SSRG International Journal of Electrical and Electronics Engineering
ISSN: 2348-8379/ https://doi.org/10.14445/23488379/IJEEE-V11I11P128

Volume 11 Issue 11, 295-305, November 2024

Original Article

Application Task Mapping in Real Time Network on Chip Systems for Latency Optimization Using Bio Inspired Greedy Firefly Algorithm

Shweta Ashtekar[1], Kushal Tuckley[2]

[1]*Department of Electronics Engineering, Ramrao Adik Institute of Technology, Dr. D.Y. Patil Deemed to be University, Maharashtra, India.*
[2]*Department of Electrical Engineering, Indian Institute of Technology, Maharashtra, India.*

[1]*Corresponding Author : shweta.ashtekar@rait.ac.in*

Received: 15 September 2024 Revised: 16 October 2024 Accepted: 14 November 2024 Published: 30 November 2024

Abstract *- Network on Chip (NoC) provides a communication framework within multiple cores in a heterogeneous computing ecosystem. While the execution of real time embedded applications like multimedia, data networks, and signal processing, mapping application tasks in NoC on appropriate cores is the most crucial, affecting overall performance and latency. This research proposes a nature-inspired metaheuristic Greedy Firefly Algorithm (GFF) for NoC, which combines the greedy approach with the firefly algorithm for mapping tasks. It is examined against three existing algorithms: NMAP, BB and Random algorithm using identical embedded traffic scenarios and simulation environment to establish the aptness of the suggested algorithm. The results of the GFF algorithm prove more efficient at higher traffic loads for applications such as PIP, MWD, CAVLC, MMS, VOPD, and E3S Consumer benchmarks and reduces average latency by almost 5 to 20% as well as increased throughput compared to other algorithms and is significant in critical applications. The simulator's generated dataset was subjected to an SVM ML model, which predicts how GFF is appropriate for the mentioned applications while considering minimal latency.*

Keywords *- Network on chip, Application task mapping, Real time embedded applications, Metaheuristic, Nature inspired, Greedy Firefly algorithm (GFF), Latency optimization, SVM.*

1. Introduction

In recent decades, Networks on Chip (NoC) based multicore systems have gained popularity for the execution of real time application specific embedded systems like multimedia, involving video and audio processing. NoC is an emerging technology that provides an efficient and scalable communication infrastructure for integrating various system components on chip (SoC).

SoC provides a complete computing platform that has application specific components like CPUs, DSPs, hardware accelerators (GPUs), and memory on a single chip for real time embedded applications such as Industrial, Robotics, Communication, Autonomous driving, Internet of Things (IoT), Multimedia, aerospace, medical. NoC replaces traditional bus-based architectures of SoC systems with a packet-switched network, enabling high speed transfer amongst different Intellectual Property (IP) cores on a chip. NoC improves performance, reduces power consumption, and enhances scalability and flexibility in SoC designs.

Routers connect the many Processing Elements (PEs) that make up NoC, an interconnect architecture. A PE is a node or core, like a DSP processor, memory controller, CPU, GPU, or Application Specific Integrated Circuit (ASIC), as shown in Figure 1. After the Network Interface (NI) transforms the data produced by PE into packets, communication between the specified source and destination occurs via a network fabric made up of routers and connection links. The data in packets/flits will travel from source to destination within the chip [1].

While implementing many core embedded systems in real time situations, the execution time of a task and the total transmission time needed to transport data between several cores and memory are the two factors that are taken into consideration to determine how long a task takes to process. Numerous challenges or restrictions exist for Real Time Network on Chip (RTNOC), including latency in computation and communication, power consumption, dependability, and Quality of Service (QoS). Several models are used to implement the real-time tasks. The Directed Acyclic Graph

(DAG) model is among the most popular models. Every task is represented by a distinct node in this model [2], with edges joining them. Indicating the data flow that must occur between these nodes. Multiple factors must be considered while implementing real time applications on the NoC platform, such as Routing, Switching, Mapping, and Scheduling [3].

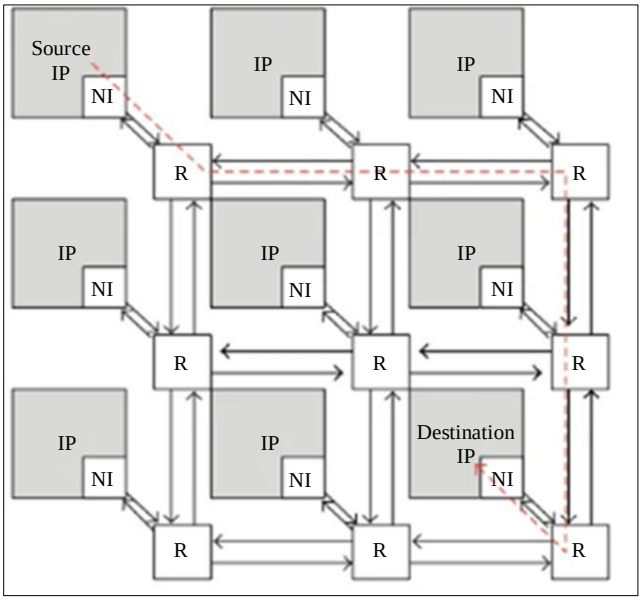

Fig. 1 A typical 3×3 mesh NoC system

Application mapping is one of the crucial issues while mapping application tasks on the platform. Any application comprises multiple tasks and communication among them. The effective mapping [4] of multiple tasks for any application results in improved performance and proper utilization of available resources of any Network on chip system.

