# Lifetime Reliability-Aware Task Allocation and Scheduling for MPSoC Platforms

Lin Huang, Feng Yuan and Qiang Xu
CUhk REliable computing laboratory (CURE)
Department of Computer Science & Engineering
The Chinese University of Hong Kong, Shatin, N.T., Hong Kong
Email: {lhuang,fyuan,qxu}@cse.cuhk.edu.hk

## Abstract

*With the relentless scaling of semiconductor technology, the lifetime reliability of embedded multiprocessor platforms has become one of the major concerns for the industry. If this is not taken into consideration during the task allocation and scheduling process, some processors might age much faster than the others and become the reliability bottleneck for the system, thus significantly 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 lifetime reliability-aware task allocation and scheduling algorithm based on 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 multiprocessor platforms and task graphs demonstrate the efficacy of the proposed approach.*

## 1 Introduction

Advancements in technology enable the integration of several microprocessors, dedicated digital hardware and sometimes mixed-signal circuits on a single silicon die, namely multiprocessor system-on-a-chip (MPSoC). There are mainly two approaches to design an MPSoC embedded system: (i). hardware/software co-synthesis; (ii). platform-based design. When compared to the flexible co-synthesis method that is able to explore more design space to obtain an application-specific architecture, the platform-based design approach has shorter time-to-market and less design cost and design risk, and hence is very popular in today's complex embedded system designs. In the platform-based design process, designers first pick a pre-defined MPSoC platform (e.g., [3]), and then map their applications onto this platform.

At the same time, the lifetime reliability of today's high-performance integrated circuits (ICs) has become a serious concern [5, 23, 26]. For embedded MPSoC platforms, if the wearout failures are not taken into consideration 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.

Most prior work in reliability-driven task allocation and scheduling (e.g., [12, 20]) assumes processors' failure rates to be constant values (i.e., independent of their usage times), which is unable to capture the system's accumulated aging effects. Since the ICs' failure mechanisms are strongly related to their operational temperatures, thermal-aware task scheduling techniques may improve the MPSoC's lifetime reliability implicitly, by balancing different processors' temperatures or keeping them under a safe threshold. However, as the ICs' wearout failures are also dependent on many other factors (e.g., internal structure, operational frequency and voltage), without explicitly taking the lifetime reliability into account during task allocation and scheduling, various processor cores may still age differently and thus result in shorter lifetime for the MPSoC embedded system.

In this paper, we propose novel solutions to increase the lifetime reliability of platform-based MPSoC embedded systems. The main contributions of our work include:

- we present an analytical model to estimate the lifetime reliability of platform-based MPSoC when executing periodical tasks;
- we propose a novel lifetime reliability-aware task allocation and scheduling algorithm that takes processors' aging effects into account, based on simulated annealing (SA) technique;
- we propose several techniques to simplify the design space exploration process with satisfactory solution quality;

*To the best of our knowledge, this is the first comprehensive work that explicitly take lifetime reliability into the task allocation and scheduling process for platform-based MPSoC embedded systems.*

The remainder of this paper is organized as follows. Section 2 reviews related prior work and motivates this paper. Our analytical model for the lifetime reliability of platform-based MPSoCs is introduced in Section 3. Next, Section 4 presents our proposed lifetime reliability-aware task allocation and scheduling algorithm. Experimental results on various hypothetical platform-based MPSoC designs are presented in Section 5. Finally, Section 6 concludes this paper.

## 2 Prior Work and Motivation

### 2.1 IC Lifetime Reliability

Various IC errors have been studied in the literature, which can be broadly classified into two categories: *extrinsic failures* and *intrinsic failures*. Extrinsic failures (e.g., interconnect shorts/opens) are mainly caused by manufacturing defects, which can be identified during the manufacturing test and burn-in process. 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, such as time dependent dielectric breakdown (TDDB), electromigration (EM), and negative bias temperature instability (NBTI).

While the fundamental causes of the above hard intrinsic failures have been studied for decades, it has recently re-attracted lots of research interests, due to their increasingly adverse effects with technology scaling. Srinivasan *et al.* [23, 24] proposed an application-aware architecture-level model *RAMP* to evaluate a processor's lifetime reliability. Shin *et al.* [21] defined reference circuits and presented a structure-aware lifetime reliability estimation framework. The above models focus on analyzing single-core processor's lifetime reliability. Coskun *et al.* [8] introduced two analytical frameworks for the lifetime reliability of multicore systems: a cycle-accurate simulation methodology and a statistical one, assuming uniform device density. Huang and Xu [15] proposed to model the lifetime reliability of homogeneous manycore systems using a load-sharing non-repairable $k$-out-of-$n$:$G$ system with general failure distributions for embedded cores.

