Reliability and Energy-Aware Mapping and Scheduling of Multimedia Applications on Multiprocessor Systems

Anup Das, *Member, IEEE,* Akash Kumar, *Senior Member, IEEE,* and Bharadwaj Veeravalli, *Senior Member, IEEE,*

Abstract—Lifetime reliability is an emerging concern in multiprocessor systems as escalating power density and hence temperature variation continues to accelerate wear-out leading to a growing prominence of device defects. In this paper, we propose a system-level approach that involves performance-aware mapping of multimedia applications on a multiprocessor system to jointly minimize energy consumption and temperature related wear-out. Fundamental to this approach is a simplified temperature model that incorporates not only the transient and the steady-state behavior (temporal effect), but also the temperature dependency on the surrounding cores (spatial effect). This model is validated against the temperature obtained using the *HotSpot* tool with transient and steady-state simulations, and is shown to be accurate within 5.5°C, leading to an MTTF estimation accuracy of an average 21% with respect to the state-of-the-art approaches. The proposed temperature model is integrated in a gradient-based fast heuristic that controls the voltage and frequency of the cores to limit the average and peak temperature leading to a longer lifetime, simultaneously minimizing the energy consumption. Lifetime computation considers task remapping, which is a common feature available in modern multiprocessor systems. A linear programming approach is then proposed to distribute the cores of a multiprocessor system among concurrent applications to maximize the lifetime. Experiments conducted with a set of synthetic and real-life applications represented as synchronous data flow graphs demonstrate that the proposed approach minimizes energy consumption by an average 24% with 47% increase in lifetime. For concurrent applications, the proposed lifetime-aware core distribution results in an average 10% improvement in lifetime as compared to performance-based core distribution.

Index Terms—Lifetime reliability, mean time to failure (MTTF), platform-based design, synchronous data flow graphs.

1 Introduction

Lifetime reliability is a crucial design concern for modern multiprocessor systems as escalating power density and hence temperature variation continues to accelerate wear-out, leading to an increase in device defects [1]. This has attracted significant attention both in the industry and academia to investigate platform-based design approaches, which involve system-level techniques such as mapping and scheduling of applications on a given multiprocessor platform to mitigate wear-out leading to an extended mean time to failure (MTTF) [2]–[5]. These studies do not minimize energy consumption, which is also crucial for battery-operated embedded devices. Modern cores support a wide range of voltages and frequencies, which are often exploited to perform dynamic voltage and frequency scaling (DVFS) to minimize the task computation energy [6], [7]. This has motivated researchers in recent years to jointly optimize lifetime reliability and energy consumption through intelligent task mappings [8]–[10]. The existing works on energy-reliability joint optimization suffer from the following two limitations – accuracy and scope.

Accuracy: Lifetime estimation and optimization at design-time require predicting the thermal behavior of an application. *HotSpot* [11] is the standard and the most prevalent thermal simulation tool used in academia. However, integrating this tool directly in the design space exploration process leads to a super exponential exploration time, even for a moderate problem size (in terms of the number of tasks and cores). This is due to the large simulation time of the *HotSpot* tool. Some of the recent studies addressed this scalability problem by modeling the heat transfer phenomena using a resistive-capacitive (RC) equivalent model. Although the RC equivalent thermal model can be solved directly to determine the temperature of a core, the solutions are too computation intensive to be incorporated in the design space exploration framework. Steady-state approximation simplifies the solution, but is accurate if the execution times of the tasks are comparable to the thermal time constant of the package, which is typically of the order of few hundreds of seconds. Finally, ignoring the spatial dependency leads to simplification of the RC equivalent model, but results in underestimation of the temperature and a corresponding overestimation of the mean time to failure. Additionally, some existing studies on lifetime reliability approximate MTTF as the time to the first fault. This is true for systems that are not provisioned to tolerate faults. In this work, multiprocessor systems are considered with support for task migration. Such a system continues to operate in the presence of failures, albeit an acceptable performance degradation. For such systems, estimating the MTTF as the time to first failure leaves a significant scope of improvement, both in terms of lifetime and energy consumption.

Scope: The existing lifetime optimization techniques are all based on sequential execution of applications represented as directed acyclic graphs (DAGs) that are not sufficiently expressive to model streaming multimedia and other data flow applications. This class of applications requires support for modeling cyclic dependency, multi-input tasks, multi-rate tasks, pipelined execution and a natural way for dealing with latency and buffer requirements. As discussed in Section 3, synchronous data flow graphs (SDFGs) allow more suitable modeling for these applications. The existing techniques for DAGs cannot be applied directly on SDFGs due to the cyclic actor dependencies and the overlapping of multiple iterations (pipelined) in the schedule. Additionally, none of the existing techniques consider multi-application use-case (defined as a collection of multiple applications that are active simultaneously on a multiprocessor system [12]), which is a common requirement for multiprocessor systems. As we show in this paper, lifetime-aware dis-

A. Das is with the School of Electronics and Computer Science, University of Southampton, United Kingdom, E-mail: a.k.das@soton.ac.uk.
A. Kumar and B. Veeravalli are with the Department of Electrical and Computer Engineering, National University of Singapore, Singapore E-mail: {akash,elebv}@nus.edu.sg.
Manuscript received Month Day, Year.

tribution of the cores among the concurrent applications leads to a significant improvement in MTTF.

To address the limitations of the existing approaches, a temperature model is first proposed that is based on thermal characterization of a multiprocessor system using the *HotSpot* tool or using other thermal measurement approaches such as using sensors and performance counters. The model incorporates the following:

1. *temporal dependency* i.e., the relationship between the temperature of a core as a function of time and its dependency on the operating voltage and frequency; and
2. *spatial dependency* i.e., the influence of the neighboring core's temperature on the temperature of a core.

A gradient-based fast heuristic is then proposed incorporating the temperature model to jointly optimize energy and lifetime reliability of a multiprocessor system with applications represented as SDFGs. The approach leverage on the native SDF^3 tool [13], and can be easily ported to applications represented as DAGs, making the approach generic for both streaming multimedia and non-multimedia applications. Following are the key contributions of this paper:

1. a simplified temperature-model considering temporal and spatial dependencies;
2. a gradient-based fast heuristic to jointly optimize lifetime reliability and energy;
3. computing the MTTF from SDFG schedule considering task remapping;
4. maximizing the MTTF of multiprocessor system considering use-cases.

Contributions 1 and 2 were proposed in our earlier work [10]. This paper extends our earlier work by providing a detailed evaluation of the proposed temperature model and demonstration of the temperature computation from an SDFG schedule, in addition to the contributions 3 and 4 above.

The remainder of this paper is organized as follows. Section 2 provides a brief overview of the state-of-the-art approaches on temperature prediction for reliability optimization. This is followed by the problem formulation in Section 3 and the proposed temperature model in Section 4. The temperature computation from a given SDFG schedule is demonstrated in Section 5. The design methodology is discussed next in Sections 6. Experimental results are presented in Section 7 and Section 8 presents the conclusions.

2 Related Works

System-level reliability management approaches can be classified into two categories – run-time- and design-time-based. The run-time reliability management techniques improve lifetime reliability by adapting the hardware power levers and operating system scheduling decisions based on the continuous feedback from a hardware performance monitoring unit [14], [15]. Design-time-based approaches use application abstraction models, such as DAGs or SDFGs, to minimize the average and peak temperature in order to maximize the MTTF. This work falls in the later category and is orthogonal to any run-time reliability management techniques.

A number of techniques have been proposed in literature to minimize the peak and average temperature of multiprocessor systems. Examples include the convex optimization-based approach of [16], mixed integer linear programming (MILP)-based approach of [17], and pre-calibration based approach of [18]. These techniques are based on the steady-state temperature only. Very few thermal management techniques exist in literature that account for both the transient and the steady-state phases. A temperature-aware technique is proposed in [19] to distribute the idle time in order to control the power consumption and hence the temperature. However, this technique considers a uni-processor system; therefore it fails to model the spatial temperature dependency when applied to multiprocessor systems. Finally, both the temporal (transient and steady-state) and the spatial dependency are modeled in [20], [21] using thermal characterization. However, the cyclic dependency between power and temperature is not considered.

TABLE 1
Related works on reliability joint optimization.

Related Works	Power Consideration	Application Model	Use-cases	Reliability Model	Temperature Model
Gu et al. [2], Meyer et al. [4]	dynamic	DAGs	×	task remapping	steady-state
Huang et al. [3], Das et al. [5]	dynamic	DAGs	×	first failure	steady-state
Chou et al. [22], Huang et al. [8]	dynamic	DAGs	×	first failure	steady-state
Ukhov et al. [9]	dynamic & leakage	DAGs	×	first failure	transient, steady-state & spatial
proposed	dynamic & leakage	SDFGs	√	task remapping	transient, steady-state & spatial

Different wear-out mechanisms are influenced by temperature differently. Hence, there are studies that optimize lifetime reliability, while explicitly considering these wear-out mechanisms, through intelligent task mapping. A reliability estimation technique is proposed in [2] for application specific multiprocessor systems to consider multiple failures by incorporating the effect of faults on the subsequent fault rates. A slack allocation technique is proposed in [4] to optimize lifetime reliability. Both these techniques incorporate the *HotSpot* tool in the design space exploration (DSE) framework to determine the steady-state temperature. However, the high simulation time of the *HotSpot* tool leads to a super-exponential increase in the exploration time with an increase in the problem size. To overcome this scalability issue, the temperature characterization step is detached from the DSE framework in [3]. Specifically, the steady-state temperature values are determined using the *HotSpot* tool for all combinations of the active tasks on different processors. These temperature data are stored in a lookup table, and used with a simulated annealing-based heuristic to determine the task mapping that maximizes the lifetime reliability of multiprocessor systems. A convex optimization approach is proposed as an alternative to simulated annealing in [5]. Although these approaches determine the impact of mapping and scheduling on the lifetime reliability, the energy savings and the lifetime improvement obtained jointly, using DVFS, is not explored.

A resource management technique is proposed in [22] to minimize processor wear-out and communication energy, simultaneously providing tolerance for transient and intermittent faults. A simulated annealing-based technique is proposed in [8] to jointly optimize energy consumption and temperature-related processor aging. Both these approaches are based on the steady-state temperature characterization, similar to the approach

proposed in [3]. An eigen-value decomposition-based approach is proposed in [9] to optimize the energy consumption, simultaneously with thermal cycling related wear-outs. Although the temperature model incorporates the transient, steady-state and spatial dependency with consideration of dynamic and leakage power, the model is too complex to be incorporated in the DSE of SDFGs, enabled individually and concurrently. Table 1 summarizes the related works.

3 Problem Formulation

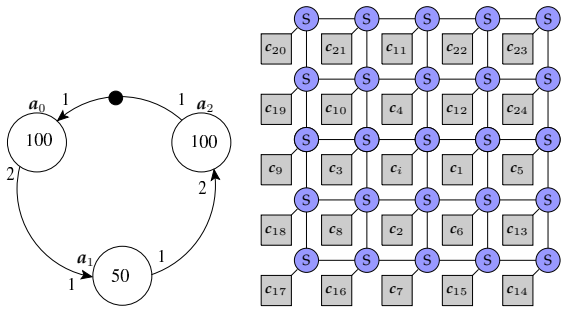
3.1 Application Model

Synchronous Data Flow Graphs (SDFGs, see [23]) are often used for modeling modern DSP applications and for designing concurrent multimedia applications implemented on a multiprocessor systems. Both pipelined streaming and cyclic dependencies between tasks can be easily modeled in SDFGs. SDFGs allow analysis of a system in terms of throughput and other performance properties, e.g. latency, buffer requirements [24].

