A Heterogeneous Static Hierarchical Expected Completion Time Based Scheduling Algorithm in Multiprocessor System

Sukhjit Singh, Nirmal Kaur

Sukhjit Singh, M.E. Student, UIET, Panjab University, Chandigarh, India
Nirmal Kaur, Assistant Professor, Dept. of Comp. Sc. & Engg, UIET, Panjab University, Chandigarh, India

---***---

Abstract - *With the advent of heterogeneous systems it is evident that achieving high performance and better resource utilization is impossible without optimal scheduling of the tasks. Keeping in view the various issues in performance, previously many task scheduling algorithms have been proposed in order to achieve optimal scheduling of tasks. This paper makes an effort in list scheduling heuristics by proposing an algorithm called level based scheduling. Simulated results of proposed algorithm have been analyzed and compared with existing list based scheduling heuristics in terms of speedup, efficiency and schedule length.*

Key words: *Directed acyclic graphs, List based scheduling, Heterogeneous environment, Task scheduling.*

1. INTRODUCTION

Heterogeneous distributed computing includes resources of varying capacity forming a fast system to execute computationally intensive, parallel and simultaneous applications. One of the major issues in heterogeneous distributed computing is to schedule the tasks of requests such that the general running time is minimized. Task scheduling problem has been known to be NP-complete [1, 2]. Many heuristics have been planned in the literature for task scheduling problem as there is no accurate solution for NP-completeness. Task scheduling is broadly classified into static scheduling and dynamic scheduling. In static scheduling [1, 2, 4, 14] all the data associated with a parallel program, for example: task managing time, communication time and data dependency are already available. In dynamic scheduling [8, 15] the choices of task scheduling is done at run time. Hence the target of dynamic scheduling is to schedule tasks and minimize the scheduling overhead. Static task scheduling have been based on various scheduling heuristics namely: List scheduling algorithms [1-4, 6, 12, 14], Duplication based algorithms [7, 10], Cluster based algorithms [13] and Random guided search algorithms [4]. In list based heuristics, tasks are placed in a priority list with every task having a unique and special priority. A task with highest priority from the priority list is scheduled onto a suitable processor by using task's LBT (Least Beginning Time) and LCT (Least Completion Time). Clustering heuristics were mostly planned for standardized frameworks and the point is to frame clusters of tasks and then scheduled onto processors. Clustering is the best way to minimize the communication delay within DAG's (directed acyclic graph) by grouping closely related tasks within a cluster on the same processor. Duplication is very useful strategy in parallel processing to reduce the communication overhead by duplicating some task on more than one processor. The duplication heuristics deliver the shortest make-span in all the heuristics, yet they have a disadvantage i.e. the higher time complexity. Guided random scheduling techniques [16] make use of the theory of evolution and ordinary genetics to produce near optimal task schedules.

