Extended List Based HEFT Scheduling using BGA in Multiprocessor System

Baldeep Singh, Priyanka Mehta

[1]*M.Tech. Student, UGI Lalru, PTU Jhalander, Punjab, India*
[2]*Assistant Professor, Dept. Of Comp. Sci. & Engg, UGI Lalru, PTU Jhalander, Punjab, India*

---***---

Abstract – *Scheduling computation tasks on heterogeneous processors is the key issue for an advanced computing. In spite of the fact that numerous scheduling heuristics have been demonstrated in the text. The present algorithms for diverse domains are not broad creative due to their high versatile quality and the quality of the result. A list based scheduling in multiprocessor system has constantly been a topic of conversation for the researchers due to its nature of solving high complexity scheduling problems and to estimate the additional time of the applied matrix. A lot of earlier research works have also included prioritization on the Jobs to reduce the computation cost and earliest finish time of the system. This paper introduce a complicated move toward which extends Heterogeneous Earliest Finish Time by performing Prioritization using bins greedy algorithm which evaluate the system on the basis of the positive weight , negative weight , positive threshold , negative threshold and a fitness function. The results are analyzed with help of various parameters namely: schedule length, speedup and efficiency.*

Key Words: *List Based Scheduling. HEFT, EST, EFT, SLR, BGA.*

1. Introduction

A range with different system resources which can be used for the execution of intensive applications and can be local or geographically distributed are heterogeneous systems. For executing parallel applications on the heterogeneous platform, the advantage from the parallel application scheduled methods used to taken. Because of the varied number of execution rates and the communication cost between the processors, the task scheduling problem become more difficult. DAG is a well-established illustration of set of tasks that known as an application with inter-task dependencies. The major goal of scheduling is divide an application into number of tasks and allocate onto best suitable processor to reduce the overall execution time. The problem for allocating the task to the best processor is NP-hard. The research efforts have focused on obtaining the low complexity heuristics for developing better schedules. Numbers of approaches have been proposed for solving the static scheduling problems. Most of them focused on homogenous processes. The target of scheduling is to decrease the execution time, by designating greatest number of resources to the executable framework. Scheduling problem can be of two types that are Static and Dynamic. In static scheduling, execution time of the job, assignment conditions are known ahead of time and it is done at assemble time. Static scheduling is also known as non preemptive, once an application is entered for scheduling that cannot stopped until execution is completes In dynamic Scheduling [6 10 22], the execution time of the assignment, undertaking conditions are known when necessary. Dynamic scheduling is also known as preemptive scheduling. Tasks are executed at the time of execution while they need. The main objective of dynamic scheduling is Minimization of the execution time of the assignment and scheduling overhead. The problem of scheduling following set of tasks to processor can be divided into categories- Job scheduling, Scheduling and mapping. Independent jobs are to be booked among the processors of a distributed registering framework to upgrade general framework execution in task Scheduling Framework and in Scheduling and Mapping; the assignment of various cooperating undertakings of a single parallel program to minimize the consummation time on the parallel PC framework is considered. Our main focus is on list scheduling algorithm which is presenting in this paper used in heterogeneous environment.

