Computer Systems Science & Engineering
DOI:10.32604/csse.2022.019531
Article

Tech Science Press

Novel Power-Aware Optimization Methodology and Efficient Task Scheduling Algorithm

K. Sathis Kumar[1,*] and K. Paramasivam[2]

[1]Department of Computer Science and Engineering, Bannari Amman Institute of Technology, Sathyamangalam, 638401, Tamilnadu, India
[2]Department of Electrical and Electronics Engineering, Kumaraguru College of Technology, Coimbatore, 641049, Tamilnadu, India
*Corresponding Author: K. Sathis Kumar. Email: sathishresearchk@gmail.com
Received: 16 April 2021; Accepted: 12 June 2021

Abstract: The performance of central processing units (CPUs) can be enhanced by integrating multiple cores into a single chip. Cpu performance can be improved by allocating the tasks using intelligent strategy. If Small tasks wait for long time or executes for long time, then CPU consumes more power. Thus, the amount of power consumed by CPUs can be reduced without increasing the frequency. Lines are used to connect cores, which are organized together to form a network called network on chips (NOCs). NOCs are mainly used in the design of processors. However, its performance can still be enhanced by reducing power consumption. The main problem lies with task scheduling, which fully utilizes the network. Here, we propose a novel random fit algorithm for NOCs based on power-aware optimization. In this algorithm, tasks that are under the same application are mapped to the neighborhoods of the same application, whereas tasks belonging to different applications are mapped to the processor cores on the basis of a series of steps. This scheduling process is performed during the run time. Experiment results show that the proposed random fit algorithm reduces the amount of power consumed and increases system performance based on effective scheduling.

Keywords: Random fit algorithm; network on chips; processor cores; power-aware optimization

1 Introduction

An MPSoC can be defined as a system-on-a-chip with multiple processors. Similarly, networks on chips (NOCs) can be defined as a subsystem that has scalable communication and high bandwidth. At present, embedded systems use NOC-based MPSoCs because of its improved performance and energy efficiency. With the rapid development of semiconductor technology in recent years, a single chip is built in a manner that it can integrate multiple transistors. For efficient power consumption [1], the frequency of a processor must be increased. In recent years, hardware manufacturers have started moving toward on-chip systems. The advantage of multiprocessor systems is that system performance is increased without increasing the frequency of the central processing unit (CPU). To establish a connection between on-chip systems, bus-based communication is used. This type of communication is fast, but the requirement for

on-chip size is high. When the communication requirement is high, using bus-based communication can be considered a bottleneck that stalls the CPU, as well as the transmission speed and power consumption. As a result, the performance of the system degrades. This issue can be resolved with the help of NOCs [2–4].

With the advancement of integrated circuit processes, the feature size is greatly reduced, and the memory components, along with the processing cores, can be integrated over a single chip. Traditional bus-based communication is inefficient due to its scalability issues and its inability to handle tasks between the processing cores. Hence, NOCs are used because of its better scalability and seamless transmission of data between the processing cores. Some of the chip's properties, such as latency and power consumption, affect the scheduling results, which are a vital part of NOCs. Obtaining a scheduling scheme that consumes minimal power and has less latency is difficult due to the scheduling problem, which is NP hard, in NOCs.

In a typical NOC architecture, the processor cores are connected with memory blocks and several other processing units [5]. The architecture consists of processor cores that are distributed not via bus connection but via lines [6,7]. On-chip systems use on-chip routers to communicate with each other and achieve high efficiency.

NOCs divide applications into several tasks so that multiple applications can run simultaneously on multiple cores. Task partitioning is essential for multicore systems to improve efficiency [8]. The necessary requirements for NOCs are different traffic characteristics of different applications. System performance is sometimes affected by heavy traffic and long latency. The main portion on the system power is consumed by such traffic [9]. The performance of multiple tasks entails high power consumption and system performance because the tasks are distributed to NOCs, which are essential in reducing power consumption and enhancing system performance to achieve reduced traffic.

The main contributions of this article are as follows:

1. This research proposes a novel approach to map online tasks, which have been proposed to achieve higher efficiency. In accordance with the analysis results, the tasks are remapped by the proposed algorithm.
2. To reduce power consumption, all tasks belonging to a single application are mapped tasks having higher communication.

This paper is organized as follows: Section 2 presents related work. Section 3 describes the system model of NOCs. Section 4 presents the design of the algorithm. Section 5 describes and discusses the experiments and results. Lastly, Section 6 presents future work and conclusions.

2 Related Work

The basic infrastructure for any NOC is the on-chip network. Task requirements must be provided to distribute the network communication process. NOCs provide on-chip networks as the basic infrastructure. Communication is distributed to the network as per the task requirements [10–13]. Compared with the bus structure, other modes of communication consume more time. These types of communication cause performance bottlenecks, which are a major problem to consider. This problem can be solved by properly scheduling the tasks to the cores. Several solutions, including scheduling algorithms based on heuristics [14], genetic algorithms [15], QoS-based algorithms [16], and several others [17], have been provided to address this problem. These algorithms determine the scheduling process before the system operation is determined, thus providing improved optimization and performance. Defining situations at run time to perform is difficult because as the situation changes, the process of scheduling must be repeated, thus consuming a substantial amount of time and becoming difficult.

Online scheduling has been investigated recently. By contrast, run-time scheduling refers to the one performed by the operating system. It assists with the dynamic scheduling of tasks. The author [18] proposed his work based on the characteristics of NOCs. It also deals with the resource allocation process and the immigration of threads at run time. Considered the sequence of tasks being constructed in accordance with the user's habit, which is then used for the prediction and allocation of tasks. The author [19] performed task allocation based on the assumption that each processor core is associated with various levels of power consumption. Thus, the core having the lowest level of power consumption is mapped with the tasks. The author [20] performed scheduling based on the scenarios' transactions. According to run-time analysis, these approaches aim at scaling down the voltage and frequency is such a way that the entire system's power consumption can be reduced.

