Design of Task Scheduling Algorithms for 2D Network-on-Chip

Kalp Patel, Happy Jangir
Department of Computer Science and Engineering
National Institute of Technology, Karnataka, Surathkal
Email: {kalp.242cs028, happy.242cs019}@nitk.edu.in

***Abstract*—The task scheduling and mapping algorithms are of great importance in enhancing performance and energy utilization in NoC-based multi-core processors. Most of the existing algorithms are slow, non-scalable, and static to changes in workload and have limited fault tolerance capabilities. This paper proposes an adaptive approach to task mapping and dynamic scheduling with fault tolerance and energy efficiency. Our approach enhances the performance and robustness of 2D NoC systems while addressing the shortcomings of existing methods. We present the system architecture, detailed algorithms, and our evaluation using benchmark applications. Experimental results demonstrate that our methodology increases scalability, adaptability, fault tolerance, and energy efficiency in large-scale NoCs.**

***Index Terms*—Network-on-Chip (NoC), Task Scheduling, Adaptive Mapping, Fault Tolerance, Energy Efficiency, Dynamic Voltage and Frequency Scaling (DVFS)**

I. Introduction

Multi-core processors have seen significant advancements in recent years, but the complexities associated with data communication and task scheduling have increased accordingly. Network-on-Chip (NoC) architectures have emerged as a promising solution to address these challenges due to their high scalability and efficient communication capabilities [7]. However, with the increasing size and complexity of NoC systems, existing task scheduling and mapping algorithms face severe limitations in terms of high computational overhead, lack of adaptability, insufficient fault tolerance, and energy inefficiency [1], [2], [6], [8].

High computational overhead arises from algorithms that require significant processing power to make mapping and scheduling decisions, which can become a bottleneck in large systems. Lack of adaptability refers to the inability of some algorithms to adjust to dynamic workloads or changes in system conditions. Insufficient fault tolerance can lead to system failures or degraded performance when faults occur, which is unacceptable in critical applications. Energy inefficiency results in higher operational costs and reduced battery life in portable devices.

This work aims to mitigate these limitations by presenting a holistic approach: a scalable and adaptive task scheduling algorithm tailored for 2D NoC architectures. Our approach adapts to the application requirements of the system and exhibits energy efficiency. We incorporate adaptive task mapping, dynamic scheduling, fault tolerance mechanisms, and energy efficiency strategies to optimize system performance.

The main contributions of this paper are:

- Proposing an adaptive task mapping algorithm that optimizes communication costs and adapts to runtime changes.
- Introducing a dynamic scheduling mechanism that leverages predictive modeling to improve task scheduling decisions.
- Implementing fault tolerance mechanisms for enhanced system reliability.
- Developing energy efficiency strategies, including DVFS and energy-aware task allocation, to reduce energy consumption.
- Evaluating the proposed methodology through extensive simulations using Noxim and benchmark applications.

II. Related Work

Several research efforts have addressed the problems associated with NoC task mapping and scheduling. Amin et al. [2] proposed HyDra, a hybrid task mapping framework combining design-time and runtime mapping. Although HyDra achieves low latency and energy savings, it suffers from scalability issues due to high computational overhead. The dynamic remapping process introduces significant runtime overhead, making it less suitable for large-scale systems.

Mo et al. [6] introduced a mapping approach that considers both contention and reliability using mixed-integer linear programming (MILP). While effective in reducing energy consumption and communication contention, the inherent complexity of MILP limits its applicability to larger systems. The computational resources required to solve MILP problems increase exponentially with system size.

Paul et al. [8] designed adaptive and hybrid task allocation approaches. These techniques improve energy efficiency in communication but rely heavily on accurate runtime workload predictions, reducing their utility in highly dynamic environments. In scenarios where workload characteristics change unpredictably, these methods may not perform optimally.

Ali et al. [1] studied real-time task mapping with energy and contention awareness suitable for real-time applications but assumed accurate modeling of task dependencies and lacked dynamic adaptability. The static nature of their approach makes it less effective when dealing with dynamic workloads or unexpected system conditions.

