International Journal of Research in Advent Technology, Special Issue, March 2019
E-ISSN: 2321-9637
Available online at www.ijrat.org

Packet based Scheduling Algorithm for Massively Parallel Systems

Savita Gautam[1], Abdus Samad[2]
[*]University Women's Polytechnic
Aligarh Muslim University, Aligarh
savvin2003@yahoo.co.in[1], abdussamadamu@gmail.com[2]

Abstract— Task migration between processing elements (nodes) in a multiprocessor system largely affects the overall performance of systems. Although number of task scheduling algorithms have been proposed and implemented, selecting best algorithm for a particular multiprocessor system always remains a challenging problem. This paper focuses attention on reducing the execution time while maintaining the load balance performance by selecting the weight for migration of tasks in terms of packets. The mechanism decides the weight of tasks to migrate in terms of packets dynamically based on the task structure as well as on the types of multiprocessor system architecture. Thus, an enhancement of packet formation in the existing approach is made to obtain optimal results. We consider the well known class of interconnection network known as cube based systems for evaluating the performance. The experimental results indicate that the proposed technique reduces the overall makespan of execution with an improved performance of the system.

Keywords: Packet Scheduling, Hypercube, Load Imbalance Factor, Execution Time, Task Migration, Cube Architectures.

1. INTRODUCTION

Efficiency of an Interconnection network depends on optimal migration of tasks by finding out the best execution sequence. In order to minimize the makespan of overall execution, several techniques are applied to find the most efficient algorithm. The main objective of these algorithms is to find the allocation method that minimizes the execution time while satisfying the load balancing criterion.

The problem of task scheduling in homogeneous computing is deciding how many tasks are to be scheduled on given processors (nodes). Node selection primarily depends upon the imbalance of load among the system. Many scheduling algorithm have been designed and proved to be efficient in terms of distributing the load uniformly, however, less attention is paid to reduce the overall makespan [1,2,3,4].

Task scheduling is widely classified into two categories: Static scheduling and dynamic scheduling. In static scheduling all information related to the tasks are available beforehand, whereas in dynamic scheduling information is made available during runtime. Among different scheduling, List scheduling approaches have been proven to be most effective [5]. These algorithms incorporate the prioritization of tasks, look-ahead attribute for processor selection and easy implantation.

Task scheduling algorithms have been well explored by the researchers with a focus on finding suboptimal solutions. High performance scheduling (HPS) algorithm [6] considers these factors i.e level sorting, task prioritization and processor selection. A similar approach is applied in [7] where cases are categorized according to their levels and priorities are set. A three levels of priority in the algorithm in which volume of task are also taken in to consideration is proposed and implemented by Dai et al. On the other hand task duplication is adopted during processor selection to reduce the schedule length [8].

This paper studies the scheduling of tasks on a network of identical processors where the load is divided into approximately equal number of packets to achieve load balancing in the system. The load balancing is directly proportional to the performance of the system. The quick is the load balancing, the efficient is the system.

The rest of the paper is organized as follows. Section 1 is introduction. Section 2 describes the network model and problem formulation. In section 3 the existing algorithm and proposed enhancement for improving the algorithm is described. The results obtained by implementing the proposed algorithm are presented and discussed in section 4. The last section is written for conclusion.

2. NETWORK MODEL AND PROBLEM FORMULATION

This section describes the network models used to address the given problem. The study of interconnection networks is an emerging research area in the design of high performance parallel system, computer networks and also in distributed computing environment. To satisfy the requirements of modern ICT technologies and to take services of massively parallel systems, computer architects have always strived to increase the performance of their architectures by designing high quality networks. It is anticipated that large parallel applications shall be

deployed on next generation high performance computing systems [9]. Therefore, it is apparent that low cost, efficient and scalable architectures design has a deep concern while working with parallel applications.

