Binary Grey Wolf Optimizer for Mapping Real Time Applications on MPSOCs Architecture*

Farid Boumaza[1,2,*,†], Djaafar Zouache[1,†], Mouhoub Belazzoug[1,†], Atmane Hadji[3,†] and Abdelkader Aroui[4,†]

[1]Computer Science Department, University of Mohamed El Bachir El Ibrahimi, Bordj Bou Arreridj 34030, Algeria
[2](LAPECI) Laboratory of Parallel, Embedded architectures and Intensive Computing, 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

Abstract

With the growing complexity of real-time applications, Multi-Processor Systems-on-Chip (MPSoCs) have become a vital solution for meeting stringent performance, power, and scalability requirements. Efficient task mapping plays a crucial role in optimizing the performance of such systems, particularly for real-time applications that demand strict timing constraints. Traditional mapping techniques, including static and dynamic strategies, struggle with balancing execution time, energy efficiency, and communication overhead in heterogeneous MPSoC architectures.

In this paper, we propose a novel approach for optimizing task mapping in MPSoCs, based on the Grey Wolf Optimizer (GWO), a bio-inspired metaheuristic renowned for its effectiveness in solving complex optimization problems. This methodology aims to minimize task execution times and communication delays by intelligently mapping real-time tasks onto heterogeneous processing elements (PEs), while also adhering to real-time constraints and reducing energy consumption.

The results confirm that the improved GWO algorithm is a powerful tool for addressing the challenges of real-time task mapping in MPSoCs, providing a robust and scalable solution for future embedded systems.

Keywords

Multi-Processor Systems-on-Chip, Grey Wolf Optimizer, Mapping, Energy Optimization

1. Introduction

Embedded systems [1, 2] are specialized electronic systems designed for specific applications, typically operating without conventional input/output interfaces like keyboards or screens. They often incorporate one or more systems-on-chip (SoCs), which are inherently heterogeneous and complex, comprising various processors such as FPGAs, DSPs, and general-purpose processors (GPs), each supporting dedicated or reconfigurable functions.

Embedded applications are frequently complex and often hierarchical, as seen in applications like MPEG [3], H.263, and H.264 encoders [4]. These applications consist of components requiring diverse processing approaches, generally divided into two main types: irregular processing, which involves task-level parallelism, and regular processing, which focuses on data-level parallelism. The latter often represents high-performance computing (HPC) functions commonly found in embedded real-time applications.

Given the specific demands of hierarchical applications, a single mapping strategy is often insufficient to address both processing types optimally. Instead, an effective approach requires separate handling of

Proceedings of the International IAM'24: International Conference on Informatics and Applied Mathematics, December 04–05, 2024, Guelma, Algeria

*Corresponding author.
†These authors contributed equally.
f.boumaaza@univ-bba.dz (F. Boumaza); djaafarzouache@yahoo.fr (D. Zouache); belazoug.mouhoub@gmail.com (M. Belazzoug); a.hadji@centre-univ-mila.dz (A. Hadji); aroui_kader@yahoo.fr (A. Aroui)
0000-0002-9785-420X (F. Boumaza); 0000-0002-0337-6105 (D. Zouache); 0000-0002-7174-5486 (M. Belazzoug); 0000-0001-6706-6360 (A. Hadji); 0000-0002-1024-2033 (A. Aroui)

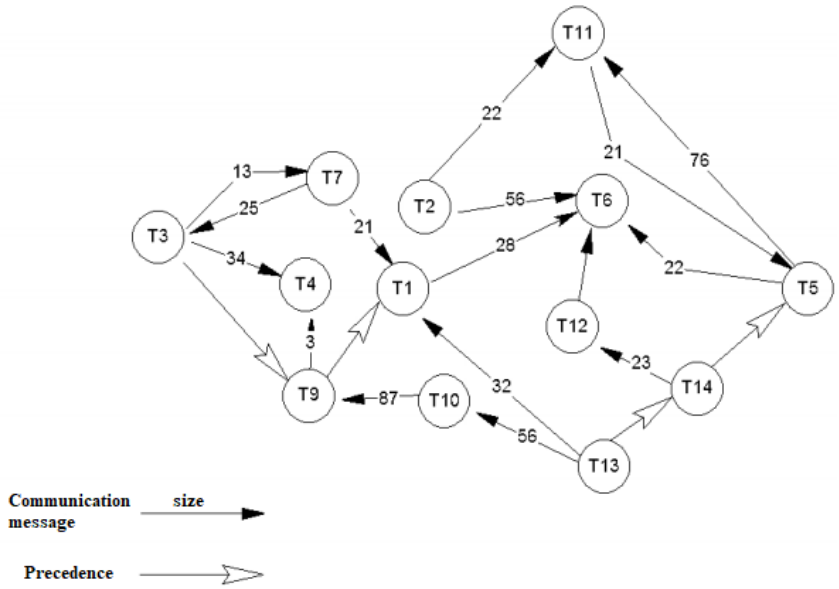

Figure 1: Application graph.

regular and irregular processing components [5, 6]. This leads us to propose a new strategy involving hierarchical mapping, where a global strategy is applied to irregular (parallel processing) tasks.