Some of the above models (e.g., [8, 23]) 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 values over time or tried to trace the temperature variations accurately. The accuracy of the former method is obviously not accurate, while the computation complexity for the latter case is too high to be adopted during design space exploration. The above motivates us to develop a new MPSoC lifetime reliability analytical model, as shown in Section 3.

## 2.2 Task Allocation and Scheduling for MPSoC Embedded Systems

There is a rich literature on static task scheduling algorithms considering various issues, e.g., performance, communication cost, task duplication, processors' arbitrary connection [16]. 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 [17] are the most commonly used techniques. At the same time, since deterministic heuristics may fall into local minimum point, various optimization techniques (e.g., genetic algorithm, simulated annealing techniques) were proposed in the literature to tackle this problem.

Most prior work in reliability-driven task allocation and scheduling (e.g., [12, 20]) assumes processors' failure rates to be independent of their usage times. While this assumption can be accepted for random soft errors, it is obviously inaccurate for the wearout-related hard errors considered in this work, because the processors' lifetime reliability will gradually decrease over time.

Since performance, leakage power consumption, and reliability all have been shown to be highly related to temperature, many recent studies on task scheduling for MPSoC systems take thermal issues into account (e.g., [9, 25]), trying to balance different processors' temperatures or to keep them under a threshold. This may improve the system's lifetime reliability implicitly. 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. To take an example, suppose we have an MPSoC platform containing two heterogeneous processors $P_1$ and $P_2$. According to [10], the mean time to failure (MTTF) due to electromigration: $MTTF_{EM} \propto J^{-n} e^{\frac{E_a}{kT}} \propto (V_{dd} \times f \times p_i)^{-n} e^{\frac{E_a}{kT}}$ (typically $n = 2$ [4]), 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. 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 conclude that it is necessary to *explicitly* take the lifetime reliability into consideration during the task allocation and scheduling process for MPSoC embedded systems. A relevant work targeting this problem was presented in [26]. The authors considered wearout during hardware/software co-synthesis. They suggested to use lookup tables that fit with lognormal distribution curves to pre-calculate processors' MTTF, but the details are missing. In addition, their work targets the co-synthesis method for MPSoC designs, different from the platform-based MPSoC embedded systems studied in our work.

# 3 Lifetime Reliability Estimation for MPSoC Embedded Systems

In this section, we present an analytical method to estimate the lifetime reliability of MPSoC embedded systems running periodical tasks.

As suggested in JEP85 [1], we use Weibull distribution to describe the wearout effects. Since the slope parameter is shown to be nearly independent of temperature [6], the reliability of a single processor at time $t$ can be expressed as

$$R(t,T) = e^{-(\frac{t}{\alpha(T)})^{\beta}} \tag{1}$$

where $T$, $\alpha(T)$, $\beta$ represent temperature, the scale parameter, and the slope parameter in the Weibull distribution, respectively. Processors' operational temperatures vary significantly with different applications. Typically, when a processor is under usage or its "neighbors" on the floorplan are being used, its temperature is higher than otherwise. Therefore, 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 $\beta$, while the cores' various operational voltages and frequencies manifest themselves on $\alpha(T)$ (see Eq. (4)).

The $MTTF$ of a Weibull distribution given in Eq. 1 is

$$MTTF(T) = \alpha(T)\Gamma(1+\frac{1}{\beta}) \tag{2}$$

Simply, we have

$$\alpha(T) = \frac{MTTF(T)}{\Gamma(1+\frac{1}{\beta})} \tag{3}$$

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 [23, 24]. For the sake of simplicity, let us take electromigration failure mechanism as an example. The scale parameter is

$$\alpha(T) = \frac{A_0(J-J_{crit})^{-n} e^{\frac{E_a}{kT}}}{\Gamma(1+\frac{1}{\beta})} \tag{4}$$

where $A_0$ is a material-related constant, $J = V_{dd} \times f \times p_i$ [10], and $J_{crit}$ is the critical current density.