There are several categories of Interconnection networks to address the problem of task scheduling. Prime examples are found in ring network, hypercube, debruijn network, tree networks,star graph and mesh networks [9,10]. Jingxi et al. considers an arbitrary connected network comprising of n nodes which accept load from the user [11, 12]. Cube based networks have all the desirable properties of efficient interconnection networks. These networks are equipped with good topological characteristics that include small diameter, symmetry, high bisection width and better connectivity. A recent study conducted by [13] introduced the profitability of these interconnection networks. To make best utilization of nodes in such networks it is must to design efficient scheduling algorithms. In this paper effort is made to enhance the benefits of existing approaches by reducing overall make span. In particular, four cube-based networks namely standard hypercube (HC), Linear Cross Cube (LCQ) (Fig. 1), Linearly Extensible Network (LEC) (Fig. 2) and debruijn (Fig. 3) networks with eight processors have been taken into consideration [9], [13], [14].

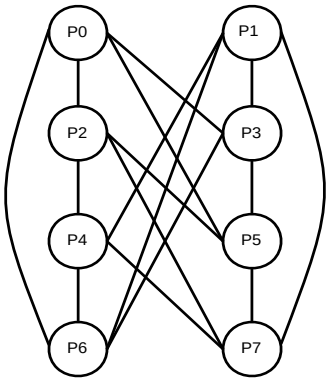

Fig. 1. LCQ Architecture with eight processors

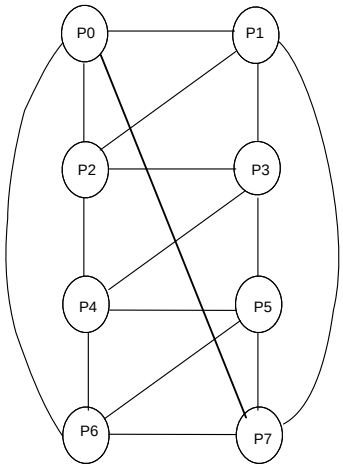

Fig. 2. LEC Architecture with eight processors

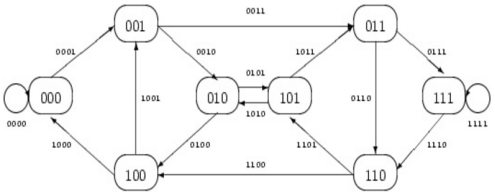

Fig. 3. debruijn network with eight processors

3. DYNAMIC ALGORITHM AND PROPOSED STATEGY

Dynamic algorithms operate to adjust the load by taking decisions on fly with undetermined policy and could be easily applied to system that supports variable task structures. With regard to dynamic allocation, there have been significant efforts in characterizing and dividing requests locally [15]. The first phase of the algorithm decides the selection of processors for task allocation randomly. The input to the networks is a set of independent tasks that can be executed on any nodes. However, to distribute the load among different nodes task are migrated between overloaded processors to underloaded processors. Such migration could be carried out level wise and processors are considered to cover the entire network. Like most of the previously proposed algorithms the performance is evaluated in terms of load imbalance and execution time by implementing on considered networks.

Similar approach could be applied on cube-based networks as well as Tree based network which have gain attention in the recent past. User submits loads to the available nodes which is ultimately balanced among the entire network by the schedulers. In the present work we considers grouping of tasks for migration. Tasks that are independent of each other are assigned to different groups forming a packet. It is possible to migrate and execute tasks in the same level in paralleled. We considered packet formation of four and eight tasks at each level.

The algorithm calculates ideal load value and execution time for each level, load imbalance is evaluated that helps to take load migration decisions. The load imbalance factor for k^{th} iteration, denoted as LIF_k, is defined as

$$LIF_k = [\max\{load_k(P_i)\} - (ideal\_load)_k] / (ideal\_load)_k$$

where, $(ideal\_load)_k = [load_k(P_0) + loadk(P_1) + \ldots + load_k(P_{N-1})] / N$,
and $\max(load_k(P_i))$ denotes the maximum load pertaining to iteration k on a processor P_i ,$0 \le i \le N-1$,and $Load_k(P_i)$ stands for the load on processor P_i due to k^{th} level. The pseudo code for the proposed strategy is given in Table 1.