Our approach uses the Binary Multi-objective Grey Wolf Optimizer (BMOGWO) to optimize the mapping on the MPSoC architecture, these techniques enable efficient task mapping that addresses both real-time constraints and energy optimization in complex embedded applications.

The rest of the paper is organized as follows: Section 2 discusses the definitions and the necessary mathematical formulations for problem mapping onto the MPSoCs architecture. In Section 3, the proposed mapping strategies are presented. Experimental results are provided in Section 4, and the paper concludes with Section 5.

2. Definition and formulation

Communications between the tasks of our application and the components of our target architecture is represented by two directed graphs.

Definition 1
The application graph, also known as the Task Graph (TG), is a directed graph $G(T, E)$, where each vertex $t_i \in T$ represents a module or task within the application. Each directed edge (t_i, t_j), denoted as $e_{ij} \in E$, signifies a communication link between tasks t_i and t_j. The weight of the edge e_{ij}, represented by Q_{ij}, indicates the volume of data transferred between t_i and t_j, reflecting communication demand and aiding in optimizing resource allocation (see Fig. 1).

Definition 2
The architecture graph, denoted as AG (Architecture Graph), is a directed graph $P(S, F)$ where each vertex $s_i \in S$ represents a node within the topology. The directed edge (s_i, s_j), denoted as $f_{ij} \in F$, signifies a physical link that directly connects two elements, s_i and s_j, within the architecture. The weight of the edge f_{ij}, represented by bw_{ij}, encapsulates critical characteristics of the physical link, including bandwidth, latency, and energy consumption. This comprehensive representation allows for effective analysis and optimization of communication pathways in Multi-Procesur-on-Chip (MPSoC) architectures (see Fig. 2).

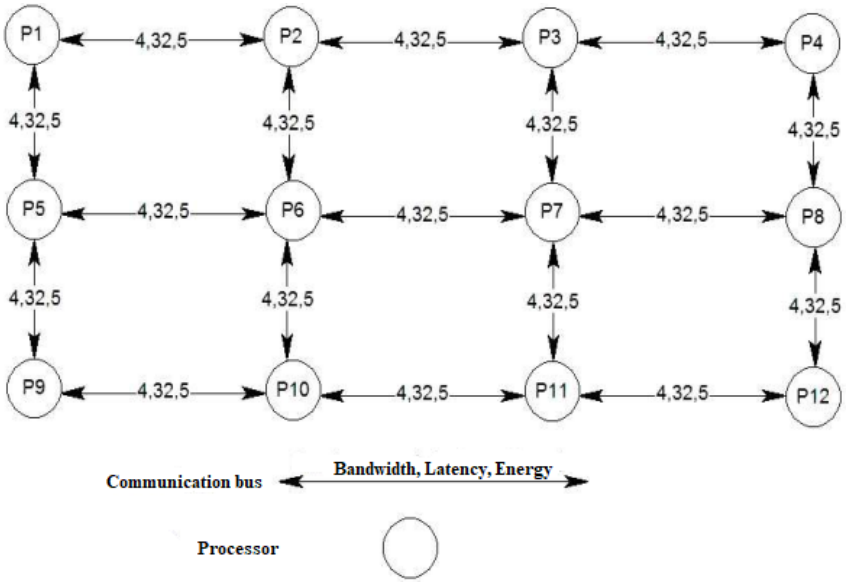

Figure 2: Architecture graph.

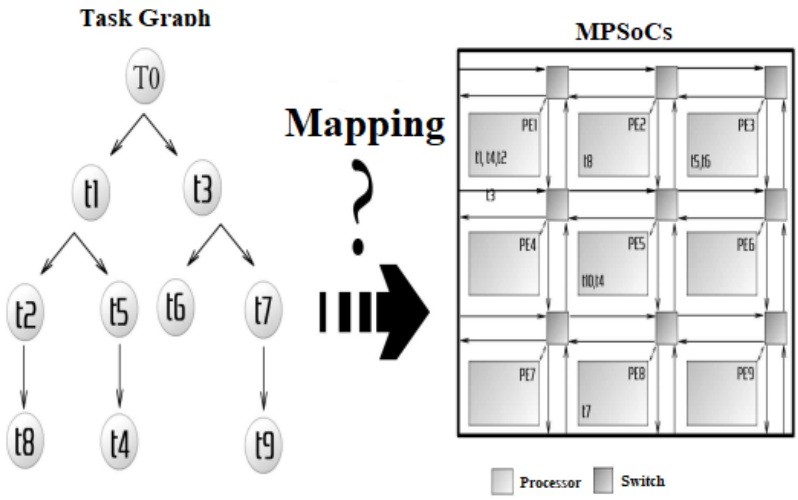

Figure 3: Mapping graph task on MPSoCs architecture.

