Task Mapping for Redundant Multithreading in Multi-Cores with Reliability and Performance Heterogeneity

Kuan-Hsun Chen, *Student Member, IEEE*, Jian-Jia Chen, *Member, IEEE*, Florian Kriebel, Semeen Rehman, *Student Member, IEEE*, Muhammad Shafique, *Member, IEEE*, and Jörg Henkel, *Fellow, IEEE*

Abstract—Due to the architectural design, process variations and aging, individual cores in many-core systems exhibit heterogeneous performance. In many-core systems, a commonly adopted soft error mitigation technique is Redundant Multithreading (RMT) that achieves error detection and recovery through redundant thread execution on different cores for an application. However, *task mapping* and *the task execution mode* (i.e., whether a task executes in a reliable mode with RMT or unreliable mode without RMT) need to be considered for achieving resource-efficient reliability. This paper explores how to efficiently assign the tasks onto different cores with heterogeneous performance properties and determine the execution modes of tasks in order to achieve high reliability and satisfy the tolerance of timeliness. We demonstrate that the task mapping problem under heterogeneous performance can be solved by employing Hungarian Algorithm as subroutine to efficiently assign the tasks onto the cores to optimize the system reliability with polynomial time complexity. To obtain the efficient task execution modes, we also propose an iterative mode adaptation technique and guarantee the tolerable timing constraint. Our results illustrate that compared to state-of-the-art, the proposed approaches achieve up to 80 percent reliability improvement (on average 20 percent) under different scenarios of chip frequency variation maps.

1 Introduction

Continuous technology scaling has led to multiple reliability threats like soft errors, aging, and process variations [1], [2], [3], [4]. Soft errors are transient faults in the underlying hardware due to high-energy particle strikes, which can corrupt the correct application execution state [1]. Most of the hardware or software-level soft error mitigation techniques primarily rely on full-scale redundancy [2]. At the hardware level, vulnerability-aware adaptation of dual modular redundancy (DMR) and triple modular redundancy (TMR) have been conducted by exploiting extra hardware circuitry [3], [5]. However, these techniques inevitably incur significant area/power overhead, which may not be feasible under the stringent design constraints of embedded systems. At the software level, several techniques have been proposed as well: Reliability-driven transformation/compilation techniques [6], [7], selective instruction redundancy [8], [9], and timeliness-aware fault-tolerance [10].

Since modern mainstream processors are typically multi-cores [11], [12], [13], [14],[1] exploiting idle cores for task redundancy provides additional means for soft error mitigation. State-of-the-art techniques like [20], [21], [22] exploit idle cores to enable spatial and temporal redundancy. In particular, the *Redundant Multithreading (RMT)* techniques, like Intel's CRT – Chip-level Redundant Threading technology [23], execute redundant copies of a given task on different cores in parallel and perform error detection/recovery using comparison/voting on the threads' outputs. However, in modern many-core systems, the individual cores may exhibit different frequencies due to process variations [24], aging effects [25], and performance heterogeneous (micro-)architecture designs [26].

For instance, *process variations* may lead to significant frequency variations (e.g., up to 30 percent [24]). *Process variations* result from manufacturing-induced variability and imprecision, and manifest as chip-to-chip and core-to-core frequency variations. The impact of the process variation aggravates with scaling technology because, precisely manufacturing with reduced dimensions is very difficult [2]. Therefore, this has been envisaged as a critical reliability threat by industry [2], [27] and various research groups [28], [29].

Fig. 1 illustrates the core-to-core frequency variations of 28 percent at $1.2V$ and 59 percent at $0.8V$ for an Intel's

- *K.-H. Chen and J.-J. Chen are with the Department of Informatics, TU Dortmund (TUD), Germany. E-mail: kuan-hsun.chen@tu-dortmund.de, jian-jia.chen@cs.uni-dortmund.de.*
- *F. Kriebel, M. Shafique, and J. Henkel are with the Department of Informatics, Karlsruhe Institute of Technology (KIT), Germany. E-mail: kriebel@ira.uka.de, shafique@informatik.uni-karlsruhe.de, henkel@kit.edu.*
- *S. Rehman is a post-doctoral researcher in TU Dresden, Technical University Dresden. E-mail: semeen.rehman@tu-dresden.de.*

Manuscript received 29 July 2015; revised 18 Jan. 2016; accepted 15 Feb. 2016. Date of publication 28 Feb. 2016; date of current version 14 Oct. 2016. Recommended for acceptance by J. D. Bruguera. For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below. Digital Object Identifier no. 10.1109/TC.2016.2532862

1. Commercial examples are: Tilera chip with 100 cores [15], Intel's Xeon Phi [16] and SCC [17], Nvidia GPUs with $1{,}024$ processing elements [18]. Due to increasing core integration, emerging on-chip systems are envisaged to contain $1{,}000$ s of cores (according to the ITRS prediction: approximately $1{,}500$ cores by 2020 and $> 5{,}000$ cores in 2026) [19].

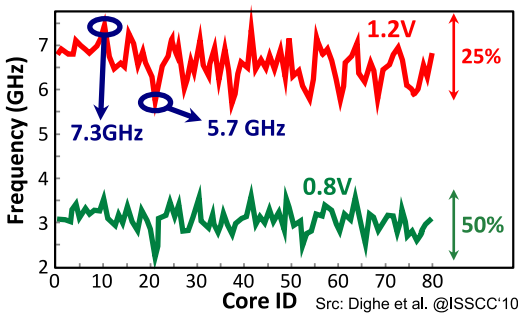


Fig. 1. Core-to-core variation [30].

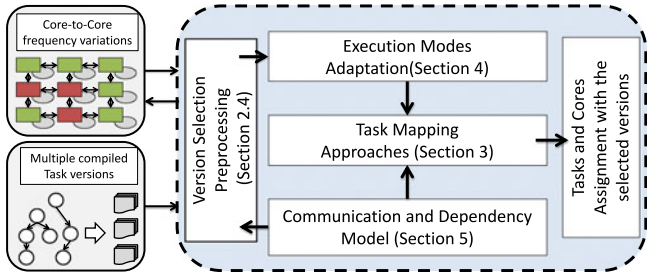


Fig. 2. Overview of our technique, illustrating interactions between different contributions for dependable application execution.

80-core test chip [30]. Aging further aggravates this issue by inducing frequency degradation at run-time.

To resolve such issues, guardbanding, voltage/frequency assignment and task scheduling at software level can be applied [31]. Typically, at design time, one of the standard industrial practices is to provide sufficient timing guardbands. However, this may lead to a significant frequency drop by a factor of Δf ($> 20\%$ over its lifetime) [2], [32], [33].[2] To alleviate this performance loss, current research trends try to relax these timing guardbands and salvage performance by exploiting the inherent resilience to such variability or by sophisticated run-time systems [28].

Another source of performance heterogeneity is iso-ISA performance-heterogeneous cores. That is, the architectural designer may integrate cores with different microarchitecture but having the same instruction set architecture (ISA). This results in diverse performance/power properties, i.e., different cores exhibit different frequencies in the system to accommodate the performance requirement with tolerable chip temperature or power consumption. Prominent examples are ARM big.LITTLE architecture [26] and Kumar et al. [36].

When applying the above-discussed state-of-the-art techniques under multiple reliability threats (i.e., soft errors, aging and process variations), these techniques may not be able to perform at their full effectiveness. *The interplay of multiple reliability threats and performance heterogeneity introduces non-trivial challenges for dependable execution*, since any approach which solely considers one specific aspect cannot guarantee effectiveness under the other threats.

The goal of this paper is to achieve resource-efficient soft-error resilient application execution in many-core systems under core-to-core frequency variations. The potential applications may have competing scenarios of concurrently executing, e.g., image recognition, data encryption, and secure video conferences, in which various thread instances have to process different sets of data. The key objective is to allocate the tasks by using RMT to improve the overall dependability, with respect to both timing and functional correctness while also accounting for application tasks with *multiple compiled versions*. Such multiple compiled versions with varying execution time and soft error vulnerabilities can be generated using reliability-aware compilers like [7], [37], and [38] exhibiting diverse performance and reliability properties. By adopting multiple compiled versions, redundant thread executions, and different frequencies of cores, we are able to fully utilize the reliability optimization space at both software and hardware-levels while exploring different area, execution time, and achieved reliability tradeoffs. The timeliness can be defined as the tolerated deadline miss rate, which is typically adopted as the quality of service (QoS) metric in many practical real-time applications.

In our previous work *dTune* [39], we have adopted a greedy mapping of reliability-critical tasks onto high-frequency cores. However, such an approach lacks effectiveness as the amount of high-frequency cores is limited. Besides the benefit of executing on high-frequency cores, the reliability degradation by executing on low-frequency cores also needs to be considered for all tasks at the same time. As shown in our motivational example (see Section 2.2), there may be scenarios where assigning the reliability-critical task to the fastest core is not reliability-wise beneficial, since the timeliness of the overall system also needs to be satisfied. As far as we know, dTune is the first approach adopting variation and aging aware RMT with reliability-aware task mapping in many-core systems. Comparing to reliability-aware task mapping works [40], [41], we consider the system reliability to be affected by the tasks' vulnerabilities and the frequencies of cores instead of the system life-time.

To achieve high reliability and timeliness on many-cores, in this paper we introduce reliability-driven task mapping means which involves the determination of the *task execution modes*, i.e., task execution with or without RMT, and *task allocation decisions*, i.e., mapping the (redundant) tasks on many-cores with heterogeneous performance characteristics. Fig. 2 illustrates an overview of our novel contributions, detailed as follows:

- We show that our studied problem can be reduced to the minimum weight perfect bipartite matching problem under two assumptions: 1) the execution modes are given, and 2) all the tasks are without communication latency. A well-known approach called Hungarian Algorithm can be applied to find out an optimal solution, when all execution modes are homogeneous. We show the optimality of the approach for heterogeneous cases if the reliability penalty of RMT task is negligible (See Section 3).
- If the execution modes are not given, we propose an iterative mode adaptation as a heuristic in combination with the proposed task mapping approaches, such that the execution mode of tasks can be determined under the polynominal time complexity (See Section 4).

2. According to [33], [34], [35], current processors typically incorporate guardbands for a device lifetime of 7-10 years considering both process variations and aging.

- We also present how to adopt a deterministic routing algorithm, e.g., XY routing, to estimate the upper bound of communication overhead on a common topology two-Dimension (2D) mesh, which can be utilized in our proposed approaches (See Section 5).
- To show the effectiveness of the proposed approaches, we compare to the existing *dTune* [39] as the state-of-the-art for different execution scenarios under various process variation maps on 8×8 cores (See Section 6).

2 Problem Definition

2.1 System Model

Hardware architecture and performance variations. We consider a many-core processor $\mathbb{C} = \{c_1, c_2, \ldots, c_M\}$ with M ISA-compatible RISC cores, which only has single thread per core. Each core c_i has its own instruction and data cache to execute tasks. Due to the performance variance, e.g., process variations [2], [42], [43], [44] and architectural design, each core c_i has its own frequency, denoted as f_i. For notational brevity, we index the M cores by a non-decreasing order of current frequencies, i.e., $f_{max} = f_1 \geq f_2 \ldots \geq f_M = f_{min}$.

Redundant multithreading. We consider RMT is provided by Chip-level Redundant Multithreading (CRT) [23], which executes the redundant threads of tasks in parallel on different cores, and activates Triple Modular Redundancy mode [5] for soft-error detection and recovery. In this paper, we only take TMR-based RMT into consideration, since the overhead of voting mechanism is negligible compared to the task's total re-execution time in a DMR-based rollback.