The process of allocating or mapping application tasks or cores to certain Processing Elements (PEs) or resources in the NoC system is known as application mapping, as shown in Figure 2. This stage is essential for designing and developing systems based on NoC since it directly affects latency, energy efficiency, and overall performance. Applications are divided into processes or tasks, each of which may call for memory access, communication, or computation. The PEs (such as CPUs, GPUs, or specialized hardware cores) that the NoC connects must have tasks mapped onto them. Numerous tasks necessitate communication and are interrelated. Putting highly interactive tasks adjacent to one another reduces communication overhead during mapping. Various factors are affected while mapping tasks, like Latency, throughput, power and energy efficiency. By strategically placing tasks, communication delays can be minimized. The workloads are distributed evenly throughout all cores for better performance. Also, reducing the power used for network data transfer will maximize energy efficiency. Certain limitations must be considered during mappings, such as hardware limitations of existing NOC, including memory bandwidth, link capacity, and the number of PEs.Another limitation can be application restrictions with dependencies, deadlines, and immediate needs. Other limitations are scalability heterogeneity and runtime variability. In contemporary multi-core and many-core systems, optimizing the potential of NoC architectures requires efficient application mapping.

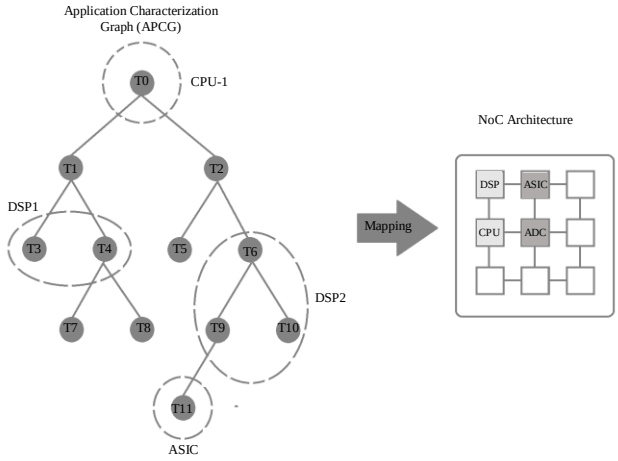

Fig. 2 Application mapping process

The primary motivation behind this research is that even though there are multiple mapping algorithms, a single algorithm will not perform best for different applications like networking and video/audio processing. So, there is a need to develop a generic algorithm suitable for multiple applications. In our research for Latency optimization, a nature inspired metaheuristic Firefly algorithm [5] with a Greedy approach has been proposed for application mapping of NoC systems and successfully evaluated on many real time applications.

The greedy algorithm in the firefly-based application mapping process is to iteratively refine the mapping configuration by making locally optimal decisions based on the attractiveness between fireflies (representing tasks or PEs) and the light intensity (representing the objective function value). By continuously improving the mapping through local search and optimization, the greedy algorithm helps to achieve better performance and efficiency in NoC based embedded systems. The main contributions of this research are,

1. An enhanced metaheuristic task mapping has been implemented using a bio inspired firefly algorithm for NoC systems.
2. The incorporated greedy approach iteratively improves the mapping configuration of the Firefly algorithm by making local searches and optimizations.
3. The experimental analysis based on the greedy firefly algorithm indicates that for some real time embedded applications, there is a notable reduction in average

network latency, especially for higher traffic loads (60 % to 100%), when compared with other existing mapping approaches like Random, Branch and Bound (BB), NMAP.

4. Support Vector Machine (SVM) machine learning model is applied to the dataset generated by the simulator to predict GFF, which proves best for which applications considering Latency minimization.

Our proposed approach is supported in the main body of the paper as follows: Section 2 summarizes the related work of application mapping for the NoC system. The preliminaries and basic concepts required for the work are discussed in Section 3. Section 4 articulates the improved Greedy Firefly algorithm for mapping application tasks. The results obtained after simulating embedded applications are brought forth in Section 5, and at last, the research work has been deduced in Section 6.

2. Literature Review

Section II overviews different static and dynamic application task mapping approaches used for NoC systems and their benefits and shortcomings. Among the various research problems related to NoC systems, application mapping is one of the crucial issues when mapping application tasks on the platform. Any application comprises multiple tasks and communication, which can be effectively represented using an Application task graph. The effective mapping of multiple tasks of any application results in improved performance and proper utilization of available resources of any NoC based system. Improved throughput, reduced power, and latency will be achieved by precisely mapping tasks at the core. Thus, the overall network's performance can be enhanced.

Application mapping techniques are broadly classified [6, 7] as static/nonadaptive/design time mapping or dynamic/adaptive/run time mapping. Also, hybrid mapping comprising both of them can be derived. In the dynamic or adaptive mapping approach, the allotment of tasks and their reordering has been carried out while executing or running the applications. These adaptive techniques are required in case of variable traffic loads, faults, congestion, or thermal variations. Here, the mapping depends on the current traffic conditions by distributing the tasks on free cores; thus, there is the reliability of packets/flits reaching the destination, but this approach may result in additional overhead while switching between multiple tasks [6]. This may result in further delay in execution and an increase in energy requirements as an algorithm needs to be re-run to obtain better results.

Considering the above contributing factors, Dynamic mapping [6] can be further categorized, and a few of them are discussed as,

a) Reliability aware mapping: Due to various factors such as manufacturing defects, crosstalks, transient faults, and core failures, the overall reliability of the NoC system gets affected. There is a need for reliability dynamic mapping techniques, as discussed in the paper [8], to handle single or multiple core failures by implementing a Kuhn Munkres efficient algorithm that has a manager tile that remaps the tasks to nearby cores considering the minimization of cost and overhead in case of core failures.
b) Congestion aware mapping: The overall performance of the NoC system gets degraded by any congestion or contention in the network, which may occur because of simultaneous communication inside the system and multiple packets getting exchanged across the number of cores. As the paper [9] mentioned, a run time mapping algorithm maps the highest communicating tasks on the same core to avoid congestion and reduce overhead.
c) Thermal aware mapping: Because of advancements in IC technology, it is possible to accommodate more cores in close space on the chip, due to which the chips are getting exposed to thermal hazards, and performance gets affected in terms of increased latency and decreased throughput. Paper [10] used a neural network-based thermal management system by migrating tasks between neighboring cores, but it requires additional training.
d) Energy aware mapping: Reducing energy consumption & overhead while the task communication occurs is one of the major goals of dynamic mapping. The paper [11]uses an approach to allocate directly communicating tasks onto the same core to minimize energy consumption and overhead. However, an increased number of tasks results in deadlines, causing delays in task completion.

