Reconfigurable Fault-tolerance Mapping Method for Real-Time and Dependent Tasks on Flexible Adapt-NoC MPSoCs

Xinmei Li*, *Student Member, IEEE,* Lei Mo†, *Member, IEEE,* Tamim M. Al-Hasan‡, *Student Member, IEEE,* Angeliki Kritikakou§, *Member, IEEE,* Xiaojun Zhai‡, *Senior Member, IEEE,* Shibo He¶, *Senior Member, IEEE,* and Olivier Sentieys§, *Senior Member, IEEE*

***Abstract*—Multi-Processor Systems-on-Chip (MPSoCs) utilizing Network-on-Chip (NoC) architectures are widely adopted in domains such as autonomous driving, wearable devices, and the Internet of Things (IoT) due to their strengths in energy efficiency, reliability, and communication performance. However, NoCs are inherently susceptible to permanent faults in cores, links, and routers, which compromise system performance, energy consumption, and reliability. Most existing fault-tolerant approaches address only specific fault scenarios and often neglect broader impacts on system-level metrics, such as communication efficiency, energy consumption, and load distribution. In this work, we address the fault-tolerant task mapping problem under multiple permanent faults across different components. Using Adapt NoC as the target platform, we develop a permanent fault model that supports topology reconfiguration and formulate a mapping problem aimed at minimizing energy consumption and communication overhead while ensuring load balance. To solve this, we introduce the Fault-Tolerant Topology Generation and Task Mapping (FTTGTM) method, which comprises a topology generation strategy based on an enhanced Feasibility-Pump approach, a dynamic programming-based frequency allocation scheme, and real-time task scheduling strategies. Experimental results demonstrate that, compared to the existing fault-tolerant methods, FTTGTM handles multiple permanent faults in cores, links, and routers more effectively, achieving an average 67.7% energy consumption reduction, while ensuring reliable scheduling with low communication overhead.**

***Index Terms*—Flexible NoC Architecture, Permanent faults, Task Mapping, Topology Reconfigurable**

I. Introduction

The continuous evolution of System-on-Chip (SoC) technology has driven the transition from single-processor SoCs to Multi-Processor SoCs (MPSoCs) to accommodate the rapidly increasing computational demands [1], [2]. Early MPSoC designs commonly employed shared-bus interconnects. However, as the number of cores increased, bus-based communication suffered from limited bandwidth and poor scalability [3]. To address these limitations, Networks-on-Chips (NoCs) have emerged as a scalable and efficient interconnect paradigm [4], enabling parallel data transfer among cores and improving overall communication efficiency. The complex communication structure of NoCs makes them susceptible to permanent faults, which may arise from manufacturing defects, device aging and temperature, or environmental variations [5], [6]. These faults can lead to data loss or transmission failures, thereby posing a serious challenge to NoC reliability.

Permanent faults can occur in various NoC components, including cores, links, and routers [7]–[9]. Existing fault-tolerant techniques are typically classified by the type of faulty component. For example, to handle core faults, task reallocation [10]–[14] and redundancy-based schemes [15], [16] enable workload migration to healthy cores to sustain system functionality. For link and router faults, fault-tolerant routing algorithms [17]–[20] and router redundancy approaches [21], [22] can be employed to reroute packets and bypass defective components. Existing fault-tolerant schemes mainly address faults in specific components and rely on static redundancy without topology reconfiguration, offering limited adaptability to multiple faults. Moreover, system-level performance metrics, e.g., energy efficiency and real-time response, are not fully considered, leading to inefficient resource use and degraded communication performance.

To address these challenges, reconfigurable NoC architectures have been proposed, e.g., Adaptive NoC (Adapt NoC) [23], allowing dynamic reconfiguration of network topology in response to detected faults. By incorporating adaptable routers, adaptable links, and centralized links, Adapt NoC achieves a balance between fault tolerance and performance efficiency. This paper proposes a reconfigurable and fault-tolerant task allocation and scheduling framework for Adapt NoC. The proposed framework adjusts task mapping and communication patterns based on current network configuration, ensuring both reliability and performance sustainability in the presence of multiple permanent faults. Our main contributions are as follows:

(1) We propose a fault-tolerant model for Adapt NoC that reconfigures core-to-router connections and activates adaptable links to bypass permanent faults. The task mapping problem is formulated as a Mixed-Integer Nonlinear Programming (MINLP) model with objectives of minimizing energy consumption, communication latency, and achieving load balance.

(2) We develop a hybrid optimization framework, termed Fault-Tolerant Topology Generation and Task Mapping

* X. Li is with the School of Cyber Science and Engineering, Southeast University, 210096 Nanjing, China. E-mail: xinmeili@seu.edu.cn.

† L. Mo is with the Key Laboratory of Measurement and Control of CSE, Ministry of Education, School of Automation, Southeast University, 210096 Nanjing, China. E-mail: lmo@seu.edu.cn.

‡ T. M. Al-Hasan and X. Zhai are with the School of Computer Science and Electronic Engineering, University of Essex, Colchester CO4 3SQ, U.K., E-mails: tamim.almulla@essex.ac.uk, xzhai@essex.ac.uk.

§ A. Kritikakou and O. Sentieys are with the University of Rennes, INRIA, IRISA, CNRS, 35042 Rennes, France. E-mails: angeliki.kritikakou@irisa.fr, olivier.sentieys@irisa.fr.

¶ S. He is with the College of Control Science and Engineering, Zhejiang University, 310027 Hangzhou, China. E-mail: s18he@zju.edu.cn.

TABLE I
COMPARISON OF FAULT-TOLERANT TECHNIQUES IN NOC-BASED MPSOCS

Method Type	Objective Function	Adaptability to Multiple Faults	System Efficiency Considered	Limitations
Task Reallocation *(e.g., [10], [11], [14], [24], [25])*	Reliability [10]; energy [11] or communication cost [14], [24], [25] under fault-tolerant constraints	Partial (Core)	Limited (focus on reliability)	Lack of communication cost and energy optimization
Redundancy-Based Mapping *(e.g., [15], [16], [21], [22], [26])*	Remapping cost [15]; communication cost [16], [26]; or reliability [21], [22]	Low (Core, Router, Link) (static redundancy)	Not explicitly	High resource overhead; poor adaptability
Fault-Tolerant Routing *(e.g., [17]–[20])*	Communication efficiency under fault-tolerant constraints	Moderate (Link, Router)	Some (e.g., congestion-aware)	Lack of energy optimization; routing-level only
Reconfigurable NoC *(e.g., [8], [27]–[30])*	Reliability [8], [27]; reliability and communication efficiency [28]–[30]	High (Link, Router)	Some (energy/latency)	Task mapping and scheduling usually not integrated
Proposed: FTTGTM (This Work)	Energy efficiency, communication latency, and load balance	High (multi-fault) (Core, Router, Link)	Yes (energy, delay, load balance)	Offline; no reconfiguration overhead modeling

(FTTGTM), which includes: a) a fault-tolerant topology generation and task mapping strategy based on the enhanced Feasibility-Pump (FP) method with low computational complexity; b) a dynamic programming-based frequency allocation strategy for energy optimization; c) a real-time task scheduling strategy under system constraints.

(3) Simulation results demonstrate that FTTGTM achieves high-quality task mapping and topology generation with linear running time and only a 26.8% average reduction in solution quality compared to the optimal. It also reduces average energy consumption by up to 50.6% compared to the existing fault-tolerant methods while lowering communication overhead.

The remainder of this paper is organized as follows: Section II presents relevant background and literature review; Section III introduces the system and fault models; Section IV formulates the fault-tolerant task mapping problem; Section V details the proposed FTTGTM method; Section VI evaluates its performance; and Section VII concludes the paper.

II. BACKGROUND AND RELATED WORK

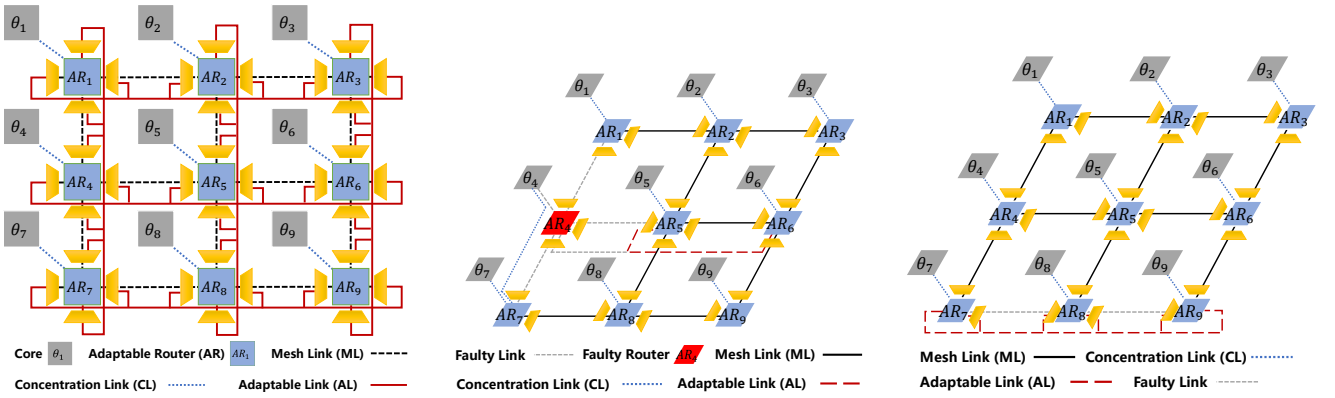
As NoC architectures become more complex and densely integrated, permanent faults due to manufacturing imperfections, aging, and environmental factors pose serious threats to system reliability and performance. Numerous fault-tolerant approaches have been developed to mitigate these challenges, broadly categorized into task reallocation, redundancy-based designs, fault-tolerant routing, and reconfigurable NoC architectures.

Task reallocation techniques enhance system reliability by dynamically assigning workloads to functional components. Fu et al. [10] proposed an ant colony optimization-based algorithm for fault-tolerant task allocation in the presence of core faults. Naghavi et al. [11] explored task replication and reallocation to tolerate core faults, duplicating high-criticality tasks and applying reliability-aware Dynamic Voltage/Frequency Scaling (DVFS) to save energy while preserving schedulability. However, these approaches often overlook the impact on system performance and communication overhead. To address this, Kumar et al. [25] proposed graph-based models that optimize both energy efficiency and performance under core faults. Cheng et al. [24] utilized active replication to schedule task backups across cores before deadlines, thereby minimizing communication overhead and improving system reliability under multiple core faults. Kadri et al. [14] developed a reinforcement learning-based mapping framework that dynamically adapts to faults while maintaining high throughput.

Redundancy-based fault tolerance employs spare components, such as cores, routers, and links, to replace failed components. Wu et al. [15] used spare cores and a row-column shifting algorithm to minimize communication delay and congestion during fault recovery. Bhanu and Soumya [16] utilized Integer Linear Programming (ILP) and Particle Swarm Optimization (PSO) for spare-core mapping in mesh-of-tree NoCs, reducing latency and energy consumption. Moreover, Chang et al. [21] implemented router-level redundancy by placing backup routers at strategic locations. Xu et al. [22] proposed a design with redundant routers and links to provide alternate paths and reduce latency. Bhanu et al. [26] tackled link faults via PSO-guided spare link placement, along with an application mapping heuristic and table-based routing. It improves latency, throughput, and energy efficiency, but ignores the faults in cores or routers.

Fault-tolerant routing is another key strategy that reroutes traffic around faulty components while maintaining throughput. Charif et al. [17] proposed turn-prohibition rules for deadlock-free routing around faults. Zhang et al. [18] introduced a deterministic routing algorithm that precomputes disjoint paths offline and then updates local routing tables to maintain connectivity. Romanov et al. [19] developed a self-organizing scheme where routers adjust their forwarding rules by local fault states. To balance reliability and congestion, Liu et al. [20] employed Q-learning to make fault-aware real-time routing decisions.

Unlike static redundancy or routing-based methods, reconfigurable NoC designs enable dynamic topology adaptation in response to faults. Xu et al. [27] introduced a reliable NoC with multifunctional reconfigurable channels for adaptive path and link configuration. Narayanasamy et al. [28] proposed topology reconfiguration algorithms to minimize energy and latency under permanent link and router faults. Ouyang et al. [29] combined adaptive routing with topology migration to cope with router-level faults. Khalil et al. [8] presented a dynamic reconfiguration strategy that uses virtual channels and crossbar reallocation to maintain fault-free data transmission



(a) 3×3 mesh-based Adapt NoC. (b) Adaptable router fault example. (c) Mesh link fault example.

Fig. 1. Adapt NoC topology and permanent fault examples.

while reducing energy and latency. Zhou et al. [30] proposed Reconfigurable Bidirectional Links (ReBLs), which dynamically adapt to and mitigate permanent link faults through hardware (links) reconfiguration.

Compared with existing works, which typically focus on specific fault scenarios (e.g., link- or router-level faults) or optimize a single objective such as reliability or communication efficiency, this paper considers a more comprehensive system-level joint design problem. Specifically, we address multi-component permanent faults across cores, links, and routers, while jointly optimizing energy consumption, communication latency, and load balancing through integrated topology reconfiguration, frequency allocation, and task scheduling. Table I summarizes the key features and limitations of existing fault-tolerant methods, highlighting the novelty of our proposed FTTGTM framework in addressing multi-component permanent faults with system-level efficiency.

