On Task Allocation and Scheduling for Lifetime Extension of Platform-Based MPSoC Designs

Lin Huang, *Student Member*, *IEEE*, Feng Yuan, *Student Member*, *IEEE*, and Qiang Xu, *Member*, *IEEE*

Abstract—With the relentless scaling of semiconductor technology, the lifetime reliability of today's multiprocessor system-on-a-chip (MPSoC) designs has become one of the major concerns for the industry. Without explicitly taking this issue into consideration during the task allocation and scheduling process, existing works may lead to imbalanced aging rates among processors, thus reducing the system's service life. To tackle this problem, in this paper, we propose an analytical model to estimate the lifetime reliability of multiprocessor platforms when executing periodical tasks, and we present a novel task allocation and scheduling algorithm that is able to take the aging effects of processors into account, based on the simulated annealing technique. In addition, to speed up the annealing process, several techniques are proposed to simplify the design space exploration process with satisfactory solution quality. Experimental results on various hypothetical multiprocessors and task graphs show that significant system lifetime extension can be achieved by using the proposed approach, especially for heterogeneous platforms with large task graphs.

Index Terms—Lifetime reliability, aging effect, multiprocessor system-on-a-chip, task allocation and scheduling.

1 Introduction

As technology advances, it is possible to integrate multiple microprocessors, dedicated hardware accelerators, and sometimes mixed-signal circuitries on a single silicon die, namely multiprocessor system-on-a-chip (MPSoC) [1]. One way to design MPSoC embedded systems is to use hardware/software cosynthesis [2]. While this method is able to explore more design space to obtain a flexible application-specific architecture, it generally takes more design time and has high design risk. Because of this, platform-based design methodology has become increasingly popular for complex embedded systems. In this approach, designers first pick a predesigned MPSoC platform, e.g., ARM11 PrimeXsys platform [3] or NXP Nexperia platform [4], and then map their applications onto this platform.

While the relentless scaling of CMOS technology has brought MPSoC designs with enhanced functionality and improved performance in every new generation, at the same time, the associated ever-increasing on-chip power and temperature densities make failure mechanisms serious threats for the lifetime reliability of such high-performance integrated circuits (ICs) [5], [6], [7]. If care is not taken during the task allocation and scheduling process, some processors might age much faster than the others and become the reliability bottleneck for the embedded system, thus significantly reducing the system's service life.

Although there are many existing works on reliability-driven task allocation and scheduling (e.g., [8], [9], [10], [11]), most of them assume an exponential distribution for failure mechanisms. In other words, processors' failure rates are assumed to be independent of their usage history, which is obviously inaccurate: a typical wearout failure mechanism will have increasing failure rate as the circuit ages [12], [13]. Recently, some thermal-aware task scheduling techniques have been proposed in the literature (e.g., [14]). As ICs' failure rates are strongly related to their operational temperature, these techniques may implicitly improve the MPSoC's lifetime reliability by balancing different processors' temperatures or keeping them under a safe threshold. However, since many other factors (e.g., internal structure, operational frequency, and supply voltage) also severely affect the circuits' failure rate [6], [15], without explicitly taking the lifetime reliability into account during the task allocation and scheduling process, processor cores may still age differently and hence result in shorter mean time to failure (MTTF) for MPSoC designs.

In this paper, we present novel solutions for the lifetime extension of platform-based MPSoC designs. The main contributions of our work are as follows:

- we propose a comprehensive lifetime reliability-aware task allocation and scheduling strategy that takes processors' aging effects into account, based on the simulated annealing (SA) technique;
- we present a novel analytical model to compute the lifetime reliability of platform-based MPSoCs when executing periodical tasks;
- we propose several speedup techniques to achieve an efficient MPSoC lifetime estimation with satisfactory solution quality.

The remainder of this paper is organized as follows: Section 2 reviews related prior work and motivates this paper. The proposed lifetime reliability-aware task allocation and scheduling strategy is presented in Section 3. We,

• *The authors are with the Department of Computer Science and Engineering, The Chinese University of Hong Kong, Shatin, N.T., Hong Kong. E-mail: {lhuang, fyuan, qxu}@cse.cuhk.edu.hk.*

Manuscript received 27 May 2009; revised 21 Aug. 2009; accepted 30 Nov. 2009; published online 12 Apr. 2011.
Recommended for acceptance by F. Petrini.
For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number TPDS-2009-05-0240.
Digital Object Identifier no. 10.1109/TPDS.2011.132.

then, introduce our analytical model for the lifetime reliability of platform-based MPSoC designs in Section 4. To meet the stringent time-to-market requirement, four speedup techniques for MPSoC lifetime approximation are presented in Section 5. Experimental results on several hypothetical platform-based MPSoC designs are presented in Section 6. Finally, Section 7 concludes this paper and points out some future research directions.

2 Prior Work and Motivation

2.1 IC Lifetime Reliability

Various failure mechanisms that could result in IC errors have been extensively studied in the literature. They can be broadly classified into two categories: *extrinsic failures* and *intrinsic failures*. Extrinsic failures, e.g., interconnect shorts/opens during fabrication, are mainly caused by manufacturing defects. Most of them are weeded out during the manufacturing test and burn-in process [16], [17]. Intrinsic failures can be further categorized into *soft errors* and *hard errors*. As soft errors [18] caused by radiation effects do not fundamentally damage the circuit, they are not viewed as lifetime reliability threats. In this paper, we mainly consider those hard errors that are permanent once they manifest. The most representative ones include time-dependent dielectric breakdown (TDDB) in the gate oxides, electromigration (EM) and stress migration (SM) in the interconnects, and negative bias temperature instability (NBTI) stresses that shift PMOS transistor threshold voltages. Many widely accepted reliability models for the above failure mechanisms at device and circuit level have been proposed and empirically validated by academia and industry [19], [20], [21], [22], [23], [24], and it is shown that they are strongly related to the temperature and voltage applied to the circuit.

The above hard intrinsic failures have recently reattracted lots of research interests, due to their increasingly adverse effects with technology scaling. Srinivasan et al. [6], [13] presented an application-aware architecture-level model named *RAMP* that is able to dynamically track lifetime reliability of a processor according to application behavior, where the sum-of-failure-rate (SOFR) model is used to combine the effects of different failure mechanisms. This model, however, is inherently inaccurate because it assumes a uniform device density over the chip and an identical vulnerability of devices to failure mechanisms. Later, to address this problem, Shin et al. [15] defined reference circuits and presented a structure-aware lifetime reliability estimation framework that takes the vulnerability of basic structures of the microarchitecture (e.g., register files, latches, and logic) to different failure mechanisms into account. The above models target a single-core processor's lifetime reliability. Coskun et al. [25] proposed a simulation methodology to evaluate the lifetime reliability of multicore systems, and used it to optimize the system's power management policy. For the sake of simplicity, most of the above models assumed exponential failure distributions and thus cannot capture the processors' accumulated aging effects. In addition, for the processors' operational temperatures, the above models either used the average temperature value over a period of time or tried to trace the temperature variations accurately. The accuracy of the former method is questionable, while the computation complexity for the latter case is too high to be adopted during design space exploration.

Recently, Huang and Xu [12] proposed to model the lifetime reliability of homogeneous manycore systems using a load-sharing nonrepairable k-out-of-n:G system with general failure distributions for embedded cores, taking core-level redundancy into account. This model assumes that a processor core is in one of three states (processing, wait, and spare), each corresponding to a unique albeit arbitrary failure distribution. In practice, however, the lifetime reliability of a processor core strongly depends on its operational temperature, which varies with different applications running on it even in the same state. In addition, how to obtain the failure distributions for each state is not shown in this work.

2.2 Task Allocation and Scheduling for MPSoC Designs

There is a rich literature on static task allocation and scheduling algorithms. Various issues have been considered, including timing constraint, communication cost, precedence relationship, reliability, static/dynamic priority, and task duplication [26], [27]. Since the problem of scheduling tasks on multiprocessors for a single objective has been proved to be an NP-complete problem, heuristic algorithms such as list scheduling [28] are widely used in the industry. To achieve better results, various statistical optimization techniques (e.g., genetic algorithm, simulated annealing, and tabu search) were also proposed to tackle this problem.

Most prior work in reliability-driven task allocation and scheduling (e.g., [8], [9], [10]) assumes processors' failure rates to be independent of their usage history. This assumption might be applicable for modeling random soft errors in IC products, but it is obviously inaccurate for the wearout-related hard errors considered in this work. As discussed earlier, for lifetime reliability threats, we should consider the more reasonable increasing failure rates during the task allocation and scheduling process.