2. Related Work

In this area, we display a brief summary of task scheduling algorithms, mainly list based heuristics. We display their time difficult quality and their next to execution. Haluk Topcuoglu et.al [4] they present two novel scheduling algorithms for a delimited number of heterogeneous processors with an objective to concurrently meet high performance and fast scheduling time, which are called the Heterogeneous Earliest Finish Time (HEFT) algorithm and the Critical Path on a Processor (CPOP) algorithm. Hwang et.al optional that the heterogeneous soon EST complete time (HEFT) scheduling algorithm [8] appoint the scheduling task priority in light of the most prompt begin time of every single task. HEFT allocate a task to the processor which minimizes the task's begin time. Rewini et.al recommended that mapping heuristic (MH) [9] allots the task scheduling priority s in light of the static b-level of every task, which is the b-level without the communication fixed cost between task s. At that point, a task is assign to the processor which gives the most prompt begin time. Iverson et.al recommended that the level zed-min time (LMT) [10] appoints the task scheduling priority in two stages. An initially, it clusters the task s into characteristic levels in light of the topology of the DAG, and later in every level, the task with the most noteworthy priority is the one with the biggest execution cost. A task is dispensed to the processor which minimizes the total of the combined communication costs with the tasks in the past level and the task's execution cost. R. Eswari and S. Nickolas et.al [12], in this paper, an additional static scheduling algorithm is planned called expected conclusion time based scheduling (ECTS) algorithm, which is used to effectively plan application tasks on to the heterogeneous processors. The ECTS algorithm discovers the task succession for finishing by allocating priority and after those maps the selected task progression on to the processors. Mohammad I. Daoud, and Nawwaf Kharma et.al [13], In this paper, they show another new high performance scheduling algorithm, called the greatest dynamic critical path (LDCP) algorithm, for HeDCSs with a imperfect number of processors. The LDCP algorithm is a list-based scheduling algorithm that uses a new attribute to productively select tasks for scheduling in HeDCSs. The proficient choice of tasks empowers the LDCP algorithm to produce high-quality task schedules in a heterogeneous computing environment. The solution of the LDCP algorithm is contrast with two of the best existing scheduling algorithms for HeDCSs: the HEFT and DLS algorithms. The assessment study demonstrates that the LDCP algorithm beats the HEFT and DLS algorithms in terms of schedule speedup and length. Samia Ijaz et.al [14], a novel method has been accessible in the paper for the creative mapping of the DAG based applications. The methodology that considers the lower and upper restrictions for the begin time of the tasks. The algorithm is taken account on list development approach and has been compare with the well known list scheduling algorithms accessible in the literature.

Section 3 describes the scheduling problem, Section 3 represents the task model, Section 4 describes the proposed algorithm, Sections 5 shows the results and discussion and section 6 finishes off the conclusion and future scope.

3. Scheduling Problem

The problem presented in this paper is the static scheduling of a reserved application in a heterogeneous structure with P set of processors, V set of vertices, E set of edges between two vertices. Overall mathematically it can be explained as G = (V, E) where V is the set of vertices and E is the edge between two vertices. As said above, task scheduling can be separated into Static and Dynamic methodologies [8]. Dynamic scheduling is satisfactory for situation where the framework and task parameter are not known at compile time, which make choices to be made at runtime however with extra overhead. An example domain is a framework where clients submit jobs, whenever, to teach computing jobs. A dynamic algorithm is obliged on the basis that the workload is just known at runtime, similar to the status of every processor when new tasks arrive. As a consequence of this, a dynamic algorithm not ensures that it have all work essentials accessible among scheduling and can't promote in light of the entire workload. By separation, a static methodology can expand a schedule by allowing for all tasks freely of execution request or time in light of the fact that the schedule is created before execution start and present no operating cost at runtime.

The problem here is to design a fitness function which can minimize the EST and EFT of the provided set of vertices. Minimizing the EST and EFT would also result into an enhancement in the efficiency of the computation and increase in the speed of the processing.

4. Task Model

A parallel list is communication as DAG (Directed Acyclic Graph) and is established as:

G = (V, E)

Where V is a set of v nodes and each node vi є V represent an application task, which include its training that must be executed on the same machine. E is a situated of e coordinated edges between tasks, each e (i, j) є E represents node in the DAG demonstrates the amount of all tasks and they must be executing successively. Edges e in the DAG speaks to the connection messages and is spoken to as (n_1, n_n). The source node is called protector node or parent node. The sink node is called child node. The node with no section is called exit node [6]. In other words it can be stated that DAG is a set of vertices, its edges and the communication and computation cost of the system. DAG or any kind of such scheduling algorithm is used to reduce the overall processing cost if the tasks and the processors are running on the same environment. Latency time plays a crucial role in terms of edge selection between the nodes. At a time frame vi nodes needs to send data elements to v_j with the exception that if both the vertices v_i and v_j are assigned to the same processor then there must be priority model which can evaluate that which task would be executed first and which task would be executed later. Prioritization can be achieved in many ways like providing the priority on the basis of the communication or computation cost. Node with least communication cost would be executed first. Another method of prioritization is ranking through HEFT. Table 1 represents the computation cost between processors.

