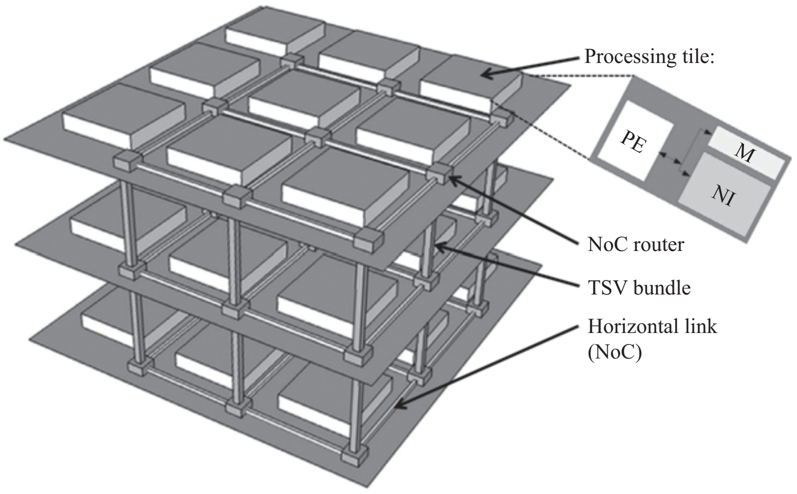


*Figure 9.6 Example 3D mesh of tiles [23]*

Tiles are connected by an NoC. In the architecture graph, the NoC is abstracted to a set of point-to-point connections between tiles.

**Definition 9.2 (Connection).** *A connection c is a 5-tuple $(u, v, l, h_h, h_v)$ with $u \in T$ the source tile, $v \in T$ the destination tile, $l \in \mathbb{N}_0$ the latency (in time units), and $h_h, h_v \in \mathbb{N}_0$, respectively, the number of horizontal and vertical hops between u and v.*

The set of tiles $T$ and the set of connections $C$ together define the architecture graph. A tile is assumed to consume *pa* W of power when active, and *pi* W when idle.

Vertical links between tiles are generally implemented using TSVs. As reported in [25], TSVs can often provide faster and more energy efficient communication compared to horizontal links, mainly because of their short length. The differences in delay and energy per bit depend on the technology and the NoC topology. For example, the NoC switches can simply be extended with two extra ports for up/down communication, or the vertical links can be implemented as a shared bus. This work does not treat all these options in detail, but the latency ($l$) and horizontal and vertical hop count ($h_h, h_v$) properties of a connection do provide some room for modeling different 3D NoC implementations in the architecture graph. For example, the latency of a connection can be specified such that it is dependent on the direction of the communication.

### 9.2.2.3 3D IC model

To be able to simulate the on-chip temperature for a given execution trace, a model describing the thermal characteristics of the 3D IC is required. The model is also used to extract information about the thermal behavior of the chip, which can be used by the mapping algorithm. The model used in this work is based on the 3D grid model available in the HotSpot thermal simulator [26]. The model contains all relevant physical properties of the IC and the heat sink, as well as a set of active and

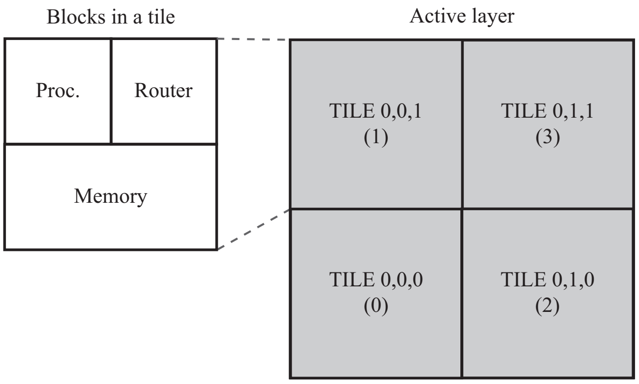


*Figure 9.7 Active layer floorplan illustration*

inactive layer specifications. Active layers correspond to layers that actually dissipate power, while inactive layers are used to model the bonds (glue, thermal interface material (TIM)) between the active layers. Figure 9.7 illustrates the floorplan of an active layer. For every layer, the thickness, material properties and a floorplan are specified. Floorplans of the active layers consist of tiles, all of which contain one or more blocks (e.g., processor, memory, router), specified by the floorplan of the specific tile type. Execution of the application models is tracked at the tile level. The power, that is dissipated in a tile, is assumed to be distributed over the blocks of that tile. For example, 80% may be dissipated in the processor block, 10% in the router block and 10% in the memory. Blocks can correspond to function blocks of the processor, but the processor can also be modeled as one block. This enables the use of both fine (detailed) and course-grained models.

TSVs are generally made of copper, which has significantly different thermal properties than silicon. To take the thermal effect of TSVs into account, the size, position and material properties of the TSVs are also specified in the 3D IC model. To be able to simulate the thermal impact of the TSVs, the HotSpot simulator is extended to take TSVs into account. This is done by changing the thermal properties (conductance and heat capacity) of grid cells in the internal HotSpot model that contain TSV material, based on the ratio of the grid cell volume that is occupied by TSV material.

### 9.2.3 Thermal-aware mapping

This section introduces the proposed mapping flow. The general structure of the flow is depicted in Figure 9.8. The flow consists of two main steps: a "*thermal profiling*" step and the actual mapping algorithm. In the "*thermal profiling*" step, the physical model of the 3D IC is used to derive a PD among the tiles that minimizes the peak temperature and spatial temperature gradients. For example, in the power distribution that minimizes the peak temperature, tiles that are on the layer closest to the heat sink are likely to dissipate more power than tiles far away from the heat sink. This is due to

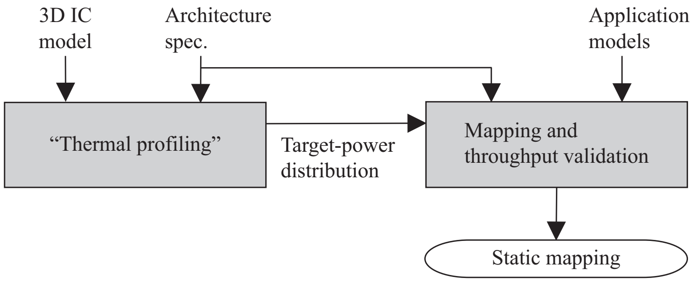


*Figure 9.8 Overview of the thermal-aware mapping flow [23]*

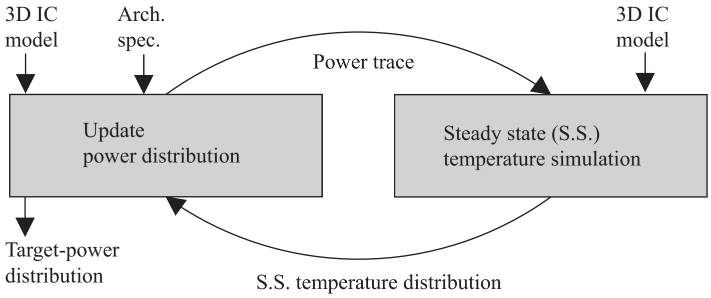

3D IC model
Arch. spec.
Power trace
3D IC model
Update power distribution
Steady state (S.S.) temperature simulation
Target-power distribution
S.S. temperature distribution

*Figure 9.9 Structure of the "thermal profiling" algorithm [23]*

the fact that the layers close to heat sink are able to get rid of the heat faster; therefore, they are able to handle more load/power without overheating. Thus, they get a higher target power ratio. The power density, the floorplan and the absolute position of a tile in the horizontal plane will also influence its target power ratio. The resulting "*target PD*" assigns a power ratio to every tile and is passed to the mapping algorithm. The mapping algorithm tries to find a mapping that approaches this distribution while minimizing the energy and meeting all timing and storage constraints.