The tasks in NOCs are associated with certain requirements for communication [21] to achieve this target. Task scheduling determines traffic density. Hence, it is made an overview from the related works that, to estimate the performance of NOC, the scheduling algorithm plays a vital role. Power consumption can be reduced by an optimized scheduling algorithm [22]. The scheduling algorithm [23] proposed by the paper relies on the analysis of the traffic in an on-chip network at run time. The candidate optimization algorithm (COA) produces minimal network transmission delay with a small number of resources consumed [24] and minimal power consumption.

3 System Model

NOCs have various designs [25] and new exciting features, considering the general architecture, application, and algorithm provided. In this section, we take a deep dive on the system model and other related topics, including routing policy, network topology, and energy models. The above mentioned are the base and principal elements of the scheduling algorithm.

3.1 On-Chip Network Topology

NOCs have an unconventional design prototype of SOC, whereas SOCs have advantages over traditional bus communication. On-chip communication provides better performance and has great improvement on efficiency when connected with on-chip devices. Although several topologies, such as ring and mesh [26], have been recommended for NOCs, the mesh is considered to be the best performing choice for the design of NOCs, as depicted in Fig. 1.

The structure of a mesh is similar to that of a matrix. The tiles that are considered nodes are connected by wires, as shown in Fig. 1. Every tile in the mesh includes the following components: routers, processor cores, input/output (I/O) interfaces, and on-chip memory (either a cache or an SPM), as depicted in Fig. 1. Data forwarding and communication between interfaces is taken care of by the routers and I/O interfaces, respectively. Tiles may also contain processing elements (PE) that are used for special purposes and operations. The cache and SPM act as local memory for all cores.

3.2 Routing Policy

Routing policies determine the path to be taken from the initial source nodes to the destination nodes. Problems, such as deadlock and congestions, can be solved; certainly, the performance of an on-chip communication on an NOC can be improved to an extent by a routing algorithm. Routing algorithms generally fall under two categories: deterministic and adaptive routing algorithms.

By using a deterministic routing algorithm, one can determine the path traveled from the initial node to the destination node in a network. By contrast, adaptive routing uses the run-time environments to choose the routing direction. However, this has a great effect on the design of the router and leads to increased complexity.

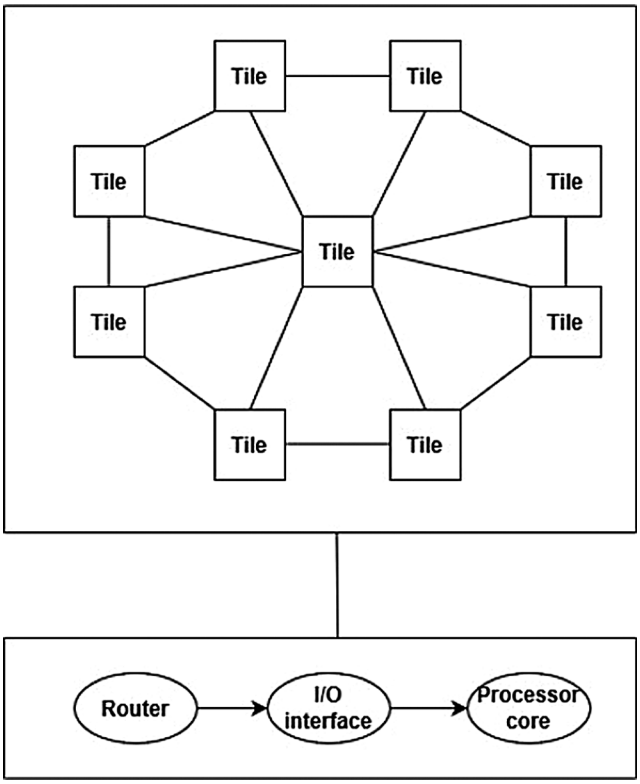

Figure 1: NOC mesh topology

Mesh-based NOCs can be determined with the help of the X–Y routing algorithm. In X–Y routing, the mesh runs in the X and Y directions, as shown in Fig. 2. The initial routing occurs in the X direction where the packets are forwarded, and then the routing proceeds with the Y direction. This prevents the deadlock from occurring for a given destination, in turn depicting high simplicity. In this study, the X–Y routing algorithm is chosen to achieve the least effect on the routing algorithm.

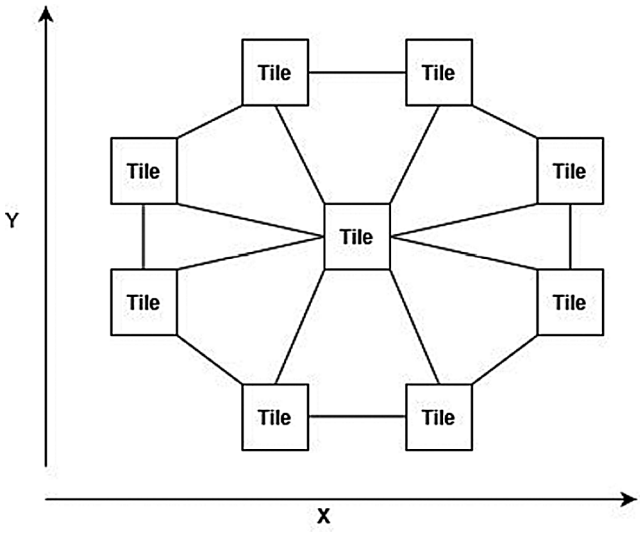

Figure 2: X–Y routing

The important characteristic for transferring on NOC is the switching mode. One of the popular and effective switching modes used now is a packet-based switching technique. Here, the packets are divided into flits, which are considered to be the smaller chunks. These flits are then transferred via a network. This flit can be forwarded and buffered by a router if a single flit has enough buffers. This method is known to reduce network latency and save buffer space. Moreover, if any of the flits are found to be blocked, then the router forwards the other flits by which the network throughput is increased. In this study, the switching mode is wormhole switching. However, wormhole switching has its own disadvantage as it may lead to latency when traffic is heavy. However, these occurrences are rare and have minimal effect on the results. Fig. 1 shows the mesh topology for NOCs.

