Binary Multi-Objective Salp Swarm Optimization for Task Mapping in NoC-based Heterogeneous MPSoCs

Farid Boumaza[1,2,*], Atmane Hadji[3], Abdelkader Aroui[4]
[1]Computer Science Department, University of Mohamed El Bachir El Ibrahimi, Bordj Bou Arreridj 34030, Algeria
[2]Laboratory of Parallel, Embedded architectures and Intensive Computing (LAPECI), University of Oran1, Oran 31000, Algeria
[3]LISI Laboratory, Computer Science Department, University Center A. Boussouf Mila, 43000 Mila, Algeria
[4]Center for Space Techniques, Palestine Avenue, 31200 Arzew, Oran, Algeria
E-mail: f.boumaaza@univ-bba.dz, a.hadji@centre-univ-mila.dz, abdelkader.aroui@gmail.com

Keywords: Energy consumption, time minimization, multi-processor systems-on-chip, task mapping, salp swarm algorithm

Received: December 24, 2024

This work presents an innovative approach for optimizing static task mapping of computation-intensive applications onto Network-on-Chip (NoC)-based heterogeneous Multi-Processor Systems-on-Chip (MPSoCs). The objective is to minimize both makespan (time) and total energy consumption, critical metrics for embedded systems performance. Task mapping, an NP-hard problem, assigns application tasks to processing elements in a 2D mesh NoC, significantly influencing system efficiency. To address this, we propose a Binary Multi-Objective Salp Swarm Algorithm (BMSSA), leveraging Pareto dominance and an external archive with crowding distance to maintain diverse, non-dominated solutions. Evaluated on benchmark applications with 10 to 50 tasks mapped to 4 to 20 processors, BMSSA achieves a reduction of up to 50% in makespan and 9% in energy compared to state-of-the-art methods, including NSGA-II, MOPSO, and MOACO. Statistical analysis via Wilcoxon signed-rank tests confirms BMSSA's significant improvements, particularly over MOPSO. These results underscore BMSSA's scalability and effectiveness for energy-aware, performance-optimized task mapping in next-generation MPSoCs.

Povzetek: Članek predstavi BMSSA, binarni večciljni algoritem s Pareto dominanco in arhivom nedominiranih rešitev, ki za statično mapiranje nalog na heterogene NoC MPSoC hkrati optimira makespan in energijo.

1 Introduction

The demand for fast, energy-efficient embedded systems is pushing Multi-Processor Systems-on-Chip (MPSoCs) to the forefront of modern computing platforms [10]. These systems combine multiple, diverse processing elements (PEs) on a single chip and connect them using scalable Network-on-Chip (NoC) infrastructures. MPSoCs are widely used in areas like multimedia processing and real-time applications, where low energy use and minimal latency are critical [6]. However, assigning application tasks to these systems efficiently is still a significant challenge, as it is a known NP-hard problem [2, 3, 5].

Task mapping assigns computational tasks to processing elements (PEs) while managing communication over NoC links. It needs to balance several conflicting goals at once, such as reducing energy use, cutting communication delays, distributing workload evenly, and staying within memory and bandwidth limits [11]. Because the problem is combinatorial in nature [7, 8], heuristic and metaheuristic algorithms are crucial for finding strong solutions within a reasonable time.

We introduce a new approach called the Binary Multi-Objective Salp Swarm Algorithm (BMSSA) for task mapping in NoC-based heterogeneous MPSoCs. BMSSA builds on the continuous Salp Swarm Algorithm (SSA) [12], adapting it to work in discrete, multi-objective optimization settings. It uses binary encoding to represent task assignments and focuses on minimizing both energy use and execution time. The algorithm respects key system constraints, such as bandwidth, memory limits, and workload distribution, while effectively exploring the solution space and quickly converging on near-optimal mappings.

To test how well BMSSA works, we compared it with three leading multi-objective metaheuristic algorithms: Non-Dominated Sorting Genetic Algorithm II (NSGA-II) [13], Multi-Objective Particle Swarm Optimization (MOPSO) [14], and Multi-Objective Ant Colony Optimization (MOACO) [34]. The results show that BMSSA performs better in most scenarios in terms of energy efficiency and latency, making it a strong and reliable choice for mapping complex MPSoC tasks.

The rest of the paper is structured as follows. Section 2 covers related work. Section 3 explains the background

concepts behind this study. Section 4 describes the proposed BMSSA approach. Section 5 shares experimental results and performance evaluation. Finally, Section 6 wraps up the paper and suggests directions for future research.

2 Related work

Efficient task and communication mapping is still a significant challenge in the design of NoC-based Multiprocessor MPSoCs. Mapping strategies usually fall into two groups: static and dynamic [36]. Static mapping is done at design time, using full architectural knowledge to optimize computational and communication loads. Dynamic mapping, on the other hand, happens at run time and allows the system to adapt to changing workloads. Both approaches aim to save energy, cut communication delays, and reduce execution time. This work focuses on static, design-time mapping because it offers predictability and lower overhead during execution.

Several approaches have been proposed to address the NP-hard nature of the task mapping problem. To place our method in context, we summarize the main contributions from the literature in Table 1, comparing algorithms by their core techniques, target NoC topologies, optimization goals, and unique features.

As summarized in Table 1, prior work spans heuristic, metaheuristic, and hybrid methods, often targeting energy and latency optimization. However, key limitations remain: (i) most approaches assume homogeneous architectures, overlooking real-world heterogeneity; (ii) many rely on single-objective or weighted-sum formulations, which can bias results and reduce solution diversity; and (iii) several metaheuristics lack a proper balance between exploration and exploitation, especially in discrete, constrained spaces.

The proposed BMSSA is designed to address these shortcomings. Unlike traditional approaches, BMSSA explicitly targets heterogeneous MPSoC architectures. It adopts a Pareto-based multi-objective framework, enabling a more accurate representation of trade-offs between energy consumption and execution time, without resorting to scalarization. Furthermore, the binary formulation of the Salp Swarm Algorithm enhances its suitability for discrete task mapping problems. The leader–follower dynamic inherent in SSA supports a robust balance between global exploration and local exploitation, improving convergence behavior in complex search spaces. These advantages are empirically validated in Section 5 through extensive comparisons with state-of-the-art techniques.

3 Definition and mapping formulation

The communication between application tasks and components of the target architecture is commonly represented using two directed graphs [5].

Definition 1. The application is modeled as a directed graph $G(T, E)$, where each node $t_i \in T$ stands for a task or computational unit. A directed edge $(t_i, t_j) \in E$, written as e_{ij}, shows communication from task t_i to task t_j. Each edge e_{ij} has a weight Q_{ij}, which indicates how much data is transferred between the two tasks. (Fig. 1).