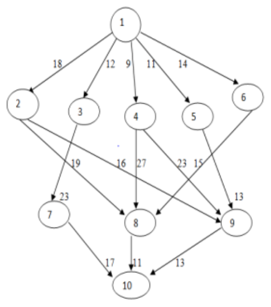

Fig-1: Task graph

P1	P2	P3
14	16	9
13	19	18
11	13	19
13	8	17
12	13	10
13	16	9
7	15	11
5	11	14
18	12	20
21	7	10

Table-1: Computation Cost

Weight w of every node communicates with the vertices to know the processing expenses w (n). W is a (v*q) defines computation cost matrix in each $w_{i,j}$ gives estimated execution time to complete application n_i on p_j processor. Average execution cost of an application n_i is describe as

$$w_i = \sum_{j==1}^{q} w_{i,j} / q \quad (1)$$

Transfer rate of data transfer among processors are stored in a matrix B, their size is q*q. Processors communication startup time is given in q_dimensional vector is L. their task n_i to task n_k, (which is scheduled onto p_m and p_n) is communication cost among edges, describe as

$$c_{i,k}=L+\{data_{(i,k)}/B_{(m,n)}\} \quad (2)$$

Both the tasks are assigning onto same processor, their interprocessor communication onto same processors is zero. The average communication cost between edges (i,j) is described as:

$$C_{i,k}=L+\{data_{(i,k)}/B\} \quad (3)$$

Where transfer rate onto processors average defined by B and average communication startup time defined by L. Now we describe the earliest startup time (EST (n_i, p_j)) AND earliest finish time (EFT (n_i, p_j)) of application n_i onto processor p_j, their entry task is

$$EST\ (n_{entry,}\ p_j) = 0 \quad (4)$$

In order to calculate the EFT of a task n_i, all immediate parent tasks of n_i must have been scheduled shown in (5) and (6).

$$EST\ (n_{i,j})=\max\{avail[j],\ (\max/(n_{m\epsilon pred(ni)})(AFT(n_m)+c_{m,i}))\} \quad (5)$$

$$EFT\ (n_i,\ p_j) = w_{i,j} +EST(n_{i,}\ p_j) \quad (6)$$

Where pred(n_i) is the list of immediate parent applications of task n_i and avail[j] is the minimum time which the p_j is ready for execution. This process is not executes in case of all the tasks are not assigned to processors. After the scheduling of all tasks onto available processors, they calculate their EST and EFT i.e. equal to the actual start time (AST) and actual completion time (ACT), or we can say overall completion time (n_{exit}), i.e. schedule length or makespan, sometimes called critical path length. The makespan is describe as

$$Makespan = \max\ \{AFT\ (n_{exit})\} \quad (7)$$

Node with higher need is analyzed for Scheduling before a node with lower need. The main objective of scheduling problem is to assigning the tasks of a given application for execution among suitable processors and tries to minimize the schedule length or makespan.

5. Proposed Algorithm

BGA algorithm uses HEFT algorithm strategy for prioritization and selection. The HEFT Algorithm [8] is an application scheduling algorithm for a limited number of heterogeneous processors. The algorithm first builds up a need outline of procedures and a short time later basically perfect project decisions for each node are made on the reason of the task's evaluated fulfillment time. Like the vast majority of list-based scheduling algorithm it has three stages.

