Customer-Aware Task Allocation and Scheduling for Multi-Mode MPSoCs

Lin Huang[†], Rong Ye[†‡] and Qiang Xu[†‡]
[†]CUhk REliable Computing Laboratory (CURE)
Department of Computer Science & Engineering
The Chinese University of Hong Kong, Shatin, N.T., Hong Kong
[‡]Shenzhen Institutes of Advanced Technology, Chinese Academy of Sciences
Email: {lhuang,rye,qxu}@cse.cuhk.edu.hk

ABSTRACT

Today's multiprocessor system-on-a-chip (MPSoC) products typically have multiple execution modes, and for each mode, all the products utilize the same task allocation and schedule strategy determined at design stage. As these products experience different usages by customers, such unified solution can at best be optimized for a hypothetical common case. It is hence likely that the product is not reliable or energy-efficient from particular customers' point of view. To tackle this problem, we propose a novel customer-aware task allocation and scheduling technique, wherein we generate an initial task schedule for each execution mode at design stage and then perform online adjustment at regular intervals for lifetime reliability improvement and/or energy reduction according to the specific usage strategy of individual products. Experimental results on several hypothetical MPSoCs with various task graphs demonstrate the effectiveness of the proposed personalized solution.

1. INTRODUCTION

When designing complex embedded systems, it is increasingly popular to take a pre-designed multiprocessor system-on-a-chip (MPSoC) platform (e.g., ARM PrimeXsys platform [1]) and map applications onto it. A critical step in the platform-based embedded system design flow is the so-called *task allocation and scheduling* (TAS) process, which determines the task mappings to processing elements (PEs) and their execution sequences to meet design specifications. Among the various optimization objectives during the TAS process, energy consumption is one of the primary concerns, especially for handheld embedded systems. Based on the widely-used dynamic voltage frequency scaling (DVFS) techniques [10], there is a rich literature on energy-efficient task allocation and scheduling techniques (e.g., [12]). Meanwhile, due to the ever-increasing temperature and power density of integrated circuits (ICs), the lifetime reliability of MPSoC products caused by wearout effects has also become a serious concern [2, 13, 17]. To tackle this problem, some recent work (e.g., [8]) proposed to explicitly take lifetime reliability into consideration during the TAS process.

Electronic products (e.g., smart phones) are seldom designed exclusively for one or a few specific customers. In most cases, a great volume of products are manufactured according to an identical design and shipped to different customers. The products, since bought by end users, however, start to experience different life stories. For instance, some users might mainly use their smart phones for making calls, while some others use it for music play most of the time. In other words, customers may have significant different usage strategies for the same system. We refer to this phenomenon as the *usage strategy deviation* of the product.

DAC 2011, June 5-10, 2011, San Diego, California, USA.

The allocation and scheduling of tasks on embedded systems at design stage, albeit carefully conducted, at best can be optimized for a hypothetical common case. It is very likely that the obtained TAS solution is not energy-efficient or reliable from particular customers' point of view, due to the fact that their usage strategy deviates significantly from the "common case". By lifting the implicit assumption of applying the unified task schedules on all products, a *personalized* TAS solution for each individualf product can be more energy-efficient and/or reliable.

Motivated by the above, we propose to design the product in such manner that it has the capability of extracting its own usage strategy and adjusting the task schedules at runtime when necessary. The customer-aware task schedules for multi-mode MPSoC embedded systems are constructed as follows. First, we generate an initial TAS solution for each execution mode such that the expected energy consumption is minimized under a given lifetime reliability constraint. Sequentially, we conduct online adjustment for each particular survival chip at regular interval based on its past usage strategy to guarantee its lifetime reliability and/or reduce its energy consumption. The performance overhead for online adjustment is negligible because the aging effect of IC products is typically in range of years and hence the adjustment interval can be set in the range of days or months. To have more flexibility during online adjustment, however, certain tasks may need to be mapped onto different types of PEs, which requires more design effort and storage space. This is taken into consideration in our solution as a constraint. Experimental results on hypothetical MPSoC products with various task graphs show that the proposed solution is able to significantly increase their lifetime reliability and is also helpful for energy reduction.

The remainder of this paper is organized as follows. Section 2 reviews related work and formulates the problem investigated in this paper. Section 3 presents the proposed simulated annealing (SA) based TAS technique used at design stage. Then, the reliability model for online adjustment and the corresponding online adjustment technique are introduced in Section 4. Experimental results are presented in Section 5. Finally, Section 6 concludes this paper.

2. PRELIMINARIES

2.1 Related Work and Motivation

Due to the ever-increasing adverse impact of relentless technology scaling on IC lifetime reliability, it is essential to take circuit aging effects into consideration during the task allocation and scheduling process. [8] first addressed this problem by generating a unique task schedule with maximum lifetime reliability for a single-mode embedded system. Later, the same authors extended their work in [7] for multi-mode MPSoCs, considering to minimize energy consumption under a given lifetime reliability constraint. The problem is solved in two phases: a set of possible task schedules are generated for each execution mode, and then a multi-mode combination algorithm is used to select a schedule from each set for global optimization.

In the above works, TAS solutions are generated at design stage and a unified task schedule for each execution mode is constructed for all the products. As discussed earlier, in the presence of customers' usage strategy deviation, the task schedules can be optimized at best with respect to a usage strategy distribution $f_{\mathbf{Y}}(\mathbf{y})$, obtained via market research or cus-

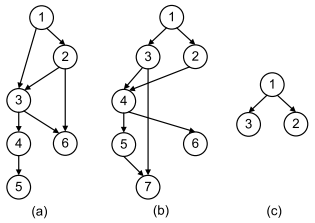


Figure 1: Task Graphs in the Example.

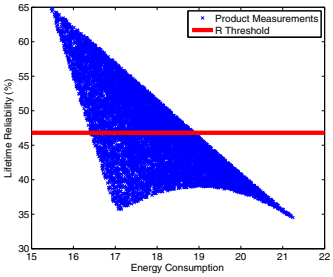


Figure 2: Impact of Usage Strategy Deviation.

tomer survey. In other words, setting energy savings as the optimization objective, we can construct task schedules to minimize the *expected* energy consumption of all products and guarantee that the *expected* lifetime reliability of all products is no less than a predefined threshold. The resulting unified schedule, however, may not perform well from particular customers' point of view.

