Mobile Networks and Applications (2022) 27:637–651
https://doi.org/10.1007/s11036-021-01827-0

Mobile Networks-on-Chip Mapping Algorithms for Optimization of Latency and Energy Consumption

Arvind Kumar[1] · Vivek Kumar Sehgal[1] · Gaurav Dhiman[2] · S. Vimal[3] · Ashutosh Sharma[4] · Sangoh Park[5]

Accepted: 8 July 2021 / Published online: 18 September 2021

Abstract
With the advancement in technology, it is now possible to integrate hundreds of cores onto single silicon semiconductor chip or silicon die. In order to provide communication between these cores, large number of resources are required and it leads to the communication problem in System-on- Chip (SoC), which is solved by introduction of Networks-on-Chip (NoC). NoC proves to be most efficient in terms of flexibility, scalability and parallelism. In this paper, the proposed mapping algorithms, Horological Mapping (HorMAP), Rotational Mapping (RtMAP) and Divide and Conquer Mapping (DACMAP) for mapping of tasks onto cores, basically concentrate on the optimization of latency, queuing time, service time and energy consumption of topology at constant bandwidth required. The experimental results discussed in this paper shows the comparison of proposed algorithms with traditional random mapping algorithm. In this paper, 2D mesh topology with XY routing is considered for the simulation of proposed algorithms.

Keywords System-on-chip · Networks-on-Chip · NoC topology · Mapping algorithm · Energy consumption

1 Introduction

Various semiconductor industries continues to build a kind of chip that can accommodate a high density of very large-scale integrated circuits (VLSI). In order to integrate all essential components such as IP cores, which includes RAM, counters, interfaces, voltage regulator etc., and System-on-Chip (SoC) is the methodology to be used for this purpose [1, 28]. SoC integrates all these components onto a single chip. In past few years, as number of cores are increasing, the design structure of SoC becomes more complex. Due to complexity, SoC design are less flexible and lead to problem of communication between different cores. Networks-on-Chip (NoC) was introduced as design concept for SoC with support of communication, providing better and powerful solution to connect different intellectual property IP cores through scalable interconnection network [2–4, 16]. NoC architecture comprises of interconnected devices like processors (IP cores, DSP, ASIC etc.), routers, network interfaces, and communication links or channels as shown in Fig. 1. Communication between different cores is achieved by sending and receiving packets over interconnection network. NoC offers flexible mechanism by supporting different interconnection networks and fault tolerance. Interconnection network communicates through various communication protocols which are useful for enhancing the flexibility of systems when merged with VLSI [5, 6, 17].

On-chip interconnected networks has benefits over shared wiring and buses, i.e., high-bandwidth utilization, less latency, low power consumption, scalability and flexibility. A lot of research work have been done in area of NoC in order to optimize the design and, the main area where designing need to be more focused are topology generation, scheduling, mapping, routing and floorplanning [7, 22]. Each area has its own important role in order to provide better performance of multiprocessor systems. In this paper, we mainly focused on mapping of task on 2D NoC architecture. Task mapping comprises of finding best placement or mapping of task in such a way so that mapping fulfill set of certain requirements like less energy consumption, reduction in congestion and less latency keeping constant bandwidth constraints. There are two approaches for mapping of tasks on the cores, i.e. static or dynamic task mapping [8, 9, 23]. Static task mapping approach states that tasks should be placed at

✉ Gaurav Dhiman
gdhiman0001@gmail.com

Extended author information available on the last page of the article.

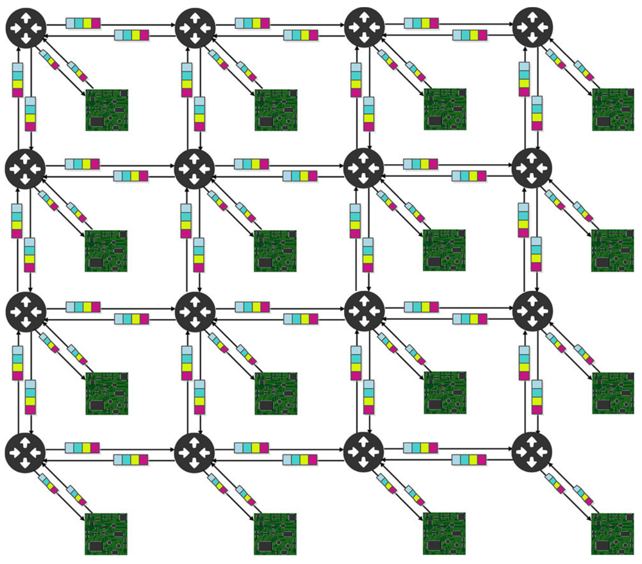


Fig. 1 NoC Mesh topology architecture

design time. As different tasks are executed at design time, static mapping uses different composite algorithms to investigate the SoC resources, which further results in optimized solutions such as less energy consumption and efficient performance of multiprocessor SoC. The major drawback of static mapping approaches are that they do not have capability to handle newly arrived tasks or application, which may be loaded during run-time. In order to tackle this problem in future, dynamic or run-time mapping techniques are introduced, which can map dynamic tasks onto the topology at run-time. In this paper, we have proposed mapping algorithms, HorMAP, RtMAP and DACMAP for mapping of tasks onto topology having different cores, so that latency, queuing time, service time and energy consumption of topology are minimized.

2 Related work

There are various mapping algorithm developed by different researchers to provide better performance in terms of energy consumption, latency, thermal behavior, and bandwidth constraints that should be minimized. Ning et al. [10] proposed a mapping algorithm named GA-MMAS, which is combination of Genetic Algorithm (GA) and MAX-MIN Ant System (MMAS), to optimize energy consumption for NoC. Jang et al. [11] proposed A3MAP which is Architecture-Aware Analytic Mapping algorithm that can

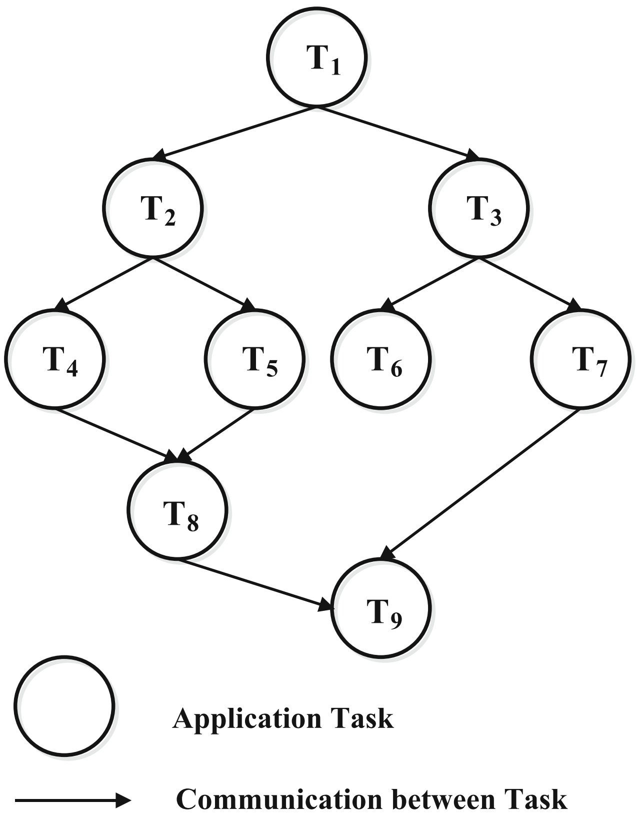


Fig. 2 Logical Application Trace graph

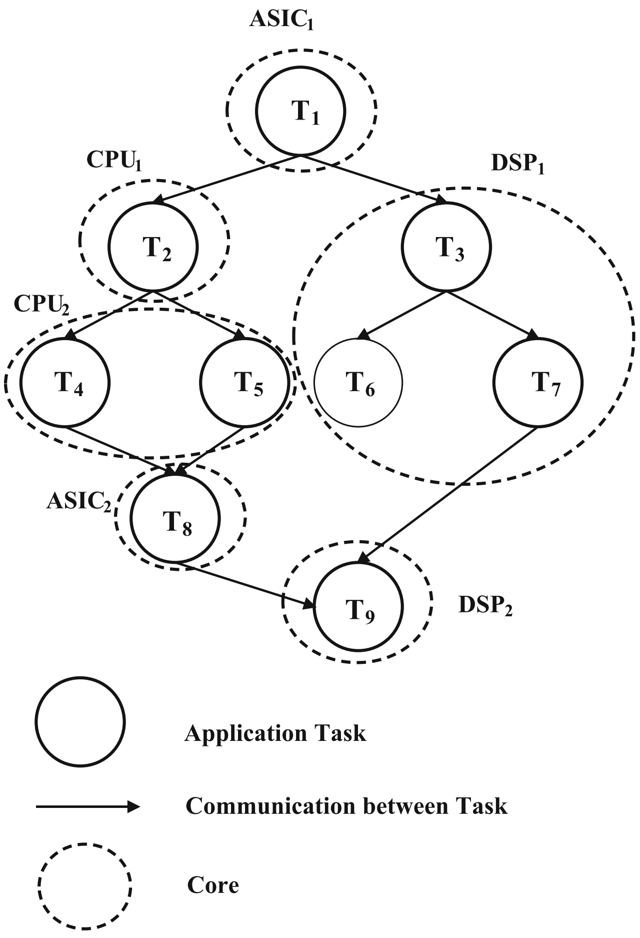


Fig. 3 NoC Architecture Characterization Graph

be applied to regular mesh architecture with homogeneous cores as well as on irregular mesh or custom architecture with heterogeneous cores. At first, author developed an interconnection matrix for modelling any task graph and network, then task mapping problem is converted to MIQP (Mixed Integer Quadratic Programming). As MIQP is NP-hard problem, then author proposed two heuristics techniques, a successive relaxation algorithm (A3MAP-SR) and a genetic algorithm (A3MAP-GA) to reduce amount of traffic by comparing regular and irregular mesh, and custom network. Yin et al. [12] proposes an application mapping technique that incorporates domain knowledge into genetic algorithm (GA) to minimize the energy consumption of NoC communication.

