International Journal of Control Theory and Applications

ISSN : 0974-5572

© **International Science Press**

Volume 9 • Number 46 • 2016

Heterogeneous Scheduling Using Controlled Duplications on Multiprocessor System

Mehak Aggarwal[a] and Nirmal Kaur[b]

[a,b]Department of CSE, University Institute of Engineering and Technology, Panjab University, Chandigarh, India. Email: [a]mehak10oct@gmail.com

Abstract: Scheduling of parallel applications on the multiprocessors is the critical issue which emerged with the parallel systems, distributed systems and with the developments in the processing technology to achieve high performance computing. In order to gain high efficiency, increased speedup and overall improved makespan of fine grain task graphs of parallel applications, we have proposed an algorithm called "Heterogeneous Scheduling with Controlled Duplications On Multiprocessor System (HCDM)" is proposed which utilizes reduced ancestor nodes duplications for finding optimal solution. The performance of the proposed algorithm is compared with well known duplication based scheduling algorithm: HLD and list based scheduling algorithms: HEFT and ECTS. The exhaustive simulation results reveal better performance in terms of efficiency and makespan.

Keywords: Heterogeneous Scheduling, Multiprocessor System, Controlled Duplication, Heuristic.

1. INTRODUCTION

In recent years, the growth of technology (powerful processors, high speed networks, standard software tools, etc.) led the parallel and distributed systems come into role for many large scale applications processing in different areas of science, engineering and commerce viz. scientific computing, image processing, system modeling and simulation, database systems, optimization problems and many more. To efficiently process the parallel applications, many key factors contribute such as hardware design, number of processors, parallelism, software tools, and scheduling. But scheduling is the keystone of the parallel computing. Scheduling is broadly classified as static and dynamic (1). In static scheduling, the characteristics of an application represented in the directed acyclic graph (DAG), such as execution time of tasks (or nodes) on processors, data size of communication between tasks, bandwidth, and tasks dependencies are known priori. On the other hand, dynamic scheduling is done at run time for load balancing and the above stated characteristics are not known in advance. To achieve high performance computing, the parallel applications are partitioned into independent subtasks and scheduled on multiple processors simultaneously. Parallel application scheduling is a scheduling problem to find the spatial and temporal assignments (14) of the tasks onto the processors of the target system which results in shortest possible execution time of the tasks; where task is a set of instructions which are executed consecutively without any preemption on the same processor. It aims to achieve high efficiency, reliability and quick response from the system.

The multiprocessor systems are classified into homogeneous and heterogeneous systems. (1) Homogeneous systems comprise of identical processors with similar computing capacity whereas heterogeneous systems may have more than single type of processor having different processing capabilities to handle particular tasks. Although scheduling on heterogeneous computing environment is much complex but the diverse set of parallelism in the application program and diverse computation capability of processing elements has triggered the use of heterogeneous computing environment. The application scheduling is NP-complete (1) problem for general cases. Optimal solutions could be found through some restrictions or through exhaustive search over all processors for all tasks. Exhaustive search method guarantee optimal solutions but not suitable for large sized task graphs due to time consuming as well as large resource utilization. Hence, the heuristics (rely on rule of thumb) are worked upon.

In this paper, we have proposed a scheduling algorithm called HCDM which hybridizes the list scheduling heuristic and duplication based scheduling heuristics based on controlled duplications for heterogeneous systems. The redundant duplications which may occur for a join task is eliminated using backtracking without increasing schedule length and therefore prevents the excess resource usage.

The organization of paper is presented as follows. In section 2, the previous related work and motivation has been described. Section 3 formulates the parallel application scheduling problem. Section 4 discusses the proposed algorithm for scheduling problem. Simulation results obtained for HCDM are presented in Section 5. Section 6 concludes the paper based on the simulation results and performance analyses.