Definition 3

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

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

This mapping is valid under the condition that the number of tasks $|T|$ is greater than or equal to the number of processing elements $|S|$ (see Fig. 3) [7].

The mapping process is crucial in optimizing resource allocation within MPSoCs, as it determines how application tasks are distributed across the available processing elements. An effective mapping strategy not only facilitates parallel execution of tasks but also minimizes communication overhead, ensuring that system performance and energy efficiency are maximized [8]. This approach is especially important in heterogeneous architectures, where diverse processing capabilities must be leveraged to meet the demands of complex applications.

Our application is defined as a set of tasks $T = \{t_1, t_2, \ldots, t_n\}$, while the target architecture is

represented as a set of processors $P = \{p_1, p_2, \ldots, p_m\}$. It is important to note that each processor can operate in multiple modes, denoted as m_1, m_2, m_3. This capability of processors to function in various modes introduces greater diversity in optimization strategies for task placement.

By enabling processors to adapt their operational modes based on the specific requirements of tasks, we can achieve more efficient resource utilization and improved performance. This multi-mode functionality allows for fine-tuning of processing capabilities, enabling better handling of both regular and irregular task types. As a result, it facilitates enhanced flexibility in mapping strategies, leading to optimized execution times and reduced energy consumption within the multiprocessor system-on-chip (MPSoC) framework [9].

2.1. Execution Time and Communication Duration

The execution time $D$of a task is defined as follows:

$$D_i = \frac{\text{Taille}(t_i^p)}{f_{mp}} \tag{2}$$

where: - D_i is the execution time of task i on processor p. - Taille(t_i^p) denotes the size of task i for processor p. - f_{mp} is the frequency of processor p operating in mode m.

The overall execution time for the application is given by:

$$D = \max(D_i) \quad \text{for} \quad i = 1, \ldots, \text{number of tasks} \tag{3}$$

where D is the maximum execution time among all tasks D_i, taking into account their dependencies.

The duration of communication D_{com}^{ij} between tasks i and j is calculated as:

$$D_{com}^{ij} = \frac{Q_{ij}}{\text{Min } B_{P|p_k,p_l|}} \times \sum \text{latency}(p_k, p_l) \tag{4}$$

where: - D_{com}^{ij} is the communication duration between tasks i and j. - Q_{ij} represents the data volume that needs to be communicated between tasks i and j. - Min $B_{P|p_k,p_l|}$ denotes the minimum bandwidth of the path connecting processors p_k and p_l. - The summation $\sum \text{latency}(p_k, p_l)$ accounts for the cumulative latency across the communication path.

This approach ensures that both execution and communication times are adequately considered for optimizing task mapping in MPSoCs, thereby improving overall system performance and efficiency.

2.2. Energy Consumption

The energy consumption E in a multiprocessor system-on-chip (MPSoC) can be categorized into execution energy and communication energy.

2.2.1. Execution Energy

The energy consumed during the execution of a task i on processor p in mode m is defined as:

$$E_{\text{exec}}^i = \text{Size}_{ip} \times e_{mp} \tag{5}$$

where:
- E_{exec}^i is the execution energy of task i.
- Size$_{ip}$ represents the number of cycles required for task i to execute on processor p in mode m.
- e_{mp} denotes the energy consumption per cycle for processor p in mode m.

2.2.2. Communication Energy

The energy consumed due to communication between tasks i and j assigned to processors p and q is given by:

$$E_{\text{com}}^{ijpq} = \sum_{i=1}^{n} Q_{ij} \times e_{p_l,p_k} \tag{6}$$

where:
- E_{com}^{ijpq} is the communication energy between tasks i and j.
- Q_{ij} represents the volume of data exchanged between tasks i and j.
- e_{p_l,p_k} is the energy cost associated with the communication link between processors p_l and p_k.

2.2.3. Total Energy Consumption

The total energy consumption for executing all tasks and their communications within the system can be expressed as:

$$E_{\text{total}} = \sum_{i=1}^{\text{task}_n} (E_{\text{exec}^i} + E_{\text{com}}^{ijpq}) \tag{7}$$

where:
- E_{total} is the overall energy consumption for all tasks in the application.
- The summation encompasses both the execution energy and communication energy across all tasks.