The nodes of an SDFG are called $actors$; they represent functions that are computed by reading $tokens$ (data items) from their input ports and writing the results of the computation as tokens on the output ports. The edges in the graph, called $channels$, represent dependencies among different actors. Figure 1a shows an example of an SDFG. There are three actors in this graph. In the example, $\boldsymbol{a}_0$ has an input rate of 1 and output rate of 2. An actor is called *ready* when it has sufficient input tokens on all its input edges and sufficient buffer space on all its output channels; an actor can only fire when it is ready. The edges may also contain $initial\ tokens$, indicated by bullets on the edges, as seen on the edge from actor $\boldsymbol{a}_2$ to $\boldsymbol{a}_0$ in Figure 1a.

Formally an SDFG is mathematically represented as $\mathcal{G}_{app} = (\mathbb{A}, \mathcal{C})$ consisting of a finite set $\mathbb{A}$ of actors and a finite set $\mathcal{C}$ of channels. Every actor $\boldsymbol{a}_i \in \mathbb{A}$ s a tuple (t_i, μ_i), where t_i is the execution time of $\boldsymbol{a}_i$ and μ_i is its state space (program and data memory). The number of actors in an SDFG is denoted by N_a where $N_a = |\mathbb{A}|$. One of the most interesting properties of SDFGs relevant to this paper is throughput. Throughput is defined as the inverse of the long term period, i.e. the average time needed for one iteration of the application. An iteration is defined as the minimum non-zero execution such that the original state of the graph is obtained. This is the performance parameter used in this paper.

Several scheduling strategies have been proposed in literature for SDFG [25]. These techniques can be classified into three categories – dynamic scheduling, static scheduling and self-timed scheduling. In dynamic scheduling the actor assignment (mapping) and firing (timing) are both performed at run-time. Although, this scheduling technique gives full flexibility to accommodate dynamic actor execution time, throughput is not guaranteed. In static scheduling, both the mapping and scheduling are performed at design-time. Although, throughput is guaranteed in this technique, this scheduling technique fails to accommodate dynamic actor execution time. Self-timed strategy is widely used for scheduling SDFGs on multiprocessor systems. In this technique, the exact firing of an actor on a core is determined at design-time using worst-case actor execution-time. The timing information is then discarded retaining the assignment and ordering of the actors on each core. At run-time, actors are fired in the same order as determined from design-time. This scheduling technique can



(a) Application SDFG. (b) Multiprocessor floorplan.

Fig. 1. Application and architecture model.

accommodate dynamic actor execution time; as long as actors finish execution earlier than worst-case execution time (WCET), the throughput can only increase, i.e., the throughput analyzed using WCET is always guaranteed, which is important for multimedia and other DSP applications. The following lemma can be stated [24].

Lemma 1: For a consistent and strongly connected SDFG, the self-timed execution consists of a transient phase followed by a periodic (steady-state) phase.

Usually, the number of steady state iterations is a large number (e.g., periodic decoding of video frames), and hence for all practical purposes the energy and reliability of the steady state phase dominates over that in the transient phase. Therefore, the reliability-energy joint optimization is performed using the reliability and energy values per iteration of the steady-state phase.

3.2 Architecture Model

The multiprocessor architecture for this work is shown in Figure 1b with cores (indicated as labeled boxes) interconnected through switches in a mesh-based topology (the label on the cores is explained in Section 4). The architecture is represented as a graph $\mathcal{G}_{arc} = (\mathbb{C}, \mathbb{E})$, where $\mathbb{C}$ is the set of nodes representing cores of the architecture and $\mathbb{E}$ is the set of edges representing communication channels among the cores. The number of cores in the architecture is denoted by N_c i.e. $N_c = |\mathbb{C}|$. Each core $\boldsymbol{c}_j \in \mathbb{C}$ supports N_f voltage-frequency pairs denoted by $\{(V_k, \omega_k) \quad \forall k \in [0, N_f - 1]\}$.

3.3 Mapping Representation

The objective of the optimization problem is to maximize the lifetime reliability (measured as MTTF) and minimize the energy consumption by solving the following:

- *actor distribution:* determine the assignment of the actors of the SDFG on the cores of the platform;
- *operating point:* determine the voltage and frequency of the cores for executing the actors of the SDFG.

For the ease of problem representation, two variables $x_{i,j}$ (representing the *actor distribution*) and $y_{i,k}$ (representing the *operating point*) are defined as follows.

$$x_{i,j} = \begin{cases} 1 & \text{if actor } \boldsymbol{a}_i \text{ is executed on core } \boldsymbol{c}_j \\ 0 & \text{otherwise} \end{cases}$$

$$y_{i,k} = \begin{cases} 1 & \text{if actor } \boldsymbol{a}_i \text{ is executed at operating point } (V_k, \omega_k) \\ 0 & \text{otherwise} \end{cases}$$

Constraints on these variables are set such that an actor is mapped to only one core at a single operating point.

$$\sum_{j=0}^{N_c-1} x_{i,j} = 1 \text{ and } \sum_{k=0}^{N_f-1} y_{i,k} = 1 \quad \forall \boldsymbol{a}_i \in \mathbb{A} \tag{1}$$

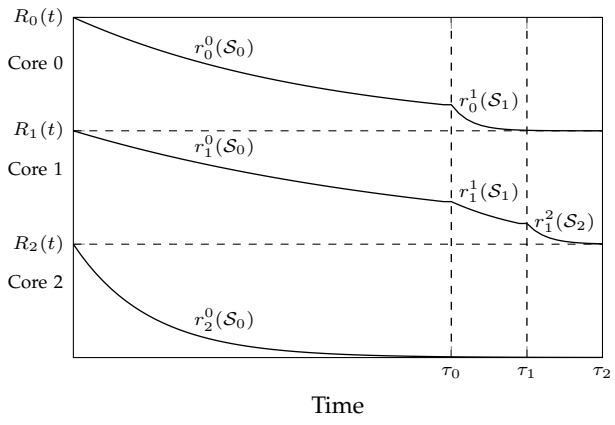


Fig. 2. MTTF computation with task remapping.

The *actor distribution* and *operating point* of SDFG are represented as two matrices:

$$\mathcal{M}_d = \begin{pmatrix} x_{0,0} & x_{0,1} & \cdots & x_{0,N_c-1} \\ x_{1,0} & x_{1,1} & \cdots & x_{1,N_c-1} \\ \vdots & \vdots & \ddots & \vdots \\ x_{N_a-1,0} & x_{N_a-1,1} & \cdots & x_{N_a-1,N_c-1} \end{pmatrix} \quad (2)$$

$$\mathcal{M}_o = \begin{pmatrix} y_{0,0} & y_{0,1} & \cdots & y_{0,N_f-1} \\ y_{1,0} & y_{1,1} & \cdots & y_{1,N_f-1} \\ \vdots & \vdots & \ddots & \vdots \\ y_{N_a-1,0} & y_{N_a-1,1} & \cdots & y_{N_a-1,N_f-1} \end{pmatrix} \quad (3)$$

The core to which actor $\boldsymbol{a}_i$ is assigned is denoted by ϕ_i and is given by $\phi_i = \mathbf{X}_i \times \mathbb{N}_{N_c}$ where $\mathbf{X}_i = (x_{i,0} \quad x_{i,1} \quad \cdots \quad x_{i,N_c-1})$ and $\mathbb{N}_{N_c}$ is the matrix of integers from 0 to N_c i.e. $\mathbb{N}_{N_c} = (0 \quad 1 \quad \cdots \quad N_c-1)^T$. The operating point of actor $\boldsymbol{a}_i$ is denoted by θ_i and is given by $\theta_i = \mathbf{Y}_i \times \mathbb{N}_{N_f}$ where $\mathbf{Y}_i = (y_{i,0} \quad y_{i,1} \quad \cdots \quad y_{i,N_f-1})$.

3.4 Wear-out Modeling

The lifetime reliability of core $\boldsymbol{c}_j$ at time t is defined as the probability of correct operation of the core upto time t and is given by $R_j(t) = e^{-(t.A_j)^\lambda}$ where A_j is the aging of the core per iteration and is given by (refer to [1]).

$$A_j = \frac{1}{t_p} \sum_l \frac{\Delta t_l}{\alpha(T^l)} \quad (4)$$

where t_p is the period of the application graph, $\alpha(T^l)$ is the fault density (typically Weibull or Lognormal distribution), λ is the slope parameter of the distribution and T^l is the average temperature in the interval Δt_l. This equation allows to model any wear-out effect such as electro-migration and negative bias temperature instability. Refer to [2] for a detailed modeling of the reliability considering different wear-out effects. The mean time to failure (MTTF) of core $\boldsymbol{c}_j$ with reliability $R_j(t)$ is given by the area under the reliability curve as

$$MTTF_j = \int_0^\infty R_j(t)dt = \int_0^\infty e^{-(t.A_j)^\lambda} dt \quad (5)$$

To demonstrate the MTTF computation considering task re-mapping, an example is provided with three cores. The initial schedule $\mathcal{S}_0$ uses all the three cores and stresses core 2 more than the other two cores. The reliability curves for the three cores are shown in Figure 2. Core 2 has the least lifetime and it breaks at time τ_0. As established in Section 2, the previous works on lifetime reliability defines MTTF to be the time to the first failure, hence the MTTF for these works is τ_0. At time $t = \tau_0$, a second schedule $\mathcal{S}_1$ (using core 0 and 1) is applied. The change in the reliability profile of core 0 and core 1 are due to different wear-out, which can be attributed to the difference in temperature from this new schedule. The new schedule stresses core 0 more than core 1 and therefore core 0 breaks at time $t = \tau_1$. At this time, all the actors are remapped to core 1. This schedule is identified in the figure as $\mathcal{S}_2$ and results in reliability profile of r_1^2. With this new reliability profile, core 1 breaks at time $t = \tau_2$. The lifetime (MTTF) of the system is therefore τ_2.

Besides MTTF, another interesting metric for multiprocessor system supporting task remapping is *processor years*, defined as the aggregate utilization of the different cores of the system over the entire lifetime. For the above example, this is calculated as follows. For the interval 0 to τ_0, all three cores are active; for the interval τ_0 to τ_1 two core are active; and for the interval τ_1 to τ_2 only one core is active. Assuming the time in this figure are all in years, the *processor years* of the above system is $3 \cdot \tau_0 + 2 \cdot (\tau_1 - \tau_0) + 1 \cdot (\tau_2 - \tau_1)$.

3.5 Energy Modeling

Actor-level voltage and frequency scaling is assumed for this work i.e, every actor of an SDFG is associated with a voltage-frequency value that is set on the core executing the actor. The energy consumed on a multiprocessor system while executing the SDFG consists of the following components:

- *computation energy:* dynamic and leakage energy consumed on the cores due to the actors execution;
- *communication energy:* dynamic and leakage energy consumed on the network-on-chip (NoC) due to the data communication among the connected actors.

A point to note here is that the leakage energy consumed on the NoC is dependent on the NoC type and the topology. For this work, a spatial division multiplexing-based NoC is assumed, and therefore the leakage power consumed on the NoC is negligible [26].