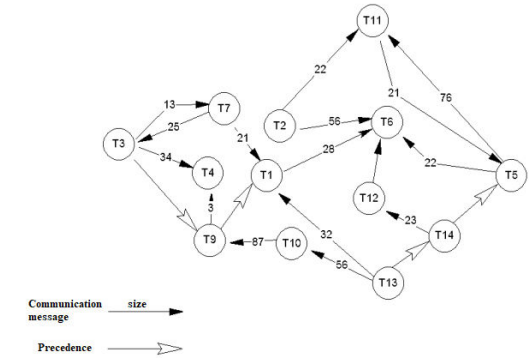


Figure 1: Application graph

Definition 2. The target MPSoC platform is modeled as a directed graph $AG = P(S, F)$, where each vertex $s_i \in S$ represents a processing or communication node within the hardware topology. A directed edge $(s_i, s_j) \in F$, denoted as f_{ij}, represents a physical communication link between nodes s_i and s_j. Each edge f_{ij} is assigned a weight bw_{ij}, which encapsulates key properties of the physical connection, such as bandwidth, latency, and energy consumption (Fig. 2).

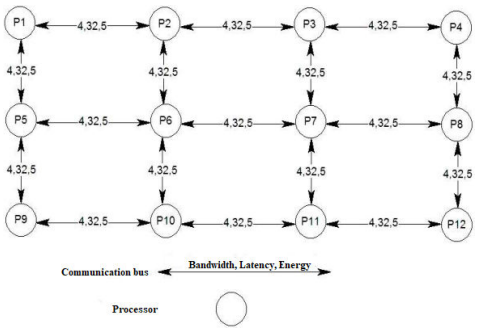


Figure 2: Architecture graph

Definition 3. The mapping of the application graph $G(T, E)$ onto the architecture graph $P(S, F)$ is defined by a function:

$$\text{map} : T \to S \quad \text{such that} \quad \text{map}(t_i) = s_j, \quad \forall\, t_i \in T\,, \exists\, s_j \in S. \tag{1}$$

The mapping is valid under the condition that the number of available processing elements satisfies $|S| \leq |T|$ [4] (Fig. 3).

Table 1: Summary of Related Work on NoC Task Mapping

Reference	Algorithm	Topology	Objectives	Key Feature
M. Darbari and all [15]	Chaotic GA	2D Mesh	Optimize latency and energy	Uses chaos theory for improved GA exploration
G. Fen and all [16]	GAMR (GA)	2D Mesh	Minimize latency	Simultaneous mapping and routing
S Tosun and all [17]	ILP	2D Mesh	Minimize Latency and Energy	Exact method for small problems
Y. Liu and all [18]	ACO	2D Mesh	NoC energy and hotspot optimization	Bandwidth-constrained
W. Jang and all [19]	A3MAP-SR + GA	2D Mesh	Minimize energy	Partition-based, two-stage
S. Tosun [20]	CastNet (Heuristic)	2D Mesh	Minimize energy	Low-complexity heuristic
P. Sahu [21]	DPSO	Mesh-of-Tree (MoT)	Computation time	Hierarchical NoC
M. Obaidullah and all [22]	Tabu Search + Deflection	2D NoC	Reduce communication volume	Hybrid, large space exploration
X. Wang and all[23]	WOA-GA Hybrid	2D NoC	Ninimze energy, Stability	Hybrid meta-heuristic
A. Alagarsamy and all [24]	SCSO + KNN	2D NoC	Minimize energy	Task clustering for mapping
A Alagarsamy and all [25]	Modified BA	2D NoC	Minimize energy	Cluster-based performance
X Wang and all [26]	PSO-SCA Hybrid	2D NoC	Computation time	Binary encoding
M. Mohiz and all [27]	Cuckoo Search	2D NoC	Performance, reduce energy	Levy Flight
M. Darbandi [28]	MOPSO	MPSoC	Reducing communication delays and costs	Crowding-distance diversity
S. Sikandar and all[29]	Sailfish Opt. (SFOA)	2D NoC	Minimize energy	Shared k-NN clustering
F. Mehmood [30]	Andean Condor (ACA)	2D NoC	Optimize both throughput and energy	For large-scale search
W. Amin [31]	iHPSA (PSO+SA)	2D NoC	Minimize energy	K-means clustering

The application is made up of tasks $T = \{t_1, t_2, \ldots, t_n\}$, while the architecture includes processing elements $P = \{p_1, p_2, \ldots, p_m\}$. Each processing element supports several operating modes m_1, m_2, m_3, which adds flexibility to task placement and helps improve optimization efficiency.

3.1 Execution time and communication duration

In a NoC-based MPSoC, the total time consumption of an application depends on both the task execution time ET and the inter-task communication time CT, as determined

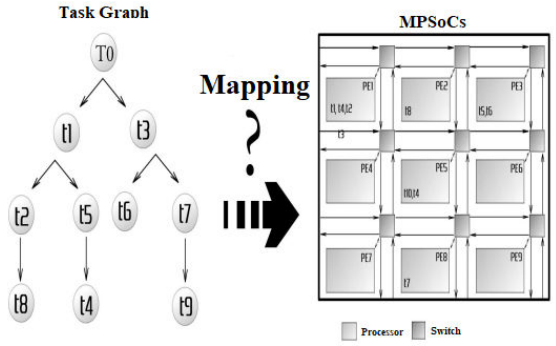


Figure 3: Mapping graph task on MPSoCs architecture

by the static mapping of tasks to processing elements.

The execution time The execution time $ET_{s_p}^{t_i}(m)$ of task t_i on processor s_p in mode m is:

$$ET_{s_p}^{t_i}(m) = \frac{\text{Cycles}_{ip}}{f_{mp}} \tag{2}$$

where Cycles_{ip} is the task's cycle count, and f_{mp} is the processor's frequency in mode m.

Communication time The communication time between tasks i and j, mapped to processors s_p and s_q, is: The communication time CT for the data transfer along edge $ij = (t_i, t_j)$ depends on the mapping of tasks t_i and t_j to processors $s_p = \text{map}(t_i)$ and $s_q = \text{map}(t_j)$.
If $s_p = s_q$ (tasks mapped to the same processing element), the CT is negligible: $CT_{ij} = 0$.

Otherwise, communication occurs over a path in the NoC architecture graph $G = P(T, E)$, let $\text{path}(s_p, s_q)$ denote the sequence of directed edges $\{f_{uv}^{(1)}, f_{uv}^{(2)}, \ldots, f_{uv}^{(h)}\}$ from s_p to s_q, where h is the number of hops.
The communication time is then:

$$CT(ij) = \sum_{f_{uv} \in \text{path}(s_p, s_q)} l_{uv} + \frac{Q_{ij}}{\min_{f_{uv} \in \text{path}(s_p, s_q)} bw_{uv}} \tag{3}$$

Where l_{uv} is the latency of link f_{uv} (derived from the characterization of bw_{uv}), bw_{uv} is the bandwidth of link f_{uv}, and Q_{ij} is the data volume transferred.
The makespan, or total application completion time, is the maximum finishing time across all tasks in the $G(T, E)$, considering precedence constraints and parallelism in the MPSoC.

