Volume 66, Issue 2, 2022
Journal of Scientific Research
of
The Banaras Hindu University

Improving system performance in Homogeneous Multicore Systems

Savita Gautam[1], Abdus Samad[2] and M. Sarosh Umar[3]

[1] University Women's Polytechnic
[2] Z. H. College of Engg. & Technology
[3] Aligarh Muslim University, Aligarh, India
[1]savvin2003@yahoo.co.in
[2]abdussamad@zhcet.ac.in
[3]saroshumar@zhcet.ac.in

Abstract. Allocation of parallel load in multicore systems has become a challenging task for high performance computing system. There are several parameters to evaluate the performance of a scheduling algorithm such as task imbalance and execution time. This paper proposes a task scheduling approach that targets multiple cores connected through appropriate interconnection network. The proposed approach utilizes the computing resources effectively by assigning the tasks dynamically among different cores of the system in a realistic time. Each task has its own timeline and multiple sequence of tasks are mapped on different cores of the system. In particular, performance is evaluated on n x n Mesh, DMesh, ZMesh and Torus networks. The load imbalance and execution times are considered as metrics to evaluate the performance of the proposed algorithm. Simulation results are obtained and compared with well-known minimum distance scheduling algorithm which shows reduction in execution time while maintaining the load imbalance. An improvement of 20-30% is obtained in load imbalance for considered multicore systems with improved execution time. The simulation study reveals that the proposed algorithm is best suited to take architectural benefits for mesh-based multicore systems.

Keywords: Multicore, Scheduling Algorithm, Load Imbalance, Execution Time.

1 Introduction

Multicore systems are found in variety of computing systems from high-performance servers to special purpose embedded systems. The industrial applications are utilizing embedded systems with more cores in processors [1]. The performance of these systems depends upon how extensively the parallelism is exploited among different cores in the system. In order to address the problem of parallelism in a multicore system, the load is partitioned into small independent tasks and are mapped onto different available cores in the systems. The problem of efficient allocation of a group of tasks to carry out parallel execution in multicore systems has drawn attention of researchers.

Designing an efficient communication network and applying efficient scheduling algorithm for utilizing computing resources is critical for achieving high performance in multiprocessor multicore systems. There is a number of studies related to simulation on mesh-based topologies that emphasizes on modeling and analyzing on chip interconnect [2] [3] [4]. Metrics such as packet delay, load imbalance factor have been used as a function of the communication load, speedup and utilization factor. Some networks are designed specifically with customized application in order to achieve better performance. The main objective behind customization is to fit the requirements of specific applications under certain conditions [5]. However, a generalized task-based programming model is inevitable solution for multicore architectures.

In this paper, we explore the interplay between architectures and algorithm design in the context of dynamic task allocation. A dynamic scheduling algorithm is designed and evaluated by mapping tasks on a number of mesh-based multicore architectures. The proposed approach is based on standard minimum distance scheduling approach that has been used extensively for conventional parallel systems in a variety of ways [6]. For better analysis of results different data sets are applied to similar architectures for the performance evaluation of the proposed algorithm.

The rest of the paper is organized as follows. In section 2, various approaches related with scheduling of tasks on Homogeneous/heterogeneous multicore system are presented. Section 3 describes the problem formation and the target systems considered for study. The proposed algorithm is explained in section 4. Based on the experimental results, the performance evaluation is carried out and presented in section 5. Concluding remarks are presented in section 6.

2 Related Work

A programming model schedule tasks dynamically according to the availability of computing resources. Mapping of ready to execute tasks to different cores of the system requires critically task aware schedular [7]. The efficient scheduling problem has been extensively studied for asymmetric multicore systems. Some of them are based on dividing the tasks into groups of critical and non-critical tasks and mapping each group to one core type. In this method deciding which task is critical is a major issue [8]. Task prioritization is another

DOI: 10.37398/JSR.2022.660207

approach which assigns priority to different tasks based on information discoverable at run-time [9].