Dynamic Energy of SDFG: The dynamic energy of an SDFG is given by $E^{dyn} = E^{dyn}_{tr} + N_{iter} \cdot E^{dyn}_{ss}$ where E^{dyn}_{tr} is the actor dynamic energy in the transient phase of the schedule, E^{dyn}_{ss} is the actor dynamic energy per iteration of the steady state phase and N_{iter} is the number of iterations of the steady state phase. The dynamic energy consumed by an actor $\boldsymbol{a}_i$ executed on core $\boldsymbol{c}_j$ at the *operating point* k is given by

$$e^{dyn}(i,j,k) = C_{eff} \cdot \beta \cdot V_k^2 \cdot \omega_k \cdot t_{ijk} \cdot Rpt[\boldsymbol{a}_i] \quad (6)$$

where β is the activity factor, C_{eff} is the effective load capacitance, t_{ijk} is the execution time of actor $\boldsymbol{a}_i$ on core $\boldsymbol{c}_j$ at *operating point* k (i.e. operating voltage V_k and operating frequency ω_k) and $Rpt[\boldsymbol{a}_i]$ is the number of firings of actor $\boldsymbol{a}_i$ per steady state iteration of the SDFG. The total dynamic energy of the SDFG is

$$E^{dyn}_{core} = \sum_{\forall \boldsymbol{a}_i \in \mathbb{A}} e^{dyn}(i, \phi_i, \theta_i) \quad (7)$$

Leakage Energy of SDFG: The leakage energy of core $\boldsymbol{c}_j$ consumed during the execution of actor $\boldsymbol{a}_i$ at *operating point* k is given by the following formula [27].

$$e^{leak}(i,j,k) = N_{gates} V_k I_0 \left[AT^2 e^{\frac{\alpha V_k + \beta}{T}} + Be^{\gamma V_k + \delta} \right] \cdot t_{ijk} \cdot Rpt[\boldsymbol{a}_i] \quad (8)$$

where N_{gates} is the number of gates of the core, I_0 is the average leakage current and $A, B, \alpha, \beta, \gamma, \delta$ are

technology dependent constants (refer to [27]) and T is the average temperature of the actor during the steady-state iteration. The total leakage energy is

$$E_{core}^{leak} = \sum_{\forall \boldsymbol{a}_i \in \mathbb{A}} e^{leak}(i, \phi_i, \theta_i) \tag{9}$$

Dynamic Energy on the NoC: In [28], bit energy (E_{bit}) is defined as the energy consumed in transmitting one bit of data through the routers and links of a NoC. This is given by [29]

$$E_{bit}(p,q) = \begin{cases} (n_{hops}(p,q)+1) \cdot E_{S_{bit}} + n_{hops}(p,q) \cdot E_{L_{bit}} & \text{if } p \neq q \\ 0 & \text{otherwise} \end{cases} \tag{10}$$

where $E_{S_{bit}}$ and $E_{L_{bit}}$ are the energy consumed in the switch and the link, respectively and $n_{hops}(p,q)$ is the number of hops between cores $\boldsymbol{c}_p$ and $\boldsymbol{c}_q$.

The dynamic energy consumed on the NoC is therefore given by Equation 11 where ϕ_i and $\phi_{i'}$ are the cores where actors $\boldsymbol{a}_i$ and $\boldsymbol{a}_{i'}$ are mapped, respectively.

$$E_{noc}^{dyn} = \sum_{\forall \boldsymbol{a}_i, \boldsymbol{a}_{i'} \in \mathbb{A}} d_{ij} \cdot E_{bit}(\phi_i, \phi_{i'}) \tag{11}$$

The total energy is

$$E^{tot} = E_{core}^{dyn} + E_{core}^{leak} + E_{noc}^{dyn} \tag{12}$$

3.6 Reliability-Energy Joint Metric

To jointly optimize reliability and energy, a single metric **lifetime quotient** (lq) is introduced, which as defined as the ratio of the MTTF to the total energy i.e.,

$$lq = \frac{MTTF}{E^{tot}} \tag{13}$$

The optimization objective is to maximize lq.

4 Proposed Temperature Model

Temperature of a multiprocessor system is usually determined using a RC equivalent thermal model. The temperature of a single core is related to its power dissipation according to the following equation [11].

$$C\frac{dT(t)}{dt} + G\left(T(t) - T_{amb}\right) = P(t,T) = P^{dyn}(V_k, \omega_k) + P^{leak}(V_k, T) \tag{14}$$

where C is the thermal capacitance, G is the thermal conductance, t is the time, T_{amb} is the ambient temperature, $T(t)$ is the instantaneous temperature and $P(t)$ is the instantaneous power that is composed of the dynamic and the static components. The dynamic power is dependent on the voltage and frequency of operation and the static power is dependent on the temperature. The solution to the above equation consists of transient and steady-state phases. In the transient phase, the temperature increases with time up to a point beyond which, the steady-state phase settles in and the temperature saturates to its steady-state value. The core's wear-out is dependent on its operating temperature, which needs to incorporate both the transient and the steady-state phases.

Referring back to Figure 1b, the temperature of any core, say core $\boldsymbol{c}_i$ of a system depends on

A.1 The time of execution of an actor on $\boldsymbol{c}_i$.
A.2 The voltage and frequency of $\boldsymbol{c}_i$.
A.3 The temperature of the cores surrounding $\boldsymbol{c}_i$.

Thus, A.1 and A.2 represents the temporal dependency and A.3 represents the spatial dependency. For such a system, the temperature, power, thermal capacitance and thermal conductance in Equation 14 are all vectors. The transient and steady-state values can be obtained by solving the above equation analytically. The solution to the differential equation is $\mathbf{T}(t) = e^{\kappa t}\mathbf{T}(0) + \kappa^{-1}\left(e^{\kappa t} - \mathbf{I}\right)\mathbf{C}^{-1}\mathbf{P}(t)$, where $\kappa = -\mathbf{C}^{-1}\mathbf{G}$, $\mathbf{T}(0)$ is the initial temperature and $\mathbf{I}$ is the identity matrix. The direct solution technique is usually slow and results in an exponential design space exploration time. Although the model of [9] incorporates all the three components, the execution time is still exponential when applied for temperature prediction using SDFG.

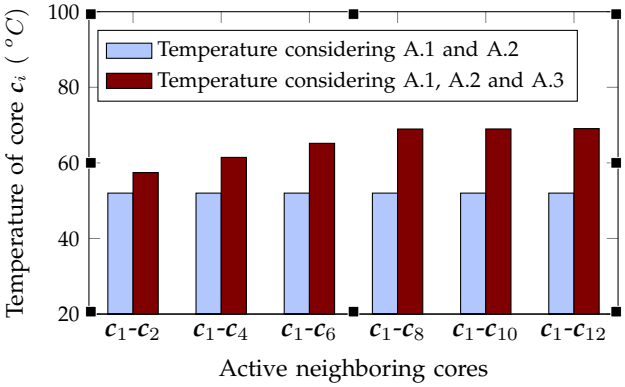


Fig. 3. Temperature underestimation ignoring the spatial dependency.

TABLE 2
Specification of the cores of the multiprocessor system

Power Mode	Frequency	Voltage ($\mathbf{V}$)	Current ($\mathbf{mA}$)	Power ($\mathbf{mW}$)
OPP50	300MHz	0.93	151.62	141.01
OPP100	600MHz	1.10	328.79	361.67
OPP130	800MHz	1.26	490.61	618.17
OPP1G	1GHz	1.35	649.64	877.01

A simplification to the analytical approach is to ignore the spatial dependency by considering the temperature for cores individually, such as the one proposed in [19]. To signify the importance of temperature underestimation by ignoring the spatial dependency (component A.3), an experiment is conducted using the *HotSpot* tool to measure the steady-state temperature. The multiprocessor architecture used for the *HotSpot* tool is shown in Figure 1b with the specifications of the cores reported in Table 2. The temperature is determined by setting the power dissipation of core $\boldsymbol{c}_i$ as 0, with a constant power corresponding to the operating point OPP1G (i.e.,1.35V, 1GHz) set for the one-hop and the two-hop neighboring cores (cores $\boldsymbol{c}_1$ - $\boldsymbol{c}_4$ are the one-hop neighbors and cores $\boldsymbol{c}_5$ - $\boldsymbol{c}_{12}$ are the two-hop neighbors of core $\boldsymbol{c}_i$ in Figure 1b). This simulates the scenario of core $\boldsymbol{c}_i$ as idle with the neighboring cores active at the highest voltage and frequency. The temperature results are reported in Figure 3 for some combinations of the neighboring core's activity. The label $\boldsymbol{c}_1 - \boldsymbol{c}_n$ in the figure indicates cores $\boldsymbol{c}_1, \boldsymbol{c}_2, \cdots, \boldsymbol{c}_n$ are active simultaneously. There are two bars shown on the plot. The left bar for each label corresponds to the temperature of core $\boldsymbol{c}_i$ obtained with all the core as idle. The right bar corresponds to the temperature of $\boldsymbol{c}_i$ with cores $\boldsymbol{c}_1, \boldsymbol{c}_2, \cdots, \boldsymbol{c}_n$ operating at the highest operating point and core $\boldsymbol{c}_i$ as idle. As seen from the figure, with only the east and the south neighbors active (i.e.,label $\boldsymbol{c}_1 - \boldsymbol{c}_2$), the temperature considering spatial dependency is 5°C higher than the temperature

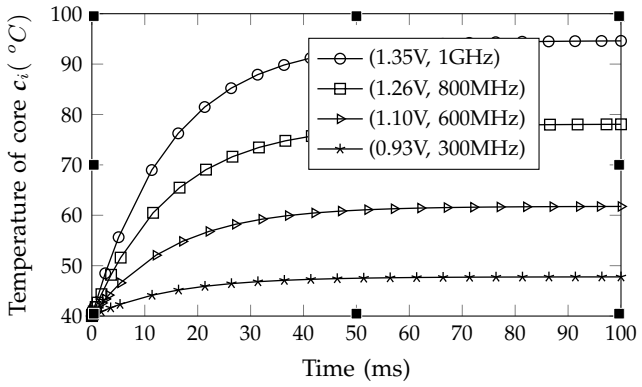


Fig. 4. Characterization for temporal dependency.

obtained by ignoring the spatial dependency (i.e., A.3). This difference increases as more number of neighbors become active. Finally, with all the one-hop and two-hop neighbors active, the temperature difference is as high as 18°C. This leads to significant MTTF misprediction.

To provide a simplified temperature model, a regression analysis technique is proposed in this paper. This temperature model is determined using:

- temperature characterization to incorporate the temporal dependency (capturing both the transient and steady-state behaviors); and
- temperature characterization to incorporate the spatial temperature dependency.

The temperature model is represented as

$$T_i(t) = f(V_i, \omega_i, t) + g(\{V_j, \omega_j \mid \forall \boldsymbol{c}_j \in \aleph(\boldsymbol{c}_i)\}) \quad (15)$$

where (V_i, ω_i) are the voltage and frequency of core $\boldsymbol{c}_i$, t is the time and $\aleph(\boldsymbol{c}_i)$ are the cores in the neighborhood of $\boldsymbol{c}_i$. The function f and g represent the temporal and spatial dependency, respectively and are derived in the following two steps.

- **Determine f:** The function f can be determined using two alternative approaches – by solving Equation 14 directly for a processing core; or by simulation using the *HotSpot* tool for the power consumption corresponding to different operating points of the processing core to capture the transient and the steady-state behaviors, as shown in Figure 4.
- **Characterize g:** The temperature data for characterizing the function g are obtained as follows. The core $\boldsymbol{c}_i$ is set to idle mode and the operating points for the neighboring cores are varied. Performing exhaustive temperature simulations for different voltage-frequency combinations of all neighbors (one-hop neighbors, two-hop neighbors, etc.) is time consuming, but only required once during the characterization step. A first order of approximation involves considering the voltage-frequency of only the immediate neighbors i.e., the east, west, north and south neighbors of a core referred to as (V_e, ω_e), (V_w, ω_w), (V_n, ω_n) and (V_s, ω_s), respectively with all other neighbors set to operate at the highest operating point. This is shown in Figure 5, where the voltage point is only shown for clarity of representation. The figure plots the temperature of core $\boldsymbol{c}_i$ as its voltage V_i is increased from 0.93V to 1.35V for few of these neighboring voltage combinations.