Application model. Consider that an application is given as a task graph $\mathrm{G} = (\Gamma, \mathbb{E})$, where Γ is a set of N nodes representing tasks, such that $\Gamma = \{\tau_1, \tau_2, \ldots, \tau_N\}$. $\mathbb{E}$ is the set of edges denoting task dependencies: $\mathbb{E} = \{e_{xy} | (\tau_x, \tau_y)\}$. Each task τ_i has K_i task versions: $\tau_i = \{\tau_{i,1}, \tau_{i,2}, \ldots, \tau_{i,K_i}\}$ generated by the reliability-aware compilation [7], [8]. Each task τ_i has its own relative deadline D_i for its all versions. Due to the *performance heterogeneity* among the cores, the *execution time* of task version $\tau_{i,k}$ depends on the core it is assigned to. We assume the given mapping function $C_{i,k,m}(e)$ denotes the continuous cumulative distribution function of execution time, in which the execution time of version $\tau_{i,k}$ is less than or equal to e when it is executed on core c_m. With this mapping function, the deadline miss rate for a version $\tau_{i,k}$ on core c_m can be estimated as Eq. (1),

$$P_{\mathrm{dm}}(\tau_{i,k}, c_m) = 1 - C_{i,k,m}(D_i). \tag{1}$$

In addition, the expected execution time of task version $\tau_{i,k}$ with the frequency of core c_m is assumed to be given as $E(\tau_{i,k,c_m})$, which can be calculated by the continuous cumulative distribution function $C_{i,k,m}(e)$ readily. For the guarantee of timeliness, we assume that the set of *tolerable* rate of deadline miss $\rho_\Gamma = \{\rho_1, \rho_2, \ldots, \rho_i\}$ can be given as a hard constraint. Each task must be guaranteed to have the probability of deadline miss rate lower than tolerable miss rate ρ_i. According to Eq. (1), we consider version $\tau_{i,k}$ is feasible on core c_m, if it's deadline miss rate $P_{\mathrm{dm}}(\tau_{i,k}, c_m)$ is not greater than the given miss rate constraint ρ_i, i.e., $P_{\mathrm{dm}}(\tau_{i,k}, c_m) \leq \rho_i$, in which $1 \leq k \leq K_i$. If there is a task mapping that all the tasks meet their tolerable miss rates with their feasible versions, we consider it as a feasible solution.

Quantification of task reliability. To quantify the task reliability, we assume a mapping function $\phi(\tau_{i,k}, c_m)$ can be given that indicates the reliability penalty of task version $\tau_{i,k}$ on core c_m. In this paper, we set this reliability penalty function as the probability that a fault during the execution of version $\tau_{i,k}$ leads to a visible error when executing on core c_m, with the probability of failure for each instruction estimated using the so-called Instruction Vulnerability Index [7], [45]. As the task vulnerability can be characterized/estimated by the composition of the instructions, the task version which has the lower vulnerability is assumed to provide a better reliability penalty than the one with the higher vulnerability on the same core. Conversely, as we can expect that transient faults may happen more within the longer execution time, for the same task version $\tau_{i,k}$, executing on a higher frequency core is assumed to have a better/lower reliability penalty than executing on the lower one. This penalty function can be set to any possible reliability penalty index as defined in the work [39].

To quantify the overall dependability, in this paper the objective function is defined in the following:

Definition 1. *The overall reliability penalty of task set Γ, denoted by ϕ_Γ, is the summation penalty of tasks given by $\phi_\Gamma = \sum_{\tau_i \in \Gamma} \phi(\tau_{i,k}, c_m)$ under the miss rate constraint ρ, where $\phi(\tau_{i,k}, c_m)$ is the reliability penalty of task version $\tau_{i,k}$ executing on core c_m.*

Task execution modes. Each task τ_i has an *execution mode* denoted by $\lambda_i = \{\mathrm{NR}, \mathrm{TMR}\}$, where NR denotes that task τ_i is executed without RMT; TMR denotes TMR-based RMT is activated for task τ_i. To execute a task in TMR-based RMT, the task requires three cores to provide the majority-voting mechanism. Let $\mathbb{G}_i$ be a subset of $\mathbb{C}$. If $\mathbb{G}_i$ has three elements, it is eligible for the task with TMR-based RMT, which is called the *core group* for brevity. For the task set Γ, we use a vector $\vec{\lambda} = (\lambda_1, \lambda_2, \ldots, \lambda_N)$ to denote the execution decision of tasks. $\delta_{\vec{\lambda}}$ denotes the demand of cores for satisfying all the tasks in $\vec{\lambda}$, which is equal to the number of NR tasks plus three times the number of TMR tasks. For simplicity, Γ_{TMR} and Γ_{NR} denote the set of tasks which are executed in TMR mode or NR mode, respectively.

Observation of TMR-based RMT. Due to the advantage of majority-voting, the reliability penalty of TMR-based RMT task τ_i with version $\tau_{i,k}$ on core c_m can be assumed to be a negligible value ϵ, i.e., $\phi(\tau_{i,k}, c_m) = \epsilon$, where $\epsilon \geq 0$. However, the majority-voting mechanism has to wait for all redundant threads completing their jobs. As shown in Fig. 3, we can observe that the fact that the slowest thread on core C_3 will dominate the execution time of task. If the redundant thread on core C_3 spends too much execution time, it may lead to the violation of deadline constraint. Therefore, we can safely estimate the execution time of task in RMT mode by the execution time of its redundant thread on the lowest-frequency core of assigned core group.

2.2 Motivational Example

In this section, we provide a motivational example to explain why the greedy mapping is not good enough for the task

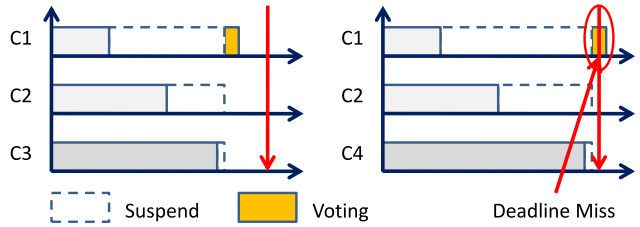


Fig. 3. Example of TMR-based RMT. As the voter has to wait for the slowest thread, the task has potential to miss deadline on the low frequency core, where $f_1 > f_2 > f_3 \gg f_4$.

mapping. For simplicity, we only present the motivational example for tasks by *single version* with given execution modes. Suppose that we are given three tasks, i.e., τ_1, τ_2, and τ_3. Task τ_1 has TMR-based RMT requirement but the others have no redundancy. Five cores are sorted by their frequencies beforehand. Now we consider the task mapping problem to allocate the cores to three tasks for minimizing the total penalty, where the reliability penalty of tasks on each core are defined in Table 1a. As shown in Fig. 3, the penalty value of Table 1a for τ_1 depends on the frequency of lowest-frequency core in the assigned core group. Please note, we denote the penalty value as ∞ to show the infeasible mapping that violates the tolerance of deadline miss rate. With the above setting, there are four visible assignments with different penalty value as illustrated in Table 1 b.

In the above example, we can check all the possible mappings to obtain the optimal result that will be 0.42 while the miss rate constraint is not violated. In this example, τ_1 cannot adopt core c_5 for RMT activation, since the tolerable miss rate will be violated. By using the greedy mapping to assign the tasks and cores, the total penalty of mappings is 0.6. RMT-activated task τ_1 uses core group $\{c_1, c_2, c_3\}$ for the minimal communication overhead, and the reliability-wise critical task, i.e., τ_3, acquires the higher-frequency core c_4 among the rest of cores. Moreover, if the allocation of TMR-based RMT task τ_1 is not assigned properly, the total penalty of mappings may be even worse in this example, i.e., ∞.

In the above example, we can observe that the greedy mapping strategy is not good enough, since the reliability-wise critical task is a suboptimal choice without considering the total benefit of system. Moreover, the miss rate constraint for each task should be considered to ensure the feasibility of task mapping as well. As a consequence, it is clear that such a task mapping problem requires better strategies, whereas the straightforward exhaustive search is obviously not feasible in practice with the expected high time complexity. It is not difficult to see that, if the reliability penalty of task increases non-linearly, it may lead to a result which is even worse than the proportional setting of this example.

2.3 Problem Definition

As shown in the motivational example, even if all the tasks have a single version, the task mapping problem still requires a comprehensive strategy for the heterogeneity of reliability and performance, in which some tasks are prohibited to execute on the low-frequency cores due to their deadline miss rate constraints. With the limited amount of redundant cores, we have to consider how to decide the execution modes for each task, i.e., NR or RMT, as different sets

TABLE 1
Individual Penalty Value and the Possible Mappings

(a) Tasks reliability penalty on each core

Penalty value	c_1	c_2	c_3	c_4	c_5
TMR $-\ \tau_1$	ϵ	ϵ	ϵ	ϵ	∞
NR $-\ \tau_2$	0.10	0.15	0.20	0.25	0.30
NR $-\ \tau_3$	0.24	0.26	0.28	0.30	0.32

(b) The possible mappings and the total penalty

Mappings	Total penalty
$\tau_1 \rightarrow \mathrm{TMR}(c_3, c_4, c_5), \tau_2 \rightarrow \mathrm{NR}(c_2), \tau_3 \rightarrow \mathrm{NR}(c_1)$	∞
$\tau_1 \rightarrow \mathrm{TMR}(c_1, c_2, c_3), \tau_2 \rightarrow \mathrm{NR}(c_5), \tau_3 \rightarrow \mathrm{NR}(c_4)$	0.6
$\tau_1 \rightarrow \mathrm{TMR}(c_2, c_3, c_4), \tau_2 \rightarrow \mathrm{NR}(c_5), \tau_3 \rightarrow \mathrm{NR}(c_1)$	0.54
$\tau_1 \rightarrow \mathrm{TMR}(c_2, c_3, c_4), \tau_2 \rightarrow \mathrm{NR}(c_1), \tau_3 \rightarrow \mathrm{NR}(c_5)$	0.42

of execution modes may lead to different inputs for task mapping problem.

Assume we are given a many-core processor $\mathbb{C}$ with M ISA-compatible homogeneous RISC cores, and a set of tasks Γ with multiple versions. Without loss of generality, the number of cores M must be greater than or equal to the number of tasks N. The studied problem can be divided into two sub-problems:

- *Task mapping:* Given the execution modes $\vec{\lambda}$ and the tolerable miss rate constraints ρ_Γ, we consider how to *select the executing version $\tau_{i,k}$ and allocate the cores with corresponding frequency for each task τ_i, so that the overall reliability penalty ϕ_Γ is minimized*. For this sub-problem, we classify the given execution modes into two classes and propose two algorithms to minimize the overall reliability penalty in Section 3.
- *Execution modes adaptation:* The objective is to *determine the task execution modes $\vec{\lambda}$ without violating the deadline miss rate.* Without checking all the combinations, we propose an iterative mode adaptation to efficiently determine the execution modes of tasks with our mapping approaches so that the overall reliability penalty is minimized (See Section 4).

For the simplicity of presentation, the above approaches are presented with the assumption that there is no data dependencies and communication among the tasks. After addressing the studied problem ideally, we consider how to enhance our system model to incorporate the overhead of execution time for the data dependencies and communication in Section 5.

2.4 Preprocessing for Version Selection

In terms of task, the assigned core(s) and the executing version both have impacts on the reliability penalty and the deadline miss rate. Moreover, different execution modes will lead to different requirement of executing versions in terms of reliability. Suppose τ_i does not need TMR-based RMT and is assigned on core c_m, the best version of task should be a specific version with the minimal $\phi(\tau_{i,j}, c_m)$ while satisfying the tolerable miss rate ρ_i. In contrast, assume that τ_i requires TMR-based RMT and the lowest frequency core in the assigned core group is c_m. Due to the negligible penalty of TMR-based RMT, the best version of task $\tau'_{i,j,m}$ should be a performance-wise best version without violating the miss rate constraint ρ_i on core c_m.

Although we know the best versions of tasks depend upon their execution scenarios, which include the frequencies of assigned cores, the constraint of miss rates, and the vulnerability of tasks, however, we can identify the best task version for each core beforehand to mitigate the complexity of studied problem. In case if no task version can satisfy the miss rate constraint on core c_m, we know that it is not feasible to run the task on core c_m. As a result, we can build up a reference table ψ and record all the corresponding best versions $\tau'_{i,j,m}$ in each entry $\psi(\tau_i, c_m)$ as Eq. (2),

$$\tau'_{i,j,m} = \underset{1\le j\le K_i}{\arg} \begin{cases} \min \phi(\tau_{i,j}, c_m) & \text{if } \lambda_i \text{ is NR} \\ \min E(\tau_{i,j,c_m}) & \text{if } \lambda_i \text{ is TMR} \end{cases}, \quad (2)$$

in which the miss rate ρ_i should be satisfied, i.e., $P_{\mathrm{dm}}(\tau_{i,j}, c_m) \le \rho_i$. If τ_i executes on core c_m without RMT, we evaluate the penalty of each version $\tau_{i,j}$ and pick the version with the minimal reliability penalty $\phi(\tau_{i,j}, c_m)$. If τ_i requires TMR-based RMT, the performance-wise best version will be selected for increasing the resilience while meeting deadline. If there is no feasible version to select, we fill the entry with infinite reliability penalty to present the infeasibility.

3 Task Mapping Problem

In this section, we present the idea of our task mapping approaches under the assumption that the execution modes for all the tasks are already known beforehand, i.e., $\vec{\lambda}$ is given. For simplicity of presentation, we assume a set of independent tasks. With given execution modes, the task mapping problems can be classified to two different cases, i.e., Homogeneous Execution Modes and Heterogeneous Execution Modes.