2.3. Task Placement Indicator

An auxiliary variable X_m^{ip} is used to indicate the placement of tasks on processors, defined as follows:

$$X_m^{ip} = \begin{cases} 1 & \text{if task } i \text{ is placed on processor } p \text{ and operates in mode } m \\ 0 & \text{otherwise} \end{cases} \tag{8}$$

This formulation allows for a comprehensive analysis of energy consumption during task execution and inter-task communication, facilitating the optimization of task mapping strategies in MPSoCs. By minimizing both execution and communication energy, the overall efficiency and sustainability of the system can be significantly enhanced.

3. Proposed Resolution Method

The comprehensive problem we aim to address is the Assignment, and Scheduling (AS) problem. This encompasses the assignment and scheduling of application tasks and their associated communications onto the resources of a target architecture, with the objective of achieving specified performance metrics.

In our approach, we consider multiple objectives, including minimizing energy consumption and maximizing performance efficiency. These goals are crucial for the operation of mobile embedded systems, as they directly influence battery life [10]. Reducing energy consumption is essential to prolong battery longevity, while maximizing execution speed necessitates minimizing task completion times. However, these objectives often conflict; for example, operating components in energy-saving modes can lead to increased execution times.

To navigate these conflicting objectives, we adopt a multi-objective optimization strategy that seeks to find an effective compromise among the various goals. Specifically, we propose an approach utilizing the Multi-Objective Grey Wolf Optimizer (MOGWO) technique. This method is improved to efficiently tackle the AS problem by balancing energy efficiency with performance requirements, ultimately enhancing the overall effectiveness of task mapping in MPSoCs.

By integrating these advanced optimization techniques, our approach aims to provide a robust solution that aligns with the dynamic demands of mobile embedded systems while addressing the critical constraints of energy consumption and execution time.

3.1. Grey Wolf Optimizer (GWO)

The Grey Wolf Optimizer (GWO), introduced by [11]. in 2014, is a metaheuristic algorithm inspired by the natural hierarchy and hunting behavior of grey wolves. In GWO, wolves are categorized into four social ranks that determine their roles in the optimization process:

- **Alpha (α) Wolf**: Holds the best solution based on the objective function.
- **Beta (β) Wolf**: Holds the second-best solution.
- **Delta (σ) Wolf**: Holds the third-best solution.
- **Omega (ω) Wolves**: All remaining solutions in the population, which follow the guidance of the top three wolves.

In the hunting process, the alpha, beta, and delta wolves primarily guide the search, while the omega wolves follow, refining their positions based on these leaders.

3.1.1. Stages of GWO

Grey wolves use three organized hunting stages in GWO: **encircling**, **hunting**, and **attacking**. The encircling behavior is mathematically represented by the following equations:

$$\vec{D} = \left| \vec{C} \cdot \vec{X}_p(t) - \vec{X}(t) \right| \tag{9}$$

$$\vec{X}(t+1) = \vec{X}_p(t) - \vec{A} \cdot \vec{D} \tag{10}$$

Here, t is the iteration counter, $\vec{X}$ represents the position of a wolf, $\vec{X}_p$ denotes the position of the prey, and $\vec{A}$ and $\vec{C}$ are coefficient vectors. The vectors $\vec{A}$ and $\vec{C}$ are defined as:

$$\vec{A} = 2 \cdot \vec{a} \cdot \vec{r}_1 - \vec{a} \tag{11}$$

$$\vec{C} = 2 \cdot \vec{r}_2 \tag{12}$$

where $\vec{r}_1$ and $\vec{r}_2$ are random vectors within $[0, 1]$, and the values of $\vec{a}$ decrease linearly from 2 to 0 as iterations progress.

3.1.2. Hunting Mechanism

In GWO, the best solutions (α, β, and δ) direct the search towards the optimal solution. The distance between each wolf and these leaders is computed as:

$$\vec{D}_\alpha = \left| \vec{C}_1 \cdot \vec{X}_\alpha - \vec{X} \right| \tag{13}$$

$$\vec{D}_\beta = \left| \vec{C}_2 \cdot \vec{X}_\beta - \vec{X} \right| \tag{14}$$

$$\vec{D}_\delta = \left| \vec{C}_3 \cdot \vec{X}_\delta - \vec{X} \right| \tag{15}$$

The wolves' updated positions are calculated as:

$$\vec{X}_1 = \vec{X}_\alpha - A_1 \cdot \vec{D}_\alpha \tag{16}$$

$$\vec{X}_2 = \vec{X}_\beta - A_2 \cdot \vec{D}_\beta \tag{17}$$

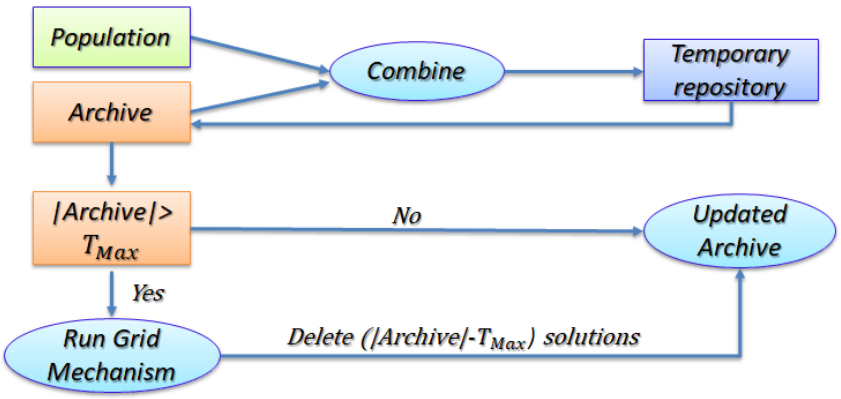

Figure 4: The management of the archive population.

$$\vec{X}_3 = \vec{X}_\delta - A_3 \cdot \vec{D}_\delta \tag{18}$$

The next position of a wolf is derived by averaging the three leaders' positions:

$$\vec{X}(t+1) = \frac{\vec{X}_1 + \vec{X}_2 + \vec{X}_3}{3} \tag{19}$$

3.1.3. Attacking Mechanism

The vector $\vec{a}$ controls the balance between exploration and exploitation, with its elements set within the range $[-a, a]$ and gradually reducing from 2 to 0. This is formulated as:

$$\vec{a} = 2 - t \cdot \frac{2}{maxIter} \tag{20}$$

where *maxIter* is the maximum number of iterations, and t is the current iteration. As the algorithm progresses, GWO transitions from exploration to exploitation, allowing wolves to converge towards the prey, representing the optimal solution.

3.2. GWO for Mullti-Objective Problems (MOGWO)

Despite the initial design of the Grey Wolf Optimizer (GWO) for single-objective problems, The authors in [12] (2016) extended the algorithm to address multi-objective optimization, introducing the Multi-Objective Grey Wolf Optimizer (MOGWO) as the first adaptation of GWO for multi-objective tasks. MOGWO integrates two crucial mechanisms for handling multiple objectives:

- An **archive**, or storage, maintains a set of non-dominated solutions during the optimization process, preserving diversity and facilitating Pareto front convergence.
- A **leader selection strategy** selects the first (α), second (β), and third (δ) leader solutions from the archive to guide the search.

The archive includes a control mechanism that determines whether new solutions should be added, which ensures it only holds relevant non-dominated solutions. At each iteration, newly obtained non-dominated solutions are compared with those already stored, updating the archive to reflect the best trade-offs achieved so far, as illustrated in Figure 4. In the multi-objective domain, comparing solutions is complex due to the Pareto front concept, where solutions are not directly comparable by a single measure [13]. To address this, MOGWO extends the GWO's hierarchy of the best three wolves (alpha, beta, and delta) by selecting leaders from the least crowded regions of the objective space. This selection mechanism directs the rest of the wolves towards promising regions, improving exploration

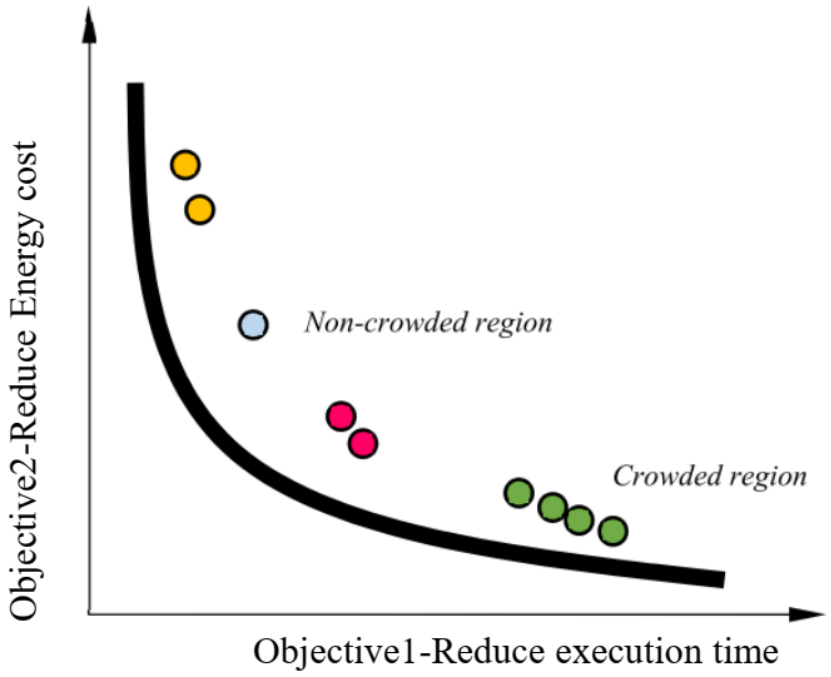

Figure 5: Managing solution diversity.

across the Pareto front and guiding convergence toward global optimality [14]. Figure 5 demonstrates how MOGWO prioritizes solutions from sparsely populated regions to ensure a well-distributed front.