Dynamic mapping works well in the scenarios mentioned above; however, this research paper mainly focuses on static mapping techniques.

In static or nonadaptive mapping, the allotment of tasks for an application is decided during design time only when considering the initial underlying network infrastructure or available resources. Deciding and developing a good solution by efficient task assignments in the offline phase is complex as it requires the best utilisation of available cores [7]. These techniques mostly run only once, resulting in less overhead and comparatively reduced latency and energy consumption, but are less suitable in unpredictable situations. Static mapping techniques are further classified as search based mapping and exact mapping.

The Exact mapping involves vast mathematical calculations with programming to arrive at the optimal solution, so it is preferable for NoC architectures with fewer tasks. Paper [12] analyses the network contention and, to minimize the intertile network contention, has suggested ILP formulation. Integer Linear Programming (ILP) is an exact

application mapping proposed onto NoC systems to acquire either optimal or closest to optimal solutions within the bounds of computational time. A Mixed Integer Linear Programming (MTLP) model has been implemented in [13] paper, using hardware/software codesign for automatic mapping on application-specific Integrating systems that perform image and signal processing within given deadlines. Search-based mapping is divided into a) Systematic and Deterministic search and, b) Heuristic search.

One algorithm example of a Systematic search approach is Branch & Bound (BB), which systematically explores the entire search space of possible solutions by bounding unallowable solutions and finally provides an optimal solution. The paper [14] presented a hybrid BEMAP algorithm that combines a BB map with an exact systematic search to get a multi-objective solution in terms of throughput and cost.

Generally, Heuristics or metaheuristics algorithms are either Transformative, like Genetic Algorithm (GA) [15], Particle Swam Optimization (PCO) [16], or Constructive, like Near optimal mapping (NMAP), which give better solutions because they explore various possible solutions to arrive at optimal solution compared to mathematical approximations and are population or swarm intelligence based. They are usually self-learning and nature inspired by emulating the natural intelligence of troops of animals and birds. Numerous heuristic/metaheuristic algorithms exist, such as the Whale Optimization Algorithm [17], which uses three operators to mimic humpback whales' bubble-net foraging behaviour, encircling prey, and prey seeking. This study was applied to 29 distinct mathematical benchmarks to analyse the algorithm's convergence. The paper proposes a metaheuristic Cuckoo search via Levy flight to maximize task placement on the Network on Chip cores [18]. A greedy approach is used for preprocessing, and after being implemented on several embedded systems, the method yields better cost, latency, power consumption, and throughput results than existing techniques.

In Paper [18], a metaheuristic Cuckoo search via Levy flight is proposed to optimize the placement of tasks on the Network on chip cores. For preprocessing, a greedy algorithm is utilized, and the overall algorithm provides improved results in terms of cost, latency, power consumption, and throughput compared with existing algorithms after implementing different embedded applications. An enhanced shuffled frog leaping algorithm is formulated in the paper [19], in which initial mapping is assorted. Because of mapping improvement of each category search space, mapping is more powerful. Therefore, the nodes connected with higher costs are mapped in adjacent locations on the NoC platform to lower communication costs. Ants naturally follow a single path in searching for food, which inspires the Ant Colony Optimization (ACO) technique [20], a population-based probabilistic method.

Based on these facts for latency optimization, NoC finds minimal and nonminimal routes within a region during run time without any software overhead.

The hybrid mapping approach combines the benefits of both static and dynamic mapping methods. Executing as per static method in normal situations and switching to a dynamic approach in case of unavoidable situations such as congestion. A better performance can be achieved by integrating machine learning algorithms into the hybrid mapping techniques.

This study proposes a greedy firefly algorithm inspired by nature and compares its performance to current classical methods like NMAP. Random and BB. The application environments considered for comparison are Picture In Picture (PIP), Multi Window Display (MWD), Multi Media Systems (MMS), and E3S Consumer Benchmark. The logical foundations of existing approaches are presented in the next section.

3. Preliminaries

The following are basic definitions and evaluation parameters that must be considered with NoC systems.

3.1. Definitions

Definition 1 (Directed Application task or communication Graph (DAG)) : As every application comprises multiple required tasks and communication between the tasks, an application having a certain number of n tasks as (t1,t2,t3,….tn) can be represented using the Directed Application task graph as DAG(T, C) where T represents the number of vertices each representing single task whereas C represents weighted edge(Ci,j) or link between any two tasks i and j [21]. The weight is the communication volume or bandwidth between a set of tasks. For example, the application Multi Window Display (MWD) has T=12 and C=13.

Definition 2 (Directed Core or architecture or topology Graph (DCG)) : The NoC architecture comprises a certain amount of processing cores on which the application tasks must be efficiently mapped. The topological architecture of NoC is represented using a directed core graph as DCG(P, E), where P signifies a set of processing cores(p1,p2,p3,….pn). In contrast, E symbolizes a set of edges (Ei,j) communicating between these cores or tiles.

3.2. Evaluation Parameters

The performance or evaluation indicators represented using mathematical models in the paper [22] need to be analysed after implementing the proposed Greedy firefly application mapping on NoC architecture using the NoCTweak Simulator [22].