The two step structure is based on the observation that (high resolution) thermal simulations have a long running time, making it impossible to simulate the temperature for every candidate mapping within a limited running time. A lot of existing approaches avoid iterative thermal simulations by just applying heuristics to optimize for temperature. However, these heuristics are often not very accurate in a quantitative way, requiring the designer to tune the heuristics by hand in order to match the actual chip properties and find good mappings. In our approach, this tuning is done automatically in the first step of the flow. In the mapping step, no thermal simulations are required, which drastically reduces the runtime compared to methods that simulate the temperature for a lot of candidate mappings. The remainder of this section discusses the steps of the flow in more detail.

### 9.2.3.1 Thermal profiling

The structure of the *thermal profiling* step is depicted in Figure 9.9. The update algorithm adjusts the power ratios $R_t$ of the tiles based on the steady state temperature distribution resulting from the previous PD. The power ratio $R_t$ of a tile corresponds to the ratio between the total chip power $P$ and the power dissipated in tile $t$. The power dissipated in tile $t$, $P_t = R_t \times P$, is distributed among the blocks in that tile based on an intra-tile PD, which may be constant in simple models. This way, a power trace is generated for every block in the chip. To limit the number of thermal simulations, a heuristic is used for updating the PD. The power ratios of tiles with a peak temperature above the average are decreased, while the power ratios of tiles with a peak temperature below the average are increased in the update step. This way, temperature differences among tiles are decreased, resulting in a lower peak temperature and smaller temperature gradients. The adaptation rate of the power ratios is defined by constant $\alpha$. The update algorithm is summarized in Algorithm 1.

**Algorithm 1:** Thermal profiling algorithm

```
Input: 3D IC model, stopping criterion $\delta \in \mathbb{R}$, max. # of iterations $I_{\max} \in \mathbb{N}$,
       adaptation constant $\alpha \in \mathbb{R}$, total chip power $P \in \mathbb{R}$.
Output: Target power ratios $R_t$, $t \in [0, N_{tiles} - 1]$.
Initialize power ratios $\forall t \in [0, N_{tiles} - 1] : R_t = 1/N_{tiles}$;
$T_{avg} = 0$, $T_{prev.\max} = \infty$, $T_{\max} = 1,000$;
$i = 0$;
while $(T_{prev.\max} - T_{\max}) \geq \delta$ and $i \leq I_{\max}$ do
    Generate power traces for all blocks based on $P$, $R_t$ and the intra-tile power
      distribution;
    Simulate steady state temperature dist.;
    $T_{avg} \leftarrow$ average chip temperature;
    $\forall t \in [0, N_{tiles} - 1] : T_{peak,t} \leftarrow$ max. temp. in tile $t$;
    $T_{\max} \leftarrow \max(T_{peak})$ ;
    if $(T_{prev.\max} - T_{\max}) < \delta$ then
        break
    end
    if $T_{prev.\max} < T_{\max}$ then
        $\alpha = \alpha/2$;
        Restart algorithm;
    end
    for all tiles $t \in [0, N_{tiles} - 1]$ do
        $d = (T_{peak,t} - T_{avg})/T_{avg}$;
        $R_t = \max(0, R_t * (1.0 - (\alpha * d)))$;
    end
    Renormalise power ratios $R_t$;
    $T_{prev.\max} = T_{\max}$;
    $i++$;
end
return power ratios $R_t$
```

In agreement with observations described in related literature [27,28], some general observations can be made regarding the PD after convergence of the algorithm:

1. Tiles on layers farther away from the heat sink get hotter than tiles closer to the heat sink with the same power dissipation. Hence, algorithm will in general assign smaller power ratios to tiles further away from the heat sink.
2. The thermal conductance in the vertical direction is generally high compared to the horizontal direction, because the layers are thin (typically 20–100 $\mu$m). As a result, blocks with a high power density that are stacked on top of each other generate high temperatures. Because of this, the combined power dissipation of horizontally aligned blocks in different layers will be limited.
3. TSVs increase the thermal conductance between layers. As a result, temperature differences between layers get decreased. The magnitude of this effect depends on the TSV material, size and density.
4. Blocks that are near the edges/corners of the die tend to get hotter than blocks farther away from the edges, since there is less material for the heat to spread to.

The above observations are used to develop heuristics in some thermal-aware mapping approaches, for example, by assigning higher costs to mappings that use tiles that are further away from the heat sink or by balancing computational load over "stacks" of vertically adjacent tiles [28]. However, these heuristics are generally not tuned to the specific IC that is considered, possibly resulting in suboptimal solutions. In our approach, the temperature-related observations are modeled implicitly in the *target PD*, which is derived directly from a model of the 3D IC. This results in assumptions that are more representative for the specific 3D IC that is considered.

#### 9.2.3.2 Runtime

The steady state temperature is iteratively simulated by the modified HotSpot thermal simulator, which determines the runtime of the algorithm. The simulation time depends on the spatial resolution and the number of layers. For an IC with three active layers and a grid resolution of 32 $\times$ 32, one simulation takes 117s on a 2.3 GHz Intel i7 CPU (single threaded). With a well-chosen value for the adaptation constant $\alpha$, the algorithm converges to a static PD in 6–10 iterations, resulting in a total runtime of up to 1,170 s. Note that the running time is independent from the number of tiles on a layer, since the thermal simulator internally uses a grid with a fixed resolution.

An overview of the mapping flow is depicted in Figure 9.10. The application graphs, the architecture specification and the target PD serve as inputs to the flow. For the memory dimensioning, constraint refinement and communication scheduling steps, existing implementations available in the SDFG$^3$ tool set [37] are applied. The other steps are described subsequently.