To take an example, let us consider a simple MPSoC product with three execution modes and two processor cores. The task graphs are generated by TGFF [3] and shown in Fig. 1. The usage strategy distribution is set to $y_1 \sim N(0.5, 0.5^2)$, $y_2 \sim N(0.5, 0.5^2)$, and $y_3 = 1 - y_1 - y_2$, provided all variables are in the range between 0 and 1. We apply the TAS solution obtained from [7], which tries to minimize the expected energy consumption over usage strategy distribution under constraints onto a volume of products with the same design. The measurements of 10,000 sample products are plotted in Fig. 2. As shown in this figure, because of the usage strategy deviation, although the reliability threshold is set to 47%, the reliability of sample products ranges from 35% to 65%. In particular, around half of products (the marks below red line) are not reliable as expected, while the remaining have too much reliability margin that can be used to reduce energy.

The above observation motivates the proposed customer-aware task allocation and scheduling technique in this paper.

2.2 Problem Formulation

To achieve the objective of this work, we need to generate an initial TAS solution at design-stage and then conduct online adjustment after the product is deployed to the market. Consequently, we partition the problem into two phases and formulate them separately in this section.

Problem 1 [Design Stage]: Given

- q execution modes. A directed acyclic task graph $\mathcal{G}^k = (\mathcal{T}^k, \mathcal{E}^k)$ for mode k, wherein each node in $\mathcal{T}^k = \{\tau_i^k : 1 \le i \le n^k\}$ represents a task in $\mathcal{G}^k$, and $\mathcal{E}^k$ is the set of directed arcs which represent precedence constraints. Each task graph $\mathcal{G}^k$ has a deadline d^k;
- The joint probability density function[1] that the system is in various modes $f_{\mathbf{Y}}(\mathbf{y})$, where y_k represents the probability that the system is in execution mode k;
- A platform-based MPSoC embedded system that consists of a set of processors $\mathcal{P} = \{P_j : 1 \le j \le m\}$, belonging to $\mathcal{V}$ categories;
- Execution time table $\{c_{k,i,j} : 1 \le k \le q, 1 \le i \le n^k, 1 \le j \le m\}$, where $c_{k,i,j}$ is the execution time of task τ_i^k on processor P_j. If a task cannot be executed by a certain processor, the corresponding $c_{k,i,j}$ is set to infinity;
- Power consumption table $\{p_{k,i,j} : 1 \le k \le q, 1 \le i \le n^k, 1 \le j \le m\}$, where $p_{k,i,j}$ is the power consumption of τ_i^k on processor P_j;
- The target service life t_L and the corresponding reliability requirement $\eta\%$;

To determine a periodical task schedule for each execution mode such that the expected energy consumption over all products is minimized under the performance constraints that all tasks are finished before deadlines and the reliability constraint that the expected reliability at the target service life is no less than $\eta\%$.

With the initial task schedules generated at the design stage, for a particular MPSoC product, its task schedule can be online adjusted at regular intervals. The problem to be solved at the end of the u^{th} interval is formulated as

Problem 2 [Online Adjustment]: Given all parameters as specified in Problem 1, and

- Interval length t_I;
- Usage strategy $\mathbf{y}_\ell$ of the ℓ^{th} interval for $1 \le \ell \le u$;
- task mapping flexibility constraints;

To determine a periodical task schedule for each execution mode such that the energy consumption of this particular product is minimized under the performance and reliability requirement. It is worth to note that the lifetime reliability of the MPSoC products is a statistical value and the reliability constraint here is set as same as that in Problem 1, i.e., the percentage of products that survive until the target service life is no less than $\eta\%$.

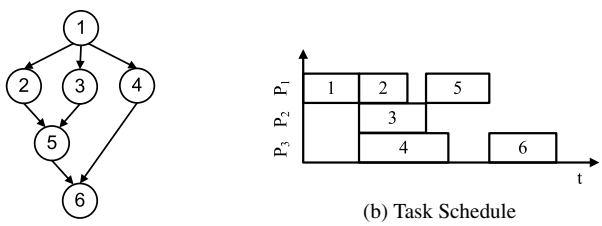
3. DESIGN-STAGE TASK ALLOCATION AND SCHEDULING

We propose to handle Problem 1 with a simulated annealing-based algorithm. Since it is impossible to generate task schedules for each individual chip, the optimization objective is the *expected* energy consumption over all the products.

3.1 Solution Representation and Moves

An effective solution representation together with the corresponding moves is very important for any SA-based technique. To the best of our knowledge, all existing techniques (e.g., [14]) do not have a 1-to-1 correspondence between the representation and the task schedule. Consider the one proposed in [8], wherein a schedule is represented as (*scheduling order sequence*; *resource assignment sequence*) together with three kinds of moves. While it has been proved that all possible task schedules are reachable starting from an arbitrary initial valid one [8], this representation suffers from redundancy problem. To clarify, we consider the task graph shown in Fig. 3(a). Based on the solution representation (1, 2, 3, 4, 5, 6; P_1, P_1, P_2, P_3, P_1, P_3), which means task 1 is scheduled on P_1 first, following by task 2, 3, 4, 5, and 6, on P_1, P_2, P_3, P_1, and P_3 respectively, we can reconstruct the unique task schedule, demonstrated in Fig. 3(b). However, some solutions, such as (1, 3, 2, 4, 5, 6; P_1, P_1, P_2, P_3, P_1, P_3) and (1, 4, 3, 2, 5, 6; P_1, P_1, P_2, P_3, P_1, P_3), correspond to the same schedule. In other words, some moves cannot lead to new schedules, which can result in significantly adverse impact on the efficiency of searching process.

The above motivates us to propose a novel solution representation, which has 1-1 correspondence property between task schedules and solution representations. Given a solution representation, if swapping any two tasks in a subsequence of the scheduling order sequence does not change the schedule, the tasks in the subsequence are referred to as *interchangeable*. In other words, swapping the order of these tasks in the



(a) Task Graph (b) Task Schedule

Figure 3: An Example of Solution Representation.