2. RELATED WORK

List based scheduling has been used by number of researchers for scheduling purposes. Some well known scheduling algorithms are Critical Path on Processor (CPOP) [1], Heterogeneous Earliest Finish Time (HEFT) [1, 9], Expected Completion Time Based Scheduling (ECTS) [14] and Performance Effective Task Scheduling (PETS) [3]. Eswri et al. [14] Expected Completion Time Based Scheduling is a static task scheduling algorithm. It is used to effectively schedule application tasks on to heterogeneous processor. The algorithm finds the sequence of execution by calculating the priorities of tasks on each level, then assign these tasks onto appropriate processors. The priority of each task is calculated with the help of expected completion time (ECT), which is calculated by using the average computation time and maximum communication cost. Haluk et al. [1] presented two novel algorithms dealing with bounded number of heterogeneous processors having a high performance and low scheduling overhead. The algorithms proposed were critical path on processor (CPOP) and heterogeneous earliest finish time (HEFT). The HEFT algorithm used upward rank method for generating a priority list and assigned these tasks onto suitable processors in a way that the earliest finish time is minimized. On the other hand the second algorithm proposed CPOP uses a summation of upward and downward ranks to estimate the priorities of the tasks. Mohammad et al. [5] betterment in the heterogeneous distributed computing system (HeDCS) has been provide by the longest dynamic critical path algorithm, with bounded number of processors. The LDCP is a list based scheduling algorithm it works on key tasks are identified and scheduled based on the minimum schedule length generated. The algorithm has three phases the task selection phase, processor selection phase and the status update phase. Time complexity of the LDCP algorithm is of O (m*n3) where m is number of processors and n is the number of tasks. The comparative study showed that the LDCP algorithm is better than the HEFT and LDS algorithm. Hamid et al. [16] proposed a novel list based scheduling algorithm for heterogeneous computing environment. The algorithm is called predict earliest finish time (PEFT). The algorithm has a complexity comparable to most of the existing state of art algorithms but offers a better makespan. This has been achieved by using a feature called lookahead which helps to calculate the optimistic cost table (OCT) without increasing the complexity. The algorithm is based on the OCT and is used for both task selection and processor assignment. There are three phases to it namely optimistic cost table calculation, task prioritization phase and processor selection phase. Remainder of the paper is organized as follows: Section 3 illustrates the Models to be followed. Section 4 describes the proposed algorithm. Section 5 represents the simulation results and analysis Section 6 shows the conclusion and future scope.

3. Models

3.1 Task model

A parallel application can be broken into a number of tasks having data dependencies among them. This configuration can be represented using a directed acyclic graph (DAG), G= (X, Y), where X is the set of 'n' tasks and Y is the set of 'e' edges representing relations between these tasks. Each edge y between task t_i and t_j ϵ Y signifies a hierarchy of execution implying that task t_j is executed after task t_i. A parent-less task is referred as entry task (t_{entry}) and a child-less task is called as exit task (t_{exit}). Each edge y between the tasks (t_i,t_j) has a value assigned to it representing the cost of communication. Execution of a particular task can start on a processor only when all the tasks above it in hierarchy have been executed and the data from them is available on the processor.

In order to set an agreement a few assumptions are made:

1. There are no conflicts between inter-processor communications.
2. Communication can be parallel to the computation.
3. The execution of tasks in an application is non-preemptive.

   The cost of communication $c_{i,j}$ is incurred when a task t_i running on a processor p_m needs to transfer β_{ij} units of data to a task t_j running on a processor p_n and can be defined as:

   $c_{i\,j}= I_m + U_{(m,n)} * \beta_{ij}$ (1)

Where I_m is the time taken by p_m to start communication, β_{ij} is the amount of data transferred from tasks t_i and t_j also called bandwidth, $U_{m,n}$ is the time taken to transfer each unit of data from p_m to p_n.

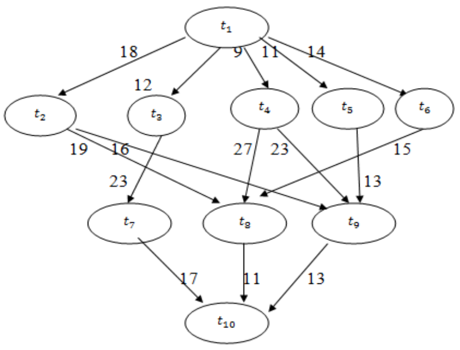

Fig -1: Directed Acyclic Graph

Fig 1 shows the one instance of DAG with 10 tasks and dependency in the form of precedence constraints. The average communication cost of sending data from the task t_i to t_j can be defined as

$C_{ij}=\bar{O}+\bar{U}*\beta_{ij}$ (2)

Where $\bar{O}$ is average communication startup time and $\bar{U}$ is the average transfer rate over all processors. If both the tasks are on the same processors then $c_{ij}=0$ as inter-processor communication time is negligible. The least beginning time (LBT) is the earliest time at which the processing of a task can start and least completion time (LCT) is the least time at which the processing can be completed. They are represented as LBT (t_i,p_j) and LCT (t_i,p_j) representing the least beginning time and least completion time for processing of the task ti on the processor pj respectively.