3.1 Homogeneous Execution Modes

In this section, we show that Hungarian Algorithm [46] can be the subroutine of our approach to solve the case that all the tasks require a homogeneous execution mode in time complexity $O(N^3)$. There exists two cases: either all the tasks are executed by TMR-based RMT mode, or none of them requires RMT. We will focus on the former case, and explain how to cope with the latter case at the end of this section.

For the sake of completeness, we link both cases to the well-known minimum weight perfect bipartite matching problem (MWPBM). In the MWPBM problem, there is a bipartite graph $\mathcal{G} = (\mathbb{V}, \mathbb{E})$ with two disjoint subsets $\mathbb{X} \subseteq \mathbb{V}$ and $\mathbb{Y} \subseteq \mathbb{V}$, where $\mathbb{E}$ is the set of edges between $\mathbb{X}$ and $\mathbb{Y}$. Each edge e in $\mathbb{E}$ is associated with a weight $w(e)$. The MWPBM problem is to find out a perfect matching of maximum weight where the weight of matching $\mathbb{M}$ is given by $w(\mathbb{M}) = \sum_{e\in\mathbb{M}} w(e)$. We use the terms $\mathbb{X}$, $\mathbb{Y}$, *and* $w(\mathbb{M})$ to refer to the MWPBM problem; the terms $\mathbb{C}$, Γ, *and* ϕ_Γ to refer to our problem, where $\phi_\Gamma = \sum_{\tau_i\in\Gamma} \phi(\tau_{i,k}, c_m)$. For the rest of this paper, the way we use the bipartite graph is defined as the following:

Definition 2. *To build bipartite graph $\mathcal{G} = (\mathbb{V}, \mathbb{E})$, we take the tasks as the nodes in subset $\mathbb{X}$, and the cores as the nodes in subset $\mathbb{Y}$, in which $\mathbb{X} \cup \mathbb{Y} = \mathbb{V}$, and $\mathbb{X} \cap \mathbb{Y} = \varnothing$. With version selection table ψ, each weight of edge $e \in \mathbb{E}$ can be referred to a corresponding entry $\psi(\tau_i, c_m)$ which connects two specific nodes, i.e., τ_i in $\mathbb{X}$ and c_m in $\mathbb{Y}$.*

RMT execution for all tasks. In this case, each task needs three cores to execute TMR-based RMT mode. Although all the tasks have the same demanded number of cores for the redundancy, the way we group and assign the cores will affect their deadline miss rate. According to the observation on TMR-based RMT, we know that the execution time of each task in TMR-based RMT mode relies on the lowest frequency core in its assigned core group. Therefore, to increase the feasibility of core grouping for the following task mapping procedure, an optimal grouping of cores should have the maximal summation of frequencies from each lowest frequency cores among all the groups. The following theorem shows that the optimal core grouping can be obtained by grouping every three cores $\{c_i, c_{i+1}, c_{i+2}\}$, $\forall i = 1, 4, 7, \ldots, (\delta_{\vec{\lambda}} - 2)$ consecutively.

Theorem 1. *Given a set of cores $\mathbb{C}$ with variation, grouping every three adjacent cores $\{c_i, c_{i+1}, c_{i+2}\}, \forall i = 1, 4, 7, \ldots, (\delta_{\vec{\lambda}} - 2)$ may obtain the optimal grouping which has the maximal summation of frequencies from each lowest frequency cores among all the groups.*

Proof. First of all, it is not difficult to see that the first $\delta_{\vec{\lambda}}$ high-frequency cores are definitely used in an optimal solution and formed into N groups, in which each group has 3 cores. Suppose that there is an optimal grouping solution, in which c_1 is in group $\mathbb{G}'_1$, c_2 is in group $\mathbb{G}'_2$, and c_3 is in group $\mathbb{G}'_3$. We only consider the case that $\mathbb{G}'_1 \neq \mathbb{G}'_2 \neq \mathbb{G}'_3$, as the other cases are simpler than this case. Let c_i, c_j, c_k be the lowest-frequency cores in each of these three groups. Without loss of generality, we index these three cores such that $i < j < k$, in which $i \ge 4$ by definition. Therefore, c_k can be in any of the three groups in this index rule and the summation of frequencies among these groups is $f_i + f_j + f_k$.

Now, we (1) swap c_2 in $\mathbb{G}'_2$ and the second fast core in group $\mathbb{G}'_1$ and (2) swap c_3 in $\mathbb{G}'_3$ and the lowest-frequency core in group $\mathbb{G}'_1$. These three groups now are called $\mathbb{G}^*_1, \mathbb{G}^*_2, \mathbb{G}^*_3$. The lowest-frequency core in $\mathbb{G}^*_1$ is c_3, which does not have lower frequency than the c_i. Moreover, after swapping, either the lowest-frequency core in $\mathbb{G}^*_2$, or the lowest-frequency core in $\mathbb{G}^*_3$ is core c_k. So, there are two cases:

- **Case 1**: lowest-frequency core in $\mathbb{G}^*_2$ is c_k: This implies that the lowest-frequency core of $\mathbb{G}'_2$ is also c_k, due to the fact that the swapping procedure does not change the lowest-frequency core in group $\mathbb{G}'_2$. If core c_i is in $\mathbb{G}'_3$ and core c_j is in $\mathbb{G}'_1$, then after swapping the lowest-frequency core of $\mathbb{G}^*_3$ is c_j. Similarly, if core c_i is in $\mathbb{G}'_1$ and core c_j is in $\mathbb{G}'_3$, then after swapping the lowest-frequency core of $\mathbb{G}^*_3$ remains as c_j. We illustrate these two scenarios in Fig. 4, where the first and second scenarios of the three groups $\mathbb{G}'_1, \mathbb{G}'_2, \mathbb{G}'_3$ are in Figs. 4a and 4b, respectively. Therefore, for such a case, we show that the lowest-frequency core in $\mathbb{G}^*_3$ is c_j.
- **Case 2**: lowest-frequency core in $\mathbb{G}^*_3$ is c_k: For such a case, with the similar procedure as in Case 1, we can show that the lowest-frequency core in $\mathbb{G}^*_2$ is c_j.

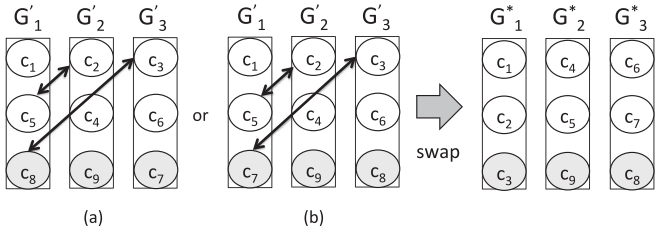


Fig. 4. Example of two swapping scenarios in case 1 in the proof of Theorem 1. In this example, c_i is c_7, c_j is c_8, and c_k is c_9. After swapping, the frequencies of cores are increased from $f_7 + f_8 + f_9$ to $f_3 + f_8 + f_9$.

As a result, the grouping $\mathbb{G}_1^*, \mathbb{G}_2^*, \mathbb{G}_3^*$ has higher total frequency (with respect to the lowest-frequency cores in the three groups) where $f_3 + f_j + f_k < f_i + f_j + f_k$ and $\mathbb{G}_1^* = \{c_1, c_2, c_3\}$. If we continue swapping, we can eliminate all differences between the grouping $\mathbb{G}_1', \mathbb{G}_2', \mathbb{G}_3'$ and the consecutive core grouping $\{c_i, c_{i+1}, c_{i+2}\}, \forall i = 1, 4, 7, \ldots, (\delta_{\vec{\lambda}} - 2)$ without decreasing the total frequency of the solution. Therefore, we reach the conclusion. □

Based on Theorem 1, the task mapping problem can be transformed to be equivalent to the MWBPM problem with N tasks and N groups, $\{c_i, c_{i+1}, c_{i+2}\}, \forall i = 1, 4, 7, \ldots, (3N - 2)$. Therefore, we can construct the corresponding bipartite graph and adopt Hungarian Algorithm as the subroutine to find an optimal assignment with the minimal overall reliability penalty while all the miss rate constraints are satisfied. According to the preprocessing, we can ensure that if a task is not feasible to execute on a certain core, the value of corresponding entry in table ϕ must be infinity. With the above setting, it is clear that if ϕ_Γ is infinity, there is no feasible assignment with such an input set of tasks Γ and cores $\mathbb{C}$. *Up to here, it should be also clear how to handle the case when none of the tasks require RMT execution. That is also a perfect matching problem by assigning N tasks to N cores.*

Algorithm 1. Homogeneous Execution Modes

Input: set of tasks Γ; vector $\vec{\lambda}$ with execution modes for tasks, $\forall \lambda_i \in \{\text{NR}, \text{TMR}\}$; set of cores $\mathbb{C}$; best versions table ψ;
Output: Mapping $\mathbb{M}$ with the set of selected versions;
1: $\text{List}_c^* \leftarrow \emptyset$;
2: **if** All execution modes are TMR **then**
3: **for** $c_j \in \mathbb{C}, j \leftarrow 1, 4, \ldots, \delta_{\vec{\lambda}-2}$ **do**
4: //Assign the core grouping by Theorem 1
5: $\text{List}_c^* \cup \mathbb{G}_j = \{c_j, c_{(j+1)}, c_{(j+2)}\}$;
6: **end for**
7: **else if** All execution modes are NR **then**
8: $\text{List}_c^* \leftarrow \mathbb{C}.\text{head}(\delta_{\vec{\lambda}})$;
9: **end if**
10: $\mathcal{G} \leftarrow$ build Bipartite Graph with Γ, List_c^*, and ψ;
11: $\mathbb{M} \leftarrow$ find the mapping by HungarianAlgorithm($\mathcal{G}$);
12: **if** ϕ_Γ is ∞ **then**
13: return **FAIL**
14: **end if**

As a result, we summarize our approaches as Algorithm 1 to comprehensively handle both cases (all tasks are protected under TMR-based RMT or none of the tasks are protected). If the tasks require the RMT execution, we prepare a core group list List_c^* to record the cores for each group with the optimal grouping of cores by Theorem 1 (line 5). If none of the tasks require RMT execution, we choose the first $\delta_{\vec{\lambda}}$ cores from $\mathbb{C}$ and record in List_c^* (line 8). Then, the corresponding bipartite graph $\mathcal{G}$ can be constructed by Γ, List_c^*, and ψ by Definition 2. With the bipartite graph $\mathcal{G}$, we can find the minimum weight bipartite perfect matching and assign the tasks and cores with mapping $\mathbb{M}$ by Hungarian Algorithm and the bipartite graph including the information of possible mappings (line 11). In particular, if the total weight of mapping is infinity, we know that there is no feasible assignment to satisfy the miss rate constraint (lines 12-14).

According to the perfect matching property, preprocessing, and definition of $\mathcal{G}$, we can ensure that Algorithm 1 will deliver a feasible mapping $\mathbb{M}$ for tasks and cores, where each core only appears once in a specific core group while the miss rate constraint is satisfied. The time complexity is dominated by Hungarian Algorithm with $2N$ nodes, i.e., $O(N^3)$.

3.2 Heterogeneous Execution Modes

In this section, we consider the task mapping problem with arbitrary $\vec{\lambda}_i$ as the heterogeneous execution modes. For such a case, the approach in Section 3.1 by reducing the assignment problem to the MWPBM problem is no longer applicable, *since the bipartite graph cannot be built due to the unknown properties of core grouping in optimal solutions.*

However, we observe that it is beneficial to assign the cores of TMR-based RMT tasks before NR tasks, as the TMR-based RMT tasks are fully protected with a negligible reliability penalty ϵ. No matter which cores are assigned to TMR-based RMT tasks, their reliability penalty is always negligible. In order to supply more resilient cores in terms of performance for the NR tasks, the frequencies of assigned cores for TMR-based RMT tasks should be as *low* as possible. Therefore, we propose our approach for this heterogeneous case, which consists of two parts: assigning TMR-based RMT tasks and assigning NR tasks. Algorithm 2 presents the pseudo code for the two portions of tasks assignment with heterogeneous execution modes.