[1]The mode execution probabilities can be estimated as in [7, 16].

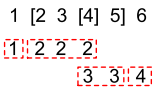


Figure 4: Zone Representation.

scheduling order sequence cannot result in a new schedule. In our example, given the solution (1, 2, 3, 4, 5, 6; P_1, P_1, P_2, P_3, P_1, P_3), tasks 2, 3, and 4 form such a subsequence. In addition, tasks 4 and 5 form another one. The task allocation and scheduling for the MPSoC is therefore represented as (*scheduling order sequence*; *resource assignment sequence*), and the interchangeable tasks are marked with square bracket. In addition, in case that two adjacent tasks in the scheduling order sequence are interchangeable, we always keep the one with smaller index in the front. For example, the schedule shown in Fig. 3(b) is represented as (1, [2, 3, [4], 5], 6; P_1, P_1, P_2, P_3, P_1, P_3). Clearly, a solution representation is regarded as a *valid* one if (i). the scheduling order sequence conforms to the partial order designated by the task graph and (ii). if two adjacent tasks are interchangeable, the one with smaller index is kept in the front.

Before the definition of the moves, we need to introduce a few concepts. With the above representation, we define *zones* in the scheduling order sequence. The tasks in the scheduling order sequence are swept from left to right. A new zone appears when we reach a task that does not belong to any subsequence or we reach a new subsequence, and ends when we move to the next task in the first case or the subsequence terminates in the second case. An example is demonstrated in Fig. 4, where four zones are identified. The tasks are labeled with the zone indexes. Note that, in case that a task belongs to two zones, we always label it with the smaller index. In our example, task 4 belongs to two zones and is labeled as a task in zone 2. To change the scheduling order sequence, we can randomly pick up a task and refer to it as *source task*. Given the source task is in zone i, if there exists a task just precedent to zone i in the scheduling order sequence, it is referred to as the *sink task* with respect to the source. For example, suppose task 3 is selected as the source, it belongs to zone 2 and hence its sink is task 1. Given the source task 4, although it belongs to two zones, it is referred to as a task in zone 2 and therefore its sink is task 1. Similarly, the source task 5 corresponds to the sink task 3.

Now we are ready to introduce the moves that guarantee the completeness of searching process. Two kinds of moves are defined:

- M1: Pick up a task in the scheduling order sequence as the source and insert it in the front of its sink, if there is no precedence constraint between the source and sink tasks.
- M2: Change the resource assignment of a task.

To move to a neighbor solution, we start with the original solution and randomly apply a move defined above. Sequentially, we need to conduct a post-processing step to identify the interchangeable subsequences. This is easily achievable because two adjacent tasks in the scheduling order sequence are interchangeable if and only if (i). there is no precedence constraint between them and (ii). they are assigned onto different processor cores. In addition, we move the tasks with small indexes forward as much as possible. The proof of completeness and 1-1 correspondence are ignored in this paper due to space limitation.

To reconstruct the schedule with a given solution representation, we assign the tasks according to the scheduling order sequence iteratively. Every time the i^{th} task in the scheduling order sequence whose index is j is picked up and its core assignment is identified by the j^{th} element in the resource binding sequence. We simply set its starting time as the earliest available time.

3.2 Cost Function

Since the objective is to minimize the expected energy consumption under the performance and reliability constraints, the cost function used in our SA-based technique consists of three terms, each for an objective or a constraint.

$$\text{Cost} = \mathbf{E}_{\mathbf{Y}}[\text{Energy}] + \mu \cdot \mathbf{I}[\exists i,k : e_i^k > d^k] + \mu \cdot \mathbf{I}[R^{sys}(t_L) < \eta\%] \quad (1)$$

where $\mathbf{E}_{\mathbf{Y}}$ indicates the expectation over usage strategy distribution $\mathbf{Y}$, μ is a significantly large number, and the indicator function $\mathbf{I}[A]$ equals to 1 if event A is true while 0 otherwise. Thus, the large cost μ is the penalty of violating constraints.

In case that no DVFS technique is enabled, the computation of energy consumption in an execution mode k is a trivial problem. We can simply sum up the energy consumption of all tasks, namely,

$$\text{Energy}_k = \sum_{i=1}^{n^k} \sum_{j=1}^{m} \mathbf{I}[i \to j] \cdot c_{k,i,j} \cdot p_{k,i,j} \quad (2)$$

Thus, given the usage strategy $\mathbf{y}$, the energy consumption of a system can be expressed as

$$\text{Energy} = \sum_{k=1}^{q} \sum_{i=1}^{n^k} \sum_{j=1}^{m} \mathbf{I}[i \to j] \cdot c_{k,i,j} \cdot p_{k,i,j} \cdot y_k \quad (3)$$

Nevertheless, it is difficult to compute the expected energy consumption $\mathbf{E}_{\mathbf{Y}}[\text{Energy}]$ according to its closed-form expression, i.e.,

$$\mathbf{E}_{\mathbf{Y}}[\text{Energy}] = \int \cdots \int_{\mathbf{Y}} \text{Energy} \cdot f_{\mathbf{Y}}(\mathbf{y}) d\mathbf{y} \quad (4)$$

In this work, we propose to discretize the range of y_k (i.e., [0,1]) into partitions of u equal intervals. As a consequence, since the state space of $\mathbf{y}$ is q-dimensional, there are u^q partitions in total. We compute the energy consumption with respect to a sample usage strategy with Eq. (3) for each partition and sequentially approximate the expected value with the weighted sum.

Given the task schedule, the second term in Eq. (1) can be determined by comparing the finish time e_i^k and deadline d_i^k of tasks and it is independent of usage strategy.