3.3 Task Model

In case of offline analysis, the application can be represented with the help of an application control graph (ACG) [16]. The application can be represented in the form of vertices V and edges E, where E represents the edge that connects the two vertices. Thus, the representation is of the form $G(V,E)$. Each vertex of the task represents the task V_i. The tasks cannot be subdivided into smaller partitions. The connection between the two vertices V_i and V_j can be represented using the edge notation E_{ij}. The traffic between the two tasks can be represented using the function $F(E_{ij})$. The total traffic off each individual task V_i and V_j can be represented as $F(E_i)$ *and* $F(E_j)$, respectively. The ACG for the given application can be obtained on the basis of the CETA method. By contrast, real-time traffic for the tasks within the application can be obtained using the SIMICS method. The application set is represented as $S = \{S_0, S_1, S_{2,\ldots\ldots}, S_m\}$. Here, each application in the application set has $N(S_i)$ number of tasks. The task model is represented as $M = \{M_0, M_1, M_{2,\ldots\ldots}, M_n\}$.

3.4 Energy Model

Several processor cores or elements are present within the NOC, which establishes communication among the tasks with the help of on-chip lines. These PEs consume energy for computing these tasks. The amount of energy consumed for computing these tasks is referred to as computation energy. The energy consumed in establishing communication between the tasks V_i and V_j can be represented as

$$R(E_{ij}) = R_l(E_{ij}) + R_r(E_{ij}) \tag{1}$$

where R_r represents the amount of energy that the router consumes. Similarly, R_l represents the amount of energy that the lines, which are present between the tasks, consume. Fig. 3. represents the coordinate system for the NOC topology. The coordinates corresponding to the core A1 is represented as A1(0, 0). Thus, the coordinates corresponding to the tile Ai is represented as Ai(X,Y). Here, X and Y represent the horizontal and vertical coordinates, respectively.

Fig. 4 depicts the ACG. In consideration of the tasks in the above ACG, the proposed algorithms are evaluated.

The distance between the cores is measured using the Manhattan distance as follows:

$$Dt(A_{ij}) = |(X_j - X_i) + (Y_j - Y_i)| \tag{2}$$

where E_l represents the energy that the line consumes by connecting the tasks. In other words, it represents the energy that the Manhattan distance consumes. Similarly, E_r represents the energy that the router consumes.

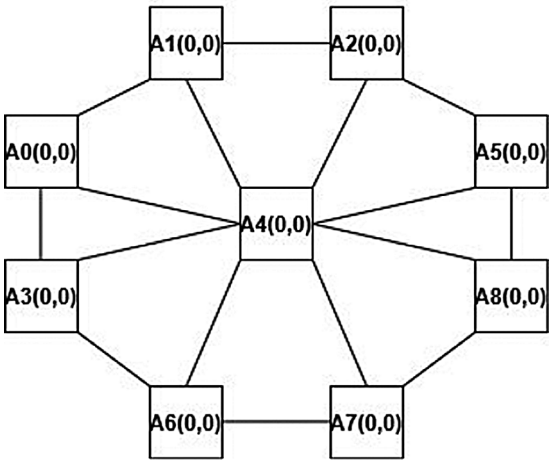

Figure 3: NOC with coordinates

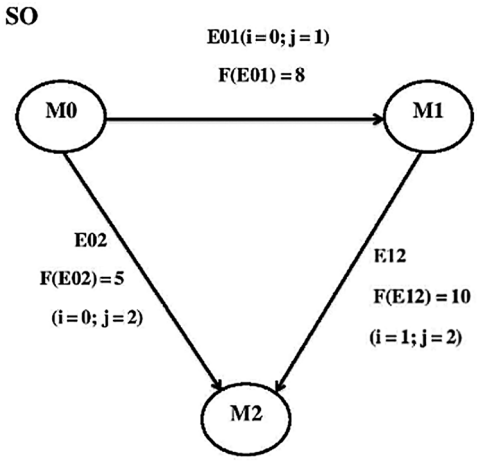

Figure 4: Application control graph

The value for $R_r(E_{ij})$, where R_r is the energy consumed by the router, is calculated.

$$R_r(E_{ij}) = (Dt(A_{ij}) + 1) * R_{rout} \tag{3}$$

Similarly, the value for $R_l(E_{ij})$ is given by

$$R_l(E_{ij}) = (Dt(A_{ij}) + 1) * R_{link} \tag{4}$$

where R_{rout} indicates the energy that the single bit of a router consumes, and R_{link} indicates the energy that the single bit of a line connecting the tasks consumes.

By summing up the above two equations, the total energy consumed can be obtained as follows:

$$R(E_{ij}) = (Dt(A_{ij}) + 1) * R_{rout} + (Dt(A_{ij}) + 1) * R_{link} \tag{5}$$

The total energy consumed by the application is given by

$$R(S_k) = \sum_{\forall A_{ij}} ((Dt(A_{ij}) + 1) * R_{rout} + (Dt(A_{ij}) + 1) * R_{link}) * F(A_{ij}) * N(flit) \quad (6)$$

where S_k represents the set of active applications in the system.

The total energy that the system consumes during all the slots is given by

$$R_s = \sum_{i=0}^{m} R(S_i) \quad (7)$$

The above equations reveal that the Manhattan distance and the amount of energy consumed have a great effect on the total energy consumed by the system. The focus of the study is to reduce the Manhattan distance because the energy consumed is considered a constant.

4 Proposed Model

The algorithm used for scheduling is discussed in this section. In case of single application, the algorithm maps the tasks to its corresponding computational units. Based on this the optimized algorithm can be defined for multiple applications.

4.1 Single-Application Algorithm Design

In case of single application, more than one tasks are partitioned. Compared with single-core chips, multitask applications can provide high parallelism. To boost the performance of the system, these tasks must be assigned to the corresponding cores. In this study, two scheduling algorithms, namely, random fit algorithm and XY routing fit algorithm, are used. In the case of the XY routing fit algorithm, the algorithm searches for the core that is idle in the X direction to be assigned to the unmapped task. If no such core is found in the X direction, then the algorithm looks for the same in the Y direction. The initial scheduling starts with the coordinate A0(0, 0). However, when the communication is larger, the algorithm has its own disadvantage as described below.

