A Level-wise Priority Based Task Scheduling for Heterogeneous Systems

R. Eswari and S. Nickolas, *Member IACSIT*

***Abstract*—Achieving high performance without proper scheduling of application tasks is impossible in a heterogeneous environment. To solve such an issue, in this paper, a new static scheduling algorithm is proposed called expected completion time based scheduling (ECTS) algorithm, which is used to effectively schedule application tasks on to the heterogeneous processors. The ECTS algorithm finds the task sequence for execution by assigning priority and then maps the selected task sequence on to the processors. In order to give the comparison of proposed algorithm over the existing algorithms, a real Fast Fourier application graphs are considered as experimental test case. The results show the effectiveness of the proposed algorithm to those presented previously. The algorithm is mainly focused on producing minimum makespan.**

***Index Terms*—Directed acyclic graph, heterogeneous distributed systems, heuristic algorithm, static task scheduling.**

I. Introduction

The availability of high speed networks and diverse sets of resources lead to a new platform, called as heterogeneous platform. Such a platform contains interconnected resources with different computing capabilities and different computing speeds. To run an application in this heterogeneous environment, several issues need to be considered such as partitioning the application, scheduling the tasks, etc. Task scheduling is one of the key factors for achieving high performance in heterogeneous environment since an improper schedule of tasks can fail to exploit the computing capability of available resources in that platform. So the efficiency of executing applications on heterogeneous systems depends on the method we used to schedule tasks onto the available processors. The main objective of task scheduling is to map the tasks onto the available processors and order their execution without violating the precedence constraints and with the aim of producing minimum schedule length [1].

Generally the task scheduling algorithms are classified into two classes, static and dynamic. In static scheduling, decisions can be made at compile-time, since the application characteristics such as execution time of tasks, data dependencies between the tasks and the amount of data to be transferred between the tasks are known priori, whereas in the dynamic scheduling decisions are made at run time [2]. The static task scheduling for a heterogeneous distributed computing system is an NP-complete problem [1], which means that there is no known algorithm that finds the optimal solution in polynomial time. Several heuristics algorithms have been proposed for homogeneous and heterogeneous systems for finding sub-optimal solutions. These heuristics are categorized into several groups, such as list-based algorithms, clustering algorithms, guided random search algorithms, and duplication-based algorithms. Among these algorithms, the list-based scheduling algorithms provide good quality of schedules and performance [3].

In list scheduling heuristics, a priority list is generated from the given graph, which contains ordered list of tasks with its priority. The tasks are selected based on its priority and are assigned to the best processors which minimize its execution time. Some well known list scheduling algorithms are heterogeneous earliest finish time algorithm (HEFT) [1], critical path on a processor (CPOP) [1], and Performance Effective Task Scheduling (PETS) [4] algorithm.

In clustering heuristics, a set of tasks that communicate among themselves are grouped together to create a cluster. If the number of created clusters is greater than the number of available processors, then clusters are merged so that the remaining number of clusters equals the number of processors. Finally, clusters are mapped to the available processors and task ordering within each processor is determined. If two tasks are assigned to the same cluster, they will be executed on the same processor. Mobility directed (MD) [5] and clustering for heterogeneous processors (CHP) [6] are examples of clustering algorithms.

Duplication based scheduling algorithms schedule a task graph by executing the tasks redundantly that have high number of dependent tasks, which reduces the interprocess communication overhead. These algorithms can be applied for an unbounded number of processors but they have much higher scheduling complexity than the algorithms in other group. Heterogeneous n-predecessor decisive path (HNPD) [7], task duplication based scheduling (TDS) [8], and heterogeneous economical duplication (HED) [9] are some of the examples of duplication based scheduling algorithms.

Guided random scheduling algorithms make use of the principles of evolution and natural genetics to evolve near-optimal task schedules. Genetic Algorithms [10] are most popular and widely used technique for task scheduling problem.

In this paper a new heuristic algorithm is proposed, called Expected Completion Time based Scheduling Algorithm for a bounded number of heterogeneous processors and is based on list-scheduling heuristics. The motivation behind this algorithm is to generate a high quality task schedules that are necessary to achieve high performance in heterogeneous environment. The paper is organized as follows: Section II describes the task scheduling problem. Section III gives an overview of the related work. Section IV presents the

Manuscript received November 11, 2011; revised November 27, 2011.

Authors are with Department of Computer Applications, National Institute of Technology, Tiruchirappalli - 620015, Tamilnadu, India (e-mail: eswari@nitt.edu; nickolas@nitt.edu).

proposed algorithm with example. Section V discusses the comparative and experimental analysis and Section VI concludes with future work.