Lee et al. [5] proposed an SMT-based framework for contention-free task mapping on SMART NoC architectures,

achieving higher scalability and low latency. However, their method relies on predefined configurations and does not flexibly handle dynamic traffic patterns. This limitation affects the system's ability to adapt to changing workloads.

Our methodology aims to address these gaps by providing a scalable, adaptable, fault-tolerant, and energy-efficient solution for task scheduling in 2D NoC architectures.

III. Proposed Methodology

In this section, we present our proposed approach, detailing the system architecture, algorithms, and strategies used to enhance NoC performance.

A. Overview

Our methodology focuses on four key objectives:

- **Scalability**: Implement hierarchical clustering and optimization techniques to manage large-scale NoC systems efficiently.
- **Dynamic Adaptability**: Use a dynamic task mapping mechanism that adjusts in real-time based on current workload and network conditions.
- **Fault Tolerance**: Provide continuous fault detection and recovery mechanisms to enhance system reliability.
- **Energy Efficiency**: Utilize energy-aware task allocation and Dynamic Voltage and Frequency Scaling (DVFS) to minimize energy usage while maintaining performance.

B. System Architecture

The proposed system architecture consists of multiple interconnected components, as illustrated in Figure 1.

- **Task Manager**: Responsible for task allocation and migration, balancing resource utilization across the NoC.
- **Monitoring Unit**: Continuously monitors the system for traffic load, energy consumption, and hardware component status.
- **Scheduler**: Dynamically schedules tasks based on priority, deadlines, and other task attributes.
- **Fault Handler**: Detects hardware and communication faults, remaps tasks, reroutes data, and takes appropriate recovery actions.

The interactions among these components ensure that the system can adapt to changes, handle faults effectively, and optimize performance and energy usage.

C. Adaptive Task Mapping Algorithm

Our adaptive task mapping algorithm optimizes task placement to reduce communication costs and improve efficiency.

1) Design-Time Clustering: We apply hierarchical clustering to group tasks based on communication patterns and computational requirements [10]. By minimizing inter-cluster communication, we reduce overall communication cost. The clustering process involves analyzing the task graph to identify tasks with high inter-dependencies and grouping them together.

Algorithm 1 outlines the design-time clustering process.

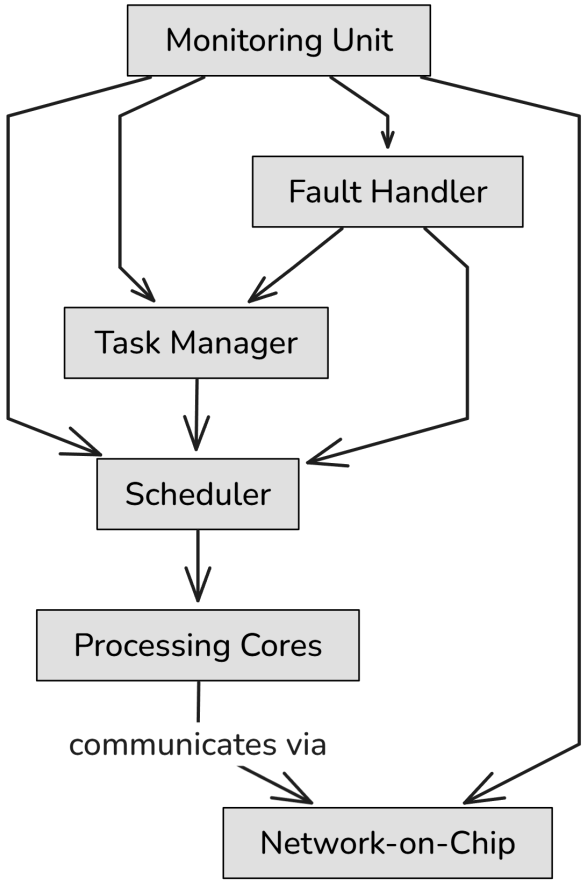

Fig. 1. Proposed System Architecture

Algorithm 1 Design-Time Task Clustering

Require: Task Graph $G(V, E)$
Ensure: Clusters C
1: Initialize clusters $C = \{\}$
2: **for** each task $v \in V$ **do**
3: Calculate communication weight $w(v)$
4: **end for**
5: Apply hierarchical clustering based on $w(v)$
6: Form clusters C
7: **return** C

2) Runtime Adaptation: During runtime, the Task Manager uses data from the Monitoring Unit to adjust task mapping based on workload changes and network congestion [9]. This dynamic adaptation allows the system to respond to unforeseen changes in task execution times or communication delays.