In this algorithm, on the basis of the Manhattan distance, the tasks are associated with the cores that are nearby. Thus, the initial task present is mapped to the core A0(0, 0). Similarly, the next task to be mapped to the core is the one having a lower Manhattan distance than the previous core A0(0, 0). In the same manner, all the other remaining tasks are mapped. The traffic caused can be reduced in this algorithm by reducing the Manhattan distance. However, the algorithm has its own disadvantages. Although optimization can be achieved with the help of this algorithm, communication energy remains high. Hence, the Manhattan distance must be reduced even further. To achieve this, the task with the highest traffic must be mapped to the nearby cores. Traffic is defined as follows:

$$Traf(T_i) = \sum_{\forall Eij \in S_k} F(Eij) \quad (8)$$

Thus, the one with the heaviest traffic is mapped to the nearest core A0(0, 0). The next task with the heaviest traffic slightly lesser than the previous one is mapped to the core that has the smallest Manhattan distance with the previously mapped core A0(0, 0). However, if the task with the heaviest traffic is found to communicate with a greater number of tasks, then the Manhattan distance for those tasks increase as per the scheduling algorithm. Assume the number of tasks in S_k is $N(S_k)$. The core with the maximum core value A_m is chosen as the initial point for task scheduling. Each core Ai has a set of chores $A(Ai)$ that are mapped. Thus, the task that has the heaviest traffic is mapped to the core A_m.

To map the remaining tasks with the next core, the tasks must satisfy the following two conditions:

- The core to be mapped must have neighbor relationship with the previously mapped core.
- The next core to be mapped should be the core that has the maximum A_m value.

Similarly, while scheduling the tasks, the task with the highest traffic must be mapped with the previously mapped task. Examples for the scheduling of the applications are shown in the figures below. The total energy consumed by using the three algorithms, namely XY routing algorithm, random fit algorithm, and energy optimization (EO) algorithm, is calculated as follows:

$$EC_{XY} = 140N(flit) * EC(router) + 100N(flit) * EC(link) \tag{9}$$

$$EC_{RF} = 90N(flit) * EC(router) + 80N(flit) * EC(link) \tag{10}$$

$$EC_{EO} = 70N(flit) * EC(router) + 60N(flit) * EC(link)s \tag{11}$$

The above equations show that energy consumption decreases with the decrease in Manhattan distance. As a result, the communication efficiency is highly increased.

Fig. 5 shows the proposed methodology of online scheduling. As mentioned in the diagram, traffic analysis is performed by static profiling. Static profiling marks the start of the online scheduling process. Initially, the EO algorithm is used to schedule the tasks to the cores. The on-chip network is then divided into regions, which are further divided into tasks. Run-time analysis is used for collecting profile information.

4.2 Multiple-Application Algorithm Design

With the number of applications increasing, the number of tasks being mapped to the network also increases. In such cases, the tasks that belong to the same application are gathered as clusters by the EO algorithm. For each application S_i, a task set partition, which is represented as $M(S_i)$, is created.

The network corresponding to the on-chip processor is represented as *N(A,D)*. Here, C represents the processor core, which is represented as $A = \{A_0, A_1, \ldots.., A_i\}$. The total number of chores is represented as N(A), and the set of path is represented by D_{ij}. The path between the two processors is represented as

$$R = |A_i \rightarrow A_j| \tag{12}$$

where R represents the router between the two cores A_i and A_j, and $A(A_i)$ represents the set of processor chores that are connected to the given core A_i.

If $N(A) \geq \sum_{i=0}^{x} N(A_i)$, then the tasks can be assigned, and partitioning is successful. If $N(A) \geq \sum_{i=0}^{x} N(A_i)$, then the tasks cannot be assigned to the given network due to insufficient number of chores. In such cases, partitioning can be performed only if a few tasks are removed from the task list. The task to be removed can be determined as follows:

- Choose a task V_x and remove all the tasks after V_x that are present in the task list M.

  Find an application S_k containing the task M_x.

  Check if M_x is the last task of the application S_k.

- If M_x is the last task, then assign the task to the network.

Each task is mapped to the core with the help of the EO algorithm. During the scheduling process, the mapped cores are not mapped again.

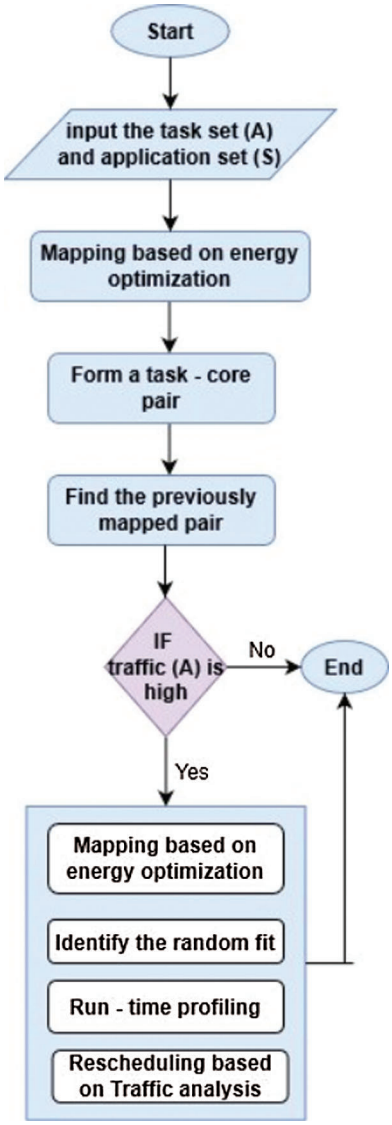

Figure 5: Proposed methodology

4.3 Scheduling Based on Energy Optimization

The communication overhead of the system can only be determined during its runtime. In case of offline scheduling. Fig. 6. the profiling information is inaccurate. However, in case of online scheduling, the communication overhead can be determined during the runtime with the help of profile information. If any changes are found in the traffic, then the tasks are remapped. The proposed algorithm becomes a complex process when the changes that happen to the different circumstances of the application are considered. Static profiling is used to analyze the traffic conditions of the application where the profile information is collected during the runtime of the tasks. The rescheduling process can be performed on the basis of the EO algorithm.