#### 9.2.3.3 Application merging

In practical situations, use cases consisting of multiple applications running simultaneously are common. To support the mapping of multiple applications, in the first step of the flow, all application graphs are merged into one application graph using the rate control principle. In this approach, a *rateControl* actor is inserted to control the

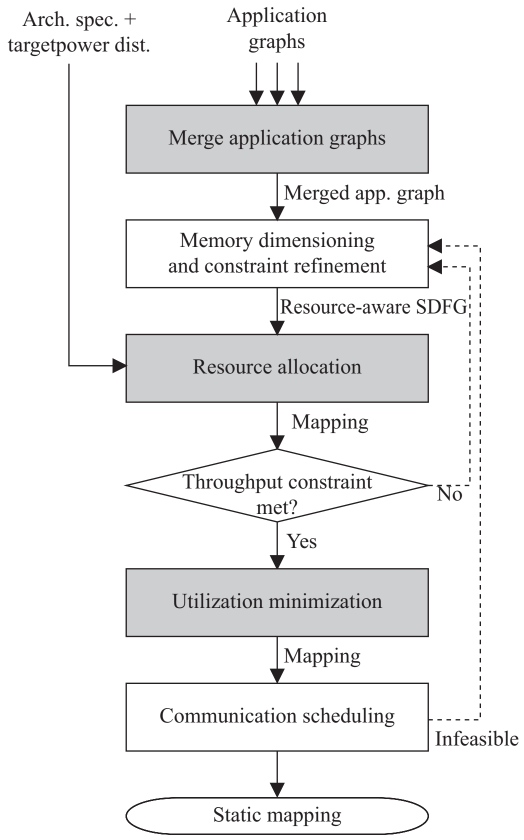


*Figure 9.10 Overview of the mapping flow [23]*

relative execution rates of the different applications that are merged into one graph. Connections between the *rateControl* actor and one actor in every application are added, with input and output rates that force the execution rates of the applications to synchronize in a desired ratio. For example, application *A* might have a throughput requirement twice as high as that of application *B*. In that case, the rate controller will force the execution of application *B* to stall until *A* has executed twice, and vice versa. The throughput constraint of the merged application is the minimum of the original individual throughput constraints, and the individual applications can have throughput constraints that are a multiple of the overall throughput constraint.

#### 9.2.3.4 Resource allocation

In the resource allocation step, every actor of the merged application is bound to a tile in the architecture graph. As a result of binding the actors, the connections between the actors will be bound to either memory (in case both connected actors are bound to the same tile) or to a set of NoC links (in case the actors are bound to different tiles). Since the tile binding defines the computational load distribution and thus the

PD within the many-core system, it is the most important step in the thermal-aware mapping flow. A feasible tile binding binds all actors to a tile and all connections to a memory or interconnect link such that no storage, connection count or bandwidth limitation is violated. After a feasible tile binding is found, a static-order schedule is generated for each tile, defining the order of execution of the actors mapped on that tile. Note that the resulting resource allocation is not guaranteed to be able to meet the throughput constraint.

An extension of the heuristic-based resource allocation strategy introduced in [38] is used to find a feasible mapping that results in a PD close to the target PD. The binding algorithm is summarized in Algorithm 2. First, all actors are sorted on criticality in descending order. The criticality is calculated as a measure for the worst case. Next to approaching the target PD, there might also be other optimization targets, such as computational LB among the tiles, memory usage balancing or communication balancing/minimization. Function *cost*$(t, a)$, defined in (9.8), assigns a cost to binding actor $a$ to tile $t$, which is used in the binding algorithm. Constants $c_1, \ldots, c_6$ weight the costs of the different optimization criteria:

$$\begin{aligned} cost(t,a) &= c_1 \times P(t,a) + c_2 \times M(t,a) + c_3 \times C(t,a) + c_4 \times L(t,a) \\ &\quad + c_5 \times PDT(t,a) + c_6 \times PDS(t,a) \end{aligned} \tag{9.8}$$

- $P(t,a) \in [0,1]$ is the normalized processor load when binding actor $a$ to tile $t$;
- $M(t,a) \in [0,1]$ is the ratio of allocated memory when $a$ is bound to $t$;
- $C(t,a) \in [0,1]$ is the ratio of allocated connections on tile $t$ when $a$ is bound to $t$;
- $L(t,a) \in [0,1]$ is the normalized average latency of all connections from/to $a$ when $a$ is bound to $t$;
- $PRT(t,a) \in [0,1]$ is the normalized cost for the power ratio of tile $t$ when $a$ is bound to $t$

  $$PRT(t,a) = c \cdot \left(\frac{r_t}{R_t}\right)$$

  where $r_t$ is the estimated power ratio of tile $t$ when binding $a$ to $t$ and $R_t$ is the target power ratio of tile $t$. $c$ is a normalizing constant to scale the cost to $[0,1]$ for all tiles; and
- $PRS(t,a) \in [0,1]$ is the normalized cost for the power ratio of the tile stack $s$ containing tile $t$ when mapping $a$ to $t$.

  $$PRS(t,a) = c \cdot \left(\frac{r_s}{R_s}\right)$$

  where $r_s$ is the estimated power ratio of stack $s$ when binding $a$ to $t$ and $R_s$ is the target power ratio of stack $s$. $c$ is a normalizing constant to scale the cost to $[0,1]$ for all stacks. A tile stack is a set containing all tiles that are in the same horizontal position at different layers. The target power ratio of a stack is defined by the sum of the ratios of the tiles it contains.

**Algorithm 2:** Tile binding algorithm

```
Input: tiles $T$, actors $A$, connections $C$.
Output: Feasible resource allocation.
// Find an initial binding
Sort all actors $a \in A$ on criticality, descending;
for all sorted actors $a \in A$ do
    Sort all tiles $t \in T$ on $cost(t, a)$, ascending (see (9.8));
    for all sorted tiles $t \in T$ do
        if binding actor a to tile t is feasible then
            Bind actor $a$ to tile $t$;
            Bind connections to/from $a$;
            break;
        end
    end
    if actor a not bound then
        return “Unable to find feasible binding”;
    end
end

// Try to improve the binding
Sort all actors $a \in A$ on criticality, ascending;
for all sorted actors $a \in A$ do
    Unbind actor $a$;
    Unbind connections to/from $a$;
    Sort all tiles $t \in T$ on $cost(t, a)$, ascending (see (9.8));
    for all sorted tiles $t \in T$ do
        if binding actor a to tile t is feasible then
            Bind actor $a$ to tile $t$;
            Bind all related connections;
            break;
        end
    end
end
```

Note that there are two terms related to the PD: $PRT(t, a)$ and $PRS(t, a)$. $PRT$ represents the deviation from the original target power ratios of the individual tiles. Since we observed that there is a large thermal correlation between vertically adjacent tiles, it makes sense to also take the target power ratios of stacks of tiles into account, which is captured by the $PRS$ term. If the PD among stacks would not be included, deviations from the target PD in the vertical direction would result in the same cost as deviations in the horizontal direction, leading to worse results in a thermal sense.