The GA is initialized with knowledge on network partition whereas the genetic crossover operator is guided with communication demands. The effects of domain knowledge GA on initial population and genetic operator are analyzed in terms of the solution quality and convergence speed. Fen et al. [13] developed GAMR, which is genetic algorithm based mapping and routing technique for 2D regular Network on chip (NoC) architecture under bandwidth constraint. The main focus of author is to minimize energy consumption and maximize bandwidth link utilization of the NoC design. GAMR mapping maps IP cores of application onto NoC topology which leads to generation of a deterministic deadlock free minimal routing path for every communication trace.

Wang explores the bandwidth and latency based IP mapping that a set of IP cores onto the tiles of mesh NoC topology in order to minimize the power consumption having inter-core communications [14]. By analyzing different applications communication characteristics with their communication trace graphs, author recognizes two connectivity templates first one is graphs with tightly coupled vertices and other one with distributed vertices. Author developed different mapping approaches for these templates, in which tightly coupled vertices are mapped onto topology tiles that are very close to one another and in other case, the distributed vertices are mapped according to graph partition scheme given by author. Murali et al. [15] introduced NMAP, a fast algorithm for mapping the cores on a mesh interconnection architecture under bandwidth constraints in order to minimizing the average communication delay. The NMAP algorithm is designed for single minimum-path routing and also for split traffic routing. The algorithm is applied to a DSP benchmark design and simulation was done by author using xpipes library. Yang et al. [18] divided NoC-based MPSoC design process into two steps that is, scheduling subtasks to appropriate processing elements having appropriate types and quantity and then mapping those processing elements onto NoC topology. Particle swarm optimization (PSO) was used to achieve first step with less amount of task execution time, less task running and transfer cost. The outcome of first step was communication diagram and second step shows least network transmission delay and least resource consumption as well as power consumption. Qianqi et al. [19] finds the Pareto optimal solutions rather than a single solution which are usually obtained through scalarization. Author proposed fault-tolerant routing and improved particle swarm optimization to meet NoC requests and have capability of searching solutions. Proposed methods solved tradeoff between high performance and reliability of the system. Srinivasan et al. [20] present a technique to reconfigure the network dynamically among different use-cases

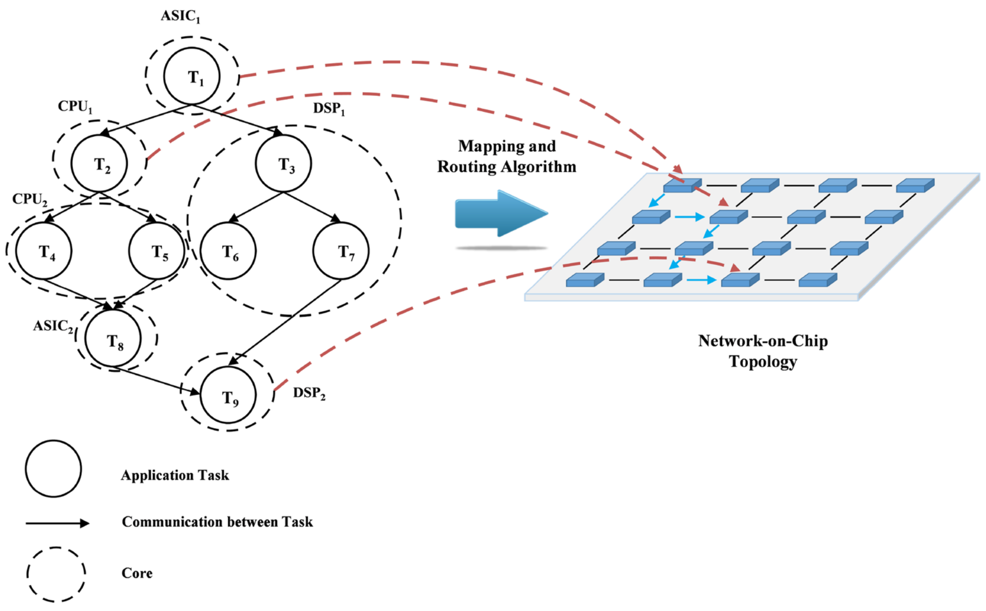


Fig. 4 NoC mapping technique

and explain the how to integrate Dynamic Voltage and Frequency Scaling (DVS/DFS) techniques with those use-case centric NoC design. This dynamically reconfiguration of the NoC along with integration of DVS/DFS schemes resulting in less power consumption for NoC systems. Mehran et al. [21] proposed SPIRAL algorithm for mapping of tasks on different cores, which minimizes energy consumption. For implementing SPIRAL algorithm, author used 2D mesh topology along with XY routing and used MATLAB tool in order to evaluate the performance of proposed SPIRAL algorithm. SMAP, a tool for generating random graphs, was used by researchers. Author have compared SPIRAL algorithm along with random mapping and genetic mapping algorithm to show improved result in terms of energy consumption. If spiral mapping algorithm is used and there are very few task to be mapped then also in case of the spiral mapping the middle core is chosen to map the task and as the middle core is farther from the task queue and hence the processing gets slower and mapping the task takes lots of time.

Marcon et al. [24] proposed combination of communication dependence as well as computation model(CDCM) for application mapping on regular NoCs. Using CDCM technique, author estimated 40% reduction in execution time