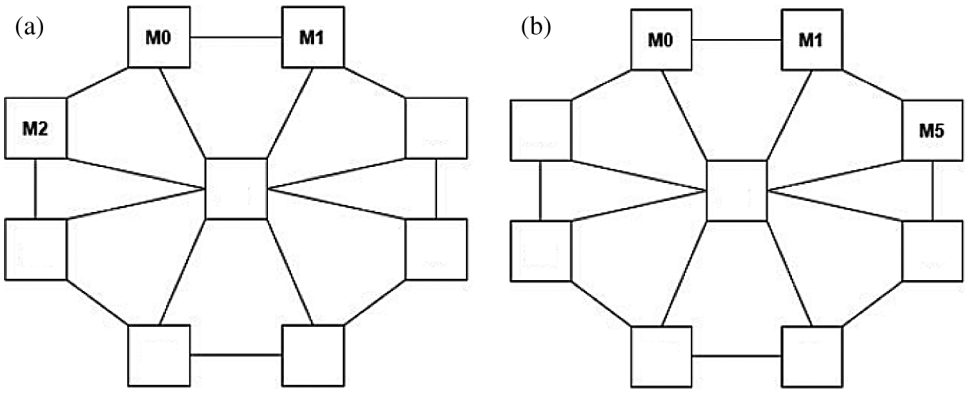

Figure 6: Scheduling results for the application S_0 a) Random fit b) XY routing

Algorithm 1:EO algorithm

Input

Set of applications S: $S_k \in S$

$N(S_k)$ represents the tasks present in S, where S is the application set.

Output

The tasks are mapped to the applications in S.

Working

- Form a task–core pair such that the task is assigned to the corresponding core.
- Identify the last mapped task.
- Identify the task with heaviest traffic when it is mapped with the task mapped previously.
- Identify the corresponding core.
- Remove the core mapped to the task from the core list.
- Repeat the process by scheduling the unmapped cores.

Algorithm 2:Random Fit Algorithm

Input

Set of applications S and processor core A

Output

The tasks are mapped to the applications in S.

Working

A:

for (a = 0;a<=c;a++){

for(b = 0;b<= $N(S_k)$;b++){

Form a task–core scheduling pair such that the task is mapped to that core.

(continued)

Algorithm 2: (continued)

Find the task having highest traffic with previously assigned task.

If (task not found)

Identify the task with the heaviest traffic in the application set S_k.

Identify the task with heaviest traffic when it is mapped with the task mapped previously.

Identify the corresponding core.

Remove the core mapped to the task from the core list.

End A

Do

If (a new task appears)

Identify the traffic between the new task and the previously mapped task.

If (traffic value > threshold assumed)

For(b = 0;b<= $N(S_k)$;b++)

Repeat **A**

}

}

5 Results and Analysis

Simulation is used to test the proposed algorithm. The traffic can be obtained on the basis of the SIMICS method. SIMICS settings include

- CPU number: 16,
- frequency: 60 GHz,
- size of the cache: 32 KB,
- size of the disk: 4 GB,
- memory size: 1024 MB, and
- operating system used: Linux version 2.6.

The NOC simulator used for the simulation process is Gem 5. Its parameter settings include

- data cache size: 64 KB,
- instruction cache size: 64 KB,
- data cache associativity: 16 way,
- instruction cache associativity: 8 way,
- hit latency: 4–35 cycles, and
- memory size: 2 GB.

The traffic of some benchmark applications, such as FFM_3 and MPGenC, are obtained and used for performing analysis via CETA. The applications of the ACG are obtained with the help of CETA. The tasks present within the application can be obtained with the help of SIMICS. The on-chip network can be simulated with the help of the GEM 5 simulator. In this process, the communication overhead that occurs due to task migration is considered. Results show that the complexity of the EO algorithm is lower.

The above graph reveals that the EO algorithm outperforms the XY routing and random fit algorithms in terms of energy consumption. This result is due to the balanced communication that exists among the tasks of an application. The number of flits transmitted can be reduced with the help of the EO and random fit algorithms. On the basis of the communication relationship, the tasks are assigned. Approximately 62% of the total flits can be reduced by using the proposed algorithm. This saves as the base for the optimization algorithm.

Fig. 7 shows that as the traffic is reduced, the amount of energy consumed can also be reduced. As the traffic remains average, the amount of energy consumed remains consistent. The maximum latency of the various algorithms used is shown in the above figure, which reveals that although the number of flits is reduced, the latency remains maximum. Mapping the tasks on the basis of the Manhattan distance alone is difficult if they do not have intensive communication. Although the random fit algorithm does not provide effective communication compared with the EO algorithm, the tasks are established to have a balanced communication. However, the rescheduling process does not reduce the number of transmitted slits where the rescheduling process is performed during the runtime based on the profiling information. The online scheduling process is more flexible for the operations of the system. The online scheduling process involves the information to be gathered to remap the tasks that involve extra time loss. Fig. 8. shows that the maximum latency is difficult to reduce despite the reduction in the total number of slits.