III. Permanent Fault-tolerant Model

1) System Model: The target platform is a two-dimensional Adapt NoC architecture organized as a mesh topology connecting heterogeneous cores, adaptable routers, and interconnecting links [23]. The platform contains $M = N_M \times N_M$ cores $\Theta = \{\theta_1, \ldots, \theta_M\}$ and M adaptable routers $\mathcal{AR} = \{AR_1, \ldots, AR_M\}$, indexed by $\mathcal{M} = \{1, \ldots, M\}$. Fig. 1(a) shows a 3×3 Adapt NoC, which consists of four elements: Adaptable Routers (ARs), Adaptable Links (ALs), Concentration Links (CLs), and Mesh Links (MLs). Adaptable links interconnect routers in both horizontal and vertical directions to enable flexible, high-throughput communication and fault bypassing, while mesh links provide the default, regular communication. Concentration links allow multiple cores to connect to a single router. These features enable the NoC to dynamically reconfigure its topology to bypass faulty components, mitigate congestion, and maintain performance under permanent faults.

Due to router limitations, each router can only connect a limited number of cores, denoted by N_{limit}. In the fault-free scenario, many-to-one core-to-router mappings may result in underutilized routers and inefficient resource usage. To prevent this, we adopt a one-to-one mapping between the core and the router. Each core θ_k connects to router AR_k via a dedicated concentration link $CL_{k,k}$, forming the concentration link set $\mathcal{CL} = \{CL_{1,1}, \ldots, CL_{M,M}\}$. The mesh link set is denoted as $\mathcal{ML} = \{ML_{1,2}, \ldots, ML_{N_M-1,N_M}\}$, and the adaptable link set as $\mathcal{AL} = \{AL_{1,2}, \ldots, AL_{N_M-1,N_M}\}$.

The architecture supports DVFS [31], enabling energy optimization by adjusting the operating Voltage/Frequency (V/F) of cores and routers. Each core θ_k operates at one of N_P V/F levels, $\mathcal{VF}_P = \{(v^P_{k,1}, f^P_{k,1}), \ldots, (v^P_{k,N_P}, f^P_{k,N_P})\}$, while each link supports N_L V/F levels, $\mathcal{VF}_L = \{(v^L_1, f^L_1), \ldots, (v^L_{N_L}, f^L_{N_L})\}$, and all links operate at a uniform V/F level. In static scheduling, reconfiguration and DVFS operations are relatively infrequent compared to task execution, and their overheads can be incorporated into task execution costs.

The system's energy consumption primarily contains: **Computing energy** (E_{comp}): consumed by cores executing tasks; **Communication energy** (E_{comm}): includes energy usage of links and routers. We define $P^{\text{comp}}_{k,l}$ as the power consumption of core θ_k with $(v^P_{k,l}, f^P_{k,l})$, and P^{link}_g as the power usage of a link at (v^L_g, f^L_g). The energy to transmit one bit of data via a router is E_{Rbit}, dependent on the router architecture of NoC [32].

2) Task Model: The real-time application is modeled as a Directed Acyclic Graph (DAG) comprising N dependent tasks, denoted as a six-tuple $\{\mathcal{T}, \mathcal{E}, \mathcal{S}, \mathcal{W}, \mathcal{D}, H\}$. For instance, tasks in control algorithms such as EKF or PID used in drone systems can be represented using DAGs [33]. Here: $\mathcal{T} = \{\tau_1, \ldots, \tau_N\}$: the set of real-time tasks; $\mathcal{E}$: directed edges representing data dependencies; edge $e_{i,j}$ indicates τ_j depends on τ_i; $\mathcal{S}$: weights $s_{i,j}$ assigned to edges $e_{i,j}$, denoting the data volume to transmit from τ_i to τ_j; $\mathcal{W}$: set of Worst Case Execution Cycles (WCEC), where $\omega_i \in \mathcal{W}$ is the WCEC of task τ_i; $\mathcal{D}$: deadlines, with each task τ_i required to finish before $D_i \in \mathcal{D}$; H: the overall scheduling horizon. If the tasks have different periods, the scheduling horizon H can be set as the least common multiple of all task periods. Due to dependencies, a task τ_i cannot begin execution until it has received all required data from its predecessors, while successors of τ_i can only begin once τ_i completes, and its output data has been transmitted. In addition, the start time of τ_i should be larger than its arrival time.

3) Permanent Fault Model: Permanent faults are assumed to be detected in advance [18], and we mainly focus on a fault-tolerant design that addresses failures in cores, routers, and communication links. To simplify the model, permanent faults in cores and routers are abstracted as faults in their associated links [21]. Accordingly, each link can be in one of three states: *enabled*, *disabled*, or *faulty*.

We define the state matrix of concentration links as $A = [a_{k,m}]_{M\times M}$, where $a_{k,m} \in \{1, 0, -1\}$ denotes the state of link $CL_{k,m}$ as enabled, disabled, or faulty, respectively. Similarly, $B = [b_{m,n}]_{M\times M}$ and $C = [c_{m,n}]_{M\times M}$ represent the state matrices for mesh links and adaptable links, respectively. For example, when a permanent fault affects router AR_4 (as shown in Fig. 1(b)), a set of associated links becomes faulty: the concentration link $CL_{4,4}$, mesh links $\{ML_{1,4}, ML_{4,1}, ML_{4,5}, ML_{5,4}, ML_{4,7}, ML_{7,4}\}$, and adaptable links $\{AL_{1,4}, AL_{4,1}, AL_{4,7}, AL_{7,4}, AL_{4,5}, AL_{5,4}, AL_{4,6}, AL_{6,4}\}$. The states of these links are updated to -1. Note that the above matrix description is a general mathematical formulation. The proposed system can flexibly construct and operate under different topologies and connectivity patterns.

Due to the reconfigurable architecture of the Adapt NoC, core-to-router mappings can transition from one-to-one to many-to-one, thereby enhancing computational resource utilization. For example, as illustrated in Fig. 1(b), core θ_4 may be reassigned to a functioning router such as AR_7. Additionally, consider the scenario depicted in Fig. 1(c), where permanent faults occur in mesh links $\{ML_{7,8}, ML_{8,7}, ML_{8,9}, ML_{9,8}\}$. The system compensates by enabling adaptable links $\{AL_{7,8}, AL_{8,7}, AL_{8,9}, AL_{9,8}, AL_{7,9}, AL_{9,7}\}$. Consequently, the mesh link states are updated from $1 \to -1$, and the corresponding adaptable link states are updated from $0 \to 1$.

4) Topology Generation Model: Topology generation in Adapt NoC involves determining core-to-router and router-to-router interconnect structures by enabling or disabling concentration, mesh, or adaptable links. To simplify the problem, we assume that each core θ_k can be connected to at most one adaptable router AR_m. Under fault-free conditions, Adapt NoC maintains a regular 2D mesh structure. In the worst-case scenario, if all concentration links connected to a core θ_k are faulty (i.e., $a_{k,1} = \ldots = a_{k,M} = -1$), the core becomes isolated. To prevent invalid connections, we impose the following constraint:

$$\sum\nolimits_{m\in\mathcal{M}} y_{k,m} \le \min\left\{\sum\nolimits_{m\in\mathcal{M}}(a_{k,m}+1), 1\right\},\ \forall k \in \mathcal{M}, \tag{1}$$

where $y_{k,m} = 1$ indicates that core θ_k connects to router AR_m via concentration link $CL_{k,m}$; otherwise, $y_{k,m} = 0$. The term $\sum_m (a_{k,m} + 1)$ accounts for the available (non-faulty) links. If all links are faulty, this sum becomes zero, ensuring $\sum_m y_{k,m} = 0$ and disabling the connection.

Additionally, to account for the limited connection capacity of routers, the number of cores connected to any adaptable router AR_m must not exceed N_{limit}:

$$\sum\nolimits_{k\in\mathcal{M}} y_{k,m} \le N_{\text{limit}},\ \forall m \in \mathcal{M}. \tag{2}$$

We model inter-router communication by examining the in-degree and out-degree of data flows for each router. Let $\eta_{i,j,m,n} = 1$ if the communication for edge $e_{i,j}$ passes through routers AR_m and AR_n; otherwise, $\eta_{i,j,m,n} = 0$. For example, consider tasks τ_i and τ_j mapped to cores θ_2 and θ_9, respectively. The communication path for edge $e_{i,j}$ originates at AR_2 (source) and terminates at AR_9 (destination). Intermediate routers like AR_3 and AR_6 lie on the path. The flow balance condition for each router AR_n is expressed as:

$$\sum_{m:L_{n,m}\in\mathcal{L}} \eta_{i,j,n,m} - \sum_{m:L_{m,n}\in\mathcal{L}} \eta_{i,j,m,n} = \begin{cases} 1, & n = s, \\ -1, & n = d, \\ 0, & n \ne s, d, \end{cases} \tag{3}$$

where s and d denote the source and destination routers. $\mathcal{L} = \{L_{s,d} \mid L_{s,d} \in \mathcal{ML} \cup \mathcal{AL},\ b_{m,n} \ne -1,\ c_{p,q} \ne -1\}$ is the set of active links (either mesh or adaptable) not affected by permanent faults. Note that $L_{s,d}$ denotes the set of all paths from s to d following the minimal dimensional-order routing.

To guide fault-aware topology generation, we define the transmission overheads for each link type under a given V/F level (v_g^L, f_g^L). $TC^C_{m,n,g}$: time to transmit one data unit via concentration link $CL_{m,n}$; $TC^M_{m,n,g}$: via mesh link $ML_{m,n}$; $TC^A_{m,n,g}$: via adaptable link $AL_{m,n}$. The overheads of faulty links are set to $+\infty$. The topology generation model operates offline, determining the fault-tolerant task mapping and NoC configuration in advance without requiring runtime remapping or hardware reconfiguration. To minimize energy and communication overheads, under multiple faults in cores, links, and routers, the topology generation model reconfigures Adapt NoC into mesh-like, torus-like, or irregular structures, through the topology-related parameters, e.g., $y_{k,m}$ and $\eta_{i,j,n,m}$.

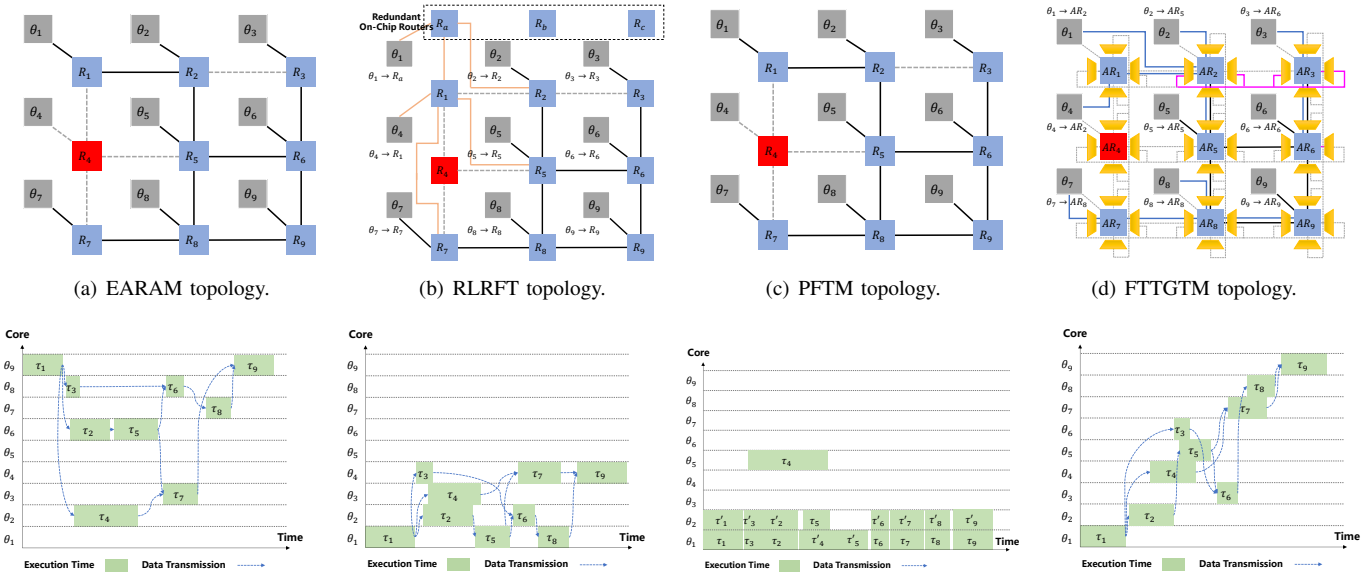
5) Illustration Example: To validate the effectiveness of the proposed FTTGTM method, we compare it against three existing fault-tolerant approaches: the Energy-Aware and Reliability-Aware Mapping method (EARAM) [25], the Router-Level Redundancy Fault-Tolerant method (RLRFT) [21], and the Permanent Fault-Tolerant Mapping method (PFTM) [24]. We consider a benchmark task set derived from the LU decomposition application [34], consisting of $N = 9$ dependent tasks. Faults are randomly generated [21]. The fault scenario includes a permanent failure in router R_5 and mesh link $CL_{2,3}$.

EARAM addresses only core-level faults, reassigning tasks from faulty to operational cores while minimizing communication overhead. Fig. 2(a) shows the resulting task mapping and scheduling, which yields a total energy consumption of $E_{\text{total}} = 231$ mJ and total communication delay of $t^{\text{comm}}_{\text{total}} = 0.54$ s.