Depending on a processor's temperature variations with respect to time, we obtain a subdivision of the time $[0,t]$: $0 = t_0 < t_1 < t_2 < \cdots < t_m = t$. Let $[t_i, t_{i+1})$ denote the $(i+1)^{\text{th}}$ time interval and set $\Delta t_i = t_{i+1} - t_i$. We assume the temperature during $[t_i, t_{i+1})$ is a constant $T_i$. The initial reliability of the processor is simply $R(0) = 1$. By Eq. (1), for the first interval $[t_0, t_1)$, since the temperature is $T_0$, we have $R(t) = e^{-(\frac{t}{\alpha(T_0)})^{\beta}}$. At the end of this interval, the reliability is $R(t_1^-) = e^{-(\frac{t_1}{\alpha(T_0)})^{\beta}}$. As for the second interval, the reliability can be written as $R(t) = e^{-(\frac{t+c}{\alpha(T_1)})^{\beta}}$, where $c$ represents the aging effect in the first interval and can be computed by the continuity of reliability function. Since $R(t_1^+) = e^{-(\frac{t_1+c}{\alpha(T_1)})^{\beta}}$, by the continuity constraint $R(t_1^-) = R(t_1^+)$, we obtain $c = \left(\frac{\alpha(T_1)}{\alpha(T_0)} - 1\right) \cdot t_1$. Substituting it into the reliability function of the second interval yields $R(t) = e^{-\left(\frac{t}{\alpha(T_1)} + (\frac{1}{\alpha(T_0)} - \frac{1}{\alpha(T_1)}) \cdot t_1\right)^{\beta}}$.

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

$$R(t) = e^{-(\frac{t}{\alpha(T_\ell)} + \sum_{i=0}^{\ell-1} (\frac{1}{\alpha(T_i)} - \frac{1}{\alpha(T_{i+1})}) \cdot t_{i+1})^{\beta}}, \qquad t_\ell \le t < t_{\ell+1} \tag{5}$$

With Eq. (5), although we can compute the lifetime by $MTTF = \int_0^\infty R(t)\mathrm{d}t$, we still need to monitor the processor's temperature continuously, which is obviously time-consuming.

Fortunately, since the tasks are executed periodically, the temperature variance with respect to time will be also 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 Eq. (5), a processor's lifetime reliability at the end of first period is given by

$$R(t_p) = e^{-(\sum_{i=0}^{p-1} \frac{\Delta t_i}{\alpha(T_i)})^{\beta}} \quad (6)$$

Similarly, a processor's reliability at the end of the $m^{\text{th}}$ period can be expressed as

$$R(t_{m\cdot p}) = e^{-(\sum_{i=0}^{m\cdot p-1} \frac{\Delta t_i}{\alpha(T_i)})^{\beta}} \quad (7)$$

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,

$$[-\ln R(t_{m\cdot p})]^{\frac{1}{\beta}} = \sum_{i=0}^{m\cdot p-1} \frac{\Delta t_i}{\alpha(T_i)} = m \sum_{i=0}^{p-1} \frac{\Delta t_i}{\alpha(T_i)} = m \cdot [-\ln R(t_p)]^{\frac{1}{\beta}} \quad (8)$$

Therefore, we can define the aging effect of a processor in a period $A$ as,

$$A = [-\ln R(t_p)]^{\frac{1}{\beta}} = \sum_{i=0}^{p-1} \frac{\Delta t_i}{\alpha(T_i)} \quad (9)$$

This aging effect $A$ 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.

Because typically $MTTF \gg t_p$, the $MTTF$ of a single processor can be approximated to

$$MTTF = \sum_{i=0}^{\infty} e^{-(i\cdot A)^{\beta}} \cdot t_p \quad (10)$$

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 $\beta_j$ 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

$$MTTF^{sys} = \sum_{i=0}^{\infty} e^{-\sum_j (i\cdot A_j)^{\beta_j}} \cdot t_p \quad (11)$$

# 4 Proposed Task Allocation and Scheduling Algorithm

Considering periodical tasks on platform-based MPSoC embedded systems, this section presents a novel lifetime reliability-aware task allocation and scheduling algorithm.

## 4.1 Problem Definition

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 $\mathbf{V} = \{v_i : i = 1, \cdots, n\}$ represents a task in $\mathbf{G}$, 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 $\infty$;
- A platform-based MPSoC embedded system that consists of a set of $k$ processors and its floorplan;
- Execution time table $\mathbf{L} = \{t_{i,j} : 1 \le i \le n, 1 \le j \le k\}$, where $t_{i,j}$ represents the execution time of task $i$ on processor $j$;
- Power consumption table $\mathbf{R} = \{r_{i,j} : 1 \le i \le n, 1 \le j \le k\}$, where $r_{i,j}$ represents the power consumption of processor $j$ when it executes task $i$;
- Parameters of failure mechanisms (e.g., $E_{aEM}$ of electromigration) and the slope parameter of Weibull distribution (i.e., $\beta$).