Communication cost: The Communication cost for NoC based system can be derived as,

$$\text{Cost} = \sum_{i,j} [\, Bti,tj \times Ni,j] \quad (1)$$

Where Bti,tj denotes communication bandwidth between any two tasks i & j. Also, Ni,j represents the Manhattan Distance between a pair of nodes represented with coordinates as (xi,xj) as well as (yi,yj) given as

$$Ni,j = |xi - xj| + |yi - yj| \quad (2)$$

Average network Latency: The average network latency or packet delay considering only packets received after warm-up time is given by the following equation below where N represents available cores in that architecture, Lti,j is the Latency, for instance, packet j after Ni packets received by core i.

$$Lt_{av} = \frac{1}{N} \sum_{i=1}^{N} \frac{1}{N_i} \sum_{j=1}^{N_i} \frac{1}{N_i} Lt_{i,j} \quad (3)$$

Network Throughput: Network throughput is the speed with which the network efficiently accepts and delivers total inserted packets during warmup. The following equation gives the average network throughput

$$T_{avg} = \frac{1}{N(T_{sim} - T_{warm})} \sum_{i=1..N} N_i \quad (4)$$

Where, T_{sim} is simulation time, and t_{warm} is warm-up time.

4. Mapping Optimization Using the Greedy Firefly Algorithm

4.1. Optimization Problem

The optimization problem is widely used in various fields, such as engineering, mathematics, computer science, and autonomous robots. Optimization mentions how to arrive at a practical solution when certain complexities, constraints or set goals exist for a specific problem.

The optimization problem has primary considerations, such as the objective function that needs to be optimized, the set of constraints, a few sets of possible solutions, and the rule for optimization, whether to minimize or maximize the values to achieve the best possible solution. Optimization improves performance or overall efficiency related to the problem.

Using the systematic search based optimization methods, time convergence occurs as these algorithms need to search the entire search space, and concurrence arises as the search may not get completed within the assigned polynomial time bounds.

Considering the abovementioned limitations, there is a need for either Heuristic or meta-heuristic-based optimization algorithms. Evolutionary Meta Heuristic algorithms are powerful in providing general-purpose solutions or structures instead of application-specific ones in case of a complex problem with a huge search space or not well defined objective function. A few characteristics of meta Heuristic mapping algorithms are that they are efficient while solving complicated optimized needs or defining some mathematical models, as they require fewer parameters to be altered during implementation.

Powerful nature or bio inspired metaheuristic approaches are gaining popularity while solving most optimization issues. Some metaheuristics algorithms have swum- or population-based intelligence algorithms based on updating individuals' positions by mimicking animal/bird behaviour, such as finding prey, hunting, and protecting themselves while finding an efficient optimization solution.

4.2. Firefly Behavior

One efficient bio inspired metaheuristic algorithm is derived from the natural behavior of fireflies. Fireflies, known as lighting bugs, are usually found before monsoon in tropical regions. Using the process of Bioluminescence, they produce unique flashing lights that are used as a signaling system with which they communicate for various reasons, such as attracting other partners, sending alert messages, or attracting prey.

As proposed in the paper [23] for formulating a firefly algorithm, some assumptions are made as a)all fireflies get attracted towards each other without consideration of their sex, b) The flashing light can be related as an Objective function that needs to be optimized. c)The brightness of the firefly can be derived considering the view of objective function when it is required to articulate an optimized solution or algorithm d) The attractivity that can be established between different fireflies is usually related to their Brightness. Depending on this fact, the firefly with low brightness will always approach the firefly with more brightness. As the distance between fireflies increases, the brightness and, in turn, attractiveness decreases.

The basic firefly algorithm can be devised as follows [24],

$$I(r) = \frac{I_0}{1 + \lambda r^{2\cdot}} \quad (5)$$

Let Light Intensity I change according to distance r as,

Where λ is the Light Absorption Coefficient, and I0 is the light intensity of the source.

In the same way, Attractiveness is derived as

$$A(r) = \frac{A}{1 + \lambda r^{2'}} \quad (6)$$

Where, A0 is attractiveness at r=0

Any firefly i approaches any other firefly j, and its movement is given by Equation (7) as,

$$x_i = x_i + \beta_0 e^{-\gamma r_{ij}^2}(x_j - x_i) + \alpha(rand - 0.5) \quad (7)$$

The 1st term is the attractiveness factor by which xi gets attracted to another brighter firefly xj, and the 2nd term is randomization, which helps firefly xi to move randomly in case of no brighter firefly.

4.3. Proposed Greedy Firefly Algorithm (GFF)

The following are a few considerations when implementing the Greedy Firefly algorithm. As per the above terms related to firefly behavior, here

- Every firefly represents a single solution. So, a number of solutions are available with multiple fireflies.
- Each firefly differs from another based on brightness.
- The brightest firefly is the current global better solution.
- The brightness varies with the objective function when calculating communication costs, as given by Equation (1).
- The algorithm is iterative based in which each iteration gives one solution, compared with the next iteration solution. If the next iteration is comparatively brighter, movement towards brighter fireflies occurs, and the solution is swapped.
- The iterative process continues until the optimal solution is achieved or the termination criteria are reached.

At first, the algorithm starts with random positioning of all tasks on available cores of the NoC system and then continues through an iterative process. The greedy algorithm in the firefly-based application mapping process iteratively refines the mapping configuration by making locally optimal decisions based on the attractiveness of the fireflies and the light intensity. The greedy algorithm selects the solution in the current phase of iteration and finds the best local solution by exploring its search space, enhancing the probability of getting the final best or optimal global solution iteratively.

The basic flow of the algorithm is as follows,

1. Evaluate the total cores onto which application tasks must be mapped.
2. Generate initial solution population
3. Excecute firefly algorithm
4. To improvise the obtained solution, run the greedy algorithm
5. Terminate when the optimized results are achieved.

The following Pseudo code summarizes the program's flow for the Greedy Firefly Algorithm.