RLRFT, in contrast, incorporates structural redundancy by introducing spare routers (R_a, R_b, and R_c) into the topology, as depicted in Fig. 2(b). This structure tolerates one permanent router fault per column. The method applies the Heterogeneous Earliest Finish Time (HEFT) algorithm for scheduling (see Fig. 2(f)), resulting in $E_{\text{total}} = 345$ mJ and $t^{\text{comm}}_{\text{total}} = 0.48$ s.

PFTM employs an active replication strategy, assigning both primary and backup tasks to separate cores to enhance reliability. As shown in Fig. 2(g), this method achieves a significantly reduced communication delay of $t^{\text{comm}}_{\text{total}} = 0.09$ s, but at the cost of higher energy consumption, with $E_{\text{total}} = 582$ mJ.

Fig. 2(h) presents the task allocation for the LU decomposition case on a 3×3 Adapt NoC. Using the proposed FTTGTM method, we generate a fault-tolerant topology and task mapping (see Fig. 2(d)). The method achieves a communication delay of $t^{\text{comm}}_{\text{total}} = 0.25$ s, which is significantly lower than both EARAM and RLRFT, while reducing energy consumption to $E_{\text{total}} = 175$ mJ, therefore outperforming all comparison methods. Fault



(a) EARAM topology. (b) RLRFT topology. (c) PFTM topology. (d) FTTGTM topology.

(e) Task scheduling of EARAM. (f) Task scheduling of RLRFT. (g) Task scheduling of PFTM. (h) Task scheduling of FTTGTM.

Fig. 2. Topology and task scheduling comparisons of EARAM, RLRFT, PFTM, and our method.

TABLE II
MAIN NOTATIONS USED IN PROBLEM FORMULATIONS

Binary Variables	
$x_{i,k}$	$=1$ if task τ_i is assigned to core θ_k, else $x_{i,k}=0$.
$y_{k,m}$	$=1$ if core θ_k is connected to router AR_m through link $CL_{k,m}$, else $y_{k,m}=0$.
$\eta_{i,j,m,n}$	$=1$ if the communication for edge $e_{i,j}$ passes through link $L_{m,n}$, else $\eta_{i,j,m,n}=0$.
$v_{m,n}$	$=1$ when using the mesh link to connect routers AR_m and AR_n, else $v_{m,n}=0$.
$c^P_{k,l}$	$=1$ if core θ_k operates with the V/F $(v^P_{k,l}, f^P_{k,l})$, else $c^P_{k,l}=0$.
c^L_g	$=1$ if all the links operates with the V/F (v^L_g, f^L_g), else $c^L_g=0$.
$p_{i,j}$	$=1$ if task τ_i starts execution before task τ_j is completed, else $p_{i,j}=0$.
Continuous Variables	
$U_{\max}$	the maximum core utilization rate.
ts_i (te_i)	the start (end) time of executing task τ_i.
$ts_{i,j}$ $(te_{i,j})$	the start (end) time of message transmission for edge $e_{i,j}$.
$t^{comm}_{i,j,g}$	the communication time with the V/F (v^L_g, f^L_g) for edge $e_{i,j}$.

positions are integrated into the fault-tolerance task mapping problem, enabling the problem solution to perform topology generation, task allocation, and scheduling with minimal energy and communication overhead.

IV. Proposed Task Mapping Problem Formulation

We aim to develop a fault-tolerant scheme that jointly performs topology generation and task scheduling when permanent faults affect components such as cores, routers, and links in an Adapt NoC. Our scheme will not change the hardware of Adapt NoC. We formulate the problem as a MINLP model, with objectives to minimize total communication delay and system energy consumption while achieving load balance. Let $\mathcal{N} \triangleq \{1,\ldots,N\}$ denote the set of tasks, $\mathcal{N}_P \triangleq \{1,\ldots,N_P\}$ the core V/F levels, and $\mathcal{N}_L \triangleq \{1,\ldots,N_L\}$ the link V/F levels. Table II summarizes the key variables used. Based on these definitions, we present the system constraints as follows:

A. Constraint Formulation

1) Task Allocation Constraints: Each task τ_i must be assigned to exactly one core for execution:

$$\sum\nolimits_{k\in\mathcal{M}} x_{i,k} = 1, \quad \forall i \in \mathcal{N}. \tag{4}$$

2) Topology Generation Constraints: Due to potential faults in cores or routers, core-to-router and router-to-router interconnect topologies must be reconstructed accordingly. We incorporate (1) and (2) to verify the effectiveness of the concentration link. Based on the set of enabled links $\mathcal{L}$, we express the routing path constraint for each communication edge $e_{i,j}$ as:

$$\sum\nolimits_{m:L_{n,m}\in\mathcal{L}} \eta_{i,j,n,m} - \sum\nolimits_{m:L_{m,n}\in\mathcal{L}} \eta_{i,j,m,n} = \sum\nolimits_{p\in\mathcal{M}} x_{i,p}y_{p,n} - \sum\nolimits_{q\in\mathcal{M}} x_{j,q}y_{q,n}, \forall e_{i,j}\in\mathcal{E}, \forall n \in \mathcal{M}, \tag{5}$$

$$\left[\sum\nolimits_{g\in\mathcal{N}_L} (TC^M_{m,n,g} - TC^A_{m,n,g})c^L_g\right] v_{m,n} \le (1-\eta_{i,j,m,n})Z, \quad \forall e_{i,j}\in\mathcal{E},\ \forall m\neq n\in\mathcal{M}, \tag{6}$$

where Z is a large positive constant. (5) enforces flow conservation and routing correctness for source, destination, and intermediate routers based on task-to-core and core-to-router mappings, as detailed in Appendix A. (6) selects the router interconnection (mesh or adaptable link) that offers minimum communication cost based on link availability and V/F configuration.

3) Frequency Assignment Constraints: To reduce energy consumption using DVFS, each core is assumed to operate at a single fixed V/F level for all assigned tasks, while all links operate at a uniform fixed V/F level. The operating V/F level for each core and the overall link system is restricted as follows:

$$\sum\nolimits_{l\in\mathcal{N}_P} c^P_{k,l} = 1, \quad \forall k \in \mathcal{M}, \tag{7}$$

$$\sum\nolimits_{g\in\mathcal{N}_L} c^L_g = 1. \tag{8}$$

4) Task Dependent Constraints: For dependent tasks, data transfer must occur in order. The timing constraints for each communication edge $e_{i,j}$ are defined by:

$$te_i \le ts_{i,j},\ te_{i,j} = ts_{i,j} + \sum\nolimits_{g\in\mathcal{N}_L} c_g^L t_{i,j,g}^{comm} \le ts_j,\ \forall e_{i,j} \in \mathcal{E}. \tag{9}$$

The communication time $t_{i,j,g}^{comm}$ includes router-to-router and core-to-router delays and is given as:

$$\forall e_{i,j} \in \mathcal{E},\ \forall g \in \mathcal{N}_L,\quad t_{i,j,g}^{comm} = \underbrace{\sum\nolimits_{m,n\in\mathcal{M}} \left[v_{m,n} TC_{m,n,g}^{M} + (1-v_{m,n}) TC_{m,n,g}^{A}\right] \eta_{i,j,m,n} s_{i,j}}_{\text{(a) Router-to-router transmission}} + \underbrace{\sum\nolimits_{k,m\in\mathcal{M}} \left(TC_{k,m,g}^{C} x_{i,k} y_{k,m} + TC_{k,m,g}^{C} x_{j,k} y_{k,m}\right) s_{i,j}}_{\text{(b) Core-to-router transmission}}. \tag{10}$$

5) Task Non-overlapping Constraints: To ensure non-preemptive execution on a single core, we introduce ordering constraints among independent tasks assigned to the same core:

$$\begin{aligned} &te_i \le ts_j + (2 - x_{i,k} - x_{j,k})H + (1 - p_{i,j})H, \\ &te_j \le ts_i + (2 - x_{i,k} - x_{j,k})H + p_{i,j}H, \\ &\forall i \ne j \in \mathcal{N},\ q_{i,j} = 0,\ \forall k \in \mathcal{M}. \end{aligned} \tag{11}$$

These constraints enforce sequential execution when τ_i and τ_j are mapped to the same core, controlled by the binary ordering variable $p_{i,j}$, as detailed in Appendix B.

6) Real-Time Constraints: Each task τ_j must complete execution before its deadline D_j:

$$te_j = ts_j + \sum\nolimits_{k\in\mathcal{M}} \sum\nolimits_{l\in\mathcal{N}_P} x_{j,k} c_{k,l}^P \frac{\omega_j}{\lambda_{j,k} f_{k,l}^P} \le D_j, \quad \forall j \in \mathcal{N}, \tag{12}$$

where $\lambda_{j,k}$ is the task-to-core heterogeneity factor indicating task execution efficiency [35], as the same task assigned to different cores leads to different task execution time and energy.

7) Utilization Balance Constraints: To achieve load balance, we define U_k as the utilization of core θ_k and constrain it relative to the system's maximum utilization $U_{\max}$:

$$U_k = \frac{\sum_{i\in\mathcal{N}} \sum_{l\in\mathcal{N}_P} x_{i,k} c_{k,l}^P \frac{\omega_i}{\lambda_{i,k} f_{k,l}^P}}{H} \le U_{\max},\ \forall k \in \mathcal{M}, \tag{13}$$

B. Objective Function

We aim to jointly optimize i) total communication delay, ii) total energy consumption, and iii) load balancing among cores. The total communication overhead is computed as:

$$t_{total}^{comm} = \sum\nolimits_{e_{i,j}\in\mathcal{E}} \sum\nolimits_{g\in\mathcal{N}_L} c_g^L t_{i,j,g}^{comm}. \tag{14}$$

The system energy consumption includes both computation and communication components:

$$E_{comm} = \underbrace{\sum\nolimits_{e_{i,j}\in\mathcal{E}} \sum\nolimits_{g\in\mathcal{N}_L} c_g^L t_{i,j,g}^{comm} P_g^{link}}_{\text{(a) link energy consumption}} + \underbrace{\sum\nolimits_{e_{i,j}\in\mathcal{E}} \left(\sum\nolimits_{m\in\mathcal{M}} \sum\nolimits_{n\in\mathcal{N}} \eta_{i,j,m,n} + 1\right) s_{i,j} E_{Rbit}}_{\text{(b) router energy consumption}}, \tag{16}$$

where E_{Rbit} is a constant [36], and $\sum_{m,n} \eta_{i,j,m,n} + 1$ estimates the number of routers traversed for edge $e_{i,j}$. The total energy is $E_{total} = E_{comp} + E_{comm}$.

We introduce weight parameters μ_1, μ_2, and μ_3 to balance the three optimization goals: communication overhead, energy consumption, and utilization, which satisfy $\mu_1, \mu_2, \mu_3 \in [0, 1]$ and $\mu_1 + \mu_2 + \mu_3 = 1$. The overall objective function becomes:

$$\min \left(\mu_1 \bar{t}_{total}^{comm} + \mu_2 \bar{E}_{total} + \mu_3 U_{\max}\right), \tag{17}$$

where $\bar{E}_{total} = E_{total}/E_{\max}$ and $\bar{t}_{total}^{comm} = t_{comm}^{total}/t_{comm}^{\max}$ are normalized metrics. $E_{\max}$ and $t_{comm}^{\max}$ are the upper bounds of energy and delay, given by the worst-case energy consumption and system latency for normalization.

C. Problem Formulation and Linearization

For clarity, we define the variable vectors and tensors used throughout the problem as follows: $\boldsymbol{x} = [x_{i,k}]_{N\times M}$: task-to-core mapping; $\boldsymbol{y} = [y_{k,m}]_{M\times M}$: core-to-router mapping; $\boldsymbol{v} = [v_{m,n}]_{M\times M}$: router link type (mesh/adaptable); $\boldsymbol{\eta} = [\eta_{i,j,m,n}]_{N\times N\times M\times M}$: routing indicator; $\boldsymbol{c^P} = [c_{k,l}^P]_{M\times N_P}$: core V/F levels; $\boldsymbol{c^L} = [c_g^L]_{1\times N_L}$: link V/F levels; $\boldsymbol{p} = [p_{i,j}]_{N\times N}$: task execution order indicator; $\boldsymbol{ts^P} = [ts_i]_{1\times N}$, $\boldsymbol{ts^L} = [ts_{i,j}]_{N\times N}$: task and communication start times; $\boldsymbol{t^{comm}} = [t_{i,j,g}^{comm}]_{N\times N\times N_L}$: communication time; $U = U_{\max}$: maximum core utilization.

Given all previously defined constraints and objective, the permanent fault-tolerant task mapping problem is as in (18):

$$\min_{\substack{\boldsymbol{x},\boldsymbol{y},\boldsymbol{\eta},\boldsymbol{v},\boldsymbol{c^P},\boldsymbol{c^L},\\ \boldsymbol{p},\boldsymbol{ts^P},\boldsymbol{ts^L},\boldsymbol{t^{comm}},U}} \left(\mu_1 \bar{t}_{total}^{comm} + \mu_2 \bar{E}_{total} + \mu_3 U_{\max}\right) \tag{18}$$

$$\text{s.t.} \begin{cases} (1), (2), (4) - (13), \\ x_{i,k}, y_{k,m}, \eta_{i,j,m,n}, v_{m,n}, c_{k,l}^P, c_g^L, p_{i,j} \in \{0, 1\}, \\ U_{\max}, ts_i, ts_{i,j}, t_{i,j,g}^{comm} \ge 0, \\ \forall i, j \in \mathcal{N},\ \forall k, m, n \in \mathcal{M},\ \forall l \in \mathcal{N}_P,\ \forall g \in \mathcal{N}_L. \end{cases}$$

Problem (18) includes nonlinear expressions involving products of binary variables (e.g., $x_{i,k}y_{k,m}$, $x_{i,k}c_{k,l}^P$, $v_{m,n}c_g^L$, and $v_{m,n}\eta_{i,j,m,n}$) and binary–continuous combinations (e.g., $c_g^L t_{i,j,g}^{comm}$). As such, it is a Mixed-Integer Nonlinear Programming (MINLP) problem, which is known to be $\mathcal{NP}$-hard.