The temperature data are fed to the Matlab regression toolbox to derive the temperature model. The proposed

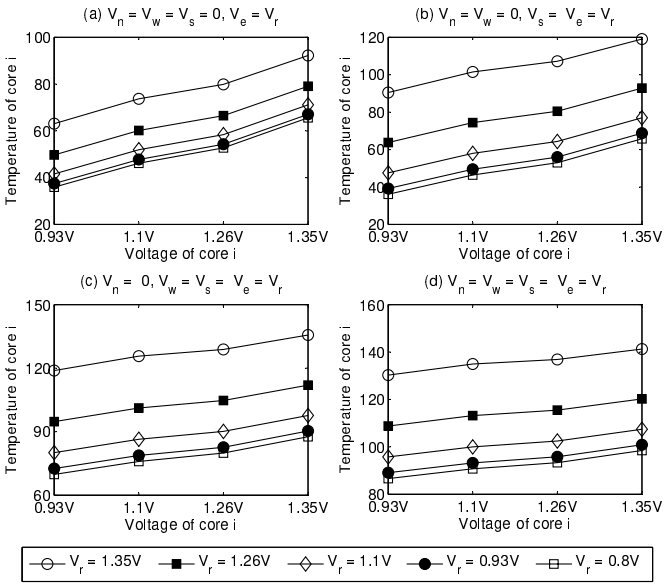


Fig. 5. Characterization for spatial dependency.

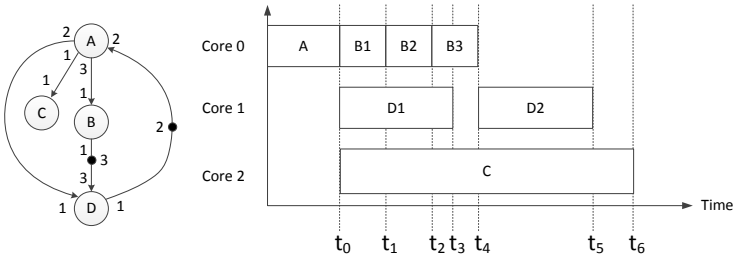


Fig. 6. Temperature computation from a SDFG schedule.

temperature model is determined once during the characterization process. The final expression for temperature (Equation 15) can be easily integrated in the design space exploration framework. However, the proposed model incorporates pessimism in three forms – separating the temporal and spatial dependency; characterizing the spatial dependency with the steady-state temperature of the nearest neighbors; and characterizing the spatial dependency with the non-nearest neighbors set to operate at the highest voltage and frequency. These pessimism lead to a temperature overestimation by as much as up to 6°C for individual applications. However, as discussed in Section 6.2, this temperature overestimation simplifies the reliability optimization for multi-application use-cases, which is a common requirement for multiprocessor systems.

5 Temperature Computation

Figure 6 shows an example of a SDFG with four actors allocated on a platform with three cores. The schedule corresponding to a particular allocation is also shown in the same figure. We demonstrate the temperature computation for core 0 using this schedule. The temperature for other cores can be determined in a similar fashion. The time duration $0 - t_6$ is divided into seven intervals by putting a time stamp at the instances when an actor starts or ends *firing*.

Core 0: Interval $(0 \rightarrow t_0)$

In this interval, core 0 executes actor A at operating point (V_A, ω_A). The temperature at time t considering temporal

effect is $f(V_A, \omega_A, t)$. The temperature considering the spatial effect is due to the idle voltages of core 1 and 2 and is given by $g(V_{idle}, V_{idle})$. The average temperature in this interval is

$$T_0(0, t_0) = \frac{1}{t_0} \int_0^{t_0} f(V_A, \omega_A, t)dt + g(V_{idle}, V_{idle}) \quad (16)$$

Core 0: Interval $(t_0 \rightarrow t_1)$
In this interval, core 0 executes the first instance of actor B. Note in the SDFG, when actor A fires, it produces 3 tokens on the channel from actor A to actor B and one of these tokens is consumed for each firing of actor B. Therefore, there are three firing of actor B (indicated in the figure by B1, B2 and B3). The temperature at time t due to the temporal effect of actor B is $f(V_B, \omega_B, t)$ and the temperature due to spatial effect is $g(V_D, V_C)$. The average temperature is

$$T_0(t_0, t_1) = \frac{1}{t_1 - t_0} \int_0^{t_1 - t_0} f(V_B, \omega_B, t)dt + g(V_D, V_C) \quad (17)$$

Core 0: Interval $(t_1 \rightarrow t_2)$
The temperature computation in this interval is similar to that in the interval $(t_0 \rightarrow t_1)$ and is given by

$$T_0(t_1, t_2) = \frac{1}{t_2 - t_1} \int_0^{t_2 - t_1} f(V_B, \omega_B, t)dt + g(V_D, V_C) \quad (18)$$

Core 0: Interval $(t_2 \rightarrow t_3)$
During the execution of actor B3, there is a change in temperature profile due to the completion of actor D1 and the interval before actor D2 is executed. Hence, the execution time of actor B3 is split into two intervals $(t_2 \rightarrow t_3)$ and $(t_3 \rightarrow t_4)$. The temperature computation in the interval $(t_2 \rightarrow t_3)$ is similar to that in the interval $(t_0 \rightarrow t_1)$

$$T_0(t_2, t_3) = \frac{1}{t_3 - t_2} \int_0^{t_3 - t_2} f(V_B, \omega_B, t)dt + g(V_D, V_C) \quad (19)$$

Core 0: Interval $(t_3 \rightarrow t_4)$
The average temperature in this interval is given by

$$T_0(t_3, t_4) = \frac{1}{t_4 - t_3} \int_0^{t_4 - t_3} f(V_B, \omega_B, t)dt + g(V_{idle}, V_C) \quad (20)$$

Core 0: Interval $(t_4 \rightarrow t_5)$
In this interval, the temporal effect is due to the idle temperature of the core and is denoted by T_0^{idle}. The average temperature is given by

$$T_0(t_4, t_5) = T_0^{idle} + g(V_D, V_C) \quad (21)$$

Core 0: Interval $(t_5 \rightarrow t_6)$
The temperature in this interval is given by

$$T_0(t_5, t_6) = T_0^{idle} + g(V_{idle}, V_C) \quad (22)$$

Reliability of Core 0
Combining these equations, the aging of core 0 is

$$r_0 = \frac{1}{t_6} \sum_{i=0}^{6} \frac{t_i - t_{i-1}}{\alpha\left(T_0(t_i, t_{i-1})\right)} \quad (23)$$

where $t_{-1} = 0$ and α is the fault density.

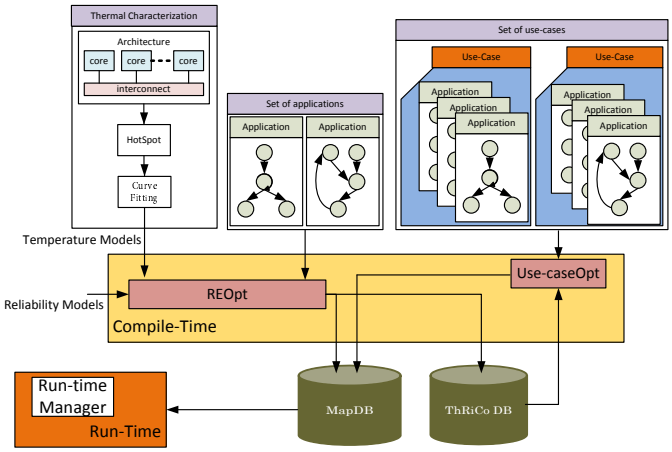


Fig. 7. Design methodology

Algorithm 1 Generate reliability and energy aware mappings.

Input: Application set $\mathbf{S}_{app}$ and multiprocessor system $\mathcal{G}_{arc}$
Output: $MapDB$ and $ThRiCoDB$
1: **for all** Application $\mathbf{A}_i \in \mathbf{S}_{app}$ **do**
2: $[\mathcal{G}_{app} \quad \mathbb{T}_c] = GetSDFG(\mathbf{A}_i)$
3: **for** $n = 1$ to N_c **do**
4: $(\mathcal{M}_d \quad \mathcal{M}_o) = REOpt(\mathcal{G}_{app}, \mathcal{G}^n_{arc}, \mathbb{T}_c)$
5: $[\mathcal{S} \quad \mathbb{T}] = MSDF^3(\mathcal{M}_d, \mathcal{M}_o, \mathcal{G}_{app}, \mathcal{G}^n_{arc})$
6: $MapDB(i, n) = (\mathcal{M}_d \quad \mathcal{M}_o \quad \mathcal{S})$
7: $M = CalculateMTTF(i, n, N_c^{min}, MapDB)$
8: $ThRiCoDB(i, n) = (\mathbb{T} \quad M)$
9: **end for**
10: **end for**

6 Design Methodology

The design methodology consists of two phases – analysis at design-time (consisting of the application and use-case optimizations) and execution at run-time. The design-time methodology is highlighted in Figure 7. The run-time manager is not part of the contribution, but is shown here for completeness. There are two databases for the multiprocessor system – the set of applications ($\mathbf{S}_{app}$) and the set of use-cases ($\mathbf{S}_{use}$). The proposed approach is to determine the actor distribution and the operating point (refer to Section 3) for every application using $n = 1$ to N_c cores of the system. Thus, $|\mathbf{S}_{app}| \cdot N_c$ optimization problems are solved at design-time to generate N_c mappings for every application supported on the system. This is performed in the *REOpt* block. The solution consists of the actor distribution and operating point matrices stored in the *MapDB* database and the three-dimensional (3D) vector – throughput, reliability (MTTF) and core count stored in the *ThRiCoDB* database.

It is to be noted that *MapDB* contains the mapping for every application with different core count. When an application is enabled individually at run-time, the entire set of cores is dedicated to the application. The optimum task-mapping for the application is fetched from the *MapDB* and applied. When one or more cores fail, the system restarts and the mapping with the reduced set of resources is fetched from *MapDB*. Thus, multiple core failures are addressed and for every fault-scenario, a mapping is applied to maximize the operational lifetime of the MPSoC. The migration to a new mapping with reduced resources is performed using the migration overhead minimization approach of [33].

Algorithm 1 provides the pseudo-code of the design flow. For every application $\mathbf{A}_i$ of the set $\mathbf{S}_{app}$, the corresponding SDFG representation and the throughput

Algorithm 2 $CalculateMTTF()$: Calculate the MTTF.