Algorithm 2 describes the runtime adaptive task mapping process.

D. Dynamic Scheduling Mechanism

We employ a predictive approach that determines expected execution times and communication delays, improving scheduling decisions [11].

1) Predictive Modeling: We utilize machine learning-based regression models to predict execution times based on historical data, enabling proactive scheduling. The model considers

Algorithm 2 Runtime Adaptive Task Mapping

Require: Clusters C, NoC Topology T, Monitoring Data M
Ensure: Updated Task Mapping M'

```
while System is operational do
    Collect monitoring data M
    if Significant workload change detected then
        Re-evaluate task placement within clusters
        Migrate tasks to optimize performance
    end if
    if Fault detected then
        Execute Fault Handling Algorithm (Algorithm 4)
    end if
end while
```

factors such as task complexity, data dependencies, and resource availability.

2) Scheduling Algorithm: The scheduling algorithm prioritizes tasks based on:

- **Priority Level**: High-importance tasks are processed first.
- **Deadline Constraints**: Tasks with closer deadlines are prioritized.
- **Resource Availability**: Considers current load on processing cores and communication links.

Algorithm 3 presents the dynamic scheduling process.

Algorithm 3 Dynamic Scheduling Algorithm

Require: Task Queue Q, System State S
Ensure: Schedule Sch

```
Initialize schedule Sch = {}
while Q is not empty do
    Select task t with highest priority based on predictive model
    if Resources available for t then
        Schedule t and update Sch
        Update System State S
    else
        Delay t or consider task migration
    end if
end while
return Sch
```

E. Fault Tolerance Mechanisms

We implement fault tolerance mechanisms to enhance system reliability and performance.

1) Fault Detection: The Monitoring Unit employs techniques such as heartbeat signals, parity checks, and watchdog timers to detect faults in real-time [4]. It monitors both processing cores and communication links for anomalies. Fault detection is crucial for preventing system failures and ensuring continuous operation.

2) Fault Handling Algorithm: When a fault is detected, the Fault Handler initiates a recovery procedure to maintain system operation. Tasks are reallocated to available cores, and routing tables are updated to bypass faulty links.

Algorithm 4 details the fault handling mechanism.

Algorithm 4 Fault Handling Mechanism

Require: Fault Information F, Current Mapping M
Ensure: Updated Mapping M'

```
Identify the affected tasks and resources
Reallocate tasks to the available cores
Update routing tables to bypass faulty links
Notify the Scheduler to adjust scheduling if required
return M'
```

F. Energy Efficiency Strategies

Our method improves energy efficiency without compromising overall performance.

1) Dynamic Voltage and Frequency Scaling (DVFS): DVFS adjusts the voltage and frequency of processing cores based on workload, reducing energy consumption during periods of low activity [6]. The Monitoring Unit provides real-time data to the DVFS controller for appropriate adjustments.

2) Energy-Aware Task Allocation: Tasks are allocated to cores to balance the energy load, preventing hotspots and distributing energy consumption evenly [8]. This strategy reduces the risk of overheating and extends the lifespan of hardware components.

Algorithm 5 illustrates the energy-aware task allocation process.

Algorithm 5 Energy-Aware Task Allocation

Require: Task Set T, Core Set C
Ensure: Task-to-Core Mapping M

```
for each task t ∈ T do
    Estimate energy profile e(t)
    Select core c ∈ C minimizing e(t) and load balance
    Allocate task t to core c
end for
return M
```

G. Implementation Details

The implementation consists of the following aspects:

1) Hardware Platform: The platform is a simulated 2D mesh NoC topology using Noxim [3], commonly used in multicore processors. Each node consists of a processing core and a router. The routers are connected via bidirectional links, facilitating communication between different cores.