Many recent studies on task scheduling for MPSoC systems aimed at balancing different processors' temperatures or keeping them under a threshold (e.g., [14], [29], [30]). These techniques might improve the system's lifetime reliability implicitly, since operational temperature has a significant impact on ICs' lifetime reliability. However, since wearout failures are also affected by many other factors (e.g., the circuit structure, voltage, and operating frequency), these thermal-aware techniques may not balance the aging effects among processors, especially for heterogeneous MPSoCs. As a result, some processors may still age faster than the others and hence result in shorter system service life. On one hand, this heterogeneity might simply come from the different processors' microarchitectures [15]. On the other hand, even for homogeneous systems, structurally identical processors can have different reliability budgets due to process variation. That is, an imperfect manufacturing process can lead to significant variation in device parameters (such as, channel length and threshold voltage) among transistors and hence reliability-related parameters among processor cores on the same die [31], [32].

Let us consider the following motivational example. Suppose we have an MPSoC platform containing two processors P_1 and P_2. The MTTF due to electromigration can be modeled as $MTTF_{EM} \propto J^{-n} e^{\frac{E_a}{kT}} \propto (V_{dd} \times f \times p_i)^{-n} e^{\frac{E_a}{kT}}$ (typically $n = 2$ [19]), where V_{dd}, f, p_i, E_a, k, and T represent the supply voltage, the clock frequency, the transition probability within a clock cycle, a material-related constant, the Boltzmann's constant, the absolute temperature, respectively [33]. Suppose $f_1 = 2f_2$, i.e., the clock frequency of P_1 is twice of that of P_2, and all other parameters are the same, the lifetime of P_2 is four times of that of P_1. That is, even if we are able to balance the operational temperatures of the two processors to be exactly the same all the time, processor P_1 will be the lifetime bottleneck of the MPSoC because it ages much faster than P_2.

From the above, we can reach to the conclusion that it is essential to *explicitly* take the lifetime reliability into consideration during the task allocation and scheduling process for MPSoC designs [34], which motivates this work. A relevant work targeting this problem was presented in [7] recently. The authors suggested to use lookup tables that fit with lognormal distribution curves to precalculate processors' MTTF, but the details are missing. In addition, their work targets the hardware/software cosynthesis design methodology, different from the platform-based MPSoC designs studied in our work.

3 Proposed Task Allocation and Scheduling Strategy

In this section, we formulate the lifetime reliability-aware task allocation and scheduling problem for platform-based MPSoC designs (Section 3.1) and we propose to use the simulated annealing technique to solve this problem. The solution representation, cost function, and simulated annealing process are presented in Sections 3.2, 3.3, and 3.4, respectively.

3.1 Problem Definition

As mentioned earlier, the platform-based MPSoC may be composed of nonidentical processors, where the heterogeneity comes from various sources. For example, the processors might be structurally identical but belong to different voltage-frequency islands or they have entirely different structures. Thus, a task may consume different execution times and powers on different processors. The problem studied in this work is formulated as follows: Given

- A directed acyclic task graph $\mathbf{G} = (\mathbf{V}, \mathbf{E})$, wherein each node in $\mathbf{V} = \{v_i : i = 1, \ldots, n\}$ represents a task, and $\mathbf{E}$ is the set of directed arcs which represent precedence constraints. Each task i has a deadline d_i. If a task does not have deadline, its d_i is set to be ∞;
- A platform-based MPSoC embedded system that consists of a set of k processors and its floorplan;
- Execution time table $\mathbf{L} = \{t_{i,j} : i = 1, \ldots, n, j = 1, \ldots, k\}$, where $t_{i,j}$ represents the execution time of task i on processor j;

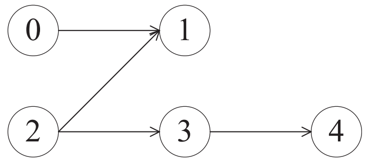


Fig. 1. An example task graph.

- Power consumption table $\mathbf{R} = \{r_{i,j} : i = 1, \ldots, n, j = 1, \ldots, k\}$, where $r_{i,j}$ represents the power consumption of processor j when it executes task i;
- Parameters of failure mechanisms (e.g., the activation energy for the diffusion processes E_a of electromigration) and the time-independent parameter of the corresponding failure distributions (e.g., the slope parameter β in Weibull distribution).

To determine a static periodical task allocation and schedule that is able to maximize the expected service life (or, lifetime) of the MPSoC embedded system under the performance constraint that every task finishes before its deadline.

Note that, while we mainly consider processor cores in this work because of their heavy wearout stress, our solution can be easily extended to take other hardware resources on the MPSoC platforms into account, if necessary. In addition, as the first step to tackle the above complicated problem, we assume that the voltage and frequency of processors do not change at runtime, although many MPSoC platforms employ dynamic voltage/frequency scaling (DVFS). This work, in spite of that, is applicable for MPSoCs with multiple voltage-frequency islands.

3.2 Solution Representation

The edges in the task graph $\mathbf{G} = (\mathbf{V}, \mathbf{E})$ indicate the dependencies of tasks, that is, there is a directed edge (v_i, v_j) in $\mathbf{E}$ if and only if task v_i must have been finished before v_j start its execution (denoted as $v_i \prec v_j$). For example, the task graph shown in Fig. 1 reflects the following relationship: $0 \prec 1, 2 \prec 1, 2 \prec 3$, and $3 \prec 4$. For any directed acyclic graph, there exists at least one order of the tasks that conforms to the partial order designated by the task graph (defined as a *valid schedule order*) and can be used as the task assignment order. For the above example, (0, 2, 1, 3, 4) and (2, 3, 4, 0, 1) are both valid schedule orders.

Thus, the task allocation and schedule for an MPSoC design can be represented as (*schedule order sequence*; *resource assignment sequence*) [35]. For example, given the task graph in Fig. 1 and two processors (P_1 and P_2) that can be used to execute any task, a solution represented as (0, 2, 1, 3, 4; P_1, P_1, P_2, P_1, P_2) means that task 0 is scheduled first, followed by tasks 2, 1, 3, and 4, respectively. As for the resource assignment, tasks 0, 2, and 3 are executed on P_1 while tasks 1 and 4 are assigned to P_2. Although this representation has been proposed in previous work for a genetic algorithm (e.g., [35]), in this paper it is used in a simulated annealing algorithm where the methodology to generate new solutions is totally different. We also provide a mathematical proof for the completeness of the search space with our proposed method (see Section 3.4).

Reconstructing schedule from the above solution representation is quite straightforward. In each step, we pick up

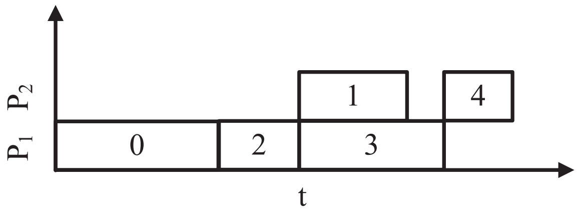


Fig. 2. A feasible task allocation and schedule.

a task according to the schedule order, assign it to the corresponding processor at its earliest available time, and then update the available time of all the processors. We can, then, obtain the ending time of every task i (denoted as e_i) to identify whether it violates the deadline constraint d_i. Clearly, a solution corresponds to a task schedule if its schedule order conforms to the partial order defined by $\mathbf{G}$. A possible schedule for the example solution representation is shown in Fig. 2.

3.3 Cost Function

As the guidance for decision making, cost function also plays an important role in simulated annealing. Generally speaking, the solution with lower cost means a preferable choice and hence should be accepted with higher possibility. The cost function is, therefore, defined following this principle. That is, in our problem, on one hand the solution should meet the performance (i.e., timing) constraints, while on the other hand we need to maximize the lifetime of platform-based MPSoC embedded systems subject to this requirement. We, therefore, introduce two terms into the cost function, respectively, as follows:

$$Cost = \mu \cdot 1_{\{\exists i: e_i > d_i\}} - MTTF^{sys}, \tag{1}$$

where the first term indicates the deadline violation penalty. To be specific, μ is a significant large number, and $1_{\{\cdot\}}$ is the indicator function. This function is equal to 1 if a schedule cannot meet deadline; otherwise, it is equal to 0. Thus, if a schedule violates the deadline constraints, the cost of this solution will be very large and hence be abandoned. Otherwise, the first term disappears and only the second term remains.

By comparing ending time e_i and deadline constraint d_i for any task i, it is easy to know whether performance constraints are violated, while, as mentioned before, the lifetime estimation is a nontrivial problem with extremely high computational complexity. Our proposed method for handling this problem is presented in Sections 4 and 5 in detail.