To solve problem (18) efficiently with tools like Gurobi™ or IBM® ILOG® CPLEX®, we apply linearization as in [37], and reformulate it into an equivalent Mixed-Integer Linear Programming (MILP) model, as detailed in Appendix C.

Remark 4.1: DVFS switching overhead depends on the initial and target V/F levels [38], e.g., f_s and f_e. The switching time and energy can be modeled as $t^{\text{switch}}(f_s, f_e) = t_X \frac{f_e - f_s}{f_e}$ and $E^{\text{switch}}(f_s, f_e) = P_s^{\text{switch}} \cdot t^{\text{switch}}(f_s, f_e)$, respectively, where t_X and P_s^{switch} are the corresponding transition time and power parameters. On this basis, we can obtain switching overheads regarding links and cores, and update time and energy constraints accordingly, e.g., (9), (12), (15), and (16). Using a similar method [39], we can model and analyze the costs of topology reconfiguration.

V. Proposed Fault-tolerant Task Mapping Method

To address (18) efficiently, we propose a low-complexity Fault-Tolerant Topology Generation and Task Mapping (FTTGTM) method that efficiently handles permanent faults across cores, routers, and links. Our method comprises three key components: 1) Fault-tolerant topology generation strategy; 2) Frequency allocation strategy; 3) Task scheduling strategy.

1) Fault-tolerant Topology Generation Strategy: As task-to-core allocation and network topology directly influence system performance, energy, and communication costs, FTTGTM first addresses the joint design of topology generation and task allocation. This involves determining variables such as task-to-core mapping $x_{i,k}$, core-to-router connection $y_{k,m}$, link routing $\eta_{i,j,m,n}$, and mesh/adaptable link selection $v_{m,n}$.

For tractability, we fix the operating V/F levels of cores and links to $c_{k,l}^{P*}$ and c_g^{L*}, respectively. Given known $\eta_{i,j,m,n}$, the link-type selection term $[\sum_{g\in\mathcal{N}_L}(TC_{m,n,g}^{M}-TC_{m,n,g}^{A})c_g^L]v_{m,n}$ in constraint (6) becomes computable, allowing us to omit (6) at this stage. Furthermore, to linearize the nonlinearity in expressions such as $x_{i,k}y_{k,m}$ (appearing in (14) and (16)), we adopt the auxiliary binary variable $z_{i,k,m}$ and corresponding linear constraints. Thus, the problem can be formulated as:

$$\min_{\substack{\boldsymbol{x},\boldsymbol{y},\boldsymbol{\eta},\\ \boldsymbol{z},U}} \left[\mu_1 \bar{t}_{total}^{comm}\left(c_g^{L*}\right) + \mu_2 \bar{E}_{total}\left(c_{k,l}^{P*}, c_g^{L*}\right) + \mu_3 U_{\max}\right] \quad (19)$$

$$\text{s.t.} \begin{cases} (1),(2),(4), \text{linear constraints introduced by } z_{i,k,m}, \\ \sum_{m:L_{n,m}\in\mathcal{L}} \eta_{i,j,n,m} - \sum_{m:L_{m,n}\in\mathcal{L}} \eta_{i,j,m,n} = \\ \quad \sum_{p\in\mathcal{M}} z_{i,p,n} - \sum_{q\in\mathcal{M}} z_{j,q,n},\ \forall e_{i,j}\in\mathcal{E}, \\ U_k = \left(\sum_{i\in\mathcal{N}}\sum_{l\in\mathcal{N}_P} x_{i,k} c_{k,l}^{P*} \dfrac{\omega_i}{\lambda_{i,k} f_{k,l}^{P}}\right)/H \le U_{\max}, \\ x_{i,k}, y_{k,m}, \eta_{i,j,m,n}, z_{i,k,m} \in \{0,1\},\ U_{\max}\ge 0. \end{cases}$$

To solve problem (19) efficiently, we design an improved Feasibility Pump (FP) algorithm [40], modified to enhance both convergence and solution quality. Let $x^{(n)} = \{x_{i,k}, y_{k,m}, \eta_{i,j,m,n}, z_{i,k,m}, U_{\max}\}$ denote the solution in the n-th iteration. Let $z = c^T x$ be the objective function, and $\hat{x}^{(n)}$ be the integer-rounded version of $x^{(n)}$. The Hamming distance between relaxed and rounded solutions is:

$$\Delta(x^{(n)}, \hat{x}^{(n)}) = \sum\nolimits_{j\in\mathcal{I},\hat{x}_j=1}(1-x_j^*) + \sum\nolimits_{j\in\mathcal{I},\hat{x}_j=0} x_j^*, \quad (20)$$

where $\mathcal{I}$ denotes the set of integer (binary) variables.

We introduce two key enhancements:

(1) **Objective Combination:** To improve integer solution quality, we define a combined objective function incorporating both the original cost and distance: $\Delta_a(x,\hat{x}) = (1-a_n)\Delta(x,\hat{x})/\|\Delta\| + a_n c^T x/\|z\|$, where $a_n \in [0,1]$ is an iteration-dependent trade-off factor. Smaller a_n favors feasibility; larger values improve objective quality.

(2) **Constraint Strengthening:** To reduce the iteration count, we add an upper bound constraint $c^T x \le UB_n$ based on:

$$UB_n = b_n z^* + (1-b_n)\Delta_a^*(x,\hat{x}), \quad (21)$$

where $b_n \in (0,1)$, z^* is the objective value from the relaxed problem, and Δ_a^* is the evaluated distance at iteration n. As n increases, UB_n tightens, accelerating convergence.

By relaxing binary variables into $[0,1]$, we transform (19) into a Linear Programming (LP) problem, referred to as the FP-relaxation. According to [41], if $\Delta(x^*,\hat{x}) = 0$, then x^* is a feasible integer solution. The FP iteration proceeds as follows:

Algorithm 1: Fault-Tolerant Topology Generation

Input : Link state matrix A, mesh link state matrix B, adaptable link state matrix C, core frequency allocation $c_{k,l}^{P*}$, and link frequency allocation c_g^{L*}.
Output: Solution $x = \{x_{i,k}, y_{k,m}, \eta_{i,j,m,n}, z_{i,k,m}, U_{\max}\}$.

1 Solve the linear relaxation problem of (19) to obtain the initial solution $x^{(0)*}$ and the objective function value z^*;
2 Round $x_j \in x^{(0)*} (j\in\mathcal{I})$ to obtain the rounded solution $\hat{x}^{(0)}$;
3 Initialize $n=0$, and set $a_0 = a$ and $b_0 = b$;
4 **while** $\Delta(x^{(n)},\hat{x}^{(n)}) > 0$ *and* $n < n_{\max}$ **do**
5 Update $a_n = a_0/n$ and $b_n = b_0$;
6 Calculate UB_n through (21);
7 Solve the FP problem to obtain the solution $x^{(n)*}$ and calculate the distance function $\Delta(x^{(n)},\hat{x}^{(n)})$;
8 **if** $\Delta(x^{(n)},\hat{x}^{(n)}) = 0$ **then**
9 Solution is found, and return $x^{(n)*}$;
10 **else**
11 Round $x_j \in x^{(n)*}$ $(j\in\mathcal{I})$ to obtain the rounded solution $\hat{x}^{(n)*}$, and set $\hat{x}^{(n+1)} = \hat{x}^{(n)*}$;
12 **if** $\hat{x}^{(n+1)} = \hat{x}^{(n)}$ *and* $n < n_{\max}$ **then**
13 Select n_f integer variables from $\hat{x}^{(n+1)}$, invert their values, and obtain $\hat{x}^{(n+1)}$;
14 **end**
15 Update $n \to n+1$;
16 **end**
17 **end**
18 **if** $n = n_{\max}+1$ *and* $\Delta(x,\hat{x}) > 0$ **then**
19 Substitute $x_{i,k}^* \in x^{(n)*}$ into problem (19), and solve it to obtain the solution $x^{(n+1)*}$;
20 **end**

(1) We begin by solving the linear relaxed problem (19) and rounding the resulting solution (Lines 1–3), yielding the initial solution $x^{(0)*}$ with objective value z^*. Subsequently, we round the binary variables $x_j \in x^{(0)*}$ $(j\in\mathcal{I})$ to obtain $\hat{x}^{(0)}$, and initialize the iteration count n and factors a_0, b_0.

(2) We iteratively solve the FP problem using an improved FP method with a combined objective and strengthened constraints (Lines 4–17). In the n-th iteration, we update factors $a_n = a_0/n$ and $b_n = b_0$. Note that a_n decreases with iterations, prioritizing solution quality initially and then fast, feasible integer solutions. We then calculate the new upper bound UB_n using (21) and solve the FP problem to get $x^{(n)*}$ and the distance $\Delta(x^{(n)},\hat{x}^{(n)})$. If $\Delta(x^{(n)},\hat{x}^{(n)}) = 0$, a feasible integer solution for (19) is found. Otherwise, we round continuous variables $x_j \in x^{(n)*}$ $(j\in\mathcal{I})$ to obtain $\hat{x}^{(n)*}$, setting $\hat{x}^{(n+1)} = \hat{x}^{(n)*}$. If $\hat{x}^{(n+1)} = \hat{x}^{(n)}$, we randomly invert n_f binary variables in $\hat{x}^{(n+1)}$ to avoid cycles. We then increment $n \to n+1$ and continue if $\Delta(x^{(n)},\hat{x}^{(n)}) > 0$ or $n < n_{\max}$.

(3) We assess the solution's feasibility after iterations (Lines 18–20). If $n = n_{\max}+1$ and $\Delta(x^{(n)},\hat{x}^{(n)}) > 0$, a feasible integer solution for (19) was not found. The iterations stop, and we take the integer part $x_j^* \in x^{(n)*}$ $(j\in\mathcal{I})$. Substituting x_j^* into (19), we calculate the maximum utilization $U_{\max}$ using (13), as task allocation $x_{i,k}$ and core V/F $c_{k,l}^P$ are known. Consequently, (19) is transformed into a smaller-scale Integer Linear Programming (ILP) problem, which we solve to obtain a feasible integer solution $x^{(n+1)*}$.

2) Frequency Allocation Strategy: To further reduce energy consumption while meeting real-time constraints and minimiz-

Algorithm 2: Frequency Allocation

Input : Task allocation variable $x^*_{i,k}$, core-to-router connection variable $y^*_{k,m}$, router-to-router connection variable $\eta^*_{i,j,m,n}$, and communication time $t^{comm*}_{i,j,g}$.

Output: Frequency allocation variables of cores and links $c^{P*}_{k,l}$ and c^{L*}_{k}.

1 Sort tasks in $\mathcal{T}$ based on the dependencies as $\hat{\mathcal{T}}$;
2 Calculate $g_i(c^P_{k,l}, c^L_g)$ for each task τ_i with (23);
3 Initialize $n = 0$ and $F_0(c^P_{k,l^*}, c^L_g) = 0$;
4 **for** *each* $\tau_i \in \hat{\mathcal{T}}$ **do**
5 Update $n \to n + 1$;
6 Solve (24) to obtain the optimal decision V_n;
7 Calculate $F_n(c^P_{k,l^*}, c^L_g)(g \in \mathcal{N}_L)$ with (24) and V_n;
8 **end**
9 Set $c^{L*}_g = 1(g = g^*)$ and $c^{L*}_g = 0(g \neq g^* \in \mathcal{N}_L)$ based on V_N;
10 **for** *each* $k \in \mathcal{M}$ **do**
11 Solve $l^* = \arg\min_{n \in \mathcal{N}, x_{i,k}=1} F_n(c^P_{k,l}, c^L_{g^*})$;
12 Update $c^{P*}_{k,l} = 1(l = l^*)$ and $c^{P*}_{k,l} = 0(l \neq l^* \in \mathcal{N}_P)$;
13 **end**

ing communication overhead, optimizing the operating V/F of cores and links is necessary. When problem (19) is solved, we can substitute task allocation variable $x^*_{i,k}$, core-to-router connection variable $y^*_{k,m}$, router-to-router connection variable $\eta^*_{i,j,m,n}$, and link allocation variable $v^*_{m,n}$ obtained in the previous subsection into the original problem (18). Note that once the value of $x^*_{i,k}$ is determined, both task allocation and core workloads are fixed. To reduce computational complexity, the utilization balance constraints (13) are omitted. Hence, the sub-problem of optimizing energy consumption and communication overhead is given by

$$\min_{\substack{\boldsymbol{c^P}, \boldsymbol{c^L}, \boldsymbol{p}, \\ \boldsymbol{ts^P}, \boldsymbol{ts^L}}} \left(\mu_1 \bar{t}^{comm*}_{total} + \mu_2 \bar{E}^*_{total}\right) \tag{22}$$

$$\text{s.t.} \begin{cases} (7), (8), (11),\ te_i \leq ts_{i,j}, \\ te_{i,j} = ts_{i,j} + \sum_{g \in \mathcal{N}_L} c^L_g t^{comm*}_{i,j,g} \leq ts_j,\ \forall e_{i,j} \in \mathcal{E}, \\ te_j = ts_j + \sum_{k \in \mathcal{M}} \sum_{l \in \mathcal{N}_P} x^*_{j,k} c^P_{k,l} \frac{\omega_j}{\lambda_{j,k} f^P_{k,l}} \leq D_j, \\ c^P_{k,l}, c^L_g, p_{i,j} \in \{0, 1\},\ ts_i, ts_{i,j} \geq 0, \end{cases}$$

where $t^{comm*}_{i,j,g} = t^{comm}_{i,j,g}(x^*_{i,k}, y^*_{k,m}, v^*_{m,n}, \eta^*_{i,j,m,n})$ is the communication time for edge $e_{i,j}$ at V/F (v^L_g, f^L_g), calculated through (10) by using the obtained values $x^*_{i,k}$, $y^*_{k,m}$, $v^*_{m,n}$, and $\eta^*_{i,j,m,n}$. Similarly, $\bar{E}^*_{total} = \bar{E}_{total}(x^*_{i,k}, \eta^*_{i,j,m,n}, t^{comm*}_{i,j,g})$ and $\bar{t}^{comm*}_{total} = \bar{t}^{comm}_{total}(t^{comm*}_{i,j,g})$ are the normalized communication overhead and energy consumption, respectively, by using the obtained values $x^*_{i,k}$ and $\eta^*_{i,j,m,n}$.