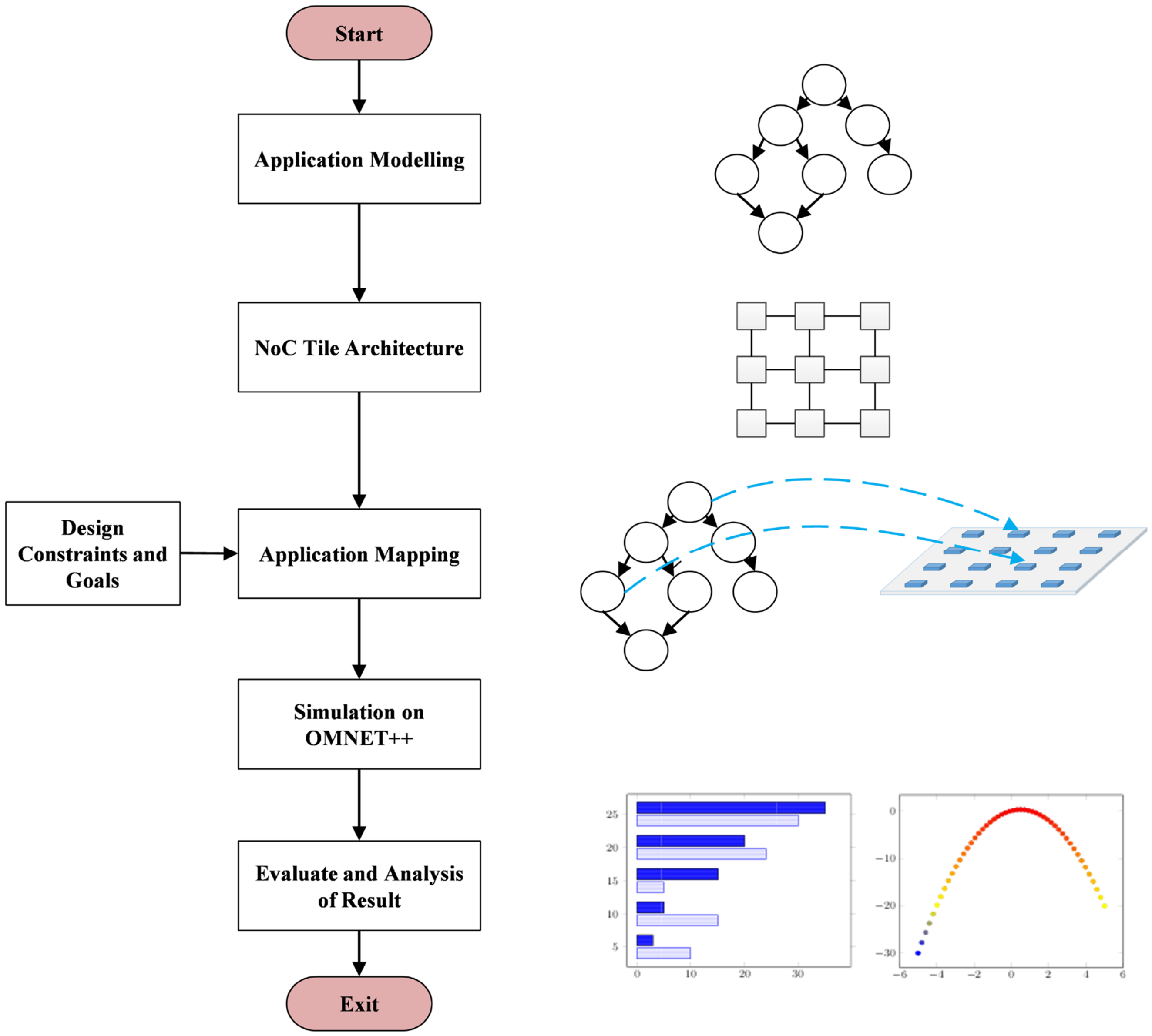


Fig. 5 Flowchart representation of application mapping onto topology

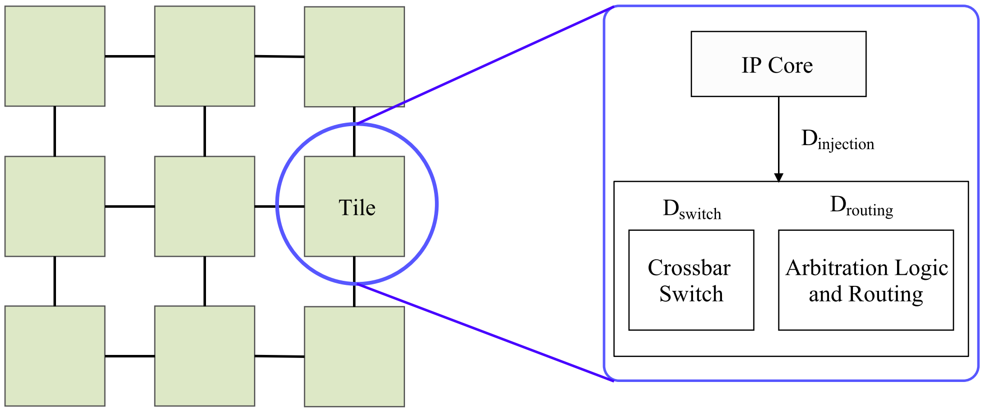


Fig. 6 NoC topology tile

and 20% reduction in energy consumption. Celik has discussed the effect of mapping of application on NoC with the help of network traffics that encapsulates the self-similarity [25]. Author considered queuing delay and packet loss rate parameters in order to analyze the effect of application mapping. Jiawen et al. [26] proposed logistic function based adaptive genetic algorithm (LFAGA) for energy efficient mapping of application on 3D NoC. The result of LFAGA is compared with chaos-genetic mapping algorithm (CGMAP), which saves 18.6% of energy consumption. Harmanani proposed an effective routing algorithm, whose main concern is to minimize blocking in routing [27]. The author uses 2D mesh topology and benchmarks like VOPD, DSP filter, LinearP15 in order to simulate the results.

The rest of the paper is organized as: Section 3 includes the problem formulation and mathematical representation of the proposed approach. Section 4 explains existing random mapping algorithm. Section 5 contains brief explanation about proposed approach. The experimental results are included in Section 6 followed by conclusion and future work mentioned in Section 7.

3 Problem formulation

Before formulating mapping problem, we assume that in order to perform mapping, we are given with application that is characterized by set of tasks which performs scheduling onto NoC cores. For appropriate understanding of mapping problem strategies, some important definition need to be explained.

Definition 1 *A Logical Application Trace Graph (LATG)* $G = (A_t, E_t)$ is an directed acyclic graph, where $a_t \in A_t$ represents task from list of application tasks and $c_{i,j} \in E_t$ is an directed arc between application tasks, that shows communication dependency between tasks a_{t1} and a_{t2}. Logical application trace graph is depicted in Fig. 2. Each directed edge or arc has one property:-

– $v(c_{i,j})$ represents volume bits transferred between from arc c_i to c_j.

Definition 2 *NoC Architecture Characterization Graph (NACG)* $G = (T, L_T)$ represents undirected graph as shown

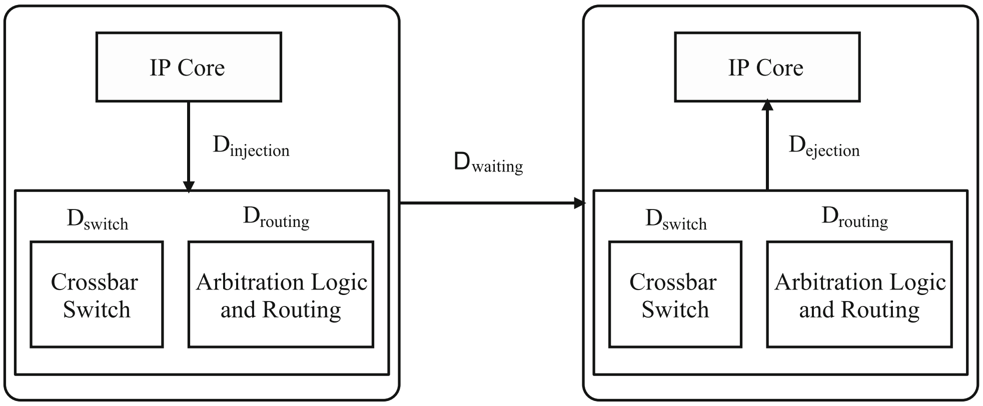


Fig. 7 Latency flow in single hop

in Fig. 3, where vertex node $t_i, t_j \in T$ shows tiles in NoC architecture, whereas $l_k = l_{i,j} = (t_i, t_j) \in L_T$ represents routing path between t_i and t_j. Routing path in NACG consists of two properties :-

– $e(l_{i,j})$ is average energy consumption of task in bits from t_i and t_j.
– $Lat(l_{i,j})$ represents average latency of task from t_i and t_j.
– $band(l_{i,j})$ is defined as bandwidth of link between t_i and t_j.

Definition 3 A mapping function (Ω) is represented as $\Omega : A_t \rightarrow T$, that shows mapping of application tasks from LATG onto tiles available in NACG, where $a_t \in A_t$ and $\Omega(a_t) \in T$ and $\Omega(a_t)$ characterizes mapped tile in NACG. Fig. 4 shows mapping of application tasks onto NoC tile based architecture.

Finally, the formulation of mapping problem is as follows:
Given: An LATG $G = (A_t, E_t)$ and NACG $G = (T, L_T)$, **Evaluate** mapping function $\Omega : A_t \rightarrow T$, that maps task $a_t \in A_t$ in LATG to tile $t_i \in T$ in NACG, **such that** energy consumption and average latency is minimized. Fig. 5 represents the flow of mapping of application task onto topology in order to get optimized results in terms of energy consumption and average latency.

3.1 Energy model

The objective function is to minimize the energy consumption, which can be mathematically represented as:

$$\min \left\{ \sum_{\forall a_t \in A_t} e_{\Omega(a_t)} + \sum_{\forall c_{i,j} \in E_t} v(c_{i,j}) \times \sum_{l_{i,j} \in R_{\Omega(a_{t1}),\Omega(a_{t2})}}^{|R_{\Omega(a_{t1}),\Omega(a_{t2})}|} e(R_{\Omega(a_{t1}),\Omega(a_{t2})}) \right\} \tag{1}$$

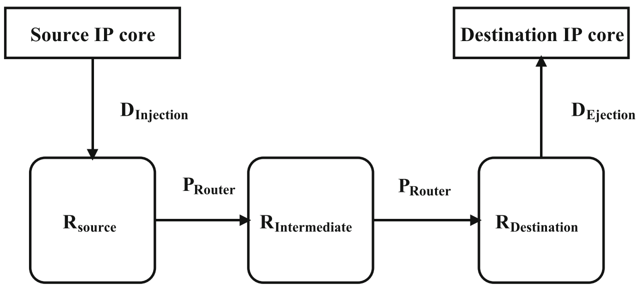


Fig. 8 Latency flow in two hop from source to destination core

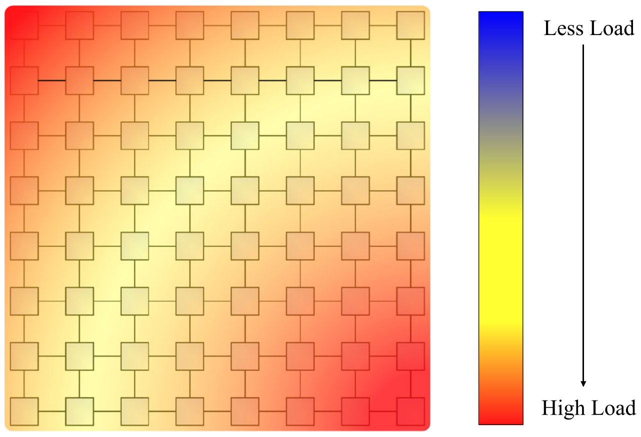


Fig. 9 Load balancing in random mapping algorithm

satisfying conditions as

$$\forall a_t \in A_t, \forall \Omega(a_t) \in T \tag{2}$$

$$\forall a_{t1} \neq a_{t2}, \forall \Omega(a_{t1}) \neq \Omega(a_{t2}) \tag{3}$$

The average energy consumption for transferring task from t_i to t_j can be represented as follows:

$$E_{task}^{t_i,t_j} = N \times num_{hops} \times E_{Link} + N \times (num_{hops} - 1) \times E_{Router} \tag{4}$$

where, E_{Link} and E_{Router} represents energy consumption of link and energy consumption of router. In order to compute E_{Router}, we have to compute energy consumption of buffer (E_{Buffer}), energy consumption of crossbar switch ($E_{Crossbar}$) and energy consumption of arbiter