Input: Application id i, the core index n, the minimum number of cores for throughput satisfaction and mapping database $MapDB$
Output: MTTF M
1: Initialize $ttf = 0$ and $ri = n$
2: **while** $ri \geq N_c^{min}$ **do**
3: $[\mathcal{M}_d \ \ \mathcal{M}_o \ \ \mathcal{S}] = MapDB(i, ri)$
4: Determine reliability profiles from $\mathcal{S}$ as demonstrated in Section 5
5: Shift the reliability profiles by ttf
6: Determine t, the time to failure of the most stressed core
7: $ttf = ttf + t$ and $ri = ri - 1$
8: **end while**

constraint are fetched from the database. This application is executed on the multiprocessor system with n cores identified as $\mathcal{G}_{arc}^n$, where n is varied from N_c^{min} to N_c. The reliability-energy joint optimization is first performed on the application (line 4) to obtain the actor distribution matrix $\mathcal{M}_d$ and the operating point matrix $\mathcal{M}_o$. These are stored in the $MapDB$ (line 6). The actor distribution is used in the $MSDF^3$ tool that leverage on the native SDF^3 tool [13], which generates one feasible actor distribution and the corresponding throughput. The $MSDF^3$ tool is the modified form of SDF^3 that generates the schedule and throughput from a given actor distribution matrix. The schedule thus obtained is used in the $CalculateMTTF()$ routine (line 7) to compute the MTTF. The throughput and the MTTF values corresponding to the number of cores are stored in the $ThRiCoDB$ for the use-case optimization.

The $CalculateMTTF$ routine determines the MTTF in an iterative manner as shown as pseudo-code in Algorithm 2. A running index ri is maintained to index to the schedule with one less core. The algorithm iterates as long as $ri \geq N_c^{min}$, where N_c^{min} is the minimum number of cores required to satisfy the throughput requirement. At the start of the iteration, the mapping and the scheduling are fetched from the $MapDB$. The schedule is used to compute the reliability profile of every core of the system. The reliability profiles are shifted to account for the aging already encountered in the cores. The time-to-failure for all the cores are determined using Equation 5. The minimum time corresponds to the failure of the most stressed core. This is added to the ttf and the running index is decremented.

6.1 Reliability Optimization for Applications

The objective function (lifetime quotient) of the optimization problem is non-linear; a gradient-based fast heuristic is proposed to solve it. This is shown as pseudo-code in Algorithm 3. The algorithm starts from an initial allocation, computed using the native SDF^3 tool (line 2). Subsequently, the algorithm remaps every actor to every core to determine a priority function defined as

$$\mathcal{P} = \begin{cases} \frac{lq_n - lq}{\mathbb{T} - \mathbb{T}_n} & \text{if } \mathbb{T}_n < \mathbb{T} \\ (lq_n - lq) & \text{otherwise} \end{cases} \quad (24)$$

Two conditions are considered in the priority computation: if the throughput of the current allocation ($\mathbb{T}_n$) is lower than the original throughput ($\mathbb{T}$), a gradient function is used to calculate its priority i.e.,assignments that increase the lifetime quotient with the least throughput degradation are given higher priorities. Conversely, if the current throughput is higher than the original one, high priorities are given to assignments with the largest increase in the lifetime quotient.

The algorithm remaps actor $\boldsymbol{a}_i$ to a core $\boldsymbol{c}_j$ at operating point k (lines 6 - 8). The actor distribution and the operating point of actor $\boldsymbol{a}_i$ are changed (line 10). These

Algorithm 3 $REOpt()$: Reliability and energy optimization for an application.

Input: $\mathcal{G}_{app}$, $\mathcal{G}_{arc}$ and throughput constraint $\mathbb{T}_c$
Output: actor distribution and operating point matrices $(\mathcal{M}_d \quad \mathcal{M}_o)$, which maximize lq
1: Initialize $\mathcal{M}_o = (\mathbf{0} \quad \mathbf{0} \quad \cdots \quad \mathbf{1})$
2: $[\mathcal{M}_d \ \ \mathcal{S} \ \ \mathbb{T}] = SDF^3(\mathcal{G}_{app}, \mathcal{G}_{arc})$
3: **while** true **do**
4: $\mathcal{P}^{best} = 0$, $\mathcal{M}_d^{best} = \mathcal{M}_d$, $best\_found = false$
5: $lq = CalculateLQ(\mathcal{M}_d, \mathcal{M}_o, \mathcal{S}, \mathbb{T})$
6: **for all** $\boldsymbol{a}_i \in \mathbb{A}$ **do**
7: **for all** $\boldsymbol{c}_j \in \mathbb{C}$ **do**
8: **for all** $k \in [0, N_f - 1)$ **do**
9: $\mathcal{M}_d^{temp} = \mathcal{M}_d$ and $\mathcal{M}_o^{temp} = \mathcal{M}_o$
10: Update $\mathcal{M}_d^{temp}, \mathcal{M}_o^{temp}$ using $x_{i,j} = y_{i,k} = 1$ and $x_{i,l} = y_{i,m} = 0, \forall l \neq j$ and $\forall m \neq k$
11: $[\mathcal{S}_n \ \ \mathbb{T}_n] = MSDF^3(\mathcal{M}_d^{temp}, \mathcal{M}_o^{temp}, \mathcal{G}_{app}, \mathcal{G}_{arc})$
12: $lq_n = CalculateLQ(\mathcal{M}_d^{temp}, \mathcal{M}_o^{temp}, \mathcal{S}_n, \mathbb{T})$
13: Compute $\mathcal{P}$ using Equation 24
14: **if** $\mathbb{T}_n > \mathbb{T}_c$ and $\mathcal{P} > \mathcal{P}^{best}$ **then**
15: $\mathcal{P}^{best} = \mathcal{P}$, $\mathcal{M}_d^{best} = \mathcal{M}_d^{temp}$, $\mathcal{M}_o^{best} = \mathcal{M}_o^{temp}$, $best\_found = true$, $\mathbb{T} = \mathbb{T}_n$
16: **end if**
17: **end for**
18: **end for**
19: **end for**
20: **if** $best\_found$ **then**
21: $\mathcal{M}_d = \mathcal{M}_d^{best}$ and $\mathcal{M}_o = \mathcal{M}_o^{best}$
22: **else**
23: break
24: **end if**
25: **end while**
26: Return $(\mathcal{M}_d \quad \mathcal{M}_o)$

matrices are used by the $MSDF^3$ tool to compute the throughput and schedule corresponding to the allocation $\mathcal{M}_d^{temp}$ (line 11). The $CalculateLQ$ function computes the lifetime quotient using Equation 12 to compute the energy and Algorithm 2 to compute the MTTF. The algorithm computes the priority function (line 13). If this priority is greater than the best priority obtained thus far and the throughput constraint is satisfied, the best values are updated (line 15). The algorithm continues to remap as long as an assignment can be found without violating the throughput requirement. When no such remapping is possible, the algorithm terminates.

6.2 Reliability Optimization for Use-cases

In this section, the use-case level optimization problem is formulated based on the results obtained in Section 6.1. It is to be noted that when multiple applications are enabled simultaneously, the temperature due to the execution of one application is dependent not only on the temperature of the cores on which it is executed, but also on the temperature due to other applications executing simultaneously. As a result, the wear-out (or the MTTF) due to single application can be significantly different than the actual wear-out (or the MTTF) for use-cases. This limitation is addressed using the pessimism introduced in the temperature model. Specifically, to determine the temperature for different cores during single application mode, all unused cores in the architecture (those, which can potentially execute other applications in multi-application use-case scenario) are considered to be operating, and their temperature effect are incorporated in determining the temperature of the actual operating cores. Although this gives a pessimistic bound on the temperature (and hence, the reliability), the approach simplifies the problem solution for multi-application use-cases.

As indicated previously, the *ThRiCoDB* contains 3D databases with throughput and MTTF number for every core count of every application. The problem addressed

Algorithm 4 Core distribution for use-cases.

Input: $ThRiCoDB$
Output: Distribution of cores among applications
1: Initialize : $z_i = 0,\ 1 \leq i \leq n$
2: Initialize : $RiList.push(A_i, z_i, 0),\ 1 \leq i \leq n$
3: **for** $j = 1$ to N_c **do**
4: $\quad RiList.sort()$
5: $\quad$ Let, A_k = Task with least MTTF
6: $\quad z_k = z_k + 1$
7: $\quad M_k = ThRiCoDB.getMTTF(A_k, z_k)$
8: $\quad RiList.update(A_k, z_k, M_k)$
9: **end for**

here is to merge these 3D databases for applications enabled simultaneously such that the distribution of the cores among these applications maximizes the system MTTF. For the ease of problem formulation, the following notations are defined:

$$\begin{aligned} A_1, \cdots, A_n &= \text{n applications enabled simultaneously} \\ z_i &= \text{number of cores for application } A_i \\ M_i &= \text{MTTF of } A_i \text{ mapped on } z_i \text{ cores} \\ &= ThRiCoDB.getMTTF(z_i) \\ \mathbb{T}_i &= \text{Throughput of } A_i \text{ mapped on } z_i \text{ cores} \\ &= ThRiCoDB.getThr(z_i) \end{aligned}$$

6.2.1 Formulation

The optimization problem is

$$\text{maximize } \min_i \{M_i\} \text{ subject to } \sum_{i=1}^{n} z_i \leq N_c \tag{25}$$

6.2.2 Solution

Algorithm 4 provides the pseudo-code to solve Equation 25. A list is defined ($RiList$) to store the applications (their IDs) of the use-case, the number of cores dedicated to it, and the corresponding MTTF value. For every core in the system (line 3), the $RiList$ is sorted to determine the application with the least MTTF (lines 4 - 5). A core is dedicated to this application (line 6); the corresponding MTTF is fetched from the *ThRiCoDB* (line 7), and the $RiList$ is updated.

7 Experiments and Discussions

Experiments are conducted with real-life as well as synthetic SDFGs on a multiprocessor system with cores arranged in a mesh architecture. The synthetic SDFGs are generated using the SDF^3 tool [13] with the number of actors ranging from nine to twenty-five. These encompass both computation and communication dominated applications. The real-life SDFGs are *H.263 Encoder, H.263 Decoder, H.264 Encoder, MPEG4 Decoder, JPEG Decoder, MP3 Encoder and Sample Rate Converter*. Additionally, two non-streaming applications are also considered for this work. These are *FFT* and *Romberg Integration* from [30]. The supported voltage and frequency pairs are reported in Table 2, based on ARM Cortex-A8 core [31]. Although these voltage-frequency pairs are assumed for simplicity, the proposed algorithm and the temperature model can be used with any architecture with any supported voltage-frequency pairs.

The bit energy (E_{bit}) for modeling communication energy of an application is calculated using the formulas provided in [28] for packet-based NoC with Batcher-Banyan switch fabric using 65nm technology parameters from [32]. The parameters used for computing MTTF are the same as in [3] [8] [10]. The scale parameter of each core is normalized so that its MTTF under idle (non-stressed) condition is 10 years. All algorithms are coded in C++ and used with SDF^3 tool for throughput and schedule construction and $HotSpot$ for temperature characterization. Additionally, Matlab regression toolbox is used for modeling the spatial dependency.