2. RELATED WORK AND MOTIVATION

Because of the NP-completeness of the scheduling problem in deterministic computing environment, a large set of heuristics have been proposed and these heuristics are characterized into three categories: List scheduling heuristics, duplication based scheduling heuristics and clustering heuristics.

In list scheduling heuristic, an ordered list of tasks is created by allotting priority to each task and thereafter based on the priority; tasks are selected from the list for execution. List scheduling algorithms are generally preferred because they generate schedules of good quality with less complexity. The heuristics in this category include HEFT (6), CPOP (6), PETS (15), ECTS (22), and PEFT (24). List scheduling heuristics are good for fine grain (i.e. less communicating) tasks. It is preferred as it generates good quality schedules with less complexity. But the performance deteriorates significantly for task graphs having high communication to computation cost ratio (CCR).

Clustering heuristics makes the clusters of heavily communicating tasks and schedule them onto the same processor even if other processors are available, thus there is a tradeoff between parallelism and inter process communication. It is also known as three phase scheduling heuristic. The first phase is comprised of grouping of the tasks which are massively communicating into a set of clusters. In the second phase, the generated clusters are mapped onto the set of available processors. In the third phase, merging of clusters or de-clustering is done based on the available set of processors. The heuristics in this category include TRIPLET (5), LDBS (7), and HCDDSL (19). But the trading off between inter process communication and parallelism is a major drawback of this heuristic.

In task duplication based scheduling heuristics; the parent node of the task to be scheduled is duplicated to the processor which minimizes task's execution time, thus reducing the communication overhead. The duplication based scheduling can lessen the communication overhead for a given task graph by allocating or duplicating some of the tasks to more than one processor. A number of different strategies can be used to duplicate ancestor nodes. The algorithms in this category include SD (8), HLD (13), HCPFD (12), RD (17), and HED (17). The complexity of heuristic increases but these heuristics outperforms when CCR values are high.

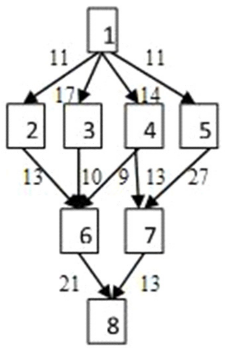
Guided random search methods are approximate algorithms which use random choice to guide themselves through the problem space. These techniques combine the knowledge of previous search result with some randomizing features to generate new results. Examples are A* algorithm (20), simulated annealing (21), Genetic algorithms (23) and particle swarm optimization (14). It provide good quality schedule. But they have higher execution time than other alternatives and hence extensive tests are required to find optimal values for the set of control parameters.

3. PARALLEL APPLICATION SCHEDULING PROBLEM

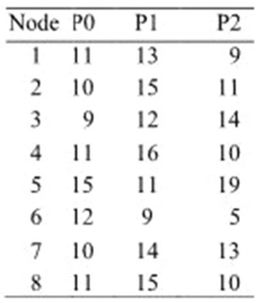
Parallel Application Scheduling problem is to schedule the application on target system in an efficient manner and is represented with two models: Application Model and Target System Model.

3.1. Application Model

The application to be scheduled is represented by the application task graph (also known as macro dataflow graph). (1). The static application scheduling problem is represented by weighted DAG in which nodes represent the tasks and edges represent the intertask dependencies or precedence relations among them as shown in Figure 1.



(a)



Node	P0	P1	P2
1	11	13	9
2	10	15	11
3	9	12	14
4	11	16	10
5	15	11	19
6	12	9	5
7	10	14	13
8	11	15	10

(b)

Figure 1: (a) Example DAG representing precedence constraints between tasks (b) Computation Costs on different processors

Each node n_i of DAG is labeled with the *computation cost,* w_i and the directed edges in the parallel program graph correspond to communication data and precedence constraints between the tasks. The weight of the edge from node n_i to node n_j is referred to as the *communication cost*, c_{ij} (or $c(n_i, n_j)$). The node at which edge starts is called the *parent* node, while the destination node is called the *child* node. A node without a parent is called an *entry* node and a node without a child is called an *exit* node. The entry or exit node can be more than one. If a task scheduling algorithm requires single-entry and single-exit task graphs, a zero-cost pseudo exit and/or entry node with zero-cost edge is used.