$LBT\ (t_{entry},p_j)=0;$ (3)
$LBT(t_i,p_j)=\max\{avail[j], \max_{t_k \epsilon pred(t_i)} (ACT(t_k)+c_{k,i}\ )\}$ (4)

Where ACT is the actual completion time of a task t_k on the processor p_j, avail[j] is the time when processor p_j becomes available and it is ready to execute task t_i. In order to calculate LCT of a task t_i, all the tasks above the current task t_i must have been scheduled.

$LCT\ (t_i,p_j\ )=h\ (t_i,t_j)+LBT(t_i,p_j)$ (5)

Where h(t_i, t_j) is the computation cost of the task t_i on processor p_j. Once the all the tasks in the given graph are scheduled, the schedule length (total execution time) will be ACT of the exit task t_{exit}, the schedule length also called makespan is defined as
$makespan =\max\ (ACT\ (t_{exit}\ ))$ (6)

3.2 MACHINE MODEL

The machines which run in a heterogeneous environment consist of processing nodes. The computing system is represented by set P= {p_i | p_i ϵ P, i = 1, 2, 3..., |P|} is the set of fully connected and bounded processors in heterogeneous environment, the computation cost of different tasks can vary among the processors due to heterogeneity as shown in Table 1.

Table -1: Computation cost matrix

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

Scheduling of tasks is considered to be non-pre-emptive and also the communication overhead on two tasks scheduled on same processor is considered zero. Once a task is scheduled then processor forwards its output data to its child tasks. The aim here is to provide a solution to the scheduling problem such that it produces minimum schedule length.

4. PROPOSED ALGORITHM

The proposed hierarchical expected completion time based scheduling (HECTS) algorithm is described in fig 2. The algorithm completes into two phases. First phase deals with the prioritization of the tasks based on levelized priority and the second phase deals with the processor selection i.e. task scheduling.

4.1. Priority Based Task sequence generation.

This phase generates a list of tasks ordered on the basis of the priorities assigned to them. The basis of an optimal solution is the sequence of tasks generated. The current phase is divided into two parts.

A. Task prioritization:

This stage deals with the priority computation to each task according to their level in the task graph. The priority of a task is based on the mean computation cost (MCC) and the maximum time taken for the data to arrive at that particular task (MDTC) i.e. Maximum Data Transfer cost.
The MCC can be defined as follows:
It is the sum of the computation costs of the tasks on each processor divided by the number of processors.
MCC (t_i) = $\sum_{j=1}^{p} h(t_i, t_j)/P$ (7)

Here h (t_i, t_j) is the estimated execution time to compute the task t_i on the processor p_j. In addition to this, there can be more than one parent to a task and the data from different parents might reach the current processor at different times. Thus the completion time for a task will also depend on the maximum cost of data arrival on the current processor from its parent task in addition to MCC. MDTC can be defined as follows.

MDTC (tj)=$\max_{t_i \,\epsilon\, pred(t_j)}$ ($C_{i,j}$) (8)

(For a DAG having n tasks and e edge the MDTC of a task (t_j) is the maximum time that a task spends to receive data from all its parents.)
Here it is assumed that the data arrival cost for the first task at level one is '0' because of the fact that it is an entry task. The tasks which have a single parent will have the MDTC equal to the time taken by the data to reach the current task from the parent task. Whereas, if the task has multiple parents the maximum of the data arrival costs from the parents is considered the MDTC of the task. A prioritization key value (PKV) is calculated to assign the priorities that will decide the task scheduling. PKV's defined as time taken for a task to be ready for processing based on computed values of MCC and MDTC values. The PKV is calculated by adding the MCC of the parent task to the MDTC of the current task and MCC of current task.