To determine a periodical task allocation and schedule that is able to maximize the lifetime of the MPSoC embedded system under the constraint that every task finishes before its deadline.

## 4.2 Solution Representation

For the given task graph $\mathbf{G} = (\mathbf{V}, \mathbf{E})$, there is a directed edge $(v_i, v_j)$ in $\mathbf{E}$ if and only if task $v_i$ must precede $v_j$ in the task schedule. In other words, task $v_j$ cannot start until $v_i$ has been finished. The task allocation and schedule for the MPSoC can be represented as (*schedule order sequence*; *resource assignment sequence*) [19].

For example, given a task graph containing five tasks (see 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 genetic algorithm (e.g., [19]), in this paper it is used in simulated annealing algorithm where the methodology to generate new solutions is totally different.

Reconstructing schedule from the above solution representation is quite straightforward. In each step, we pick up 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 $d_i$. Apparently, a solution corresponds to a task schedule if its schedule order conforms to the partial order defined by $\mathbf{G}$.

## 4.3 Cost Function

As our objective is to maximize the lifetime of platform-based MPSoC embedded systems under the performance (i.e., timing) constraints, the cost function is given as follows:

$$Cost = \mu \cdot 1_{\{\exists i: e_i > d_i\}} - MTTF^{sys} \quad (12)$$

where, $\mu$ 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. Apparently, if a schedule violates the deadline constraints, the cost of this solution will be very large and hence be abandoned.

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 Eq. (11) directly, however, is very time-consuming, which limits our design space exploration capability. To tackle this problem, three speedup techniques are introduced, as discussed in the following.

**Speedup Technique I –** Instead of computing the aging effect in every period, we propose to compute the aging effect of $\nu$ periods at one time. Therefore, we obtain an approximated MTTF value $MTTF^{sys}_{appxI}$ as follows:

$$MTTF^{sys}_{appxI} = \sum_{i=0}^{\infty} e^{-\sum_j (i\cdot A_j\cdot \nu)^{\beta_j}} \cdot t_p \cdot \nu \quad (13)$$

Fig. 2 shows the impact of this estimation, 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., $\nu$ times faster than the case without this technique.

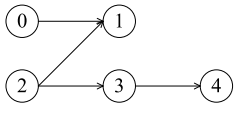



**Figure 1. An Example Task Graph.**

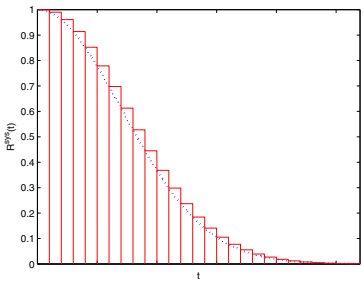



**Figure 2. Approximation for the System's MTTF.**

**Speedup Technique II –** To obtain an accurate $A_j$ used in Eq. (13) is a very 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 [7, 13]. We therefore propose to calculate $A_j$ at a much coarser time scale, i.e., based on the task changes. The temperature variations for an example MPSoC containing three processor cores are shown in Fig. 3(a), obtained from HotSpot. Fig. 3(b) 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.

**Speedup Technique III –** Even though $A_j$ could be calculated efficiently with the above speedup technique, we have to run HotSpot temperature estimation [22] 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 algorithm is $10^2$ and $10^{-5}$ respectively, cooling rate is 0.95, and 1000 neighbor solutions are searched at the same algorithm temperature, the HotSpot temperature estimation is called $1000 \times \log_{0.95} \frac{10^2}{10^{-5}} \approx 3 \times 10^5$ times. Apparently this is not affordable. To avoid this problem, we propose to conduct the HotSpot temperature estimation in a pre-calculation phase.

To pre-calculate the processors' temperatures, we define a series of *time slots* for task schedules. Each one is identified by the set of under-used processors and the power consumption of the tasks running on these processors[1], as shown in Fig. 3. Since the same task may consume different powers when executing on distinct processors, the number of possible time slots is huge and it is very difficult, if not impossible to run HotSpot once and pre-calculate the ageing 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 under-used processor has $m$ possible power consumption values, there can be at most $\sum_{i=1}^{k} m^i \binom{k}{i} = \left((1+m)^k - 1\right)$ kinds of time slots in task schedules.