3.4 Simulated Annealing Process

The proposed SA-based algorithm starts with an initial solution obtained by any deterministic task scheduling algorithm (e.g., list scheduling) and the "temperature" of this solution is initialized as a high value. This temperature gradually decreases during the simulated annealing process. At each temperature T_a, a certain amount of iterations is conducted and some neighbor solutions are considered. Once we reach a new solution, its cost (denoted as $Cost_{new}$) is computed using (1) where $MTTF^{sys}$ is substituted by (26), and compared to that of the old one (denoted as $Cost_{old}$). If $Cost_{new} < Cost_{old}$, the new solution is accepted;

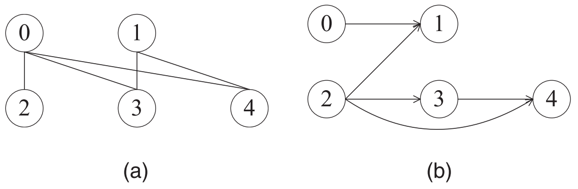


Fig. 3. Two transforms of directed acyclic graph. (a) $\widehat{\mathbf{G}}$ and (b) $\widetilde{\mathbf{G}}$.

otherwise, the probability that the new solution is accepted is $e^{-(Cost_{new} - Cost_{old})/T_a}$. When T_a is as low as the preset ending temperature, the simulated annealing process is terminated and the solution with the lowest cost obtained so far is regarded as the final solution. During the simulated annealing process, it is important to be able to reach the entire solution space from an initial solution, in order not to fall into local minimum point.

Before introducing the details on how we identify new solutions from a random initial solution, we first introduce two transforms of directed acyclic graph. With the given task graph $\mathbf{G}$, we can construct an *expanded task graph* $\widehat{\mathbf{G}} = (\mathbf{V}, \widehat{\mathbf{E}})$, which has the same nodes as $\mathbf{G}$, but with more directed edges. That is, if the task graph implies a precedence constraint, an edge is added into $\widehat{\mathbf{G}}$. Fig. 3a shows the corresponding expanded task graph to the task graph in Fig. 1. While there is no edge (2, 4) in Fig. 1, task 2 must be executed before task 4 because $\mathbf{E}$ contains edges (2, 3) and (3, 4). Thus, an edge (2, 4) is included in $\widehat{\mathbf{E}}$. Moreover, we construct an undirected *complement graph* $\widetilde{\mathbf{G}} = (\mathbf{V}, \widetilde{\mathbf{E}})$. There is an undirected edge (v_i, v_j) in $\widetilde{\mathbf{E}}$ if and only if there is no precedence constraints between v_i and v_j (denoted as $v_i \bigcirc v_j$). The corresponding complement graph to Fig. 1 is shown in Fig. 3b. $v_i \preceq v_j$ is used to represent that either $v_i \prec v_j$ or $v_i \bigcirc v_j$.

With these notations, we theoretically prove that any valid schedule order sequence of the given task graph $\mathbf{G}$ is reachable, starting from an arbitrary initial sequence, as shown in the following.

Lemma 1. *Given a valid schedule order* $A = (a_1, a_2, \cdots, a_{|\mathbf{V}|})$, *swapping adjacent nodes lead to another valid schedule order, provided that there is an edge between these two nodes in graph* $\widetilde{\mathbf{G}}$.

Proof. Since A is a valid schedule order, we have the property: $a_1 \preceq a_2 \preceq \cdots \preceq a_i \preceq a_{i+1} \preceq \cdots \preceq a_{|\mathbf{V}|}$. If the edge $(a_i, a_{i+1}) \in \widetilde{\mathbf{E}} (1 \le i \le |\mathbf{V}| - 1)$, there is no precedence constraints between them. In other words, $a_1 \preceq a_2 \preceq \cdots \preceq a_i \bigcirc a_{i+1} \preceq \cdots \preceq a_{|\mathbf{V}|}$. If we swap the position of task a_i and a_{i+1}, no precedence constraints are violated and hence we have another valid schedule order. □

Theorem 2. *Starting from a valid schedule order* $A = (a_1, a_2, \ldots, a_{|\mathbf{V}|})$, *we are able to reach any other valid schedule order* $B = (b_1, b_2, \ldots, b_{|\mathbf{V}|})$ *after finite times of adjacent swapping.*

Proof. A feasible procedure is shown in Fig. 4. In this procedure, the positions of nodes in the sequence are adjusted one by one. That is, we first find node b_1 in sequence A, move it to the first position A by a series of adjacent swapping. And then, find node b_2 in sequence A and move it to the second position of sequence A. After the

```
$(a_1, a_2, \cdots, a_{|\mathbf{V}|}) \Rightarrow (b_1, b_2, \cdots, b_{|\mathbf{V}|})$
1 For $i = 1$ to $n-1$
2     If $a_i \neq b_i$
3         find $j$, where $a_j = b_i$
4         For $k = j-1$ to $i$
5             swap $a_k$ and $a_{k+1}$ in sequence $A$
```

Fig. 4. Swapping procedure.

first $(i-1)$ iterations, $a_1 = b_1, a_2 = b_2, \ldots, a_{i-1} = b_{i-1}$. At the ith iteration, if $a_i = b_i$, there is no need to adjust the position of a_i. Otherwise, we move node a_j to the ith position of sequence A by $(j-i)$ times of swapping (line 5). None of them violate precedence constraints. The main reason is: since A is a valid schedule order, we have $a_i \preceq a_{i+1} \preceq \cdots \preceq a_j$. On the other hand, since B is also a valid order, $b_i \preceq b_{i+1} \preceq \cdots \preceq b_{|\mathbf{V}|}$. Note that $a_j = b_i$. Thus, $a_i \preceq a_{i+1} \preceq \cdots \preceq a_j \preceq b_{i+1} \preceq \cdots \preceq b_{|\mathbf{V}|}$. In addition, set $\{a_i, a_{i+1}, \ldots, a_{j-1}\} \subseteq$ set $\{b_{i+1}, \ldots, b_{|\mathbf{V}|}\}$. Consequently, $a_j \bigcirc a_{j-1}, a_j \bigcirc a_{j-2}, \ldots a_j \bigcirc a_i$. □

Accordingly, three moves are introduced to reach all possible solutions, starting with an arbitrary valid initial solution

- M1: Swap two adjacent nodes in both schedule order sequence and resource assignment sequence, provided that there is an edge between these two nodes in graph $\widetilde{\mathbf{G}}$.
- M2: Swap two adjacent nodes in resource assignment sequence only.
- M3: Change the resource assignment of a task.

With the above moves, all possible task schedules are reachable starting from an arbitrary initial one. This is because, for a certain resource and task binding M1 essentially can visit all other valid schedule orders starting from an initial one, while M2 and M3 guarantee that all resource assignment sequence can be tried.

In the following two sections, we present how to efficiently obtain $MTTF^{sys}$ in (1), i.e., the MTTF of an MPSoC design with a particular task allocation and schedule.

4 Lifetime Reliability Computation for MPSoC Embedded Systems

The well-accepted failure mechanism models in the literature (e.g., [20]) typically provide the relationship between $MTTF$ and a fixed temperature T. However, processors' operational temperature varies significantly with different applications. Generally speaking, when a processor is under usage or its "neighbors" on the floorplan are being used, its temperature is relatively higher than otherwise. In this section, we introduce a novel analytical method to estimate the lifetime reliability of MPSoC embedded systems running periodical tasks, within which the existing failure models are taken as inputs and the influence of temperature variation caused by task alternations is reflected.

We use Weibull distribution to describe the wearout effects, as suggested in JEP85 [36]. Since the slope parameter is shown to be nearly independent of temperature [37], the reliability of a single processor at time t can be expressed as

$$R(t,T) = e^{-\left(\frac{t}{\alpha(T)}\right)^{\beta}}, \tag{2}$$

where $T, \alpha(T)$, and β represent temperature, the scale parameter, and the slope parameter in the Weibull distribution, respectively. Instead of assuming T as a fixed value, we consider the temperature variations in our analytical model for more accuracy. At the same time, it is important to note that the other factors that affect a processor's lifetime reliability are also considered in the model. That is, the architecture properties of processor cores are reflected on the slope parameter β, while the cores' various operational voltages and frequencies manifest themselves on $\alpha(T)$ (see (6)). Since temperature T varies with respect to time t, it can be regarded as a function of t. This allow us to eliminate the notation T from $R(t,T)$ in the rest of this paper.

In general, mean time to failure is defined as

$$MTTF = \int_0^{\infty} R(t)\mathrm{d}t. \tag{3}$$

Considering Weibull distribution given in (2), this equation can be deduced into the formation shown below [38]

$$MTTF(T) = \alpha(T)\Gamma\left(1 + \frac{1}{\beta}\right). \tag{4}$$

Rearranging the equations yields the expression of the scale parameter, i.e.,

$$\alpha(T) = \frac{MTTF(T)}{\Gamma\left(1 + \frac{1}{\beta}\right)}. \tag{5}$$

Our analytical framework takes the hard error models as inputs, and hence it is applicable to analyze any kinds of failure mechanisms, including the combined failure effects shown in [6], [13]. For the sake of simplicity, we take electromigration failure mechanism as an example. By substituting its lifetime model into (5), we obtain the corresponding scale parameter

$$\alpha(T) = \frac{A_0(J - J_{crit})^{-n} e^{\frac{E_a}{kT}}}{\Gamma\left(1 + \frac{1}{\beta}\right)}, \tag{6}$$

where A_0 is a material-related constant, $J = V_{dd} \times f \times p_i$ [33], and J_{crit} is the critical current density.