MOGWO employs a roulette-wheel selection approach based on probabilities assigned to each segment or hypercube of the objective space:

$$P_j = \frac{m}{H_j} \tag{21}$$

Where H_j is the number of non-dominated solutions in the j-th segment, and m is a constant greater than 1. This approach ensures a bias toward selecting solutions from less crowded regions, fostering diversity and enhancing the search effectiveness across the Pareto front.

3.3. Binary Grey Wolf Optimizer for Multi-Objective Problems (BMOGWO)

The MOGWO was initially developed for continuous optimization tasks and thus cannot directly address the mapping challenges on MPSoCs architectures. To adapt MOGWO for such discrete multi-objective mapping tasks, a binary version was developed by introducing a sigmoid-based activation function to transform continuous position vectors into binary values.

In the original MOGWO, candidate solutions move continuously within the real-valued search space. However, to facilitate binary movement, the continuous position update equation must be adapted. This modified position update equation for binary space is defined as:

$$x_d^{t+1} = \begin{cases} 1 & \text{if sigmoid}\left(\frac{x_1+x_2+x_3}{3}\right) \geq \text{rand} \\ 0 & \text{otherwise} \end{cases} \tag{22}$$

Here, x_d^{t+1} represents the binary position in dimension d at iteration t, while *rand* is a uniformly distributed random value between [0,1], and the *sigmoid* function is defined as:

$$\text{sigmoid}(a) = \frac{1}{1 + e^{-10(a-0.5)}} \tag{23}$$

The intermediate variables x_1, x_2, and x_3, originally defined in equations (16), (17), and (18), are transformed to binary space as follows:

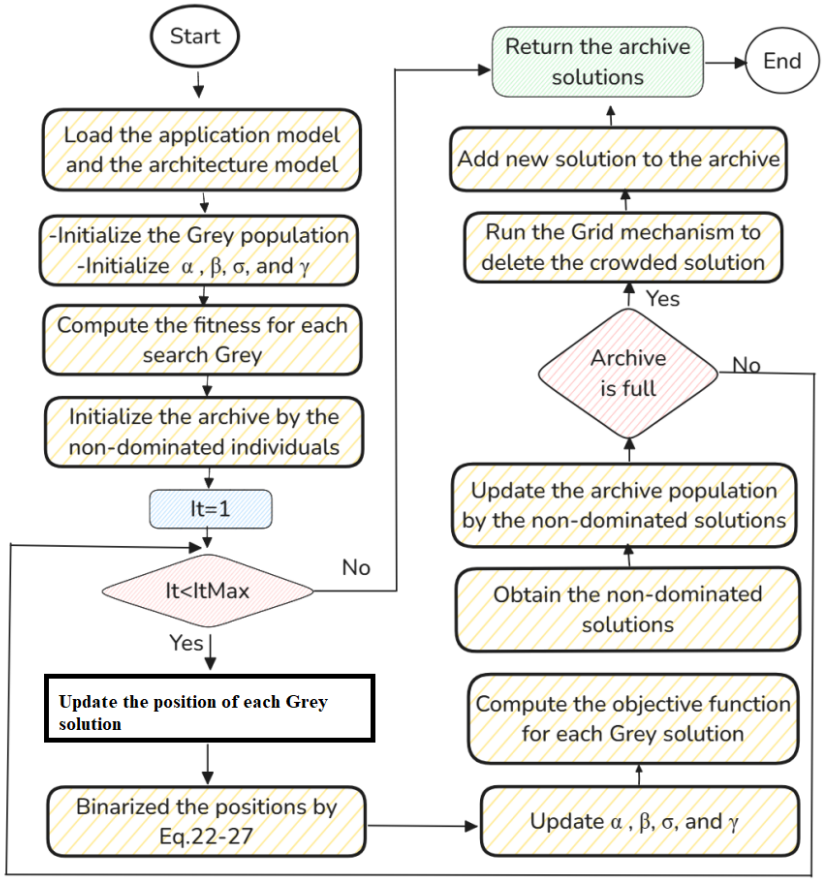

Figure 6: Proposed solution model for task mapping onto MPSoCs in embedded systems.