TABLE 1: Proposed Algorithm

```
Input: N  processor network
output: Balanced network
I.  Packet Scheduling
1.  Consider  N as total number of Processors.
2.  Randomly generate Load at all processors N.
    /* load is the number of Tasks assigned to a
    processor  and tasks are divided into packets* /
3.  Calculate TotalLoad on all processors.
    TotalLoad=Load(P1) + Load(P2) +_ _ __ _ _
    +load(PN).
   /* P1 is Load assigned to processor P1  and so on */
4. Compute IdealLoad
          IdealLoad=TotalLoad/No. of Processors
          /* Totalload is calculated as in step 4*/

5.  According to the Load assigned to each processor
identify AcceptorSet  processors.
6.  For each processor in AcceptprSet ,perform
packetization.
7.  Compute maximum Load for all processor in a network.
8.  Calculate Load Imbalance Factor(LIF)
II.packetization
 1 For each processor in AcceptprSet ,find out
ConnectedProcessor collection
2.  For each connectedprocessor
       if   load(ConnectedProcessor)>=IdealLoad+
            packetsize
    Migrate packets from ConnectedProcessor to
     AcceptorProcessor
                 while
      load(AcceptorProcessor)<=IdealLoad and

load(ConnectedProcessor)>=IdealLoad+ packetsize
                              load(AcceptorProcessor)=
load(AcceptorProcessor)+packet
                              load(ConnectedProcessor)=
load(ConnectedProcessor)-packet
if, prev = 0, n_donr = 0, n_accr = 0;
    Let level of connectivity = 1;
  Identify the donors and set n_donr.
  Identify the acceptors and set n_accr
. while (lifac > LIF)
    {
      for (i = 0; i < n_donor; i++)
    {
      while (donr[i] is overloaded)
    {
    { for (j = 0; j < n_accr; j++)
       {
      if (donr[i] is connected to accr [j])
        migrate load;
     }
      donr (i) is exhausted or balanced;
     }}
        if desired level of Lif is not achieved then set the
level of connectivity to 2 and repeat step 4
      }
  End of procedure
```

4. EXPERIMENTAL RESULTS AND ANALYSIS

In this section, we report the comparative evaluation of the proposed strategy with four well known architectures using various performance metrics. A dynamic algorithm allocate load at runtime based on no or a little priori information, which may determine when and whose tasks can be migrated. The approach used for task migration has already been discussed in section 3. In simulation environment, the results obtained by implanting the algorithm are presented in the form of curves. Mostly any load balancing algorithm considers the overall load at a processor. However, the proposed algorithm takes into account the active load only for balancing. Donor and acceptor nodes are identified and with the scheduling policy, with appropriate packet size the migration of load balances the overall load of the system.

5. COMPARATIVE STUDY

In this section efforts are made to compare the performance of cube based networks that have already been studied. These networks are similar in terms of their topological parameters. All the networks have equal number of nodes (eight nodes).

To study the behaviour of the scheduling scheme on the considered networks, the different values of LIF are evaluated by forming packets of tasks and migrating them on different directly connected nodes. In particular packets of unit, four and eight tasks are formed to monitor the load imbalance. Similarly, the value of execution time is evaluated while keeping the same load. Comparisons are made and the curves are plotted for each considered networks and shown in Figures 4 to 7. The Figures demonstrate that in all the cases similar pattern of reduction in LIF is obtained. However, the value of highest LIF varies in each case. If we consider the standard hypercube no significant improvement is obtained after packet formation (Fig. 4) and network performs better with unit migration. On the other hand lesser value of initial LIF is obtained when packet of eight tasks is considered for migration (Fig. 5). Similar results are obtained for LCQ network (Fig. 6). In case of debruijn network the algorithm produces similar results as that in hypercube architecture (Fig. 7).

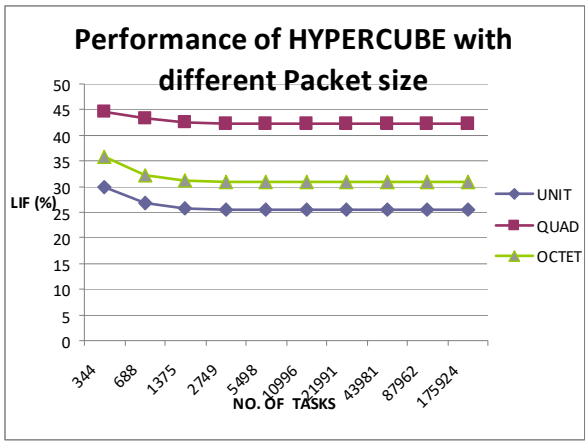

Fig. 4. Performance of Hypercube with different packet sizes.