Depending on processor's temperature variations with respect to time, we obtain a subdivision of the time $[0,t]$: $0 = t_0 < t_1 < t_2 < \cdots < t_m = t$. For all $\varepsilon_T > 0$, there exists $\delta_t > 0$ such that if the largest partition $\max_i(t_{i+1} - t_i) < \delta_t$, then for all i the difference between the highest temperature in this interval $\max_{t_i<\tau<t_{i+1}} T(\tau)$ and the lowest one $\min_{t_i<\tau<t_{i+1}} T(\tau)$ is less than ε_T. Denote by $[t_i, t_{i+1})$ the $(i+1)$th time interval and let $\Delta t_i = t_{i+1} - t_i$. We assume that the temperature during $[t_i, t_{i+1})$ is an arbitrary constant T_i within the range $[\min_{t_i<\tau<t_{i+1}} T(\tau), \max_{t_i<\tau<t_{i+1}} T(\tau)]$. We know that the initial reliability of the processor is given by

$$R(t)\big|_{t=t_0} = 1. \tag{7}$$

For the first interval $[t_0, t_1)$, since the temperature is fixed to T_0, by (2), we have

$$R(t) = e^{-\left(\frac{t}{\alpha(T_0)}\right)^{\beta}}, \qquad t_0 \le t < t_1. \tag{8}$$

At the end of this interval, the reliability is

$$R(t_1^-) = e^{-\left(\frac{t_1}{\alpha(T_0)}\right)^{\beta}}. \tag{9}$$

Using a quantity c to represent the aging effect in $[t_0, t_1)$, we express the reliability in the second interval as

$$R(t) = e^{-\left(\frac{t+c}{\alpha(T_1)}\right)^{\beta}}, \qquad t_1 \le t < t_2. \tag{10}$$

At the beginning of the second interval

$$R(t_1^+) = e^{-\left(\frac{t_1+c}{\alpha(T_1)}\right)^{\beta}}, \tag{11}$$

c can be computed by the continuity of reliability function, that is, $R(t_1^-) = R(t_1^+)$, yielding

$$c = \left(\frac{\alpha(T_1)}{\alpha(T_0)} - 1\right) \cdot t_1. \tag{12}$$

Substituting it into (10), we obtain

$$R(t) = e^{-\left(\frac{t}{\alpha(T_1)} + \left(\frac{1}{\alpha(T_0)} - \frac{1}{\alpha(T_1)}\right)\cdot t_1\right)^{\beta}}, \qquad t_1 \le t < t_2. \tag{13}$$

More generally, the reliability function must satisfy the following continuity constraints:

$$R(t_\ell^-) = R(t_\ell^+), \qquad \ell = 1, 2, \ldots, m-1. \tag{14}$$

By generalizing the above calculation steps, the lifetime reliability of a processor at time t can be written as

$$R(t) = e^{-\left(\frac{t}{\alpha(T_\ell)} + \eta_\ell\right)^{\beta}}, \qquad t_\ell \le t < t_{\ell+1}, \tag{15}$$

where

$$\eta_\ell = \sum_{i=0}^{\ell-1} \left(\frac{1}{\alpha(T_i)} - \frac{1}{\alpha(T_{i+1})}\right) \cdot t_{i+1}. \tag{16}$$

With (15) and (16), we can compute $MTTF$ by (3), but we still need to monitor the processor's temperature, which is obviously time consuming.

Fortunately, since the tasks are executed periodically, the temperature variance with respect to time will alsobe periodical after it is stabilized. We, hence, can divide each period into the same subdivisions. Given each task execution period is divided into p time intervals, by (15) and (16), a processor's lifetime reliability at the end of first period is given by

$$R(t_p) = e^{-\left(\sum_{i=0}^{p-1} \frac{\Delta t_i}{\alpha(T_i)}\right)^{\beta}}. \tag{17}$$

Similarly, a processor's reliability at the end of the mth period can be expressed as

$$R(t_{m\cdot p}) = e^{-\left(\sum_{i=0}^{m\cdot p-1} \frac{\Delta t_i}{\alpha(T_i)}\right)^{\beta}}. \tag{18}$$

We notice that the changes of reliability function $R(t)$ in different periods are different, while $\sum_{i=0}^{p-1} \frac{\Delta t_i}{\alpha(T_i)}$ does not vary from period to period. That is,

$$\begin{aligned} [-\ln R(t_{m\cdot p})]^{\frac{1}{\beta}} &= \sum_{i=0}^{m\cdot p-1} \frac{\Delta t_i}{\alpha(T_i)} = m \sum_{i=0}^{p-1} \frac{\Delta t_i}{\alpha(T_i)} \\ &= m \cdot [-\ln R(t_p)]^{\frac{1}{\beta}}. \end{aligned} \tag{19}$$

We, therefore, introduce the concept of *aging effect* of a processor in a period A, which enables us to integrate all lifetime reliability-related characteristics (including temperature, voltage, clock frequency, etc.) of a processor and its utilization together in this single value

$$A = [-\ln R(t_p)]^{\frac{1}{\beta}} = \sum_{i=0}^{p-1} \frac{\Delta t_i}{\alpha(T_i)}. \tag{20}$$

Because typically $MTTF \gg t_p$, the $MTTF$ of a single processor defined as (3) can be approximated to

$$MTTF = \sum_{i=0}^{\infty} e^{-(i\cdot A)^{\beta}} \cdot t_p. \tag{21}$$

The above is the lifetime estimation of a single processor. For an MPSoC platform, let us denote processor j's aging effect as A_j and its slope parameter as β_j and assume that there is no spare processors in the system (i.e., the system fails if one processor fails), the $MTTF$ of the entire system can be approximately expressed as

$$MTTF^{sys} = \sum_{i=0}^{\infty} e^{-\sum_j (i\cdot A_j)^{\beta_j}} \cdot t_p. \tag{22}$$

While from the mathematical point of view the extension from the lifetime estimation of single processor to that of MPSoC platform is simply a production operation, (22) essentially does not lose any information about the correlation between processors. To clarify, let us consider an important feature of an MPSoC platform as an example, that is, the execution of a processor affects the temperature of its neighbors. When we estimate the system lifetime, the heating effect of processor j caused by not only itself but also other processors is reflected in its A_j. Similarly, the influence of processor j's behavior on others also affects their aging effect parameters. These A_j's, finally, bring the correlation between processors into the system lifetime estimation $MTTF^{sys}$.

5 Efficient MPSoC Lifetime Approximation

It is essential to be able to quickly evaluate the cost of a solution during the simulated annealing process because this task needs to be conducted whenever we find a solution. Calculating $MTTF^{sys}$ according to (22) directly, however, is quite time consuming, which limits our design space exploration capability. To tackle this problem, four speedup techniques are introduced in this section.

5.1 Speedup Technique I—Multiple Periods

Remind that the aging effect A_j of processor j is the same for every period. Obviously, its aging effect of ν periods can be expressed as $A_j \cdot \nu$. As long as the condition $t_p \cdot \nu \ll MTTF$ is still satisfied (i.e., ν is much less than the number of operational periods before permanent system failure),

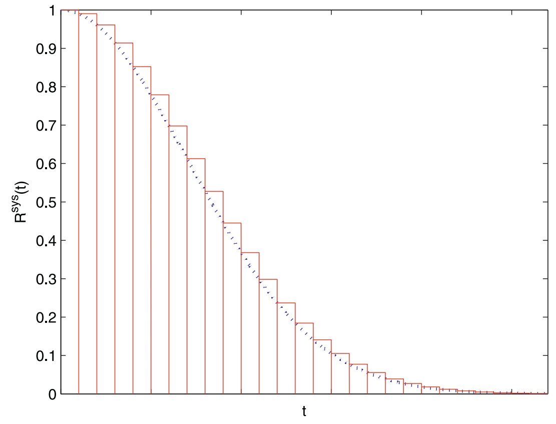


Fig. 5. Approximation for the system's MTTF.

$MTTF^{sys}_{appxI}$ defined by the following equation can be a lifetime approximation

$$MTTF^{sys}_{appxI} = \sum_{i=0}^{\infty} e^{-\sum_j (i \cdot A_j \cdot \nu)^{\beta_j}} \cdot t_p \cdot \nu. \tag{23}$$

The idea behind this estimation is shown in Fig. 5, wherein the area inside the dotted curve is the system's actual MTTF while the approximated $MTTF^{sys}_{appxI}$ is the area inside the solid rectangles. As can be easily observed, although $MTTF^{sys}_{appxI}$ is not the accurate mean time to failure of the system, it is an effective indicator for the lifetime with different task schedules, because a task schedule with relatively larger MTTF tends to have larger $MTTF^{sys}_{appxI}$. This technique benefits us significantly in terms of computational time, i.e., ν times faster than the case without this technique.