A number of programming models have been developed for high-performance computing such as task parallelism [10], data parallelism for example OpenMp loops [11] to exploit parallelism in multicore systems architectures. These models support both inter-task parallelism as well as intra-task parallelism. In general, the sequence of tasks is mapped as a group of parallel sub tasks that are allowed to execute in parallel on multiple cores. The directed acyclic graph (DAG) is one of the most famous parallel task models used in multicore architectures [12]. A DAG consists of directed edges between a set of nodes in which each node is a sequential sub-task that are allowed to execute on any core using directed edges. Subtasks are allowed to execute on different cores that can significantly improve resource utilization. On a multicore system meeting deadlines of parallel tasks is more complex due to possible interleaving of threads across the cores. Therefore, to incorporate full speed up there is a great challenge to maximize the utilization of parallel multicore architectures which meet the deadlines of application cores. List scheduling has been used in variety of ways to obtain optimal/sub-optimal solutions [13]. List scheduling is designed on the basis of assigning priorities to the tasks of DAG and arranging the tasks in the form of list which are configured in descending order of priorities. Task having higher priority is allowed to execute first. The algorithm performs better with small heterogeneity factor for randomly generated applications. However, to reduce task execution time a duplication approach to identify heavily communicating tasks is applied.

In heterogeneous computing system the cost of executing a task may vary from one core to another. The priority of tasks is not fixed rather change when migrated between different cores. To handle this problem, Heterogenous Earliest Finish Time Schedular (HEFT) [14] and Heterogeneity through Limited Duplicated [HLD] approach [15] are used in order to get a single computation cost of a task. However, performance of these algorithms is limited with the significant variations in the execution makespan.

System performance can also be improved by non-contiguous allocation of parallel jobs in multicomputer systems [16]. In this approach the author claimed better performance in terms of execution time for different traffic pattern particularly with uniform-decreasing job size distribution. The algorithm, however, is not tested for Torus type architecture.

3 Problem Formation and Target Systems

3.1 Task Scheduling Model

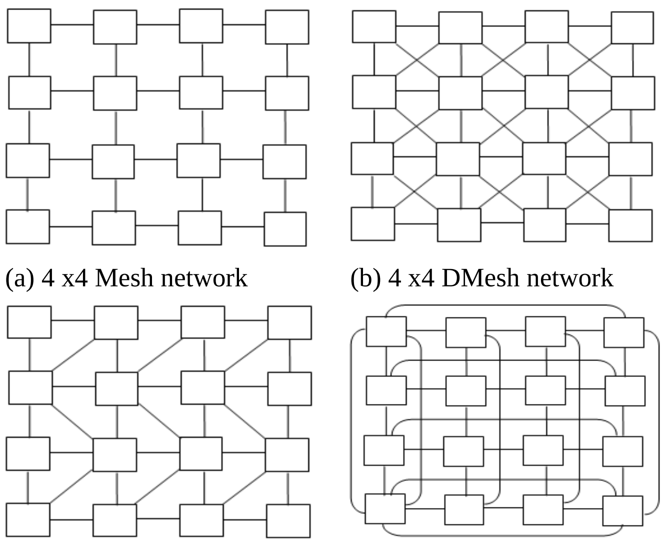
The task scheduling problem has been widely studied for both homogeneous and heterogeneous multicore systems. The implementation of these algorithms performs action on the state of tasks depending upon the architecture of the target system. The main objective is to map the ready tasks onto available cores until all the ready tasks are assigned evenly. Task dependency is another factor that effect the performance of the scheduling policy. However, for simplicity we considered all tasks as independent tasks. Tasks are submitted uniformly and assigned to a particular core depending upon the active load or when the core becomes idle. At a particular point of time the system manages a uniform distribution of tasks. The resource utilization and uniform allocation of tasks are carried out dynamically in parallel among different available cores of the system. If tasks in an application are unbalanced, the overloaded and underloaded cores are identified and tasks migration take place until the system obtain an even distribution of tasks. Therefore, in application of wide range graph such as Zmesh and higher-level mesh having large number of cores or with large volume of tasks the task schedular reconfigures the tasks dynamically based on the value of ideal load and load imbalance factor (LIF).