II. Problem Description

In a distributed environment, an application is decomposed into multiple tasks with data dependencies among them. It can be represented by a directed acyclic graph (DAG), *G (T, E)*, where *T* is the set of '*n'* tasks and *E* is the set of '*e'* edges between the tasks. Each task $t_i \in T$ represents a task in the distributed application, and each edge $(t_i, t_j) \in E$ represents a precedence constraint, such that the execution t_j starts after the execution of t_i. A task without any parent is called an entry task (t_{entry}), and a task without any child is called an exit task (t_{exit}). Each edge $(t_i, t_j) \in E$ has a value that represents the communication overhead when data is transferred from task t_i to task t_j. A task can start execution on a processor only when all data from its parents become available to that processor.

A heterogeneous distributed environment consists of a set *Q* of *m* processors connected in a fully connected topology. The following assumptions are made:

1) All inter-processor communications are performed without contention.
2) Computation can be overlapped with communication.
3) Task execution of a given application is non-preemptive.

The communication cost of the edge $e_{i,j}$, which represents the cost of transferring $\mu_{i,j}$ units of data from task t_i scheduled on processor p_m to task t_j scheduled on processor p_n, is defined as

$$c_{i,j} = S_m + R_{m,n} * \mu_{i,j} \quad (1)$$

where S_m is the communication startup time of p_m, $\mu_{i,j}$ is the amount of data transferred from task t_i to task t_j, and $R_{m,n}$ is the communication time per transferred unit from p_m to p_n.

The average communication cost of sending data from task t_i to task t_j is defined by

$$C_{i,j} = \bar{S} + \bar{R} * \mu_{i,j} \quad (2)$$

where $\bar{S}$ is the average communication startup costs over all processors, $\bar{R}$ is the average communication cost per transferred unit over all processors. If t_i and t_j are on the same processor then $c_{i,j} = 0$ since intraprocessor communication is negligible.

EST(t_i, m_j) and EFT(t_i, m_j) [1] are the earliest execution start time and earliest execution finish time of task t_i on processor m_j.

$$EST(t_{entry}, m_j) = 0 \quad (3)$$

$$EST(t_i, m_j) = max\{avail[j], \max_{t_k \in pred(t_i)}(AFT(t_k) + c_{k,j})\} \quad (4)$$

where $AFT(t_k)$ is the actual finish time of a task t_k on the processor m_j, *avail[j]* is the time that the processor m_j is free and it is ready to execute task t_i. The inner max block in equation (4) returns the ready time, i.e., the time when all data needed by t_i, has arrived at processor m_j.

To compute EFT of a task t_i, all immediate predecessor tasks of t_i must have been scheduled.

$$EFT(t_i, m_j) = w_{i,j} + EST(t_i, m_j) \quad (5)$$

where $w_{i,j}$ is the computation cost of task t_i on processor m_j.

After all tasks in a graph are scheduled, the schedule length (overall execution time) will be the AFT of the exit task t_{exit}. The schedule length also called makespan [1] is defined as

$$makespan = \max(AFT(t_{exit})) \quad (6)$$

The objective function of task scheduling problem is to assign tasks onto the available processors in such a way to produce minimum schedule length.

III. Related Work

This section presents two existing task scheduling algorithms for heterogeneous systems, taken for comparison, namely, heterogeneous-earliest-finish-time (HEFT) [1] algorithm, and Performance Effective Task Scheduling (PETS) [4] algorithm.

A. The Heterogeneous-Earliest-Finish-Time (HEFT) Algorithm

HEFT algorithm has two phases: The task prioritizing phase assigns value to each task called upward rank, $rank_u$, which is based on mean computation and mean communication costs. The task list is then generated by sorting the tasks in decreasing order of $rank_u$. In processor selection phase the unscheduled task which has the highest upward rank is selected and assigned to the processor that minimizes its finish execution time, using the insertion-based scheduling policy.

B. The Performance Effective Task Scheduling (PETS) Algorithm

This algorithm starts with level sorting phase in which tasks at each level are sorted in order to group the tasks that are independent of each other. The priority is computed and assigned to each task using the attributes such as Average Computation Cost which finds average computation cost of each task on all processors, Data Transfer Cost which finds the amount of communication cost incurred to transfer the data from a task to all its immediate successors tasks and the Rank of Predecessor Task calculates the highest rank of all immediate predecessor tasks for any particular task. The tasks are selected and assigned to the processor which gives minimum finish time.

IV. Proposed Algorithm

The proposed Expected Completion Time based Scheduling (ECTS) algorithm [11] is given in Fig.1. It consists of two phases namely, Task Prioritization phase and Processor Selection phase. The first phase consists of two stages. The first stage is called as level wise task priority in which the priority of each task at each level is computed and the second stage is called as task selection in which the tasks are selected from all levels based on their priority. And in the second phase, the selected tasks are assigned to the best processor, which minimizes its completion time.