5.2 Speedup Technique II—Steady Temperature

To obtain an accurate A_j used in (23) is a quite time-consuming process because the time interval $[t_i, t_{i+1})$ needs to be set as a very small value. Fortunately, the time for processors to reach steady temperature with task changes in the platform is typically much shorter than the execution time of tasks [39], [40]. As an example, we demonstrate the temperature variations for a sample MPSoC platform containing three processors in Fig. 6a, obtained from HotSpot [41], an efficient and accurate thermal simulator that is able to calculate transient and/or steady temperature of on-chip computing elements. Fig. 6b shows the corresponding processors that are under usage at a particular time. From this figure, we can observe that the processors stay at a relatively stable temperature most of the time when the tasks do not change. With this observation, we propose to calculate A_j at a much coarser time scale based on such steady temperature within each time slot as shown in Fig. 6b.

5.3 Speedup Technique III—Temperature Precalculation

Even though A_j could be calculated efficiently with the above speedup techniques, we have to run HotSpot temperature simulator [41] to obtain the temperature information every time the simulated annealing algorithm reaches a solution. Let us perform a simple calculation. Suppose the initial and end temperature of an algorithm are 10^2 and 10^{-5}, respectively, cooling rate is 0.95, and 1,000

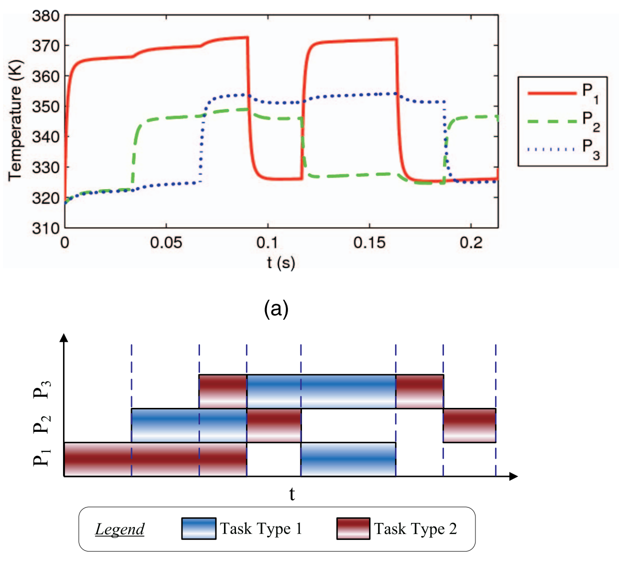


Fig. 6. An example of slot representation and the corresponding temperature variations. (a) Temperature variations and (b) slot representation.

neighbor solutions are searched at the same algorithm temperature, the time-consuming HotSpot simulator needs to be called $1{,}000 \times \log_{0.95} \frac{10^2}{10^{-5}} \approx 3 \times 10^5$ times, which is obviously unaffordable. To avoid this problem, we propose to conduct the HotSpot simulation in a precalculation phase.

To precalculate the processors' temperatures, we define a series of *time slots* for task schedules. Each one is identified by the set of busy processors and the power consumption of the tasks running on these processors,[1] as shown in Fig. 6. Since the power consumptions can be different when the same task execute on distinct processors and when different tasks execute on the same processor, the number of possible time slots is huge and it is very difficult, if not impossible, to run HotSpot once and precalculate the aging effects for all the cases. To tackle this problem, we categorize the tasks into m types (m is a user-defined value) based on their power consumptions when running on a processor and we assume the tasks belonging to the same category have the same power consumption value when they run on the same processor. Since every processor can be either used or unused in a time slot, and each processor in use has m possible power consumption values, there can be at most $\sum_{i=1}^{k} m^i \binom{k}{i} = ((1+m)^k - 1)$ kinds of time slots in task schedules, where k is the quantity of processors on the platform. Here, m^i means that when i processors are in use, each has m possible power consumption values. In total, there are $\binom{k}{i}$ possible combinations that i-out-of-k processors are under used. The possible values of occupied processor quantity i are $1, 2, \ldots, k$.

Denote by x^i_ℓ $(1 \le i \le k, 1 \le \ell \le m)$ the event that processor i is under usage in a time slot, and the task running on this processor belongs to type ℓ. Note that the time slot here is independent of the length of time interval. We separate these two concepts based on the observation that as long as the time interval is not very short, its aging effect

1. In practice, the power consumption for a task may vary with different inputs, and hence we use the average power consumption here, as in [6].

can be approximated by the steady temperature (as mentioned in Section 5.2), which is independent of the length. Each slot can be described by a set of x_ℓ^i, denoted by $\mathbf{X}$. We omit the processors in idle state in the representation of set $\mathbf{X}$. A task schedule is composed of a list of time slots. A feasible way to identify the time slot in a schedule is to cut the schedule into a series of time intervals at the task starting or ending time points. For example, suppose an embedded system contains three processors and its tasks are classified into two types, in the time order the schedule shown in Fig. 6b consists of seven slots: $\{x_2^1\}, \{x_2^1, x_1^2\}, \{x_2^1, x_1^2, x_2^3\}, \{x_2^2, x_1^3\}, \{x_1^1, x_1^3\}, \{x_2^3\}, \{x_2^2\}$.

Let $T_j^{\mathbf{X}}$ be the steady temperature of processor j in time slot $\mathbf{X}$. Because the steady temperature depends on power consumption of processors and floorplan, and all are fixed in a slot, there are exactly $((1+m)^k - 1)$ possible $T_j^{\mathbf{X}}$ values for processor j. Given the steady temperature of processor j in time slot $\mathbf{X}$ (i.e., $T_j^{\mathbf{X}}$), we calculate the aging effect factor of processor j, denoted as $\phi_j^{\mathbf{X}}$. Here, *aging effect factor* is the aging effect in unit time, defined as

$$\phi_j^{\mathbf{X}} = \frac{1}{\alpha(T_j^{\mathbf{X}})}. \tag{24}$$

For example, since $\mathbf{X}$ of the first slot in Fig. 6 is $\{x_2^1\}$, processor P_1's steady temperature is $T_1^{\{x_2^1\}}$. Its aging effect factor $\phi_1^{\{x_2^1\}}$ equals $1/\alpha(T_1^{\{x_2^1\}})$. Given the length of the first slot $\Delta t_0, P_1$'s aging effect in this slot is $\Delta t_0/\alpha(T_1^{\{x_2^1\}})$. It is necessary to highlight that, in any slot, not only under usage processors but also idle ones have aging effect. For processor P_1, we should also estimate its steady temperature and aging effect factor for the time slots where P_1 is not under usage (e.g., the 4th slots). The aging effect of P_1 in this schedule in a period can be computed by

$$A_1' = \frac{\Delta t_0}{\alpha(T_1^{\{x_2^1\}})} + \frac{\Delta t_1}{\alpha(T_1^{\{x_2^1, x_1^2\}})} + \frac{\Delta t_2}{\alpha(T_1^{\{x_2^1, x_1^2, x_2^3\}})} + \frac{\Delta t_3}{\alpha(T_1^{\{x_2^2, x_1^3\}})} + \frac{\Delta t_4}{\alpha(T_1^{\{x_1^1, x_1^3\}})} + \frac{\Delta t_5}{\alpha(T_1^{\{x_2^3\}})} + \frac{\Delta t_6}{\alpha(T_1^{\{x_2^2\}})}. \tag{25}$$

The aging effect of other processors in a period can be computed in the same method.

Then, combining the speedup techniques II and III, for a task schedule we can compute A_j' for every processor j. Replacing the accurate A_j in (23) with A_j' yields

$$MTTF_{appxII}^{sys} = \sum_{i=0}^{\infty} e^{-\sum_j (i \cdot A_j' \cdot \nu)^{\beta_j}} \cdot t_p \cdot \nu. \tag{26}$$

5.4 Speedup Technique IV—Time Slot Quantity Control

We notice that the number of possible time slots $((1+m)^k - 1)$ increases exponentially with the increase of on-chip processor cores m. This issue can be effectively resolved based on the observation that when a core is in execution, usually only nearby cores' temperatures are affected. Therefore, we can identify those neighboring processor cores based on the MPSoC's floorplan and precalculate the temperatures for a much less number of time slots. In practice, the processor cores on an MPSoC platform oftentimes do not crowd together (i.e., separated by other functional blocks), and hence can be naturally divided into a few regions and we conduct temperature estimation for them separately during the precalculation phase.

6 Experimental Results

6.1 Experimental Setup

To evaluate the effectiveness and efficiency of the proposed methodology, we conduct experiments on a set of random task graphs generated by TGFF [42] running on various hypothetical MPSoC platforms. The number of tasks ranges from 20 to 260, and the maximum in and out degree of a task is set to be the default values used in TGFF (i.e., 3 and 2, respectively). The number of processor cores varies between two and eight. By the speedup technique IV, a large platform that contains six or eight processors is partitioned into two domains for precalculation. Unless specified otherwise, all the speedup techniques presented in Section 5 are applied on the proposed algorithm for approximation. We have also considered the homogeneity of platforms. For homogeneous platforms, all processor cores have the same execution time for a certain task. For heterogeneous ones, two kinds of processor cores are assumed: main processors and coprocessors. The former ones have relatively higher processing capability than the latter ones in most cases. For all task graphs and platforms, we compare the proposed strategy with an existing thermal-aware task allocation and scheduling algorithm proposed in [14] (abbreviated in tables to thermal aware). List scheduling is utilized in [14], i.e., a list of unscheduled tasks is maintained and the task with the highest priority is scheduled iteratively in a deterministic manner. To reduce the peak temperature, task energy consumptions are taken into consideration in [14] when calculating the priority. Once the task schedule is constructed, its makespan (i.e., the time interval that all periodical tasks need to finish their executions once) becomes known. For fair comparison, it is used as the reference deadline for the proposed approach.