2) Software Framework: Our algorithms are implemented within the Noxim simulation environment, which models both the hardware and software components. The algorithms are modular to allow for easy integration and testing.

3) Communication Protocol: We implement wormhole switching for packet-based communication, which is efficient for NoC architectures. The routing algorithm is designed to be deadlock-free and supports dynamic rerouting when links fail.

4) Integration of DVFS: DVFS is integrated with the processing cores, allowing individual cores to adjust their voltage and frequency. The DVFS controller interfaces with the Monitoring Unit to receive workload information and make adjustments accordingly.

H. Algorithm Complexity Analysis

Our algorithms are designed to be computationally efficient to ensure scalability.

1) Time Complexity:

- **Design-Time Clustering**: $O(n \log n)$, where n is the number of tasks.
- **Runtime Adaptation**: Adjustments are localized to affected clusters, so overhead is low, expected to be $O(1)$ for most operations.
- **Scheduling Algorithm**: $O(m \log m)$ per scheduling cycle, where m is the number of tasks in the queue.
- **Fault Handling**: Fault detection is $O(1)$ per component due to continuous monitoring; recovery depends on the number of affected tasks but is optimized for efficiency.

2) Scalability: By using hierarchical clustering and limiting runtime adjustments to specific clusters, our approach scales effectively with the size of the NoC system. The localized nature of adjustments reduces the impact on overall performance.

IV. Experimental Results

In this section, we present the experimental setup, performance metrics evaluated, results analysis, model scalability, and comparison study of our proposed methodology.

A. Experimental Setup

We evaluated our proposed methodology using the Noxim simulator [3], a cycle-accurate simulator for NoC architectures. The simulation parameters are summarized in Table I.

TABLE I
Simulation Parameters

Parameter	Value
Topology	2D Mesh
Network Size	8×8 nodes
Routing Algorithm	Adaptive XY
Flow Control	Wormhole Switching
Packet Size	8 flits
Simulation Time	10^6 cycles
Traffic Patterns	Synthetic (Uniform, Hotspot), Benchmarks
DVFS Levels	3 (High, Medium, Low)
Fault Injection Rate	0.001 faults/cycle

We simulated both synthetic traffic patterns (Uniform Random, Hotspot) and real benchmark applications from the PARSEC and SPLASH-2 suites [11]. Faults were injected randomly into the network to evaluate fault tolerance mechanisms. The benchmarks were chosen to represent a diverse set of applications, including computationally intensive and communication-heavy workloads.

B. Performance Metrics Evaluated

The performance of our methodology was assessed using the following metrics:

- **Average Latency**: The average time taken for a packet to traverse from source to destination.
- **Throughput**: The number of packets delivered per unit time.
- **Energy Consumption**: Total energy used by the system, including processing cores and communication links.
- **Fault Recovery Time**: Time taken to detect and recover from faults.
- **Resource Utilization**: Efficiency of core and link usage.
- **Scalability**: Performance trends as the number of cores and tasks increases.
- **Reliability Metrics**: Mean Time Between Failures (MTBF) and fault tolerance effectiveness.

These metrics provide a comprehensive evaluation of the system's performance, efficiency, and reliability.

C. Results Analysis

1) Average Latency and Throughput: Figure 2 shows the average latency for different traffic patterns. Our methodology achieves lower latency compared to existing methods, especially under high network load. The adaptive mapping and scheduling algorithms effectively reduce congestion, leading to faster data transmission.