The start time of task t_i mapped to $s_j = \text{map}(t_i)$ in mode m is

$$ST(t_i) = \begin{cases} 0 & \text{if } t_i \text{ has no predecessors,} \\ \max\limits_{t_p : pi \in E} \left(FT^{t_p} + CT(pi)\right) & \text{otherwise.} \end{cases} \tag{4}$$

And its finishing time is

$$FT^{t_i} = ST^{t_i} + ET_{s_j}^{t_i}(m). \tag{5}$$

The makespan is therefore

$$M = \max_{t_i \in T} FT(t_i). \tag{6}$$

3.2 Energy consumption

The energy consumption E in a multiprocessor system-on-chip (MPSoC) comes from two sources: execution energy from computation and communication energy from the network.

Computational energy The energy consumed during the execution of a task t_i on processor s_p operating in mode m is defined as:

$$E_{\text{exec}}^{i} = \text{Cycles}_{ip} \times e_{mp} \tag{7}$$

Where the execution energy of task i, denoted as E_{exec}^{i}(in Joules), is given by the product of the number of clock cycles needed to run it on processor p ($Cycles_{ip}$) and the energy consumed per cycle by processor p in mode m (e_{mp}).

Communication energy The communication energy between tasks t_i and t_j, mapped to processors s_p and s_q, is expressed as:

$$E_{\text{com}}^{ijpq} = \sum_{e_{ij} \in E} Q_{ij} \cdot \sum_{f_{uv} \in \text{path}(s_p, s_q)} e_{L,uv} \tag{8}$$

where $e_{ij} \in E$ represents communication from task t_i to t_j, with Q_{ij} as the transferred data volume. Tasks are mapped to processors $s_p = \text{map}(t_i)$ and $s_q = \text{map}(t_j)$, connected via the NoC path $\text{path}(s_p, s_q) = f_{uv}^{(1)}, f_{uv}^{(2)}, \ldots, f_{uv}^{(h)}$ of h hops. Each link f_{uv} consumes $e_{L,uv}$ energy per unit data. If $s_p = s_q$ (same processor), no NoC communication occurs, and $E_{\text{comm}}(e_{ij}) = 0$.

Total energy The total energy consumed by the MPSoC during the application's execution, combining the energy from computation and communication, can be expressed as:

$$E_{\text{total}} = \sum_{i=1}^{n} E_{\text{exec}}^{i} + \sum_{(i,j) \in E} E_{\text{com}}^{ijpq} \tag{9}$$

3.3 Salp swarm algorithm (SSA)

The Salp Swarm Algorithm (SSA) [12] is a nature-inspired metaheuristic used to solve complex optimization problems. It models the swarming behavior of salps, marine organisms that move in coordinated chains as they ride ocean currents. SSA balances exploration (global search) and exploitation (local refinement), which helps it navigate large and complex search spaces effectively.

3.3.1 Biological inspiration

Salps move in synchronized chains. The leading salp (*leader*) sets the direction, while the others (*followers*) adjust their positions based on the leader and nearby salps. This coordinated movement enables them to explore their environment efficiently and forms the biological basis of SSA, which utilizes this behavior to navigate complex solution spaces (Fig. 4).

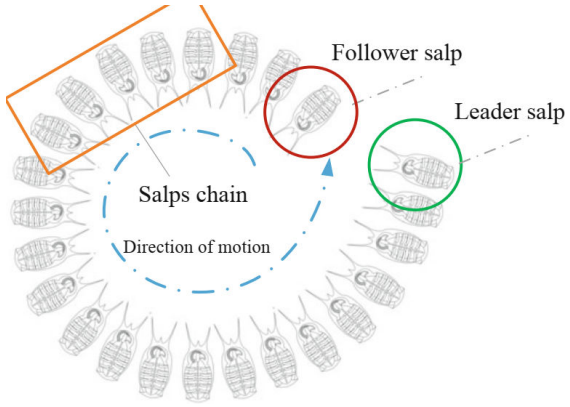


Figure 4: Illustration of salp behavior in SSA [12]

3.3.2 Mathematical model of SSA

In SSA, the salp population is divided into leaders and followers, with their movements defined by the following equations.

Leader's movement The leader updates its position based on the best-known solution, enabling broad exploration while directing the swarm toward promising regions of the search space, as defined by Eq.(3.3.2).

$$x_1^j = \begin{cases} F_j + c_1 \cdot \left(\frac{(ub-lb)}{2}\right), & \text{if } c_2 \geq 0.5, \\ F_j - c_1 \cdot \left(\frac{(ub-lb)}{2}\right), & \text{if } c_2 < 0.5, \end{cases}$$

where x_1^j denotes the leader's position in the j^{th} dimension, F_j is the best-known solution, ub and lb represent the upper and lower search-space bounds, c_1 is a random factor regulating exploration, and c_2 is a random number uniformly distributed in $[0, 1]$.

Followers' movement The followers update their positions relative to the salps ahead of them, as defined by Eq. (10).

$$x_i^j = \frac{x_{i-1}^j + x_i^j}{2}, \tag{10}$$

where x_i^j denotes the position of the i^{th} follower in the j^{th} dimension, and x_{i-1}^j represents the position of the preceding salp.

4 Binary multi-objective salp swarm algorithm for task mapping

The proposed method tackles the Assignment and Scheduling (AS) problem, which involves mapping application tasks to processing elements (PEs) and scheduling their communication over a NoC-based heterogeneous MPSoC.

We model this as a multi-objective optimization problem that aims to reduce total energy use and makepan at the same time. These goals often conflict; low-power modes save energy but increase latency, so the solution must show the trade-offs instead of forcing a single optimal point.

To search this trade-off space, we propose a Binary Multi-Objective Salp Swarm Algorithm (BMSSA). BMSSA adapts the continuous Salp Swarm Algorithm for discrete task mapping by using binary encoding, Pareto dominance, and an external archive with crowding distance to maintain diversity among non-dominated solutions. System-level limits, such as PE load balancing, NoC bandwidth, and local memory capacity, are handled through feasibility checks and simple repair steps.

4.1 Multi-objective salp swarm algorithm (MSSA)

The Multi-Objective Salp Swarm Algorithm (MSSA) [12] extends the original Salp Swarm Algorithm (SSA) to address problems involving multiple conflicting objectives. Unlike single-objective approaches that yield a single optimal solution, MSSA identifies a set of non-dominated solutions representing optimal trade-offs. This set enables decision-makers to select outcomes that align with specific application requirements.

4.1.1 Principles of MSSA

The MSSA retains the core mechanisms of SSA, such as the leader-follower dynamic and chain-based movement, but incorporates specific techniques for handling multi-objective optimization problems [9, 35]. Key principles include:

Pareto dominance A solution A is said to dominate another solution B if A is no worse than B in all objectives and strictly better in at least one objective. MSSA uses this concept to guide the salps toward optimal trade-offs between objectives.