We then move to discuss the third term. We assume the reliability function of a processor core follows Weibull distribution [8], where the shape parameter β reflects the structural property of processor core while the scale parameter θ indicates the rate of suffering from aging effect and hence depends on operational temperature, supply voltage, and frequency.

$$R(t) = \exp\big(-(\frac{t}{\theta})^{\beta}\big) \quad (5)$$

Let $\Omega(s^k)$ be the aging rate[2] of the processor in execution mode k [8], induced by the task schedule s^k. With the similar argument in [6], we can express the expected aging rate as

$$\Omega(\mathbf{s}) = \sum_{k=1}^{q} \Omega(s^k) \cdot y^k \quad (6)$$

and approximate the reliability function as

$$R^{sys}(t_L) = \exp\big(-\sum_{j=1}^{m} (\frac{t_L}{\Omega(\mathbf{s})})^{\beta_j}\big) \quad (7)$$

Similar to the expected energy consumption, $R^{sys}(t_L)$ is a function of usage strategy. We therefore use the approximated value obtained by discretization for cost evaluation.

3.3 Impact of DVFS

For DVFS-enabled MPSoC products, we need to revise our cost function. Let us define the voltage scaling parameter ρ as a value within the range $[0,1]$, where $\rho = 1$ implies the processor is working under maximum supply voltage. The scaled supply voltage is therefore ρV_{dd}.

Without DVFS, the power dissipation of a processor core can be described by the following equation [11]

$$\begin{aligned} p &= P_{\text{on}} + P_{\text{dynamic}} + P_{\text{leakage}} \\ &= P_{\text{on}} + C_{\text{eff}} \cdot V_{\text{dd}}^2 \cdot f + (V_{\text{dd}} \cdot I_{\text{subn}} + |V_{\text{bs}}| \cdot I_{\text{j}}) \end{aligned} \quad (8)$$

where, P_{on} is the inherent power cost for keeping the processor on, which can be assumed as a constant value [11]; P_{dynamic} represents the dynamic power consumption, which is quadratic dependent on supply voltage V_{dd} and proportional to operating frequency f and the effective switching capacitance C_{eff}; P_{leakage} indicates the leakage power dissipation due to subthreshold leakage current I_{subn} and reverse bias junction

[2]This value indicates the rate of a processor core suffering from aging effect.

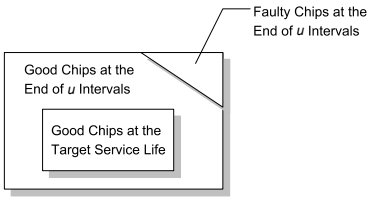


Figure 5: Conditional Reliability.

current $I_{\rm j}$. $V_{\rm bs}$ is the body bias voltage. $I_{\rm subn}$ is a voltage-dependent parameter, given by

$$I_{\rm subn} = K_1 e^{K_2 V_{\rm dd}} \tag{9}$$

where, K_1 and K_2 are constant fitting parameters. $I_{\rm j}$ can be approximated as a constant value, as pointed out in [15].

The operational frequency of a processor core is bounded by its slowest path, whose delay can be expressed in terms of supply voltage $V_{\rm dd}$, threshold voltage $V_{\rm th}$, and logic depth of the critical path $L_{\rm d}$ [15],

$$\tau = \frac{L_{\rm d} K_3}{(V_{\rm dd} - V_{\rm th})^{\kappa}} \tag{10}$$

where, κ is a material-related constant. K_3 is a constant related to process technology. Given the voltage scaling parameter ρ, substituting the scaled supply voltage $\rho V_{\rm dd}$ into Eq. (10) yields the scaling operating frequency, which is inversely proportional to circuit delay, as

$$f(\rho) = \frac{1}{\tau(\rho)} = \frac{(\rho V_{\rm dd} - V_{\rm th})^{\kappa}}{L_{\rm d} K_3} \tag{11}$$

The power consumption of a task (that is, Eq. (8)) with voltage scaling parameter ρ is then redefined as

$$p(\rho) = C_{\rm eff} \cdot \rho^2 V_{\rm dd}^2 \cdot f(\rho) + \rho V_{\rm dd} \cdot K_1 e^{K_2 \cdot \rho V_{\rm dd}} + |V_{\rm bs}| \cdot I_{\rm j} + P_{\rm on} \tag{12}$$

It can be also normalized by calculating the effective switching capacitance $C_{\rm eff}$ with the condition that $p(\rho)|_{\rho=1} \equiv p$. The material and technology-related parameters in this model (e.g., K_1) are set according to [15] with 70nm technology.

Since DVFS provides benefits to both energy saving and reliability enhancement, for a given task schedule we reduce its ρ as much as possible. To be specific, we extend the execution time of a task to $c \cdot d^k$/makespan, where makespan is the time interval that all periodical tasks need to finish their execution once. By the normalization that $c(\rho)|_{\rho=1} \equiv c$, the execution time with DVFS can be expressed as

$$c(\rho) = \frac{(V_{\rm dd} - V_{\rm th})^{\kappa}}{(\rho V_{\rm dd} - V_{\rm th})^{\kappa}} \cdot c \tag{13}$$

We combine this equation with the following one to compute ρ.

$$c(\rho) = \frac{c \cdot d^k}{\text{makespan}} \tag{14}$$

These parameters $p(\rho)$ and $c(\rho)$ are substituted into the cost function presented in Section 3.2.

4. ONLINE ADJUSTMENT

Given an MPSoC product, its usage strategy becomes available at runtime, which enables us to online adjust the task schedules. Since the aging effect of IC product is a slow process (in the range of years), we propose to perform the task schedule adjustment at regular intervals and each interval can be in range of days or months. From this aspect, the online adjustment can be regarded as a special task of the embedded system and executed regularly.