Task prioritization stage

Task selection stage

Processor allocation stage

In task prioritization stage, it utilized $rank_u$ to transfer need to the task in this phase rank is finding out according to their priority. $Rank_u$ is the upward rank computes rank of all tasks by using mean communication and computation cost and then computes the priority by using sum of upward and downward rank. In task selection stage task are arranged by their priority of each node. Which node has highest priority, that task must be toped in the rank list. For batter utilization the behavior and specification should be same for e.g. large applications are not suitable for small processor. In our algorithm we use a fitness function which uses values at their own choice and give a value to each node and set the priority according to that number. In processor selection stage, the algorithm selects the task with most important need from the prepared list as the selected task. Also, the processor selection stage, the algorithm selects in processor that permits the EFT (Earliest Finish Time) of the selected task according to their utilization of processes. However, the HEFT algorithm uses an insertion policy that try to find to bring a task in an earliest idle time slot between two previously listed tasks on a processor, if the slot is large enough to hold the task. The goal of useful Scheduling is to guide the accomplishments onto the center processors and execution of applications is situated so that task priority requirements are fulfilled and minimum amount of schedule length is given.

The BGA algorithm first computes normal execution time for every node and normal correspondence time between edges of two dynamic nodes. That execution time of the task is evaluated before finding the rank, in this algorithm the scheduling priority is based on upward and downward scheme of a task ni is defined as

$$Rank_u(n_i)=W+\max_{nj\epsilon succ(ni)}\ (C_{ij})+(rank_u\ (n_j)) \quad (8)$$

C_{ij} is the edge of average communication cost of (i,j), n_i is the immediate child defined by succ (n_i) computation cost is defined by (w_i). In case of calculating the rank, which starts from exit node n_{exit} i.e. upward rank value given below

$$Rank_u\ (n_{exit}) = (w_{exit}) \quad (9)$$

In case of evaluate the downward rank of a task ni is recursively clear by

$$Rankd(n_i) = \max_{nj\epsilon pred(n\_i)}\ \{rank_d\ (n_j)+(C_{(i,j)})\} \quad (10)$$

Where set of immediate parents of task ni is pred(n_i). Traversing the task graph according to downward rank which is starting from entry node n_{entry}, that rank value is equal to zero. $Rank_d$ (n_i) is the critical path of the task graph, i.e. maximum traversing time from entry node to task n_i.

The algorithm then sorts the task by diminishing request of their rank qualities. The endeavor with higher rank worth is given higher need. In the task determination stage, nodes are planned by needs and every task is allocated to the asset that can complete the undertaking at the soonest time [14].

The BGA algorithm has to be designed in such a manner that it overcomes the problems of the algorithm and reduces the computation and communication cost of the system. The major problem of HEFT is prioritization which is made on the base of the communication edge between two vertices. The

BGA algorithm introduces a fitness function on the basis of the computation cost as well as the positive threshold, negative threshold, positive weight and negative weight of the system which is explained in the pseudo code in the later sections. There can be many different greedy strategies for the same problem. Which one is the best usually depends on the application. In some rare cases, analysis may also show that a strategy is better than another one in all cases. Here Greedy algorithm has been used.

Many algorithms obtain a solution to a problem by making a sequence of choices. Greedy algorithms make at each point the choice that seems best at the moment. Such heuristic strategies do not always produce an optimal solution, but sometimes they do. Sometimes computing an optimal solution requires too many resources, and greedy heuristics can provide a reasonable solution in reasonable time. There can be many different greedy heuristics for the same problem. The proposed algorithm enhances the HEFT algorithm using the BGA Algorithm. The algorithm starts when the HEFT is done with its prioritization. The main aspects of this algorithm are as follows.

```
Start
Draw a graph G (V,E) where V is the vertex and E is the edge of two vertices.
Compute $rank_u$ for all nodes traversing upward.
Compute rank $n_d$ for all nodes by traversing downward.
$Cp=rank\ k_d(ni) + rank_u\ (n_i)$   (computes the priority for each task)
Initialize fitness function to minimize
$\sum n,e$ EST, EFT
$F(v_{i,k})=\{(\sum v_j € A_k\ w_{ij}+)\ Q_j+ (\sum -v_j€A_k\ w_{ij})\ Q\ j-\}/(Qj+ + Q\ j-)$
While there is any unscheduled node in the queue
If $(f(v_i, k_i)) >$ If $(f(v_i, k_{i+1}))$
Process the task
Else assign If $(f\ (v_i, k_i))$ as max value and repeat fitness function computation for all tasks in the queue.
End while
Compute EST and EFT for deciding number of iterations.
Find minimum of EST and EFT iteration.
End
```