Pseudo code for Greedy Firefly algorithm (GFF):

```
Algorithm: Greedy Firefly Algorithm(GFF)
Input: DAG(T,C) ,DCG(P,E)
Output: Optimized mapping /Firefly solution
Begin
Initialise parameters, Set max ← max Iteration
Generate initial solution F0 population
While (iteration <max) Do
    Calculate Brightness(Communication cost)to find
    a new Firefly F1
    Estimate distance & attractiveness between
    fireflies
    If F1>F0 Then
        Execute a Greedy approach for local search
        optimisation
        Swap the solution by moving F0 towards F1
    Else
        F1=F0
    End If
    ++Max
    Return Optimized Firefly
End While
    Calculate Average Latency Ltav & Calculate
    Throughput Tavg
End
```

5. Evaluation

The Evaluation section elaborates on implementing the proposed Greedy Firefly algorithm (GFF) to verify its efficacy when compared with other mapping algorithms for various real time embedded traffic.

The system configurations utilized to implement the algorithm with Intel core I7 Quad-core Processor with operating frequency 3.2 GHz and 8 GB RAM. The evaluation indicators are Average Latency, Throughput and Power estimated on a 65nm cell library model in NoCTweak Simulator [22].

NoCTweak is a popular cyclic accurate system C/C++ simulator that incorporates multiple random and embedded traffics to perform various algorithms of NoC-based systems precisely. Taking advantage of this highly parameterizable NoCTweak Simulator, the proposed Greedy Firefly algorithm has been implemented to analyze and check its aptness against existing mapping algorithms under identical experimental environments and traffic conditions.

5.1. Simulation Environment / Experimental Setup & Embedded Applications

NoCTweak simulator offers distinct parameter options for setting synthetic and embedded traffic situations in various routing techniques to realize the proposed GFF mapping algorithm on embedded applications. The following system configurations were availed, as indicated in Table 1.

Table 1. Platform specifications for simulation environment

Platform Parameters	Description
Network Topology :	2D Mesh
Type of Platform :	Real Time Embedded
Distribution of Packet :	As Exponential
Length of Packet :	Fixed 10 Flits
Offered Load or Traffic (Flit Injection Rate)	0.1 to 1.0 (flits/cycle/node)
Router Type:	Pipelined, Wormhole Switching
Selected Routing Algorithm :	Xy Ordered
Output Channel Selection:	Round Robin
Embedded Task Mapping Algorithms:	Random, NMAP, BB, GFF
Size of Buffer :	8 (Flits)
Length between Routers :	1000 (um)
Type of Pipeline :	5
Clock Frequency (Input):	1000 MHz
Clock Frequency (Operating) :	1000 MHz
Time Taken for Warmup :	20000 Cycles

Here, six representative real-time embedded applications and benchmarks are selected from multimedia and communication scenarios, such as PIP, MWD, CAVLC, MMS, VOPD, and E3S Consumer benchmarks, as mentioned in Table 2. Those applications outperformed the proposed GFF algorithm and comparatively showed significant latency reduction, especially at higher traffic loads, between 60 % and 100%.

Table 2. Embedded benchmarks selected for implementation

Applications	Description	No. of Tasks	No. of Edges
MWD	Multi Window Display	12	13
PIP	Picture In Picture	8	8
CAVLC	Context-Adaptive Variable-Length Coder	16	23
MMS	Multi Media System	25	33
VOPD	Video Object Planar Decoder	16	21
E3S Consumer Benchmark	Consumer Application	12	12

The suggested results of this Metaheuristic GFF algorithm are contrasted against the NMAP algorithm, a constructive heuristic search approach, the BB algorithm, a Systematic Search approach, the SA Self adaptive algorithm and the Random algorithm. The Random and NMAP are already part of the simulator. The BB algorithm has been implemented additionally for comparison. The basic workings of BB and NMAP algorithms have been explained in the literature survey. The proposed Greedy Firefly algorithm has been successfully implemented in the simulator.

5.1.1. Task Mapping on Cores Using GFF Algorithm on MWD Application

Figure 3 (a) shows the Directed Application Task or communication Graph (DAG) for the Multi Window Display (MWD) application [25], where 12 nodes are communicating with each other using 13 edges. Each directed edge has communication volume bandwidth written on it.

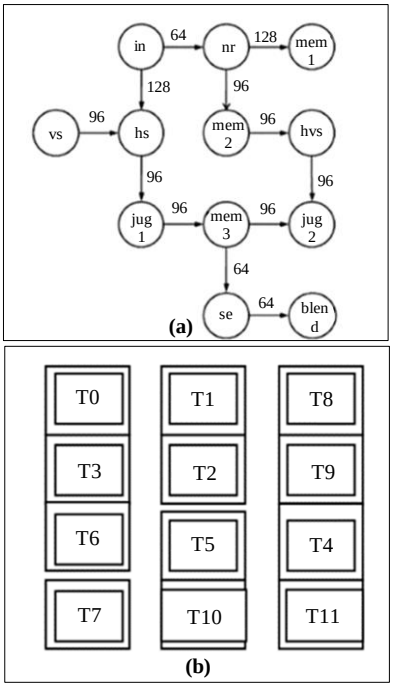

Fig. 3 (a) Directed Application task or communication Graph (DAG), and b) Core mapping on 4 × 3 mesh using GFF algorithm for Multi Window Display (MWD) application.

5.2. Latency and Throughput

Latency is important for analysing the algorithm's workings and ensuring improved performance. The average Latency of an application is the delay produced by all tasks when total packets travel from source to destination.