3.2. Target System Model

The target system model is a set of heterogeneous multiple processors, P = $\{p_1, p_2, \ldots, p_m\}$. The processor heterogeneity is given by the factor, β which is the range percentage of computation cost on each processor for each task in a task graph. There are few assumptions made about target computing system for scheduling problem: the communication network of processors is fully connected, communications among processors can be performed concurrently, local communication has zero cost, and there is dedicated communication subsystem.

3.3. Basic Terminology of Scheduling Algorithm

(a) *pred(n_i)*: It denotes the immediate predecessors of task n_i in a DAG.

(b) *avail$_j$*: It is defined as the earliest available time of processor at which it is ready for task execution and is calculated by analyzing the suitable idle time slot for node n_i.

(c) *Makespan*: Makespan is the total time span for executing an application and is given by

$$\text{Makespan} = \max\{\text{AFT}_i\} \tag{1}$$

where, AFT_i is the actual finish time of node n_i.

(d) *DAT(n_i,p_j)*: The DAT is the latest data arrival time of node n_i from all of its predecessors or immediate parent on processor p_j. The node n_i can start its execution only after collecting data from its parent nodes.

$$\text{DAT}(n_i, p_j) = \max\{\text{AFT}_y + c_{i,y}\}\ \forall n_y \in \text{pred}(n_i) \tag{2}$$

where, AFT_y is the actual finish time of node n_y.

(e) *EST(n_i,p_j)*: The EST refers to the expected execution start time of node n_i on processor p_j. It is constrained by the precedence between the nodes and is given by

$$\text{EST}(n_i, p_j) = \max\{\text{DAT}(n_i, p_j), \text{avail}_j\} \tag{3}$$

(f) *EFT(n_i,p_j)*: The EFT refers to the expected earliest finish time of node n_i on processor p_j and is given by

$$\text{EFT}(n_i, p_j) = \text{EST}(n_i, p_j) + w_{i,j} \tag{4}$$

3.4. Scheduling Constraints

The scheduling problem has considered two types of constraints to be followed:

1. Precedence Constraint
2. Processor Constraint

Precedence Constraint states that a task node n_j in a DAG can start it's execution on processor p_k only after the arrival of all data from it's all immediate parent node on that processor.

$$\text{ST}(n_i, p_k) \geq \text{DAT}(n_i, p_k),\ \forall n_i \in \text{pred}(n_j),\ \forall p_k \in \text{P}_y \tag{5}$$

where, P_y is the subset of processors on which parent node of the task node is either scheduled or duplicated, $\text{ST}(n_j, p_k)$ is the start time of node n_j on processor p_k.

Processor Constraint states that for any two nodes n_i and n_j to be scheduled on the same processor than other task can start it's execution after the completion of execution of first task.

$$\text{FT}(n_i, p_k) \geq \text{FT}(n_j, p_k) \text{ or } \text{FT}(n_j, p_k) \geq \text{FT}(n_i, p_k),\ \forall p_k \in \text{P} \tag{6}$$

where, $\text{FT}(n_i, p_k)$ is the finish time of node n_j on processor p_k.

4. PROPOSED ALGORITHM HCDM

The proposed algorithm is a three phase algorithm, namely task prioritization phase, processor selection phase and redundant duplication removal phase.

4.1. Prioritization Phase

In this phase, priority list is maintained by assigning priorities to each task of the DAG using level prioritization and rank of the task at each level. The tasks which are at the same level are independent to each other and may be executed parallel. The task with the highest priority is considered as candidate node from the priority list for scheduling till the list is empty. Hence, they are assigned a same level priority using bottom up traversing technique. In the next step, Expected Time (ET) is calculated using Average Computation Cost (ACC), Maximum Data Arrival of Communication (MDC) and maximum rank of parent node.