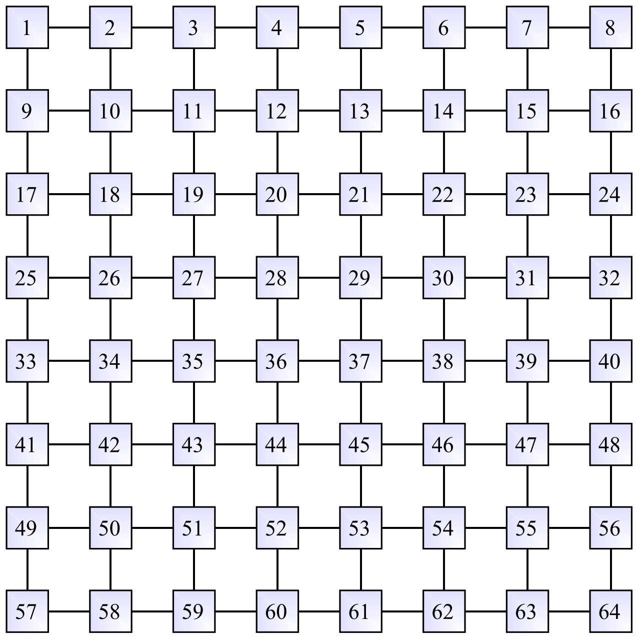


Fig. 10 Horological mapping algorithm

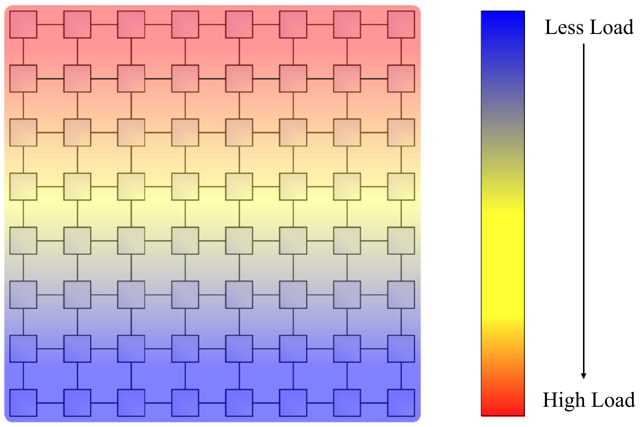


Fig. 11 Load balancing in horological mapping algorithm

($E_{Arbiter}$). $E_{Arbiter}$ is further divided into two parts : (i) $E_{Crossbar\_Allocation}$, energy consumption of switch allocation and (ii) $E_{VC\_Allocation}$, energy consumption of virtual channel allocation. E_{Link} can be computed as gievn in Eq. 7. Energy consumption of topology is calculated for all N tasks is given in Eq. 8.

$$E_{Router} = E_{Buffer} + E_{Crossbar} + E_{Arbiter} \tag{5}$$

$$E_{Arbiter} = E_{Crossbar\_Allocation} + E_{VC\_Allocation} \tag{6}$$

$$E_{Link} = \frac{P_{Link}}{Freq.} \tag{7}$$

$$E_{Total} = \sum_{i=1}^{N} E_{task_i} \tag{8}$$

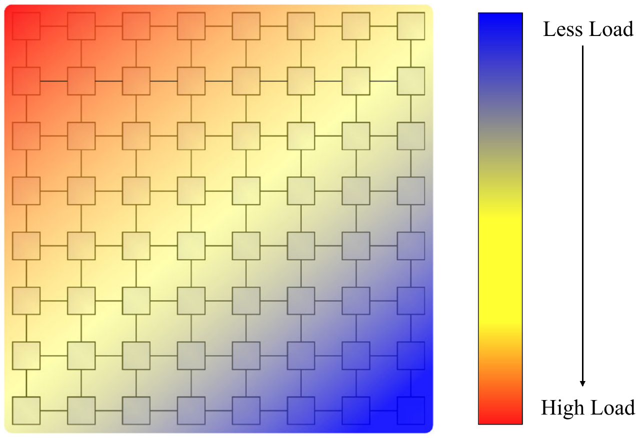


Fig. 12 Load balancing in rotational mapping algorithm

3.2 Latency model

The mapping function for minimization of average latency of topology can be mathematically formulated as:

$$\min \left\{ \sum_{\forall a_t \in A_t} Lat_{\Omega(a_t)} + \sum_{\forall c_{i,j} \in E_t} v(c_{i,j}) \times \sum_{l_{i,j} \in R_{\Omega(a_{t1}),\Omega(a_{t2})}}^{|R_{\Omega(a_{t1}),\Omega(a_{t2})}|} Lat(R_{\Omega(a_{t1}),\Omega(a_{t2})}) \right\} \tag{9}$$

satisfying conditions as

$$\forall a_t \in A_t, \forall \Omega(a_t) \in T \tag{10}$$

$$\forall a_{t1} \neq a_{t2}, \forall \Omega(a_{t1}) \neq \Omega(a_{t2}) \tag{11}$$

The latency from tile t_i to tile t_j can be computed according to Eq. 12. The overall latency for all N tasks is calculated by Eq. 13.

$$Lat_{task}^{t_i,t_j} = N \times num_{hops} \times Lat_{Link} + N \times (num_{hops} - 1) \times Lat_{Router} \tag{12}$$

$$Lat_{Total} = \sum_{i=1}^{N} Lat_{task_i} \tag{13}$$

Figure 6 shows the 3 × 3 NoC topology in the form of tile, where each tile consist of cores (that can be IP core, DSP core etc.) and routers (consists of crossbar switch, routing algorithm and arbitration logic). Latency of single task to be transferred across channel are $D_{injection}$ and $D_{ejection}$ respectively and latency of a task across router are D_{switch}, $D_{routing}$ and $D_{waiting}$. In Fig. 7, we have considered link injection latency ($D_{injection}$), latency of first router ($D_{switch} + D_{routing}$), inter-tile latency ($D_{waiting}$) which is defined as how long, the task takes to arrive to the destination from time the first bit is sent out from source for a single hop, second router latency ($D_{routing} + D_{switch}$), and link ejection latency ($D_{ejection}$). Latency flow of single hop can be calculated according to Eq. 14:

$$\begin{aligned} Latency\_single\_hop = {} & D_{injection} + (D_{routing} + D_{switch}) + D_{waiting} \\ & + (D_{routing} + D_{switch}) + D_{ejection} \end{aligned} \tag{14}$$

In order to calculate the latency from source to destination core, we have assumed that as task arrives to destination core, then the task is immediately accessible for processing by destination core. In Fig. 8, the latency involved, is considered from source IP core to destination IP core passing through routers are R_{source}, $R_{intermediate}$ and $R_{Destination}$. The latency of task having two hops between source and destination core ($L_{source \rightarrow destination}$) is calculated as given in Eq. 15, where W^{source}, $W^{destination}$ and $W^{intermediate}$ represents the waiting time in routers.

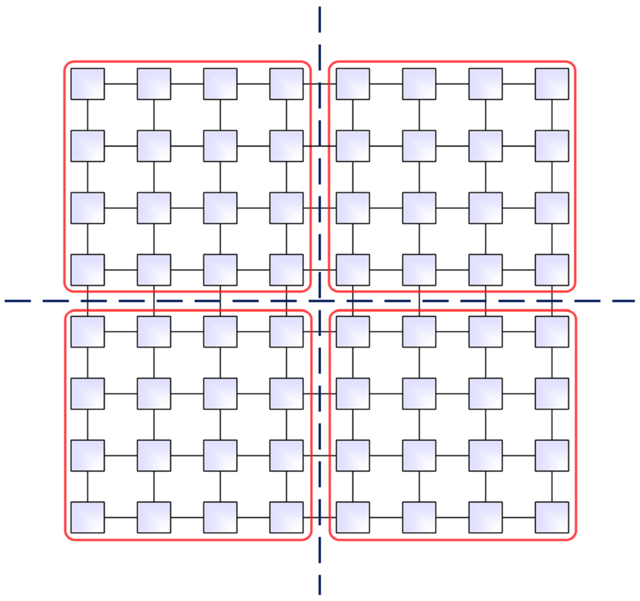


Fig. 13 Grid Divison into sub-grid

The average latency of task (L) can be calculated in Eq. 16, where $P_{source \to destination}$ is probability of task to be generated.

$$\begin{aligned} L_{source \to destination} &= D_{injection} + (D_{routing} + W^{source}_{inj \to port} + D_{switch}) \\ &+ D_{waiting} + (D_{routing} + W^{intermediate}_{port \to port} + D_{switch}) \\ &+ D_{waiting} + (D_{routing} + W^{destination}_{port \to ejc} + D_{switch}) \\ &+ D_{ejection} + (m-1)(D_{switch} + D_{waiting}) \end{aligned} \quad (15)$$