The above pseudo code represents the BGA algorithm through which the optimization has been done. The abbreviations have been provided already. Each node would be assigned with a positive weight, a negative weight, a positive threshold and a negative threshold. The systems would be also assigned with positive and negative threshold weight whose fitness function would be calculated at each iteration. If the calculated fitness function value comes out to be positive then, it would be assigned with the highest priority at the first time and further on if the next value of the fitness function is greater than that of the previous one, the swapping would be performed that means the sorted values would arranged in descending order. The sum of all positive thresholds would be subtracted from the EFT and the sum of all negative values of the fitness function would be added to the EFT. In such a manner the following results have been computed. And used metrics has been defined below: The BGA algorithm searches for the best solution each and every time the iteration takes place. The best possible solution is always evaluated using a fitness function which has been already described in the pseudo code. The BGA algorithm always evaluates the results in phases in which the output of one phase goes as the input to other phase. The phases are generation of positive weight, generation of negative wt, generation of positive threshold, generation of negative threshold and evaluation of the fitness function. In figure 5.1 represents a DAG, which contains 10 nodes each node is connected with vertices and shows us dependency between parent nodes with child node. Parent nodes are executed before child nodes. In DAG represent communication cost between to edges.

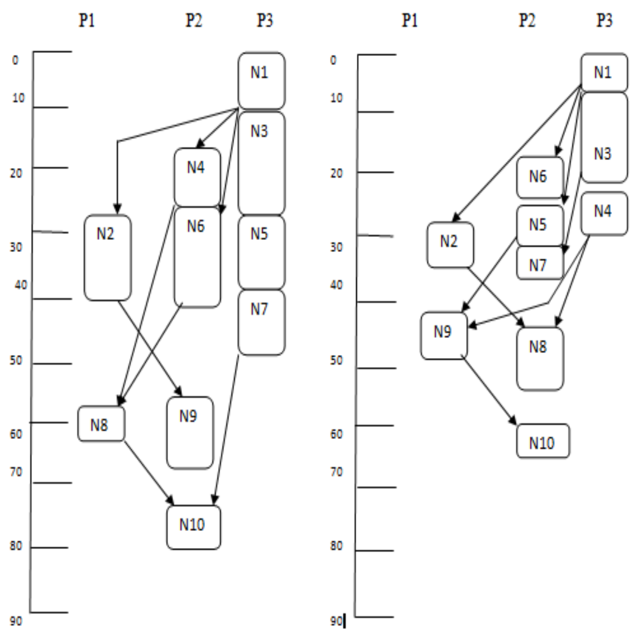

Fig- 2: Scheduled Tasks on Processors

A schedule is produced shown in Fig-2 by using communication cost and computation cost, which represent the maximum finish time of all processes or called schedule length ratio (SLR) or makespan.

6. Result and Discussion

In this section we present the comparative evolution of our algorithm and compared with HEFT algorithm. Which represent from different aspects like schedule length according to change in result and in efficiency and their schedule length.

6.1 Comparison Metrics: We compare algorithms by these three metrics:

- Speedup

The speedup value for a given graph is computed by dividing the sequential execution time by the parallel execution time. It is defines as:

Speedup = $\min_{pj \in Q} \{\sum_{ni \in V} Wij\}$ / make span

- Efficiency

It is defined as speedup divided with number of processors in each run.

- Schedule Length Ratio

Schedule length is the completion time of a DAG or called makespan.

Makespan = max {AFT (n_{exit})}

6.2 Performance results

The performance of the BGA is compared with well known scheduling algorithm namely HEFT algorithm for heterogeneous system using task graphs on various performance metrics as described previously. Different number of graphs is generated with varying task sizes from 18 to 22. The available processor in each case was taken to be three. Chart-1 shows that the BGA algorithm is better than HEFT algorithm by 6.65% in term of average schedule length (ASL). Figure 4.3 shows the BGA is better than HEFT algorithm by 5.73% in terms of average speedup. Figure 4.4 shows the BGA gives better results as compared with HEFT algorithm in terms of average efficiency by 4.6%. Figure 4.5 gives the improvement of BGA then HEFT algorithm in terms of varying CCR values. It shows that schedule length increases when number of nodes increases.