Definition 1: Given the graph G, the ACC(n_i) or ACC$_i$ of a node n_i to be scheduled in heterogeneous system is the average of the execution time on each processor and is given by

$$\mathrm{ACC}(n_i) = \sum_{j=0}^{m} \frac{w_{i,j}}{m} \tag{7}$$

where, $w_{i,j}$ represents the computation cost of task n_i on processor p_j and $j = 1, 2, ..., \mathrm{m}$.

Definition 2: MDC is the maximum time the task node is waiting from the start of the execution the of parent node to the data arrived from that parent node and is calculated as:

$$\mathrm{MDC}(n_j) = \max_{ni \in \mathrm{pred}(nj)} \{\mathrm{ACC}(n_j) + \mathrm{C}_{i,j}\} \tag{8}$$

Rank of the task node n_j is computed as

$$\mathrm{Rank}_j = \mathrm{ACC}(n_j) + \mathrm{MDC}(n_j) \tag{9}$$

At same level, highest priority is assigned to task with highest rank. Priority list is maintained by starting from top level of DAG.

4.2. Processor Selection Phase

In the second phase, the best fit processor which provides earliest finish time of the node is selected for task scheduling. For estimating the finish time on each processor, node insertion and node duplication techniques are used. The node n_i can start execution after receiving the data from all its immediate parents to satisfy the precedence constraint. The parent node from which the data arrives latest is termed as Most Important Immediate Parent (MIIP). After receiving the data from MIIP, the node n_i can start execution either at the available time of processor (avail$_k$) or at the earlier if suitable idle slot is available. The earliest idle slot is found by scanning the whole time span of processor. The duplication can be done redundantly by duplicating the ancestor nodes to the top of graph and horizontally.

4.3. Redundant Duplication Removal

In the third phase, redundant duplications have been removed. According to K. Shin et. al, if a join node n_i has only one child and this child node has multiple parent node than n_i can contribute to redundant duplication condition. K. Shin et. al has explained the two necessary conditions for the redundant duplications as follows:

1. A join node is the only child node of atleast one of its parent node.
2. The computation cost of the parent node is smaller than communication cost between any one of the parents and the join node.

The above two conditions are referred as Redundant Duplication Condition (RDC) To handle RDC, a set of tuples (J(m), RDC(n)) is prepared, where J(m) is the array of join nodes and RDC(n) is the array of parents of join node in J(m). If the node to be scheduled belongs to join set and the parent node has single child than original allocation will be redundant after duplication, which would be removed. For two or more such tuples are cascading, the bottom up approach has been used to remove redundant duplications. The pseudo code for proposed algorithm HCDM is shown in Figure 2.

```
HCDM Algorithm
Assume P is the set of m processors; G is the DAG of an application with n subtasks.
Begin
Phase1: // Task prioritization Phase
Step 1: Construct a priority list, L of task sequence to be scheduled based on level prioritization and rank:
a) Determine the task nodes at same level, $l_k$ using bottom up traversing ,
b) For each task, $n_i$ at level $l_k$, do
    {
     Calculate $ACC_i$, $MDC_i$ and $ET_i$ and compute rank($n_i$), using equations 7, 8, 9 respectively.
     Assign priority in decreasing order of $ET_i$ starting from top level $l_k$
    }
Step 2: Find the set of all tuples of (J(m), RDC(n)) satisfying redundant duplication conditions (RDC).

Phase 2: // Processor Selection Phase
Step 3: For all task $n_i \in L$, do
       {
        call Find_AFT_Using_Dup($n_i$,$p_j$) for all $p_j \in P$ and record AFT($n_i$,$p_j$).
        schedule $n_i$ to $p_s$ that gives minimum AFT($n_i$,$p_j$). // $p_s$ is selected processor.
         undo all the duplications just performed on the other processors except on $p_s$.
Phase 3: // Redundant duplication removal phase
        if ( $n_i \in$ J(m) for some m and $n_i \notin$ RDC(n) for any n) do
        {
          for each $n_k \in$RDC(n) do
          call Elimination($n_k$,$p_s$).
        }
        schedule $n_i$ on selected $p_s$ at $AFT_i$.
       }
End

Find_AFT_Using_Dup

Begin
Step 1: Find EFT($n_i$, $p_j$) .
Step 2: If (any new MIIP($n_i$) do
       {
        determine EFS(MIIP($n_i$), $p_s$).
        if (EFT($n_i$, $p_s$) is improved by duplicating its MIIP) do
        {
          duplicate MIIP($n_i$) on $p_s$ and mark them.
          call Find_AFT_Using_Dup(MIIP($n_i$), $p_s$)
        }
Step 3: Calculate AFT($n_i$, $p_s$) and return.
End

Elimination($n_k$, $p_s$)

Begin
Step 1: if ($n_k$ is duplicated on $p_s$) do
            remove the original allocation of $n_k$ and repeatedly remove any marked duplication.
       else
          for each $n_k \in$ J(m) for any m, do
          {
            for each $n_y \in$ J(m) RDC(n) do
            call Elimination($n_y$, $p_n$) where $p_n$ is the processor on which $n_y$ is assigned.
          }
End
```