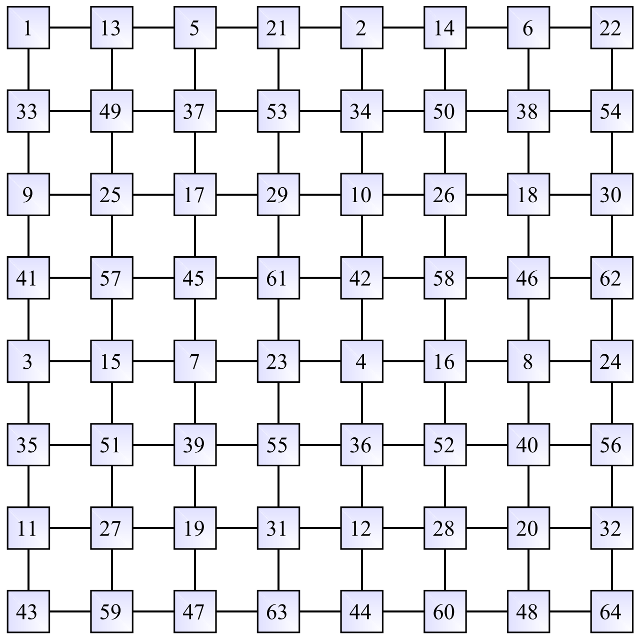


Fig. 14 Divide and conquer mapping algorithm

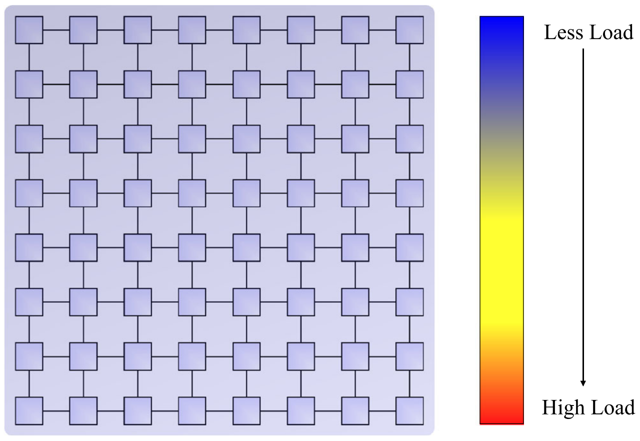


Fig. 15 Load balancing in divide and conquer mapping algorithm

$$L = \sum_{source} \sum_{destination} P_{source \to destination} \times L_{source \to destination} \quad (16)$$

4 Existing random mapping algorithm

There are many issues involved using random mapping algorithm, such as load balancing as shown in Fig. 9, latency, service time and queuing time are not handled by random algorithm for NoC. In random algorithm, tasks are mapped on the cores randomly as discussed in Algorithm 1 . The worst case of the algorithm is, when every time the same core is chosen for mapping the task. As all tasks are mapped on the same core, so, the new tasks to be mapped will remain in the queue and wait for an infinite period of time till the core is not ready to process the new task. Once the core is available task is mapped on the core. In the best case

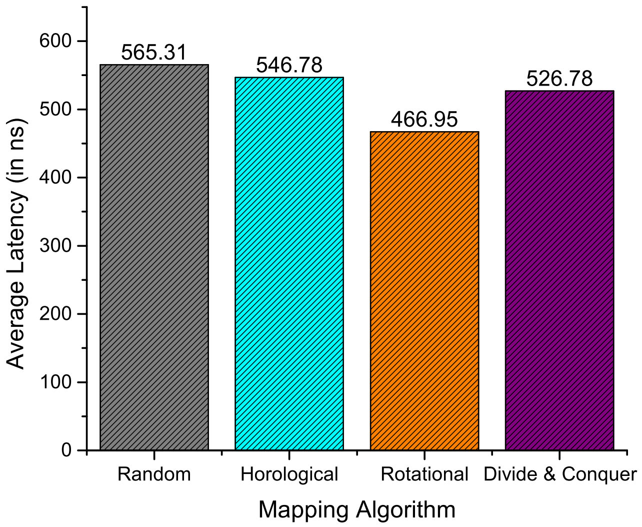


Fig. 16 Average latency (in ns) of mapping algorithms in mesh topology

of random algorithm for mapping, the randomly chosen cores will have an equal probability to be chosen, and task will be mapped on to these cores uniformly. There are rare chances to obtain the best case of the random algorithm. Let us consider a scenario that every time the last core of the grid is chosen to map the tasks. If such a case exist then latency involved to map the tasks on the cores will be very high. So mapping the task on to the cores in case of random algorithm consumes a large amount of latency, service time, queuing time and the energy consumption. To improve the performance of the mapping algorithm in this paper, the horological, rotational and divide and conquer mapping algorithms are proposed.

Algorithm 1 Random mapping algorithm.

```
Data: $dst\_core$ as the destination core, id be the unique
      number assigned to each core and $id \in [0, n)$, n × n
      mesh topology is given having $n^2$ cores and $t_n$ be the
      number of task.
Result: Task mapped on cores randomly
while ($t_n > 0$) do
    dst_core = (id + intuniform(1,n)) % n;
    //returns a random core from n cores available.
    Assign task to dst_core;
    $t_n$–;
```

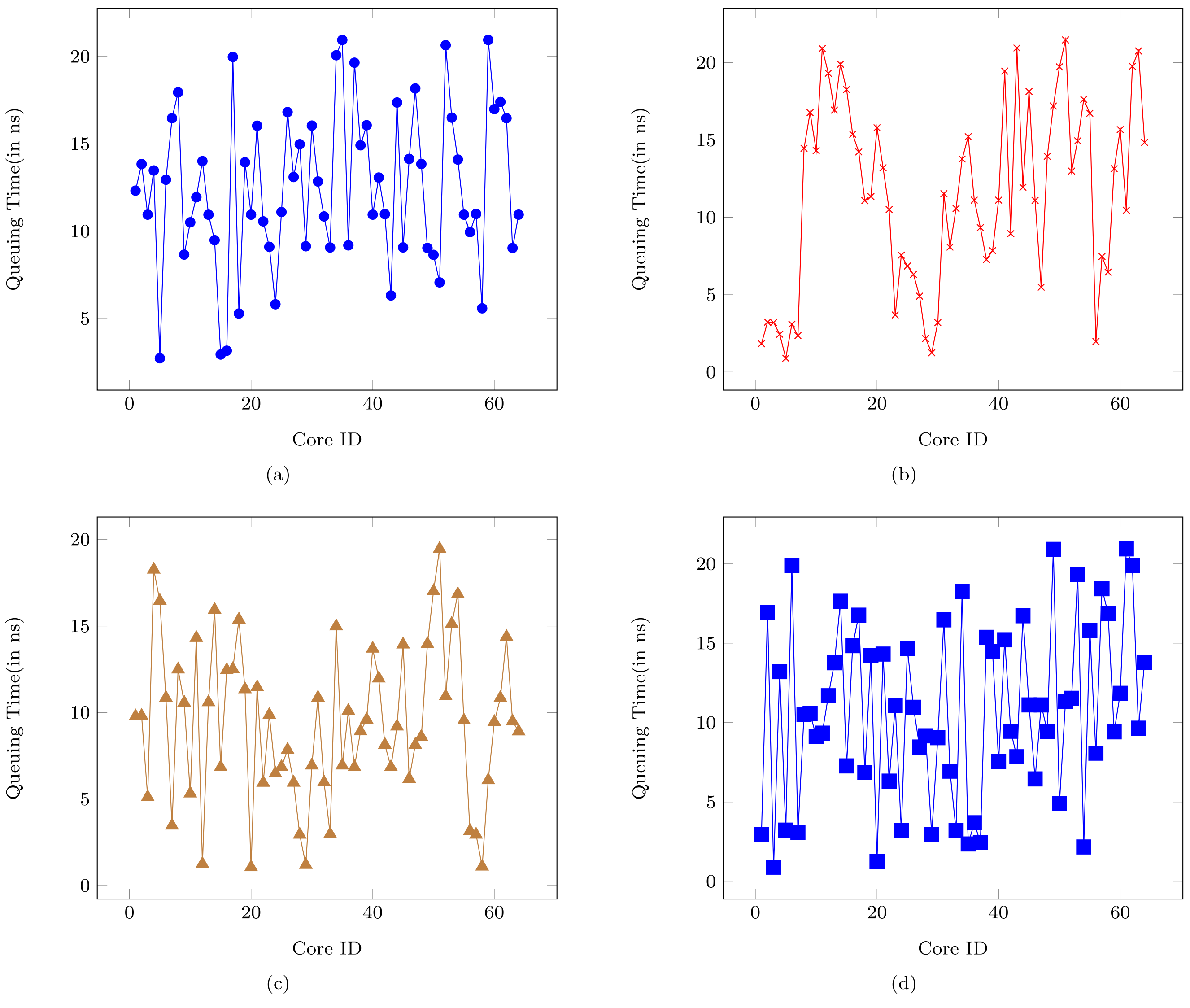


Fig. 17 Queuing Time of mesh topology (a) random mapping (b) horological mapping (c) rotational mapping (d) divide and conquer mapping

5 Proposed approach

In this section the three proposed approaches are discussed which proves to be better than the existing random mapping algorithm in terms of latency, load balancing and energy consumption. First approach discussed is horological mapping algorithm, in which the cores are visited one by one guaranteeing load balancing over the cores of the grid. Second approach is the rotational mapping algorithm. In this the task are assigned to the cores in rotation one by one guaranteeing the least latency involved during mapping of tasks. The third algorithm proposed in the paper is the divide and conquer mapping algorithm, which provides an assurity of load balancing on the grid.