Assigning TMR-based RMT tasks. First of all, s_i denotes the resilience of TMR-based RMT task by the *lowest acceptable core frequency* of task τ_i as Eq. (3) (line 4 in Algorithm 2)

$$s_i = \underset{1 \le j \le \delta_{\vec{\lambda}}}{\arg} \left\{ \psi(\tau_i, c_j) \neq \infty, \text{and } \psi(\tau_i, c_{j+1}) = \infty \right\}, \quad (3)$$

where $\psi(\tau_i, c_{\delta_{\vec{\lambda}}+1})$ is set to a dummy version with ∞ penalty for notational brevity. To maximize the number of tasks satisfying their deadline, the assignment of TMR-based RMT tasks should start from the most resilient task, which accepts the lowest frequency core among all the redundant cores.

To find out the most resilient tasks, we sort all the TMR-based RMT tasks by a non-increasing order of c_{s_i}'s speed, and re-index them by the sorted list, in which ties are broken arbitrarily (line 6). The assigning procedure starts from the most resilient task τ_k, which has the maximum index s_k of cores in $\mathbb{C}$, with the lowest-frequency group $\mathbb{G}_k$, where $\mathbb{G}_k = \{c_{s_k-2}, c_{s_k-1}, c_{s_k} | s_k \geq 3\}$ (line 8). Then, we exclude the cores of $\mathbb{G}_k$ from $\mathbb{C}$ and consider the next resilience-wise task τ_{k-1} (line 9). For task τ_i, s_i should be the lowest

frequency core among the rest of cores, if the original s_i is assigned to the task already (lines 11-13). By repeating the above procedure, the frequencies of assigned cores will be as low as possible which satisfies the minimal requirement of core frequency for each TMR-based RMT task.

Assigning NR tasks. After assigning the TMR-based RMT tasks, the rest of cores and NR tasks can be transformed to MWPBM problem as Section 3.1. As a result, we can make a bipartite graph $\mathcal{G}$ (line 18) by Definition 2 and finish a perfect matching $\mathbb{M}$ by Hungarian Algorithm with the minimal ϕ_Γ as the optimal result (line 19). If the procedure cannot find a feasible mapping, the algorithm returns that there is no feasible solution (lines 20-22). With the TMR-based RMT tasks assignment and the perfect matching property, we can ensure that the mapping assignment $\mathbb{M}$ derived by Algorithm 2 is feasible for tasks and cores, where each core is only assigned to one unique task while all the miss rate constraints in ρ_Γ are satisfied. The time complexity is similar as Algorithm 1, which is dominated by the Hungarian Algorithm, i.e., $O(N^3)$.

Algorithm 2. Heterogeneous Execution Modes

Input: set of tasks Γ; vector $\vec{\lambda}$ with execution modes for tasks, $\forall\lambda_i \in \{\mathrm{NR}, \mathrm{TMR}\}$; set of cores $\mathbb{C}$; best versions table ψ;
Output: Mapping $\mathbb{M}$ with the set of selected versions;
1: //Assigning TMR tasks
2: $\mathrm{List}_c^* \leftarrow \mathbb{C}$; $\mathrm{List}_L \leftarrow \Gamma_{\mathrm{TMR}}$;
3: **for each** $\tau_i \in \mathrm{List}_L$ **do**
4: find out s_i based on Eq. (3);
5: **end for**
6: sort List_L by s_i and re-index them;
7: //τ_k has the maximum index as s_k
8: $\mathbb{M} \leftarrow$ assign $\mathbb{G}_k = \{c_{(s_k-2)}, c_{(s_k-1)}, c_{s_k} | s_k \geq 3\}$ to τ_k;
9: remove the cores of $\mathbb{G}_k$ from List_c^*;
10: **for each** $\tau_i \in \mathrm{List}_L, i = (k-1), (k-2), \ldots, 1$ **do**
11: **if** $s_i > s_{(i+1)} - 3$ **then**
12: $s_i \leftarrow s_{(i+1)} - 3$;
13: **end if**
14: $\mathbb{M} \leftarrow$ assign τ_i with $\mathbb{G}_i = \{c_{(s_i-2)}, c_{(s_i-1)}, c_{s_i}\}$;
15: remove the cores of $\mathbb{G}_i$ from List_c^*;
16: **end for**
17: //Assigning NR tasks
18: $\mathcal{G} \leftarrow$ build Bipartite Graph with Γ_{NR}, List_c^*, and ψ;
19: $\mathbb{M} \leftarrow$ find the mapping by HungarianAlgorithm($\mathcal{G}$);
20: **if** ϕ_Γ is ∞ **then**
21: return **FAIL**;
22: **end if**

The solution derived from Algorithm 2 can be proved to be *optimal* in terms of overall reliability penalty if there exists a feasible solution for the input. If we try to handle TMR and NR tasks concurrently, there is no efficient way to decide the core grouping beforehand. However, as the reliability penalty of TMR tasks are negligible, we can reach the optimality as shown in the following theorem.

Theorem 2. *Given a set of cores $\mathbb{C}$, a set of tasks Γ, heterogeneous execution modes of tasks $\vec{\lambda}$, and tolerable deadline miss rates ρ_Γ. Algorithm 2 provides a feasible task mapping with the minimal overall reliability penalty under heterogeneous execution modes $\vec{\lambda}$.*

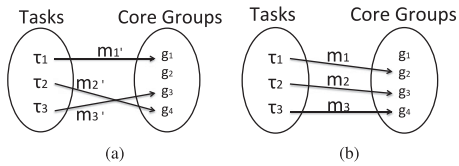


Fig. 5. Example of task mappings for RMT tasks and core groups in the proof of Theorem 2, in which (a) is an optimal solution and (b) is delivered by Algorithm 2.

Proof. As the reliability penalty of TMR-based RMT is a negligible value ϵ in our model, the overall reliability ϕ_Γ under heterogeneous execution modes can be reformulated from Definition 1 to:

$$\phi_\Gamma = \sum_{\tau_i \in \Gamma_{\mathrm{NR}}} \phi(\tau_{i,k}, c_m), c_m \in \mathbb{C}. \quad (4)$$

According to Eq. (4), we can observe that the overall reliability ϕ_Γ fully relies on the frequencies of assigned cores for NR tasks, if all TMR-based RMT tasks are feasible to execute with their assigned cores. Therefore, we know that the delivered task mapping will be an optimal mapping if the assigned cores of NR tasks have the maximal summation of frequencies to obtain the minimal overall reliability penalty.

As the candidate cores for NR tasks are the remaining cores after mapping the TMR-based RMT tasks, the assigned cores for TMR-based RMT tasks must have the lowest summation frequencies (for those lowest frequency cores in each group) to let the remaining cores have the maximal summation frequencies. The core grouping in Algorithm 2 groups every three adjacent low-frequency cores, which guarantees the optimal feasibility for TMR-based RMT tasks by Theorem 1 and the lowest frequencies among the candidate cores. In the following, we will prove that the assignment for TMR-based RMT tasks in Algorithm 2 based on the above grouping which can find out an optimal mapping such that the rest of cores for NR tasks have the maximal summation of frequencies.

Assume there is an optimal task mapping between TMR-based RMT tasks and core groups as Fig. 5a, and the mapping Fig. 5b is the result of Algorithm 2, in which the core groups are sorted by their lowest frequency core, i.e., $g_1 > g_2 > g_3 > g_4$. Then there are two cases:

- **Case 1:** There are two consecutive mappings in a different order in Fig. 5a than they are in Fig. 5b: For such a case, we swap the order for these two consecutive mappings, i.e., m_2' and m_3', and they become m_2 and m_3. After the swapping procedure, the total frequency of remaining cores will be the same, as the assigned cores for TMR-based RMT tasks are not changed.
- **Case 2:** There is an element of Fig. 5a not in Fig. 5b and an element of Fig. 5b not in Fig. 5a: We swap g_1 and g_2 for the element of mapping m_1', and now m_1' becomes m_1. As the rest of the cores are changed from group g_1 to g_2, we know that the total frequency of rest of the cores is not less than before i.e., $g_2 < g_1$.

As a result, the swapping procedure shows that the delivered mapping is no worse than before. The differences between Figs. 5a and 5b are eliminated without worsening the total frequency of the solution. We know that the delivered mapping is as good as any optimal solution in which the rest of cores have the maximal total frequency. As the optimality of Hungarian Algorithm has been proved in [46], the delivered task mapping must be optimal with the minimal overall reliability penalty. As a consequence, we reach the conclusion that the delivered task mapping by Algorithm 2 is optimal. □

4 Execution Modes Adaptation

Until now, the assumption was that the execution modes of tasks $\vec{\lambda}$ are given. In this section, we present an iterative mode adaptation which determines the execution modes of tasks with the proposed mapping approaches in Section 3.2.

To minimize the overall reliability penalty, it is beneficial to execute as many tasks as possible in TMR-based RMT mode. However, which tasks should execute in TMR-based RMT mode is not that trivial to determine. Some tasks may suffer from their higher vulnerability, whereas some of tasks may suffer from their tighter deadline miss rate.

Intuitively, activating the TMR-based RMT mode for the task with the highest reliability penalty is a reasonable way to decrease the overall reliability penalty as the greedy approach in [39]. However, the task with the "highest reliability penalty" is only relative to a specific core frequency, e.g., on the highest frequency core. If we greedily execute this task in the TMR-based RMT mode, all the possible task mappings for the rest of tasks may even lead to an inferior overall system reliability. In addition, we are not able to know how a task is vulnerable under the core grouping, as the core grouping for TMR-based RMT execution is still unknown at this moment. Since checking all the combinations of execution modes may not be possible, here we propose an iterative approach exploiting our task mapping approaches as the subroutine to guarantee the feasibility and efficiency of execution modes.

Algorithm 3 presents the pseudo-code of mode adaptation. It first adopts Algorithm 1 to find out the mapping between the tasks and cores for the initial case that none of the tasks require TMR-based RMT execution, which helps us find out a reasonable reference to upgrade the execution modes (lines 2-3). Then, the following procedure is repeated until there is no more improvement. At first, we consider the task with the maximal reliability penalty in the current mapping solution (line 7). The objective is to upgrade one more task from NR to TMR-based RMT with the current solution for the reliability improvement. For this task, we greedily upgrade its mode by picking up two unused cores that can satisfy the miss rate constraint of the task. If the upgrade is feasible, we can adopt Algorithm 2 to find out the next mapping $\mathbb{M}$ (lines 9-10); otherwise, we rollback the infeasible upgrade (line 12). Then, we continue the procedure finding the next high penalty task for upgrading from NR to TMR (line 14). When there is no more improvement and all the tasks are considered, we can terminate the iterative procedure.

Algorithm 3. Modes Adaptation and Task Mapping

Input: set of tasks Γ; set of cores $\mathbb{C}$; best versions table ψ;
Output: Mapping $\mathbb{M}$ with the set of selected versions;
1: //Mapping the first case
2: $\text{Vector}_{\vec{\lambda}} \leftarrow$ assign all execution modes of tasks as NR;
3: $\mathbb{M} \leftarrow$ apply Algorithm 1 with Γ and $\mathbb{C}$ to find out the mapping;
4: **if** ϕ_Γ is ∞ **then**
5: return **FAIL**;
6: **end if**
7: find out task τ_h with the highest penalty in mapping $\mathbb{M}$;
8: **for each** $\tau_h, \tau_h \in \Gamma_{\text{NR}}$ **do**
9: $\lambda_h \leftarrow$ assign the execution mode of task τ_h as TMR;
10: $\mathbb{M} \leftarrow$ apply Algorithm 2 to find out the task mapping;
11: **if** ϕ_Γ is ∞ **then**
12: restore λ_h to NR
13: **end if**
14: check the next τ_h in mapping $\mathbb{M}$;
15: **end for**

Algorithm 3 may deliver a feasible mapping $\mathbb{M}$ and minimize the system reliability penalty ϕ_Γ with as many as possible RMT tasks. The time complexity is only scaled by the number of tasks N, which is still applicable to be used online.

5 Communication and Dependency

In this section, we present how to deal with the communication and dependency among the tasks when we considering the task mapping problem on a many-core processor. We consider the communication fabric with the most popular deterministic routing algorithm, i.e., XY routing (proven to be deadlock-free) [47], on the most common topology, i.e., two-Dimension mesh. Since the many-core processor only has a single thread per core, to parallelize the execution of dependent tasks and utilize all the redundant cores concurrently, one way is to adopt the well-known technique, i.e., software pipelining, to dispatch the dependencies into different pipeline stages, where the dependent inputs of tasks can be transformed by the predecessors before the execution of next pipeline stage. With the software pipelining, we can consider the task mapping with the data dependencies on all the redundant cores concurrently. We assume that the communication overhead is significantly less than the computation overhead, so that the system reliability may not be dramatically changed.