Since the online adjustment targets a specific MPSoC product, we need to calculate the energy consumption and reliability with respect to its usage strategy rather than the expected values over all products. While a lifetime reliability model for MPSoC embedded system has been proposed in [8], it is not readily applicable for online lifetime reliability evaluation. This is because, it cannot describe the design-stage reliability requirement given a certain product survives until time t without any information on other products. With the information of task schedules and usage strategy in the past u intervals, we are interested in the *conditional*

label	task #	edge #	in-degree mean/std	out-degree mean/std	task # on longest path
$\mathcal{G}_a$	6	7	1.17 / 0.75	1.17 / 0.98	4
$\mathcal{G}_b$	7	8	1.14 / 0.69	1.14 / 0.90	4
$\mathcal{G}_c$	3	2	0.67 / 0.58	0.67 / 1.15	2
$\mathcal{G}_d$	31	40	1.29 / 0.81	1.29 / 1.17	10
$\mathcal{G}_e$	69	100	1.45 / 0.99	1.45 / 1.17	13
$\mathcal{G}_f$	152	233	1.53 / 0.79	1.53 / 0.74	28

Table 1: Description of Task Graphs.

label	processor cores		
	type	quantity	β
$\mathcal{P}_1$	v_1, v_2	1,2	2,1.5
$\mathcal{P}_2$	v_1, v_2, v_3	1,1,4	2.5,2,1.5

Table 2: Description of MPSoCs.

reliability at the target service life t_L for the online decision making. Naturally, $t_L > u \cdot t_I$, because no adjustment is required after the target service life.

Given the product has been utilized for u intervals, the estimation of the reliability at time t_L depends on not only the past history but also the prediction for the future. Since the usage strategy for the entire service life is not available during usage, we need to infer the future usage strategy (denoted by $\mathbf{y}$) from the traced values. We suggest a forgetful scheme to highlight the recent usage preference: $\mathbf{y}$ is initialized as $\mathbf{y}_1$ at the end of the first interval. Since then, at the end of interval ℓ it is updated to $(1-\alpha) \cdot \mathbf{y}_\ell + \alpha \cdot \mathbf{y}$, where α is a small number. In other words, $\mathbf{y}$ has the form

$$\mathbf{y} = (1-\alpha) \cdot \mathbf{y}_u + (1-\alpha) \cdot \alpha \cdot \mathbf{y}_{u-1} + \cdots + \alpha^{u-1} \mathbf{y}_1 \tag{15}$$

In addition, we assume the newly-generated schedules $\mathbf{s}$ is always used in the rest of service life. Denoting by $\mathbf{s}_\ell$ the task schedules in the $\ell^{\rm th}$ interval, we represent the reliability requirement for online adjustment as

$$R^{sys}(t_L; \mathbf{y}, \mathbf{s} | \mathbf{y}_1, \mathbf{s}_1; \cdots; \mathbf{y}_u, \mathbf{s}_u) \geq \eta\% \tag{16}$$

Although we conduct the $u^{\rm th}$ online adjustment for the products that survive at time $u \cdot t_I$ only, the above requirement is consistent with the reliability constraint specified in Problem 1. To clarify, we denote by $R^{sys}(u \cdot t_I | \mathbf{y}_1, \mathbf{s}_1; \cdots; \mathbf{y}_u, \mathbf{s}_u)$ the probability of a product surviving at time $u \cdot t_I$. Given N products with identical usage strategy and task schedules where N is a sufficient large number, the quantity of faulty products at time $u \cdot t_I$ is given by $N \cdot (1 - R^{sys}(u \cdot t_I | \mathbf{y}_1, \mathbf{s}_1; \cdots; \mathbf{y}_u, \mathbf{s}_u))$, as shown in Fig. 5. Thus, to meet the reliability requirement $R^{sys}(t_L) \geq \eta\%$, the conditional reliability of a product at the target service life given its survival at time $u \cdot t_I$ should be no less than the ratio between good chips at the target service life and that at the end of u intervals, i.e.,

$$R_c(t_L | u \cdot t_I) \geq \frac{N \cdot \eta\%}{N \cdot R^{sys}(u \cdot t_I | \mathbf{y}_1, \mathbf{s}_1; \cdots; \mathbf{y}_u, \mathbf{s}_u)} \tag{17}$$

By definition, the conditional reliability is given by

$$R_c(t_L | u \cdot t_I) = \frac{R^{sys}(t_L; \mathbf{y}, \mathbf{s} | \mathbf{y}_1, \mathbf{s}_1; \cdots; \mathbf{y}_u, \mathbf{s}_u)}{R^{sys}(u \cdot t_I | \mathbf{y}_1, \mathbf{s}_1; \cdots; \mathbf{y}_u, \mathbf{s}_u)} \tag{18}$$

Substituting Eq. (18) into Eq. (17) results in the reliability requirement for online adjustment, namely, Eq. (16).

With the above definition, we then move to the analytical modeling of the quantity $R^{sys}(t_L; \mathbf{y}, \mathbf{s} | \mathbf{y}_1, \mathbf{s}_1; \cdots; \mathbf{y}_u, \mathbf{s}_u)$ in the following.

We use subscript to indicate the intervals and rewrite Eq. (6) as

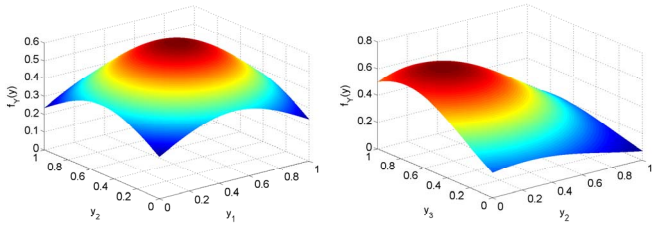
$$\Omega(\mathbf{s}_1) = \sum_{k=1}^{q} \Omega(s_1^k) \cdot y_1^k \tag{19}$$

The reliability at the end of the first interval is then given by

$$R(t_I | \mathbf{y}_1, \mathbf{s}_1) = \exp\big(-(\frac{t_I}{\Omega(\mathbf{s}_1)})^{\beta}\big) \tag{20}$$

By the continuity of reliability function, we obtain the reliability at time $u \cdot t_I$, namely,

$$R(u \cdot t_I | \mathbf{y}_1, \mathbf{s}_1; \cdots; \mathbf{y}_u, \mathbf{s}_u) = \exp\big(-(\sum_{\ell=1}^{u} \frac{t_I}{\Omega(\mathbf{s}_\ell)})^{\beta}\big) \tag{21}$$



(a) $\mathcal{U}_{\mathrm{I}}$ (b) $\mathcal{U}_{\mathrm{II}}$

Figure 6: Description of Usage Strategies.