5.1 Horological algorithm

As the name suggest, in this mapping algorithm the tasks are mapped horologically on the cores one by one. As the task are assigned to the cores, then the core will process these task, and after the processing of task, the core gets ready to execute the next task in the queue. In this, the cores are allotted an core_id horologically. The first task in the queue is allocated to the first core, second task to the second core and so on. When the task on some core is completed, then a new task is allocated to this core. This algorithm produces good results in terms of load balancing on the cores, but the accessing time of the core increases as we moves towards the last core, with the last core having the maximum access time. So the accessing time of the cores is increased moving towards the last core. Fig. 10 shows the allocation of task on 8 × 8 mesh topology. For an instance, suppose there are 8 tasks which are to be mapped on the cores then even if the core with core_id 8 is closer to the queue the task will not be assigned to it, instead tasks will be assigned to the cores having core_id 0 to core_id 7. Horological mapping proves to be better than the random mapping in terms of load balancing as shown in Fig. 11, queuing time and service time. It also resolve the issue of bottleneck existing in random mapping algorithm. Hence the horological mapping algorithm proves to be better over the random mapping algorithm. Horological mapping algorithm is given in Algorithm 2.

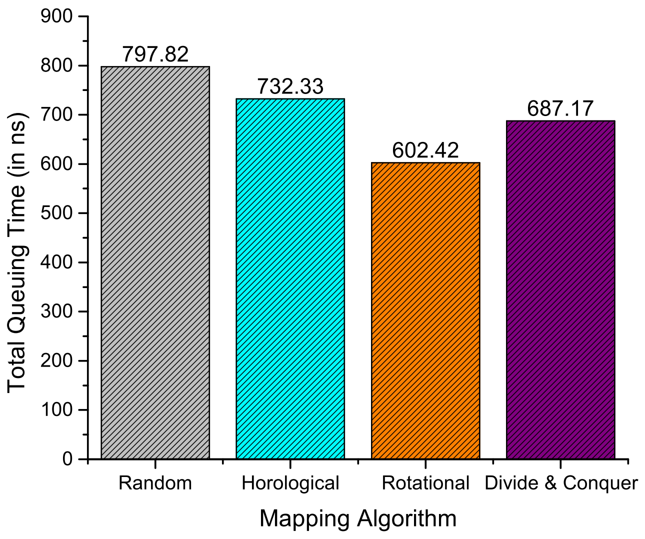


Fig. 18 Total queuing time (in ns) of mapping algorithms in mesh topology

Algorithm 2 Horological mapping algorithm.

```
Data: n × n Mesh topology having $n^2$ cores represented as
      c[i][j], $t_n$ be the number of task.
Result: Cores chosen horologically to map task.
while ($t_n > 0$) do
    for i:=0 to n-1 step 1 do
        for j:=0 to n-1 step 1 do
            Assign the task to core c[i][j];
            $t_n - -$;
```

5.2 Rotational mapping algorithm

Rotational mapping algorithm is proposed in this paper in order to minimize the latency involved during the mapping of the task on to the core, but there is no assurity of load balancing in this mapping algorithm as shown in Fig. 12. In rotational algorithm, task are mapped on the cores in the rotational manner. The basic concern of the proposed approach is to reduce the amount of time required for mapping task on the core. In order to achieve the goal, it is required to map the task on the core which is placed nearest to the task allocation queue, so whenever task has to be mapped, it is mapped on to the core which is nearest to the allocation queue and is in ready state, i.e. it is ready to accept the task for execution. For this purpose, the ports of routers are considered to be very important. In rotational, task to be mapped is routed on to the elements (routers or cores) attached to the ports of the router. For each router starting from port zero to the last port, task are passed to each port in an sequential order. Once all the ports are visited then this procedure repeats from first port of the router to the last port. In this way the algorithm is capable of mapping multiple task on the cores till the task allocation queue is not empty. As the procedure repeats for each router considering all the ports every time hence the algorithm is called as rotational algorithm. Rotational mapping algorithm is given in Algorithm 3.

Algorithm 3 Rotational mapping algorithm.

```
Data: n × n Mesh topology having $n^2$ cores represented as
      c[i][j], $t_n$ be the number of task, r[i][j] be the router
      corresponding to core c[i][j], P[i][j] be the number of
      ports associated to each router, variable assign to track
      the assignment of the task on the core and k[i][j]=0.
Result: Task mapped on cores by rotational Algorithm
while ($t_n > 0$) do
    assign = 0; while ($assign != 1$) do
        k[i][j] = k[i][j] % P[i][j]; //K[i][j] and P[i][j] are
        counters.
        if c[i][j] is attached to port k[i][j] then
            Assign the task to the c[i][j];
            assign = 1;
            k[i][j]++;
            $t_n - -$;
        else
            Pass the task on the router r[m][n] attached at
            port k[i][j];
            i = m;
            j = n;
            $k[i][j] + +$;
```

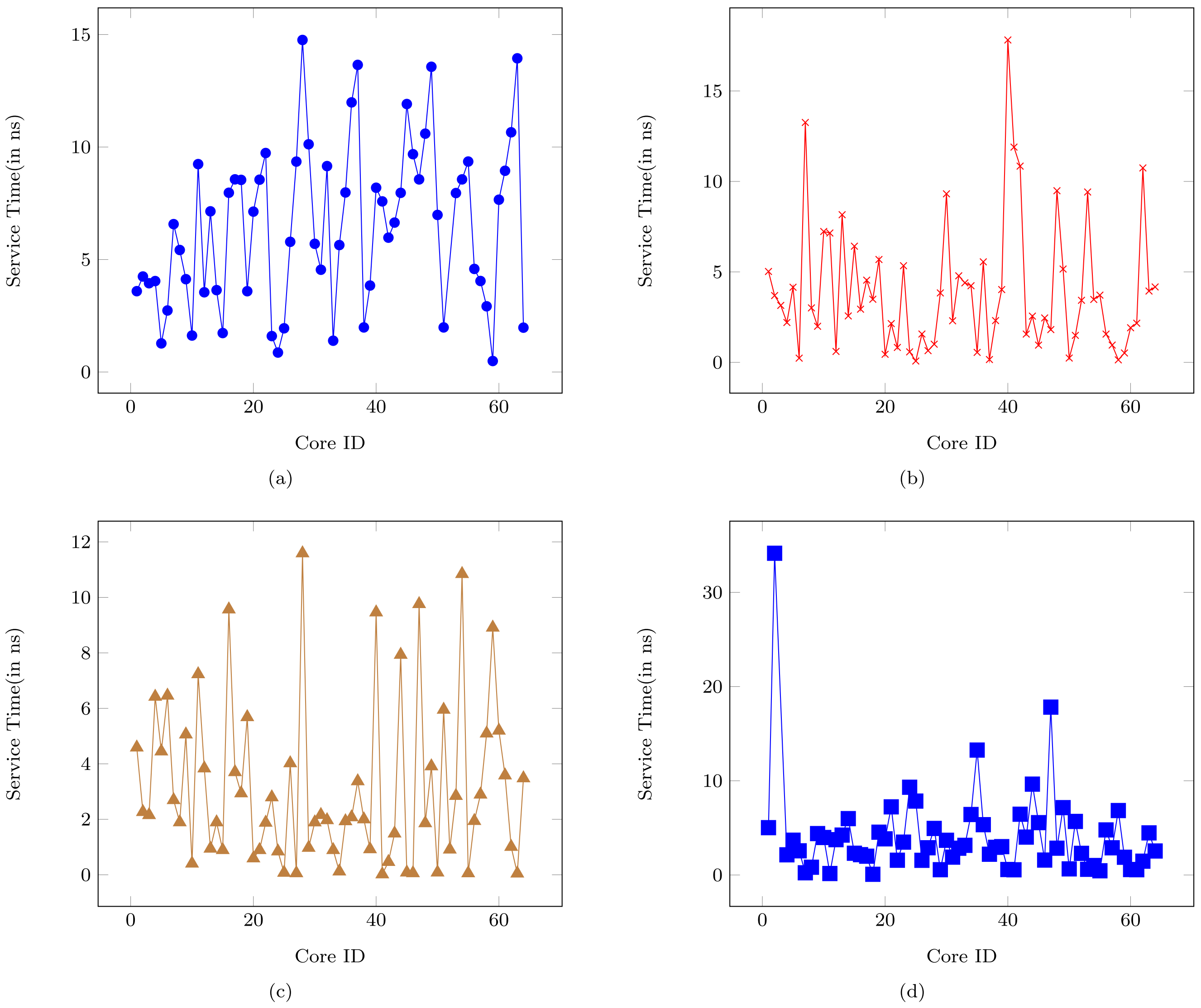


Fig. 19 Service Time of mesh topology (a) random mapping (b) horological mapping (c) rotational mapping (d) divide and conquer mapping

5.3 Divide and conquer mapping algorithm

In divide and conquer mapping algorithm, the main emphasis is on load balancing on n × n mesh topology. As the name suggest, in this algorithm first 2D Mesh topology is divided vertically into two (nearly equal) parts and then the division is carried out horizontally. After each vertical and horizontal division the topology is divided into 4 sub-grids of nearly equal dimensions(rows × columns) as shown in Fig. 13. Different tasks from task list, which are maintained in queue, are being mapped onto sub-grids in such a way that load is equally balanced on the mesh topology. For this purpose each time the task has to be mapped, the grids and sub-grids are further divided both vertically and horizontally. The task is assigned to the core belonging to that sub-grid in which there are least number of task mapped. In this way the task mapped on the cores of sub-grid are balanced, hence there is an assurity of load balancing during the mapping of the task to the cores. For an instance let us consider a simple scenario for mapping task on the 8 × 8 mesh topology, first task from task list is mapped onto first core of first sub-grids. Second task from task list, is mapped onto 5th core belonging to second sub-grids. In the similar way, 3rd task mapped onto 33th core belonging to third sub-grids and next task mapped onto 37th core which belongs to fourth sub-grids. So in this way, all task is mapped onto mesh topology as shown in Fig. 14, assuring the researcher to get a NoC architecture with complete load balancing in Fig. 15. Divide and conquer mapping algorithm is given in Algorithm 4.