For the given task graph $\mathbb{G}$, we prepare π_i to denote whether task τ_i has a predecessor: π_i is equal to 1 if task τ_i has a predecessor; otherwise, π_i is 0. Although all the dependent tasks can execute at the same time with software pipeline, the communication of dependent pipelines has to be considered with the allocation of assigned cores. As shown in Fig. 6a, the different allocation of assigned cores, may lead to a different communication distance. In addition, the TMR-based RMT also has internal communication among the redundant threads for the majority-voting mechanism. As shown in Fig. 6b, the allocation of cores dedicated for TMR-based RMT execution has to be considered for avoiding any unnecessary performance penalty.

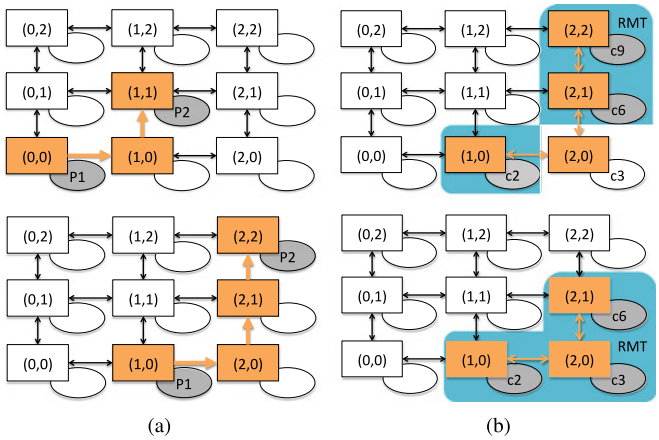


Fig. 6. The communication on 2D mesh topology with XY routing. In (a), pipelines P_1 and P_2 have the communication. In (b), RMT adopts $\{c_2, c_6, c_9\}$ and $\{c_2, c_3, c_6\}$, respectively.

In general, the communication overhead can be estimated by considering the data size, the required cycles per hop, and the distance of communication. However, the realistic distance of communications can only be calculated after the allocation of the tasks and cores is done. To mitigate the uncertainty, we propose to estimate the overhead with the maximal distance on the 2D mesh to cover the worst execution scenario. As shown in Fig. 7, if a task is assigned to core c_i in a 3×3 mesh topology, the maximal distance will be 4 hops as c_i to c_k. If it is assigned to the pipeline on core c_j, the maximal distance will be 2 hops as c_j to c_k. Therefore, by applying XY routing, the maximal distance of communication on core c_m, denoted by q_m can be calculated statically.

For the communication of dependent pipelines, the execution time overhead can be estimated by the input data size of task and the maximal distance q_m of assigned core c_m of pipeline as the following:

$$\Delta_{\text{in}}(\tau_i, c_m) = q_m \times inputData\_size(\tau_i). \quad (5)$$

Similarly, since the majority-voting mechanism has to wait for all the output data of the redundant threads, the internal communication in RMT can be estimated by the output data size of task and the maximal distance q_m of assigned core c_m as the following:

$$\Delta_{\text{out}}(\tau_i, c_m) = q_m \times outputData\_size(\tau_i). \quad (6)$$

Assume the data transfer spends μ cycles per hop. Since the communications prolong the execution time of tasks, the interval between the release and deadline is reduced by all the possible communications as the following:

$$D_i' = D_i - \mu \cdot (\lambda_i \cdot \Delta_{\text{out}}(\tau_i, c_m) + \pi_i \cdot \Delta_{\text{in}}(\tau_i, c_m)). \quad (7)$$

As a consequence, the deadline miss rate of task as Eq. (1) with the pipeline on core c_m should be reformulated as Eq. (8):

$$P_{\text{dm}}(\tau_{i,k}, c_m) = 1 - C_{i,k,m}(D_i'). \quad (8)$$

With the reformulated deadline miss rate Eq. (8), we can incorporate the communication overhead into our proposed approaches. Please note that the applicability is not limited to XY routing. The approximation can be easily extended for the other deterministic routing algorithms by changing Eq. (5) and (6) accordingly.

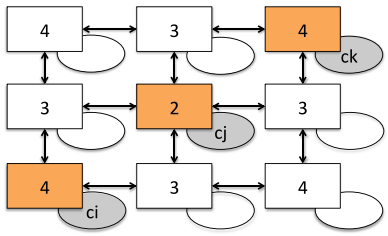


Fig. 7. Example of the maximal distance/hops estimation in a 3×3 mesh. The maximal distance will be 4 if the task is assigned to c_i. It can be bounded by 2, if it assigned to c_j.

If none of the tasks require RMT execution, Algorithm 1 is still optimal to the task mapping with the data dependency due to the optimality of Theorem 1 and Hungarian Algorithm for the worst case. However, if the compatibility of cores is arbitrary to each task, Theorem 1 can not hold any more, in which the feasibility and frequencies do not have the absolute relation. Even if a core has the highest frequency among the others, it may not be suitable for the task which has significant communication overhead. As the cores' positions also affect the feasibility of mappings, it is not good enough to determine the assignment sequence only by the frequencies of cores.

Algorithm 4 takes the above issues into consideration and solves the task mapping problem with the communication and data dependencies. At first, we have to reformulate the deadline miss rate of task versions in the best versions table ψ with Eq. (8) (line 2), which can be done in the preprocessing. To present the impact of communication, we denote the number of available cores for task τ_i as a_i, which can be obtained by calculating the number of feasible entries in the reformulated best version table ψ (line 3). Then, we sort the tasks with the corresponding a_i by a non-increasing order. (line 4). The assignment starts from the task with the minimal number of available cores and assign three lower frequency cores among the available cores (lines 5-12). If the task cannot be satisfied by the remaining available cores, it is clear that there is no further feasible solution (lines 9-11). By checking all the RMT tasks, all the assigned cores and TMR tasks are excluded as the procedure in Section 3.2. As the rest of tasks are only NR tasks in Γ, we follow the same procedure as Algorithm 2 to build up the bipartite graph and find out a perfect matching $\mathbb{M}$ by Hungarian Algorithm with the minimal ϕ_Γ (lines 14-18). Please note that, in case of dependent tasks where the predecessor output has soft errors, we assume that the errors can be recovered by task re-execution before it is served to its dependent task, which is similar to *dTune* [39].

Consequently, we can find out a reasonable task mapping $\mathbb{M}$ by using Algorithm 4 and the reformulated versions table ψ. The time complexity is the same as Algorithm 2, i.e., $O(N^3)$.

6 Results and Discussion

6.1 Experimental Setup

To evaluate the performance of our schemes, we use the same setting as *dTune* [39]. The overview of the

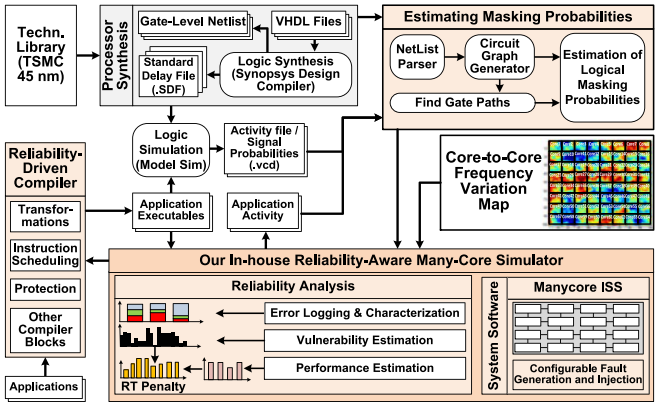


Fig. 8. Experimental setup with reliability-driven compiler, system software, and processor simulator.

experimental setup is shown in Fig. 8 and briefly explained in the following. More details can be found in [7], [48], [49].

Algorithm 4. Task Mapping with Data Dependency

Input: set of tasks Γ; set of cores $\mathbb{C}$; best versions table ψ;
Output: Mapping $\mathbb{M}$ with the set of selected versions;
1: $\text{List}_L \leftarrow \Gamma_{\text{TMR}}$, $\text{List}_c \leftarrow \mathbb{C}$;
2: reformulate best versions table ψ with Eq. (8);
3: calculate the number of available cores a_i for Γ_{TMR};
4: sort and re-index List_L by a_i;
5: **for each** $\tau_i \in \text{List}_L, i = 1, 2, \ldots, k$ **do**
6: //τ_k has the minimal number of available cores
7: assign three lower frequency cores to τ_i in List_c;
8: remove the assigned cores from List_c;
9: **if** τ_i is not able to activate RMT **then**
10: return **FAIL**;
11: **end if**
12: **end for**
13: //Assigning NR tasks
14: $\mathcal{G} \leftarrow$ build Bipartite Graph with Γ_{NR}, List_c, and ψ;
15: $\mathbb{M} \leftarrow$ find the mapping by HungarianAlgorithm($\mathcal{G}$);
16: **if** ϕ_Γ is ∞ **then**
17: return **FAIL**;
18: **end if**

We deploy seven functions mixed with a complex video encoder H.264 from an embedded benchmark MiBench [50]: (1) SAD, (2) ADPCM, (3) CRC, (4) SusanS, (5) SHA, (6) SATD, and (7) DCT, where the data dependencies are as follows:

- DCT → SAD → SATD
- ADPCM → CRC
- SUSAN → SHA

Each application function is compiled with a reliability-driven compiler, which is based on the GCC framework. Different reliable function versions are generated by applying different reliability-driven transformations [7], [51] and a reliability-driven instruction scheduling algorithm [45]. These reliable function versions provide trade-off points for reliability versus performance. Fig. 9a shows an example in our setup that different function versions under the same frequency core have different performance. A subset of Pareto-optimal versions is selected and used by the runtime system. For each compiled version, we estimate the

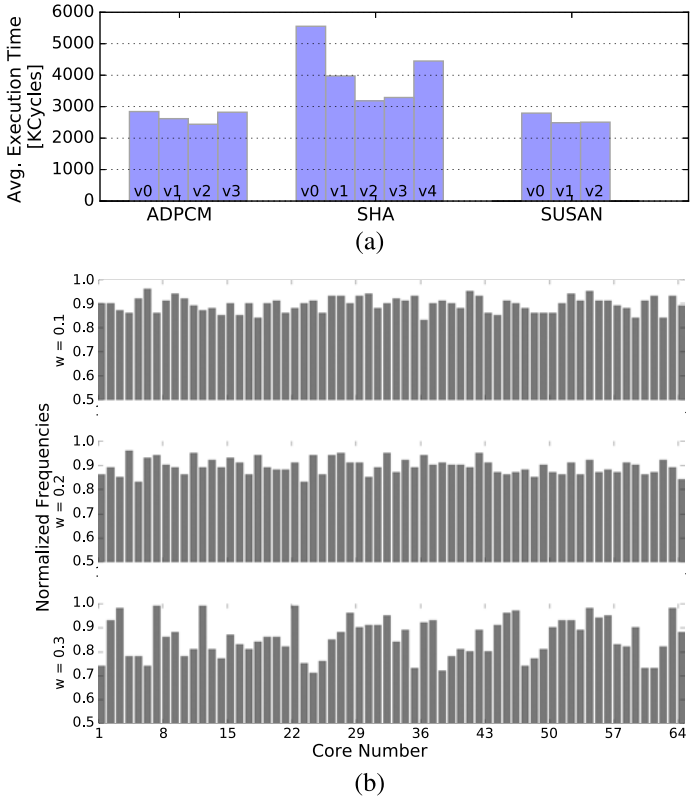


Fig. 9. Examples of the experimental setup: (a) Performance of different on the same frequency core; (b) Core-to-core frequency variation in normal distrbution for 8×8 cores.

performance and the values of reliability penalties under two different fault rates, i.e., $\eta = 10^{-6}$ and 10^{-7} (in the unit of $\#fault/cycles$) to realize high fault scenarios as adopted by the related works [52], [53]. The reliability penalty for each function/task is estimated using the approach of [7], [49].

Overall, our framework employs a reliability-aware many-core simulator with integrated configurable fault generation and injection modules. Each core implements the SPARC-v8 ISA (used in LEON2 and LEON3 cores), which is generated using the ArchC architecture description language and related tools [54]. We extended the simulator with in-house developed configurable fault generation and injection modules, and error analysis/logging functionalities. These are required to perform an in-depth reliability/vulnerability analysis. For accurate reliability estimation, we synthesized the LEON3 cores using the Synopsys Design Compiler for a TSMC 45 nm technology library to obtain area, frequency, and logical masking probabilities. We performed gate-level error masking/propagation analysis on the netlists to obtain logical masking probabilities of different processor components. These probabilities are then used to obtain the instruction vulnerabilities that are later used to estimate task reliability penalties; see detailed procedure in [7], [49]. For fault scenario generation, different parameters (like number of bit flips per fault, fault rate using the neutron flux calculator [55] and coordinates of a given location, fault distribution, etc.) are used. Faults in different processor components are randomly injected (as also done in [3], [56]) during the execution of a given function version. Their effects on the application output are monitored using

an error logger. Errors are categorized based on the severity from the user's perspective (e.g., application failure, incorrect output, correct output). The results of the fault injection experiments are used: (1) to estimate the software-level vulnerability and masking properties of the applications; and (2) to analyze the reasons for application failures, e.g., accessing prohibited memory regions and non-decodable instructions.