Let $x_\ell^i$ $(1 \le i \le k, 1 \le \ell \le m)$ be the event that processor $i$ is under usage in a time slot, and the task running on this processor belongs to type $\ell$; each slot can be described by a set of $x_\ell^i$, (denoted as $\mathbf{X}$). A task schedule is composed of a list of time slots. For example, suppose an embedded system contains 3 processors and its tasks are classified into 2 types, in the time order the schedule shown in Fig. 3(b) consists of 7 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 $\left((1+m)^k - 1\right)$ 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, where

$$\phi_j^{\mathbf{X}} = \frac{1}{\alpha(T_j^{\mathbf{X}})} \tag{14}$$

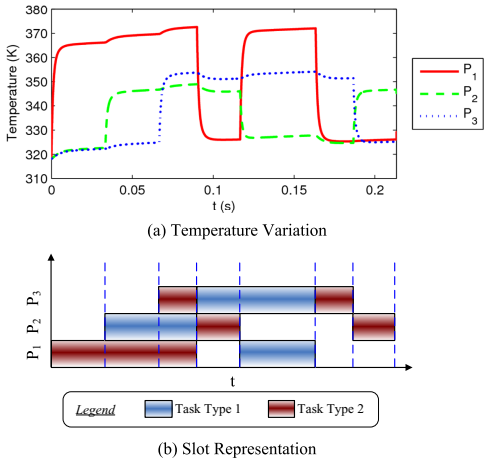
For example, in the first slot processor $P_1$'s steady temperature is $T_1^{\{x_2^1\}}$ (see Fig. 3). Its aging effect factor $\phi_1^{\{x_2^1\}}$ equals to $1/\alpha(T_1^{\{x_2^1\}})$. Given $\Delta t_0$ is the time interval of the first slot, $P_1$'s aging effect in this slot is $\Delta t_0/\alpha(T_1^{\{x_2^1\}})$. Note 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

$$A_1' = \frac{\Delta t_0}{\alpha(T_1^{\{x_2^1\}})} + \frac{\Delta t_1}{\alpha(T_1^{\{x_2^1, x_1^2\}})} + \frac{\Delta t_2}{\alpha(T_1^{\{x_2^1, x_1^2, x_2^3\}})} + \frac{\Delta t_3}{\alpha(T_1^{\{x_2^2, x_1^3\}})} + \frac{\Delta t_4}{\alpha(T_1^{\{x_1^1, x_1^3\}})} + \frac{\Delta t_5}{\alpha(T_1^{\{x_2^3\}})} + \frac{\Delta t_6}{\alpha(T_1^{\{x_2^2\}})} \tag{15}$$

Combining the speedup techniques II and III, for a task schedule we can compute $A_j'$ for every processor $j$. Substituting $A_j'$ for the accurate $A_j$ in Eq. (13) yields

$$MTTF_{appxII}^{sys} = \sum_{i=0}^{\infty} e^{-\sum_j (i \cdot A_j' \cdot \nu)^{\beta_j}} \cdot t_p \cdot \nu \tag{16}$$

Finally, the number of possible time slots increase exponentially with the increase of on-chip processor cores. 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 pre-calculate 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 pre-calculation phase.




(a) Temperature Variation

(b) Slot Representation

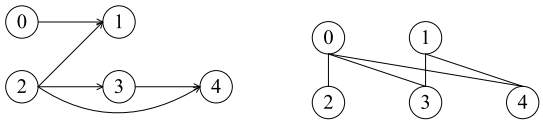
**Figure 3. An Example of Slot Representation and the Corresponding Temperature Variations.**

## 4.4 Simulated Annealing Process

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

[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 [23].

a precedence constraint, an edge is added into $\widehat{\mathbf{G}}$. Fig. 4(a) 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 added 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 $v_j$. The corresponding complement graph to Fig. 1 is shown in Fig. 4(b).




(a) $\widehat{\mathbf{G}}$ (b) $\widetilde{\mathbf{G}}$

**Figure 4. Two Transforms of Directed Acyclic Graph.**

We define a *valid schedule order* to be an order of the tasks that conforms to the partial order defined by task graph $\mathbf{G}$. For example, both (2, 3, 0, 4, 1) and (0, 2, 3, 1, 4) are valid schedule orders for the task graph in Fig. 1. It can be proved that

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

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

According to the above theorem[2], three kinds of moves are introduced to reach all possible solutions.

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

With the above moves, all possible task schedules are reachable starting from an arbitrary initial valid one. This is because, 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.