5.2.1. Application : Multi Window Display (MWD)

The detailed results of the simulator of average latency value comparison for the MWD application are indicated in Table 3, and the graph is shown in Figure 4. The observation shows that for lower traffic loads, almost all algorithms perform equally; however, the proposed algorithm GFF gives better latency reduction values for higher traffic loads above 0.7. This is due to the Greedy approach integrated with the Firefly algorithm, which explores more search space in local searches and gives optimal global solutions. The percentage

improvement in latency reduction for the proposed GFF algorithm is 16.3 % against Random mapping, 1.7 % against NMAP and 14 % against the BB mapping algorithm.

Table 3. Average latency comparisons

Average Latency (ns)					
Fir	**Random**	**NMAP**	**SA**	**GFF**	**BB**
0.1	12.242	8.636	8.014	10.163	11.41
0.2	12.556	8.798	8.299	10.333	11.704
0.3	13.385	9.054	8.703	10.617	12.211
0.4	19.618	9.507	9.492	11.067	13.655
0.5	2741	10.566	11.354	12.069	724.755
0.6	4461.672	17.876	27.01	18.149	2652.96
0.7	7277.772	2032.577	4877.33	2079.017	6012.064
0.8	9397.886	4482.09	10296.01	4337.568	9130.324
0.9	10515.27	6325.818	15451.31	6275.664	11980.25
1	12160.58	7646.411	19524.92	7611.291	14552.27

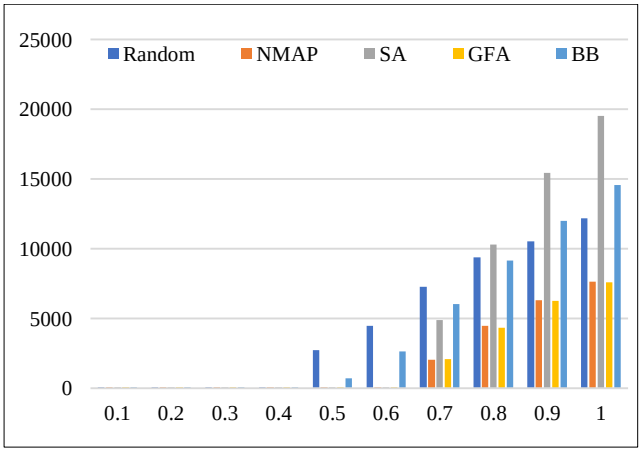

Fig. 4 Graph of average latency vs traffic load (Fir)

In the same way, for all other remaining applications, the Average latency for higher loads, e.g. 0.8, shown in Table 4

Table 4. Comparative analysis for average network latency

Embedded Application	Average Latency (ns) at Fir= 0.8 Application Mapping Algorithm			
	Random	NMAP	BB	GFF
PIP	5752	4425	4329	3614
CAVLC	3257	13	3733	12
MMS	4876	4201	4973	4368
VOPD	6071	2671	2608	2587
E3S Consumer	5631	6854	5333	5331

5.2.2. Throughput

The maximum traffic accepted and delivered by the network in a unit of time is given as Throughput [18]. The overall performance is improved by efficient mapping of tasks on available cores. A detailed analysis of the MWD application is shown. The proposed GFF algorithm provides similar results to NMAP and outperforms other mapping algorithms, as shown in Table 5 and the graph in Figure 5.

Table 5. Throughput (cycles/pkt) comparison for MWD application

Fir	Random	NMAP	SA	GFF	BB
0.1	0.074	0.074	0.04	0.074	0.074
0.2	0.151	0.151	0.081	0.151	0.151
0.3	0.229	0.229	0.124	0.229	0.229
0.4	0.307	0.307	0.167	0.307	0.307
0.5	0.366	0.387	0.21	0.387	0.382
0.6	0.421	0.466	0.253	0.466	0.438
0.7	0.46	0.524	0.271	0.523	0.475
0.8	0.493	0.56	0.278	0.57	0.504
0.9	0.527	0.61	0.278	0.628	0.524
1	0.554	0.649	0.278	0.649	0.539

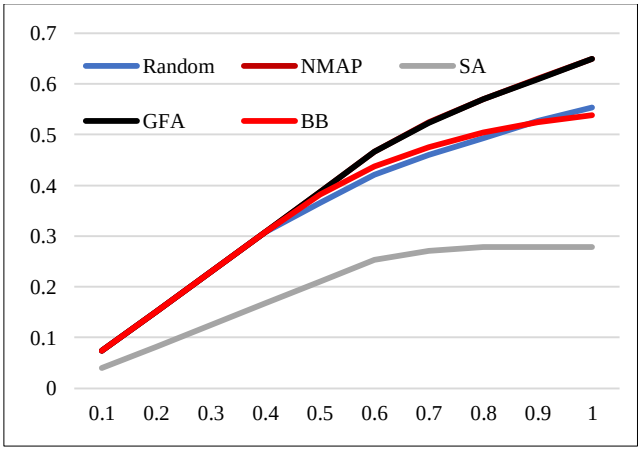

Fig. 5 Graph of throughput vs traffic load (Fir)

In the same way, the throughput comparison is shown for the remaining applications, as indicated in Table 6.

Table 6. Comparative analysis for average throughput

Embedded Application	Throughput (cycles/packet) at Fir= 0.8 Application Mapping Algorithm			
	Random	NMAP	BB	GFF
PIP	0.204	0.218	0.218	0,223
CAVLC	0.229	0.247	0.221	0.247
MMS	0.089	0.092	0.088	0.175
VOPD	0.347	0.37	0.37	0.37
E3S Consumer	0.379	0.377	0.388	0.388

5.3. Support Vector Machine (SVM) Machine Learning Model Implementation

The dataset was generated by running the different application mapping algorithms, Random, NMAP, BB, and

GFF, on 12 different embedded applications. Four parameters, Latency, Throughput, Energy and power, are considered while running on a simulator. To reverify and validate the results of the proposed GFF algorithm, the SVM machine learning model has been implemented on the generated data set from the NoCTweak simulator. SVM ML model will predict how GFF is suitable considering minimal latency. Since the dataset is small to medium, SVM is preferred over other ML algorithms, such as random forest or decision tree. The test size for training and testing is selected as 0.8. By implementing this model, the metric Mean Squared Error (MSE) is calculated for each specific application, giving an average squared difference between the predicted values from the model and actual target values from the generated dataset. Also, the mean value for latency was examined for each application, as shown in Table 7. For the mentioned applications, such as PIP, CAVLC, MMS, VOPD, MWD, and E3S Consumer benchmark from 12 different applications, the Mean values for Latency are minimal compared with other mapping algorithms. The proposed Greedy FireFly (GFF) algorithm has been predicted as the best mapping algorithm considering minimal Latency for the mentioned applications.