$$x_1^d = \begin{cases} 1 & \text{if } \left(x_\alpha^d + bstep_\alpha^d\right) \geq 1 \\ 0 & \text{otherwise} \end{cases} \tag{24}$$

$$x_2^d = \begin{cases} 1 & \text{if } \left(x_\beta^d + bstep_\beta^d\right) \geq 1 \\ 0 & \text{otherwise} \end{cases} \tag{25}$$

$$x_3^d = \begin{cases} 1 & \text{if } \left(x_\delta^d + bstep_\delta^d\right) \geq 1 \\ 0 & \text{otherwise} \end{cases} \tag{26}$$

where $bstep_{\alpha,\beta,\delta}^d$ is determined by:

$$bstep_{\alpha,\beta,\delta}^d = \begin{cases} 1 & \text{if } cstep_{\alpha,\beta,\delta}^d \geq \text{rand} \\ 0 & \text{otherwise} \end{cases} \tag{27}$$

Here, *rand* is a random number drawn from a uniform distribution $[0, 1]$, and $cstep_{\alpha,\beta,\delta}^d$ represents the continuous step size for dimension d, which is computed using a sigmoid transformation as follows:

$$cstep_{\alpha,\beta,\delta}^d = \frac{1}{1 + e^{-10\left(A_1^d D_{\alpha,\beta,\delta}^d - 0.5\right)}} \tag{28}$$

Figure 6 provides an overview of the proposed BMOGWO methodology.

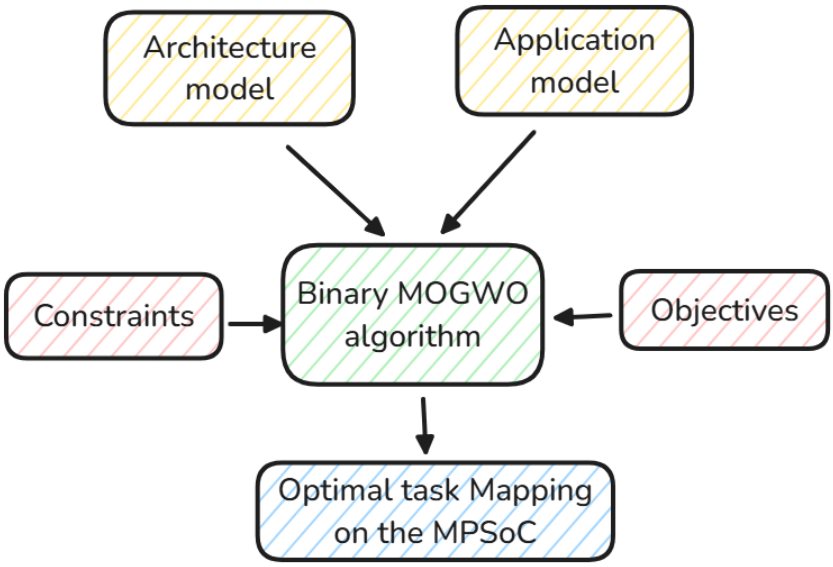

Figure 7: Global description of our solution.

3.4. Description of Our Approach

In our design flow, the placement and scheduling phase is crucial as it directly influences the application's implementation on a specialized architecture. This phase takes the following inputs:

- **Application model**: A detailed representation of the application, outlining its structure and dependencies.
- **Target architecture model**: A model of the hardware architecture, defining the available resources and their interconnections.
- **Performance and energy constraints**: Specific requirements that the implementation must meet, including limits on execution time and energy consumption.
- **Objective functions**: Metrics to be optimized, such as minimizing latency, energy usage.

The output of this phase is a mapped assignment of tasks and communications to physical resources, with an optimized scheduling of tasks across these resources to meet the specified performance and energy constraints. The figure 7 present the global description of our approach.

4. Experimentation and Results

Our approach was implemented using the JAVA programming language, and all experiments were conducted on a system with an Intel(R) Core(TM) i5-7300HQ CPU running Windows 10. Following the execution of our binary MOGWO-based mapping solution, configured with the properties and parameters detailed in Table 1, we obtained the following results:

4.1. Comparison of BMOWGO, MOPSO, and NSGA-II

In validating our proposed solution for mapping applications on MPSoCs, we implemented the Binary Multi-Objective Grey Wolf Optimizer (BMOGWO) and compared it with two other widely used multi-objective optimization techniques: Multi-Objective Particle Swarm Optimization (MOPSO) and Non-dominated Sorting Genetic Algorithm II (NSGA-II). We conducted the comparison on a set of examples of average size, evaluating execution time and energy consumption across varying numbers of processors and task quantities.

Table 1
The basic paprameters of BMOGWO

Algorithm	Basic parameters	
Binary MOGWO	- Number of tasks	21
	- Number of processors	8
	- Architecture type	Star topology
	- Latency	1 unit
	- Population size	20
	- Archive size	20
	- number of Grid	3
	- α	0.1
	- β	4
	- γ	2
	- Number of iterations	20

The table 2 summarizes the performance of the three methods, highlighting BMOGWO's effectiveness in optimizing both execution time and energy consumption. The study offers insight into each algorithm's performance under different MPSoC configurations, demonstrating the adaptability and efficiency of BMOGWO for discrete mapping challenges in MPSoC environments.