# 5 Experimental Results

## 5.1 Experimental Setup

To evaluate the efficacy of our algorithm, we consider a set of random task graphs generated by TGFF [11] whose task numbers ranges from 20 to 260 and a set of hypothetical MPSoC platforms with the number of processor cores ranging from 2 to 8. When pre-calculating the steady temperature, each large platform (containing 6 or 8 processors) is partitioned into two domains. 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 co-processors. The former ones have relatively higher processing capability than the latter ones in most cases. The same task graphs and platforms are also tested on a thermal-aware task scheduling algorithm [25]. For fair comparison, we use the makespan computed in [25] as the reference deadline, i.e., the time interval that all periodical tasks need to finish their executions once.

[2]The proof for this theorem is omitted due to space limitation of the paper.

We set the simulated annealing parameters as follows: initial temperature = 100, cooling rate = 0.95, end temperature = $10^{-5}$, and the number of random moves at each temperature = 1000. Moreover, the electromigration failure model presented in [14] is used in our experiments[3]. We set the cross-sectional area of conductor $A_c = 6.4 \times 10^{-8} cm^2$, the current density $J = 1.5 \times 10^6 A/cm^2$ and the activation energy $E_a = 0.48eV$. Further, the power density of platforms is in the range of 3.33 to 12.5 $W/cm^2$; and the tasks are categorized into 3 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 of the main processors and the co-processors are set to be 2.5 and 2, respectively. The clock frequency of the main processors in heterogeneous platforms is set to be twice of that of the co-processors and the one in homogeneous platforms.

Finally, we define a reference platform, which contains a single processor core with a fixed temperature $351.5K$, slope parameter $\beta = 2$, and the same clock frequency as the processor cores in homogeneous ones. Its $MTTF$ is set to be 1000 units. We normalize the $MTTF$ obtained in our experiments to this reference platform for easier comparison.

## 5.2 Results and Discussion

First of all, to validate the approximated $MTTF$ used in our cost function, we take the valid task schedules obtained from our algorithm (i.e., the task schedules meet the deadlines) and compute the approximated $MTTF$ using Eq. (16), where $\nu$ 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. 5 for a homogeneous 2-processor platform, our approximation is able to reflect the quality of different solutions. That is, if a schedule has larger mean time to failure, it tends to have larger approximated value. In addition, there is a trade-off between the accuracy and CPU execution time for our estimation. It is also 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.

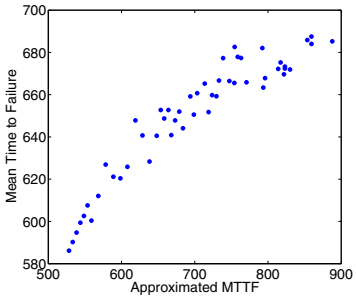



**Figure 5. Comparison between Approximated $MTTF$ and Accurate Value.**

Next, we present some results obtained with various platforms and task graphs in Table 1. Column 1 indicates the number of main processors and co-processors on the platform; Column 2 describes the task graph; Column 3 is the makespan obtained by thermal-aware scheduling algorithm and is used as the baseline deadline of our algorithm; Column 4 is the platforms' lifetime obtained by thermal-aware algorithm. In the last 6 columns, we obtain platforms' lifetime by using our algorithm, relaxing the deadline used in our algorithm by 0%, 5%, 10% respectively. As shown in this table, in most cases the re-

[3]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 [23, 24].

| Main / Co-PE | Task / Edge | Thermal Aware | | Simulated Annealing | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | $\mathcal{D}$ | $MTTF$ | 0% $\mathcal{DR}$ | | 5% $\mathcal{DR}$ | | 10% $\mathcal{DR}$ | |
| | | | | $MTTF$ | $\Delta$ (%) | $MTTF$ | $\Delta$ (%) | $MTTF$ | $\Delta$ (%) |
| 2/0 | 22/23 | 535 | 492.47 | 492.47 | 0 | 582.30 | 18.24 | 582.30 | 18.24 |
| 4/0 | 49 | 1106 | 216.05 | 226.87 | 5.01 | 247.31 | 14.47 | 263.38 | 21.91 |
| 2/2 | /76 | 697 | 137.44 | 161.33 | 17.38 | 171.20 | 24.56 | 185.59 | 35.03 |
| 6/0 | 76 | 918 | 228.87 | 239.91 | 4.82 | 256.73 | 12.17 | 273.28 | 19.40 |
| 2/4 | /106 | 676 | 97.18 | 125.07 | 28.70 | 137.93 | 41.93 | 150.00 | 54.35 |
| 8/0 | 131 | 1227 | 227.24 | 235.78 | 3.76 | 250.86 | 10.39 | 265.56 | 16.86 |
| 2/6 | /190 | 984 | 88.00 | 130.42 | 48.20 | 143.71 | 63.31 | 159.99 | 81.81 |