We evaluate our mapping approach as Algorithm 2, the mode adaptation approach as Algorithm 3, and Algorithm 4 with the generated reliability penalty value, different execution modes, and cores performance heterogeneity. For evaluation, we generate 128 different execution modes for the above seven functions, i.e., 2^7, to test our approaches and the greedy mapping approaches used in *dTune*. Depending upon the performance heterogeneity, the infeasible scenarios in the evaluation are excluded. To simulate the performance heterogeneity of cores, the evaluation is performed by three different scenarios with variations ω on 8×8 cores as follows:

- *Grouping frequency levels*: such scenarios are for evaluating architectures with heterogeneous performance, e.g., ARM big.LITTLE architecture [26]. We evaluate four different frequency levels in a multicore processor. We assume the performance variation is ω, where the cores are with frequencies f_1, $(1-\omega)f_1$, $(1-2\omega)f_1$, and $(1-3\omega)f_1$.
- *Uniform distribution*: Based on the variation model of [42], we uniformly generate the frequencies of cores from the highest one f_1 to the lowest one f_M to consider process variations.
- *Normal distribution*: The various frequencies of cores are normally distributed/generated [57] in the range of $(0,1]\cdot f_1$ with the mean $1-\omega$ and standard deviation $\sigma = 0.05$ to consider process variations. As it is possible that the normal distribution has a random variable greater than 1 or less than 0, we take such cases to the corresponding boundary conditions.

Considering real-word scenarios on performance variations, we only evaluate our proposed approaches while ω is up to 30 percent [24]. Fig. 9b shows the example variation scenarios under normal distribution. For simplicity of presentation, we set all the individual miss rate $\forall\rho_i \in \rho_\Gamma$ with the same rate ρ.

For each configuration of core frequencies, we generate 500 different processors with different variations, and report the average results. As we are not aware of any other state-of-the-art related works, we normalize our results to the greedy mapping and compare the efficiency with the same set of task versions and core configurations for fairness, in which the normalized ratio is calculated as ϕ_γ of the resulting solution divided by ϕ_γ of the greedy mapping. By definition, the lower normalized penalty ratio is better.

6.2 Simulation Results for Task Mapping

In these simulations, we evaluate our mapping approach with all the possible execution modes. Each bar in the presented figures is obtained by averaging the reliability results through these 128 different execution modes. Since the greedy mapping cannot guarantee the feasibility of the task mapping, it may be possible that the greedy mapping is not a feasible one to meet the miss rate constraint.

Simulation without data dependency. Fig. 1 in the supplementary material, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TC.2016.2532862, shows the evaluation results under two different fault rates, i.e., 10^{-6} and 10^{-7}. Overall, we can observe that our approach outperforms the greedy mapping approach, and the average improvement is around 20 percent among all the cases. In particular, the improvement can be up to 80 percent (0.2 in the bar plot) when the fault rate is 10^{-7} under Grouping Frequency Levels. In such scenarios, the reliability penalties may play a minor role, whereas the greater penalties of timing constraint violations dominate the value of the penalty function. Moreover, when the variations of performance among the cores are higher, our approach is typically more effective than the greedy mapping approach. It is because our approach prevents the severe degradation of reliability, in which the cores are not grouped properly. Please note, if the design constraints are too strict, none of the approaches can deliver a feasible solution. Even if there exists a feasible solution, there is no space to further improve the reliability among different approaches.

In case the difference of frequencies between different grouping levels is large enough, the greedy mapping approach may suffer from the sequential assignment of cores, in which the task with RMT mode may have severe performance degradation due to the domination of its lowest-frequency core in the majority-voting. In particular, the most improvement can reach up to 80 percent when $\omega = 0.3$, $\eta = 10^{-7}$, and $\rho_\Gamma = 5\%$.

Interestingly, we can observe that the improvement is not significant under the scenario of uniform distribution. Among all the possible combinations, the differences of overall reliability penalty between both approaches are not significant, since the frequencies of the cores are degraded smoothly. Nevertheless, our task mapping approaches can still perform well in some cases. For example, in the lower fault rate as 10^{-7}, the improvement can be up to 31 percent, when the tolerable miss rate is higher, i.e, $\rho_\Gamma = 30\%$.

In the scenario of normal distribution, some of the results with the severe performance variations, i.e., $\omega \geq 0.16$, have no feasible solutions in the simulation. Due to the lack of high-frequency cores, most of the execution modes cannot be satisfied, in which most of the cores are degraded as the middle-frequency under normal distribution. When the tolerable miss rate is strict with the lower fault rate, i.e., $\eta = 10^{-7}$ and $\rho = 15\%$, the results in our simulations depend upon the timeliness of task mapping, in which the improvement is less because of the negligible differences of feasible mappings. Among all the feasible constraints, our proposed approach outperforms the greedy mapping approach under both fault rates.

Fig. 10 presents the comparison results under different fault rates for a more complicated application scenario. We construct this application by using the functions selected from MiBench as mentioned previously, and duplicate the functions to increase the demand for cores. Based on 14 functions in the complicated application, we examine $16{,}384$ different execution modes, i.e., 2^{14}, and present the overall reliability penalty ratio in average. As a result, we know that

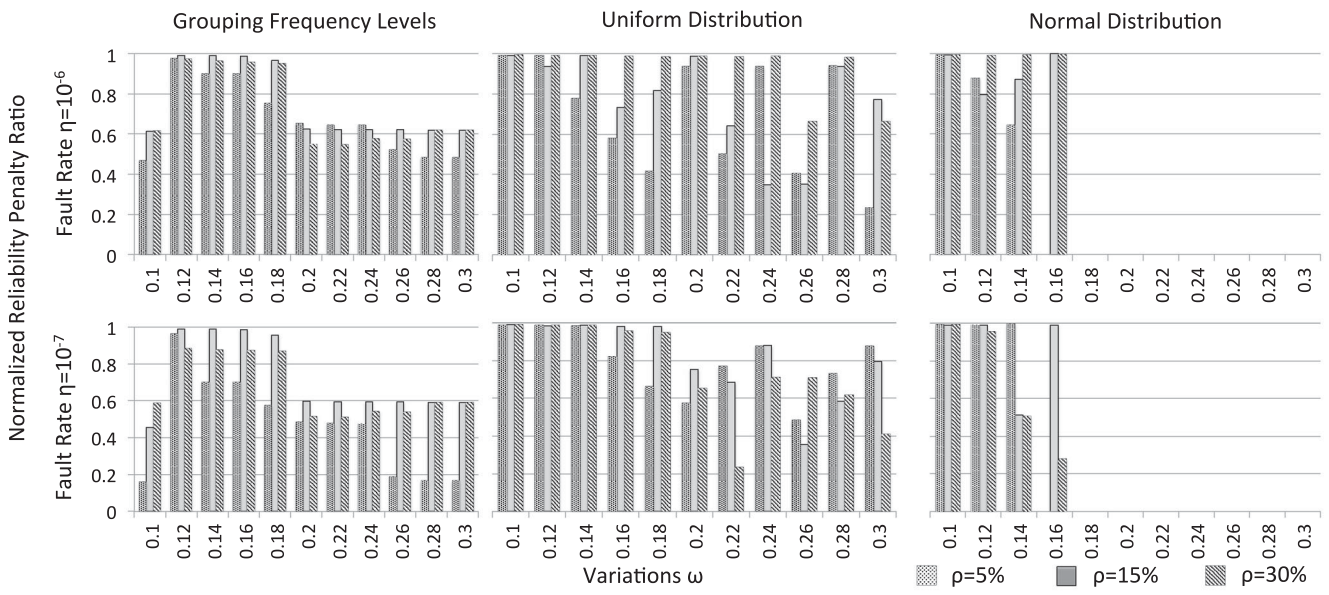


Fig. 10. Overall reliability penalty ratio for the complicated application under two different fault rates 10^{-6} and 10^{-7}.

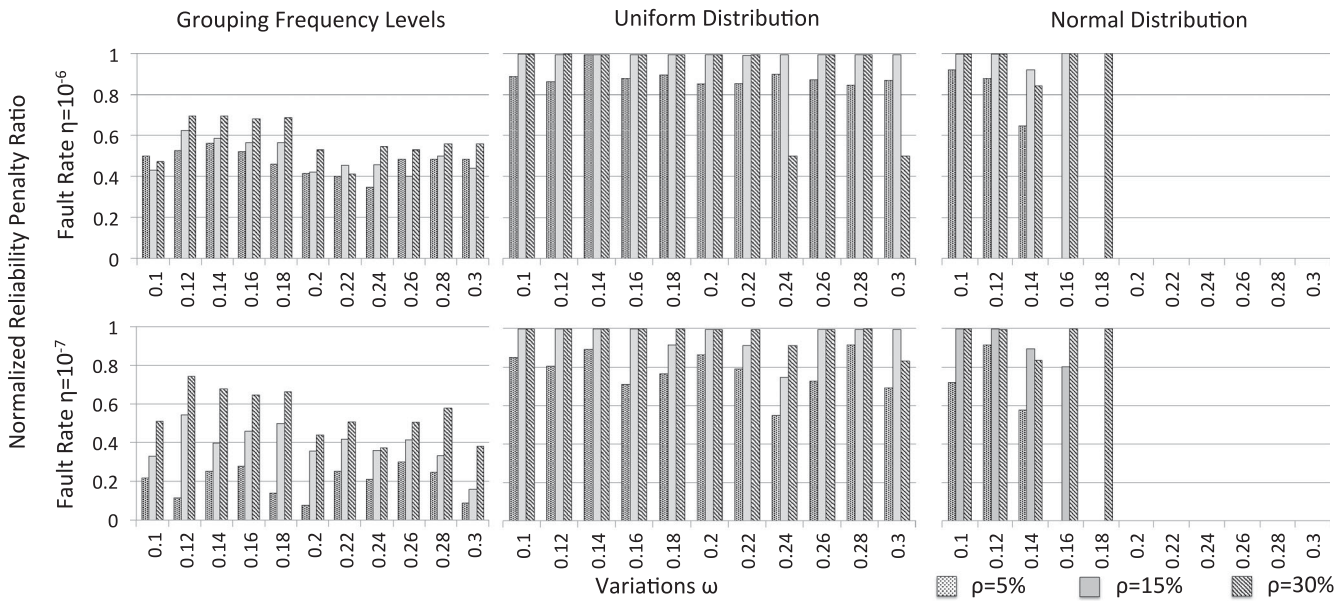


Fig. 11. Evaluation of the reliability penalty ratio with the communication overhead under different fault rates.

our approach is still applicable and outperforms the greedy approach in case the application is more complicated.

Simulation with communication overhead. By applying the communication model presented in Section 5, we reformulate the deadline miss rate of tasks in the preprocessing and adopt Algorithm 4, to obtain the simulation results in Fig. 11. At first, we can observe that the trends of results are similar as the previous case (without data dependency). Since the overhead of communication increases the hardness of meeting deadline, the feasible versions of tasks are reduced greatly, in which most of the tasks have a few choices to utilize the different frequencies of cores. However, the reliability improvement among all the different mappings is generally more than the case without dependency consideration.

Time consumption. To compare the timing overhead, here we report the average execution time for the experiments reported in Fig. 1 of supplementary material, available online, page and Fig. 10. As shown in Fig. 12, we can see that if the input number is as small as seven tasks, our method can still be efficient. However, when the input number is increased to 21 tasks, the overhead difference is more

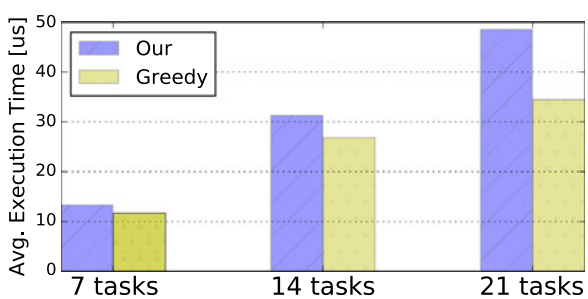


Fig. 12. Overhead between our approach and greedy mapping.