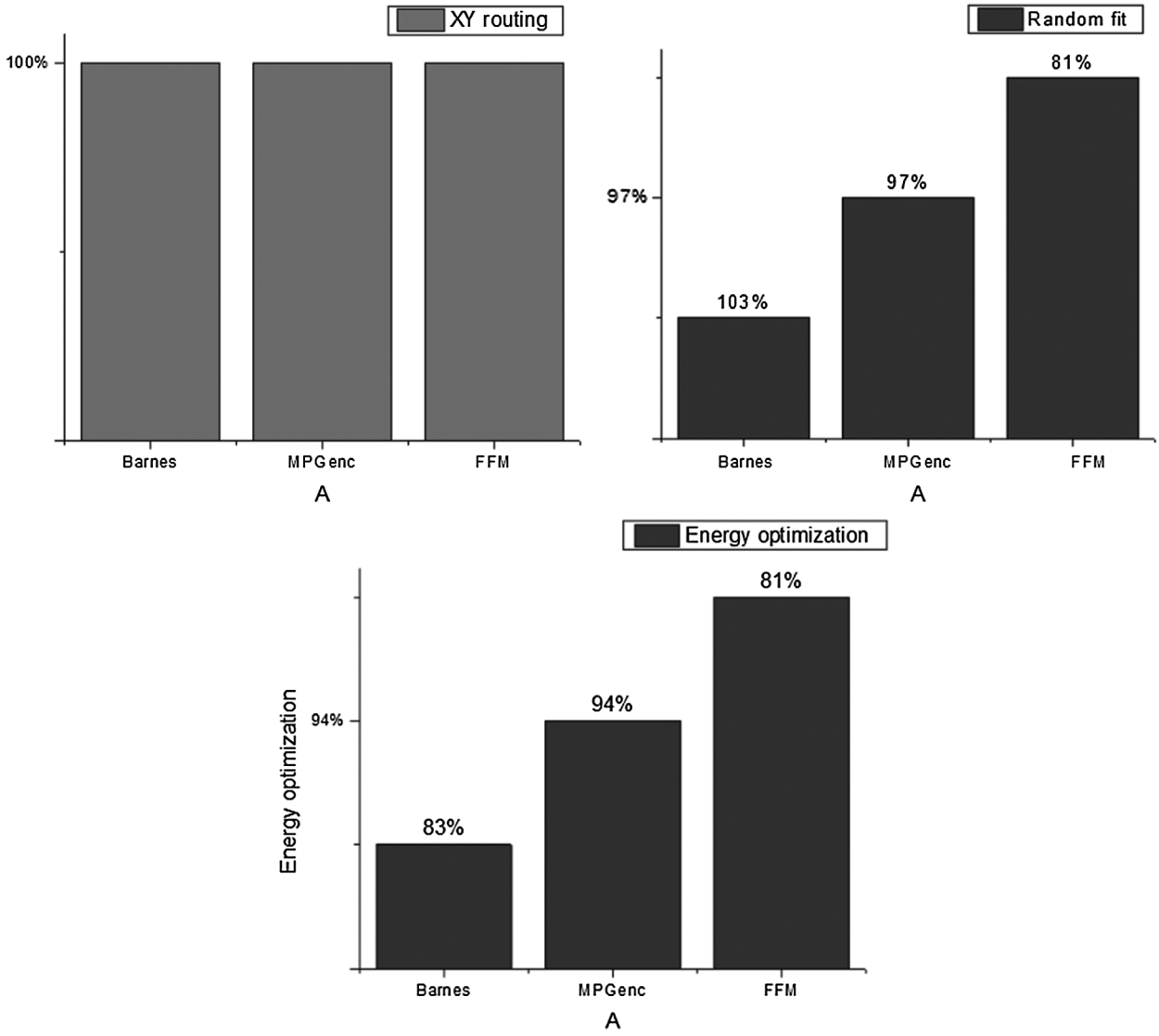

Figure 7: Total energy consumed

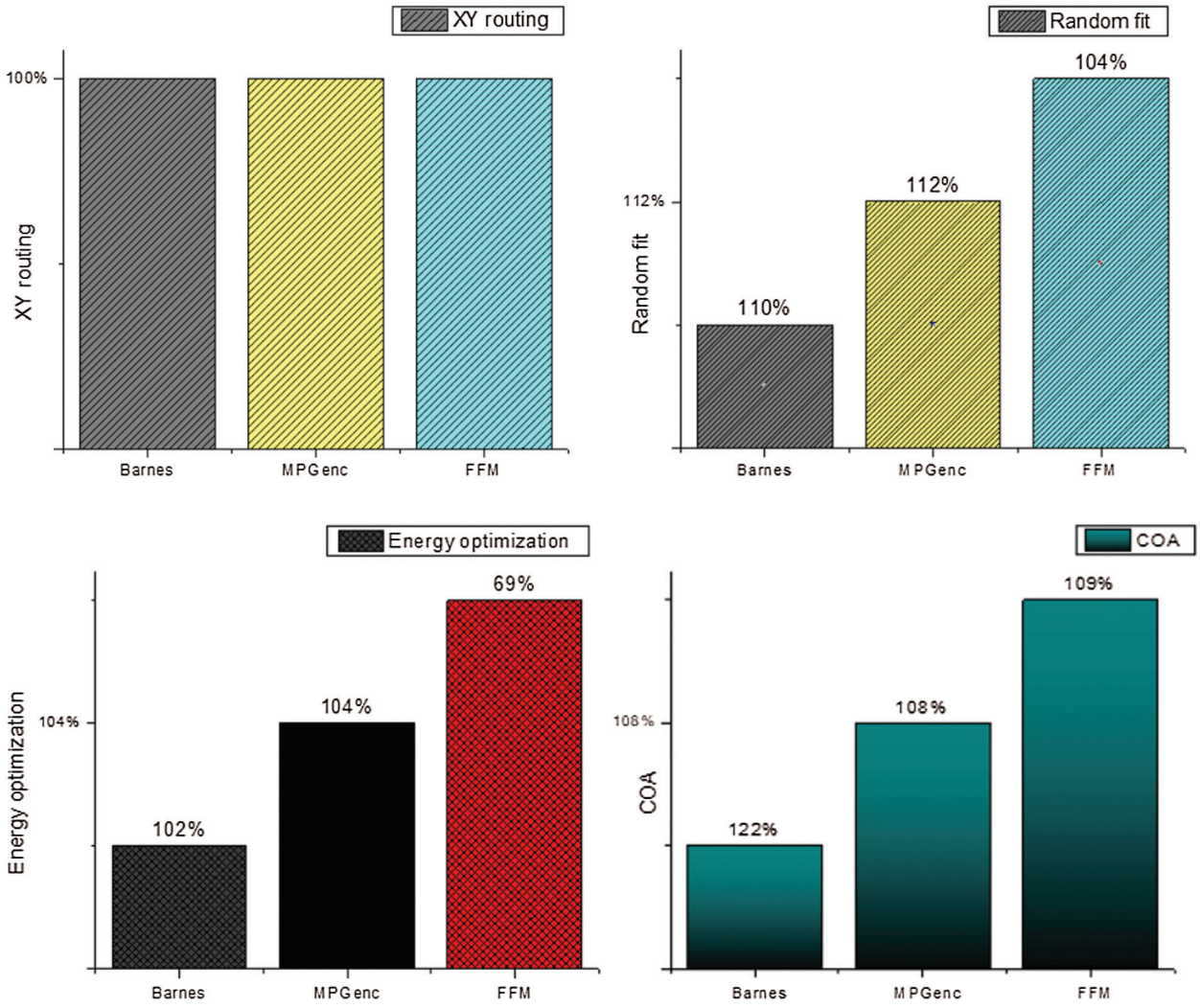

Figure 8: Maximum latency graph

The average latency of the three mentioned algorithms is shown in the above graph diagram. If the communication is intensive, then the task efficiency is a guaranteed one. In the case of the random fit and EO algorithms, the average latency is reduced to approximately 29%–33%.

Thus, the experimental result shows Fig. 9. that the random fit algorithm can outperform the COA by increasing the task scheduling efficiency for an NOC-based multicore system. Thus, even without considering the traffic, the tasks can be assigned to its corresponding cores with the help of this algorithm. Despite the overhead, it has high performance and reduces power consumption compared with COA.

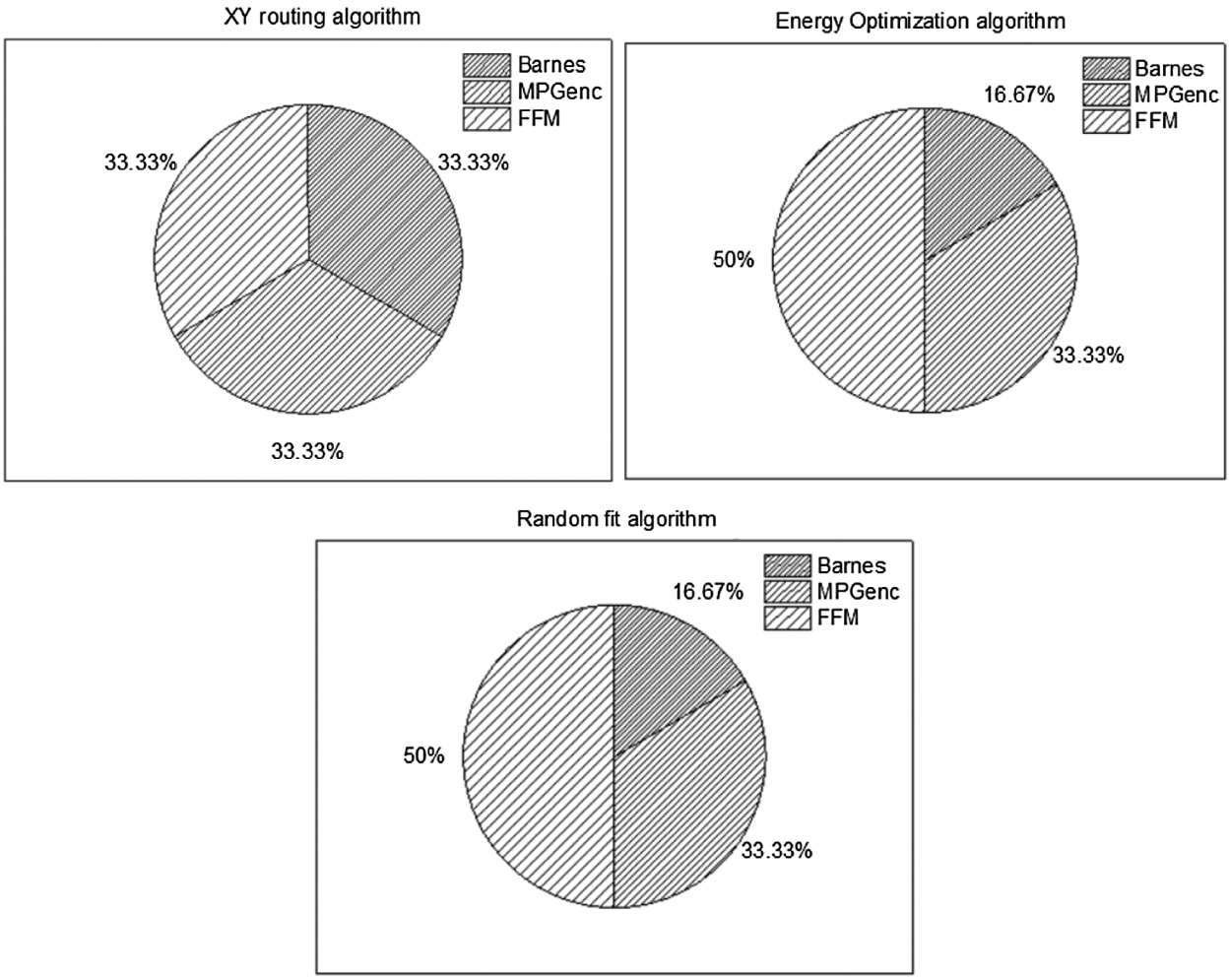

Figure 9: Average latency of the three different algorithms

6 Conclusions

A promising paradigm to break the bottleneck problem in on-chip communication is NOCs. Here, the cores are connected to form a core network. However, the greatest challenge faced by NOCs in terms of performance and energy consumption is network traffic. The network of on-chip cores produces a remarkable effect on the basis of the network characteristics. The key issue in scheduling lies in the scheduling of the tasks to the associated network. In this study, an optimized scheduling algorithm called random fit algorithm is used for NOC-based multicore systems. The communication energy in the scheduling process can be saved with the help of the EO algorithm. The tasks can be remapped to ensure improved performance and power consumption. Experimental analysis shows that the performance can be increased with the help of the rescheduling process. However, the proposed scheduling algorithm does not consider migration cost. In the future, the scheduling algorithm can be reframed by considering migration cost.