Archive management Non-dominated solutions in the search space are stored in an external archive, which is updated iteratively. Diversity-preservation techniques, such as crowding distance, ensure an even distribution of solutions across the non-dominated solutions set.

Leader selection Rather than relying on a single leader, MSSA selects leaders probabilistically from among the optimal solutions, promoting exploration across the Pareto front and preserving solution diversity.

4.2 Binary Multi-objective salp swarm algorithm (BMSSA)

The Binary Multi-Objective Salp Swarm Algorithm (BMSSA) is a discrete extension of MSSA, tailored for binary optimization problems. While the original MSSA operates in continuous domains, BMSSA addresses solution spaces where candidates are encoded as binary vectors, such as in task mapping, assignment, and resource allocation. A position-to-binary transformation mechanism converts continuous salp positions into binary decisions, enabling compelling exploration of discrete spaces while retaining MSSA's leader–follower dynamics.

The following subsections present our key contributions in adapting MSSA to the multi-objective task mapping problem on NoC architectures.

Binary representation of solutions Each BMSSA solution is encoded as a binary matrix $X \in \{0,1\}^{n\times p}$, where n is the number of tasks and p the number of processing elements (PEs). A value $X_{ij} = 1$ indicates that task t_i is mapped to PE s_j. To ensure valid mappings, each task is assigned to exactly one PE, satisfying $\sum_{j=1}^{p} X_{ij} = 1$.

Initialization strategy The initialization strategy for BMSSA generates a diverse initial population of N solutions, each represented as a binary matrix $X^k \in \{0,1\}^{n\times p}$, where n is the number of tasks and p is the number of processing elements in the 2D mesh NoC. For each salp k, the matrix X^k is constructed by assigning each task t_i to exactly one processor s_j, ensuring the constraint $\sum_{j=1}^{p} X_{ij}^k = 1$. Specifically, for each task i, a processor j is selected uniformly at random from $\{1, 2, \ldots, p\}$, setting $X_{ij}^k = 1$ and $X_{il}^k = 0$ for all $l \neq j$. This random initialization promotes exploration across the vast search space of possible task mappings, facilitating the discovery of a diverse Pareto front for minimizing makespan and total energy consumption.

Transformation mechanism To operate in a binary search space, Binary MSSA applies a transformation function to convert continuous salp positions into binary values. Common techniques include:

– Sigmoid Function:

$$S(x) = \frac{1}{1 + e^{-x}} \tag{11}$$

Where x is the continuous position of the salp. If $S(x) \geq \text{rand}(0,1)$, the position is set to 1; otherwise, it is set to 0.

Leader and follower movement In BMSSA, the leader's movement follows the same principle as in standard MSSA but is adapted to operate within the binary search space. Followers update their positions based on the leader's state, allowing the swarm to balance exploration and exploitation across the discrete solution space.

Archive and diversity maintenance In each BMSSA iteration, the current salp population is combined with the existing Pareto archive to form a temporary repository of candidate solutions. This merged set includes both newly generated and previously stored non-dominated solutions, enhancing exploration and preserving elite solutions.

To ensure the archive remains within its predefined size limit, crowding distance [33] is applied to rank solutions based on their distribution in the objective space. If the temporary repository exceeds the allowed archive size, solutions with lower crowding distance, i.e., those in denser regions of the Pareto front, are discarded. This maintains a well-distributed and diverse set of non-dominated solutions (Fig. 2).

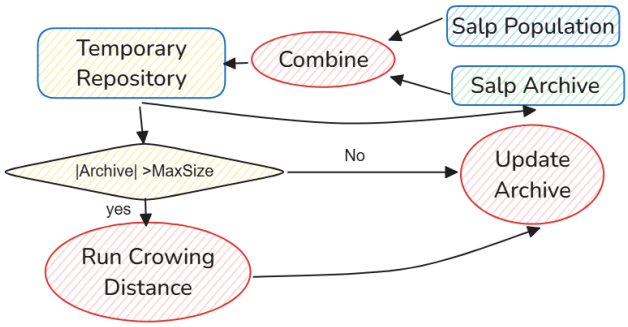


Figure 5: The management of the archive population

4.3 Objective function

The task mapping problem is formulated as a multi-objective optimization, aiming to minimize both the makespan and the total energy consumption. The objective is to determine a mapping function map : $T \to S$ that assigns each task $t_i \in T$ to a processing element $s_p \in S$, optimizing these objectives while satisfying architectural constraints. The two objective functions are defined as follows:

Objective function (F_T) The makespan represents the maximum completion time among all tasks, considering execution delays, task dependencies, and inter-task communication over the 2D mesh NoC. It is formally defined in Eq. (6).

Objective function (F_E) The total energy consumption, defined in Eq. (9), is the sum of the execution energy of all tasks and the communication energy of all inter-task data transfers.

The outcome of the proposed approach is an optimized mapping of application tasks and communication flows onto the physical resources of the architecture, along with a scheduling strategy that satisfies both performance and energy constraints. Figure 6 illustrates the overall process, and Algorithm **??** summarizes the key steps of the BMSSA-based multi-objective task mapping on the NoC platform.

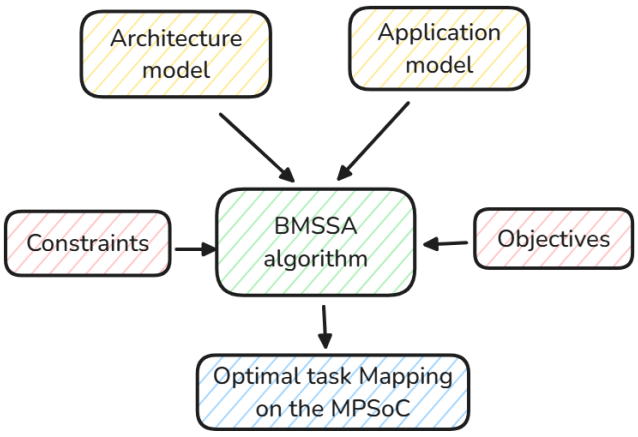


Figure 6: Global description of our solution

4.4 Algorithm complexity

The BMSSA has the following computational complexity for task mapping on NoC-based heterogeneous MPSoCs, considering n tasks, p processing elements (PEs), population size N, archive size archive_size, and task graph edges $|E|$.

Time complexity The main computational steps per iteration (max $\_iter$) are: (i) position updates and binary transformation with complexity $O(N \cdot np)$, (ii) fitness evaluation (makespan and energy) with $O\big(N \cdot (n + |E|)\big)$, and (iii) archive management (non-dominance and crowding distance) with $O\big((N + \text{archive\_size})^2\big)$.

$$O\Big(\max\_iter \cdot \big(N \cdot np + N \cdot (n + |E|) + (N + \text{archive\_size})^2\big)\Big) \quad (12)$$

Space complexity The population and archive require $O\big((N + \text{archive\_size}) \cdot np\big)$ storage.

This complexity is acceptable for typical benchmarks (e.g., $n \leq 50$ and $p \leq 20$), although archive management may become a bottleneck for larger scales.