By the similar argument, we can estimate the reliability at the target service life by

$$R(t_L;\mathbf{y},\mathbf{s}|\mathbf{y}_1,\mathbf{s}_1;\cdots;\mathbf{y}_u,\mathbf{s}_u) = \exp\big(-(\frac{t_L-u\cdot t_I}{\Omega(\mathbf{s})}+\sum_{\ell=1}^{u}\frac{t_I}{\Omega(\mathbf{s}_\ell)})^{\beta}\big) \quad (22)$$

where, $\Omega(\mathbf{s})$ is the future aging rate, induced by task schedules $\mathbf{s}$.

Without redundant cores, the entire system functions if all processor cores are functioning. To differentiate the embedded cores, we use the subscript to represent the core index. For example, $R_j(t)$ is the reliability function of core j, and $\Omega_j(\mathbf{s}_\ell)$ is the expected aging rate of processor P_j in the ℓ^{th} interval. The system reliability is therefore given by

$$R^{sys}(t_L;\mathbf{y},\mathbf{s}|\mathbf{y}_1,\mathbf{s}_1;\cdots;\mathbf{y}_u,\mathbf{s}_u) = \exp\big(-\sum_{j=1}^{m}(\frac{t_L-u\cdot t_I}{\Omega_j(\mathbf{s})}+\sum_{\ell=1}^{u}\frac{t_I}{\Omega_j(\mathbf{s}_\ell)})^{\beta_j}\big) \quad (23)$$

Finally, we obtain the new cost function for online adjustment as,

$$\begin{aligned}\text{Cost} \;=\; & \mathbf{E}_{\mathbf{y}}[\text{Energy}]+\mu\cdot\mathbf{I}[\exists i,k: e_i^k > d^k] \\ & +\mu\cdot\mathbf{I}[R^{sys}(t_L;\mathbf{y},\mathbf{s}|\mathbf{y}_1,\mathbf{s}_1;\cdots;\mathbf{y}_u,\mathbf{s}_u)\geq\eta\%] \end{aligned} \quad (24)$$

and we can then resort to the similar technique presented in Section 3.1 to solve Problem 2.

5. EXPERIMENTAL RESULTS

5.1 Experimental Setup

To demonstrate the effectiveness of the proposed technique, we setup our experiments as follows. The task graphs are generated by TGFF [3], whose attributes are described in Table 1. These task graphs are scheduled on hypothetic MPSoCs with features shown in Table 2. The power consumption values are randomly generated while the range is set according to IBM PowerPC 750CL [9]. Although the proposed analytical model is applicable for any failure mechanisms or their combinations, because of the lack of public data on the relative weights of different failure mechanisms, we consider a widely-used electromigration model [5] in our experiments.

In addition, we assume most variables y_i in the joint probability density functions that used to describe the usage strategies follow truncated Gaussian distribution, as shown in Fig. 6. In Fig. 6(a), $y_1 \sim N(0.5,0.5^2)$ and $y_2 \sim N(0.5,0.5^2)$. The conditional domain is set to $\mathbb{D} = \{0 \leq y_1 \leq 1, 0 \leq y_2 \leq 1, 0 \leq y_1+y_2 \leq 1\}$ and $y_3 = 1-y_1-y_2$. In Fig. 6(b), the probabilities of the product being in execution modes follow $y_2 \sim N(0.25,0.5^2)$ and $y_3 \sim N(0.75,0.5^2)$, and $y_1 = 1-y_2-y_3$. In addition, we assume two usage strategies for the MPSoCs with two execution modes: (i). $\mathcal{U}_{\mathrm{I}}^{\mathrm{s}}$: $y_1 \sim N(0.5,0.5^2)$ and $y_2 = 1-y_1$ and (ii). $\mathcal{U}_{\mathrm{II}}^{\mathrm{s}}$: $y_2 \sim N(0.25,0.5^2)$ and $y_1 = 1-y_2$, provided $0 \leq y_1 \leq 1$ and $0 \leq y_2 \leq 1$. For each case of our experiments, 1,000 sample products are generated, whose usage strategies are independent identically distributed samples of $f_{\mathbf{Y}}(\mathbf{y})$.

5.2 Results and Discussion

We first evaluate the effectiveness of the proposed method, given every task can be allocated onto any processor core. As shown in Table 3, we compare the solutions obtained by using the proposed design-stage and online strategies. In this table, $\mathbf{E}_{\mathbf{Y}}[\text{Energy}]$ (Column 2) is the estimation value and available at design stage, indicating the expected value of energy consumption of all products. Column 3-5 are obtained by applying the initial TAS solution on sample products. To be specific, C_N^{init} and C_M^{init} (Column 3 and 4) represent the sets of sample products that cannot meet reliability requirements and those that can, respectively. The expected energy consumption over the products that meet reliability requirement with initial solutions is denoted by $\mathbf{E}_{C_M^{\text{init}}}[\text{Energy}]$ (see Column 5). Columns 6-9 are obtained by applying online adjustment on the sample products. In particular, for the products that meet reliability constraints with initial solution, we evaluate their average energy by applying online adjustment and demonstrate the results in Column 8. We also report the expected energy consumption of the products that meet reliability requirements with online adjustment in Column 9.

As shown in this table, the online adjustment significantly enhances the lifetime reliability of sample products. The percentage of products that meet reliability requirement with online adjustment is much higher than that with initial solution. For example, in case of $\{\mathcal{G}_a,\mathcal{G}_b,\mathcal{G}_c\},\mathcal{P}_1,\mathcal{U}_{\mathrm{I}}$, only 45.5% chips are able to meet the reliability requirement that the probability of surviving until the target service life is no less than 40% by applying the initial solution. With online adjustment, by contrast, this value is improved to 98.8% (see Line 3). In average, the proposed online adjustment can lead to 30% more products meeting reliability requirements.