TABLE 3
Execution time (s) of the $\mathbf{MSDF^3}$ tool

Actors	Multiprocessor platform			
	6 cores	9 cores	12 cores	16 cores
8	3.1	7.6	7.6	7.6
16	6.8	10.1	26.8	101.1
24	217.4	241.7	323.0	409.8
32	899.4	1021.4	2211.0	2789.9

7.1 Time Complexity

The time complexity of the algorithms are calculated as follows. There are N_c loops in Algorithm 1 for each application. In each loop, the algorithm executes the $REOpt()$, the $MSDF^3()$, and the iterative technique to compute the MTTF (i.e., Algorithm 2). The complexity of Algorithm 2 is calculated as follows. Assuming lines 3 - 7 can be computed in unit time, the worst case complexity of this algorithm is $C_2 = O(N_c)$ since $N_c^{min} \leq N_c$. The complexity of $REOpt()$ (Algorithm 3) is computed as follows. Let there be η iterations of the outer while loop (lines 3 - 25). In each iteration, the algorithm maps each actor to each core at each operating point to determine its reliability. The complexity of the this algorithm (C_3) is $C_3 = O\left(\eta \cdot N_a \cdot N_c \cdot N_f \cdot O(MSDF^3) \cdot C_2\right)$

The $MSDF^3$ engine computes the schedule starting from a given actor distribution. This can be performed in $O(N_a \log N_a + N_a \cdot \pounds)$ (ref. [33]), where $\pounds$ is the average number of successors of an actor. Therefore,

$$C_3 = O(\eta \cdot N_a \cdot N_c \cdot N_f \cdot (N_a \log N_a + N_a \cdot \pounds) \cdot N_c) = O\left(N_a^5 \cdot N_f\right) \tag{26}$$

where $N_c, \pounds \leq N_a$. The overall complexity of the reliability-energy joint optimization for each application is $C_1 = O(C_3 + O(MSDF^3) + C_2) = O(N_a^5 \cdot N_f)$. The execution time of the $MSDF^3$ tool is reported in Table 3.

Finally, the complexity of Algorithm 4 is calculated as follows. For every iteration of the outer loop (number of cores), sorting of MTTF is performed once followed by the memory lookup. If the memory lookup time is assumed to be constant and there are n applications enabled simultaneously on N_c cores, every loop is executed in $O(n \log n)$. The overall complexity of Algorithm 4 is therefore $O(N_c \times n \log n)$. On the multiprocessor platform considered, this algorithm takes between 80-100μsec for two to six simultaneous applications on an architecture with nine homogeneous cores.

7.2 Validation of the Temperature Model

The temperature model in Equation 15 incorporates only the voltage and frequency of the one-hop neighbors with all other cores operating at the highest operating point of (1.35V, 1GHz). To determine the pessimism in this approach, Figure 8 plots the temperature variation obtained using the simplified model of Equation 15, in comparison with the temperature obtained using the *HotSpot* tool by varying the operating points of the other neighbors. For this experiment, the execution time of the synthetic task is set to 300s to enable the proposed temperature model to reach its steady-state phase. The

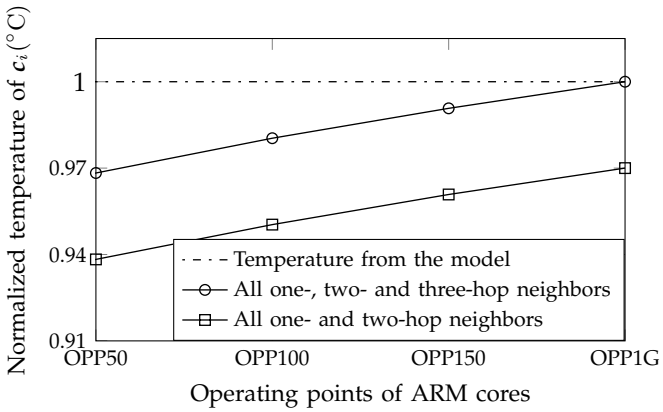


Fig. 8. Temperature variation of the proposed model.

TABLE 4
MTTF estimation accuracy. The table reports average temperature (°C) and MTTF (years) for 5 applications.

Applications	HotSpot	Model of [3], [5]	Proposed Model
FFT	60.5°C (8.0 yrs)	79.4°C (4.8 yrs)	68.0°C (6.7 yrs)
MPEG4	52.2°C (9.0 yrs)	67.2°C (6.7 yrs)	56.7°C (8.5 yrs)
JPEG	43.9°C (9.8 yrs)	55.4°C (8.2 yrs)	47.6°C (9.6 yrs)
MP3	58.0°C (8.1 yrs)	74.1°C (5.6 yrs)	62.7°C (7.5 yrs)
SRC	45.9°C (9.9 yrs)	61.7°C (7.2 yrs)	53.3°C (8.7 yrs)

temperature data obtained from the *HotSpot* tool are the steady-state values generated by varying the operating point of core c_i and all of its one- and two-hop neighbors in lock-step, with all other cores set as idle. In terms of the *HotSpot* specification, this setup translates to varying the power of c_i and its one- and two-hop neighbors with the values from Table 2, and setting the power dissipation as zero for all other cores. The temperature of core c_i obtained from the *HotSpot* tool (in °C) is normalized with respect to the temperature obtained from the model for the different operating points. Similarly, the results for one-, two-, and three-hop neighbors are obtained. As seen from the figure, with the one- and two-hop neighboring cores operating at OPP50, the temperature from the proposed model is an overestimate by 6.4% (9.5 °C in absolute terms). This overestimation decreases as the operating point is increased. This is expected, as more cores operate at the highest operating point, the temperature from the model is close to the temperature from the *HotSpot* tool. A similar trend is obtained for the one-, two-, and three-hop neighbors. For this plot, the temperature difference between the proposed model and the *HotSpot* tool is less than 0.1% at OPP1G.

7.3 MTTF Estimation Accuracy

To determine the accuracy of the proposed model in estimating the MTTF, an experiment is conducted with a set of real-life applications on the multiprocessor system. For each of these applications, one mapping and the corresponding schedule are determined using the unmodified SDF^3 tool. These are then fed to three temperature models – the *HotSpot* tool, the temperature model of [3], and the proposed temperature model. The average temperature for the schedule and the corresponding MTTF are reported in Table 4. As can be seen, the temperature using the proposed model is an overestimate by an average of 5.5°C for all these applications. This overestimation is due to the pessimism introduced in the model to simplify the solution. In terms of MTTF, this temperature overestimation leads to an MTTF underestimation by an average 0.8 years.

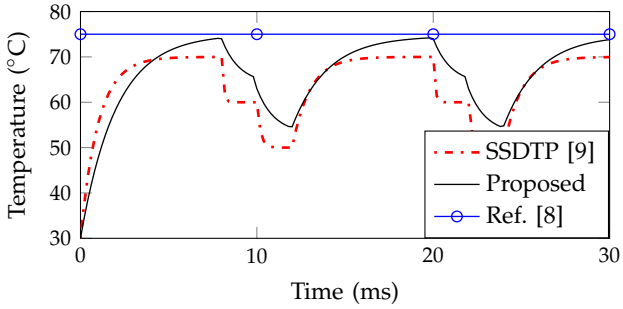


Fig. 9. Comparison with existing temperature models.

In comparison with the temperature model of [3] and [5], the proposed model improves the MTTF estimation efficiency by an average 21%. Although not explicitly shown here, despite of the *HotSpot* tool being accurate, the inclusion of the same in the design space exploration leads to an exponential execution time. An experiment with an SDFG composed of 8 tasks executed on a multiprocessor platform of 6 cores takes more than 12 hours when the *HotSpot* tool is integrated in the exploration framework. On the other hand, the same exploration with the proposed model takes an approximate 4.5 hours, clearly demonstrating its superior accuracy-execution time trade-off.

7.4 Comparison with Existing Temperature Models

Finally, the proposed temperature model is compared with the steady-state dynamic temperature profile (SSDTP) generated using the iterative technique of [9], and the steady-state temperature model of [8]. A synthetic SDFG is considered for this experiment with a throughput requirement of 80 iterations per second. This translates to a steady-state period of 12.5ms. This SDFG is executed on a multiprocessor system with 9 cores. The steady-state iteration of the SDFG corresponds to a period of 12ms. The power profile of the SDGF varies within iteration, and this variable power profile is repeated every iteration. With such a variable power profile repeated periodically, the steady-state temperature is not constant, but varies according to the periodic power pattern as shown in Figure 9, with the red dashed line showing the results obtained using the temperature model of [9]. For the same power profile, the results with the proposed model are shown in the same figure with black solid line. The mean temperature for this two temperature plots are 63.5°C and 66.1°C, respectively. The temperature model of [8] assumes a steady-state value for the duration of operation, which corresponds to the average power in this duration. This is shown with blue solid line in the figure and corresponds to a temperature of 75°C. (11.5°C difference from the average temperature of [9]). Thus, in comparison to the temperature model of [9], the proposed temperature model is more accurate than the model of [8].

A point to note here is that, although the proposed model results in an average temperature close to that obtained using the accurate model of [9], the thermal cycling is not captured accurately leading to a misprediction of the thermal cycling related MTTF. However, the

TABLE 5
Impact of Ignoring the Temperature Transient Phase.

Apps	MTTF using the model of [3]	MTTF using the model of [19]	MTTF using the *proposed model*
FFT	6.1	5.4	6.7
MPEG4	7.2	6.8	8.5
JPEG	8.6	9.4	9.6
MP3	6.4	6.1	7.5
SRC	7.9	8.7	8.7
synth16	6.8	6.0	6.8

advantage is its simple form (non-iterative as opposed to the iterative technique of [9]), which can be included in the design space exploration, especially for multi-application use-cases.

7.5 Impact of Temperature Misprediction

As mentioned in Section 4, some existing techniques ignore the transient phase of the temperature. This leads to an inaccuracy in the temperature prediction and a corresponding inaccuracy in the MTTF computation. Furthermore, ignoring the spatial dependency also leads to temperature misprediction. To demonstrate the importance of the transient phase and the spatial dependency of the temperature on the MTTF results, an experiment is conducted with six applications (five real-life and one synthetic) on the multiprocessor platform with nine cores. Table 5 reports three MTTF values (in years): the MTTF obtained using the proposed technique with the temperature model of [3] that considers steady-state temperature phase only; the MTTF obtained using the proposed technique with the temperature model of [19] that considers the temporal dependency only; and the MTTF obtained using the proposed technique with the proposed temperature model.

For the FFT application, the MTTF value estimated by ignoring the transient phase is 0.6 years lower than the MTTF value estimated using the proposed approach (column 2 vs column 4). For this application, the MTTF value estimated by ignoring the spatial dependency is 1.3 years lower than that estimated using the proposed approach (column 3 vs column 4). A similar trend is observed for MP3 Decoder and H.264 Encoder. These results highlight the fact that the MTTF estimation accuracy is lower when the spatial dependency is ignored as compared to ignoring the transient phase. This signifies the importance of spatial temperature component in the temperature estimation. Finally, as discussed in Section 1, considering the steady-state temperature is accurate only if the execution times of the actors of an application are comparable to the time constant of the RC equivalent circuit. This is shown for the synthetic application synth16 (with 16 actors) in the table. The execution times of the actors are generated with a mean of 200s and standard deviation of 20s. As can be seen, the MTTF obtained using the proposed model and the model of [3] are the same.

Thus far, the validation of the proposed temperature model is presented. In the next few subsections, we present the results to validate the proposed approach.

7.6 MTTF Considering Task Remapping

As indicated in Section 3.4, modern multiprocessor systems support remapping of tasks (actors in the SDFG terminology) on detection of faults. The MTTF for these systems need to be computed by considering task remapping, as opposed to the naive way of considering the time to the first fault. To determine the MTTF differences

TABLE 6
Processor years considering task remapping.