The tile power ratios resulting from a candidate binding can be calculated since the active and idle powers of all tiles are known, along with the execution time of every actor on every possible tile.

#### 9.2.3.5 Throughput computation

When a feasible resource allocation has been found, the maximum throughput of the mapped application has to be calculated in order to validate the throughput constraint. This is done by modeling the mapped application as a binding-aware SDFG and performing a state-space exploration by simulating the self-timed execution of the graph [36]. The throughput is calculated as soon as a recurrent state is found during the execution.

#### 9.2.3.6 Utilization minimization

It is possible that the maximum throughput of the mapped application is higher than the throughput constraint. From an energy and temperature perspective, it makes sense to slow down the execution as long as the throughput constraint is satisfied. This is done in the utilization minimization step. It is assumed that every processor contains a TDMA system, in which a time slice can be reserved during which the processor is idle. To slow down execution, an idle timeslice is inserted in the TDMA schedule of every active processor. The appropriate sizes of the idle time slices are determined by performing a binary search and recalculating the throughput after every step. The search is terminated once the actual throughput is not more than 10% above the throughput constraint.

### *9.2.4 Experimental results*

The performance of the proposed approach is tested by applying it to a set of synthetic benchmark applications as well as a set of real-life multimedia applications.

#### 9.2.4.1 Benchmark applications

To evaluate the performance of the thermal-aware mapping approach, a set consisting of four application graphs is generated. Every application consists of eight actors with random (Gaussian distributed) execution times and storage requirements. To evaluate the effect of the weights in the tile binding cost function, multiple mappings are generated for each set of applications, based on different cost function weights. To eliminate the effects of the random generator, three sets of applications are generated. Next to the synthetic benchmark application set, a real-life application set consisting of four independent H.263 encoders (five actors each) with a throughput constraint of 60 frames per second is constructed.

#### 9.2.4.2 Target 3D many-core system

The sets of benchmark applications are mapped on a tile-based 3D many-core system consisting of three layers of $2 \times 2$ identical tiles. Each tile consists of a processor, memory and NI, all modeled as a single block as depicted in Figure 9.7. The active power of each tile is set to 1.5 W, the idle power is set to 10% of the active power. For the placement of blocks in a tile, two different tile floorplans are used, such that the processor blocks do not overlap. Tile floorplan 1 is used on the bottom and top layer, while floorplan 2 is used on the middle layer to avoid stacking all processor blocks exactly on top of each other, since the power density is the highest in that block. The heat sink is connected (via a heat spreader) to the bottom layer. The other active (power dissipating) layers are thinned down to 50 μm. Between two active layers, a 10 μm thin layer containing TIM is modeled. The most important physical properties of the 3D IC model are listed in Table 9.1. In the center of the NI block of every tile, a bundle of $8 \times 9$ TSVs is placed. For the interconnect, a hybrid NoC-Bus design is assumed, consisting of a regular NoC in the horizontal plane and a multi-drop shared bus for vertical communication [25]. In this setup, every tile is assumed to have its own NoC switch, and every stack of tiles contains a shared bus. In the architecture graph, communication links are modeled as point-to-point connections. The latencies of all possible tile-to-tile connections are calculated based on the delay of the shortest path between the tiles. A hop in the horizontal plane is modeled as a delay of 2 time units, a hop in the vertical direction as 1 time unit.

#### 9.2.4.3 Temperature simulation

For every mapping, an execution trace of 0.5 s is generated. The execution patterns are periodic with a period much shorter than 0.5 s, making longer simulations obsolete. From the execution trace and the architecture specification, power traces are derived for every block. The power traces are used in the modified HotSpot 5.02 thermal simulator to simulate the temperature with a grid resolution of $32 \times 32$ and a temporal resolution of 10 μs. First, a steady state simulation is performed to find a representative initial temperature distribution. Next, the transient temperature simulation is performed. Table 9.1 lists the most important HotSpot parameters.

*Table 9.1 Physical properties and HotSpot parameters*

| **Parameter** | **Value** |
|---|---|
| Tile size [mm] | $2 \times 2$ |
| Silicon thermal conductance [W/(m K)] | 150 |
| Silicon specific heat [J/(m$^3$ K)] | $1.75 \cdot 10^6$ |
| TIM thermal conductance [W/(m K)] | 4 |
| TIM specific heat [J/(m$^3$ K)] | $4 \cdot 10^6$ |
| TSV thermal conductance [W/(m K)] | 300 |
| TSV specific heat [J/(m$^3$ K)] | $3.5 \cdot 10^6$ |
| TSV diameter [μm] | 10 |
| TSV pitch [μm] | 20 |
| Bottom layer thickness [μm] | 200 |
| Non-bottom layer thickness [μm] | 50 |
| TIM layer thickness [μm] | 10 |
| Convection resistance to ambient [K/W] | 3.0 |
| Heat sink side/thickness [mm] | $14 \times 14 \times 10$ |
| Heatsink conductance [W/(m K)] | 400 |
| Heat sink specific heat [J/(m$^3$ K)] | $3.55 \cdot 10^6$ |
| Ambient temperature [K] | 300 |

#### 9.2.4.4 Interconnect energy computation

Since the interconnect energy consumption can be a significant part of the total energy consumption [39], it is also interesting to investigate the communication intensity and interconnect energy consumption resulting from different mappings. Note that the computational energy consumption will be close to identical for all mappings, since a homogeneous architecture is considered and every application is slowed down to match the throughput constraint.

The interconnect consumes energy to facilitate communication between the titles and consumed energy is also referred to as communication energy. In between two tiles, communication has to take place when actors (tasks) mapped on them need to communicate with each other. The communication energy depends on the data volume and the relative locations of the communicating task (actor) pair. For each communicating task pair mapped to tile $i$ and tile $j$ and connected by edge $e$, the communication energy is estimated by the product of the number of transferred bits ($nrTokens[e] \times tokenSize[e]$) and the energy required to transfer one bit between tiles $i$ and $j$ ($E_{bit}(i,j)$), as defined in (9.9). The value of $E_{bit}(i,j)$ is calculated based on the energy required for horizontal link traversals, vertical link traversals and the energy consumed in routers between tiles $i$ and $j$, as shown in (9.10). Vertical interconnects are implemented as shared buses, so no intermediate routers are involved when traversing multiple layers. Therefore, hops in the vertical direction will increase the total number of routings by just 1, independent of the number of vertical hops. The total communication energy is estimated by summing over all communicating task pairs (edges).

$$E_{comm}(e) = \big(nrTokens[e] \times tokenSize[e]\big) \times E_{bit}(i,j) \tag{9.9}$$

$$\begin{aligned} E_{bit}(i,j) = {} & \big(E_{bit}^{horizontal} \times hops_{horizontal}(i,j)\big) + \big(E_{bit}^{vertical} \times hops_{vertical}(i,j)\big) \\ & + \big(E_{bit}^{router} \times numOfRouters(i,j)\big) \end{aligned} \tag{9.10}$$