5 Experimentation and results

This section evaluates the proposed Binary Multi-Objective Salp Swarm Algorithm (BMSSA) for task mapping on NoC-based heterogeneous MPSoCs. The optimization aims to jointly minimize execution time (makespan) and total energy (computation + communication). BMSSA is compared with three established multi-objective baselines: Multi-Objective Particle Swarm Optimization (MOPSO) [32], Non-dominated Sorting Genetic Algorithm II (NSGA-II) [33], and Multi-Objective Ant Colony Optimization (MOACO) [34].

Table 2: Parameter configuration for the evaluated algorithms.

Algorithm	Parameters
BMSSA	c_1: linearly decreased from 2 to 0
NSGA-II	Crossover probability = 0.8, Mutation probability = 0.1
MOPSO	Inertia weight = 0.4, Personal and social learning coefficients = 2
MOACO	Pheromone influence (α) = 1, Heuristic influence (β) = 2, Evaporation rate (ρ) = 0.5

5.1 Implementation and experimental setup

All algorithms were implemented in Matlab and executed on a machine with an Intel® Core™ i5-7300HQ CPU under Windows 10. Benchmarks consist of task graphs with 10–50 tasks mapped onto 2D-mesh NoC platforms with 4–20 processing elements (PEs). To account for stochasticity and ensure statistical robustness, each configuration was run for 100 iterations and repeated independently 30 times; unless otherwise noted, we report the *median* over the 30 runs. For a fair comparison, all methods were given similar computational budgets, and parameters followed recommendations from the original papers with light tuning (Table 2).

5.2 Comparative results of the multi-objective mapping

Table 3 together with the per-configuration bar charts (Figs. 7) and the cross-configuration trends (Fig. 8) show that BMSSA is *highly competitive* across the benchmark set, though not universally dominant. Counting ties, BMSSA attains the best execution time in 5/9 configurations and the best energy in 6/9 configurations (Table 4). In small-to-medium platforms (4–14 PEs), the bar plots highlight frequent BMSSA wins with narrow margins over NSGA-II and MOACO (e.g., 4/10 and 10/25). For larger platforms, leadership alternates: at 16/40 and 20/50, MOACO achieves both the fastest time and the lowest energy (e.g., 16/40: 5.4 s and 16.7 J vs. BMSSA 5.5 s and 16.9 J), and at 18/45, NSGA-II leads on both metrics (6.2 s and 18.2 J vs. BMSSA 6.4 s and 18.8 J). These observa-

Table 3: Comparison of BMSSA, MOPSO, NSGA-II, and MOACO on NoC Mapping

Proc.	Tasks	BMSSA		MOPSO		NSGA-II		MOACO	
		Time (s)	Energy (J)	Time (s)	Energy (J)	Time (s)	Energy (J)	Time (s)	Energy (J)
4	10	**0.6**	**5.1**	1.2	5.6	1.1	5.4	0.7	5.2
6	15	**1.5**	**6.9**	1.7	7.1	1.6	7.0	1.6	6.9
8	20	**2.3**	**8.4**	2.5	8.9	2.4	8.7	2.4	8.5
10	25	**2.9**	**10.4**	3.3	10.8	3.0	10.6	3.0	10.5
12	30	3.8	**12.3**	4.0	12.6	3.8	12.4	**3.7**	12.3
14	35	**4.5**	**14.6**	4.8	15.0	4.5	14.8	4.6	14.7
16	40	5.5	16.9	5.7	17.0	5.5	16.8	**5.4**	**16.7**
18	45	6.4	18.8	6.5	18.7	**6.2**	**18.2**	6.3	18.3
20	50	6.9	20.1	7.3	20.5	7.0	20.3	**6.8**	**20.0**

Table 4: Count of configurations where each method attains the best value.

Method	Best Time (count / 9)	Best Energy (count / 9)
BMSSA	**5**	**6**
NSGA-II	1	1
MOACO	3	2
MOPSO	0	0

tions correct earlier overstatements and confirm that different optimizers can prevail depending on scale and traffic.

The trend lines in Fig. 8 clarify how performance evolves with instance size: time and energy increase monotonically for all methods; BMSSA generally traces the lower envelope, but its curves intersect with MOACO around 16–20 PEs and with NSGA-II near 18 PEs precisely where Table 3 reports non-BMSSA optima. Across all nine configurations, BMSSA remains near-optimal: its average relative gap to the best method is only about 1.03% in time and 0.55% in energy. Meanwhile, MOPSO is consistently dominated on both metrics in the tables and figures.

Overall, BMSSA offers the strongest balance across configurations, leading most often and staying very close to the best elsewhere, while explicitly acknowledging the specific large-scale setups where NSGA-II or MOACO take the lead. This nuanced, data-aligned interpretation reconciles the table and figures and removes any absolute claims.

These performance patterns reflect how each algorithm interacts with NoC-specific characteristics, such as topology size, traffic distribution, and platform heterogeneity. BMSSA performs best in small to medium NoCs (e.g., 4–14 PEs), where early placement of heavily communicating tasks onto nearby processing elements (PEs) reduces communication hops, avoids congestion, and improves both execution time and energy. Its leader–follower model and archive-based exploration help quickly find such balanced mappings.

As the NoC grows (16–20 PEs), communication paths lengthen and traffic becomes more complex. Here, the advantages of local clustering fade, and global traffic balancing becomes more important. MOACO, with its pheromone-based construction, and NSGA-II, with recombination and selection pressure, can better reorganize mappings across the mesh, explaining their wins in larger setups like 16/40, 18/45, and 20/50.

Task graph structure also matters. When communication is focused on a few tasks, BMSSA can easily cluster them. However, when traffic is more evenly spread, broader search methods like MOACO and NSGA-II have an edge.

BMSSA also benefits from platform heterogeneity by assigning compute-heavy tasks to stronger PEs early and placing their partners nearby. However, larger systems may require redistributing tasks more widely, where other methods adapt better.

MOPSO underperforms in all cases due to early convergence and a lack of diversity control, limiting its ability to adapt to changing trade-offs.