PKV(t_i)=$\max_{t_j \epsilon pred(t_i)}${MCC($t_j$)}+MCC($t_i$)+MDTC($t_i$) (9)

B. Task Selection:

This is the second step of task prioritization phase dealing with the setting of priorities for the different tasks. In this stage all tasks are sorted in a non-increasing order for the PKV values which were calculated in the previous stage and priorities are assigned to them. Thus the task with higher PKV is ranked higher in the priority list.

Table -2: Priority computation for given application

Number of tasks	Task Level	PKV	priority
1	1	13	1
2	2	47.67	1
3	2	39.33	3
4	2	34.67	5
5	2	35.67	4
6	2	39.67	2
7	3	48.33	3
8	3	53.67	2
9	3	56.33	1
10	4	48.34	1

Table 2 shows the level wise computed PKV corresponding to different tasks and the priorities which have been assigned to the tasks.

```
Input a graph with their attributes (tasks and vertices with communication costs).
While unscheduled tasks in the DAG do
        Graph is portioned into number of levels L1, L2… Ln. (based on priorities)
        For level $L_i$ in the graph do
                For nodes $t_i$ in the level $L_i$ do
                        Compute MCC, MDTC and PKV
                End for
            Arrange all nodes in decreasing order of PKV, assign priority for each task set into a list.
            Pick task from the level according to their priority.
        End for
        For each processor ($p_k$) in the processor set do
            Compute EFT of $t_j$ on $p_K$.
        End for
End while
Schedule tasks $t_i$ on processor $p_k$ that gives minimum ACT.
```

Fig-2: HECTS algorithm

4.2. Processor selection

Once the priorities for the different tasks are set, the focus can shift to the assignment of tasks to a certain processor based on insertion scheduling. The insertion based policy works by considering the time slots between two already assigned tasks and assigns the tasks in those slots if possible, without violating the precedence constraints of the tasks in the given graph. The ideal time slot can be calculated by subtracting the finish time from the starting time of the two tasks assigned. Thus a time slot is usable for current task if it satisfies the precedence constraints specified in the task graph and is greater or equal to the computation time required for the current task. Starting time for search can be set the time t_i, i.e. when the current task is ready with all the data on the current processor. It continues till it finds a time slot long enough to accommodate the current task. In-case no ideal slot is found in-between, insertion based policy inserts the current task after the last task scheduled on that processor.

Using above mentioned processor selection strategy all the tasks in the given graph are assigned to the respective processors. Table 3 shows the actual mapping of tasks to their corresponding processors.

Table -3: Scheduled Processors

Selected tasks	Scheduled processors
T1	P3
T2	P3
T3	P1
T4	P2
T5	P2
T6	P3
T7	P1
T8	P2
T9	P2
T10	P2

5. Simulation results and Analysis:

Using the experimental setup discussed earlier, a number of DAGs representing various task models were scheduled to test the proposed algorithm. The results were recorded and measured according to different matrices explained below:

5.1. Performance metrics:

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

Speedup= $(\min p_j \in P\{\sum t_i \in X\ h(t_i,t_j)\}/\text{makespan}$

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

Schedule Length: It is the completion time of an exit task in the given DAG. It is also called overall makespan.

makespan=max(ACT (t_{exit}))

5.2. Comparative Analysis

Schedule generated by the HECTS algorithm are compared against CPOP algorithm shown in fig 3. The DAG example as shown in fig 1 and its computation cost shown in Table-1 is used for results. The makespan of HECTS algorithm is 73, which is smaller than the makespan of CPOP algorithm which is 86.