The simulated annealing parameters are set as follows: initial $\text{temperature} = 100$, cooling $\text{rate} = 0.95$, end temperature $= 10^{-5}$, and the number of random moves at each $\text{temperature} = 1{,}000$. Moreover, because of the lack of public empirical data on the weight of influence of various failure mechanisms on real circuit, we use the electromigration failure model presented in [43] in our experiments.[2] The corresponding parameters are set as the cross-sectional area of conductor $A_c = 6.4 \times 10^{-8}\ \text{cm}^2$, the current density $J = 1.5 \times 10^6\ \text{A/cm}^2$, and the activation energy $E_a = 0.48\ \text{eV}$. Further, the power density of platforms is in the range of 3.33 to $12.5\ \text{W/cm}^2$ and the tasks are categorized into three groups depending on their power consumption. The slope parameter in Weibull distribution used for describing the processor cores' lifetime reliability in homogeneous platforms is set as $\beta = 2$. While in heterogeneous ones, the slope parameters

2. Our model can be applied to other failure mechanisms as well. We can also combine the effects of multiple failure mechanisms and derive an overall MTTF based on [6] and [13].

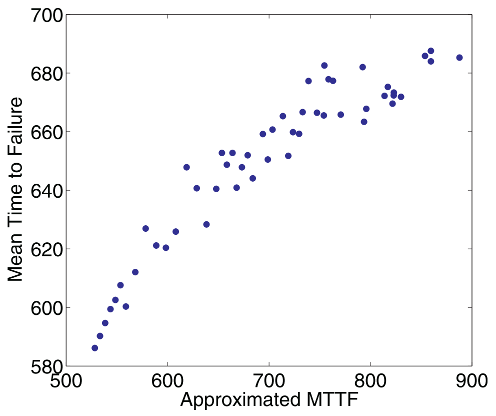


Fig. 7. Comparison between approximated $MTTF$ and accurate value.

of main processors and coprocessors are set to 2.5 and 2, respectively. Unless otherwise specified, the clock frequency of the main processors in heterogeneous platforms is set to be twice of that of the coprocessors and the one in homogeneous platforms, i.e., the *frequency ratio* is 2.

In addition, we define a reference platform, which contains a single processor core with a fixed temperature 351.5 K, slope parameter of Weibull distribution $\beta = 2$, and the same clock frequency as the processor cores in homogeneous ones. Its $MTTF$ is set to be 1,000 units. The $MTTF$ obtained in our experiments are normalized to this reference case for easier comparison.

6.2 Results and Discussion

Let us first validate the approximation techniques used for MTTF estimation. By using our algorithm, we obtain a set of valid task schedules (i.e., the task schedules that meet the deadlines) for a homogeneous 2-processor platform. For each schedule, the approximated $MTTF$ are computed using (26), where ν is set to 100. Then, we derive the accurate $MTTF$ values by monitoring the temperature variation using HotSpot for the same schedules, and compare them to the approximated values. As shown in Fig. 7, our approximation is able to reflect the quality of different valid schedules. That is, if a schedule has larger mean time to failure, it tends to have larger approximated value. Also, it is worth noting that because of exponentially increased CPU execution time overhead with respect to the number of processors in the platform, we are not able to provide accurate $MTTF$ for larger platforms.[3]

Next, we present experimental results obtained with various platforms and task graphs in Table 1. Columns 1 and 2 indicate the number of main processors and coprocessors on the platform; Columns 3 and 4 describe the task graph; Column 5 is the makespan obtained by thermal-aware task allocation and scheduling algorithm in [14] and is used as the baseline deadline of our algorithm; Column 6 is the corresponding platforms' lifetime. In the last six columns, we obtain platforms' lifetime by using our algorithm, relaxing the deadline used in our algorithm by 0, 5, and 10 percent, respectively.

As shown in this table, in most cases the results obtained with our algorithm have longer lifetime than that of thermal-aware one even if the deadlines of both algorithms are the same (see Columns 7 and 8). The only exception is the "2 processors 22 task" case (Row 4). When the same deadline is assumed, we observe the same lifetime resulted from our algorithm and that in [14]. We attribute this phenomenon to the simple MPSoC and the small task graph. That is, in this case, the schedules that are able to meet the deadlines are quite limited and we are not able to find a solution with extended MTTF. If we relax the deadline by 5 or 10 percent, the advantage of the proposed lifetime reliability-aware task scheduling algorithm is more obvious (see Columns 9 to 12). Taking the last row as an example, with the deadline relaxation, the lifetime extension ratio increases from 38.09 to 54.64 percent, and to 76.31 percent. Also, we notice that our algorithm provides more benefit if the platform is a heterogenous one. For example, when we relax the deadline by 5 percent, the lifetime improvement on heterogeneous 6-processor platform is 41.93 percent, much higher than that on homogenous 6-processor platform, which is 12.17 percent. This is mainly because, for heterogeneous platforms, the thermal-aware task allocation and scheduling algorithm in [14] is based on the list scheduling technique and tends to assign tasks to main processors because the main processors have better performance. In this case, it is very likely that the aging effects of the main processors are much serious than that of the coprocessors, while our algorithm is able to achieve more balanced aging among these processors.

A closer observation for 8-processor platforms is shown in Table 2. For the same platform, more task graphs are scheduled on it. When we target a larger task graph, the lifetime improvement obtained by our algorithm tends to be larger. For example, if we relax the deadline constraints by 10 percent, the lifetime improvements on the homogeneous platform for a task graph with 131 tasks and that with 201 tasks are 16.86 and 20.62 percent, respectively. We attribute it to the more valid solutions with larger number of tasks. However, it should be noted that the effectiveness of the proposed methodology also depends on many other factors, such as the detailed precedence dependencies.

Fig. 8 shows the difference ratio between the MTTF obtained from the proposed approach and [14], as a function of frequency ratio. Here, frequency ratio γ is an important factor that reflects platform heterogeneity, representing that the clock frequency of the main processors is set to be $\gamma\times$ of that of the coprocessors. We schedule the task graph with 131 tasks on the 8-core heterogeneous platform. The benefit provided by the proposed approach significantly increases as the ratio between main processors and coprocessors grows. In other words, the proposed method performs better when the heterogeneity of the MPSoC system is high. For example, as frequency ratio increases from 1 to 4, even when no deadline relaxation is allowed, the lifetime extension ratio grows by a factor of 2.65. Also, the improvement achieved by deadline relaxation is more significant when the system heterogeneity is high. Consider the deadline extension by 5 percent as an example. The lifetime extension ratio improvements are 7.33 and 20.47 percent for the same frequency ($1\times$) case and $4\times$ case, respectively.

3. The MTTF values shown in the following experiments are approximated ones.

TABLE 1
Lifetime Reliability of Various MPSoC Platforms with Different Task Graphs

Platform Description		Task Description		Deadline (s)	Thermal-Aware [14]	Simulated Annealing					
						0% $\mathcal{DR}$		5% $\mathcal{DR}$		10% $\mathcal{DR}$	
Main PE	Co-PE	Task	Edge		$MTTF$	$MTTF$	Δ (%)	$MTTF$	Δ (%)	$MTTF$	Δ (%)
2	0	22	23	535	492.47	492.47	0	582.30	18.24	582.30	18.24
4	0	49	76	1106	216.05	226.87	5.01	247.31	14.47	263.38	21.91
2	2			697	137.44	161.33	17.38	171.20	24.56	185.59	35.03
6	0	76	106	918	228.87	239.91	4.82	256.73	12.17	273.28	19.40
2	4			676	97.18	125.07	28.70	137.93	41.93	150.00	54.35
8	0	131	190	1227	227.24	235.78	3.76	250.86	10.39	265.56	16.86
2	6			984	88.00	121.78	38.09	136.38	54.64	155.50	76.31

Δ: Difference ratio between $MTTF$ of simulated annealing and $MTTF$ of thermal-aware; $\mathcal{DR}$: Deadline Relaxation