According to (22), the frequency allocation of cores and links directly affects the scheduling results, as well as the communication and energy consumption overheads. Thus, we propose a frequency allocation strategy based on dynamic programming to optimize communication and energy consumption overheads, summarized in Algorithm 2. The specific steps are as follows.

(1) We sort the tasks $\tau_i \in \mathcal{T}$ according to task dependency $e_{i,j} \in \mathcal{E}$, and obtain the sorted task set $\hat{\mathcal{T}}$ (Lines 1–3). Since each core can execute tasks with different operating V/F levels, and all links in Adapt NoC operate at a single V/F throughout the scheduling period, we treat the frequency allocation sub-problem (22) for cores as a multi-stage decision problem. In the n-th stage, the frequency allocation strategy assigns the V/F $(v^P_{k,l}, f^P_{k,l})$ for task τ_i executed on core θ_k and the V/F (v^L_g, f^L_g) for links, respectively, while optimizing energy consumption and communication overhead to obtain the decision $V_i = \{c^{P*}_{k,l}, c^{L*}_g\}$, where the overhead function of task τ_i is given by

$$g_i(c^P_{k,l}, c^L_g) = \mu_0 \underbrace{\sum\nolimits_{k \in \mathcal{M}} x^*_{i,k} c^P_{k,l} \frac{\omega_i P^{comp}_{k,l}}{\lambda_{i,k} f^P_{k,l}}}_{\text{(a)}} + (1 - \mu_0) \underbrace{c^L_g t^{comm}_{i,g}}_{\text{(b)}}, \tag{23}$$

where $\mu_0 \in [0, 1]$ is an adjusting factor that determines the proportion of energy consumption and communication time in the overhead function, and $t^{comm}_{i,g} = \sum_{e_{j,i} \in \mathcal{E}} t^{comm*}_{j,i,g}$ is the sum of the communication time between task τ_i and its predecessor tasks. (23)-(a) is the energy consumption of executing task τ_i, and (23)-(b) is the communication overhead between the task τ_i and its predecessor tasks.

(2) We apply the dynamic programming scheme to achieve frequency allocation (Lines 4–9), where we use the recursive-based scheme to sequentially determine the optimal frequency allocation variables $V_i = \{c^P_{k,l^*}, c^L_{g^*}\}$ for each task $\tau_i \in \hat{\mathcal{T}}$. As we have N tasks, there exist N iterations. In the n-th $(n \leq N)$ iteration, we assume that in the previous $(n-1)$-th iteration, the frequency allocation of tasks $\{\tau_1, \ldots, \tau_{i-1}\}$ has completed, and we aim to determine the V/F used to execute task $\tau_i \in \hat{\mathcal{T}}$, to minimize the cost of frequency allocation:

$$F^*_n(c^P_{k,l^*}, c^L_{g^*}) = \min_{\substack{l \in \mathcal{N}_P, \\ g \in \mathcal{N}_L}} \left\{F_{n-1}(c^P_{k,l^*}, c^L_g) + g_i(c^P_{k,l}, c^L_g)\right\}, \tag{24}$$

where $F_n(c^P_{k,l}, c^L_g)$ is the cost of frequency allocation with the core V/F $(v^P_{k,l}, f^P_{k,l})$ and the link V/F (v^L_g, f^L_g) in the n-th stage. According to (24), we minimize the sum of frequency allocation costs, including the link cost $F_{n-1}(c^P_{k,l^*}, c^L_g)$ for the previous $(n-1)$ iterations and the overhead $g_i(c^P_{k,l}, c^L_g)$ for executing task τ_i, and obtain the optimal frequency allocation $V_i = \{c^P_{k,l^*}, c^L_{g^*}\}$ of task τ_i. Note that the initial link cost $F_0(c^P_{k,l^*}, c^L_g)$ is set to 0.

(3) After N iterations, we obtain frequency allocation decisions $\{V_1, \ldots, V_N\}$ for tasks $\{\tau_1, \ldots, \tau_N\}$. We have to adjust frequency allocation results (Lines 10–13), as the obtained results $\{V_1, \ldots, V_N\}$ may violate the frequency assignment constraints (7) and (8). For the link frequency allocation decision $c^L_{g^*}$, we select the value of the N-th iteration (the last iteration) as the final result, since the solution found by dynamic programming converges gradually. On the other hand, when calculating core frequency allocation decision c^P_{k,l^*}, we consider task-level DVFS, i.e., each task has a specific execution frequency. However, in constraints (7), we consider core-level DVFS, i.e., each core uses the same frequency to handle the assigned tasks. Since each core θ_k may execute multiple tasks $\tau_i \in \mathcal{N}$, where $x_{i,k} = 1$, we set $l^* = \arg\min_{n \in \mathcal{N}, x_{i,k}=1} F_n(c^P_{k,l}, c^L_{g^*})$ for core θ_k to execute the assigned tasks, to make sure that the cost is minimal.

3) Task Scheduling Strategy: The previous fault-tolerant topology generation and frequency allocation processes can determine the variables of task allocation $x^*_{i,k}$, core-to-router connection $y^*_{k,m}$, router-to-router connection $\eta^*_{i,j,m,n}$, link selection $v^*_{m,n}$, core frequency allocation $c^{P*}_{k,l}$, and link frequency

Algorithm 3: Task Scheduling

Input : Sorted task set $\hat{\mathcal{T}}$, task allocation $x^*_{i,k}$, core frequency allocation $c^{P*}_{k,l}$, link frequency allocation c^{L*}_g, communication time $t^{comm*}_{i,j,g}$.
Output: Start/end execution time ts^*_i/te^*_i, start/end communication time $ts^*_{i,j}/te^*_{i,j}$, frequency allocation variables of cores $c^{P*}_{k,l}$ and links c^{L*}_k.

```
Get scheduling results according to Algorithm 4;
for each τ_i ∈ T̂ do
    while te_i > D_i do
        Set c^{P*}_{k,l*} = 1 → c^{P*}_{k,l*+1} = 1 (k ∈ M);
        Update the scheduling results through Algorithm 4;
    end
end
```

Algorithm 4: Scheduling Time Calculation

Input : Sorted task set $\hat{\mathcal{T}}$, task allocation variable $x^*_{i,k}$, core frequency allocation variable $c^{P*}_{k,l}$, link frequency allocation variable c^{L*}_g, communication time $t^{comm*}_{i,j,g}$.
Output: Start/end execution time ts^*_i/te^*_i, and start/end communication time $ts^*_{i,j}/te^*_{i,j}$.

```
Initialize ts_i = 0 and ts_{i,j} = 0, and set End(θ_k) = 0 (k ∈ M);
for each τ_j ∈ T̂ do
    if task τ_i is the initial task then
        Set ts_j = 0, and apply (26) to calculate te_j;
    else
        Calculate ts_j and ts_{i,j} through (25) and (26);
        Update End(θ_k) = te_j(x^*_{j,k} = 1);
    end
end
```

allocation c^{L*}_g. On this basis, for task τ_i in the sorted task set $\hat{\mathcal{T}}$ and edge $e_{i,j} \in \mathcal{E}$, the start execution and communication times ts_i and $ts_{i,j}$ can be calculated by (9) and (12), as follows:

$$ts_j = \max_{e_{i,j}\in\mathcal{E}} \left\{ \underbrace{ts_{i,j} + \sum\nolimits_{g\in\mathcal{N}_L} c^{L*}_g t^{comm*}_{i,j,g}}_{\text{(a)}}, \underbrace{\sum\nolimits_{k\in\mathcal{M}} x^*_{j,k} End(\theta_k)}_{\text{(b)}} \right\}, \tag{25}$$

$$ts_{i,j} = te_i = ts_i + \sum\nolimits_{k\in\mathcal{M}} \sum\nolimits_{l\in\mathcal{N}_P} \frac{x^*_{i,k} c^{P*}_{k,l} \omega_i}{\lambda_{i,k} f^P_{k,l}}, \tag{26}$$

where $End(\theta_k)$ is the end time of the current task executed by core θ_k, (25)-(a) is the end communication time of edge $e_{i,j}$, and (25)-(b) is the end time to execute tasks on core $theta_k$. Due to dependencies between tasks, we consider that the start execution time of task τ_j is the maximum value between the end communication time of τ_j's predecessors and the end execution time of the core executing τ_j. Note that the start execution time of task τ_j is set to 0 if task τ_j is an entry task. The start communication time $ts_{i,j}$ of each edge $e_{i,j}$ is calculated by (26), which is set to the end execution time te_i of predecessor τ_i.

Note that the above task scheduling strategy ignores the real-time constraint (12). Therefore, we need to adjust the scheduling results that violate the real-time constraint. For example, if $te_i > D_i$ $(\exists i \in \mathcal{N})$, we increase the core V/F to execute task τ_i to reduce task execution time and update the core frequency allocation variables to $c^{P*}_{k,l*} = 1 \rightarrow c^{P*}_{k,l*+1} = 1$ $(k \in \mathcal{M})$. The above task scheduling strategy is shown in Algorithms 3 and 4, and finally, we can obtain task scheduling results that satisfy real-time constraints.

4) Complexity Analysis: Algorithm 1 applies an iterative scheme, solving the LP-relaxation of problem (19) up to $n_{\max}$ times in the worst case. If no feasible integer solution is found, an additional ILP must be solved as a fallback. Therefore, the overall computational complexity of Algorithm 1 is $O(n_{\max} + 1)$. Next, Algorithm 2 performs task sorting based on data dependencies in $\mathcal{T}$, which incurs a computational complexity of $O(N^2)$. Then, it assigns frequencies to cores and links per task τ_i, and adjusts the frequency levels for each core θ_k, resulting in an additional cost of $O(N + M)$. Thus, the complexity of Algorithm 2 is $O(N^2 + N + M)$.

Finally, Algorithm 3 handles real-time task scheduling using Algorithm 4 to compute execution time parameters for each task τ_i. This operation has a complexity of $O(N)$. If any tasks fail to meet real-time constraints, scheduling adjustments are triggered. In the worst-case scenario, frequency allocation for each core may be modified up to $M \times N_P$ times, where N_P is the number of available V/F levels per core. In the worst case, the complexity of Algorithm 3 is $O(M \times N_P \times N)$.

Combining the above, the overall computational complexity of the proposed FTTGTM method is $O(N^2 + (N_P \times M + 2) \times N + M + n_{\max} + 1)$. Given that the number of tasks N is typically much larger than the number of cores M and the number of V/F levels N_P, i.e., $N \gg M$ and $N \gg N_P$, the complexity then simplifies to $O(N^2 + N + n_{\max})$.