Funding Statement: The authors received no specific funding for this study.

Conflicts of Interest: The authors declare that they have no conflicts of interest to report regarding the present study.

References

[1] J. Xu, W. Wolf, J. Henkel and S. Chakradhar, "A design methodology for application-specific networks-on-chip," *ACM Transactions on Embedded Computing Systems*, vol. 5, no. 2, pp. 263–280, 2016.

[2] T. Bjerregaard and S. Mahadevan, "A survey of research and practices of network-on-chip," *ACM Computing Survey*, vol. 38, no. 1, pp. 1–51, 2016.

[3] T. Mak, P. Y. Cheung, W. Luk and K. Lam, "A DP-network for optimal dynamic routing in network-on-chip," in *Proc. CODES ISSS*, Playa del Carmen, Mexico, pp. 119–128, 2019.

[4] K. Chang, J. Shen and T. Chen, "Tailoring circuit-switched network-on-chip to application-specific system-on-chip by two optimization schemes," *ACM Transactions on Design Automation of Electronic Systems*, vol. 13, no. 1, pp. 1–31, 2018.

[5] G. Chen, F. Li, S. W. Son and M. Kandemir, "Application scheduling for chip multiprocessors," in *Proc. DAC*, San Francisco, California, USA, pp. 620–625, 2018.

[6] D. Barcelos, E. W. Brião and F. R. Wagner, "A hybrid memory organization to enhance task migration and dynamic task allocation in NoC-based MPSoCs," in *Proc. SBCCI*, New York, USA, pp. 282–287, 2017.

[7] V. Nollet, T. Marescaux, D. Verkest, J. Mignolet and S. Vernalde, "Operating-system controlled network on chip," in *Proc. DAC*, San Francisco, USA, pp. 256–259, 2014.

[8] C. Chou and R. Marculescu, "User-aware dynamic task allocation in networks-on-chip," in *Proc.DATE*, Munich, Germany, pp. 1232–1237, 2018.

[9] J. Hu and R. Marculescu, "Energy-aware communication and task scheduling for network-on-chip architectures under realtime constraints," in *Proc.DATE*, Paris, France, pp. 234–239, 2014.

[10] E. W. Briao, D. Barcelos, F. Wronski and F. R. Wagner, "Impact of task migration in NoC-based MPSoCs for soft realtime applications," in *Proc. VLSI-SoC*, Atlanta, GA, USA, pp. 296–299, 2017.

[11] S. Bertozzi, A. Acquaviva, D. Bertozzi and A. Poggiali, "Supporting task migration in multiprocessor systems-on-chip: a feasibility study," in *Proc. DATE*, Munich, Germany, pp. 15–20, 2016.

[12] C. Chou and R. Marculescu, "Incremental run-time application scheduling for homogeneous NoCs with multiple voltage levels," in *Proc. CODES+ISSS*, Salzburg, Austria, pp. 161–166, 2017.

[13] H. Wang, L. Peh and S. Malik, "A technology-aware and energy-oriented topology exploration for on-chip networks," in *Proc. DATE*, Munich, Germany, vol. 2, pp. 1238–1243, 2015.

[14] J. Wu, "A deterministic fault-tolerant and deadlock-free routing protocol in 2-D meshes based on odd-even turn model," in *Proc. ICS*, New York, USA, pp. 67–76, 2012.

[15] S. Taktak, J. Desbarbieux and E. Encrenaz, "A tool for automatic detection of deadlock in wormhole networks on chip," *ACM Transactions on Design Automation of Electronic Systems*, vol. 13, no. 1, pp. 1–22, 2018.

[16] M. M. Pastrnak, P. H. N. de, S. Stuijk and J. V. Meerbergen, "Parallel implementation of arbitrary-shaped MPEG-4 decoder for multiprocessor systems," in *Proc. PWSM*, San Jose, California, United States, pp. 1–10, 2006.

[17] A. H. Liu and R. P. Dick, "Automatic run-time extraction of communication graphs from multithreaded applications," in *Proc. ODES+ISSS*, New York, NY, United States, pp. 46–51, 2016.

[18] T. T. Ye, G. D. Micheli and L. Benini, "Analysis of power consumption on switch fabrics in network routers," in *Proc. DAC*, New Orleans, LA, USA, pp. 524–529, 2012.

[19] T. Bjerregaard, "A survey of research and practices of network on-chip," *ACM Computing Surveys*, vol. 3, no. 38, pp. 1–51, 2016.

[20] N. Concer, S. Iamundo and L. Bononi, "A equalized: A novel routing algorithm for the spidergon network on chip," in *Proc. DATE*, Europe, pp. 749–754, 2012.

[21] F. Karim, A. Nguyen and S. Dey, "An interconnect architecture for networking systems on chips," IEEE *Micro*, vol. 22, no.5, pp. 36–45, 2002.

[22] M. Moadeli, A. Sharabi, W. Vanderbauwhede and M. OuldKhaoua, "An analytical performance model for the spidergon NoC," in *Proc. AINA*, Ontario, Canada, pp. 1014–1021, 2017.

[23] F. Karim, M. Root and C. Mesh, "Interconnection architectures for network-on-chip systems," *World Academy of Science, Engineering and Technology*, vol. 54, pp. 137–144, 2019.

[24] M. Mirza Aghatabar, S. Koohi, S. Hessabi and M. Pedram, "An empirical investigation of mesh and torus NoC topologies under different routing algorithms and traffic models," in *Proc. DSD*, Lubeck, Germany, 2007, pp. 19–26, 2017.

[25] P. P. Pande, C. Grecu, A. Ivanov and R. Saleh, "Design of a switch for network on chip applications," in *Proc. ISCAS*, Bangkok, Thailand, pp. 217–220, 2003.

[26] V. Nollet, P. Avasare, H. Eeckhaut, D. Verkest and H. Corporaal, "Run-time management of a MPSoC containing FPGA fabric tiles," *IEEE Transactions on Very Large-Scale Integration (VLSI) Systems*, vol. 16, no. 1, pp. 24–33, 2008.