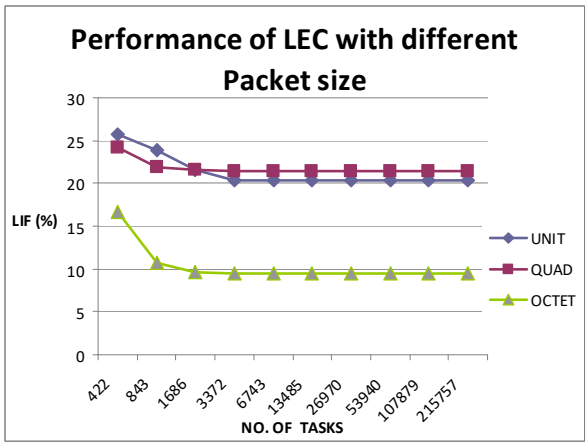

Fig. 5. Performance of LEC with different packet sizes.

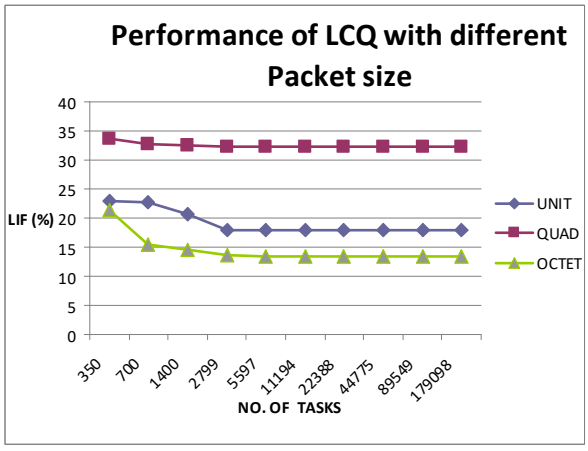

Fig. 6. Performance of LCQ with different packet sizes.

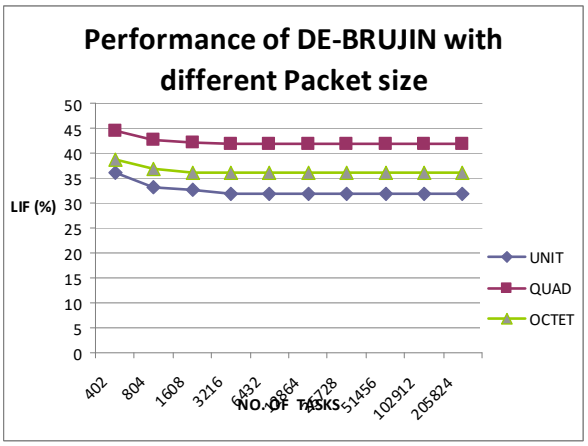

Fig. 7. Performance of de bruijn with different packet sizes.

Another parameter to compare the performance is the execution time. The balancing time reduces significantly if the block sizes of the tasks are increase. Considering the value obtained for hypercube network the execution time is reduced from 50% and 40%. All other considered networks take smaller time to attain the optimum value of LIF with packet formation. The results obtained for hypercube network is depicted in Fig. 8. If the packet size is increased continuously then the variance in the balancing time will appear, however, the larger load imbalance may results.

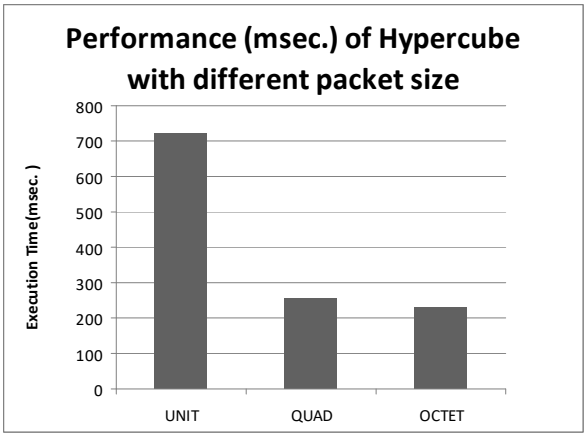

Fig. 8. Performance of Hypercube with different packet sizes in terms of execution time.