Figure 2: Pseudo code for HCDM

5. SIMULATION AND RESULT ANALYSIS

In this paper, the performance of the proposed algorithm is compared with list scheduling algorithms: HEFT and ECTS, and duplication based heuristic: HLD and are simulated on a set of irregular benchmark task graphs (4).

5.1. Performance Metrics

To measure the performance of the simulated graphs, following performance metrics have been used:

1. *Makespan:* It is the schedule length of the application graph on heterogeneous system which is computed by considering the maximum value of finish time of each node on every processor (equation 1).
2. *Efficiency:* It is the ratio of speedup and number of processors.

$$\text{Efficiency} = \frac{\text{Speedup}}{\text{Number of processors}} \times 100,$$

where,

$$\text{Speedup} = \frac{\text{Schedule length on uniprocessor system}}{\text{Schedule length on multiprocessor system}}$$

To have optimum utilization of resources, lower schedule length and high efficiency is desirable.

6. SIMULATION ENVIRONMENT

To simulate the stated algorithms (HCDM, HEFT, ECTS, and HLD) on regular benchmark task graphs, the different parameters and corresponding range of values used have been shown in Table 1.

Table 1
Parameters used in simulation environment

Parameter	*Range of values*
Number of nodes	10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30, 32
Number of processors	3, 5, 7, 9, 11
CCR	0.1, 1.0, 10
Heterogeneity Factor, β	0.1, 0.2, 0.5, 1.0, 2.0

The DAG for heterogeneous system is generated by modifying the benchmark task graphs for homogeneous system (1) using heterogeneity factor, β. A total of 900 different DAGs was generated using for three different values of CCR = {0.1, 1.0, 10}, five different values of heterogeneity factor, β = {0.1, 0.2, 0.5, 1.0, 2.0} and five different number of processors which are fully connected, P = {3, 5, 7, 9, 11}. For each DAG, the size varies from 10 to 32 with an increment of 2.

7. PERFORMANCE RESULTS

In this subsection, the performance results and comparisons of the four scheduling algorithms is presented. The performance of the heuristics for a given DAG are compared with respect to different graph characteristics viz. number of nodes, number of processors, CCR, and β.

The average makespan produced by the four algorithms is plotted as a function of number of nodes in Figure 3. The graph reveals that the makespan increases with the number of nodes in a graph for all algorithms but proposed algorithm HCDM gives substantial improvement than HEFT and ECTS and somewhat better than HLD in each case.