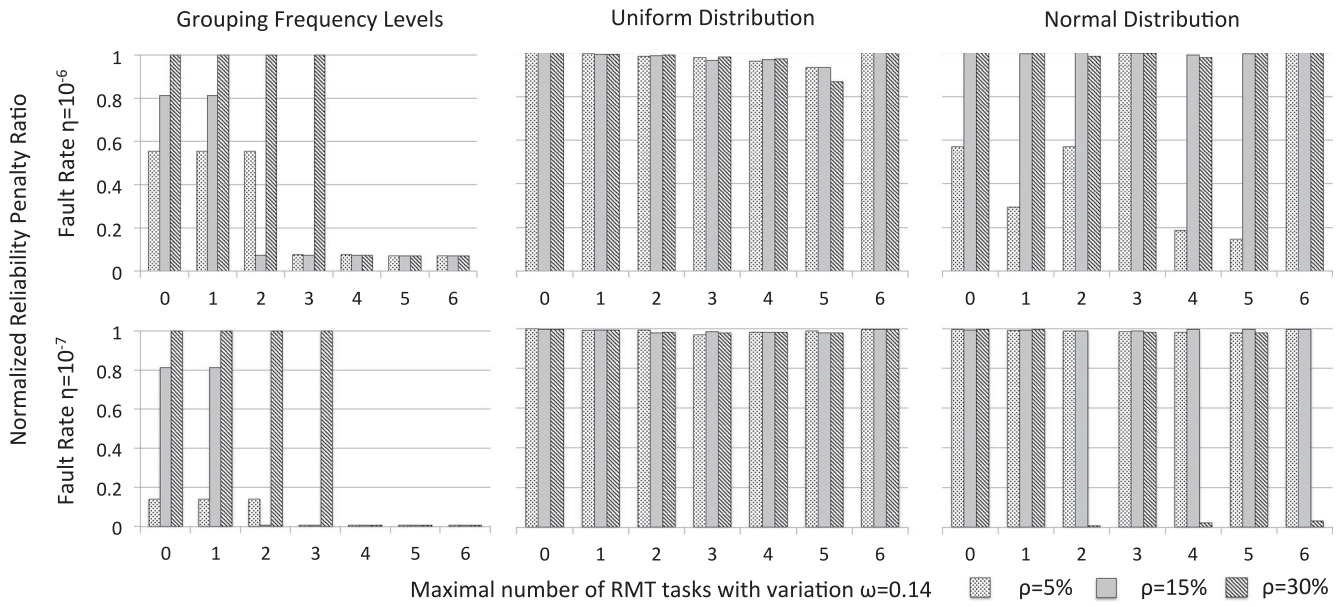


Fig. 13. Evaluation of the resulting execution modes with variation $\omega = 0.14$ under different fault rates.

obvious. As the greedy mapping sorts the tasks by the reliability penalties, the execution time is mainly dominated by the sorting algorithm. However, the time complexity of our approach is $O(N^3)$. When there are many tasks, some approximations are required to trade the execution time for the efficiency.

6.3 Simulation Results for Modes Adaptation

This section presents the results when the execution modes of tasks are not determined beforehand. To evaluate the effectiveness, we adopt Algorithm 3 to deliver the execution modes with different maximal numbers of RMT tasks, in which the maximal number of RMT tasks is determined by the number of available cores. Here we present the evaluation under different fault rates 10^{-6} and 10^{-7} with variation $\omega = 0.14$. As we are not aware of any other approaches of execution modes adaptation, we show the normalized reliability penalty ratios with our mapping approach.

As shown in Fig. 13, we can see that the trends in the charts with the delivered execution modes still follow our observation in Section 6.2. If the frequency variation among the cores is not negligible as in the case of grouping frequency levels, the proposed mapping approach perform well most of time. If the frequencies of cores are degraded smoothly as the case of uniform distribution, the improvement of overall reliability penalty is not significant. In the case of normal distribution, the improvement is still significant with the delivered execution modes when the tolerable miss rate is tighter, i.e., $\rho = 5\%$.

We also compare the resulting execution modes with the optimal execution modes for seven tasks, which is obtained by a brute-force search with factorial timing complexity. We observe that the task mapping and execution modes delivered by our proposed approaches are equivalent tp the optimal task mapping under the optimal execution modes.

7 Conclusions

In this paper, we have introduced reliability-driven mapping techniques to allocate the tasks onto a many-core processor by taking application vulnerability and performance heterogeneity into consideration. We show that a special case of the studied problem with homogeneous execution modes is equivalent to the minimum weight perfect bipartite matching problem, and an approach is developed to optimally handle heterogeneous execution modes. To consider communication and data dependencies, we provide a viable way to estimate the transfer overhead with a deterministic routing algorithm and show how it enhances our proposed mapping approaches. From the evaluation we conclude that our proposed approaches may improve greedy method drastically up to 80 percent when the frequency variation among the cores is not negligible. For different scenarios of chip frequency variation maps, the improvement on average may achieve 20 percent. As the time complexity of proposed approaches are polynomial time based, all can be triggered either in on-line reconfiguration for aging-induced effects or process variations, or offline configuration for heterogeneous architectures to pursue the dependable application design. For the implementation, the interactions between the compiler and the operating system are required. The reliable compilation help us exploit the resilience of tasks with varying execution time and vulnerabilities The proposed approaches needs to be implemented in the scheduler to determine the task mapping and the executing modes.

Acknowledgments

This work is supported in parts by the German Research Foundation (DFG) as part of the priority program "Dependable Embedded Systems" (SPP 1500 - spp1500.itec.kit.edu). Kuan-Hsun Chen is the corresponding author.

References

[1] R. Baumann, "Radiation-induced soft errors in advanced semiconductor technologies," *IEEE Trans. Device Mater. Rel.*, vol. 5, no. 3, pp. 305–316, Sep. 2005.

[2] J. Henkel, L. Bauer, N. Dutt, P. Gupta, S. Nassif, M. Shafique, M. Tahoori, and N. Wehn, "Reliable on-chip systems in the nano-era: Lessons learnt and future trends," in *Proc. 50th Annu. Des. Autom. Conf.*, 2013, pp. 99:1–99:10.

[3] S. S. Mukherjee, C. Weaver, J. Emer, S. K. Reinhardt, and T. Austin, "A systematic methodology to compute the architectural vulnerability factors for a high-performance microprocessor," in *Proc. 36th Annu. IEEE/ACM Int. Symp. Microarchit.*, 2003, p. 29.

[4] P. Shivakumar, M. Kistler, S. Keckler, D. Burger, and L. Alvisi, "Modeling the effect of technology trends on the soft error rate of combinational logic," in *Proc. Int. Conf. Dependable Syst. Netw.*, 2002, pp. 389–398.

[5] R. Vadlamani, J. Zhao, W. Burleson, and R. Tessier, "Multicore soft error rate stabilization using adaptive dual modular redundancy," in *Proc. Conf. Des., Autom. Test Eur.*, 2010, pp. 27–32.

[6] S. Rehman, A. Toma, F. Kriebel, M. Shafique, J.-J. Chen, and J. Henkel, "Reliable code generation and execution on unreliable hardware under joint functional and timing reliability considerations," in *Proc. IEEE 19th Real-Time Embedded Technol. Appl. Symp.*, 2013, pp. 273–282.

[7] S. Rehman, M. Shafique, F. Kriebel, and J. Henkel, "Reliable software for unreliable hardware: Embedded code generation aiming at reliability," in *Proc. 9th Int. Conf. Hardw./Softw. Codes. Syst. Synthesis*, 2011, pp. 237–246.

[8] M. Shafique, S. Rehman, P. V. Aceituno, and J. Henkel, "Exploiting program-level masking and error propagation for constrained reliability optimization," in *Proc. 50th Annu. Des. Autom. Conf.*, 2013, pp. 17:1–17:9.

[9] G. A. Reis, J. Chang, N. Vachharajani, R. Rangan, D. I. August, and S. S. Mukherjee, "Software-controlled fault tolerance," *ACM Trans. Archit. Code Optim.*, vol. 2, pp. 366–396, 2005.

[10] V. Izosimov, P. Pop, P. Eles, and Z. Peng, "Synthesis of fault-tolerant embedded systems with checkpointing and replication," in *Proc. 3rd IEEE Int. Workshop Electron. Des., Test Appl.*, 2006, pp. 440–447.

[11] J. Henkel, A. Herkersdorf, L. Bauer, T. Wild, M. Hubner, R. Pujari, A. Grudnitsky, J. Heisswolf, A. Zaib, B. Vogel, V. Lari, and S. Kobbe, "Invasive manycore architectures," in *Proc. 17th Asia South Pacific Des. Autom. Conf.*, 2012, pp. 193–200.

[12] J. Jahn, M. Faruque, and J. Henkel, "Carat: Context-aware runtime adaptive task migration for multi core architectures," in *Proc. Des., Autom. Test Eur. Conf. Exhib.*, 2011, pp. 1–6.

[13] T. Ebi, M. Faruque, and J. Henkel, "Tape: Thermal-aware agent-based power econom multi/many-core architectures," in *Proc. Int. Conf. Comput.-Aided Des.*, 2009, pp. 302–309.

[14] M. Al Faruque, R. Krist, and J. Henkel, "Adam: Run-time agent-based distributed application mapping for on-chip communication," in *Proc. 45th Annu. Des. Autom. Conf.*, 2008, pp. 760–765.

[15] T. Corporation. (2013). Tile-gx processor family [Online]. Available: http://www.tilera.com

[16] (2012). Intel xeon phi^{TM}product family [Online]. Available: http://www.intel.com/content/www/us/en/processors/xeon/xeon-phi-detail. html

[17] (2009). Intel single-chip cloud computer [Online]. Available: http://techresearch.intel.com/ProjectDetails.aspx?Id=1

[18] N. Tesla. (2013). Tesla processor family [Online]. Available: http://www.nvidia.com

[19] ITRS. (2011). System drivers [Online]. Available: http://www.itrs.net

[20] E. Rotenberg, "AR-SMT: A microarchitectural approach to fault tolerance in microprocessors," in *Proc. 29th Annu. Int. Symp. Fault-Tolerant Comput.*, 1999, p. 84.

[21] S. K. Reinhardt and S. S. Mukherjee, "Transient fault detection via simultaneous multithreading," in *Proc. 27th Annu. Int. Symp. Comput. Archit.*, 2000, pp. 25–36.

[22] J. Smolens, B. Gold, B. Falsafi, and J. Hoe, "Reunion: Complexity-effective multicore redundancy," in *Proc. 39th Annu. IEEE/ACM Int. Symp. Microarchit.*, 2006, pp. 223–234.

[23] S. S. Mukherjee, M. Kontz, and S. K. Reinhardt, "Detailed design and evaluation of redundant multithreading alternatives," in *Proc. Annu. Int. Symp. Comput. Archit.*, 2002, pp. 99–110.

[24] K. Bowman, S. Duvall, and J. Meindl, "Impact of die-to-die and within-die parameter fluctuations on the maximum clock frequency distribution for gigascale integration," *IEEE J. Solid-State Circuits*, vol. 37, no. 2, pp. 183–190, Feb. 2002.

[25] M. Shafique, S. Garg, J. Henkel, and D. Marculescu, "The EDA challenges in the dark silicon era: Temperature, reliability, and variability perspectives," in *Proc. 51st Annu. Des. Autom. Conf.*, 2014, pp. 1–6.

[26] ARM. (2013). big.little technology: The future of mobile [Online]. Available: http://www.arm.com/files/pdf/big_LITTLE_Technology_the_Futue_of_Mobile.pdf

[27] K. Kuhn, C. Kenyon, A. Kornfeld, M. Liu, A. Maheshwari, S. Weikai, S. Sivakumar, G. Taylor, P. VanDerVoorn, and K. Zawadzki, "Managing process variation in Intel's 45nm CMOS technology," *J. Intel. Technol.*, vol. 12, pp. 93–109, 2008.

[28] P. Gupta, Y. Agarwal, L. Dolecek, N. Dutt, R. Gupta, R. Kumar, S. Mitra, A. Nicolau, T. Rosing, M. Srivastava, S. Swanson, and D. Sylvester, "Underdesigned and opportunistic computing in presence of hardware variability," *IEEE Trans. Comput.-Aided Des. Integrated Circuits Syst.*, vol. 32, no. 1, pp. 8–23, Jan. 2013.

[29] J. Xiong, V. Zolotov, and L. He, "Robust extraction of spatial correlation," *IEEE Trans. Comput.-Aided Des. Integrated Circuits Syst.*, vol. 26, no. 4, pp. 619–631, Apr. 2007.