In our 3D IC model, the horizontal link energy per bit, $E_{bit}^{horizontal}$, is taken as 0.127 PJ, which is estimated from [25]. The vertical link energy per bit, $E_{bit}^{vertical}$, is determined by the used TSVs and is therefore referred to as $E_{bit}^{TSV}$. $E_{bit}^{TSV}$ is estimated to be $9.56 \times 10^{-3}$ PJ [40]. For a horizontal link length of 2 mm, the per bit router energy $E_{bit}^{router}$ is approximately 70% of $E_{bit}^{horizontal}$ [41]. $E_{bit}^{TSV}$ is only 7.5% of $E_{bit}^{horizontal}$, providing substantial space for communication energy optimization by exploiting the low link energy in the vertical direction. However, using more vertical links may result in a higher peak temperature due to increased power density because of mapping communicating tasks on stacked tiles.

#### 9.2.4.5 Thermal profiling results

The target PD obtained by running the thermal profiling algorithm on the considered 3D IC is given in Table 9.2. It is observed that the power is almost completely balanced in the horizontal plane, indicating that the power dissipated in a stack of tiles is

*Table 9.2 Target power distribution for the considered 3D IC*

| | **Tile position in the horizontal plane** | | | |
|---|---|---|---|---|
| | **(0,0)** | **(0,1)** | **(1,0)** | **(1,1)** |
| Top layer (%) | 2.8 | 2.8 | 2.7 | 2.4 |
| Middle layer (%) | 7.5 | 7.6 | 7.4 | 6.9 |
| Bottom layer (%) | 15.0 | 15.3 | 15.0 | 14.7 |

minimized. In the static PD that minimizes the peak temperature on this specific 3D IC, about 60% of the total chip power is dissipated in the bottom layer. About 29% is dissipated in the middle layer, and the remaining 11% is dissipated in the top layer. Further experiments show that this distribution mainly depends on the PD within the tiles, the layer thickness and the interlayer bonds. Although the thermal properties of the heat sink have a large impact on the average chip temperature, the optimal PD is almost independent from it.

#### 9.2.4.6 Benchmark application results

To evaluate the performance of the resource allocation strategy, five different combinations of tile-binding cost-function weights are evaluated, corresponding to the following different optimization objectives:

1. **Load balancing (LB)**:
   $(c_1, c_2, c_3, c_4, c_5, c_6) = (1, 0, 0, 0, 0, 0)$. Balance the computational load as much as possible.
2. **Communication latency minimization (CLM)**:
   $(c_1, c_2, c_3, c_4, c_5, c_6) = (0, 0, 0, 1, 0, 0)$. Minimize the interconnect latency.
3. **Load balancing + latency minimization (LB+CLM)**:
   $(c_1, c_2, c_3, c_4, c_5, c_6) = (1, 0, 0, 1, 0, 0)$. Combine computational LB and latency minimization with equal weights.
4. **Power balancing by stack (PBS)**:
   $(c_1, c_2, c_3, c_4, c_5, c_6) = (0, 0, 0, 0, 0, 1)$. The power ratios of all tile stacks are set equal, causing power balancing in the horizontal plane.
5. **Optimize power distribution (PD)**:
   $(c_1, c_2, c_3, c_4, c_5, c_6) = (0, 0, 0, 0, 1, 1)$. Optimize for a PD close to the target distribution.
6. **Optimize power distribution + latency minimization (PD+CLM)**:
   $(c_1, c_2, c_3, c_4, c_5, c_6) = (0, 0, 0, 1, 1, 1)$. Combine PD optimization with latency minimization.

*Temperature results*

Figure 9.11 shows the lowest and highest observed temperatures resulting from mapping the benchmark application set using the different optimization objectives. All mappings result in a throughput within 10% above the constraint. The results are averaged over the three application sets to remove effects of the random generator.

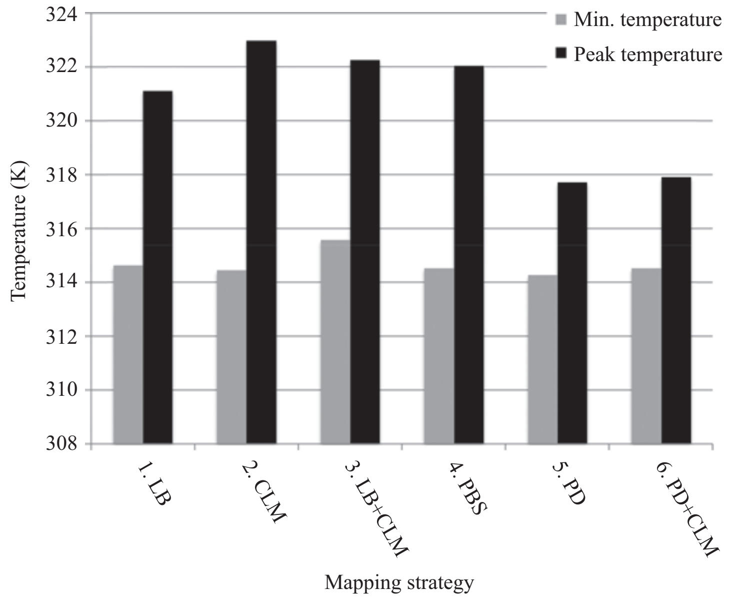


*Figure 9.11 Minimum and peak temperature resulting from mappings with different optimization objectives [23]*

It is clear that trying to minimize the communication latency alone (scenario 2) results in the highest peak temperature. This is due to the fact that vertically adjacent tiles have smaller communication delays, which results in communicating actors being mapped on vertically adjacent tiles. This can cause a power imbalance in the horizontal plane, explaining the increased temperature. Including the target power ratio terms leads to a peak temperature decrease of 5.3K compared to the latency minimization case and 3.4K compared to the computational LB case. It is clear that only balancing the load in the horizontal plane (scenario 4) does not result in the minimum temperature.

### *Interconnect usage and energy consumption results*

Figure 9.12 depicts the average normalized number of bit hops, as well as the average interconnect power consumption for different optimization criteria. A bit hop is defined as 1 bit of data that is transferred 1 hop through the NoC. The interconnect power is estimated as the average communication energy per second. It can be observed that including the latency cost term results in a significant decrease in interconnect utilization. This can be explained by the observation that the latency cost term assigns high costs to mappings in which communicating actors are mapped on tiles that are far apart in the 3D NoC. In scenario 6, the interconnect utilization is roughly halved compared to scenario 5, with almost no increase in peak temperature (Figure 9.11).