The minimum distance scheduling (MDS) is considered suitable for parallel interconnection networks in traditional parallel systems [17]. The algorithm relies on minimum distance property in which only adjacent cores are allowed to migrate the tasks. This is followed in order to reduce makespan and complexity of scheduling algorithm. Several variations of MDS have been proposed and found suitable for a particular class of architectures. The performance of these algorithms has not been studied for multicore systems. The proposed algorithm is an effort to extend the concept of minimum distance property with some alteration and tested for considered multicore systems.

3.2 The Target Architectures

To evaluate the performance of proposed scheduling algorithm the topology of target system is a modeled un-directed graph G (C_i, E_i) where C is a finite set of cores/vertices and E is a finite set connected edges. A vertex C_i represents the processor core i and E_i represents a bidirectional communication link between adjacent cores. The resource graph is a complete graph consisting of n fully connected cores. We assume contention free communication between cores.

For the purpose of simulation four similar topologies namely Mesh, Dmesh, Zmesh and Torus networks are considered [18]. The system consists of a set of homogeneous cores and all considered topologies are modeled as 4 x 4 networks shown in Fig. 1. Task-to-core assignment is identical in all the considered topologies.


(a) 4 x4 Mesh network (b) 4 x4 DMesh network

(c) 4 x4 ZMesh network (d) 4 x4 Torus network

Fig.1. Target Systems

4 Proposed Algorithm

As discussed in section 3 we propose a dynamic task scheduling algorithm that detects the load imbalance among different available cores and map the tasks accordingly during runtime. Among different models the tasks are first created and then made ready after certain level of input. In the proposed approach we assume that the ready tasks are available and at a given point of time tasks are assigned to different cores based on the scheduling policy. Overloaded cores receive the tasks from underloaded cores based on the value of ideal load and LIF.

The load Imbalance Factor (LIF) at a particular stage of task (k) structure is calculated as.

$$LIF = [\max\{loadk(Ci)\}-(ideal\_load)k] / (ideal\_load)k \qquad (1)$$

The ideal load is calculated by the ratio of the total number of tasks and the number of available cores (N).

$$(ideal\_load)i =[loadk(C_0)+loadk(C_1)+\ldots+loadk(C_{N-1})] / N \qquad (2)$$

Maximum load denoted as $\max(loadk(C_i))$ is the value of maximum load on a particular core Ci ,where, Ci ,$0 \leq i \leq N-1$. For the same stage of task structure, the execution time is evaluated which is the total time the schedule algorithm takes to produce LIF after the balancing process is complete.

However, task migration is allowed only after examine the connectivity of cores. Selecting the communicating core directly affects the complexity of algorithm and leads larger execution time. The five steps of the proposed algorithms are as follows.

1. A valid taskset is generated to map on available number of cores connected through bi-directional links.
2. The adjacency matrix is scanned to examine the direct connectivity of cores.
3. The connected cores are identified and tasks are assigned from one core to another until the value reaches to ideal load.
4. The LIF is evaluated and allocation of tasks is continued. To maintain the integrity of MDS only directly connected cores are allowed to migrate the tasks.
5. For optimum results step 4 is repeated for non-adjacent cores to migrate the tasks between overloaded or underloaded cores.