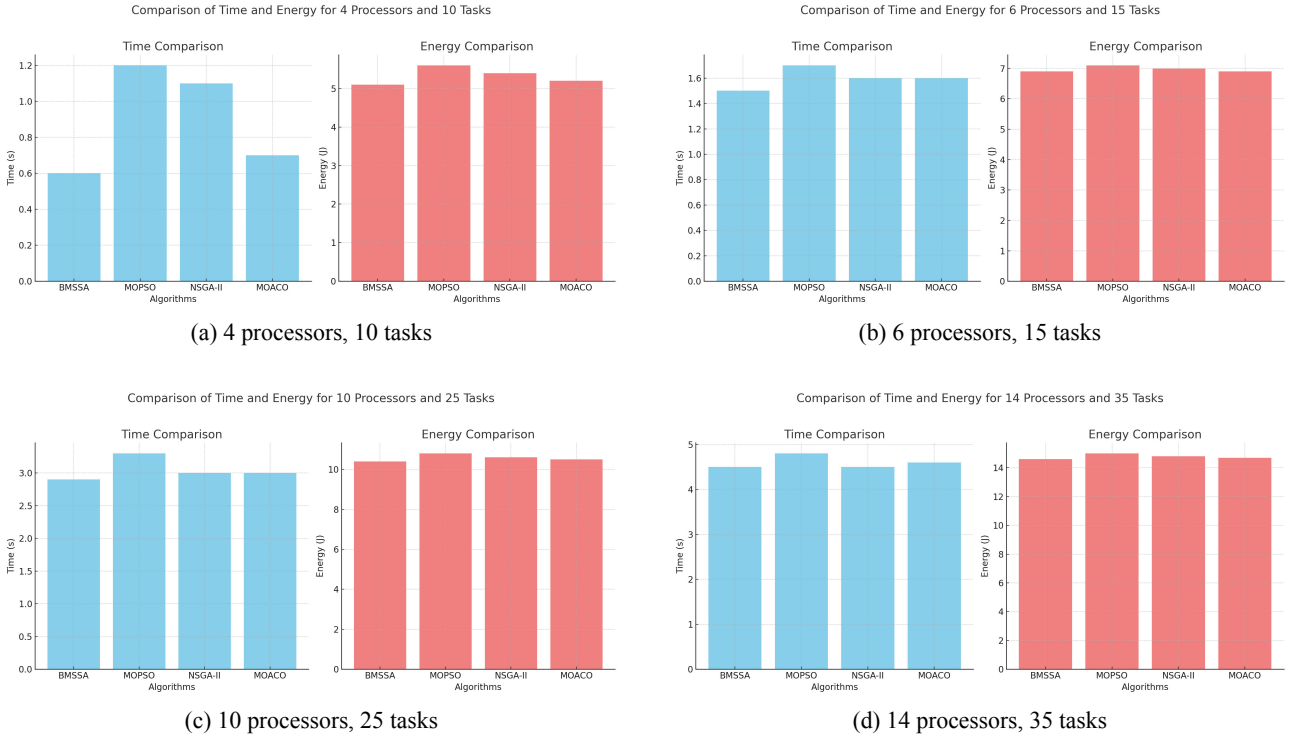
In summary, BMSSA is strongest when local, early decisions have high impact, while NSGA-II and MOACO may excel in larger or more uniformly loaded platforms.

5.3 Convergence analysis

Figures 9 and 10 show, for the 10 processors / 25 tasks case, the *median best-so-far* values over 30 runs (at each iteration, the lowest time or energy seen so far in a run, then the median across runs). The curves allow us to compare how quickly each method improves and what level it reaches at the end.

BMSSA decreases fastest at the beginning of both time and energy, then continues to improve more slowly and finishes with the lowest final values among the four methods in this setup. This matches Table 3 for the 10/25 case. NSGA-II improves steadily and regularly from start to finish; it does not drop as quickly as BMSSA early on, but ends close behind with slightly higher final values for both objectives. MOACO starts slower, with gentler early decreases, then accelerates in later iterations and becomes competitive by the end, especially in terms of energy, ending very close to NSGA-II. MOPSO improves quickly at first but soon plateaus; its curves flatten early, and it finishes with the highest (worst) time and energy.

Figure 7: Comparison of Time and Energy for Different Configurations



(a) 4 processors, 10 tasks

(b) 6 processors, 15 tasks

(c) 10 processors, 25 tasks

(d) 14 processors, 35 tasks

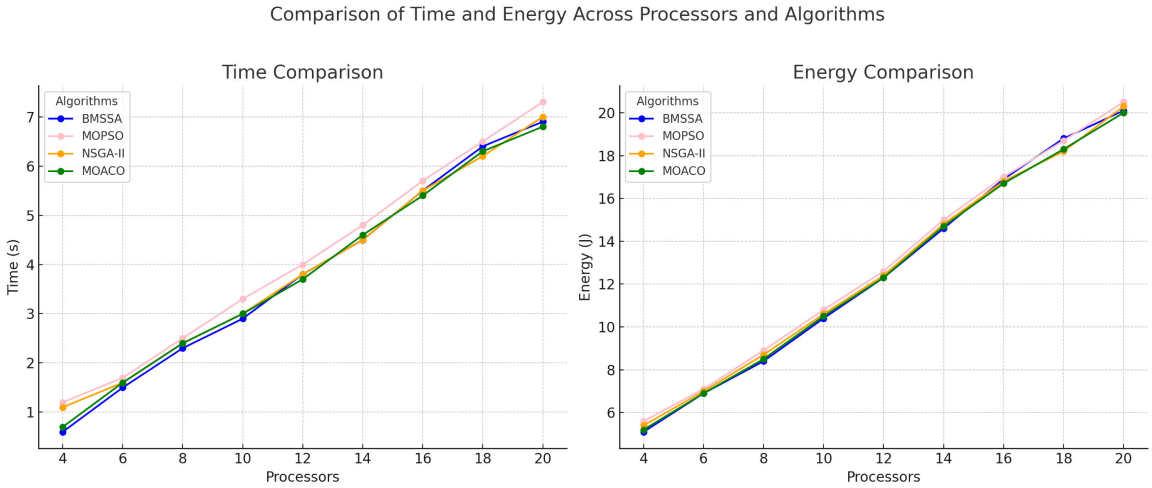


Figure 8: Comparison of time and energy across processors and algorithms

Overall, these plots tell a consistent story: BMSSA finds strong mappings early and reaches the best final values on this instance; NSGA-II and MOACO follow different improvement patterns yet end up close, while MOPSO stalls early. This agrees with the results in Table 3 and helps explain why BMSSA often outperforms NSGA-II or MOACO on small and medium platforms, whereas NSGA-II or MOACO can be preferable in some larger settings.

5.4 Wilcoxon signed-rank test

To assess the statistical significance of the differences between BMSSA and the baseline algorithms (MOPSO, NSGA-II, MOACO) in terms of makespan and total energy consumption, we applied the non-parametric Wilcoxon signed-rank test [37]. This test was performed pairwise, using the median values from the experimental results table across the nine benchmark configurations (4–20 processors, 10–50 tasks). The null hypothesis (H_0) assumes no significant difference between the paired samples ($p > 0.05$ indicates failure to reject H_0; $p \leq 0.05$ indicates significance).