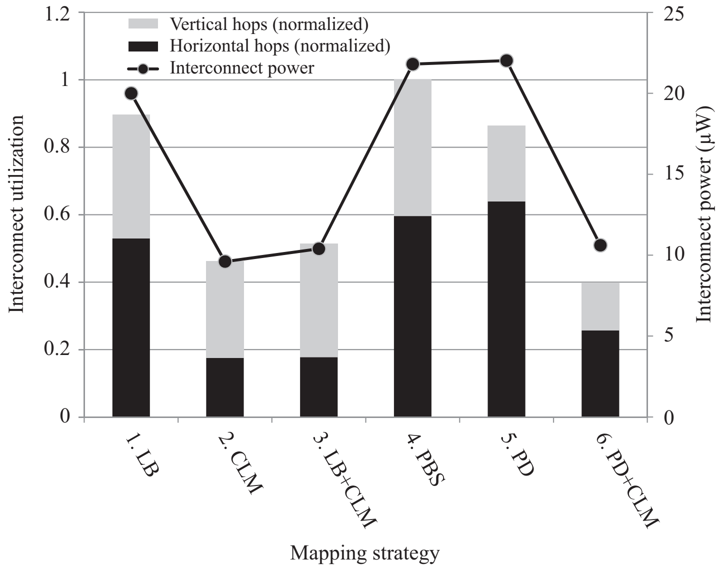


*Figure 9.12 Average normalized horizontal/vertical interconnect utilization and interconnect power [23]*

The average interconnect power depends on the usage of the horizontal and vertical interconnect links, as can be seen in Figure 9.12. A couple of observations can be made from Figure 9.12. First, the total power consumption is mainly governed by the usage of horizontal links, as they consume much more power than the vertical links. Second, the power consumption for the *CLM* and *LB+CLM* strategies are low due to more usage of vertical links than horizontal links, but they show high peak temperatures (Figure 9.11). This indicates that using more vertical links instead of horizontal links facilitates lower interconnect power consumption at the cost of a higher peak temperature due to more power stacking within the chip. Thus, a trade-off exists between minimizing interconnect energy consumption and peak temperature. Third, the *PD+CLM* strategy shows almost the same power consumption as the *CLM* and *LB+CLM* strategies, along with lower interconnect utilization and peak temperature. This indicates that the *PD+CLM* strategy results in a good balance between peak temperature, interconnect utilization and interconnect energy consumption. The absolute interconnect energy reduction highly depends on the communication intensity of the applications as well as the NoC type and technology.

#### 9.2.4.7 Case-study for real-life applications

To test the applicability of our approach for real-life applications, four independent H.263 encoder applications are mapped on the 3D many-core system using our mapping flow. Two different mapping strategies are applied: *LB+CLM* and *PD+CLM*, as introduced earlier. The *LB+CLM* strategy tries to balance the computational load

*Table 9.3 Average interconnect power consumption, minimum and maximum temperature for four independent H.263 encoder applications*

| | **LB+CLM** | **PD+CLM** |
|---|---|---|
| Interconnect power consumption (μW) | 166.62 | 85.10 |
| Minimum temperature (K) | 310.75 | 310.15 |
| Maximum temperature (K) | 317.05 | 314.25 |

while minimizing the interconnect latency, whereas the *PD+CLM* strategy aims at optimizing the PD while also minimizing interconnect latency.

Table 9.3 shows the average interconnect power consumption, minimum and maximum temperature when the mapping strategies *LB+CLM* and *PD+CLM* are employed. Strategy *PD+CLM* outperforms strategy *LB+CLM* for all the performance figures, i.e., it results in a lower interconnect power consumption, minimum temperature and maximum temperature. Minimizing the communication latency (*CLM*) results in a significant reduction in interconnect utilization and interconnect power consumption (Figure 9.12), making it an important optimization criterion to be considered. The results in Table 9.3 indicate that in addition to minimizing the communication latency, optimizing the *PD* is a better choice than balancing the computational load (*LB*).

## 9.3 Conclusions and future directions

We proposed a flexible and fast approach for thermal-aware mapping of throughput-constrained streaming applications on 3D many-core systems. As compared to the LB case, the proposed approach reduces the peak temperature by 7% (in °C) and interconnect energy consumption by 47% for a set of benchmark applications on a three-layer IC, while meeting all storage and throughput constraints. We showed that the approach can also be used in combination with other optimization criteria, such as interconnect utilization minimization. The average runtime of the total flow is 20 min, with about 90% being spent in the thermal profiling step. The runtime of the resource allocation and throughput validation step highly depends on the size and complexity of the application graph.

Despite the significant progress on thermal management for 2D and 3D ICs, there are open challenges that need to be addressed in future. One emerging challenge is the dissipation of heat in the absence of atmosphere, e.g., when systems are employed in space. Specifically, it is important to manage the thermal aspects more aggressively in such situations, otherwise a simple task can lead to the system temperature to surpass the safe limit, leading to damaging the system. Recently, significant efforts are been made to improve the thermal dissipation of many-core systems. A thermal management of battery system using novel phase-change materials is proposed in [42].

Two-phase liquid cooling is proposed for 3D systems [43]. Optimal placement of liquid microchannels for heat management of 3D ICs is discussed in [44]. Integrated flow cells are used in [45] for quick heat dissipation. Integrated microfluidic power generation and cooling is described in [46]. For a comprehensive study of these techniques, readers are referred to [47].

Another possible solution is to harvest useful energy from heat, mitigating the impact. In [48,49], energy harvesting from thermoelectrics is discussed for thermal management of many-core systems. Self-powered distributed networks is proposed in [50]. A new design of energy-harvesting systems is proposed in [51]. A scheduling technique for mixed critical tasks is discussed in [52] for heterogeneous many-core systems powered by energy harvesting sources. There is also approach for scheduling sporadic tasks on systems with energy harvesting sources [53]. Readers are referred to [54] for a survey on recent advances in thermal management using energy-harvesting sources.

For a 3D IC, the positive/negative impact of increasing the number of stacked layers on the temperature needs to be addressed in future. Simultaneous multilayer access in a 3D system is discussed in [55]. Placement of the 3D layers is described in [56]. Simulated annealing-based layer placement is proposed for 3D systems in [57]. Readers are referred to [58] for a survey of the challenges in layers optimization of 3D many-core systems.

Finally, the design-space exploration needs to be extended to consider emerging design principles such as approximate computing. An online quality-management system for approximate computing is discussed in [59]. A hardware–software framework for approximate computing is proposed in [60]. Recent works that propose approximate arithmetic blocks such as multiplier and adders may also be considered for low-area and low-power implementations of the DSE algorithms [61,62]. Readers are referred to [63] for a survey of architectural techniques for approximate computing.

## References

[1] Jung H, Pedram M. Continuous frequency adjustment technique based on dynamic workload prediction. In: Proceedings of the International Conference on VLSI Design; 2008. p. 249–254.