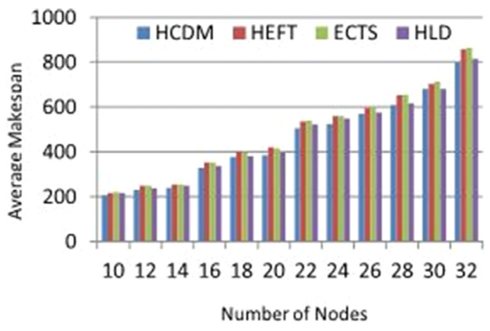


Figure 3: Average makespan for various number of nodes

Figure 4 shows the average makespan for different CCR values. It can be observed that the average makespan increases when the CCR increases. From the Figure 4, it is observed that when the CCR is low i.e. for computation intensive applications, HLD performs slightly well than HCDM whereas HCDM outperforms the ECTS and HEFT. But with increasing CCR i.e. for coarse grain DAG, HCDM performs much better than HLD, HEFT and ECTS.

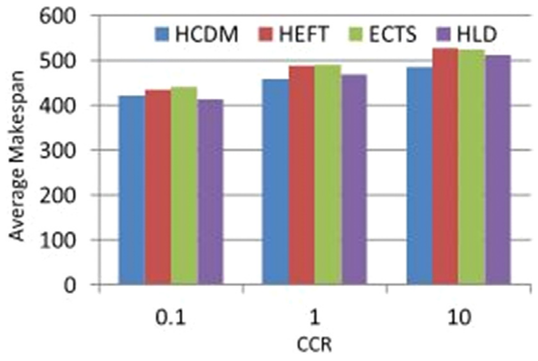


Figure 4: Average makespan for various values of CCRs

Figure 5 analyses the performance in terms of average makespan for a DAG scheduled on different number of processors. From the analysis, it has been observed that with increasing number of processors, the makespan of list based heuristics decreases up to a point and then starts increasing whereas duplication based heuristics improve the makespan consistently. When number of processors is 3, HCDM performs better than HEFT and ECTS but lower than HLD. But as the number of processors increases, the gap of improved performance in terms of average makespan between HCDM and other three heuristics increase. HCDM outperforms the HEFT, ECTS and HLD.

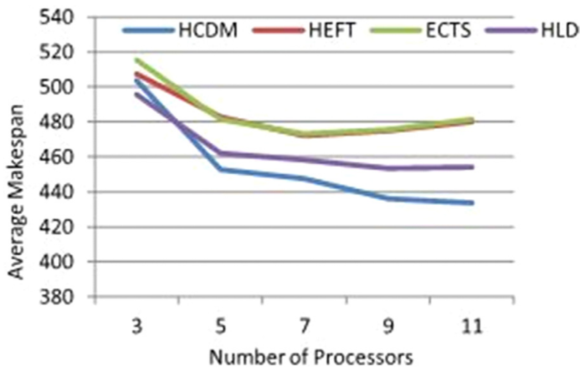


Figure 5: Average makespan for various numbers of processors

In Figure 6, the average makespan is observed with the heterogeneity factor, β. As the β value increases, the makespan of an application decreases due to different computing capability of processors. The figure shows that the HCDM is better than other comparing heuristics.

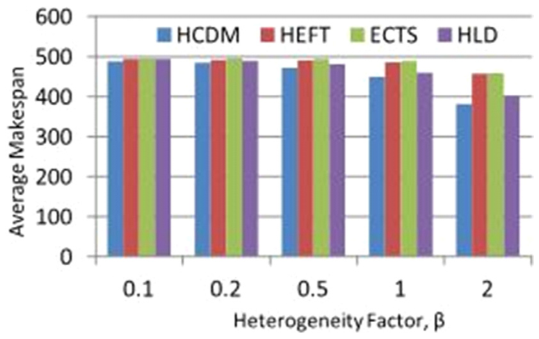
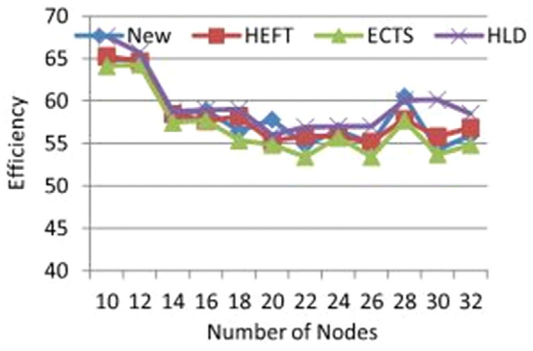