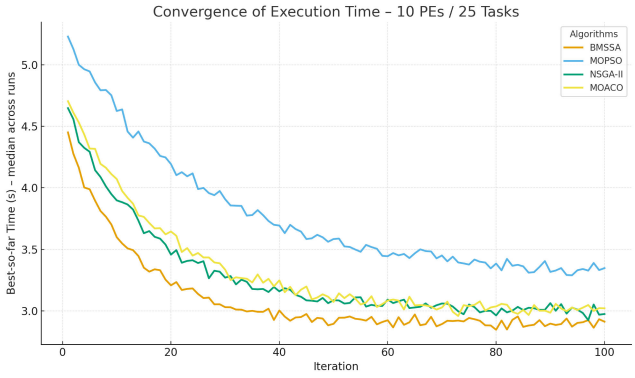


Figure 9: Convergence of execution time-10 PEs/tasks

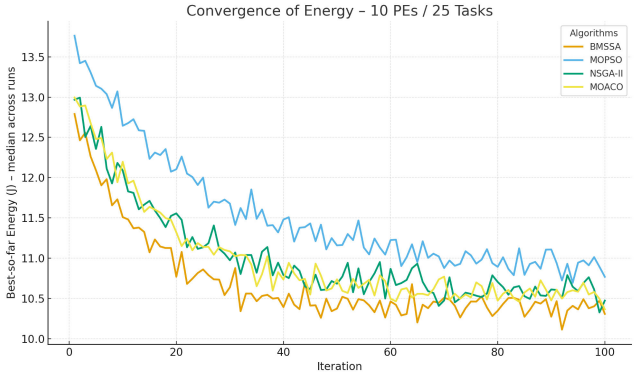


Figure 10: Convergence of energy-10 PEs/tasks

The tests confirm BMSSA's superiority over MOPSO in both objectives (significant reductions). At the same time, differences with NSGA-II and MOACO are generally not statistically significant, aligning with the observed close performance in larger benchmarks.

Table 5: Wilcoxon test results for makespan.

BMSSA vs.	p-value	Significance
MOPSO	**0.0039**	Yes ($p < 0.05$)
NSGA2	0.2436	No
MOACO	0.9102	No

Table 6: Wilcoxon test results for total energy.

BMSSA vs.	p-value	Significance
MOPSO	**0.0078**	Yes ($p < 0.05$)
NSGA2	0.1641	No
MOACO	0.4959	No

6 Conclusion

This paper investigated application mapping for NoC-based heterogeneous MPSoCs using a Binary Multi-Objective Salp Swarm Algorithm (BMSSA). We benchmarked BMSSA against NSGA-II, MOPSO, and MOACO, focusing on the dual objectives of minimising time and energy.

Across nine platform–workload configurations, BMSSA achieved the best execution time in 5/9 cases and the best energy in 6/9 cases, while remaining close to the best elsewhere (average relative gap ≈1.03% in time and ≈0.55% in energy). Notably, NSGA-II led at 18 PEs / 45 tasks, and MOACO led at 16 and 20 PEs, underscoring that different optimizers can prevail on specific large-scale settings. Complementing these pointwise results, BMSSA exhibited favorable convergence on the representative setup studied, indicating strong overall solution quality.

In summary, BMSSA provides a robust and competitive trade-off for discrete NoC mapping: it leads in most configurations, stays near the best otherwise, and converges rapidly to the optimal solutions. Future work includes scaling to larger NoCs and task graphs, integrating energy-aware routing and dynamic re-mapping, and validating on additional real-world workloads (e.g., multimedia and ML pipelines).

References

[1] Zurawski, R. (2018) *Embedded Systems Handbook: Embedded Systems Design and Verification*, CRC Press. https://doi.org/10.1201/9781315218281-13

[2] Aroui, A., Boulet, P., Benhaoua, K., Singh, A. K., et al. (2020) Novel metric for load balance and congestion reducing in network on-chip, *Scalable Computing: Practice and Experience*, 21(2):309–321. https://doi.org/10.12694/scpe.v21i2.1690

[3] Yu, J. (2025) Intelligent Construction Scheduling Based on MOEA/D-DE, SPEA2+ SDE, and NSGA-III by Integrating Safety Assessment with Resource Efficiency, *Informatica*, 49(11). https://doi.org/10.31449/inf.v49i11.7147

[4] Laredj, K., Belarbi, M., and Benyamina, A. E. (2019) Metrics for Real-Time Solutions Design, *Intelligent Computing: Proceedings of the 2018 Computing Conference, Volume 2*, Springer, pp. 411–425. https://doi.org/10.1007/978-3-030-01177-2_30

[5] Boumaza, F., Zouache, D., Belazzoug, M., Hadji, A., and Aroui, A. (2024) Binary Grey Wolf Optimizer for Mapping Real Time Applications on MPSOCs Architecture.

[6] Ali, H., Tariq, U. U., Zheng, Y., Zhai, X., and Liu, L. (2018) Contention & energy-aware real-time task mapping on NoC based heterogeneous MPSoCs, *IEEE Access*, 6:75110–75123. https://doi.org/10.1109/ACCESS.2018.2882941

[7] Zhang, Y. (2025) Enhanced Multi-objective Artificial Physics Optimization Algorithms for Solution Set Distributions, *Informatica*, 49(6). https://doi.org/10.31449/inf.v49i6.7024

[8] Wen, Q. (2025) Multi objective optimization system for bridge design based on multi-objective optimization theory and improved ant colony algorithm, *Informatica*, 49(8). https://doi.org/10.31449/inf.v49i8.7220

[9] Boumaza, F., Benyamina, A. E. H., Zouache, D., Abualigah, L., and Alsayat, A. (2023) An Improved Harris Hawks Optimization Algorithm Based on Bi-Goal Evolution and Multi-Leader Selection Strategy for Multi-Objective Optimization, *Ingénierie des Systèmes d'Information*, 28(5). https://doi.org/10.18280/isi.280503

[10] Schranzhofer, A., Chen, J.-J., and Thiele, L. (2010) Dynamic power-aware mapping of applications onto heterogeneous MPSoC platforms, *IEEE Transactions on Industrial Informatics*, 6(4):692–707. https://doi.org/10.1109/TII.2010.2062192

[11] Muhsen, Y. R., Husin, N. A., Zolkepli, M. B., Manshor, N., Al-Hchaimi, A. A. J., and Ridha, H. M. (2023) Enhancing NoC-based MPSoC performance: a predictive approach with ANN and guaranteed convergence arithmetic optimization algorithm, *IEEE Access*. https://doi.org/10.1109/ACCESS.2023.3305669

[12] Mirjalili, S., Gandomi, A. H., Mirjalili, S. Z., Saremi, S., Faris, H., and Mirjalili, S. M. (2017) Salp Swarm Algorithm: A bio-inspired optimizer for engineering design problems, *Advances in Engineering Software*, 114:163–191. https://doi.org/10.1016/j.advengsoft.2017.07.002

[13] Deb, K., Agrawal, S., Pratap, A., and Meyarivan, T. (2000) A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: NSGA-II, *Parallel Problem Solving from Nature PPSN VI: 6th International Conference, Paris, France, September 18–20, 2000 Proceedings*, Springer, pp. 849–858. https://doi.org/10.1007/3-540-45356-3_83