Moreover, the energy consumption of the reliable chips with initial solution (i.e., the chips in set C_N^{init}) can be reduced by using the online adjustment. To take a closer observation, we plot the measurements of all sample products in terms of energy consumption and lifetime reliability on Fig. 7, for the case $\{\mathcal{G}_a,\mathcal{G}_b,\mathcal{G}_c\},\mathcal{P}_1,\mathcal{U}_{\mathrm{I}}$. As demonstrated in this figure, with respect to the chips in set C_M^{init} the proposed online adjustment essentially trades reliability margin for energy reduction. As for the chips in set C_N^{init}, we achieve reliability enhancement by sacrificing some energy. As a result, the expected energy consumption of the products that meet reliability requirements with online adjustment (Column 9) is higher than the design stage estimation (Column 2).

We are also interested in the task allocation and scheduling problems within which some tasks must be allocated on a certain type of processor cores (i.e., task mapping flexibility constraints). In the experiments, we vary the percentage of tasks with such constraints and randomly select a set of tasks. The experimental results for the case $\{\mathcal{G}_a,\mathcal{G}_b,\mathcal{G}_c\},\mathcal{P}_1,\mathcal{U}_{\mathrm{I}}$ are depicted in Table 4.

The benefit provided by the proposed online adjustment gradually reduces when more tasks have the task mapping constraints. But still we achieve some reliability enhancement and energy saving. Consider the case that 75% tasks have mapping constraints (see the last line). The percentage of products that meet reliability requirement increases from 45.5% to 70.8%, and the improvement is as high as 25.3%. This is because, even if most tasks are allocated onto appointed core type, there still exists some flexibility induced by the cores assignment and scheduling order. Based on this observation, we can prepare codes for the tasks that significantly affect the lifetime reliability on various cores only while fixing the core assignment of the remaining tasks.

Finally, since both design-stage task allocation and scheduling and online adjustment are simulated annealing-based techniques, the execution time highly depends on its parameters. In our experiments, we set initial temperature, cooling rate, and end temperature to be 100, 0.8, end $10^(-3)$, respectively. The number of random moves at each temperature is 1000. With this setup, initial solution or task schedule adjustment for a chip requests a few seconds of execution time on a 2.13GHz Intel CPU.

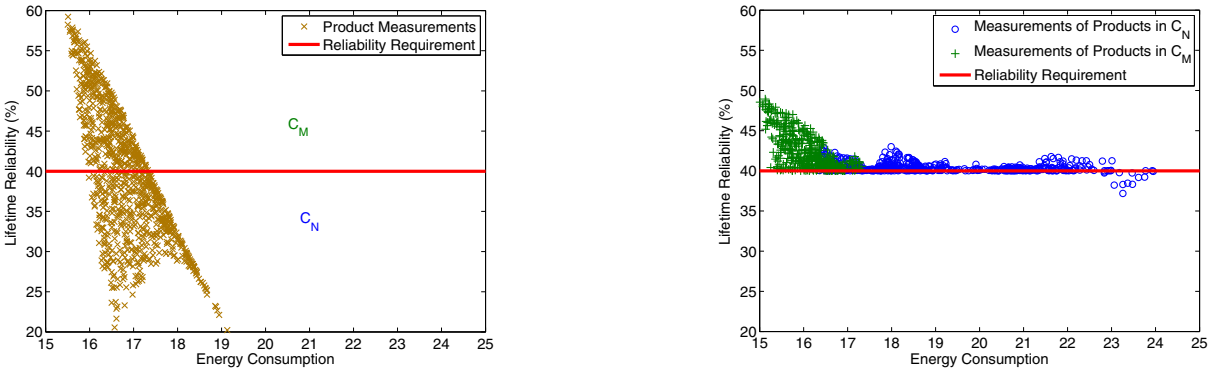
6. CONCLUSION

Because of the usage strategy deviation of modern multi-mode embedded systems, a unified task schedule for every execution mode generated at design-stage may not be reliable or energy-efficient for individual products. In this paper, we propose a novel customer-aware task allocation and scheduling technique to tackle this problem, wherein initial schedules are generated at design stage and each product is optimized separately with online adjustment at regular intervals for lifetime reliability and/or energy-efficiency improvement according to the specific usage strategy of individual products. Experimental results on several hypothetical MPSoCs with various task graphs show that the proposed solution is able to significantly increase the lifetime reliability of MPSoC products and is also helpful for energy reduction.

label	initial solution				online adjustment				ΔC_M (%)
	$\mathbf{E_Y}$[Energy]	C_N^{init} (%)	C_M^{init} (%)	$\mathbf{E}_{C_M^{\text{init}}}$[Energy]	C_N^{on} (%)	C_M^{on} (%)	$\mathbf{E}_{C_M^{\text{init}}}$[Energy]	$\mathbf{E}_{C_M^{\text{on}}}$[Energy]	
$\{\mathcal{G}_a, \mathcal{G}_b, \mathcal{G}_c\}, \mathcal{P}_1, \mathcal{U}_{\text{I}}$	**16.8221**	**54.5**	**45.5**	**16.3751**	**1.2**	**98.8**	**16.0905**	**17.6617**	**53.3**
$\{\mathcal{G}_a, \mathcal{G}_b, \mathcal{G}_c\}, \mathcal{P}_1, \mathcal{U}_{\text{II}}$	17.4577	61.6	38.4	16.8657	4.1	95.9	16.4147	18.5707	57.5
$\{\mathcal{G}_a, \mathcal{G}_b\}, \mathcal{P}_1, \mathcal{U}_{\text{I}}^{\text{s}}$	22.1729	60.3	39.7	22.4243	28.3	71.7	22.2263	22.5648	32.0
$\{\mathcal{G}_a, \mathcal{G}_b\}, \mathcal{P}_1, \mathcal{U}_{\text{II}}^{\text{s}}$	22.2991	32.5	67.5	22.5677	21.1	78.8	22.3834	22.5153	11.3
$\{\mathcal{G}_a, \mathcal{G}_b, \mathcal{G}_d\}, \mathcal{P}_1, \mathcal{U}_{\text{I}}$	19.5647	45.9	54.1	20.0546	14.4	85.6	19.3790	19.8103	31.5
$\{\mathcal{G}_a, \mathcal{G}_b, \mathcal{G}_d\}, \mathcal{P}_1, \mathcal{U}_{\text{II}}$	18.4024	47.4	52.6	18.4549	1.3	98.7	18.4124	18.6625	46.1
$\{\mathcal{G}_a, \mathcal{G}_e, \mathcal{G}_f\}, \mathcal{P}_2, \mathcal{U}_{\text{I}}$	20.9612	20.9	79.1	21.2033	0	100	21.0301	20.9117	20.9
$\{\mathcal{G}_a, \mathcal{G}_e, \mathcal{G}_f\}, \mathcal{P}_2, \mathcal{U}_{\text{II}}$	20.7217	15.3	84.7	20.8443	0	100	20.5244	20.5135	15.3