TABLE III
EXPERIMENT SET-UP

Platform Model		
Core Type 1	$f^P_{k,l}$ (GHz)	2.10, 1.81, 1.53, 1.26, 1.01
	$P^{comp}_{k,l}$ (mW)	654, 500, 371, 267, 185
Core Type 2	$f^P_{k,l}$ (GHz)	0.4, 0.3, 0.2, 0.1, 0.1
	$P^{comp}_{k,l}$ (mW)	292, 157, 87, 32, 32
Link	$f_{l,k}$ (MHz)	200, 400, 600, 800, 1000
	P^{link}_k (mW)	160, 180, 520, 880, 1600
Platform	$b_\omega = 32$ bps	$E_{Rbit} = 0.01$ nJ
	$M = 3 \times 3$	$N_P = 5$, $N_L = 5$
Task Model		
LU ($N = 9$), GE ($N = 14$), FFT ($N = 15$), LE ($N = 16$), MW ($N = 24$), synthetic DAGs ($N \in [10, 50]$), $\omega_i \in [4 \times 10^7, 6 \times 10^8]$		
$D_i = \delta(D_{\max} - D_{\min,i}) + D_{\min,i}$, $H = D_{\max}$		
$D_{\max} = \sum_{i\in CPT} \left[\frac{\omega_i}{\lambda_{i,\max} f^P_{\max}} + \sum_{e_{j,i}\in\mathcal{E}} 2 \times (N_M - 1) \times \frac{s_{j,i}}{b_\omega f^L_{\max}} \right]$		
$D_{\min,i} = \frac{\omega_i}{\lambda_{i,\max} f^P_{\max}} + \sum_{e_{j,i}\in\mathcal{E}} \frac{s_{j,i}}{b_\omega f^L_{\max}}$		
Adjustable Parameters		
	N_{limit}	δ	μ_1, μ_2, μ_3	λ_1/λ_2
Min/Max/Step	1/4/1	0/1.2/0.1	0/1/0.1	0.2/1/0.2

VI. SIMULATION RESULTS AND ANALYSIS

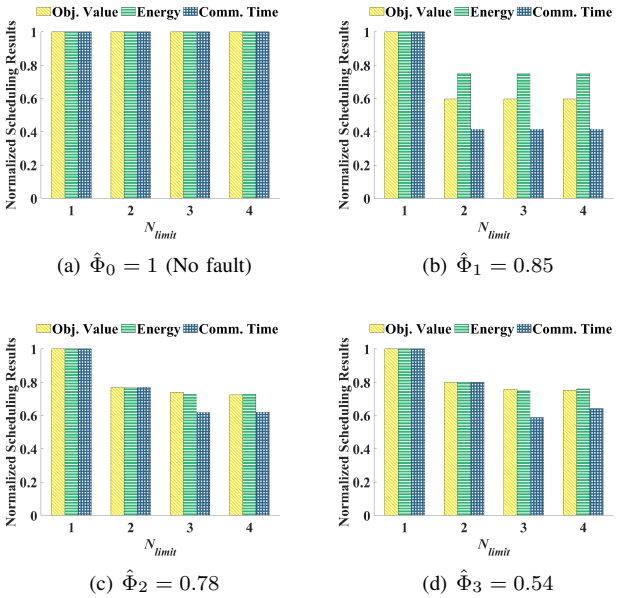
1) Simulation Setup Overview: We consider a heterogeneous Adapt NoC as the target platform, where two types of heterogeneous cores, Type 1 (Transmeta® Crusoe™) and Type 2 (Intel® PXA250™), are deployed, and the core parameters are adopted from [42]. The platform scale is set to 3×3 [14], [21], as it is sufficient to capture the essential performance and reliability trends, and keeps the target problem computationally tractable. In addition, the cores and links can support the DVFS

technique: Type 1 and Type 2 cores support 5 and 4 discrete V/F levels, respectively, while links support 5 discrete V/F levels. For the sake of problem formulation, we extend the number of V/F levels for Type 2 cores to 5, i.e., $f^P_{k,5} = f^P_{k,4}$ and $P^{comp}_{k,5} = P^{comp}_{k,4}$. Therefore, the number of V/F levels for cores and links can be set to $N_P = 5$ and $N_L = 5$, respectively. Due to heterogeneity across cores, the efficiency of different cores executing the same task may differ. Therefore, we introduce the heterogeneity factor $\lambda_{i,k}$ [35], where the value of $\lambda_{i,k}$ is within the range $[0.2, 1]$. To facilitate the comparison of the heterogeneity among the cores in Type 1 and Type 2, let λ_1 and λ_2 denote the heterogeneity factors of the cores in Type 1 and Type 2, respectively, and we will evaluate the influence of heterogeneity factor ratio λ_1/λ_2 on the proposed FTTGTM method. Besides, let N_{limit} denote the upper bound of the core number connected to a router, and we set $N_{limit} \in [1, 4]$, depending on the scale of the target platform.

We evaluate the fault-tolerant performance and scheduling overhead of the proposed FTTGTM method against existing fault-tolerant approaches: Energy-Aware and Reliability-Aware Mapping (EARAM) [25], Router-Level Redundancy Fault-Tolerant (RLRFT) [21], Permanent Fault-Tolerant Mapping (PFTM) [24], Deterministic-Path Routing Algorithm (DPRA) [18], and fault-tolerant method with Corvus Seek Algorithm (CoSA) [28]. Moreover, we adopt realistic application DAGs, including Laplace Equation (LE), LU Decomposition (LU), Gaussian Elimination (GE), and Fast Fourier Transform (FFT) [34], [43], and further randomly generate DAGs with TGFF [44], where the task number N ranges from 10 to 50.

To evaluate the effectiveness of the FTTGTM method, we consider permanent faults that occur randomly across components, including cores, routers, and links. Note that we assume that each permanent fault is independent of the others. Then, we introduce the network efficiency Φ to measure the communication efficiency of different NoC topologies, which is defined as the average value of the communication efficiency between all node pairs in NoC, i.e., $\Phi = \frac{\sum_{s \neq d \in G} \phi_{s,d}}{M \times (M-1)}$, where $M \times (M-1)$ is the number of communication node pairs in NoC, and $\phi_{s,d} = 1/h_{s,d}$ is the communication efficiency between nodes s and d, which is the inverse of the shortest communication path $h_{s,d}$ between the nodes. Since permanent faults can degrade the communication efficiency of NoCs and reduce the global efficiency Φ, we evaluate fault-tolerant performance for different methods in terms of Φ. Table III summarizes the main parameters involved in the simulations. Note that we can apply the proposed method to fault-tolerant task mapping problems with different parameters, since changes in the parameters do not affect the structure of our problem.

2) Results and Discussions: First, we evaluate the impact of the number of cores connected to routers N_{limit} on the scheduling results, as shown in Fig. 3. We consider the task set of LE and set $\delta = 0.5$, $\mu_1 = 0.3$, $\mu_2 = 0.4$, and $\mu_3 = 0.3$. Note that we consider the normalized performance metrics, using the maximum scheduling overhead as a benchmark, to avoid the influence of different units and magnitudes. Fig. 3(a) considers the no-fault situation, and we set the global efficiency to Φ_0. In this case, the Adapt NoC topology is a mesh, where each core is connected to a single router; thus, the normalized



(a) $\hat{\Phi}_0 = 1$ (No fault) (b) $\hat{\Phi}_1 = 0.85$

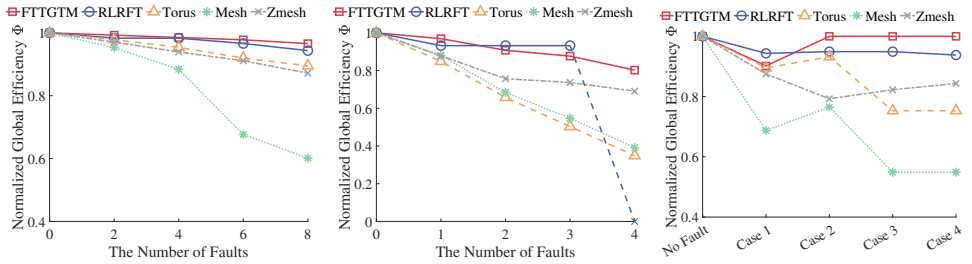
(c) $\hat{\Phi}_2 = 0.78$ (d) $\hat{\Phi}_3 = 0.54$

Fig. 3. Comparison of objective function, energy consumption, and communication overhead under different N_{limit} in no-fault and three-fault situations with LE application.

scheduling overheads for different N_{limit} values are identical. Fig. 3(b) considers the permanent faults that occur in the links $\{AL_{2,8}, AL_{8,2}, AL_{5,8}, AL_{8,5}, AL_{2,5}, AL_{5,2}\}$, and the normalized global efficiency is $\hat{\Phi}_1 = \Phi_1/\Phi_0 = 0.85$. The results show that the FTTGTM method can generate a fault-tolerant topology when $N_{limit} = 2$. Therefore, increasing N_{limit} will not change the generated topology, and the scheduling overhead remains unchanged. When the number of permanent faults increases, Adapt NoC needs to connect more cores, i.e., N_{limit} needs to be increased so that the FTTGTM method can generate the fault-tolerant topology. As shown in Fig. 3(c) and Fig. 3(d), the normalized global efficiencies are $\hat{\Phi}_2 = \Phi_2/\Phi_0 = 0.78$ for the faulty links $\{AL_{9,6}, AL_{6,9}, AL_{9,8}, AL_{8,9}, AL_{3,9}, AL_{9,3}, AL_{7,9}, AL_{9,7}\}$ and $\hat{\Phi}_3 = \Phi_3/\Phi_0 = 0.54$ for the faulty router AR_6, and the overhead for $N_{limit} = 3$ is less than $N_{limit} = 2$. Therefore, for different failure scenarios, the FTTGTM method can generate a fault-tolerant topology with the minimum scheduling overhead by adjusting N_{limit}.

Second, we compare the fault-tolerant performance of the FTTGTM method with the RLRFT and topology-fixed methods, which adopt common structures, including Torus, Mesh, and Zmesh. We consider the task set of LE and set $\lambda_1/\lambda_2 = 0.6$, $\delta = 0.5$, $N_{limit} = 2$, $\mu_1 = \mu_3 = 0.3$, and $\mu_2 = 0.4$. Considering permanent faults with different quantities that occur in different components, we take the global efficiency in the no-fault situation as a benchmark and evaluate the normalized global efficiency $\hat{\Phi}$ and scheduling overheads, including normalized energy consumption (En.) and communication overhead (Com.), see Fig. 4 and Table IV. In Table IV, the symbol $\times$ indicates that the scheduling method fails to cope with the permanent faults, and the energy consumption and communication overhead are normalized to the fault-free case for the sake of comparison. Moreover, Cases 1-4 correspond to four scenarios involving random combinations of core, link, and router faults.

Fig. 4 shows that the global efficiency of the FTTGTM method is higher than that of the other methods. Since the



(a) Link faults (b) Router faults (c) Core, link, and router faults

Fig. 4. Comparison of the normalized global efficiency $\hat{\Phi}$ under different fault situations with LE application.

TABLE IV
COMPARISON OF THE SCHEDULING OVERHEADS UNDER DIFFERENT FAULT SITUATIONS WITH LE APPLICATION.

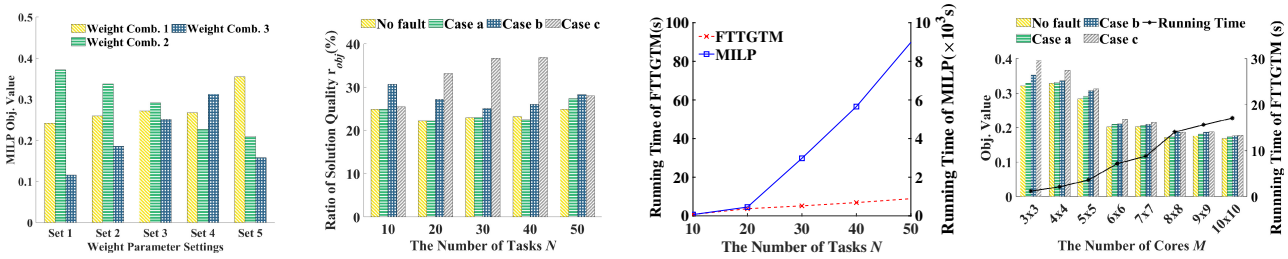
Fault		FTTGTM★		RLRFT		Torus		Mesh		Zmesh	
Position	Number	En.	Com.	En.	Com.	En.	Com.	En.	Com.	En.	Com.
Link	2	**1.00**	**1.00**	1.92	1.51	1.10	1.56	1.11	1.56	1.10	1.56
	4	**1.00**	**1.01**	2.14	1.60	1.11	1.61	1.15	1.74	1.12	1.67
	6	**1.01**	**1.04**	2.14	1.60	1.19	2.03	×	×	1.25	2.29
	8	**1.01**	**1.04**	2.14	1.60	1.19	2.03	×	×	1.35	2.82
Router	1	1.12	1.66	2.14	**1.60**	**1.01**	3.18	×	×	×	×
	2	**1.21**	2.11	2.14	**1.60**	1.23	3.78	×	×	×	×
	3	**1.25**	2.33	2.14	**1.60**	1.42	4.10	×	×	×	×
	4	**1.34**	**2.81**	×	×	×	×	×	×	×	×
Core	1	**1.04**	**1.04**	1.50	1.42	1.33	2.39	1.34	2.39	1.34	2.39
	2	**1.06**	**1.06**	1.85	1.92	1.73	2.85	1.74	2.85	1.74	2.85
	3	**1.11**	**1.11**	2.17	1.97	2.45	3.92	2.45	3.92	2.45	3.92
	4	**1.23**	**1.23**	×	×	×	×	×	×	×	×
Case 1		**1.02**	**1.13**	2.16	1.81	1.40	3.73	×	×	1.37	4.01
Case 2		**1.00**	**1.00**	2.10	1.25	1.38	3.62	×	×	×	×
Case 3		**1.00**	**1.00**	2.13	1.57	×	×	×	×	1.36	3.84
Case 4		**1.00**	**1.00**	2.32	1.75	×	×	×	×	1.27	3.18

topology-fixed methods can only use the normal components of the original topology for computation and communication, the global efficiency $\hat{\Phi}$ is lower than that of the FTTGTM and RLRFT methods. Furthermore, the RLRFT method cannot handle multiple router faults in each column; for example, it fails to schedule when the number of router faults is 4. However, the FTTGTM method can achieve finer-grained topology generation via structures such as adaptable links and routers and can handle multiple faults across components.

On the other hand, Table IV illustrates that the FTTGTM method has a lower energy consumption and communication overhead than other methods and can achieve feasible scheduling results even when the number of permanent faults is large. This is because the FTTGTM method utilizes adaptable links and routers to generate a fault-tolerant topology, thereby minimizing energy consumption and communication overhead. In contrast, the RLRFT method uses redundant routers to achieve fault-tolerant topology reconstruction, but when there are few or no permanent faults, redundant routers are not fully utilized, resulting in higher energy consumption and communication overhead than other methods. Furthermore, the topology-fixed methods cannot provide diverse topology connection options and fail the task scheduling when the number of faults is large.