[30] S. Dighe, S. R. Vangal, P. Aseron, S. Kumar, T. Jacob, K. A. Bowman, J. Howard, J. Tschanz, V. Erraguntla, N. Borkar, V. De, and S. Borkar, "Within-die variation-aware dynamic-voltage-frequency-scaling with optimal core allocation and thread hopping for the 80-core teraflops processor," *IEEE J. Solid-State Circuits*, vol. 46, no. 1, pp. 184–193, Jan. 2011.

[31] I. Kadayif, M. Kandemir, and I. Kolcu, "Exploiting processor workload heterogeneity for reducing energy consumption in chip multiprocessors," in *Proc. Des., Autom. Test Eur. Conf. Exhib.*, 2004, pp. 1158–1163.

[32] K. Kang, S. Gangwal, S. P. Park, and R. Kaushik, "NBTI induced performance degradation in logic and memory circuits: How effectively can we approach a reliability solution," in *Proc. Asia South Pacific Des. Autom. Conf.*, 2008, pp. 726–731.

[33] A. Tiwari and J. Torrellas, "Facelift: Hiding and slowing down aging in multicores," in *Proc. 41st Annu. IEEE/ACM Int. Symp. Microarchit.*, 2008, pp. 129–140.

[34] C. R. Lefurgy, A. J. Drake, M. S. Floyd, M. S. Allen-Ware, B. Brock, J. A. Tierno, J. B. Carter, and R. W. Berry, "Active guardband management in power7+ to save energy and maintain reliability," *IEEE Micro*, vol. 33, no. 4, pp. 35–45, Jul./Aug. 2013.

[35] A. G. J. Abella and X. Vera, "Penelope: The NBTI-aware processor," in *Proc. 40th Annu. IEEE/ACM Int. Symp. Microarchit.*, 2007, pp. 85–96.

[36] R. Kumar, K. I. Farkas, N. P. Jouppi, P. Ranganathan, and D. M. Tullsen, "Single-ISA heterogeneous multi-core architectures: The potential for processor power reduction," in *Proc. 36th Annu. IEEE/ACM Int. Symp. Microarchit.*, 2003, p. 81.

[37] A. Benso, S. Chiusano, P. Prinetto, and L. Tagliaferri, "A C/C++ source-to-source compiler for dependable applications," in *Proc. Int. Conf. Dependable Syst. Netw.*, 2000, pp. 71–78.

[38] J. Xu, Q. Tan, and R. Shen, "The instruction scheduling for soft errors based on data flow analysis," in *Proc. IEEE Pacific Rim Int. Symp. Dependable Comput.*, Nov. 2009, pp. 372–378.

[39] S. Rehman, F. Kriebel, D. Sun, M. Shafique, and J. Henkel, "dtune: Leveraging reliable code generation for adaptive dependability tuning under process variation and aging-induced effects," in *Proc. 51st Annu. Des. Autom. Conf.*, 2014, pp. 1–6.

[40] C. Zhu, Z. Gu, R. Dick, and L. Shang, "Reliable multiprocessor system-on-chip synthesis," in *Proc. 5th IEEE/ACM/IFIP Int. Conf. Hardw./Softw. Codes. Syst. Synthesis*, 2007, pp. 239–244.

[41] A. Hartman, D. Thomas, and B. Meyer, "A case for lifetime-aware task mapping in embedded chip multiprocessors," in *Proc. IEEE/ACM/IFIP Int. Conf. Hardw./Softw. Codes. Syst. Synthesis*, 2010, pp. 145–154.

[42] B. Raghunathan, Y. Turakhia, S. Garg, and D. Marculescu, "Cherry-picking: Exploiting process variations in dark-silicon homogeneous chip multi-processors," in *Proc. Des., Autom. Test Eur. Conf. Exhib.*, 2013, pp. 39–44.

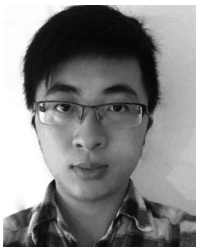
[43] S. Herbert, S. Garg, and D. Marculescu, "Exploiting process variability in voltage/frequency control," *IEEE Trans. Very Large Scale Integr. Syst.*, vol. 20, no. 8, pp. 1392–1404, Aug. 2012.

[44] S. Herbert and D. Marculescu, "Characterizing chip-multiprocessor variability-tolerance," in *Proc. Annu. Des. Autom. Conf.*, 2008, pp. 313–318.

[45] S. Rehman, M. Shafique, and J. Henkel, "Instruction scheduling for reliability-aware compilation," in *Proc. 49th Annu. Des. Autom. Conf.*, 2012, pp. 1288–1296.

[46] H. W. Kuhn, "The Hungarian method for the assignment problem," *Naval Res. Logistics Quarterly*, vol. 2, pp. 83–97, 1955.

[47] G. D. Micheli and L. Benini, *Networks on Chips: Technology and Tools*. Amsterdam, The Netherlands: Elsevier, 2006.
[48] S. Rehman, "Reliable software for unreliable hardware—a cross-layer approach," Ph.D. Thesis, Karlsruhe Institute of Technology, 2015.
[49] S. Rehman, K. Chen, F. Kriebel, A. Toma, M. Shafique, J. Chen, and J. Henkel, "Cross-layer software dependability on unreliable hardware," *IEEE Trans. Comput.*, vol. 65, no. 1, pp. 80–94, Jan. 2016.
[50] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, and R. B. Brown, "Mibench: A free, commercially representative embedded benchmark suite," in *Proc. IEEE Int. Workshop Workload Characterization*, 2001, pp. 3–14.
[51] S. Rehman, F. Kriebel, M. Shafique, and J. Henkel, "Reliability-driven software transformations for unreliable hardware," *IEEE Trans. Comput.-Aided Des. Integrated Circuits Syst.*, vol. 33, no. 1, pp. 1597–1610, Nov. 2014.
[52] J. Hu, S. Wang, and S. Ziavras, "In-register duplication: Exploiting narrow-width value for improving register file reliability," in *Proc. Int. Conf. Dependable Syst. Netw.*, 2006, pp. 281–290.
[53] L. Li, V. Degalahal, N. Vijaykrishnan, M. Kandemir, and M. Irwin, "Soft error and energy consumption interactions: A data cache perspective," in *Proc. Int. Symp. Low Power Electron. Des.*, 2004, pp. 132–137.
[54] R. Azevedo, S. Rigo, M. Bartholomeu, G. Araujo, C. C. de Araujo, and E. Barros, "The ArchC architecture description language and tools," *Int. J. Parallel Program.*, vol. 33, pp. 453–484, 2005.
[55] (2006, Sep.). Flux calculator [Online]. Available: http://www.seutest.com/cgi-bin/FluxCalculator.cgi. [Online]. Available: http://www.seutest.com/cgi-bin/FluxCalculator.cgi
[56] G. P. Saggese, N. J. Wang, Z. Kalbarczyk, S. J. Patel, and R. K. Iyer, "An experimental study of soft errors in microprocessors," *IEEE Micro*, vol. 25, no. 6, pp. 30–39, Nov./Dec. 2005.
[57] S. Garg and D. Marculescu, "System-level throughput analysis for process variation aware multiple voltage-frequency island designs," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 13, no. 4, pp. 59:1–59:25, 2008.



Kuan-Hsun Chen received the MSc degree in computer science from the National Tsing Hua University, Hsinchu, Taiwan, in 2013, and is currently working toward the PhD degree from the chair for design automation of embedded systems, TU-Dortmund, Germany. His current research interests include dependable computing, embedded systems, and reliability-aware resource management. He is a student member of the IEEE.

Jian-Jia Chen received the PhD degree from the Department of Computer Science and Information Engineering, National Taiwan University, Taiwan in 2006. He is a professor at the Department of Informatics in TU Dortmund University, Germany. He was a Junior professor at the Department of Informatics in the Karlsruhe Institute of Technology, Germany from May 2010 to March 2014. Between January 2008 and April 2010, he was a postdoc researcher at ETH Zurich, Switzerland. His research interests include real-time systems, embedded systems, energy-efficient scheduling, power-aware designs, temperature-aware scheduling, and distributed computing. He received Best Paper Awards at CODES+ISSS 2014, RTCSA 2005 and 2013, and SAC 2009. He has been involved with technical committees in many international conferences. He is a member of the IEEE.

Florian Kriebel received the MSc degree in computer science from the Karlsruhe Institute of Technology (KIT), Germany, in 2013 and is currently working toward the PhD degree from the chair for embedded systems, KIT, Germany. His current research interests include dependable computing, cross-layer reliability modeling, and optimization. He received the CODES+ISSS 2011 and 2015 Best Paper Award and two HiPEAC Paper Awards.

Semeen Rehman (S'11) received the PhD degree in computer science from the Karlsruhe Institute of Technology, Germany, in 2015. She is a postdoc researcher at Technical University Dresden, Germany. From 2005 to 2007, she was an information systems manager at the Mardan Surgical Centre Pvt. (Ltd.), Pakistan. Her current research interests include cross-layer reliability modeling and optimization covering various system layers like compiler and run-time system, approximate computing, and embedded systems. She received the CODES + ISSS 2011 and 2015 Best Paper Awards and several HiPEAC Paper Awards. She is a student member of the IEEE.

Muhammad Shafique (M'11) received the PhD degree in computer science from the Karlsruhe Institute of Technology (KIT), Germany, in 2011. He is currently a research group leader at the Chair for Embedded Systems, KIT. He has more than 10 years of research and development experience in power-/performance-efficient embedded systems in leading industrial and research organizations. He holds one US patent. His current research interests include design and architectures for embedded systems with focus on low power and reliability. He received the 2015 ACM/SIGDA Outstanding New Faculty Award, six gold medals, the CODES+ISSS 2011, 2014, and 2015 Best Paper Awards, AHS 2011 Best Paper Award, DATE 2008 Best Paper Award, DAC 2014 Designer Track Poster Award, ICCAD 2010 Best Paper Nomination, several HiPEAC Paper Awards, and the Best Masters Thesis Award. He is the TPC co-Chair of ESTIMedia 2015 and 2016 and has served on the TPC of several IEEE/ACM conferences like ICCAD and DATE. He is a member of the IEEE.

Jörg Henkel received the PhD degree from Braunschweig University with "Summa cum Laude". He is currently with the Karlsruhe Institute of Technology (KIT), Germany, where he is directing the Chair for Embedded Systems CES. Before, he was a senior research staff member at the NEC Laboratories in Princeton, NJ. He has/is organizing various embedded systems and low-power ACM/IEEE conferences/symposia as a general chair and program chair and was a guest editor on these topics in various Journals like the *IEEE Computer Magazine*. He was a program chair of CODES'01, RSP'02, ISLPED06, SIPS'08, CASES'09, Estimedia'11, VLSI Design'12, ICCAD12, PATMOS13, NOCS14 and served as a general chair for CODES'02, ISLPED09, Estimedia12, ICCAD13 and ESWeek'16. He is/has been a steering committee member of major conferences in the embedded systems field like at ICCAD, ESWeek, ISLPED, Codes+ISSS, CASES and is/has been an editorial board member of various journals like the *IEEE Transactions on Very Large Scale Integration*, *IEEE Transactions on Computer-Aided Design*, *IEEE Transactions on Multi-Scale Computing Systems*, *ACM Transactions on. Cyber-Physical Systems*, *Journal of Low Power Electronics* etc. In recent years, he has given around 10 keynotes at various international conferences primarily with focus on embedded systems dependability. He has given full/half-day tutorials at leading conferences like DAC, ICCAD, DATE etc. He received the 2008 DATE Best Paper Award, the 2009 IEEE/ACM William J. Mc Calla ICCAD Best Paper Award, the Codes+ISSS 2015, 2014, and 2011 Best Paper Awards, and the MaXentric Technologies AHS 2011 Best Paper Award as well as the DATE 2013 Best IP Award and the DAC 2014 Designer Track Best Poster Award. He is the chairman of the IEEE Computer Society, Germany Section, and was the editor-in-chief of the *ACM Transactions on Embedded Computing Systems* (ACM TECS) for two consecutive terms. He is an initiator and the coordinator of the German Research Foundation's (DFG) program on 'Dependable Embedded Systems' (SPP 1500). He is the site coordinator (Karlsruhe site) of the Three-University Collaborative Research Center on "Invasive Computing" (DFG TR89). He is the editor-in-chief of the *IEEE Design&Test Magazine* since January 2016. He holds 10 US patent and is a fellow of the IEEE.