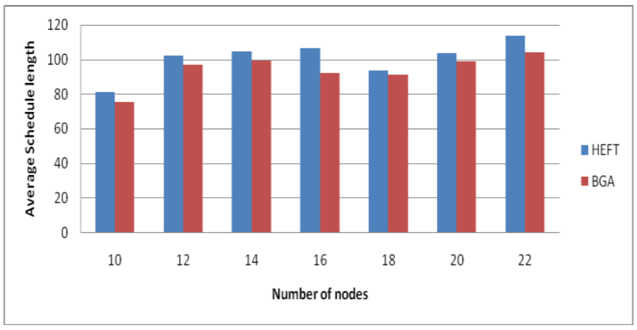

Chart-1: ASL with respect to number of nodes

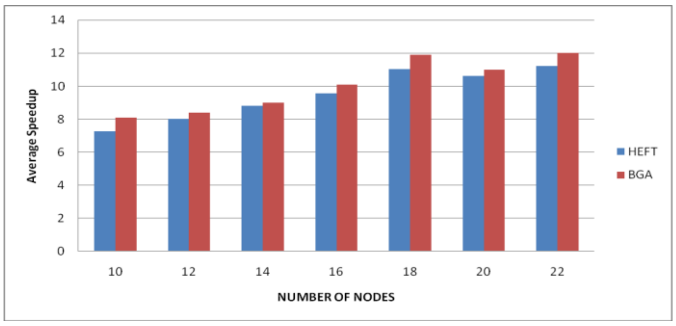

Chart-2: Average speedup with respect to number of nodes

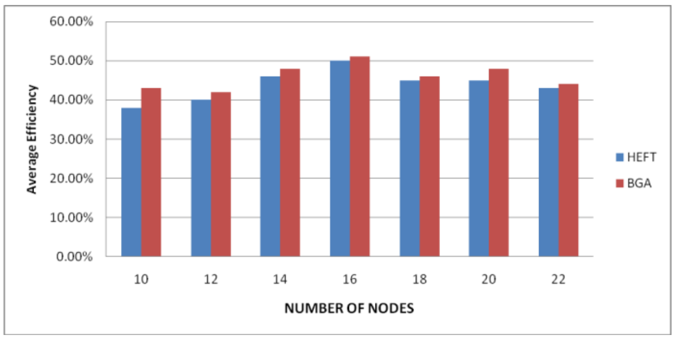

Chart- 3: Average efficiency with respect to number of nodes

7. Conclusion and Future Scope

In this paper a hybrid algorithm has been presented called BGA which combines two algorithms namely HEFT and BGA Algorithm for the scheduling applications graphs in Heterogeneous processes .The advantages of HEFT have been utilized to generate a new algorithm and the drawbacks have been removed by BGA algorithm. The results have been evaluated on the basis of 4 parameters namely Speed Up, Schedule Length, Efficiency and Communication cost ratio. The proposed algorithm shows a significant improvement in terms of cost cutting due to highly effective designed fitness function whose description is provided in the pseudo code and the performance is compared with the traditional HEFT Algorithm. The current presented work opens up a lot of future gates for the upcoming research workers. The proposed algorithm has not been tested with the Directed acyclic graphs which can be a point of interest. The algorithm has not been tested with more than 22 nodes which can be tested. Introduction to genetic algorithm in this contrast can also become an interesting aspect to check whether it increases the speed up and efficiency of the system.

REFERENCES

[1] Garey, M. R. e D. S., "A Guide to the Theory of NP-Completeness", W. H. Freeman & Co, New York, NY, USA , 1979.

[2] Yang, T. and Gerasoulis, "A DSC Scheduling parallel tasks on an unbounded number of processors", IEEE Transaction Parallel Distributed System, 5(9), pp:951–967, 1994.