Third, Table V compares En. and Com. across FTTGTM and baseline methods under three scenarios: **Case a**: Faults in $\{ML_{2,5}, ML_{4,5}\}$; **Case b**: Fault in router R_1; **Case c**: Faults in core θ_2, router R_4, and $\{ML_{2,5}, ML_{5,2}\}$. The normalized global efficiencies of Cases a, b, and c are $\hat{\Phi}_a = 0.99$, $\hat{\Phi}_b = 0.85$, and $\hat{\Phi}_c = 0.78$, respectively. In all faulty scenarios, FTTGTM achieves minimal performance degradation, reducing energy by 50.6% and 84.8% compared with EARAM and PFTM, respectively. Even under severe fault conditions (e.g., Case c), it incurs only moderate increases in energy (up to 17%) and communication cost (up to 86%), compared to the no-fault case. At the same time, other methods degrade significantly or even fail to complete scheduling (e.g., RLRFT, DPRA, and CoSA). Unlike EARAM's graph-based and PFTM's replication-based task reallocation, which both incur high communication overhead under multiple core faults, FTTGTM efficiently reconfigures the topology and remaps tasks with minimal performance loss. RLRFT utilizes static spare-router redundancy, which increases energy consumption while tolerating only one fault per mesh column, thereby failing under multiple faults. Moreover, DPRA relies on the precomputed paths, and CoSA lacks flexible task mapping, leading to unbalanced load and higher communication latency. Note that the above methods do not explicitly optimize the energy issues. Although FTTGTM considers energy efficiency as an objective, its energy reduction also benefits from topology reconfiguration, task remapping, and DVFS adjustment. We consider energy consumption and communication overhead as the costs introduced by fault tolerance. FTTGTM mainly focuses on offline optimization and static task DAGs, which may limit its applicability in dynamic scenarios.

Fig. 5(a) compares the optimal objective function values of LE application under no-fault situations with five groups of weight settings, where the settings cover both balanced configurations (e.g., $\{0.3, 0.4, 0.3\}$) and asymmetric configurations (e.g., $\{0.1, 0.6, 0.3\}$). The results show that the weight



(a) Objective function value (b) Solution quality (c) Running time (d) FTTGTM performance

Fig. 5. Weight parameter evaluation with LE application and FTTGTM on solution quality, runtime efficiency, and scalability with synthetic DAGs.

TABLE V
COMPARISON OF THE SCHEDULING OVERHEADS OF FOUR REAL TASK SETS UNDER DIFFERENT FAULT SITUATIONS AND APPLICATIONS.

Realistic Applications	Fault	FTTGTM★		EARAM		PFTM		RLRFT		DPRA		CoSA	
		En.	Com.	En.	Com.	En.	Com.	En.	Com.	En.	Com.	En.	Com.
LE	Case a	**1.00**	1.00	1.75	1.37	2.27	1.25	1.65	**0.93**	1.82	1.19	1.80	1.09
	Case b	**1.02**	**1.13**	1.48	1.25	1.74	2.29	1.83	1.25	1.82	1.19	1.81	1.14
	Case c	**1.12**	1.66	1.79	**1.25**	×	×	×	×	×	×	×	×
LU	Case a	**1.00**	**1.00**	1.83	1.56	2.50	1.68	1.75	1.20	1.98	1.26	1.98	1.26
	Case b	**1.05**	**1.22**	1.84	1.50	2.30	2.49	1.89	1.31	1.98	1.26	1.98	1.26
	Case c	**1.17**	1.86	1.39	**1.50**	3.87	4.00	×	×	×	×	×	×
GE	Case a	**1.00**	**1.00**	1.01	1.40	2.18	1.47	1.55	1.10	2.04	1.34	1.99	1.10
	Case b	**1.02**	**1.09**	1.76	1.48	2.14	2.56	1.61	1.21	2.04	1.34	2.01	1.22
	Case c	**1.17**	1.77	1.71	**1.40**	×	×	×	×	×	×	×	×
FFT	Case a	**1.00**	1.00	1.27	0.90	1.94	1.09	1.68	0.92	1.78	0.95	1.76	**0.88**
	Case b	**1.05**	1.20	1.79	**0.90**	1.51	1.35	1.68	0.92	1.78	0.95	1.83	1.07
	Case c	**1.11**	1.54	1.47	**0.90**	3.90	4.29	×	×	×	×	×	×

combinations $\{0.3, 0.3, 0.4\}$, $\{0.3, 0.4, 0.3\}$, and $\{0.4, 0.3, 0.3\}$ (i.e., Weight Comb. 1, 2, and 3) in Weight Set 3, produce closer objective function values, indicating a stable balance among the optimization objectives. We set $\mu_1 = 0.3$, $\mu_2 = 0.4$, and $\mu_3 = 0.3$, with a higher priority on energy efficiency due to its importance in the MPSoC designs.

Finally, Fig. 5(b)–5(d) compare the solution quality and the running time of the FTTGTM method and the MILP-based optimal method. We consider synthetic task sets with $N = \{10, 20, 30, 40, 50\}$, and set $\delta = 0.5$, $N_{limit} = 2$, and $\lambda_1/\lambda_2 = 0.6$. We take the optimal value of the objective function as a benchmark and consider the reduction ratio of solution quality $r_{obj} = \frac{obj_{FTTGTM} - obj_{OPT}}{obj_{OPT}} \times 100\%$ as a performance metric, where obj_{FTTGTM} and obj_{OPT} are the objective function values of FTTGTM method and the MILP-based optimal method, respectively. As shown in Fig. 5(b), under different task numbers N and fault situations, the maximum quality reduction of FTTGTM is around 36.9% (26.8% on average), compared with the optimal method. Fig. 5(c) compares the average running time of the FTTGTM method and the MILP-based optimal method for different numbers of tasks. Although the MILP-based optimal method can obtain the optimal solution, its running time increases exponentially with the number of tasks. Therefore, the FTTGTM method can achieve a balance between solution quality and computation time, yielding a high-quality solution in linear time. Fig. 5(d) evaluates the FTTGTM performances under different NoC scales ($3 \times 3 \rightarrow 10 \times 10$). As the feasible region of problem (18) increases with core number M, and it is a minimizing problem, the objective function value decreases with M. In addition, more faults will activate different components of the Adapt NoC to provide fault tolerance, thereby increasing energy consumption and communication overhead. The computation time of FTTGTM increases linearly with M (no faults); for the given M parameters, the computation times across different cases are similar since they have the same problem size.

VII. CONCLUSION

This paper presented a comprehensive fault-tolerant task mapping approach for real-time systems with permanent faults, using the Adapt NoC as the target platform. We formulated the permanent fault-tolerant task mapping problem as a MINLP problem that jointly optimizes energy consumption, communication delay, and load balancing under multiple fault conditions. To address the inherent complexity of the MINLP formulation, we proposed the FTTGTM method, which integrates a reconfigurable topology generation strategy, dynamic frequency allocation, and real-time task scheduling. The method effectively accommodates multiple permanent faults in diverse components: cores, routers, and links, through topology reconfiguration and task mapping refinement. Extensive simulations demonstrated the superiority of the proposed method over state-of-the-art approaches. FTTGTM achieves an average 67.7% reduction in energy consumption compared to existing fault-tolerant schemes, while maintaining low communication overhead and fault tolerance through integrated topology, frequency, and scheduling optimization. In future work, we aim to extend our model to address hybrid fault scenarios, incorporating both transient and permanent faults.

APPENDIX A
EXPLANATION OF TOPOLOGY GENERATION CONSTRAINT

Constraint (5) defines the routing path structure and can be interpreted through several cases depending on the roles of

the routers involved. For the edge $e_{i,j}$, assume that tasks τ_i and τ_j are assigned to cores θ_{p^*} and θ_{q^*}, i.e., $x_{i,p^*} = 1$ and $x_{j,q^*} = 1$. Further, suppose these cores are connected to routers AR_{s^*} and AR_{d^*} via $y_{p^*,s^*} = 1$ and $y_{q^*,d^*} = 1$. We have the following three cases: 1) If $n = s^*$ (i.e., the source router), then $\sum_{p\in\mathcal{M}} x_{i,p}y_{p,s^*} = 1$ and $\sum_{q\in\mathcal{M}} x_{j,q}y_{q,s^*} = 0$, and (5) simplifies to $\sum_{m:L_{n,m}\in\mathcal{L}} \eta_{i,j,n,m} - \sum_{m:L_{m,n}\in\mathcal{L}} \eta_{i,j,m,n} = 1$, which corresponds to the routing constraint at the start node (i.e., out-degree minus in-degree equals 1); 2) If $n = d^*$ (i.e., the destination router), then $\sum_p x_{i,p}y_{p,d^*} = 0$ and $\sum_q x_{j,q}y_{q,d^*} = 1$, and the equation becomes $\sum_{m:L_{n,m}\in\mathcal{L}} \eta_{i,j,n,m} - \sum_{m:L_{m,n}\in\mathcal{L}} \eta_{i,j,m,n} = -1$, enforcing the end-node constraint; 3) If $n \neq s^*$ and $n \neq d^*$ (i.e., an intermediate router), then both summations are 0, and the expression reduces to $\sum_{m:L_{n,m}\in\mathcal{L}} \eta_{i,j,n,m} - \sum_{m:L_{m,n}\in\mathcal{L}} \eta_{i,j,m,n} = 0$, ensuring flow conservation at intermediate nodes.

Appendix B
Explanation of Task Non-overlapping Constraint

Constraint (11) enforces the sequential execution of tasks τ_i and τ_j on the same core using the ordering variable $p_{i,j}$. For the independent tasks τ_i and τ_j with $q_{i,j} = 0$, if both tasks are mapped to the same core, e.g., θ_k, as $x_{i,k} = x_{j,k} = 1$, (11) is relaxed to $te_i \leq ts_j + (1 - p_{i,j})H$ and $te_j \leq ts_i + p_{i,j}H$. On the one hand, if τ_i is scheduled before τ_j (i.e., $p_{i,j} = 1$), we have $te_i \leq ts_j$ and $te_j \leq ts_i + H$, enforcing a sequential execution order of $\tau_i \rightarrow \tau_j$ on core θ_k. On the other hand, if $p_{i,j} = 0$ (i.e., τ_j precedes τ_i), then we get $te_j \leq ts_i$ and $te_i \leq ts_j + H$. However, if τ_i and τ_j are assigned to different cores, as $x_{i,k}+x_{j,k} \leq 1$, (11) is changed to $te_i \leq ts_j+\alpha H$ and $te_j \leq ts_i + \beta H$, where α and β are constants $(1 \leq \alpha, \beta \leq 3)$, and (11) is always satisfied.

Appendix C
Linearization of the proposed task mapping problem

The fault-tolerant task mapping problem in (18) is a MINLP, and we apply the linearization method in [37] to handle the nonlinear items caused by the products of binary-binary variables. First, we introduce auxiliary variables $z_{i,k,m}$, $\alpha_{i,k,l}$, $\beta_{m,n,g}$, and $\gamma_{i,j,m,n}$ to replace $x_{i,k}y_{k,m}$, $x_{i,k}c^P_{k,l}$, $v_{m,n}c^L_g$, and $v_{m,n}\eta_{i,j,m,n}$, respectively, and then add the following constraints into (18): 1) $z_{i,k,m} \leq x_{i,k}$, $z_{i,k,m} \leq y_{k,m}$, and $z_{i,k,m} \geq x_{i,k} + y_{k,m} - 1$; 2) $\alpha_{i,k,l} \leq x_{i,k}$, $\alpha_{i,k,l} \leq c^P_{k,l}$, and $\alpha_{i,k,l} \geq x_{i,k} + c^P_{k,l} - 1$; 3) $\beta_{m,n,g} \leq c^L_g$, $\beta_{m,n,g} \leq v_{m,n}$, and $\beta_{m,n,g} \geq c^L_g + v_{m,n} - 1$; 4) $\gamma_{i,j,m,n} \leq \eta_{i,j,m,n}$, $\gamma_{i,j,m,n} \leq v_{m,n}$, and $\gamma_{i,j,m,n} \geq \eta_{i,j,m,n} + v_{m,n} - 1$. Similarly, we set $\delta_{i,j,g} = c^L_g t^{comm}_{i,j,g}$ and introduce constraints $0 \leq \delta_{i,j,g} \leq c^L_g t^{\max}_{i,j}$, $\delta_{i,j,g} \leq t^{comm}_{i,j,g}$, and $\delta_{i,j,g} - c^L_g t^{\max}_{i,j} - t^{comm}_{i,j,g} + t^{\max}_{i,j} \geq 0$ to deal with the nonlinear items caused by the products of binary-continuous variables, where $t^{\max}_{i,j}$ is the upper bound of communication time for the edge $e_{i,j}$. Through the above transformations, (18) is reformulated as an equivalent MILP problem.

Acknowledgment

The authors confirm that all scientific content in this article, including ideas, methodology, data analysis, and results, is their original work and was developed without the use of generative AI tools. Grammarly was used solely for grammar checking and improving readability.

References

[1] W. Wolf, A. A. Jerraya, and G. Martin, "Multiprocessor system-on-chip (MPSoC) technology," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 27, no. 10, pp. 1701–1713, 2008.