The outline of the given algorithm is illustrated in Fig. 2. It is clearly shown that allocation of tasks always succeeds if the underloaded cores with direct connectivity exist. The proposed algorithm and the well-known MDS algorithm were implemented in Java with Windows 10 on 2.60 GHz Intel(R) Core (TM) i7 x64 base processor and 16.0 GB of RAM. Many different graphs were drawn by varying the task structure for input into the proposed algorithm and discussed in the next section.

```
Void TaskMigration (int overloaded_Nodes, int under-
loaded_Nodes)
{
    Int p=0, Idealload, q=0,Max= underloaded_Nodes;
    for(p=0;p<=overloaded_Nodes;p++)
      {
        While(value[p]>Idealload)
       {
          for(q=0;q<=underloaded_Nodes;q++)
            shift(Task[p],Task[p][q]);
                }}
return   overloaded_Nodes;
}
int TaskAllocation(int n)
{  int totaltask,i,n;
   Generate_Random_Task(n);
for(i=1;i<=n;i++)
  totalTask+=Task(n);
return Totaltask;
}

Int maximu(int nodes)
{
  int i;
   for(i=0;i<nodes;i++)
     compare(max_value,node_value);
     return(max_value);
}

float  lif(int max, int idealload)
{
load_imbalance=(float)(max-idealload)*100/idealload;
return load_imbalance;
}
```

Fig. 2. Pseudo-Code for task allocation and migration

5 Simulation Experiment Results

In this section, we evaluate the performance of the proposed algorithm by carrying out experiments on different multicore architectures in a wide spectrum of input types. We have measured load imbalance and execution time for three sets of task structures. To show how well the proposed algorithm is contributed the results obtained are compared with standard minimum distance scheduling (MDS) algorithm in terms of LIF and execution time by implementing both the algorithms on same architectures under same environment. Fig. 3 shows the performance of proposed algorithm by comparing against the MDS algorithm when applied on 16-cores mesh network.

The results show that initial value of LIF in case of MDS algorithm is much larger than the value obtained by implementing the proposed algorithm. The best-case performance for average load is improved by 20% throughout the generation of tasks. The changing behavior of LIF, however, similar for both the scheduling algorithms.

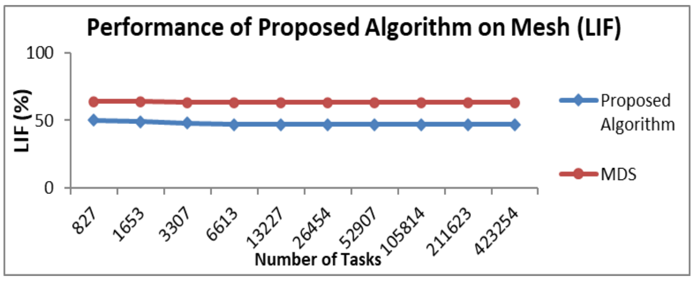


Fig. 3. Performance of proposed scheduling on 16-core mesh network

Another important parameter to evaluate the performance is execution time. The total time to make the network fully balanced after generation a finite number of tasks is evaluated and shown in Fig. 4. The results of the effect of enhancing task migrations by considering non-adjacent cores in the proposed algorithm are undoubtedly depicted in Fig. 4 which shows an increasing trend as compared to when MDS is applied on the same network. This is due to the fact that the proportion of task migration on cores other than adjacent cores increases to obtain the desirable value of LIF. If we consider conventionally acceptable value of average LIF between 30-40%, then the increase in execution time will be insignificant.

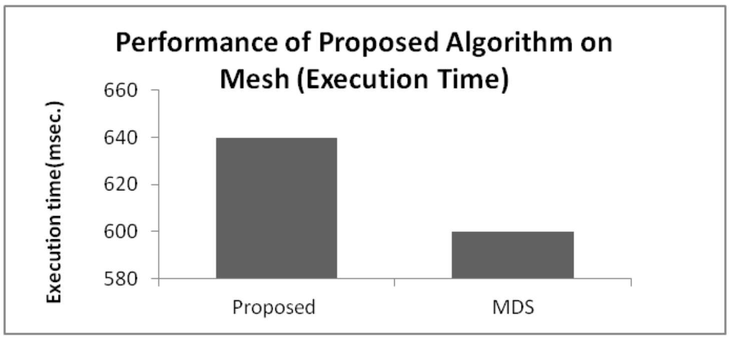


Fig. 4. Performance of proposed scheduling on 16-core mesh network

Motivated form the results obtained for 4 x 4 mesh network and to test the actual performance of the proposed algorithm the same is also applied on 16-cores Dmesh, Zmesh and Torus networks with three data sets. Each data set consists of finite range of task structure. The first set of experiment is carried out with data set having tasks ranging from 1000 to 4,00,000 tasks, second data set covers tasks up to 16,00,000 and the third data set may go up to 50,00,000. The simulation results obtained for all four considered networks using LIF and execution time as metrices are shown in the form of graphs and are presented in Fig. 5 to Fig. 7.