TABLE 2
Lifetime Reliability of 8-Processor Platforms

Task Description	8 Core Homogenous Platform				8 Core Heterogenous Platform			
	Thermal-Aware [14]	Simulated Annealing			Thermal-Aware [14]	Simulated Annealing		
		$\mathcal{DR}$(%)	$MTTF$	Δ (%)		$\mathcal{DR}$(%)	$MTTF$	Δ(%)
# of Task: 101 # of Edge: 142	Deadline: 1059s $MTTF$:240.07	0	247.79	3.21	Deadline: 809s $MTTF$:91.64	0	129.04	40.81
		5	264.25	10.07		5	146.01	59.33
		10	279.64	16.48		10	160.50	75.14
# of Task: 131 # of Edge: 190	Deadline: 1227s $MTTF$:227.24	0	235.78	3.76	Deadline: 984s $MTTF$:88.00	0	121.78	38.09
		5	250.86	10.39		5	136.38	54.64
		10	265.56	16.86		10	155.50	76.31
# of Task: 201 # of Edge: 292	Deadline: 1809s $MTTF$:207.26	0	221.03	6.64	Deadline: 1416s $MTTF$:85.27	0	130.42	52.95
		5	235.95	13.84		5	143.71	68.54
		10	250.00	20.62		10	149.71	75.57
# of Task: 251 # of Edge: 366	Deadline: 2014s $MTTF$:191.38	0	203.37	6.27	Deadline: 1693s $MTTF$:85.73	0	124.21	44.89
		5	216.56	13.16		5	137.88	60.83
		10	230.17	20.27		10	151.10	76.25

We are also interested in the trade-off between performance and mean time to failure. The experimental results for two sample cases, a heterogeneous 8-processor platform and a homogeneous 4-processor one, are shown in Fig. 9. We can observe that the $MTTF$ generally increases with the relaxation of deadlines. This is mainly because the flexibility of selecting task schedules increases with respect to the deadline relaxation. We can also observe that when the deadline constraint is relaxed to a certain point (e.g., deadline relaxation exceeds 160 percent in Fig. 9b), $MTTF$ starts to saturate. This is because the task schedule with the longest lifetime does not violate deadline constraints and has been selected as the solution.

Finally, as for the efficiency of our algorithm, the simulated annealing process requests 50-200 s of CPU time on Intel(R) Core(TM)2 CPU 2.13 GHz for each case in our experiments. For example, "4 processors 49 tasks" needs 84 seconds, while "8 processors 101 tasks" costs 158 seconds. The CPU time spending on precalculation (i.e., steady temperature estimation of time slots) ranges from 3 to 160 seconds. We have tried the precalculation for 8-processor platform without partitioning the platform into two regions. As expected, it requires extremely long CPU time (more than 5 hours), which illuminates the need of time slot quantity control (speedup technique IV). We also attempted to classify the tasks into five groups and keep the platform partitioning; the precalculation for 8-processor platform needs around 12 minutes. The

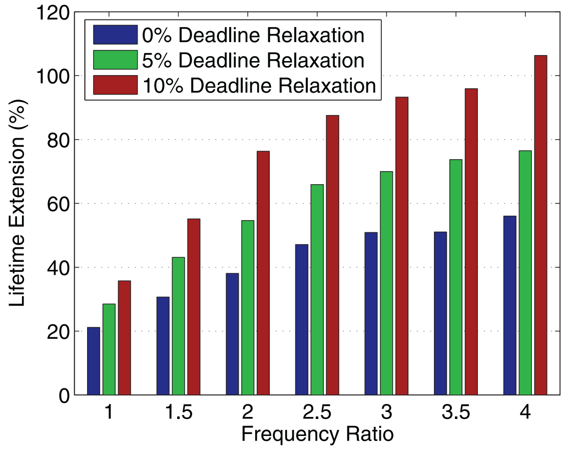


Fig. 8. The impact of heterogeneity on the effectiveness of the proposed strategy.

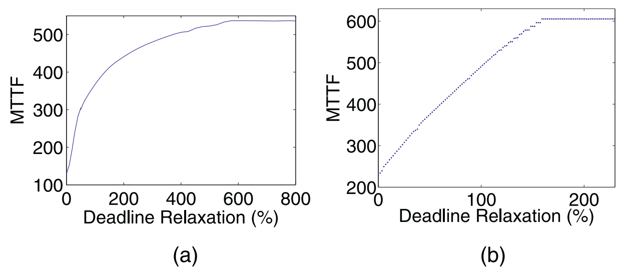


Fig. 9. The extension of $MTTF$ with the relaxation of deadlines. (a) Heterogeneous 8-processor platform. (b) Homogeneous 4-processor platform.

impact on MTTF accuracy highly depends on the floorplan of MPSoC platforms. In particular, when the processor cores are placed on the silicon die in a loose manner, we observe negligible temperature difference between the cases with and without time slot quantity control. In contrast, if the cores are centered in a small region, the temperature difference could be a few centigrade.

7 Conclusion and Future Work

The lifetime reliability of MPSoC designs has become a serious concern for the industry with technology scaling. To tackle this problem, different from prior work, we propose a simulated annealing-based task allocation and scheduling strategy that maximizes the lifetime of platform-based MPSoC designs under performance constraints. In order to efficiently estimate the lifetime reliability of various solutions with acceptable accuracy, we propose an analytical model and several speedup techniques, by explicitly taking the aging effects of processors into account. Experimental results show that the proposed techniques significantly extend the lifetime of platform-based MPSoC designs, especially for heterogeneous platform with large task graphs.

While our current solution has considered the possible processor cores' voltage/frequency differences and hence is applicable for designs with multiple voltage-frequency islands, we assume that they do not change at runtime. Since DVFS has been employed in many MPSoC platforms, we plan to take this effect into account in our future work.

Acknowledgments

The authors wish to thank the anonymous reviewers for their constructive comments. This work was supported in part by the General Research Fund CUHK417406, CUHK417807, and CUHK418708 from Hong Kong SAR Research Grants Council (RGC), in part by National Science Foundation of China (NSFC) under grant No. 60876029, in part by a grant N_CUHK417/08 from the NSFC/RGC Joint Research Scheme, and in part by the National High Technology Research and Development Program of China (863 program) under grant no. 2007AA01Z109.

References

[1] A. Jerraya, H. Tenhunen, and W. Wolf, "Guest Editors' Introduction: Multiprocessor Systems-on-Chips," *Computer,* vol. 38, no. 7, pp. 36-40, July 2005.

[2] *Hardware/Software Co-Design: Principles and Practice,* J. Staunstrup and W. Wolf, eds. Kluwer Academic Publishers, 1997.

[3] ARM, "ARM11 PrimeXsys Platform," http://www.jp.arm.com/event/images/forum2002/02-print_arm11_primexsys_platform_ian.pdf, 2011.

[4] B. Vermeulen, S. Oostdijk, and F. Bouwman, "Test and Debug Strategy of the PNX8525 Nexperia™ Digital Video Platform System Chip," *Proc. IEEE Int'l Test Conf. (ITC),* pp. 121-130, 2001.

[5] S. Borkar, "Designing Reliable Systems from Unreliable Components: The Challenges of Transistor Variability and Degradation," *IEEE Micro,* vol. 25, no. 6, pp. 10-16, Nov./Dec. 2005.

[6] J. Srinivasan, S.V. Adve, P. Bose, and J.A. Rivers, "The Case for Lifetime Reliability-Aware Microprocessors," *Proc. IEEE/ACM Int'l Symp. Computer Architecture (ISCA),* pp. 276-287, 2004.

[7] C. Zhu, Z. Gu, R.P. Dick, and L. Shang, "Reliable Multiprocessor System-on-Chip Synthesis," *Proc. IEEE/ACM Int'l Conf. Hardware/Software Codesign and System Synthesis,* pp. 239-244, 2007.

[8] A. Dogan and F. Ozguner, "Matching and Scheduling Algorithms for Minimizing Execution Time and Failure Probability of Applications in Heterogeneous Computing," *IEEE Trans. Parallel and Distributed Systems,* vol. 13, no. 3, pp. 308-323, Mar. 2002.

[9] S.M. Shatz, J.-P. Wang, and M. Goto, "Task Allocation for Maximizing Reliability of Distributed Computer Systems," *IEEE Trans. Computers,* vol. 41, no. 9, pp. 1156-1168, Sept. 1992.

[10] S. Srinivasan and N.K. Jha, "Safety and Reliability Driven Task Allocation in Distributed Systems," *IEEE Trans. Parallel and Distributed Systems,* vol. 10, no. 3, pp. 238-251, Mar. 1999.

[11] S. Tosun, N. Mansouri, E. Arvas, M. Kandemir, Y. Xie, and W.-L. Hung, "Reliability-Centric Hardware/Software Co-Design," *Proc. Int'l Symp. Quality of Electronic Design (ISQED),* pp. 375-380, 2005.

[12] L. Huang and Q. Xu, "On Modeling the Lifetime Reliability of Homogeneous Manycore Systems," *Proc. 14th IEEE Int'l Symp. Pacific Rim Dependable Computing (PRDC),* pp. 87-94, 2008.

[13] J. Srinivasan, S.V. Adve, P. Bose, and J.A. Rivers, "Exploiting Structural Duplications for Lifetime Reliability Enhancement," *Proc. IEEE/ACM Int'l Symp. Computer Architecture (ISCA),* pp. 520-531, 2005.