Table 7. MSE and Mean value for latency using the SVM model

Embedded Application	MSE Using SVM	Mean value Latency in ns			
		Random	NMAP	BB	GFF
PIP	0.003739	2713	2138	1975	1695
CAVLC	0.0021114	1637	283	1874	277
MMS	0.0016874	2586	1868	2445	2432
VOPD	0.00739234	2845	1202	1170	1163
E3S Consumer	0.0058282	3411	4048	2848	2824
MWD	0.00634626	4661	2055	4510	2047

5.4. Result Analysis

Some of the mentioned parameters impacting Latency and throughput must be considered while executing these algorithms. To obtain the network latency, throughput, and power consumption data for comparison, w the Flit Injection Rate (fir) or the load has been altered that is applied to the system for each run. The rate at which packets or flits are injected into the router is indicated by Fir. The latency impacts the network's total performance, which is sensitive to fir. Saturation eventually sets in, and the average delay is impacted by congestion. The buffer depth specifies the number of flits/cycles. Network performance is enhanced by increasing buffer depth.

Also, based on the packet size, the network saturates at different loads. In general, larger packet length results in lower average latency and high saturation load. Another factor to consider while mapping is the bandwidth requirement between the number of tasks in that application. The observations indicated in the result part show that the proposed Greedy Firefly algorithm is most suited for multimedia applications. So, the results for high-end signal/video processing applications like VOPD, PiP, MMS, CAVLC and MWD applications indicate that GFF algorithms outperform existing algorithms.

The various tasks involved in these applications are audio/video compression/decompression, encoding/decoding, quantization, filtering, etc. Also, it requires the storage of large volumes of converted data in the memory. In these scenarios, GFF performs a local search using a greedy approach that provides a locally optimal solution by mapping the tasks of the application on nearby cores and then using the firefly algorithm, and a global search is done in order to get optimal firefly by comparing it with an earlier solution in each iteration. This results in reduced latency and increased overall throughput. In contrast, in systematic search-based optimization methods like the BB algorithm, time convergence occurs as these algorithms need to search the entire search space, and concurrence arises as the search may get delayed and latency increases.

6. Conclusion and Future Scope

Motivated by high performance implementation of on chip multicore interconnection NoC systems and satisfying today's rising demands of real time Multimedia & Networking applications, through this paper, a metaheuristic nature inspired Greedy Firefly algorithm was proposed and evaluated using the NoCTweak simulator as a small contribution to this field.

Different static mapping techniques such as Random, NMAP, BB, and proposed Greedy Firefly are analyzed and implemented using various real-time embedded benchmarks. After evaluation, the results of the proposed GFF mapping proved more efficient for PIP, MWD, CAVLC, MMS, VOPD, and E3S Consumer benchmarks. There is a significant latency reduction from 5 to 20 %. It is 16.3 % compared with Random, 1.7 % with NMAP, and 14% with BB algorithms for MWD application, an example of significantly higher traffic loads (60 % to 100%). There are certain limitations of GFF for large scale or complicated tasks, such as the computing complexity of the GFF algorithm may become unfeasible as the number of fireflies increases or the algorithm may converge too soon.

An SVM machine learning model has been implemented on the generated dataset from the simulator that predicts the GFF as the best mapping algorithm compared with existing algorithms for the mentioned applications. Continuing this work, we plan to implement a congestion-aware adaptive or hybrid mapping technique [26] incorporating a machine learning algorithm. This approach will help make proper application mapping decisions when combined with static approach.

References

[1] K. Paramasivam, "Network On-Chip and Its Research Challenges," *ICTACT Journal on Microelectronics*, vol. 1, no. 2, pp. 83-87, 2015. [CrossRef] [Google Scholar] [Publisher Link]

[2] Chawki Benchehida et al., "Memory-Processor Co-Scheduling for Real-Time Tasks on Network-On-Chip Manycore Architectures," *International Journal of High Performance Systems Architecture*, vol. 11, no. 1, pp. 1-11, 2022. [CrossRef] [Google Scholar] [Publisher Link]

[3] Djalila Belkebir, and Fateh Boutekkouk, "Two-Steps into Energy Consumption Optimisation Due to the Mapping of Multimedia Application to Network on Chip Architecture," *International Journal of Intelligent Systems Technologies and Applications*, vol. 15, no. 4, pp. 353-378, 2016. [CrossRef] [Google Scholar] [Publisher Link]

[4] Coskun Celik, and Cuneyt F. Bazlamacci, "Effect of Application Mapping on Network-on-Chip Performance," *2012 20th Euromicro International Conference on Parallel, Distributed and Network-Based Processing*, Munich, Germany, pp. 465-472, 2012. [CrossRef] [Google Scholar] [Publisher Link]

[5] Xin-She Yang, *Nature-Inspired Metaheuristic Algorithm*, 2nd ed., Luniver Press, Beckington, UK, 2010. [Google Scholar] [Publisher Link]

[6] Sharoon Saleem et al., "A Survey on Dynamic Application Mapping Approaches for Real-Time Network-On-Chip-Based Platforms," IEEE Acces, vol. 11, pp. 122694-122721, 2023. [CrossRef] [Google Scholar] [Publisher Link]