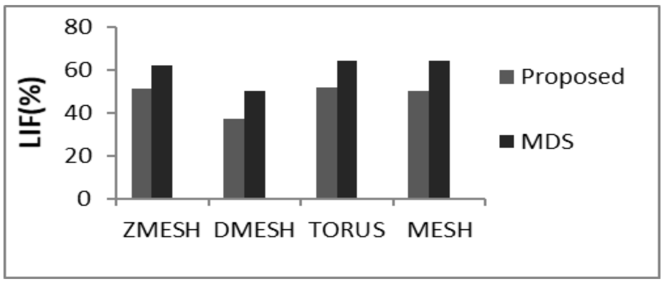


Fig. 5 (a). Performance of proposed algorithm with low task structure (LIF)

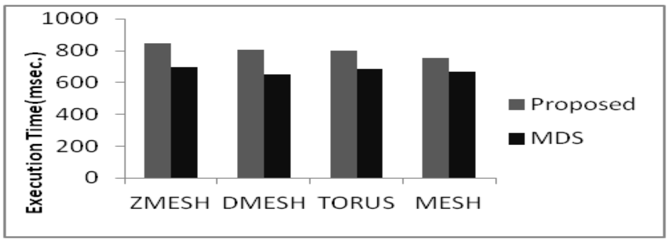


Fig. 5 (b). Performance of proposed algorithm with low task structure (Execution Time)

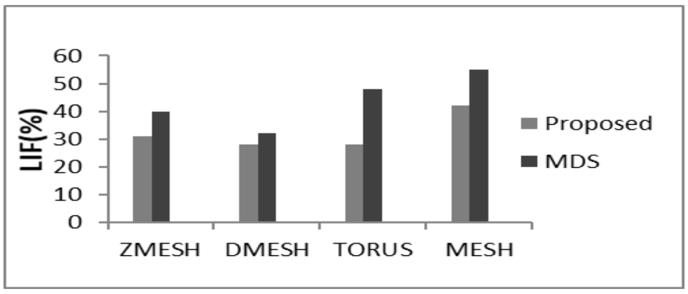


Fig. 6 (a). Performance of proposed algorithm with medium task structure (LIF)

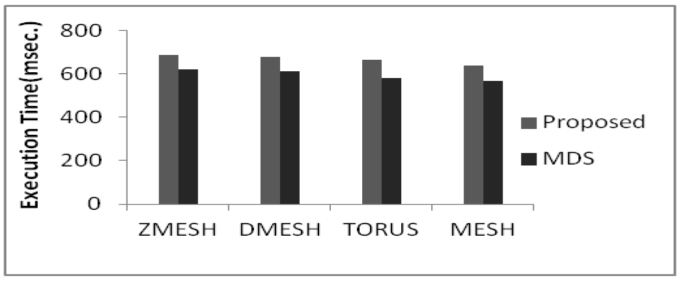


Fig. 6 (b). Performance of proposed algorithm with low medium structure (Exec. Time)

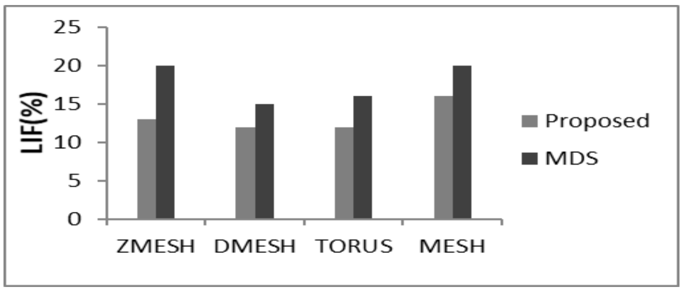


Fig. 7 (a). Performance of proposed algorithm with high task structure (LIF)