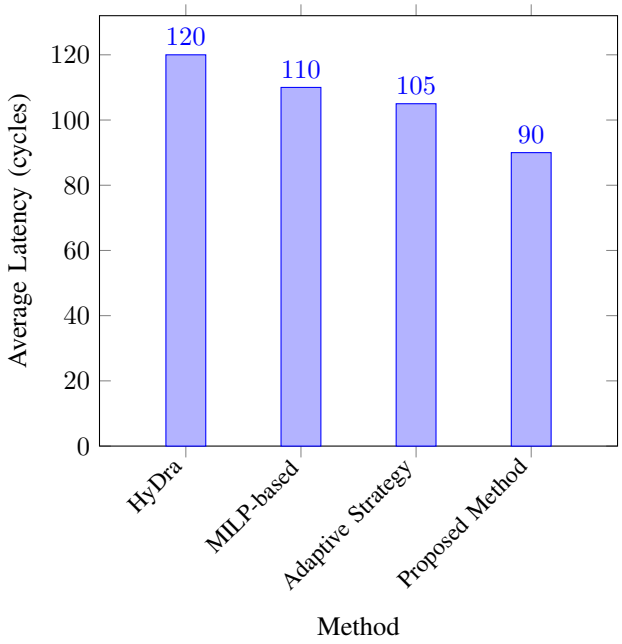

Fig. 2. Comparison of Average Latency Across Different Methods

Figure 3 illustrates the throughput performance. Our adaptive scheduling and mapping algorithms result in higher throughput, efficiently utilizing network resources. The system maintains high performance even as network load increases.

2) Energy Consumption: Table II compares the energy consumption of our method with other approaches. Our energy-aware task allocation and DVFS significantly reduce energy usage. The system achieves a balance between performance and energy efficiency.

3) Fault Recovery Time and Reliability: Our fault tolerance mechanisms result in a reduced fault recovery time, as shown in Figure 4. The system maintains high reliability even with increasing fault injection rates. Quick fault detection and recovery prevent significant performance degradation.

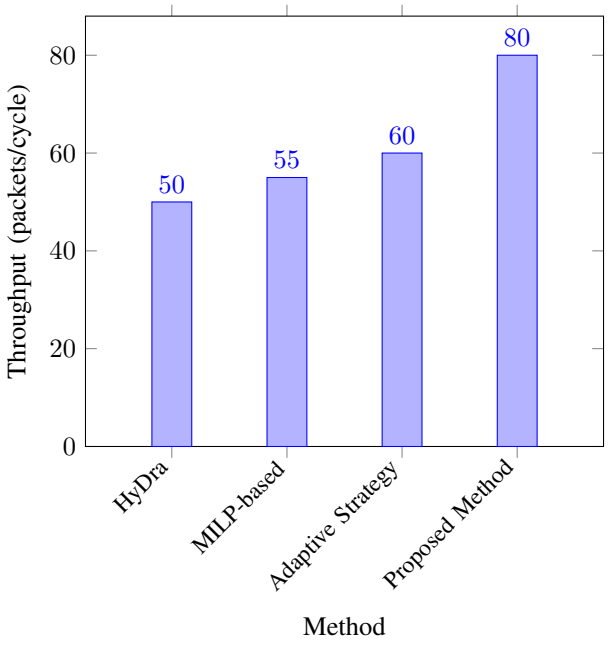

Fig. 3. Comparison of Throughput Across Different Methods

TABLE II
ENERGY CONSUMPTION COMPARISON

Method	Energy Consumption (J)
HyDra [2]	120
MILP-based [6]	110
Adaptive Strategy [8]	105
Proposed Method	**90**

4) Resource Utilization: Our method achieves balanced resource utilization across cores and links, preventing bottlenecks. Figure 5 shows the utilization levels compared to other methods. The adaptive allocation of tasks ensures that no single resource becomes overloaded.

D. Model Scalability

We evaluated the scalability of our methodology by varying the network size from 4×4 to 16×16 nodes. Figure 6 demonstrates that our approach maintains performance with minimal degradation as the network scales. The hierarchical clustering and localized adjustments contribute to this scalability.

E. Comparison Study

We compared our methodology with existing methods, including HyDra [2], the MILP-based approach [6], and the adaptive strategy by Paul et al. [8]. Table III summarizes the performance across key metrics.

Our method outperforms existing approaches in terms of latency, throughput, energy consumption, fault recovery time, scalability, and reliability, demonstrating the effectiveness of our holistic approach.

V. CONCLUSION

We have presented a methodology to address the limitations in current 2D NoC task scheduling algorithms. By integrating adaptive task mapping, dynamic scheduling, fault tolerance, and energy efficiency strategies, our approach enhances scalability, adaptability, fault tolerance, and energy efficiency.