Apps	PY with TTFF	PY with task remapping				Throughput Performance			
		6 cores	5 cores	4 cores	Total	Constraint	6 cores	5 cores	4 cores
FFT	37.9	37.9	1.6	0.3	39.8	1	1.1	0.90	0.88
MPEG4	39.0	39.0	8.8	2.6	50.4	1	1.01	0.98	0.79
H.264	42.4	42.4	5.5	0.8	48.7	1	1.00	0.97	0.89
JPEG	46.9	46.9	8.2	2.4	57.5	1	1.00	1.00	1.00
MP3	42.6	42.6	1.6	0.3	44.5	1	1.00	0.99	0.97
SRC	45.2	45.2	6.0	0.9	52.1	1	1.01	0.92	0.77
Average					15.1%			0.96	0.88

in the two computation techniques, experiments are conducted on a multiprocessor system with six cores and a set of six real-life applications. This is shown in Figure 10a. There are two bars for every application. The left bar is for the MTTF considering the first failure and the right one for MTTF considering re-mapping. As seen from the figure, the two MTTF values are similar for FFT and MP3 decoder applications. However, for MPEG4 application (with MTTF constraint of 6.5 years), the two MTTFs differ by ≈25% (time to first failure is 6.8 years and MTTF considering remapping is 8.5 years). This difference implies that if task remapping is allowed for a multiprocessor system, the design space exploration can potentially search for another energy efficient mapping, trading the 1.7 years lifetime but still satisfying the MTTF requirement. On average for all applications considered, the MTTF improvement is 15%. To give more insight on the reason for such low MTTF difference for applications such as FFT, as opposed to say, JPEG decoder, Figure 10b plots the mean and the standard deviation of the aging of the different cores for the six applications. The standard deviation of the aging values is a measure of how much the aging of the individual cores differ from the mean value. A low standard deviation indicates a balanced situation with all the cores suffering similar wear-out. On the other hand, a high standard deviation indicates some cores age faster than others. The standard deviation is normalized with respect to the mean value of the aging.

As seen from the figure, for applications such as FFT and MP3 Encoder, the standard deviation of the aging parameters is close to zero and thus the wear-out experienced in the cores due to these applications are similar. For these applications, the MTTF considering the first failure is similar (less than 0.5% lower on average) to the MTTF considering remapping. This is intuitive, because with all cores suffering similar wear-outs, the break point (the time at which a core fails due to wear-out) for all the cores are similar and therefore remapping leads to an insignificant gain in lifetime. For all the other applications, the standard deviations are high, with some applications having standard deviation as high as 60% of the corresponding mean value. For these applications, the aging values are not balanced. Although a balanced aging leads to a higher overall MTTF, a further investigation into these applications reveal that the balanced aging mapping for these applications consumes high energy; therefore, the proposed gradient-based heuristic selects the mapping with non-balanced aging, but with significantly low energy consumption. For these applications, the MTTF considering remapping is higher by as much as 24% (average 10%) than the MTTF computation considering the time to the first failure (TTFF).

Table 6 reports the *processor years* considering the time to the first failure and the overall lifetime considering task remapping. For demonstration purpose, only two faults are allowed, and therefore the table reports up to

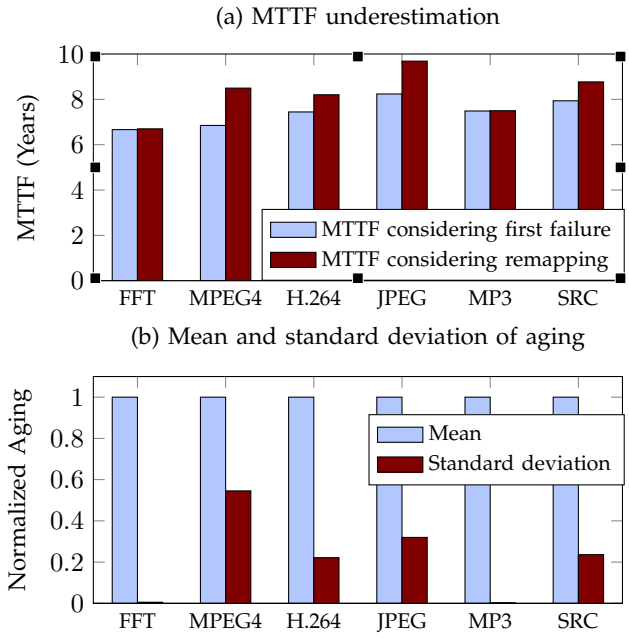


Fig. 10. MTTF considering task remapping.

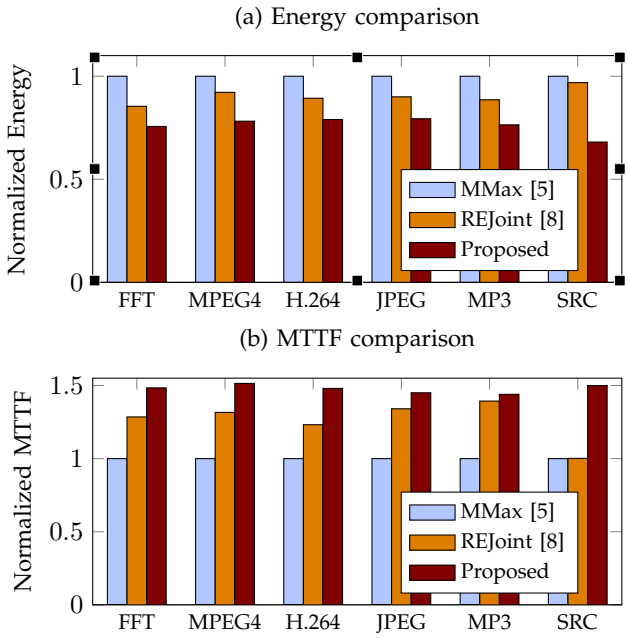


Fig. 11. Energy-reliability joint optimization results.

4 cores used. Column 2 reports the *processor years* considering the time to the first failure. This is the aggregate years spent with all the 6 cores active. The *processor years* with task remapping are shown in columns 3, 4 and 5, with the total in column 6. Specifically, column 3 reports the aggregate years spent with all the 6 cores active; column 4 reports the aggregate years with 5 cores active; and column 5 reports the aggregate years spent with 4 active cores. As seen from the table, the total *processor years* considering task remapping for MPEG4 is 30% higher than the *processor years* considering TTFF. This improvement is due to the non-zero *processor years* with 5 and 4 active cores. This improvement signifies that, even after the first fault, the multiprocessor system can be exploited to deliver 30% of the performance delivered during the time to the first fault. A similar trend is observed for the other applications in the table. On average, the *processor years* considering task remapping is 15% higher than the *processor years* considering the time to the first failure.

Finally, columns 7-10 of Table 6 report the normalized throughput obtained with different number of operating cores. Throughput numbers for an application are normalized with respect to the throughput constraint for the application. For some application such as JPEG decoding, there is no throughput degradation even with 4 cores, signifying that task remapping increases lifetime of a device with no throughput degradation. For other applications such as MPEG4 decoding, the throughput constraint is satisfied with 6 cores only. The throughput degradation increases with reduced number of cores, with a degradation of 21% with 4 cores. In terms of frames per second (fps) requirement, MPEG4 achieves 19fps (instead of required 24fps) with 4 cores, causing a video quality degradation. On average for all the applications considered, the average throughput degradation with 5 and 4 cores are respectively 4% and 12%. To conclude, a multiprocessor system with task remapping allows operation beyond first failure, however, at the expense of reduced throughput.

7.7 Reliability and Energy Improvement

Figure 11 plots the energy and reliability results of the proposed approach in comparison to the existing reliability-energy joint optimization technique of [8] for six real-life application. Additionally, to determine the reliability benefit of the dynamic voltage and frequency scaling, these two techniques are compared with the highest MTTF technique of [5] (referred to as MMax), which determines MTTF by solving a convex optimization problem. These results are represented as three bars corresponding to each application. A point to note here is that, all the application SDFGs are first converted to homogeneous SDFGs (HSDFGs) before applying the techniques of [5] and [8]. It is to be noted that the conversion of an SDFG to HSDFG is of exponential complexity and therefore the proposed technique is the first technique for reliability-energy-performance optimization for SDFGs.

The following trends can be followed from the figure. The energy consumption using the proposed approach and the existing energy-reliability joint optimization technique of [8] are lower than the highest MTTF technique of [5] that does not consider dynamic voltage and frequency scaling. The MTTF obtained using these techniques are also higher than the MTTF of [5]. These results signify that, by slowdown of the actor computation, the reliability can be improved significantly.

On average for all the applications considered, the existing optimization technique minimizes energy consumption by 10% with a corresponding reliability improvement of 26% as compared to the highest MTTF technique. A point to note here is that, this technique is based on sequential execution of applications; therefore, the throughput slack (difference between the actual throughput and the throughput constraint) is low, implying a limited scope for actor slowdown. The energy improvement in this technique is, therefore, not significant. The proposed technique achieves better results than this technique by minimizing energy consumption further by an average 15%, and increasing lifetime by an additional 18%. In comparison to [5], the proposed

TABLE 7
Design space exploration time.

Actors	Design Space Exploration Time (in minutes)					
	cores = 4			cores = 6		
	Convex [5]	SA [8]	*Proposed*	Convex [5]	SA [8]	*Proposed*
4	8	10	4	18	18	6
6	79	51	23	157	83	36
8	677	319	154	1013	524	274

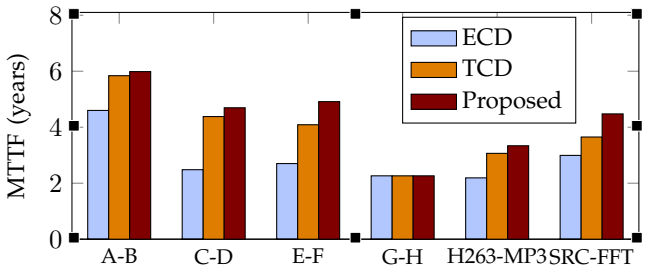


Fig. 12. MTTF improvements with synthetic use-cases.

technique minimizes energy consumption by 24% and increases lifetime by 47%.

7.8 Design Space Exploration Speed-up

To highlight the speedup achieved by using the proposed design space exploration heuristic to jointly optimize energy and reliability, Table 7 reports its execution time in comparison with the convex optimization based technique of [5] and the simulated annealing based technique of [8]. The execution time are recorded by running synthetic SDFGs with varying number of actors on two multiprocessor systems – with four and six cores, respectively. The number of actors is limited to 8 as the convex optimization fails to provide results beyond 8 actors, even for running more than 12 hours. The time reported in this table are the average results obtained by generating multiple SDFGs. For example, the execution time for 6 actors on 4 cores is the average time taken by the three techniques for 10 different synthetic SDFGs, with 6 actors each. For fair comparison, the time taken by the temperature pre-characterization step is omitted for all these techniques and the time reported are the time for the respective technique – convex solver for [5], simulated annealing for [8], and proposed heuristic of Algorithm 3. As seen from the table, for small number of actors the execution time of the convex solver is comparable to that of the simulated annealing (better for 4 actors on 4 cores). However, as the number of actors is increased, the simulated annealing outperforms the convex solver. In comparison to these two techniques, the proposed approach improves execution time significantly, achieving benefits for small and large problem sizes. On average, the execution time using the proposed technique is 70% and 50% lower with respect to [5] and [8], respectively.