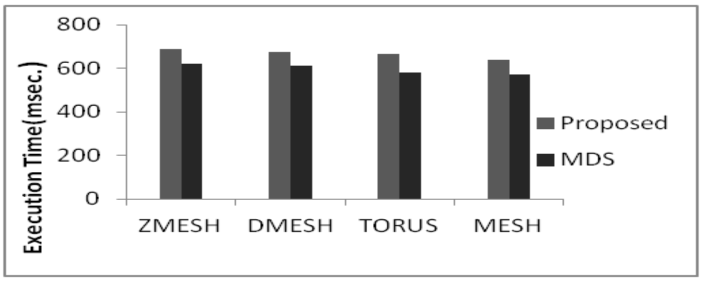


Fig. 7 (b). Performance of proposed algorithm with high task structure (Exec. Time)

In all the graphs shown from Fig. 5 to Fig. 7, it is clearly observed that with the proposed algorithm the initial value of LIF is improved for all the considered topologies. In particular, there is an improvement of approximately 20% for Dmesh and Torus networks. It is because that Dmesh and Torus both are having extra links that constitutes alternative path for task migration. Due to this reason, the execution time also has no significant increment for these networks. Performance with other networks is also comparable. Figures also show that the performance of proposed algorithm is approximately same for different types of task structures. This is because most of the tasks are having similar execution times. The schedular dynamically select the best available path when the application has a large number of tasks.

In Fig. 5(b), Fig. 6(b) and Fig. 7(b), the execution times for minimum value of LIF obtained by the proposed algorithm are plotted for all the considered networks with different volume of system loads. The results reveal that there is small increment in execution times with the proposed algorithm, however, the performance is similar for all the considered networks. This is due to the fact that we considered networks in which each core is connected by bidirectional communicating links to its neighbor cores, as depicted in Fig.1. The minor increment in execution time is tolerable with highly reduced value of LIF which ultimately improve the system utilization. The main attraction of the proposed algorithm is the independent load which does not have impact on the efficacy of selected architecture.

6 Conclusion and Future Work

In this paper, we have incorporated an enhancement to the minimum distance scheduling (MDS) algorithm to obtain a suboptimal solution for task scheduling on multicore systems. The performance of proposed algorithm is tested for 4x4 mesh-based networks i.e., Mesh, DMesh, ZMesh and Torus networks. The main objective is to schedule the independent tasks on 16-cores systems uniformly with minimum makespan of execution.

The performance is measured by considering homogeneous cores of the system. The load imbalance and execution times are considered as metrics to evaluate the performance of the proposed algorithm. The makespan is minimized by exploiting duplication approach in which non-adjacent cores of the system are effectively utilized for computations. Curves are drawn and comparative analysis is carried out. Simulation results show an improvement of 20-30% in load imbalance while maintaining an overall execution makespan.

A promising future direction in this area is to consider the performance of proposed scheduling approach on heterogeneous computing systems. We plan to extend the presented algorithm to the dynamic environment where process load, computing resources and network conditions during the execution of varying input applications. Apart from LIF and execution time, other performance metrics such as Computation-To-Communication Ratio (CCR), Normalized Schedule Length (NSL), Speedup Rate (SR), etc. will be considered for performance evaluation.

References

[1] Geer, D.: Chip makers turn to multicore processors. *Computer*, *38*, 11-13 (2005).

[2] Al-daloo, M., Soltan, A., Yakovlev, A.: Overview study of on-chip interconnect modelling approaches and its trend. In: Proceedings of 7th International Conference on Modern Circuits and Systems Technologies (MOCAST). pp. 1-5 (2018).