[2] H. Ali, U. U. Tariq, J. Hardy *et al.*, "A survey on system level energy optimisation for MPSoCs in IoT and consumer electronics," *Computer Science Review*, vol. 41, p. 100416, 2021.

[3] M. S. Gaur, V. Laxmi, M. Zwolinski *et al.*, "Network-on-chip: Current issues and challenges," in *International Symposium on VLSI Design and Test*, 2015, pp. 1–3.

[4] A. V. Bhaskar and T. Venkatesh, "Performance analysis of network-on-chip in many-core processors," *Journal of Parallel and Distributed Computing*, vol. 147, pp. 196–208, 2021.

[5] P. V. Bhanu and P. V. Kulkarni, "Fault-tolerant network-on-chip design with flexible spare core placement," *ACM Journal on Emerging Technologies in Computing Systems*, vol. 15, no. 1, pp. 1–23, 2019.

[6] E. Taheri, M. Isakov, A. Patooghy, and M. A. Kinsy, "Addressing a new class of reliability threats in 3-D network-on-chips," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 39, no. 7, pp. 1358–1371, 2019.

[7] S. Werner, J. Navaridas, and M. Luján, "A survey on design approaches to circumvent permanent faults in networks-on-chip," *ACM Computing Surveys*, vol. 48, no. 4, pp. 1–36, 2016.

[8] K. Khalil, A. Kumar, and M. Bayoumi, "Dynamic fault tolerance approach for network-on-chip architecture," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 14, no. 3, pp. 384–394, 2024.

[9] R. Fernandes, C. Marcon, R. Cataldo, and J. Sepúlveda, "Using smart routing for secure and dependable NoC-based MPSoCs," *IEEE/ACM Trans. Netw.*, vol. 28, no. 3, pp. 1158–1171, 2020.

[10] F. Fu, B. Lou, Y. Chen, and J. Wang, "Improving reliability in NoCs by reconstructing location distribution of management cores," *Microelectronics journal*, vol. 90, pp. 133–140, 2019.

[11] A. Naghavi, S. Safari, and S. Hessabi, "Tolerating permanent faults with low-energy overhead in multicore mixed-criticality systems," *IEEE Trans. Emerg. Top. Comput.*, vol. 10, no. 2, pp. 985–996, 2021.

[12] B. N. K. Reddy, A. James, and A. S. Kumar, "Fault-tolerant core mapping for NoC based architectures with improved performance and energy efficiency," in *IEEE International Conference on Electronics, Circuits and Systems*, 2022, pp. 1–4.

[13] B. N. K. Reddy, M. Zia Ur Rahman, and A. Lay-Ekuakille, "Enhancing reliability and energy efficiency in many-core processors through fault-tolerant network-on-chip," *IEEE Trans. Netw. Serv. Manag.*, vol. 21, no. 5, pp. 5049–5062, 2024.

[14] N. Kadri, A. Chenine, Z. Laib, and M. Koudil, "Reliability-aware intelligent mapping based on reinforcement learning for networks-on-chips," *The Journal of Supercomputing*, vol. 78, no. 16, pp. 18 153–18 188, 2022.

[15] J. Wu, Y. Wu, G. Jiang, and S. K. Lam, "Algorithms for reconfiguring NoC-based fault-tolerant multiprocessor arrays," *Journal of Circuits, Systems and Computers*, vol. 28, no. 7, p. 1950111, 2019.

[16] P. V. Bhanu and J. Soumya, "Fault-tolerant application mapping on mesh-of-tree based network-on-chip," *Journal of Systems Architecture*, vol. 116, p. 102026, 2021.

[17] A. Charif, A. Coelho, N.-E. Zergainoh, and M. Nicolaidis, "A dynamic sufficient condition of deadlock-freedom for high-performance fault-tolerant routing in networks-on-chips," *IEEE Trans. Emerg. Top. Comput.*, vol. 8, no. 3, pp. 642–654, 2017.

[18] Y. Zhang, X. Hong, Z. Chen *et al.*, "A deterministic-path routing algorithm for tolerating many faults on very-large-scale network-on-chip," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 26, no. 1, pp. 1–26, 2020.

[19] A. Romanov, N. Myachin, and A. Sukhov, "Fault-tolerant routing in networks-on-chip using self-organizing routing algorithms," in *Annual Conference of the IEEE Industrial Electronics Society*, 2021, pp. 1–6.

[20] Y. Liu, R. Guo, C. Xu *et al.*, "A Q-learning-based fault-tolerant and congestion-aware adaptive routing algorithm for networks-on-chip," *IEEE Embedded Systems Letters*, vol. 14, no. 4, pp. 203–206, 2022.

[21] Y.-C. Chang, C.-S. A. Gong, and C.-T. Chiu, "Fault-tolerant mesh-based NoC with router-level redundancy," *Journal of Signal Processing Systems*, vol. 92, pp. 345–355, 2020.

[22] Q. Xu, H. Geng, T. Ni *et al.*, "Fortune: A new fault-tolerance TSV configuration in router-based redundancy structure," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 41, no. 10, pp. 3182–3187, 2021.

[23] H. Zheng, K. Wang, and A. Louri, "Adapt-NoC: A flexible network-on-chip design for heterogeneous manycore architectures," in *IEEE International Symposium on High-Performance Computer Architecture*, 2021, pp. 723–735.

[24] D. Cheng, W. Hu, J. Liu *et al.*, "Permanent fault-tolerant scheduling in heterogeneous multi-core real-time systems," in *IEEE International Conference on Systems, Man, and Cybernetics*, 2021, pp. 673–678.

[25] A. S. Kumar and B. Naresh Kumar Reddy, "An efficient real-time embedded application mapping for NoC based multiprocessor system on chip," *Wireless Personal Communications*, vol. 128, no. 4, pp. 2937–2952, 2023.

[26] P. V. Bhanu, R. Govindan, R. Kumar *et al.*, "Fault-tolerant application-specific topology-based NoC and its prototype on an FPGA," *IEEE Access*, vol. 9, pp. 76 759–76 779, 2021.

[27] D. Xu, Y. Ouyang, W. Zhou *et al.*, "RMC_NoC: A reliable on-chip network architecture with reconfigurable multifunctional channel," *IEEE Trans. Very Large Scale Integr. Syst.*, vol. 31, no. 12, pp. 2061–2074, 2023.

[28] P. Narayanasamy and S. Gopalakrishnan, "Novel fault tolerance topology using Corvus seek algorithm for application specific NoC," *Integration*, vol. 89, pp. 146–154, 2023.

[29] Y. Ouyang, T. Zhang, J. Li, and H. Liang, "Fault-tolerant routing for reliable packet transmission in on-chip networks," *Microelectronics Journal*, vol. 153, p. 106425, 2024.

[30] W. Zhou, D. Xu, S. Xu *et al.*, "Reconfigurable fault-tolerant link with bandwidth expansion for 2.5-D chiplet-based systems," *IEEE Trans. Very Large Scale Integr. Syst.*, 2025.

[31] U. U. Tariq, H. Wu, and S. Abd Ishak, "Energy-aware scheduling of conditional task graphs on NoC-based MPSoCs," in *International Conference on System Sciences*, 2018, p. 5707–5716.

[32] K. Kiran and J. Jacob, "Energy-aware application mapping onto 3D mesh-based network-on-chip using heuristic mapping algorithms," *Journal of Universal Computer Science*, vol. 31, no. 2, p. 136, 2025.

[33] J. He, Y. Xiao, C. Bogdan *et al.*, "A design methodology for energy-aware processing in unmanned aerial vehicles," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 27, no. 1, pp. 1–20, 2021.

[34] H. Topcuoglu, S. Hariri, and M. Wu, "Performance-effective and low-complexity task scheduling for heterogeneous computing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 13, no. 3, pp. 260–274, 2002.

[35] D. Li and J. Wu, "Minimizing energy consumption for frame-based tasks on heterogeneous multiprocessor platforms," *IEEE Trans. Parallel Distrib. Syst.*, vol. 26, no. 3, pp. 810–823, 2015.

[36] S. K. Mandal, A. Krishnakumar, and U. Y. Ogras, "Energy-efficient networks-on-chip architectures: Design and run-time optimization," *Network-on-Chip Security and Privacy*, pp. 55–75, 2021.

[37] X. Li, L. Mo, A. Kritikakou, and O. Sentieys, "Approximation-aware task deployment on heterogeneous multicore platforms with DVFS," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 42, no. 7, pp. 2108–2121, 2022.

[38] B. Acun, K. Chandrasekar, and L. V. Kale, "Fine-grained energy efficiency using per-core DVFS with an adaptive runtime system," in *International Green and Sustainable Computing Conference*, 2019, pp. 1–8.

[39] A. Gupta, S. Ahmed, A. K. Jain *et al.*, "Run-time reconfiguration of NoC in Xilinx ACAP architecture," in *International Workshop on Network on Chip Architectures*, 2020, pp. 1–6.

[40] M. Fischetti and D. Salvagnin, "Feasibility pump 2.0," *Mathematical Programming Computation*, vol. 1, no. 2, pp. 201–222, 2009.

[41] L. Mo, X. Li, A. Kritikakou, and X. Zhai, "Contention and reliability-aware energy efficiency task mapping on NoC-based MPSoCs," *IEEE Trans. Reliab.*, vol. 74, no. 1, pp. 2010–2026, 2025.

[42] H. Ali, U. U. Tariq, Y. Zheng *et al.*, "Contention and energy-aware real-time task mapping on NoC based heterogeneous MPSoCs," *IEEE Access*, vol. 6, pp. 75 110–75 123, 2018.

[43] H. Arabnejad and J. G. Barbosa, "List scheduling algorithm for heterogeneous systems by an optimistic cost table," *IEEE Trans. Parallel Distrib. Syst.*, vol. 25, no. 3, pp. 682–694, 2013.

[44] A. Esmaili, M. Nazemi, and M. Pedram, "Energy-aware scheduling of task graphs with imprecise computations and end-to-end deadlines," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 25, no. 1, pp. 1–21, 2019.

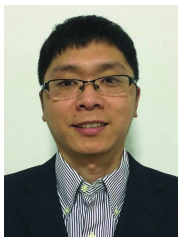
Xinmei Li received the B.S. degree in automation engineering from Sichuan University, Chengdu, China, in 2021, and the M.S. degree in control science and engineering from the School of Automation, Southeast University, Nanjing, China, in 2024. She is currently pursuing the Ph.D. degree with the School of Cyber Science and Engineering, Southeast University, Nanjing, China. Her current research interests include system reliability, network security, and networked control systems.

Lei Mo is an Associate Professor at the School of Automation, Southeast University, Nanjing, China. He received a B.S. degree from the College of Telecom Engineering and Information Engineering, Lanzhou University of Technology, Lanzhou, China, in 2007, and a Ph.D. degree from the College of Automation Science and Engineering, South China University of Technology, Guangzhou, China, in 2013. From 2013 to 2015, he was a research fellow with the Department of Control Science and Engineering at Zhejiang University, China. From 2015 to 2019, he was a research fellow with INRIA Nancy–Grand Est, France, and INRIA Rennes–Bretagne Atlantique, France. His research interests include estimation and control in networked networks, task mapping, and resource allocation in embedded systems.

Tamim M. Al-Hasan received a B.S. in Electrical Engineering from Qatar University, Doha, Qatar, in 2022. He is currently pursuing a Ph.D. degree in Computer Science and Electronic Engineering at the University of Essex, Colchester, United Kingdom. His research interests include embedded systems design, machine learning, computer security, and cryptography.

Angeliki Kritikakou is an Associate Professor at the University of Rennes and IRISA - INRIA Rennes research center. She received a Ph.D. degree in 2013 from the Department of Electrical and Computer Engineering at the University of Patras, Greece, in collaboration with IMEC Research Center, Belgium. She worked for one year as a Postdoctoral Research Fellow at the Department of Modelling and Information Processing (DTIM) at ONERA in collaboration with the Laboratory of Analysis and Architecture of Systems (LAAS) and the University of Toulouse, France. Her research interests include embedded systems, real-time systems, mixed-critical systems, hardware/software co-design, mapping methodologies, design space exploration methodologies, memory management methodologies, low-power design, and fault tolerance.

Xiaojun Zhai is currently a Reader in the Embedded Intelligent Systems Laboratory at the University of Essex. His research interests include designing and implementing digital image and signal processing algorithms, custom computing using FPGAs, embedded systems, and hardware/software co-design.



Shibo He received a Ph.D. degree in control science and engineering from Zhejiang University, Hangzhou, China, in 2012. From Nov. 2010 to Nov. 2011, he was a visiting scholar with the University of Waterloo, Waterloo, ON, Canada. He was an Associate Research Scientist from Mar. 2014 to May. 2014, and a postdoctoral scholar from May. 2012 to Feb. 2014, with Arizona State University, Tempe, AZ, USA. He is currently a Professor at Zhejiang University. His research interests include wireless sensor networks, crowd sensing, and big data analysis.

Olivier Sentieys is a Professor at the University of Rennes holding an INRIA Research Chair on Energy-Efficient Computing Systems. He is leading the Cairn team common to INRIA and IRISA Laboratory. He is also the head of the Computer Architecture department of IRISA. From 2012 to 2017, he was on secondment at INRIA as a Senior Research Director. His research interests are in the area of computer architectures, embedded systems, and signal processing, with a focus on system-level design, energy efficiency, reconfigurable systems, hardware acceleration, approximate computing, and power management of energy harvesting sensor networks.