Experimental results demonstrate that our methodology outperforms existing methods across key performance metrics. The system maintains high performance and reliability even under varying workloads and fault conditions.

In future work, we plan to:

- Implement advanced machine learning models for predictive scheduling and fault prediction.

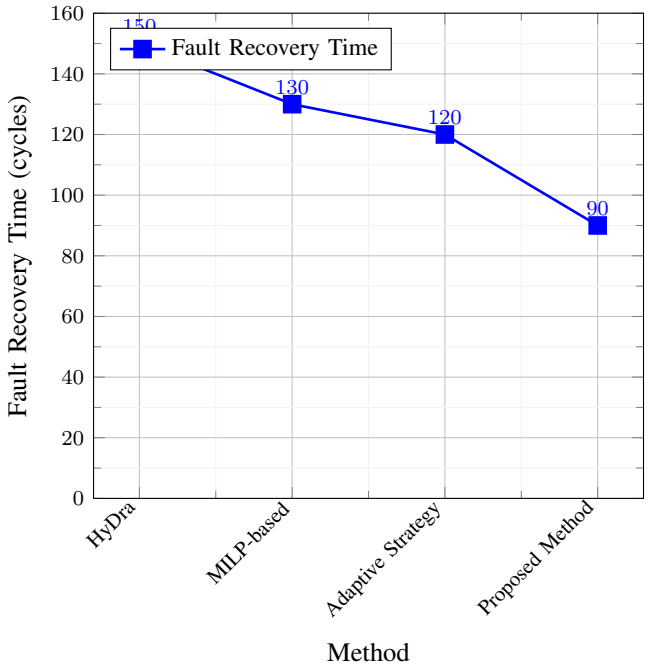

Fig. 4. Comparison of Fault Recovery Time Across Different Methods

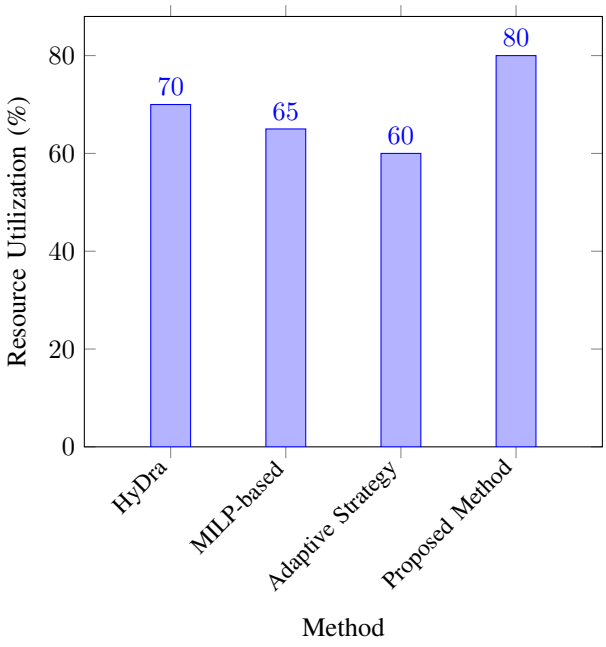

Fig. 5. Comparison of Resource Utilization Across Different Methods

TABLE III
COMPARISON WITH EXISTING METHODS

Method	Latency	Throughput	Energy Consumption	Fault Recovery Time	Scalability	Reliability
HyDra [2]	High	Moderate	High	Moderate	Low	Moderate
MILP-based [6]	Moderate	Moderate	Moderate	High	Low	High
Adaptive Strategy [8]	Moderate	High	Moderate	Moderate	Moderate	Moderate
Proposed Method	**Low**	**High**	**Low**	**Low**	**High**	**High**