[7] Pradip Kumar Sahu, and Santanu Chattopadhyay, "A Survey on Application Mapping Strategies for Network-on-Chip Design," *Journal of Systems Architecture*, vol. 59, pp. 60-76, 2013. [CrossRef] [Google Scholar] [Publisher Link]

[8] Cristinel Ababei, and Rajendra Katti, "Achieving Network on Chip Fault Tolerance by Adaptive Remapping," *2009 IEEE International Symposium on Parallel & Distributed Processing*, Rome, Italy, pp. 1-4, 2009. [CrossRef] [Google Scholar] [Publisher Link]

[9] Amit Kumar Singh et al., "Run-Time Mapping of Multiple Communicating Tasks on MPSoC Platforms," *Procedia Computer Science*, vol. 1, no. 1, pp. 1019-1026, 2010. [CrossRef] [Google Scholar] [Publisher Link]

[10] Yang Ge, Qinru Qiu, and Qing Wu, "A Multi-Agent Framework for Thermal Aware Task Migration in Many-Core Systems," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 20, no. 10, pp. 1758-1771, 2012. [CrossRef] [Google Scholar] [Publisher Link]

[11] Amit Kumar Singh et al., "Communication Aware Heuristics for Run-Time Task Mapping on NoC-Based MPSoC Platforms," *Journal of Systems Architecture*, vol. 56, no. 7, pp. 242-255, 2010. [CrossRef] [Google Scholar] [Publisher Link]

[12] Chen-Ling Chou, and Radu Marculescu, "Contention-Aware Application Mapping for Network-on-Chip Communication Architectures," *2008 IEEE International Conference on Computer Design*, Lake Tahoe, CA, pp. 164-169, 2008. [CrossRef] [Google Scholar] [Publisher Link]

[13] A. Bender, "MILP Based Task Mapping for Heterogeneous Multiprocessor Systems," *Proceedings EURO-DAC '96. European Design Automation Conference with EURO-VHDL '96 and Exhibition*, Geneva, Switzerland, pp. 190-197, 1996. [CrossRef] [Google Scholar] [Publisher Link]

[14] Sarzamin Khan et al., "An Efficient Algorithm for Mapping Real Time Embedded Applications on NoC Architecture," *IEEE Access*, vol. 6, pp. 16324-16335, 2018. [CrossRef] [Google Scholar] [Publisher Link]

[15] Sourabh Katoch, Sumit Singh Chauhan, and Vijay Kumar, "A Review on Genetic Algorithm: Past, Present, and Future," *Multimedia Tools and Applications*, vol. 80, pp. 8091-8126, 2021. [CrossRef] [Google Scholar] [Publisher Link]

[16] J. Kennedy, and R. Eberhart, "Particle Swarm Optimisation," *Proceedings of ICNN'95 - International Conference on Neural Networks*, Perth, WA, Australia, vol. 4, pp. 1942-1948, 1995. [CrossRef] [Google Scholar] [Publisher Link]

[17] Seyedali Mirjalili, and Andrew Lewis, "The Whale Optimisation Algorithm," *Advances in Engineering Software*, vol. 95, pp. 51-67, 2016. [CrossRef] [Google Scholar] [Publisher Link]

[18] Muhammad Junaid Mohiz et al., "Application Mapping Using Cuckoo Search Optimisation with Lévy Flight for NoC-Based System," *IEEE Access*, vol. 9, pp. 141778-141789, 2021. [CrossRef] [Google Scholar] [Publisher Link]

[19] Bahador Boroumand, Elham Yaghoubi, and Behrang Barekatain, "An Enhanced Cost-Aware Mapping Algorithm Based on Improved Shuffled Frog Leaping in Network on Chips," *The Journal of Supercomputing*, vol. 77, pp. 498-522, 2021. [CrossRef] [Google Scholar] [Publisher Link]

[20] Nidhi Anantharajaiah, Felix Knopf, and Juergen Becker, "Ant Colony Optimisation Based NoCs for Flexible Spatial Isolation in Mixed Criticality Systems," *2021 IEEE 34th International System-on-Chip Conference (SOCC)*, Las Vegas, NV, USA, pp. 248-253, 2021. [CrossRef] [Google Scholar] [Publisher Link]

[21] Jingcao Hu, and R. Marculescu, "Energy-Aware Mapping for Tile-Based NoC Architectures under Performance Constraints," *Proceedings of the 2003 Asia and South Pacific Design Automation Conference*, Kitakyushu, Japan, pp. 233-239, 2003. [CrossRef] [Google Scholar] [Publisher Link]

[22] Anh T. Tran, and Bevan M. Baas, "*Noctweak: A Highly Parameterizable Simulator for Early Exploration of Performance and Energy of Networks On-Chip*," Technical Report, VLSI Computation Lab, ECE Department, University of California, 2012. [Google Scholar] [Publisher Link]

[23] Xin-She Yang, "Firefly Algorithm, Levy Flights and Global Optimization," *Research and Development in Intelligent Systems XXVI*, pp. 209-218, 2010. [CrossRef] [Google Scholar] [Publisher Link]

[24] Surafel Luleseged Tilahun, and Hong Choon Ong, "Modified Firefly Algorithm," *Journal of Applied Mathematics*, 2012. [CrossRef] [Google Scholar] [Publisher Link]

[25] Débora Matos et al., "Reconfigurable Routers for Low Power andHigh Performance," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 19, no. 11, pp. 2045-2057, 2011. [CrossRef] [Google Scholar] [Publisher Link]

[26] Waqar Amin et al., "HyDra: Hybrid Task Mapping Application Framework for NoC-Based MPSoCs," *IEEE Access*, vol. 11, pp. 52309-52326, 2023. [CrossRef] [Google Scholar] [Publisher Link]