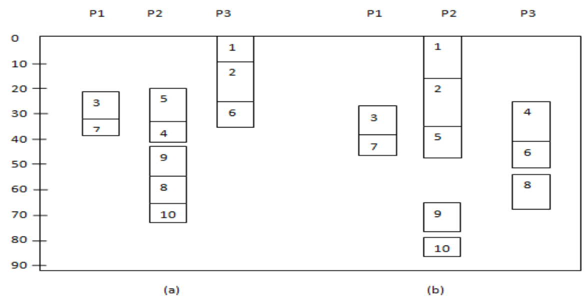

Fig -3: Schedules generated by (a) HECTS and (b) CPOP

5.3Performance comparison:

The performance comparison of HECTS is done with CPOP on various performance matrices as described previously. Fifty different graphs are generated with varying number of task sizes from 8 to 26 in increments of 2 and CCR vary from 1to 2. The available processors in each case were taken to be three and can be applied to any number of processors.

Chart 1 shows the comparison of HECTS with the CPOP algorithm based on average schedule length (ASL). The Chart shows how ASL varies for CPOP and HECTS as the number of tasks increases. It is observed that schedule length increases as number of tasks increases in term of ASL. HECTS algorithm is better than CPOP algorithm by 7.36% and 5.28% when number of tasks is 8 and 26.

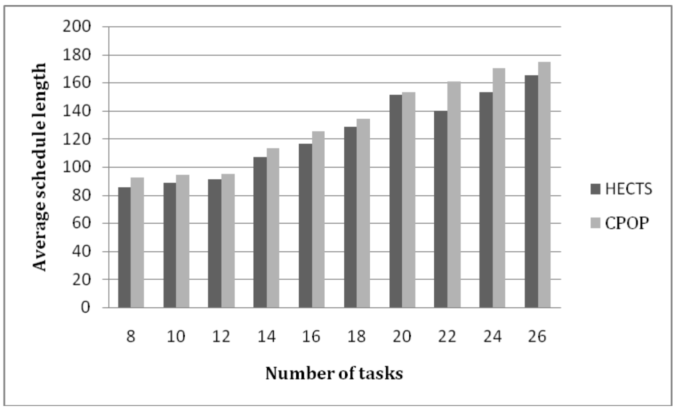

Chart -1: Average schedule length for varying task sizes.

Chart 2 shows the variation of average speedup with the increase in number of tasks. HECTS is observed to be better than that of CPOP algorithm by 5% and 5.75% when number of tasks is 8 and 26. Chart 3 shows the comparison of HECTS with CPOP algorithm based on average efficiency. HECTS algorithm is better than CPOP algorithm by 2.69% and 5.85% when number of tasks is 8 and 26.

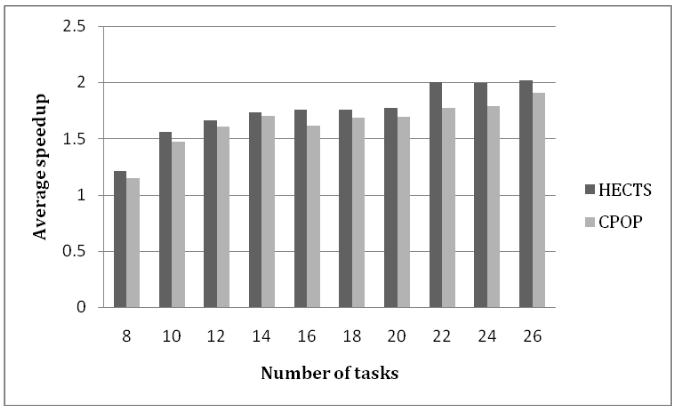

Chart -2: Average speedup for varying task sizes.

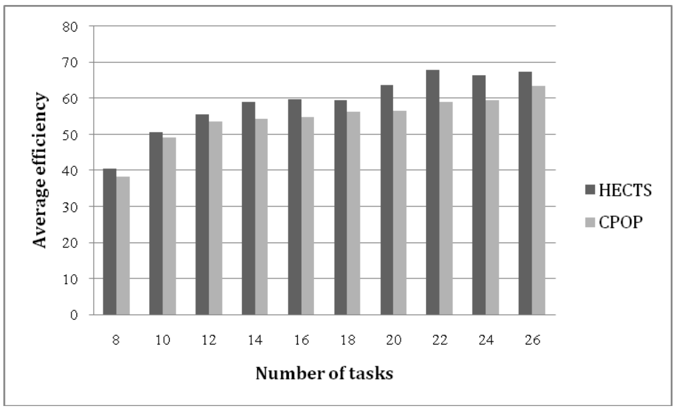

Chart -3: Average efficiency for varying task sizes.