Table 2
Comparison of BMOGWO, MOPSO, and NSGA-II on MPSoC Mapping

Processors	Tasks	BMOGWO		MOPSO		NSGA-II	
		Exec Time (s)	Energy (J)	Exec Time (s)	Energy (J)	Exec Time (s)	Energy (J)
4	10	0.9	5.2	1.2	5.6	1.1	5.4
6	15	1.5	6.8	1.7	7.1	1.6	7.0
8	20	2.1	8.3	2.5	8.9	2.4	8.7
10	25	2.8	10.2	3.3	10.8	3.0	10.6
12	30	3.5	12.1	4.0	12.6	3.8	12.4
14	35	4.2	14.5	4.8	15.0	4.5	14.8
16	40	5.0	16.3	5.7	17.0	5.5	16.8
18	45	5.7	18.0	6.5	18.7	6.2	18.5
20	50	6.5	19.9	7.3	20.5	7.0	20.3

The results illustrate that BMOGWO consistently delivers lower execution times and energy consumption compared to MOPSO and NSGA-II, particularly in configurations with higher task loads and processor counts. This demonstrates BMOGWO's potential as a highly effective solution for application mapping in MPSoC environments, particularly where discrete and energy-efficient mapping is critical.

5. Conclusion

In this paper, we introduced a novel approach leveraging the multi-objective variant of the Grey Wolf Optimizer (GWO) to tackle the challenging problem of mapping hierarchical real-time applications onto a hierarchical MPSoC architecture. Our approach was further refined by adapting GWO with a binary encoding scheme, enabling effective optimization of both execution time and energy consumption—critical factors in real-time embedded systems.

The results obtained from our experimental analysis were benchmarked against two well-established metaheuristic algorithms, demonstrating that our proposed solution consistently surpassed these alternatives in both execution time and energy efficiency. These promising findings underscore the efficacy of our approach for optimizing task mapping in MPSoC environments.

With additional experiments and simulations, we are confident that our method will continue to prove effective in addressing similar multi-objective optimization challenges within real-time and embedded systems, contributing valuable insights to this field.

Declaration on Generative AI

The authors have not employed any Generative AI tools.

References

[1] T. Noergaard, Embedded systems architecture: a comprehensive guide for engineers and programmers, Newnes, 2012.

[2] R. Zurawski, Embedded Systems Handbook: Embedded systems design and verification, CRC press, 2018.

[3] P. Noll, Mpeg digital audio coding, IEEE signal processing magazine 14 (1997) 59–81.

[4] J. Bialkowski, M. Barkowsky, A. Kaup, Overview of low-complexity video transcoding from h. 263 to h. 264, in: 2006 IEEE International Conference on Multimedia and Expo, IEEE, 2006, pp. 49–52.

[5] F. Boumaaza, A. E. H. Benyamina, Mapping multi objectifs d 'application intensive sur architecture mpsoc (2012).

[6] A. E. H. Benyamina, P. Boulet, Multi-objective mapping for noc architectures., J. Digit. Inf. Manag. 5 (2007) 378–384.

[7] K. Laredj, M. Belarbi, A. E. Benyamina, Metrics for real-time solutions design, in: Intelligent Computing: Proceedings of the 2018 Computing Conference, Volume 2, Springer, 2019, pp. 411–425.

[8] A. Aroui, P. Boulet, K. Benhaoua, A. K. Singh, et al., Novel metric for load balance and congestion reducing in network on-chip, Scalable Computing: Practice and Experience 21 (2020) 309–321.

[9] W. Wolf, A. A. Jerraya, G. Martin, Multiprocessor system-on-chip (mpsoc) technology, IEEE transactions on computer-aided design of integrated circuits and systems 27 (2008) 1701–1713.

[10] A. Mehran, S. Saeidi, A. Khademzadeh, A. Afzali-Kusha, Spiral: A heuristic mapping algorithm for network on chip, IEICE Electronics Express 4 (2007) 478–484.

[11] S. Mirjalili, S. M. Mirjalili, A. Lewis, Grey wolf optimizer, Advances in engineering software 69 (2014) 46–61.

[12] S. Mirjalili, S. Saremi, S. M. Mirjalili, L. d. S. Coelho, Multi-objective grey wolf optimizer: a novel algorithm for multi-criterion optimization, Expert systems with applications 47 (2016) 106–119.

[13] F. Boumaza, A. E. H. Benyamina, D. Zouache, L. Abualigah, A. Alsayat, 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 (2023).

[14] Q. Al-Tashi, S. J. Abdulkadir, H. M. Rais, S. Mirjalili, H. Alhussian, M. G. Ragab, A. Alqushaibi, Binary multi-objective grey wolf optimizer for feature selection in classification, IEEE Access 8 (2020) 106247–106263.