C_N^{init}: sample products that cannot meet reliability constraint with initial solution; C_M^{init}: sample products that meet reliability constraint with initial solution;
C_N^{on}: sample products that cannot meet reliability constraint with online adjustment; C_M^{on}: sample products that meet reliability constraint with online adjustment;
$\mathbf{E}_{C_M^{\text{init}}}$[Energy]: expected energy consumption of sample products that meet reliability constraint with initial solution;
$\mathbf{E}_{C_M^{\text{on}}}$[Energy]: expected energy consumption of sample products that meet reliability constraint with online adjustment;

Table 3: Effectiveness of The Proposed Technique.



(a) Initial Solution (b) Online Adjustment

Figure 7: Comparison of Initial Solution and Online Adjustment in Product Measurements.

initial solution				online adjustment					ΔC_M (%)
$\mathbf{E_Y}$[Energy]	C_N^{init} (%)	C_M^{init} (%)	$\mathbf{E}_{C_M^{\text{init}}}$[Energy]	tasks w. const (%)	C_N^{on} (%)	C_M^{on} (%)	$\mathbf{E}_{C_M^{\text{init}}}$[Energy]	$\mathbf{E}_{C_M^{\text{on}}}$[Energy]	
16.8221	54.5	45.5	16.3751	0	1.2	98.8	16.0905	17.6617	53.3
				25	16.9	83.1	16.1755	16.9475	37.6
				50	25.8	74.2	16.1815	16.6902	28.7
				75	**29.2**	**70.8**	**16.3702**	**16.7894**	**25.3**

Table 4: Effectiveness of The Proposed Strategy with Mapping Constraints.

7. ACKNOWLEDGEMENT

This work was supported in part by National Science Foundation of China (NSFC) under grant No. 60876029, in part by the General Research Fund CUHK417807 and CUHK418708 from Hong Kong SAR Research Grants Council (RGC), and in part by a grant N_CUHK417/08 from the NSFC/RGC Joint Research Scheme.

8. REFERENCES

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

[2] S. Borkar. Designing Reliable Systems from Unreliable Components: The Challenges of Transistor Variability and Degradation. IEEE Micro, 25(6):10–16, Nov.-Dec. 2005.

[3] R. P. Dick, D. L. Rhodes, and W. Wolf. TGFF: Task Graphs for Free. In *Proc. International Conference on Hardware/Software Codesign and System Synthesis (CODES)*, pp. 97–101, 1998.

[4] J. E. Gentle. In *Random Number Generation and Monte Carlo Methods*. Springer-Verlag, 2003.

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

[6] L. Huang and Q. Xu. AgeSim: A Simulation Framework for Evaluating the Lifetime Reliability of Processor-Based SoCs. In *Proc. IEEE/ACM Design, Automation, and Test in Europe (DATE)*, pp. 51–56, 2010.

[7] L. Huang and Q. Xu. Energy-Efficient Task Allocation and Scheduling for Multi-Mode MPSoCs under Lifetime Reliability Constraint. In *Proc. IEEE/ACM Design, Automation, and Test in Europe (DATE)*, pp. 1584–1589, 2010.

[8] L. Huang, F. Yuan, and Q. Xu. Lifetime Reliability-Aware Task Allocation and Scheduling for MPSoC Platforms. In *Proc. IEEE/ACM Design, Automation, and Test in Europe (DATE)*, pp. 51–56, 2009.

[9] IBM. IBM PowerPC 750CL Microprocessor Revision Level DD2.x. http://www-01.ibm.com/chips/techlib/techlib.nsf /techdocs/2F33B5691BBB8769872571D10065F7D5/$file /750cldd2x_ds_v2.4_pub_29May2007.pdf.

[10] T. Ishihara and H. Yasuura. Voltage Scheduling Problem for Dynamically Variable Voltage Processors. In *Proc. International Symposium on Low Power Electronics and Design (ISLPED)*, pp. 197–202, 1998.

[11] R. Jejurikar, C. Pereira, and R. Gupta. Leakage Aware Dynamic Voltage Scaling for Real-Time Embedded Systems. In *Proc. ACM/IEEE Design Automation Conference (DAC)*, pp. 275–280, 2004.

[12] N. K. Jha. Low Power System Scheduling and Synthesis. In *Proc. IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 259–263, 2001.

[13] E. Karl, D. Blaauw, D. Sylvester, and T. Mudge. Multi-Mechanism Reliability Modeling and Management in Dynamic Systems. *IEEE Transactions on VLSI Systems*, 16(4):476–487, April 2008.

[14] M. Lin and L. T. Yang. Genetics algorithms for scheduling real-time tasks onto multi-processors. In Y.-S. Dai, Y. Pan, and R. Raje, editors, *Distributed Computing: Evaluation, Improvement and Practice*, pp. 213–238. Nova Science Publishers, 2007.

[15] S. Martin, K. Flautner, T. Mudge, and D. Blaauw. Combined Dynamic Voltage Scaling and Adaptive Body Biasing for Lower Power Microprocessors under Dynamic Workloads. In *Proc. IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 721–725, 2002.

[16] M. T. Schmitz, B. M. Al-Hashimi, and P. Eles. Cosynthesis of Energy-Efficient Multimode Embedded Systems With Consideration of Mode-Execution Probabilities. *IEEE Transactions on Computer-Aided Design*, 24(2):153–169, February 2005.

[17] J. Srinivasan, S. V. Adve, P. Bose, and J. A. Rivers. The Case for Lifetime Reliability-Aware Microprocessors. In *Proc. International Symposium on Computer Architecture (ISCA)*, pp. 276–287, 2004.