[3] Alimi, I., Patel, R., Aboderin, O., Abdalla, A.: Network-On-Chip Topologies: Potentials, Technical Challenges, Recent Advances and Research Direction. Reviewed Chapter (2021). 10.5772/intechopen.97262.

[4] Ghosh., A., Sinha, A., Nancy, Chatterjee, A.: Exploring Network on Chip Architectures Using GEM5. In: International Conference on Information Technology (ICIT), pp. 50-55, (2017).

[5] Augonnet. C., Thibault, S., Namyst, R., Wacrenier, P. StarPU: A unified platform for task scheduling on heterogeneous multicore architectures. Concurr Comput Pract Exper. 23(2): 187-198 (2011).

[6] Lakshmanan, K., Kato, S., Rajkumar, S.: Scheduling Parallel Real-Time Tasks on Multi-core Processors. In: Proceedings of 31st IEEE Real-Time Systems Symposium, pp. 259-268, (2010).

[7] Chronaki, K., Rico, A., Casas, M., Moretó, M., Badia, R.M., Ayguadé, E., Labarta, J., & Valero, M.: Task Scheduling Techniques for Asymmetric Multi-Core Systems. IEEE Transactions on Parallel and Distributed Systems, 28(7), 2074-2087 (2017).

[8] Chronaki, K., Rico, A., Badia, R.M., Ayguadé, E., Labarta, J., & Valero, M.: Criticality-Aware Dynamic Task Scheduling for Heterogeneous Architectures. In: Proceedings of the 29th ACM on International Conference on Supercomputing, pp. 329-459, (2015).

[9] Yao, X., Geng, P., Du, X.: A Task Scheduling Algorithm for Multi-core Processors. In proceedings: International Conference on Parallel and Distributed Computing, Applications and Technologies. pp. 259-264 (2013).

[10] Zheng, Z., Chen, X., Wang, Z., Shen, Li., Li, J.: Performance model for OpenMP parallelized loops. In: Proceedings 2011 International Conference on Transportation, Mechanical, and Electrical Engineering (TMEE). pp. 383-387 (2011).

[11] Yuan, L., Jia, P., Yang, Y.: Efficient scheduling of DAG tasks on multi-core processor based parallel systems. In: Proceedings: IEEE Region Conference pp. 1-6 (2015).

[12] Wunderlich, S., Cabrera, J., Fitzek, F., Reisslein, M.: Network Coding in Heterogeneous Multicore IoT Nodes with DAG Scheduling of Parallel Matrix Block Operations. IEEE Internet of Things Journal, 4(4) pp. 917-933 (2017).

[13] Tang, x., Li, K., Liao, G., Li, R.: List scheduling with duplication for heterogeneous computing systems. J. Parallel Distrb. Computing. 70(4) pp. 322-329 (2010).

[14] Topcuoglu, H., Hariri, S., Wu, M.: Performance-effective and low-complexity task scheduling for heterogeneous computing. IEEE Transactions on Parallel and Distributed Systems. 13 (3) pp. 260-274 (2002).

[15] Bansal, S., Kumar, P., Singh, K.: Dealing with heterogeneity through limited duplication for scheduling precedence constrained task graphs. J. Parallel Distrb. Computing. 70(4) pp. 479-491 (2005).

[16] Mohammad, S., Ababneh.: Improving system performance in non-contiguous processor allocation for mesh interconnection networks. J. Simulation Modelling Practice and Theory.

[17] Simulation Modelling Practice and Theory, 80, pp. 19-31 (2018).

[18] Manaullah.: Minimum Distance Scheduling Scheme on Linearly Extensible Multiprocessor Network. International Journal of Emerging Technology and Advanced Engineering. 3 (10) pp. 536- 541 (2013).

[19] Prasad, N., Mukherjee, P., Chattopadhyay, S., Chakrabarti, I.: Design and Evaluation of ZMesh Topology for On-Chip Interconnection Networks. Journal of Parallel and Distributed (5) pp. 17-36 (2018).

***