[2] Das A, Al-Hashimi BM, Merrett GV. Adaptive and hierarchical runtime manager for energy-aware thermal management of embedded systems. ACM Transactions on Embedded Computing Systems (TECS). 2016;15(2):24.

[3] Das A, Shafik RA, Merrett GV, *et al.* Reinforcement learning-based inter-and intra-application thermal optimization for lifetime improvement of multicore systems. In: Proceeding of the Annual Design Automation Conference (DAC). ACM; 2014. p. 1–6.

[4] Das A, Kumar A, Veeravalli B. Reliability and energy-aware mapping and scheduling of multimedia applications on multiprocessor systems. IEEE Transactions on Parallel and Distributed Systems. 2016;27(3):869–884.

[5] Ma Y, Chantem T, Dick RP, *et al.* Improving system-level lifetime reliability of multicore soft real-time systems. IEEE Transactions on Very Large Scale Integration (VLSI) Systems. 2017;25(6):1895–1905.

[6] Yang Z, Serafy C, Lu T, *et al.* Phase-driven learning-based dynamic reliability management for multi-core processors. In: Proceeding of the Annual Design Automation Conference (DAC). ACM; 2017. p. 46.

[7] Dhiman G, Rosing TS. Dynamic voltage frequency scaling for multi-tasking systems using online learning. In: Proceedings of the ACM/IEEE International Symposium on Low Power Electronics and Design (ISLPED). New York, NY, USA: ACM; 2007. p. 207–212.

[8] Shen H, Lu J, Qiu Q. Learning based DVFS for simultaneous temperature, performance and energy management. In: Proceedings of the International Symposium on Quality Electronic Design (ISQED); 2012. p. 747–754.

[9] Shen H, Tan Y, Lu J, *et al.* Achieving autonomous power management using reinforcement learning. ACM Transactions on Design Automation of Electronic Systems (TODAES). 2013;18(2):24:1–24:32.

[10] Ye R, Xu Q. Learning-based power management for multicore processors via idle period manipulation. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD). 2014;33(7):1043–1055.

[11] Dhiman G, Kontorinis V, Tullsen D, *et al.* Dynamic workload characterization for power efficient scheduling on CMP systems. In: Proceedings of the ACM/IEEE International Symposium on Low Power Electronics and Design (ISLPED); 2010. p. 437–442.

[12] Jung H, Pedram M. Supervised learning based power management for multicore processors. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD). 2010;29(9):1395–1408.

[13] Cochran R, Hankendi C, Coskun AK, *et al.* Pack & Cap: Adaptive DVFS and thread packing under power caps. In: Proceedings of the IEEE/ACM International Symposium on Microarchitecture (MICRO). ACM; 2011. p. 175–185.

[14] Cochran R, Hankendi C, Coskun A, *et al.* Identifying the optimal energy-efficient operating points of parallel workloads. In: Proceedings of the International Conference on Computer Aided Design (ICCAD); 2011. p. 608–615.

[15] Mercati P, Bartolini A, Paterna F, *et al.* Workload and user experience-aware dynamic reliability management in multicore processors. In: Proceeding of the Annual Design Automation Conference (DAC). ACM; 2013. p. 2:1–2:6.

[16] Pallipadi V, Starikovskiy A. The ondemand governor. In: Proceedings of the Linux Symposium. vol. 2; 2006. p. 215–230.

[17] Das A, Kumar A, Veeravalli B, *et al.* Workload uncertainty characterization and adaptive frequency scaling for energy minimization of embedded systems. In: Proceedings of the Conference on Design, Automation and Test in Europe (DATE). IEEE; 2015. p. 43–48.

[18] Michie D, Spiegelhalter DJ, Taylor CC. Machine Learning, Neural and Statistical Classification. New York, NY: Ellis Horwood; 1994.

[19] Anderson JA, Richardson SC. Logistic discrimination and bias correction in maximum likelihood estimation. Technometrics. 1979;21(1):71–78.

[20] Derf. Test Media; 2014. Available from: http://media.xiph.org/video.

[21] Knickerbocker JU, Patel CS, Andry PS, *et al.* 3-D silicon integration and silicon packaging technology using silicon through-vias. IEEE Journal of Solid-State Circuits. 2006;41(8):1718–1725.

[22] Feero BS, Pande PP. Networks-on-chip in a three-dimensional environment: A performance evaluation. IEEE Transactions on Computers. 2009;58(1):32–45.

[23] Cox M, Singh AK, Kumar A, *et al.* Thermal-aware mapping of streaming applications on 3D Multi-Processor Systems. In: IEEE Symposium on Embedded Systems for Real-time Multimedia (ESTIMedia). IEEE; 2013. p. 11–20.

[24] Lee EA, Messerschmitt DG. Static scheduling of synchronous data flow programs for digital signal processing. IEEE Transactions on Computers. 1987;100(1):24–35.

[25] Cheng Y, Zhang L, Han Y, *et al.* Thermal-constrained task allocation for interconnect energy reduction in 3-D homogeneous MPSoCs. IEEE Transactions on Very Large Scale Integration (VLSI) Systems. 2013;21(2): 239–249.

[26] Skadron K, Stan MR, Sankaranarayanan K, *et al.* Temperature-aware microarchitecture: Modeling and implementation. ACM Transactions on Architecture and Code Optimization (TACO). 2004;1(1):94–125.

[27] Coskun AK, Ayala JL, Atienza D, *et al.* Dynamic thermal management in 3D multicore architectures. In: Design, Automation & Test in Europe Conference & Exhibition, 2009. DATE'09. IEEE; 2009. p. 1410–1415.

[28] Zhou X, Yang J, Xu Y, *et al.* Thermal-aware task scheduling for 3D multicore processors. IEEE Transactions on Parallel and Distributed Systems. 2010;21(1):60–71.

[29] Addo-Quaye C. Thermal-aware mapping and placement for 3-D NoC designs. In: SOC Conference, 2005. Proceedings. IEEE International. IEEE; 2005. p. 25–28.

[30] Chantem T, Hu X, Dick RP. Temperature-aware scheduling and assignment for hard real-time applications on MPSoCs. IEEE Transactions on Very Large Scale Integration (VLSI) Systems. 2011;19(10):1884–1897.

[31] Sun C, Shang L, Dick RP. Three-dimensional multiprocessor system-on-chip thermal optimization. In: Hardware/Software Codesign and System Synthesis (CODES+ ISSS), 2007 5th IEEE/ACM/IFIP International Conference on. IEEE; 2007. p. 117–122.

[32] Thiele L, Schor L, Bacivarov I, *et al.* Predictability for timing and temperature in multiprocessor system-on-chip platforms. ACM Transactions on Embedded Computing Systems. 2013;12(1s):48:1–48:25. Available from: http://doi.acm.org/10.1145/2435227.2435244.