Algorithm 4 Divide and conquer mapping algorithm.

```
Data: n × n mesh having $n^2$ cores and $t_n$ number of tasks.
Step I : Divide grid vertically with one partition having
$\lfloor \frac{n}{2} - 1 \rfloor$ cores and other partition having $\lfloor \frac{n}{2} \rfloor$ cores in each row.
Step II : Divide grid horizontally with one partition having
$\lfloor \frac{n}{2} - 1 \rfloor$ cores and other partition having $\lfloor \frac{n}{2} \rfloor$ cores in each column.
Step III : Assign task to first core of each partition.
Step IV : Repeat above steps for each sub-grids obtained
with vertical and horizontal line partitions till grid having
exactly one core is obtained.
Step V : If all task are not assigned to the cores then repeat the
full process given above considering the full n × n grid again.
```

6 Experimental results

For implementation purpose, we have used OMNET++ simulator along with the use of in-built mapping package. In order to implement proposed mapping algorithms, we have considered the 2-dimensional 8 × 8 mesh topology for NoC. Initially, the application tasks are maintained in the task list, which can be the queued. From that task list, tasks are mapped on the cores, following the proposed mapping algorithms as mentioned in section 4. We have perform simulation varying the number of tasks from 64 to 128 and compared the results in terms of latency, queuing time, service time and energy consumption. Fig. 16 shows average latency of proposed mapping algorithms for mesh topology, and results are compared with random mapping algorithm.

Figure 17 gives graphical analysis of queuing time for random and proposed mapping algorithms, and comparison of total queuing time is given in Fig. 18. Results obtained for service time required by each task, using OMNET++ simulator, are shown in Fig. 19. Best mapping algorithm, in terms of total service time can be obtained by the comparative analysis of mapping algorithms as shown in Fig. 20.

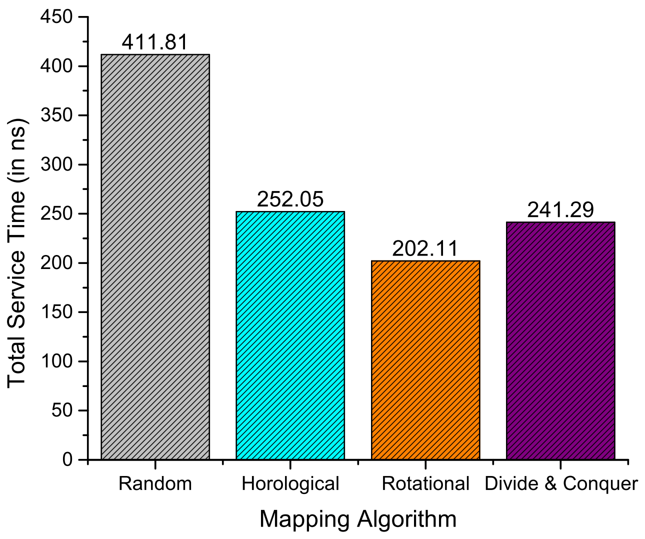


Fig. 20 Total service time (in ns) of mapping algorithms in mesh topology

Table 1 Energy of router (in pJ) at different load

S. No.	Load	Energy of Router (in pJ)
1	0.2	16.8
2	0.4	27.138
3	0.6	37.46
4	0.8	47.78
5	1	58.09

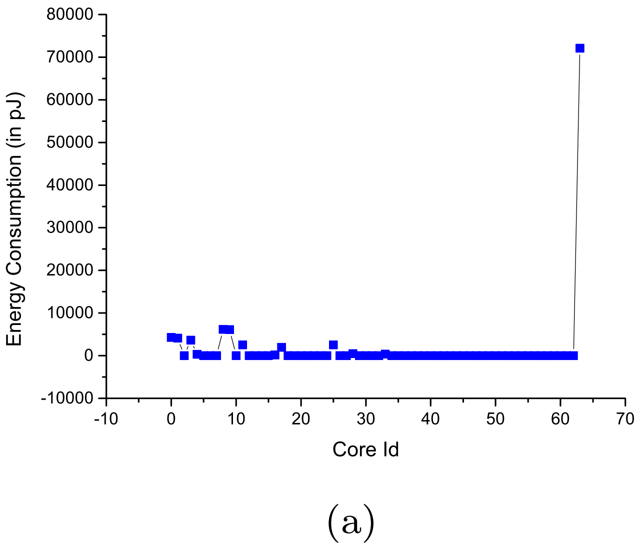
In order to compute the energy consumption of topology, we have used Orion 2.0 simulator. With the help of orion simulator, we calculate the energy consumption of link represented as E_{Link} and energy consumption of router represented as E_{Router}. Table 1 shows the energy of router at different loads. Table 2 represents the energy consumption of link at different link length and different load. With the help of Eq. 4, we compute the energy consumption of individual core as well as energy consumption of topology as shown in Fig. 21 -22. Table 3 shows the comparison of proposed and random mapping algorithm in terms of average latency, total queuing time and total service time for mesh topology.

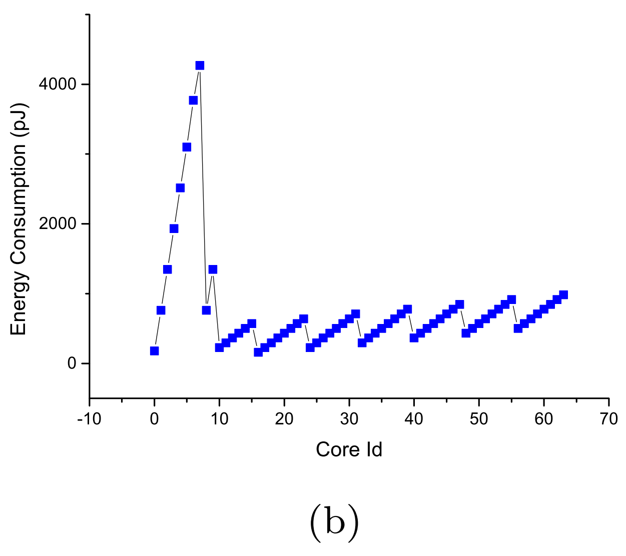
7 Conclusion and future work

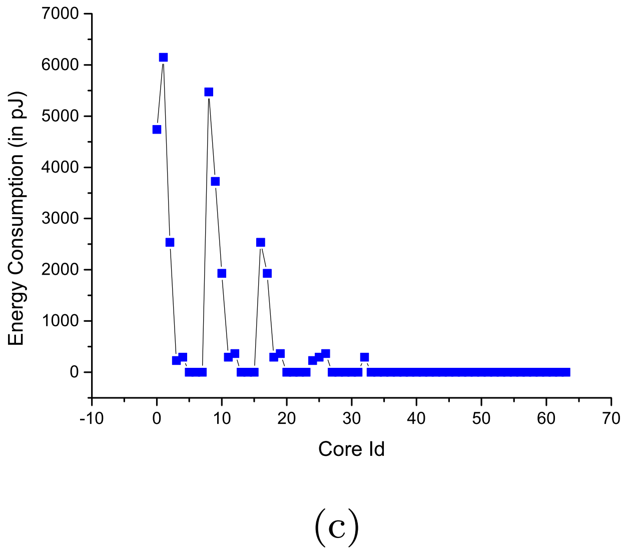
In this paper, we have proposed the mapping algorithms for tile based NoC mesh topology that maps application tasks onto NoC tiles and develops a function such that energy consumption and average latency is minimized satisfying some performance constraints.The processing tiles with high computational power in big little approach are mostly used in ARM based SoC like Apple's M1 processor. These task mapping algorithms can be dynamically applied to these clusters of processing cores with optimized QoS. As future work, our main emphasis is to apply these mapping algorithms as machine learning blended algorithms over different NoC topologies dynamically. The possible further extension may be the formulation of an efficient mapping algorithm for different 3D NoC tile based architectures.