Figure 6: Average makespan for different heterogeneity of processors

Figure 7 and 8 show the simulation results for the efficiency of the algorithms for different number of processors and various CCRs respectively.



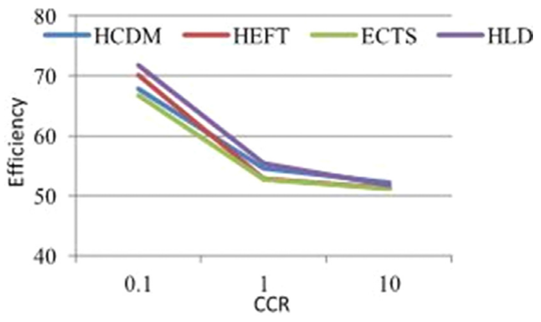


Figure 7: Average Efficiency for different number of nodes **Figure 8: Average Efficiency for various values of CCR**

Figure 7 shows that the performances, in terms of average efficiency, of the four heuristics tend to degrade as the number of nodes increases because of more number of nodes spread the execution on large processors. Figure 8 shows the comparison of average efficiency with the different values of CCR. The exploitation of the processors drastically deteriorates with increasing CCR duplications that come to be required. However, the proposed algorithm HCDM is able to cope effectively with this problem, by reducing the number of duplications in comparison with the other algorithms. For higher CCR, HCDM appears to perform better than other heuristics.

8. CONCLUSIONS

In this paper, a hybridized (list scheduling heuristics and duplication based heuristics) heuristic has been developed for heterogeneous systems and parallel applications which try to minimize the duplications by exploring the redundant duplication conditions at join node and removing those redundant duplications. Hence sophisticated scheduling strategy saves the number of duplications on the system. The duplication strategy tries to exploit the idle time slots for duplications of ancestor nodes in order not to increase the schedule length via duplication. The priority list maintained enables all the nodes at previous level to start their execution at earliest, thus maintaining precedence among nodes. From the performance analysis, it is concluded that the proposed algorithm improves the overall performance of the system and maintains efficiency. Performance comparison with algorithms HEFT, HLD, and ECTS shows that proposed algorithm, HCDM outperforms them.

REFERENCES

[1] Ashfaq A. Khokhar, Viktor K. Prasanna, Muhammad E. Shaaban, and Cho-Li Wang. Heterogeneous Computing: Challenges and Opportunities. IEEE International Symposium on High-Performance Distributed Computing; 1993.

[2] I. Ahmad, Y.K. Kwok. A New Approach to Scheduling Parallel Programs Using Task Duplication. Proceedings of International Conference Parallel Processing; volume 2, 1994. P. 47–51.

[3] I. Ahmad, Y.K. Kwok. On Exploiting Task Duplication in Parallel Program Scheduling. IEEE Transactions on Parallel and Distributed Systems; 1998. P. 872–892.

[4] Y.K. Kwok and I. Ahmad. Benchmarking and Comparison of the Task Graph Scheduling Algorithms. Parallel and Distributed Computing. 1999 Dec.; 59(3): 381-422,.

[5] Bertrand Cirou and Emmanuel Jeannot. Triplet : a Clustering Scheduling Algorithm for Heterogeneous Systems. Proceedings of the IEEE; 2001.