[33] Nookala V, Lilja DJ, Sapatnekar SS. Temperature-aware floorplanning of microarchitecture blocks with IPC-power dependence modeling and transient analysis. In: Proceedings of the 2006 International Symposium on Low Power Electronics and Design. ACM; 2006. p. 298–303.

[34] Zhou P, Ma Y, Li Z, *et al.* 3D-STAF: Scalable temperature and leakage aware floorplanning for three-dimensional integrated circuits. In: Computer-Aided Design, 2007. ICCAD 2007. IEEE/ACM International Conference on. IEEE; 2007. p. 590–597.

[35] Pathak M, Lim SK. Thermal-aware Steiner routing for 3D stacked ICs. In: Computer-Aided Design, 2007. ICCAD 2007. IEEE/ACM International Conference on. IEEE; 2007. p. 205–211.

[36] Ghamarian AH, Geilen M, Stuijk S, *et al.* Throughput analysis of synchronous data flow graphs. In: Application of Concurrency to System Design, 2006. ACSD 2006. Sixth International Conference on. IEEE; 2006. p. 25–36.

[37] Stuijk S, Geilen M, Basten T. SDF^3: SDF for free. In: Application of Concurrency to System Design, 2006. ACSD 2006. Sixth International Conference on. IEEE; 2006. p. 276–278.

[38] Stuijk S, Basten T, Geilen M, *et al.* Multiprocessor resource allocation for throughput-constrained synchronous dataflow graphs. In: Design Automation Conference, 2007. DAC'07. 44th ACM/IEEE. IEEE; 2007. p. 777–782.

[39] Hoskote Y, Vangal S, Singh A, *et al.* A 5-GHz mesh interconnect for a teraflops processor. IEEE Micro. 2007;27(5):51–61.

[40] International Technology Roadmap for Semiconductors; 2010. Available from: http://www.itrs.net/reports.html.

[41] Bhat S. Energy Models for Network-on-Chip Components [MSc. thesis]. Eindhoven University of Technology; 2005.

[42] Wang Q, Rao Z, Huo Y, *et al.* Thermal performance of phase change material/oscillating heat pipe-based battery thermal management system. International Journal of Thermal Sciences. 2016;102:9–16.

[43] Chiou HW, Lee YM. Thermal simulation for two-phase liquid cooling 3D-ICs. Journal of Computer and Communications. 2016;4(15):33.

[44] Dash R, Pangracious V, Risco-Mart JL, *et al.* Thermal management in 3D homogeneous NoC systems using optimized placement of liquid microchannels. In: Embedded Multicore/Many-core Systems-on-Chip (MCSoC), 2017 IEEE 11th International Symposium on. IEEE; 2017. p. 37–44.

[45] Andreev AA, Sridhar A, Sabry MM, *et al.* PowerCool: Simulation of cooling and powering of 3D MPSoCs with integrated flow cell arrays. IEEE Transactions on Computers. 2018;67(1):73–85.

[46] Sabry MM, Sridhar A, Atienza D, *et al.* Integrated microfluidic power generation and cooling for bright silicon MPSoCs. In: Design, Automation and Test in Europe Conference and Exhibition (DATE), 2014. IEEE; 2014. p. 1–6.

[47] Murshed SS, de Castro CN. A critical review of traditional and emerging techniques and fluids for electronics cooling. Renewable and Sustainable Energy Reviews. 2017;78:821–833.

[48] Jayakumar S, Reda S. Making sense of thermoelectrics for processor thermal management and energy harvesting. In: Low Power Electronics and Design (ISLPED), 2015 IEEE/ACM International Symposium on. IEEE; 2015. p. 31–36.

[49] Lee Y, Kim E, Shin KG. Efficient thermoelectric cooling for mobile devices. In: Low Power Electronics and Design (ISLPED, 2017 IEEE/ACM International Symposium on. IEEE; 2017. p. 1–6.

[50] Brunelli D, Passerone R, Rizzon L, *et al.* Self-powered WSN for distributed data center monitoring. Sensors. 2016;16(1):57.

[51] Merrett GV, Al-Hashimi BM. Energy-driven computing: Rethinking the design of energy harvesting systems. In: Proceedings of the Conference on Design, Automation & Test in Europe. European Design and Automation Association; 2017. p. 960–965.

[52] Xiang Y, Pasricha S. Mixed-criticality scheduling on heterogeneous multicore systems powered by energy harvesting. Integration. 2018;61:114–124.

[53] Housseyni W, Mosbahi O, Khalgui M, *et al.* Real-time scheduling of sporadic tasks in energy harvesting distributed reconfigurable embedded systems. In: Computer Systems and Applications (AICCSA), 2016 IEEE/ACS 13th International Conference of. IEEE; 2016. p. 1–8.

[54] Zhang Y. Improving the Efficiency of Energy Harvesting Embedded System. Syracuse University; 2016.

[55] Lee D, Ghose S, Pekhimenko G, *et al.* Simultaneous multi-layer access: Improving 3D-stacked memory bandwidth at low cost. ACM Transactions on Architecture and Code Optimization (TACO). 2016;12(4):63.

[56] Banerjee S, Majumder S, Varma A, *et al.* A placement optimization technique for 3D IC. In: Embedded Computing and System Design (ISED), 2017 7th International Symposium on. IEEE; 2017. p. 1–5.

[57] Zhu HY, Zhang MS, He YF, *et al.* Floorplanning for 3D-IC with through-silicon via co-design using simulated annealing. In: 2018 IEEE International Symposium on Electromagnetic Compatibility and 2018 IEEE Asia-Pacific Symposium on Electromagnetic Compatibility (EMC/APEMC). IEEE; 2018. p. 550–553.

[58] Chan WTJ, Kahng AB, Li J. Revisiting 3DIC benefit with multiple tiers. Integration, the VLSI Journal. 2017;58:226–235.

[59] Khudia DS, Zamirai B, Samadi M, *et al.* Rumba: An online quality management system for approximate computing. In: Computer Architecture (ISCA), 2015 ACM/IEEE 42nd Annual International Symposium on. IEEE; 2015. p. 554–566.

[60] Mishra AK, Barik R, Paul S. iACT: A software-hardware framework for understanding the scope of approximate computing. In: Workshop on Approximate Computing Across the System Stack (WACAS); 2014.

[61] Ullah S, Rehman S, Prabakaran BS, *et al.* Area-optimized low-latency approximate multipliers for FPGA-based hardware accelerators. In: Proceedings of the 55th Annual Design Automation Conference. ACM; 2018. p. 159.

[62] Ullah S, Murthy SS, Kumar A. SMApproxlib: Library of FPGA-based approximate multipliers. In: Proceedings of the 55th Annual Design Automation Conference. ACM; 2018. p. 157.

[63] Mittal S. A survey of techniques for approximate computing. ACM Computing Surveys (CSUR). 2016;48(4):62.