6. CONCLUSION

The work presented in this paper is based on the implementation of a new dynamic scheduling algorithm that operates on different block sizes known as packets. For our study we implanted the proposed techniques on cube based networks. The curves obtained from the simulation studies indicate the average behaviour LIF obtained at different levels and time taken for execution. The simulation results show that the proposed algorithm improves overall makespan by grouping independent tasks

while maintaining the load balance errors. The graphical representation demonstrates that the value of LIF in almost all cases is lesser than that obtained by applying standard scheduling algorithm and further reduced when greater numbers of tasks are applied on the systems.

The proposed technique is especially beneficial to schedule the load on cube based or mesh type interconnection networks. We have evaluated and presented the results to monitor the performance of these networks with eight nodes, however, the proposed technique is equally good for networks having more than eight processors. It may be concluded that with the proposed algorithm the system with mesh topology or having cubic architecture may constitute a class of high-performance interconnection networks.

REFERENCES

[1] C. H. Yang, P. Lee and Y. C. Chung, "Improving static task scheduling in heterogeneous and homogeneous computing system." IEEE Parallel Processing,, pp. 45-50, 2007.

[2] L. C. Canon, E. Jeannot, R. Sakellarious, and W. Jheng, "Comparative evaluation of the robustness of dag scheduling heuristics, Springer US in Grid Computing, pp. 73-84, 2008.

[3] S.Venugopalan and O. Sinnen, "Optimal linear programming solutions for multiprocessor scheduling with communication delays," Springer Berlin Heidelberg, pp. 129-138, 2012.

[4] S. Su., K. Li., H. Chen and K. Li, "Cost-efficient task scheduling for execution large programs in the cloud," Parallel Computing, Vol. 39. No. 4, pp. 177-188, 2013.

[5] H. Arabnejad, "List based task scheduling algorithms on heterogeneous systems-an overview, 2003.

[6] E. Havarasan, P. Thambidurai and R. Mahilmannan, 'High performance task scheduling algorithm for heterogeneous computing system," Springer Berlin Heidelberg in Distrubited and Parallel Computing, pp. 193-203, 2005.

[7] E. Ilavarasan, P. Thambidurai and R. Mahilmannan, "Low complexity performance effective task scheduling algorithm for heterogeneous computing environment," Journal of Computer Science, Vol. 3, No. 2, pp. 94-103, 2007.

[8] Y. Dai and X. Zhang, "A synthesized heuristic task scheduling algorithm," The Scientific World Journal, pp. 1-9, 2014.

[9] N. Parasad, P. Mukkherjee, S. Chattopadhyay and I. Chakrabarti, "Design and evaluation of ZMesh topology for on-chip interconnection networks," Journal of Parallel Distributing Computing, pp. 17-36, 2008.

[10] B. Parhami, "Challenges in interconnection network design in the era of multiprocessor and massively parallel Microchips," Proc. Int'l Conf. in Computing." pp. 241-246, June 2000.

[11] D.M Kwai and B. Parhami, "Incomplete K-ary n-cube and its Derivatives," Journal of Parallel and Distributed Computing, Vol.16, No. 2, pp.183-190, Feb.2004.

[12] D. Zou., H. Liu, L. Gao and S. Li, "An improved differential evolution algorithm for task assignment problem," Journal of Engineering Applications of AI, Vol. 24, pp. 616-624, 2011.

[13] Z. A. Khan, J. Siddiqui and A. Samad, "Properties and Performance of Cube-based Mutiprocessor Architectures," Int. Journal of applied evolutionary computation (IJCNIS). Vol. 7, No. 1pp. 67-82, 2016.

[14] Z. A. Khan, J. Siddiqui and A. Samad, "Linear Crossed Cube (LCQ): A new Interconnection Network Topology for Massively Parallel Architectures," Int. J. of Computer Network and Information Science (IJCNIS), ISSN/ISBN NO: 2074-9090, Vol. 7, No. 3, pp. 18-25, 2015.

[15] Z. Zeng and B. Veeravalli, "Design and Performance Evaluation of Queue-and-Rate-Adjustment Dynamic Load Balancing Policies for Distributed Networks," IEEE Trans. on Computers, Vol. 55, No. 11, pp. 1410-1422, Nov. 2006.