Chart 4 shows how the ASL changes when CCR is varied from 1 to 2. It shows the HECTS algorithm is better than CPOP algorithm by 5.91% and 9.22% when CCR value is 1 and 2. Chart 5 shows how the average speedup changes when CCR is varied from 1 to 2. It shows the HECTS algorithm is better than CPOP algorithm by 3.64% and 6.63% when CCR value is 1 and 2.

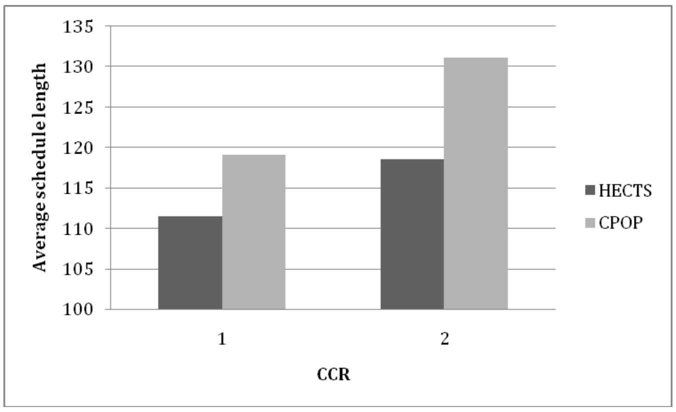

Chart -4: Average schedule length for varying CCR values.

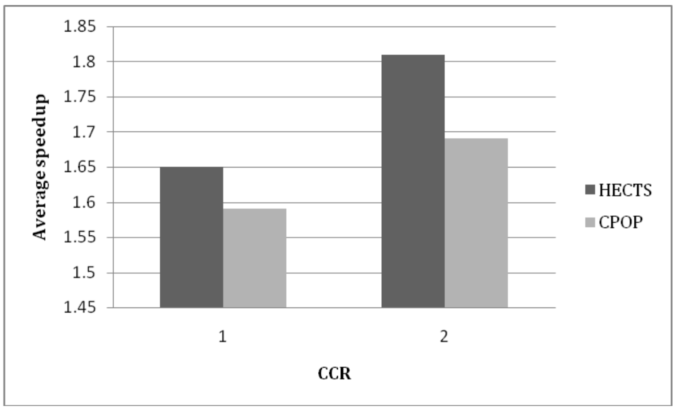

Chart -5: Average speedup for varying CCR values.

Chart 6 shows how the average efficiency changes. It shows the HECTS algorithm is better than CPOP algorithm by 1.92% and 5.59% when CCR value is 1 and 2.

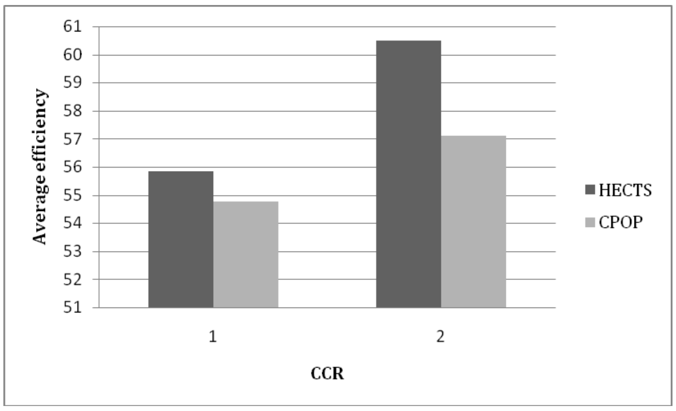

Chart -6: Average efficiency for varying CCR values.