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

**Table 1. Lifetime Reliability of Various Platforms with Different Task Graphs.**

| Task / Edge | $\mathcal{DR}$ | 8 Core Homogenous Platform | | | | 8 Core Heterogenous Platform | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | Thermal Aware | | SA | | Thermal Aware | | SA | |
| | | $\mathcal{D}$ | $MTTF$ | $MTTF$ | $\Delta$ (%) | $\mathcal{D}$ | $MTTF$ | $MTTF$ | $\Delta$(%) |
| 101 | 0% | 1059 | 240.07 | 247.79 | 3.21 | 809 | 91.64 | 129.04 | 40.81 |
| / | 5% | | | 264.25 | 10.07 | | | 146.01 | 59.33 |
| 142 | 10% | | | 279.64 | 16.48 | | | 160.50 | 75.14 |
| 131 | 0% | 1227 | 227.24 | 235.78 | 3.76 | 984 | 88.00 | 130.42 | 48.20 |
| / | 5% | | | 250.86 | 10.39 | | | 143.71 | 63.31 |
| 190 | 10% | | | 265.56 | 16.86 | | | 159.99 | 81.81 |
| 201 | 0% | 1809 | 207.26 | 221.03 | 6.64 | 1416 | 85.27 | 130.42 | 52.95 |
| / | 5% | | | 235.95 | 13.84 | | | 143.71 | 68.54 |
| 292 | 10% | | | 250.00 | 20.62 | | | 149.71 | 75.57 |
| 251 | 0% | 2014 | 191.38 | 203.37 | 6.27 | 1693 | 85.73 | 124.21 | 44.89 |
| / | 5% | | | 216.56 | 13.16 | | | 137.88 | 60.83 |
| 366 | 10% | | | 230.17 | 20.27 | | | 151.10 | 76.25 |

**Table 2. Lifetime Reliability of 8-Processor SoC Platforms.**

sults obtained by using our algorithm have longer lifetime than that of thermal-aware scheduling algorithm even if the deadlines of both algorithms are the same (see Column 7-8). If we relax the deadline by 5% or 10%, the advantage of the proposed lifetime reliability-aware task scheduling algorithm grows more obvious (see Column 9-12). 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%, the lifetime improvement on heterogeneous 6-processor platform is 41.93%, much higher than that on homogenous 6-processor platform, which is 12.17%. This is mainly because, for heterogeneous platforms list scheduling-base thermal-aware algorithm [25] tends to assign tasks to main processors because the main processors have better processing capability. In this case it is very likely that the aging effect of main processors is much serious than that of co-processors. Our algorithm is able to solve this problem effectively. While for homogeneous platforms this tendency is less significant.

A closer observation for 8-processor platforms is shown in Table 2. For the same platform, 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% 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%, respectively. We attribute it to the more valid solutions with larger number of tasks.

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$s$, and "8 processors 101 tasks" costs 158$s$. While the CPU time spending on pre-calculation (i.e., steady temperature estimation of time slots) ranges from 3$s$ to 160$s$. We have also tried the pre-calculation for 8-processor platform without partitioning the platform into two regions. As expected, it requests extremely long CPU time (more than 5 hours). If we classify the tasks into 5 groups and keep the platform partitioning, the pre-calculation for 8-processor platform needs around 12 min.

# 6 Conclusion

Technology scaling has increasingly adverse effects on the lifetime of MPSoC embedded systems. In this work, we propose an analytical model to estimate the lifetime reliability of platform-based MPSoC embedded systems when executing periodical tasks, and we present a novel lifetime reliability-aware task allocation and scheduling algorithm that is able to take the aging effects of processors into account, based on simulated annealing technique. We have also presented how to simplify the design space exploration process with satisfactory solution quality. Experimental results show that our proposed techniques are able to increase the lifetime of platform-based MPSoCs significantly, especially for heterogeneous MPSoC platforms.

# 7 Acknowledgements

This work was supported in part by the General Research Fund 417406, 417807, and 418708 from Hong Kong SAR Research Grants Council (RGC), in part by National Science Foundation of China (NSFC) under grant No. 60876029, and in part by a grant N_CUHK417/08 from the NSFC/RGC Joint Research Scheme.