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

[7] A. Dogan and F Ozguner. LDBS: A duplication based scheduling algorithm for heterogeneous computing systems. Proceedings of the International Conference on Parallel Processing (ICPP'02); 2002 Aug, Vancouver, B.C., Canada; 2002, P. 352.

[8] S. Bansal, P. Kumar, K. Singh. An improved duplication strategy for scheduling precedence constrained graphs in multiprocessor systems. IEEE Transactions on Parallel and Distributed Systems. 2003; 15, P. 533–544.

[9] R. Bajaj, D.P. Agrawal. Improving scheduling of tasks in a heterogeneous environment. IEEE Transactions on Parallel and Distributed Systems. 2004; 15(2). P. 107–118.

[10] T. Hagras and J. Janeˇcek. A Simple Scheduling Heuristic for Heterogeneous Computing Environments. Proceedings of Second International Symposium on Parallel and Distributed Computing. 2004; P. 104-110.

[11] T. Hagras and J. Janeˇcek. A high performance, low complexity algorithm for compile-time task scheduling in heterogeneous systems. Parallel Computing. 2005; 31: 653–670.

[12] Sanjeev Baskiyar, Christopher Dickinson. Scheduling directed a-cyclic task graphs on a bounded set of heterogeneous processors using task duplication. Parallel Distributed Computing. 2005; 65: 911 – 921.

[13] S. Bansal, P. Kumar, Kuldip Singh. Dealing with heterogeneity through limited duplication for scheduling precedence constrained task graphs. Journal of Parallel and Distributed Computing.2005; 65.

[14] Oliver Sinnen, Leonel Augusto Sousa, Senior Member, IEEE, and Frode Eika Sandnes. Toward a Realistic Task Scheduling Model. IEEE Transactions on Parallel and Distributed Systems. 2006 March; 17(3).

[15] E. Ilavarasan and P. Thambidurai. Low complexity performance effective task scheduling algorithm for heterogeneous computing environments. Journal of Computer sciences. 2007; 3(2): 94-103.

[16] Kwang Sik Shin, Myong Jin Cha, MunSuck Jang, JinHa Jung, Wan Oh Yoon, SangBang Choi. Task scheduling algorithm using minimized duplications in homogeneous systems. Parallel Distributed Computing. 2008; 68:1146–1156.

[17] Amit Agarwal and Padam Kumar. Economical Duplication Based Task Scheduling for Heterogeneous and Homogeneous Computing Systems. WEE International. Advance Comnputing Conference (LACC 2009). 2009 March, Patiala, India; 2009. P. 6-7.

[18] L.F. Bittencourt, R. Sakellariou and E.R.M. Madeira. Scheduling Using a Lookahead Variant of the Heterogeneous Earliest Finish Time Algorithm. 18th Euromicro International Conference on Parallel, Distributed and Network-Based Processing(PDP'10). 2009; P. 27-34.

[19] Hui Cheng. A High Efficient Task Scheduling Algorithm Based on Heterogeneous Multi-core processor. IEEE; 2010.

[20] Ahmed Zaki, Semar Shahul, and Oliver Sinnen. Scheduling task graphs optimally with A *. Supercomputing. 2010; 51: 310–332.

[21] Mahboobeh Houshmand et. al. Efficient Scheduling of Task Graphs to Multiprocessors Using a Combination of Modified Simulated Annealing and List based Scheduling. IEEE 3rd International Symposium on Intelligent Information Technology and Security Informatics; 2010.

[22] R. Eswari and S. Nickolas. A Level-wise Priority Based Task Scheduling for Heterogeneous Systems. International Journal of Information and Education Technology. Dec 2011; 1(5).

[23] Yan Kang and Defu Zhang. A Hybrid Genetic Scheduling Algorithm to Heterogeneous Distributed System. Scientific Research. 2012; 3: 750-754.

[24] Hamid Arabnejad and Jorge G. Barbosa. List Scheduling Algorithm for Heterogeneous Systems by an Optimistic Cost Table. 2014.