[14] Y. Xie and W.-L. Hung, "Temperature-Aware Task Allocation and Scheduling for Embedded Multiprocessor Systems-on-Chip (MPSoC) Design," *J. VLSI Signal Processing Systems,* vol. 45, pp. 177-189, 2006.

[15] J. Shin, V. Zyuban, Z. Hu, J. Rivers, and P. Bose, "A Framework for Architecture-Level Lifetime Reliability Modeling," *Proc. IEEE/IFIP Int'l Conf. Dependable Systems and Networks (DSN),* pp. 534-53, 2007.

[16] M. Bushnell and V. Agrawal, *Essentials of Electronic Testing.* Kluwer Academic Publishers, 2000.

[17] Q. Xu and N. Nicolici, "Resource-Constrained System-on-a-Chip Test: A Survey," *IEE Proc.- Computers and Digital Techniques,* vol. 152, no. 1, pp. 67-81, Jan. 2005.

[18] M. Nicolaidis, "Design for Soft Error Mitigation," *IEEE Trans. Device and Materials Reliability,* vol. 5, no. 3, pp. 405-418, Sept. 2005.

[19] J.R. Black, "Electromigration—A Brief Survey and Some Recent Results," *IEEE Trans. Electron Devices,* vol. ED-16, no. 4, pp. 338-347, Apr. 1969.

[20] "Failure Mechanisms and Models for Semiconductor Devices (jep122c)," *JEDEC Publication,* 2003.

[21] C.-K. Hu, R. Rosenberg, H.S. Rathore, D.B. Nguyen, and B. Agarwala, "Scaling Effect on Electromigration in On-Chip Cu Wiring," *Proc. IEEE Int'l Conf. Interconnect Technology,* pp. 267-269, 1999.

[22] J.H. Stathis, "Reliability Limits for the Gate Insulator in CMOS Technology," *IBM J. Research and Development,* vol. 46, nos. 2/3, pp. 265-283, 2002.

[23] S. Zafar, A. Kumar, E. Gusev, and E. Cartier, "Threshold Voltage Instabilities in High-κ Gate Dielectric Stacks," *IEEE Trans. Device and Materials Reliability,* vol. 5, no. 1, pp. 45-64, Mar. 2005.

[24] Z. Lu, W. Huang, M.R. Stan, K. Skadron, and J. Lach, "Interconnect Lifetime Prediction for Reliability-Aware Systems," *IEEE Trans. Very Large Scale Integration Systems,* vol. 15, no. 2, pp. 159-172, Feb. 2007.

[25] A. Coskun, T. Rosing, K. Mihic, G.D. Micheli, and Y.L. Lebici, "Analysis and Optimization of MPSoC Reliability," *J. Low Power Electronics,* vol. 15, no. 2, pp. 159-172, Feb. 2006.

[26] T.D. Braun, H.J. Siegel, N. Beck, L.L. Boloni, M. Maheswaran, A.I. Reuther, J.P. Robertson, M.D. Theys, B. Yao, D. Hensgen, and R.F. Freund, "A Comparison of Eleven Static Heuristics for Mapping a Class of Independent Tasks onto Heterogeneous Distributed Computing Systems," *J. Parallel and Distributed Computing,* vol. 61, no. 6, pp. 810-837, June 2001.

[27] Y.-K. Kwok and I. Ahmad, "Static Task Scheduling and Allocation Algorithms for Scalable Parallel and Distributed Systems: Classification and Performance Comparison," *Annual Review of Scalable Computing,* Y.C. Kwong, ed., pp. 107-227, Singapore Univ. Press, 2000.

[28] G. Liao, E.R. Altman, V.K. Agarwal, and G.R. Gao, "A Comparative Study of Multiprocessor List Scheduling Heuristics," *Proc. Hawaii Int'l Conf. System Sciences,* pp. 68-77, 1994.

[29] A.K. Coskun, T.S. Rosing, and K. Whisnant, "Temperature Aware Task Scheduling in MPSoCs," *Proc. Conf. Design, Automation, and Test in Europe (DATE),* pp. 1659-1664, 2007.

[30] K. Stavrou and P. Trancoso, "Thermal-Aware Scheduling: A Solution for Future Chip Multiprocessors Thermal Problems," *Proc. EUROMICRO Conf. Digital System Design (DSD),* pp. 123-126, 2006.

[31] S. Herbert and D. Marculescu, "Characterizing Chip-Multiprocessor Variability-Tolerance," *Proc. ACM/IEEE Design Automation Conf. (DAC),* pp. 313-318, 2008.

[32] S.R. Sarangi, B. Greskamp, R. Teodorescu, J. Nakano, A. Tiwari, and J. Torrellas, "VARIUS: A Model of Process Variation and Resulting Timing Errors for Microarchitects," *IEEE Trans. Semiconductor Manufacturing,* vol. 21, no. 1, pp. 3-13, Feb. 2008.

[33] A. Dasgupta and R. Karri, "Electromigration Reliability Enhancement via Bus Activity Distribution," *Proc. ACM/IEEE Design Automation Conf. (DAC),* pp. 353-356, 1996.

[34] L. Huang and Q. Xu, "Lifetime Reliability-Aware Task Allocation and Scheduling for MPSoC Platforms," *Proc. Design, Automation, and Test in Europe (DATE),* pp. 51-56, 2009.

[35] J. Oh and C. Wu, "Genetic-Algorithm-Based Real-Time Task Scheduling with Multiple Goals," *J. Systems and Software,* vol. 71, no. 3, pp. 245-258, May 2004.

[36] "Methods for Calculating Failure Rates in Units of Fits (jesd85)," *JEDEC Publication,* 2001.

[37] S.-C. Chang, S.-Y. Deng, and J.Y.-M. Lee, "Electrical Characteristics and Reliability Properties of Metal-Oxide-Semiconductor Field-Effect Transistors with Dy2O3 Gate Dielectric," *Applied Physics Letters,* vol. 89, no. 5, pp. 053504-1-053504-3, July 2006.

[38] C.E. Ebeling, *An Introduction to Reliability and Maintainability Engineering.* Waveland Press, 2005.

[39] R.C. Correa, A. Ferreira, and P. Rebreyend, "Scheduling Multiprocessor Tasks with Genetic Algorithms," *IEEE Trans. Parallel and Distributed Systems,* vol. 10, no. 8, pp. 825-837, Aug. 1999.

[40] A. Gerasoulis and T. Yang, "On the Granularity and Clustering of Directed Acyclic Task Graphs," *IEEE Trans. Parallel and Distributed Systems,* vol. 4, no. 6, pp. 686-701, June 1993.

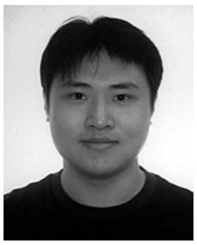
[41] K. Skadron, M.R. Stan, W. Huang, S. Velusamy, K. Sankaranarayanan, and D. Tarjan, "Temperature-Aware Microarchitecture," *Proc. IEEE/ACM Int'l Symp. Computer Architecture (ISCA),* pp. 2-13, 2003.

[42] R.P. Dick, D.L. Rhodes, and W. Wolf, "TGFF: Task Graphs for Free," *Proc. Int'l Conf. Hardware Software Codesign,* pp. 97-101, 1998.

[43] A.K. Goel, *High-Speed VLSI Interconnections,* second ed. IEEE Press, 2007.

Lin Huang (S'08) received the BS degree in electronic engineering from Shanghai Jiaotong University, Shanghai, China, in 2007. She received the MPhil. degree in computer science and engineering from The Chinese University of Hong Kong, Hong Kong, China, in 2010. Her research interests include reliability analysis of multicore systems, fault-tolerant computing, and network-on-chip testing. She is a student member of the IEEE.

Feng Yuan (S'08) received the BS degree in information engineering from Zhejiang University, Hangzhou, China, in 2006, and the MS degree in computer science and engineering from the Chinese University of Hong Kong in 2009. He is currently working toward the PhD degree in the CUHK Reliable Computer Laboratory (CURE Lab.), the Department of Computer Science and Engineering at the Chinese University of Hong Kong. His research interests include manufacturing test and fault-tolerant computing. He is a student member of the IEEE.



Qiang Xu (M'06) received the PhD degree in electrical and computer engineering from McMaster University, Hamilton, ON, Canada, in 2005. Since 2005, he has been an assistant professor with the Department of Computer Science and Engineering, The Chinese University of Hong Kong. He leads the CUHK Reliable Computing Laboratory (CURE Lab.). His research interests range from test and debug of system-on-a-chip integrated circuits to fault-tolerant computing. He has published more than 70 technical papers in these areas. He received the Best Paper Award at DATE '04 and has several other papers nominated for best paper awards at prestigious conferences such as ICCAD and DATE. He is a member of the IEEE, the ACM SIGDA, and the IEEE Computer Society. He is currently serving as an associate editor for *IEEE Design and Test of Computers*.