Table 2 Energy of link (in pJ) at different load and link length (in mm)

	Link Length					
Load	1 mm	2 mm	3 mm	4 mm	5 mm	6 mm
0.2	7.65	15.31	22.97	30.63	38.28	45.94
0.4	12.10	24.20	36.30	48.40	60.50	72.60
0.6	16.54	33.08	49.62	66.17	82.71	99.20
0.8	20.98	41.97	62.95	83.94	104.93	125.91
1	25.42	50.085	76.28	101.71	127.14	152.57







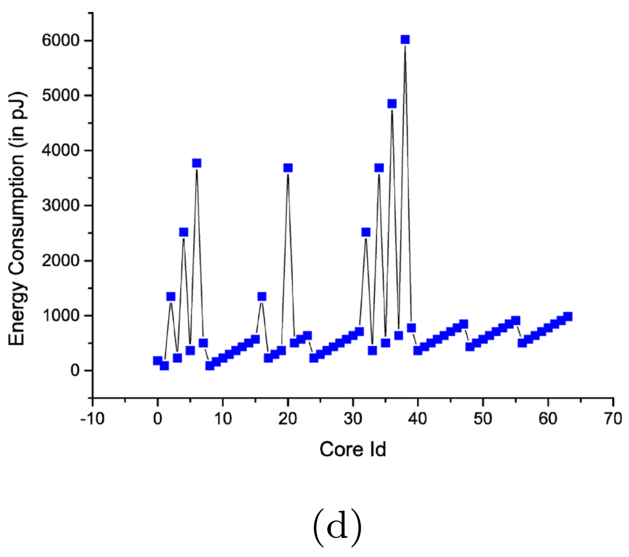


Fig. 21 Energy consumption of tasks in (a) random mapping (b) horological mapping (c) rotational mapping (d) divide and conquer mapping

Table 3 Comparison of average Latency, total queuing time and total service time of mapping algorithms (in ns)

S. No.	Mapping algorithms	Average latency	Total queuing time	Total service time
1	Random	565.31	797.82	411.81
2	Horological	546.78	732.33	252.05
3	Rotational	466.95	602.42	202.11
4	Divide and Conquer	526.78	687.17	241.29

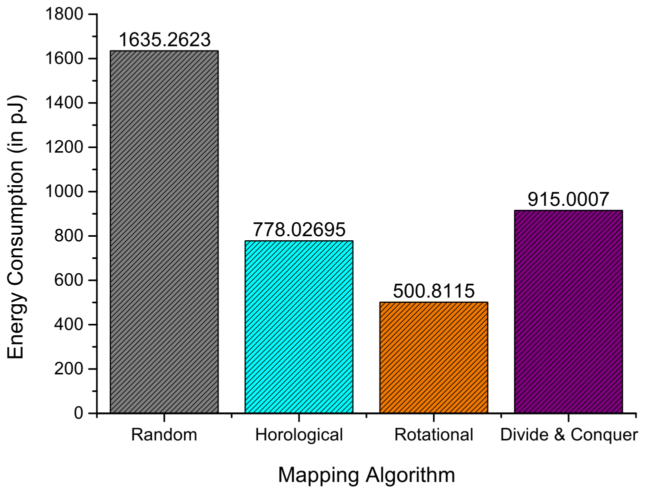


Fig. 22 Comparison of energy consumption (in pJ) of mapping algorithms

References

1. Saleh R, Wilton S, Mirabbasi S, Hu A, Greenstreet M, Lemieux G, Pande PP, Grecu C, Ivanov A (2006) System-on-chip: reuse and integration. Proceedings of the IEEE 94(6):1050-1069
2. Dafali R, Diguet JP, Sevaux M (2008) Key research issues for reconfigurable network-on-chip, international conference on reconfigurable computing and FPGAs (ReConFig) Cancun, pp 181–186
3. Bjerregaard T, Mahadevan S (2006) A survey of research and practices of network-on-Chip. ACM Computing Surveys 1(1):1–51
4. Liu S, Wang S, Liu X, Gandomi AmirH., Daneshmand M, Muhammad K, de Albuquerque VHC (2021) Human memory update strategy: a multi-layer template update mechanism for remote visual monitoring. IEEE Trans Multimed, pp. 1–11
5. Pavlidis VF, Friedman EG (2007) 3-D Topologies for networks-on-chip. IEEE Trans Very Large Scale Integr (VLSI) Syst 15(10):1081–1090
6. Benini L, De Micheli G (2002) Networks on chip: A new SoC paradigm. IEEE Computer 35(1):70–78
7. Marculescu R, Hu J, Ogras UY (2005) Key research problems in NoC design: a holistic perspective. Third IEEE/ACM/IFIP international conference on hardware/software codesign and system synthesis (CODES+ISSS), Jersey City, NJ, USA, pp 69–74
8. Carvalho E, Marcon C, Calazans N, Moraes F (2009) Evaluation of static and dynamic task mapping algorithms in NoC based MPSoCs. International symposium on system-on-chip, tampere, pp 87–90
9. Liu S, Wang S, Liu X, Lin Chin-Teng, Lv Z (2020) Fuzzy detection aided real-time and robust visual tracking under complex environments. IEEE Trans Fuzzy Syst 29(1):90–102
10. Wu N, Mu Y, Ge F (2012) GA-MMAS: an energy- and latency-aware mapping algorithm for 2D network-on-chip. IAENG Int J Comput Sci 39(1)
11. Jang W, Pan DZ (2012) A3MAP: Architecture-Aware Analytic Mapping for Networks-on-Chip. ACM Trans Des Autom Electron Sys 17(3):1 - 22. Article No. 26
12. Tei YZ, Hau YW, Shaikh-Husin N, Marsono MN (2014) Network partitioning domain knowledge multiobjective application mapping for large-scale network-on-chip. Appl Comput Intell Soft Comput 2014:1–11
13. Fen GE, Ning WU (2010) Genetic algorithm based mapping and routing approach for network on chip architectures. Chin J Electron 19(1):91–96
14. Wang X, Yang M, Jiang Y, Liu P (2009) Power-aware mapping for network-on-chip architectures under bandwidth and latency constraints. 4th International conference on embedded and multimedia computing, Jeju, pp. 1–6
15. Murali S, De Micheli G (2004) Bandwidth-constrained mapping of cores onto NoC architectures, design, automation and test in Europe conference and exhibition 2:896–901
16. Oliva D, Esquivel-Torres S, Hinojosa S, Pérez-Cisneros M., Osuna-Enciso V, Ortega-Sánchez N, Dhiman G, Heidari AA (2021) Opposition-based moth swarm algorithm. Expert Sys Appl 184:115481
17. Subramanian S, Sankaralingam C, Dhiman G, Singh H (2021) Hysteretic controlled inter-leaved buck-converter based AC-DC micro-grid system with enhanced response. Materials Today: Proceedings
18. Yang Peng-Fei, Wang Q (2014) Effective task scheduling and IP mapping algorithm for heterogeneous NoC-Based MPSoC. Math Problems Eng 2014:1–8
19. Le Q, Yang G, Hung WNN, Song X, Zhang X (2015) Pareto optimal mapping for tile-based network-on-chip under reliability constraints. Int J Comput Mathematics 92(1):41–58
20. Murali S, Coenen M, Radulescu A, Goossens K, De Micheli G (2006) Mapping and configuration methods for multi-use-case networks on chips. Asia and South Pacific conference on design automation, Yokohama
21. Mehran A, Saeidi S, Khademzadeh A, Afzali-Kusha A (2007) Spiral : a heuristic mapping algorithm for network on chip. IEICE Electronic Express 4(15):478–484
22. Haque AB, Bhushan B, Dhiman G (2021) Conceptualizing smart city applications: Requirements, architecture, security issues, and emerging trends. Expert Systems
23. Kothai G, Poovammal E, Dhiman G, Ramana K, Sharma A, AlZain MA, Gaba GS, Masud M (2021) A new hybrid deep learning algorithm for prediction of wide traffic congestion in smart cities. Wirel Commun Mob Comput 2021
24. Marcon C, Calazans N, Moraes F, Susin A, Reis T, Hessel F (2005) Exploring NoC mapping strategies: an energy and timing aware technique, design automation and test in Europe. 1:502–507
25. Celik C, Bazlamacci CF (2012) Effect of application mapping on network-on-chip performance. 20th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP) Garching, pp 465–472
26. Jiawen W, Li LI, Zhongfeng W, Rong Z, Yuang Z (2014) Energy-efficient mapping for 3D NoC using logistic function based adaptive genetic algorithms. Chin J Electron 23(2):254–262
27. Harmanani HM, Farah R (2008) A method for efficient mapping and reliable routing for NoC architectures with minimum bandwidth and area. Circuits and systems and TAISA conference, Montreal, QC, pp 29–32
28. Houssein EH, Hussain K, Abualigah L, Elaziz MA, Alomoush W, Dhiman G, Djenouri Y, Cuevas E (2021) An improved opposition-based marine predators algorithm for global optimization and multilevel thresholding image segmentation. Knowl-Based Systems: 107348

Affiliations

Arvind Kumar[1] · **Vivek Kumar Sehgal**[1] · **Gaurav Dhiman**[2] · **S. Vimal**[3] · **Ashutosh Sharma**[4] · **Sangoh Park**[5]

Arvind Kumar
er.arvindkumar1989@gmail.com

Vivek Kumar Sehgal
vivekseh@ieee.org

S. Vimal
svimalphd@gmail.com

Ashutosh Sharma
sharmaashutosh1326@gmail.com

Sangoh Park
sopark@cau.ac.kr

1 Department of Computer Science and Engineering, Jaypee University of Information Technology, Waknaghat, India

2 Department of Computer Science, Government Bikram College of Commerce, Patiala, India

3 Department of Computer Science and Engineering, Ramco Institute of Technology, Tamil Nadu, India

4 Southern Federal University, Rostov-on-Don, Russia

5 School of Computer Science, Engineering, Chung-Ang University, Seoul, South Korea