[3] Ilavarasan, E. and Thambidurai, "Low Complexity Performance Effective Task Scheduling Algorithm for Heterogeneous Computing Environments", Journal of Computer Sciences 3 (2), pp: 94- 103, 2007.

[4] Topcuoglu, H. Hariri, S. and Wu, M.Y. "Performance-effective and low-complexity task scheduling for heterogeneous computing", IEEE Trans. Parallel Distributed System, 13(3), pp: 260–274, 2002.

[5] Adam, T. L. Chandy K. M. and Dickson J. R. "A comparison of list schedules for parallel processing systems", Communication ACM, 17(12), pp: 685–690, 1974.

[6] Kwok, Y.K. and Ahmad, I., "Benchmarking and comparison of the task graph scheduling algorithms", J. Parallel Distrib. Comput, 59(3), pp: 381–422, 1999.

[7] Liu, G. Q. Poh, K. L. and Xie, M. "Iterative list scheduling for heterogeneous computing", J. Parallel Distrib. Comput, 65(5), pp: 654–665, 2005.

[8] J.-J. Hwang, Y.-C. Chow, F. D. Anger, and C.-Y. Lee, "Scheduling precedence graphs in systems with interprocessor communication times," SIAM Journal on Computing, vol. 18, no. 2, pp. 244–257, 1989.

[9] M. Y. Wu and D. D. Gajski, "Hypertool a programming aid for message-passing systems," IEEE Transactions on Parallel and Distributed Systems, vol. 1, no. 3, pp. 330–343, 1990.

[10] G. C. Sih and E. A. Lee, "Compile-time scheduling heuristic for interconnection-constrained heterogeneous processor architectures," IEEE Transactions on Parallel and Distributed Systems, vol. 4, no. 2, pp. 175–187, 1993.

[11] H. El-Rewini and T. G. Lewis, "Scheduling parallel program tasks onto arbitrary target machines,"Journal of Parallel and Distributed Computing, vol. 9, no. 2, pp. 138–153, 1990.

[12] M. Iverson, F. Özgüner, and G. Follen, "Parallelizing existing applications in a distributed heterogeneous environment," in Proceedings of the IEEE International Conference on Heterogeneous Computing Workshop (HCW '95), pp. 93–100, 1995.

[13] E. Ilavarasan, P. Thambidurai, and R. Mahilmannan, "High performance task scheduling algorithm for heterogeneous computing system", Volume 3719 of Lecture Notes in Computer Science, pp.193-203. Springer, 2005.

[14] Luiz F. Bittencourt, Rizos Sakellariou and Edmundo R. M. Madeira," DAG Scheduling Using a Lookahead Variant of the Heterogeneous Earliest Finish Time Algorithm", 18th Euromicro Conference on Parallel Distributed and Network-based Processing, pp. 27-34, 2010.

[15] Savina Bansal, Padam Kumar and Kuldip Singh, "Dealing with heterogeneity through limited duplication for scheduling precedence constrained task graphs", J. Parallel Distrib. Comput, vol. 65, pp. 479 – 491, 2005).

[16] Tomasz Kalinowski , Iskander Kort and Denis Trystram ,"List scheduling of general task graphs under LogP" ,Parallel Computing, vol.26, pp. 1109-1128, 2000.

[17] R. Eswari and S. Nickolas, "A Level-wise Priority Based Task Scheduling for Heterogeneous Systems" International Journal of Information and Education Technology, Vol. 1, No. 5, pp. 371-375, December 2011.

[19] Mohammad I. Daoud and Nawwaf Kharma, "A high performance algorithm for static task scheduling in heterogeneous distributed computing systems" J. Parallel Distrib. Comput, 68, pp. 399 – 409, 2008.

[20] Samia Ijaz, Ehsan Ullah Munir, Waqas Anwar, and Wasif Nasir, " Efficient Scheduling Strategy for Task Graphs in Heterogeneous Computing Environment" The International Arab Journal of Information Technology, Vol. 10, No. 5, pp. 486-492, September 2013.