[14] Lin, Q., Li, J., Du, Z., Chen, J., and Ming, Z. (2015) A novel multi-objective particle swarm optimization with multiple search strategies, *European Journal of Operational Research*, 247(3):732–744. https://doi.org/10.1016/j.ejor.2015.06.071

[15] Moein-Darbari, F., Khademzade, A., and Gharooni-Fard, G. (2009) Cgmap: a new approach to network-on-chip mapping problem, *IEICE Electronics Express*, 6(1):27–34. https://doi.org/10.1587/elex.6.27

[16] GE, F., and WU, N. (2010) Genetic algorithm based mapping and routing approach for network on chip architectures, *Chinese Journal of Electronics*, 19(1):91–96. https://cje.ejournal.org.cn/article/id/5132

[17] Tosun, S., Ozturk, O., and Ozen, M. (2009) An ILP formulation for application mapping onto network-on-chips, *Proceedings of the 2009 International Conference on Application of Information and Communication Technologies*, IEEE, pp. 1–5. https://doi.org/10.1109/ICAiCT.2009.5372524

[18] Liu, Y., Ruan, Y., Lai, Z., and Jing, W. (2011) Energy and thermal aware mapping for mesh-based NoC architectures using multi-objective ant colony algorithm, *Proceedings of the 2011 3rd International Conference on Computer Research and Development*, Vol. 3, IEEE, pp. 407–411. https://doi.org/10.1109/ICCRD.2011.5764225

[19] Jang, W., and Pan, D. Z. (2012) A3MAP: Architecture-aware analytic mapping for networks-on-chip, *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 17(3):1–22. https://doi.org/10.1145/2209291.2209299

[20] Tosun, S. (2011) New heuristic algorithms for energy aware application mapping and routing on mesh-based NoCs, *Journal of Systems Architecture*, 57(1):69–78. https://doi.org/10.1016/j.sysarc.2010.10.001

[21] Sahu, P. K., Sharma, A., and Chattopadhyay, S. (2012) Application mapping onto mesh-of-tree based network-on-chip using discrete particle swarm optimization, *Proceedings of the 2012 International Symposium on Electronic System Design (ISED)*, IEEE, pp. 172–176. https://doi.org/10.1109/ISED.2012.17

[22] Obaidullah, M., and Khan, G. N. (2017) Application mapping to mesh NoCs using a tabu-search based swarm optimization, *Microprocessors and Microsystems*, 55:13–25. https://doi.org/10.1016/j.micpro.2017.09.004

[23] Wang, X., Sun, Y., Gu, H., and Liu, Z. (2018) WOAGA: A new metaheuristic mapping algorithm for large-scale mesh-based NoC, *IEICE Electronics Express*, 15(17):20180738. https://doi.org/10.1587/elex.15.20180738

[24] Alagarsamy, A., Gopalakrishnan, L., Mahilmaran, S., and Ko, S.-B. (2019) A self-adaptive mapping approach for network on chip with low power consumption, *IEEE Access*, 7:84066–84081. https://doi.org/10.1109/ACCESS.2019.2925381

[25] Alagarsamy, A., and Gopalakrishnan, L. (2018) MBA: a new cluster based bandwidth and power

aware mapping for 2D NoC, *Proceedings of the 2018 International Conference on Circuits and Systems in Digital Enterprise Technology (ICCSDET)*, IEEE, pp. 1–4. https://doi.org/10.1109/ICCSDET.2018.8821150

[26] Wang, X., Sun, Y., and Gu, H. (2019) BMM: A binary metaheuristic mapping algorithm for mesh-based network-on-chip, *IEICE Transactions on Information and Systems*, 102(3):628–631. https://doi.org/10.1587/transinf.2018edl8208

[27] Mohiz, M. J., Baloch, N. K., Hussain, F., Saleem, S., Zikria, Y. B., and Yu, H. (2021) Application mapping using cuckoo search optimization with Lévy flight for NoC-based system, *IEEE Access*, 9:141778–141789. https://doi.org/10.1109/ACCESS.2021.3120079

[28] Darbandi, M., Ramtin, A. R., and Sharafi, O. K. (2020) Tasks mapping in the network on a chip using an improved optimization algorithm, *International Journal of Pervasive Computing and Communications*, 16(2):165–182. https://doi.org/10.1108/IJPCC-07-2019-0053

[29] Sikandar, S., Baloch, N. K., Hussain, F., Amin, W., Zikria, Y. B., and Yu, H. (2021) An optimized nature-inspired metaheuristic algorithm for application mapping in 2D-NoC, *Sensors*, 21(15):5102. https://doi.org/10.3390/s21155102

[30] Mehmood, F., Baloch, N. K., Hussain, F., Amin, W., Hossain, M. S., Zikria, Y. B., and Yu, H. (2022) An efficient and cost effective application mapping for network-on-chip using Andean condor algorithm, *Journal of Network and Computer Applications*, 200:103319. https://doi.org/10.1016/j.jnca.2021.103319

[31] Amin, W., Hussain, F., and Anjum, S. (2022) iH-PSA: An improved bio-inspired hybrid optimization algorithm for task mapping in Network on Chip, *Microprocessors and Microsystems*, 90:104493. https://doi.org/10.1016/j.micpro.2022.104493

[32] Coello Coello, C. A., and Salazar Lechuga, M. (2002) MOPSO: A proposal for multiple objective particle swarm optimization, *Proceedings of the 2002 Congress on Evolutionary Computation (CEC'02)*, Vol. 2, IEEE, pp. 1051–1056. https://doi.org/10.1109/CEC.2002.1004388

[33] Chatterjee, S., Sarkar, S., Dey, N., Ashour, A. S., and Sen, S. (2018) Hybrid non-dominated sorting genetic algorithm: II-neural network approach, *Advancements in Applied Metaheuristic Computing*, IGI Global, pp. 264–286. https://doi.org/10.4018/978-1-5225-4151-6.ch011

[34] Awadallah, M. A., Makhadmeh, S. N., Al-Betar, M. A., Dalbah, L. M., Al-Redhaei, A., Kouka, S., and Enshassi, O. S. (2024) Multi-objective Ant Colony Optimization, *Archives of Computational Methods in Engineering*, pp. 1–43. https://doi.org/10.1007/s11831-024-10178-4

[35] Zouache, D., Abualigah, L., and Boumaza, F. (2024) A guided epsilon-dominance arithmetic optimization algorithm for effective multi-objective optimization in engineering design problems, *Multimedia Tools and Applications*, 83(11):31673–31700. https://doi.org/10.1007/s11042-023-16633-x

[36] Kumar, A. S., and Reddy, B. N. K. (2023) An efficient real-time embedded application mapping for NoC based multiprocessor system on chip, *Wireless Personal Communications*, 128(4):2937–2952. https://doi.org/10.21203/rs.3.rs-880912/v1

[37] Taheri, S. M., and Hesamian, G. (2013) A generalization of the Wilcoxon signed-rank test and its applications, *Statistical Papers*, 54(2):457–470. https://doi.org/10.1007/s00362-012-0443-4