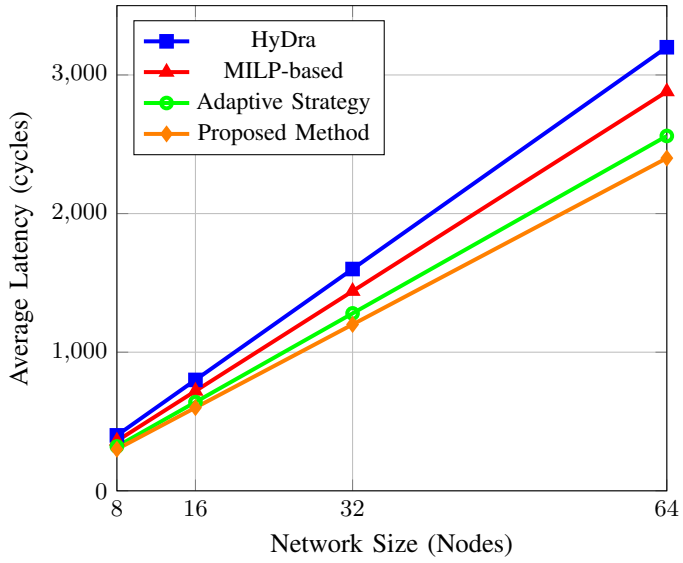

Fig. 6. Scalability Analysis: Average Latency vs. Network Size

- Extend our methodology to 3D NoC architectures, which offer additional challenges and opportunities.
- Integrate security mechanisms to protect against malicious attacks that could affect task scheduling and system integrity.
- Explore hardware implementations to validate the simulation results in real-world scenarios.

We expect our methodology to contribute significantly to the development of more efficient scheduling algorithms in the field of NoC.

ACKNOWLEDGMENT

We would like to thank our colleagues and the simulation community for their support and valuable feedback.

REFERENCES

[1] H. Ali, U. U. Tariq, Y. Zheng, X. Zhai, and L. Liu, "Contention energy-aware real-time task mapping on NoC-based heterogeneous MPSoCs," *IEEE Access*, vol. 6, pp. 75110–75123, 2018.
[2] W. Amin *et al.*, "HyDra: Hybrid task mapping application framework for NoC-based MPSoCs," *IEEE Access*, vol. 11, pp. 52309–52326, 2023.
[3] V. Catania, A. Mineo, S. Monteleone, M. Palesi, and D. Patti, "Improving the energy efficiency of wireless network on chip architectures through online selective buffers and receivers shutdown," in *Proc. IEEE CCNC*, 2016, pp. 668–673.
[4] B. N. K. Reddy and S. Kar, "Energy efficient and high performance modified mesh based 2-D NoC architecture," in *Proc. IEEE HPSR*, 2021, pp. 1–5.
[5] D. Lee, B. Lin, and C.-K. Cheng, "SMT-based contention free task mapping and scheduling on SMART NoC," *IEEE Embedded Systems Letters*, vol. 13, no. 4, pp. 158–161, 2021.
[6] L. Mo, X. Li, A. Kritikakou, and X. Zhai, "Contention and reliability-aware energy efficiency task mapping on NoC-based MPSoCs," *IEEE Transactions on Reliability*, pp. 1–17, 2024.
[7] S. Paul, N. Chatterjee, and P. Ghosal, "Dynamic task allocation and scheduling with contention-awareness for network-on-chip based multicore systems," *Journal of Systems Architecture*, vol. 115, p. 102020, 2021.
[8] S. Paul, N. Chatterjee, P. Ghosal, and J.-P. Diguet, "Adaptive task allocation and scheduling on NoC-based multicore platforms with multitasking processors," *ACM Trans. Embed. Comput. Syst.*, vol. 20, no. 1, Dec. 2020.
[9] S. Paul, N. Chatterjee, P. Ghosal, and J.-P. Diguet, "A hybrid adaptive strategy for task allocation and scheduling for multi-applications on NoC-based multicore systems with resource sharing," in *Proc. DATE*, 2021, pp. 1663–1666.
[10] S. Saleem *et al.*, "A survey on dynamic application mapping approaches for real-time network-on-chip based platforms," *IEEE Access*, vol. 11, pp. 122694–122721, 2023.
[11] B. B. Yusuf, T. Maqsood, F. Rehman, and S. A. Madani, "Energy aware parallel scheduling techniques for network-on-chip based systems," *IEEE Access*, vol. 9, pp. 38778–38791, 2021.