# References

[1] Methods for calculating failure rates in units of fits (jesd85). *JEDEC Publication*, 2001.
[2] Failure mechanisms and models for semiconductor devices (jep122c). *JEDEC Publication*, 2003.
[3] ARM. ARM11 primeXsys platform. http://www.jp.arm.com/event/images/forum2002/02-print_arm11_primexsys_platform_ian.pdf.
[4] J. R. Black. Electromigration - a brief survey and some recent results. ED-16(4):338–347, Apr. 1969.
[5] S. Borkar. Designing reliable systems from unreliable components: the challenges of transistor variability and degradation. 25(6):10–16, Nov.-Dec. 2005.
[6] 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. *App. Phy. Let.*, 89(5), Aug. 2006.
[7] R. C. Correa, A. Ferreira, and P. Rebreyend. Scheduling multiprocessor tasks with genetic algorithms. *IEEE Trans. on Paral. and Distrib. Sys.*, 10(8):825–837, Aug. 1999.
[8] A. Coskun, *et al*. Analysis and optimization of mpsoc reliability. *J. of Low Power Electronics*, 15(2):159–172, Feb. 2006.
[9] A. K. Coskun, T. S. Rosing, and K. Whisnant. Temperature aware task scheduling in MPSoCs. In *Proc. DATE*, pp. 1659–1664, 2007.
[10] A. Dasgupta and R. Karri. Electromigration reliability dnhancement via bus activity distribution. In *Proc. DAC*, pp. 353–356, 1996.
[11] R. P. Dick, D. L. Rhodes, and W. Wolf. TGFF: Task Graphs for Free. In *Proc. CODES+ISSS*, pp. 97–101, 1998.
[12] A. Dogan and F. Ozguner. Matching and scheduling algorithms for minimizing execution time and failure probability of applications in heterogeneous computing. *IEEE Trans. on Paral. and Distrib. Sys.*, 13(3):308–323, Mar. 2002.
[13] A. Gerasoulis and T. Yang. On the granularity and clustering of directed acyclic task graphs. *IEEE Trans. on Paral. and Distrib. Sys.*, 4(6):686–701, June 1993.
[14] A. K. Goel. *High-speed VLSI interconnections*. IEEE Press, 2nd edition, 2007.
[15] L. Huang and Q. Xu. On modeling the lifetime reliability of homogeneous many-core systems. In *Proc. PRDC*, pp. 87–94, 2008.
[16] Y.-K. Kwok and I. Ahmad. Static task scheduling and allocation algorithms for scalable parallel and distributed systems: classification and performance comparison. In Y. C. Kwong, editor, *Annual Review of Scalable Computing*, pp. 107–227. Singapore University Press, 2000.
[17] G. Liao, *et al.* A comparative study of multiprocessor list scheduling heuristics. In *Proc. HICSS*, pp. 68–77, 1994.
[18] M. Nicolaidis. Design for soft error mitigation. *IEEE Trans. on Dev. and Mat. Rel.*, 5(3):405–418, Sep. 2005.
[19] J. Oh and C. Wu. Genetic-algorithm-based real-time task scheduling with multiple goals. *J. of Sys. and Softw.*, 71(3):245–258, May 2004.
[20] S. M. Shatz, J.-P. Wang, and M. Goto. Task allocation for maximizing reliability of distributed computer systems. *IEEE Trans. Comput.*, 41(9):1156–1168, Sep. 1992.
[21] J. Shin, *et al.* A framework for architecture-level lifetime reliability modeling. In *Proc. DSN*, pp. 534–543, 2007.
[22] K. Skadron, *et al.* Temperature-aware microarchitecture. In *Proc. ISCA*, pp. 2–13, 2003.
[23] J. Srinivasan, *et al.* The case for lifetime reliability-aware microprocessors. In *Proc. ISCA*, pp. 276–287, 2004.
[24] J. Srinivasan, *et al.* Exploiting structural duplications for lifetime reliability Enhancement. In *Proc. ISCA*, pp. 520–531, 2005.
[25] Y. Xie and W.-L. Hung. Temperature-aware task allocation and scheduling for embedded multiprocessor systems-on-chip (mpsoc) design. *J. of VLSI Sig. Proc.*, 45:177–189, 2006.
[26] C. Zhu, *et al.* Reliable multiprocessor system-on-chip synthesis. In *Proc. CODES+ISSS*, pp. 239–244, 2007.