7.9 Use-case Optimization Result

Since this work is the first work on use-case level MTTF optimization, there is no reference for comparison. However, two standard strategies are developed to distribute the cores among concurrent applications in a use-case – throughput-based core distribution (TCD) and equal core distribution (ECD). For implementing these approaches, the cores of the architecture are first distributed to the applications based on the corresponding strategy (equally or in the ratio of the throughput). The proposed optimization technique is then applied on individual applications to determine their MTTF. The overall MTTF of the use-case is the minimum of the MTTFs of the concurrent applications. The MTTFs obtained for a use-case using both these strategies, are compared with the MTTF obtained using the proposed MTTF-based core distribution technique. To demonstrate the advantage of the proposed approach for use-case optimization, a set of six synthetic use-cases are generated. Four of these uses-cases are composed of synthetic applications and the two others are composed of real-life applications. These use-cases are executed on a multiprocessor system with nine cores. Figure 12 plots the MTTF for the three approaches for these uses-cases. The composition of each use-case is indicated in the label of the figure, where the application with alphabets are the synthetic applications. For the use-case A-B, the MTTF obtained by distributing the cores equally is 4.6 years. The TCD achieves better results by distributing the cores in the ratio of their throughput requirements. The improvement in this technique is 27%. The proposed technique improves this further by achieving 3% higher lifetime. A similar trend is observed for all other use-cases. As seen from the figure, for some use-cases such as A-B and G-H, the improvements using the proposed technique are insignificant. For other use-cases such as E-F and SRC-FFT, the improvements are more than 20%. On average for all the six synthetic use-cases, the proposed technique improves MTTF by 10% as compared to TCD and 140% as compared to ECD.

8 Conclusions

In this work, a simplified temperature model is proposed, based on off-line temperature characterization using the *HotSpot* tool. Based on this model, a gradient-based fast heuristic is proposed to determine the voltage and frequency of cores such that the energy consumption is minimized, simultaneously maximizing the system mean time to failure (MTTF). Experiments are conducted on a multiprocessor system using a set of synthetic and real-life application SDFGs, executed individually as well as concurrently. Results demonstrate that the proposed approach minimizes energy consumption by an average 24% and maximizes lifetime by 47% as compared to the existing work. Additionally, the proposed MTTF-aware core distribution for concurrent applications results in an average 10% improvement in lifetime as compared to the performance-aware core distribution.

References

[1] J. Srinivasan, S. V. Adve, P. Bose, and J. A. Rivers, "The Case for Lifetime Reliability-Aware Microprocessors," in *Proceedings of the Annual International Symposium on Computer Architecture*, 2004.
[2] Z. Gu, C. Zhu, L. Shang, and R. Dick, "Application-Specific MPSoC Reliability Optimization," *IEEE Transactions on Very Large Scale Integration Systems*, 2008.
[3] L. Huang, F. Yuan, and Q. Xu, "On Task Allocation and Scheduling for Lifetime Extension of Platform-Based MPSoC Designs," *IEEE Transactions on Parallel and Distributed Systems*, 2011.
[4] B. H. Meyer, A. S. Hartman, and D. E. Thomas, "Cost-effective Lifetime and Yield Optimization for NoC-based MPSoCs," *ACM Transactions on Design Automation of Electronic Systems*, 2014.
[5] A. Das, A. Kumar, and B. Veeravalli, "Reliability-driven Task Mapping for Lifetime Extension of Networks-on-chip Based Multiprocessor Systems," in *Proceedings of the Conference on Design, Automation and Test in Europe*, 2013.
[6] A. K. Singh, A. Das, and A. Kumar, "Energy Optimization by Exploiting Execution Slacks in Streaming Applications on Multiprocessor Systems," in *Proceeding of the Annual Design Automation Conference*, 2013.

[7] M. Damavandpeyma, S. Stuijk, T. Basten, M. Geilen, and H. Corporaal, "Throughput-constrained DVFS for Scenario-Aware Dataflow Graphs," in *Proceedings of the IEEE Symposium on Real-Time and Embedded Technology and Applications*, 2013.
[8] L. Huang and Q. Xu, "Energy-efficient Task Allocation and Scheduling for Multi-mode MPSoCs Under Lifetime Reliability Constraint," in *Proceedings of the Conference on Design, Automation and Test in Europe*, 2010.
[9] I. Ukhov, M. Bao, P. Eles, and Z. Peng, "Steady-state Dynamic Temperature Analysis and Reliability Optimization for Embedded Multiprocessor Systems," in *Proceeding of the Annual Design Automation Conference*, 2012.
[10] A. Das, A. Kumar, and B. Veeravalli, "Temperature Aware Energy-Reliability Trade-offs for Mapping of Throughput-Constrained Applications on Multimedia MPSoCs," in *Proceedings of the Conference on Design, Automation and Test in Europe*, 2014.
[11] K. Skadron, M. R. Stan, K. Sankaranarayanan, W. Huang, S. Velusamy, and D. Tarjan, "Temperature-Aware Microarchitecture: Modeling and Implementation," *ACM Transactions on Architecture and Code Optimization*, 2004.
[12] A. Kumar, B. Mesman, H. Corporaal, and Y. Ha, "Iterative Probabilistic Performance Prediction for Multi-Application Multiprocessor Systems," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2010.
[13] S. Stuijk, M. Geilen, and T. Basten, "SDF3: SDF For Free," in *Proceedings of the International Conference on Application of Concurrency to System Design*, 2006.
[14] P. Mercati, A. Bartolini, F. Paterna, T. S. Rosing, and L. Benini, "Workload and User Experience-aware Dynamic Reliability Management in Multicore Processors," in *Proceeding of the Annual Design Automation Conference*, 2013.
[15] A. Das, R. A. Shafik, G. V. Merrett, B. M. Al-Hashimi, A. Kumar, and B. Veeravalli, "Reinforcement Learning-Based Inter- and Intra-Application Thermal Optimization for Lifetime Improvement of Multicore Systems," in *Proceeding of the Annual Design Automation Conference*, 2014.
[16] A. Mutapcic, S. Boyd, S. Murali, D. Atienza, G. De Micheli, and R. Gupta, "Processor Speed Control With Thermal Constraints," *IEEE Transactions on Circuits and Systems I: Regular Papers*, 2009.
[17] T. Chantem, X. Hu, and R. Dick, "Temperature-Aware Scheduling and Assignment for Hard Real-Time Applications on MPSoCs," *IEEE Transactions on Very Large Scale Integration Systems*, 2011.
[18] J. Cui and D. Maskell, "A Fast High-Level Event-Driven Thermal Estimator for Dynamic Thermal Aware Scheduling," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2012.
[19] M. Bao, A. Andrei, P. Eles, and Z. Peng, "Temperature-aware Idle Time Distribution for Energy Optimization with Dynamic Voltage Scaling," in *Proceedings of the Conference on Design, Automation and Test in Europe*, 2010.
[20] L. Shang, L.-S. Peh, A. Kumar, and N. K. Jha, "Thermal modeling, characterization and management of on-chip networks," in *Proceedings of the International Symposium on Microarchitecture*, 2004.
[21] D. Rai, H. Yang, I. Bacivarov, and L. Thiele, "Power Agnostic Technique for Efficient Temperature Estimation of Multicore Embedded Systems," in *Proceedings of the International Conference on Compilers, Architecturesand Synthesis for Embedded Systems*, 2012.
[22] C.-L. Chou and R. Marculescu, "FARM: Fault-aware Resource Management in NoC-based Multiprocessor Platforms," in *Proceedings of the Conference on Design, Automation and Test in Europe*. European Design and Automation Association, 2011.
[23] E. Lee and D. Messerschmitt, "Synchronous data flow," *Proceedings of the IEEE*, 1987.
[24] S. Stuijk, M. Geilen, and T. Basten, "Exploring Trade-offs in Buffer Requirements and Throughput Constraints for Synchronous Dataflow Graphs," in *Proceeding of the Annual Design Automation Conference*, 2006.
[25] S. Sriram and S. Bhattacharyya, *Embedded Multiprocessors; Scheduling and Synchronization*. Marcel Dekker, 2000.
[26] A. Leroy, D. Milojevic, D. Verkest, F. Robert, and F. Catthoor, "Concepts and Implementation of Spatial Division Multiplexing for Guaranteed Throughput in Networks-on-Chip," *IEEE Transactions on Computers*, 2008.
[27] 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*, 2005.
[28] T. T. Ye, L. Benini, and G. De Micheli, "Packetized On-Chip Interconnect Communication Analysis for MPSoC," in *Proceedings of the Conference on Design, Automation and Test in Europe*, 2003.
[29] J. Hu and R. Marculescu, "Energy-Aware Communication and Task Scheduling for Network-on-Chip Architectures Under Real-Time Constraints," in *Proceedings of the Conference on Design, Automation and Test in Europe*, 2004.
[30] D. Bertozzi, A. Jalabert, S. Murali, R. Tamhankar, S. Stergiou, L. Benini, and G. De Micheli, "NoC Synthesis Flow for Customized Domain Specific Multiprocessor Systems-on-Chip," *IEEE Transactions on Parallel and Distributed Systems*, 2005.
[31] G. Coley, "Beagleboard System Reference Manual," *BeagleBoard.org*, p. 81, 2009.
[32] W. Zhao and Y. Cao, "Predictive Technology Model for nano-CMOS Design Exploration," *ACM Journal on Emerging Technologies in Computing Systems*, 2007.
[33] A. Das, A. Kumar, and B. Veeravalli, "Energy-aware Task Mapping and Scheduling for Reliable Embedded Computing Systems," *ACM Transactions on Embedded Computing Systems*, 2014.

Anup Das Dr. Anup Das received the B.Eng. degree in Electronics and Telecommunication Engineering from Jadavpur University, India, in 2004. He received the Ph.D. degree in computer engineering in the area of embedded systems from the National University of Singapore, in 2014. He is currently a post-doctoral research fellow at the University of Southampton. From 2004 to 2007 he was with STMicroelectronics Ltd as an IC design engineer. From 2007 to 2011 he was with LSI Corporation (formerly Agere Systems) as design-for-test engineer of storage SoCs. His research interests include reliability and energy-aware system architecture, application mapping and scheduling on multiprocessor platforms and resource management for multimedia multiprocessor systems.

Akash Kumar Dr. Akash Kumar received the B.Eng. degree in computer engineering from the National University of Singapore (NUS), Singapore, in 2002. He received the joint Master of Technological Design degree in embedded systems from NUS and the Eindhoven University of Technology (Tue), Eindhoven, The Netherlands, in 2004, and received the joint Ph.D. degree in electrical engineering in the area of embedded systems from TUe and NUS, in 2009. His thesis is entitled Analysis, Design and Management of Multimedia Multiprocessor Systems. In 2004, he was with Philips Research Labs, Eindhoven, The Netherlands, where he worked on Reed Solomon codes as a Research Intern. From 2005 to 2009, he was with TUe as a researcher under project PreMaDoNA. Since 2009, he has been with the Department of Electrical and Computer Engineering, NUS. Currently, he is an Assistant Professor in the department. His research interests include analysis, architectures, design methodologies, and resource management of embedded multiprocessor systems. He has published over 30 papers in leading international electronic design automation journals and conferences on these topics.

Bharadwaj Veeravalli Dr. Bharadwaj Veeravalli received the BSc degree in physics from Madurai Kamaraj Uiversity, India, in 1987, the masters degree in electrical communication engineering from the Indian Institute of Science, Bangalore, India, in 1991 and the PhD degree from the Department of Aerospace Engineering, Indian Institute of Science, Bangalore, India, in 1994. He did his postdoctoral research in the Department of Computer Science, Concordia University, Montreal, Canada, in 1996. He is currently with the Department of Electrical and Computer Engineering at the National University of Singapore, as a tenured associate professor. His main stream research interests include multiprocessor systems, cluster/grid computing, scheduling in parallel and distributed systems, bioinformatics and computational biology, and multimedia computing. He is one of the earliest researchers in the field of divisible load theory (DLT). He is currently serving on the Editorial Board of the IEEE Transactions on Computers, the IEEE Transactions on SMC-A, and the International Journal of Computers and Applications, as an associate editor. He is a senior member of the IEEE and the IEEE Computer Society.