6. Conclusion

The paper presents a new scheduling technique for the non-preemptive heterogeneous systems. The algorithm tries to minimize the schedule length for a given application. For the purpose of empirical evaluation three parameters are used namely: schedule length, speedup and efficiency. The comparative study shows HECTS algorithm is better than CPOP algorithm.

REFERENCES

[1] Haluk Topcuoglu,Salim Hariri, "Performance Effective and Low Complexity Task Scheduling for heterogeneous Computing," IEEE transactions and distributed systems, Vol.13, No.3, pp:1045-9219 March2002.
[2] Hui, C.C. and S.T. Chanson, "Allocating task interaction graphs to processors in heterogeneous networks" IEEE Trans. Parallel and Distributed Systems, Vol. 8, pp: 908-926, 1997.
[3] E. Illavarsan and P. Thambidurai, "Low complexity performance effective task scheduling algorithm for heterogeneous computing environment" Journal of computer Sci. Vol. 3(2). pp: 94-103, February 2007.
[4] Bajaj, R and D.P. Agrawal, "Improving scheduling of tasks in a heterogeneous environments" IEEE Trans. on Parallel and Distributed Systems, vol.15 pp: 107-118, 2004.
[5] Mohammad I. Daoud, Nawwaf Kharma, "A high performance algorithm for static task scheduling in heterogeneous distributed computing system", J. Parallel Distrib. Comput., 68, pp: 399-409, 2008.
[6] Wahid Nasri and Wafa Nafti, "A New DAG Scheduling Algorithm for Heterogeneous Platforms," 2nd IEEE International Conference on Parallel Distributed and Grid Computing, 2012.
[7] S. Ranaweera and D.P. Agrawal, "task duplication based scheduling algorithm for heterogeneous systems," Proc. International parallel and Distributed Processing symposium, pp: 445-450, 2000.
[8] D.I. George and G.J. Joyce Mary, "A new DAG based Dynamic Task Scheduling Algorithm (DYTAS) for Multiprocessor Systems" International Journal of Computer Applications, Vol.19, April 2011.
[9] Karan R. Shetti et.al, "Optimization of the HEFT algorithm for a CPU-GPU environment," International Conference on Parallel and Distributed Computing, Applications and Technologies, 2013.
[10] A. Agarwal and P. Kumar, "Ecnomical duplication based task scheduling for heterogeneous computing system," IEEE International conf. in parallel Processing, pp: 6-7, 2009.
[11] Guoqi Xie et.al, "A High-performance DAG Task Scheduling Algorithm for Heterogeneous networked Embedded Systems," IEEE 28th International Conference on Advanced Information Networking and Applications, 2014.
[12] Rashmi Bajaj and DP Agrawal "Improving scheduling of tasks in a heterogeneous environment", IEEE Transactions on Parallel and Distributed Systems, vol.15 (2), pp: 107-118, 2004.
[13] S.C. Kim and S. Lee, "Push-pull Guided Search DAG scheduling for heterogeneous clusters," international conf. on parallel processing. Pp:603-610, 2005.
[14] R. Eswari and S. Nikolas, "A Level-wise Priority Based Task Scheduling for Heterogeneous System", International Journal of information and education technology, Vol. 1, No. 5, December 2011.
[15] G.C. Sih and E.A. Lee, "A Compile-Time Scheduling Heuristic for Interconnection-Constrained Heterogeneous Processor Architecture", IEEE Transactions on Parallel and Distributed Systems, vol. 4, no. 2, pp: 175-187, 1993.
[16] Hamid Arabnejad and Jorge G. Barbosa, "List scheduling algorithm for heterogeneous systems by an optimistic cost table", IEEE transaction on parallel and distributed systems, pp: 1-13, 2013.