A. Task Prioritization Phase

The optimized schedule length depends on the order in which tasks are generated. In this phase an ordered task list is generated for execution by two stages.

```
Read the DAG with the corresponding attributes such as
    computation cost, communication, number of processors
While unscheduled tasks in the DAG do
  For each level Li in the DAG do
    For each task ti in the level Li do
      Find Average Computation Cost, Maximum Data Arrival Cost,
        and Expected Completion Time
    End For
    Sort all tasks in decreasing order of ECT and
      assign priority for each task
    Select tasks from the level by its priority
  End For
  For each processor pk in the processor set do
    Compute earliest finish time of tj on pk using the insertion based policy
  End For
    Assign task ti to the processor pk that minimizes its finish time
End while
End
```

Fig. 1. ECTS algorithm

Stage 1 – Level-wise Task Priority: In this first stage, the priority of all tasks at each level is computed based on their average computation cost and maximum data arrival cost. The Average Computation Cost of all tasks for the given graph is calculated using the equation (7).

Definition 1. Given a DAG with n tasks and m processors, the Average Computation Cost (ACC) of a task (t_i) is computed by dividing the sum of computation cost of the task on each processor by the number of available processors.

$$ACC(t_i) = \overline{w_i} = \sum_{j=1}^{m} w_{i,j} \; / \; m \qquad (7)$$

where $w_{i,j}$ is the estimated execution time to complete task t_i on processor m_j.

A task can have more than one parent. Hence, the Completion time of any task depends on the highest Data Arrival Cost among its parents in addition to its average computation, as given in definition 2.

Definition 2. Given a DAG with n tasks and e edges, the Maximum Data Arrival Cost (MDAC) of a task (t_j) is the highest amount of time that the task needs to spend to receive data among its parents.

$$MDAC(t_j) = max_{t_i \in pred(t_j)}(C_{i,j}) \qquad (8)$$

where t_i is the set of predecessors of task t_j.

For the given graph, the MDAC value for the first task at *level-1* is 0, since it is an entry task. The MDAC value for all tasks at *level-2* and the first task (t_7) of *level-3* is their data arrival cost, since they have only one parent, whereas the second task (t_8) of *level-3* has three parents such as t_2, t_4, t_6 and their data transfer costs are 19, 27, and 15 respectively. The highest data arrival cost (MDAC), 27 is assigned to task t_8 from its parent t_4. In this way the MDAC value for all tasks for the given graph is assigned by definition 2.

The Expected Completion Time of all tasks is calculated as per the definition 3.

Definition 3. The Expected Completion Time (ECT) of a task (t_j) is computed by summing the average computation cost of that task and maximum data arrival cost of the same task.

$$ECT(t_j) = ACC(t_j) + MDAC(t_j) \qquad (9)$$

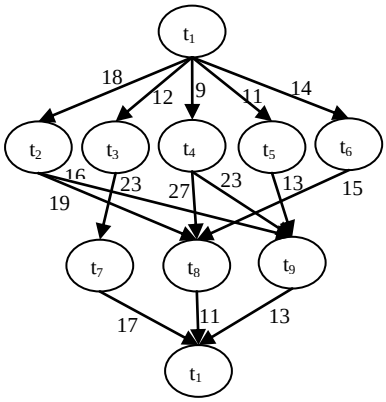

Fig. 2. Sample DAG

TABLE I: COMPUTATION COST MATRIX

Task	M_1	M_2	M_3
t_1	14	16	9
t_2	13	19	18
t_3	11	13	19
t_4	13	8	17
t_5	12	13	10
t_6	13	16	9
t_7	7	15	11
t_8	5	11	14
t_9	18	12	20
t_{10}	21	7	10

For the given graph (fig.2.), the ECT value for the entry task is its ACC, so t_1 has the ECT value 13. The ECT value for the tasks with single parent is calculated by adding data arrival cost with its average computation cost. Hence ECT value for all tasks at *level-2* and the first task at *level-3* is shown in TABLE II. The ECT value for the tasks that are having more than one parent is computed by adding their ACC with MDAC. So, the ECT value for the second task (t_8) at *level-3* is 37, where its ACC is 10 and MDAC is 27. Likewise the ECT value is computed for all tasks. The computed value of ACC, MDAC and ECT of all tasks is shown in TABLE II.

TABLE II: PRIORITY COMPUTATION FOR THE GIVEN GRAPH

Level	Task	ACC	Parent tasks	MDAC	ECT	Priority
1	1	13	0	0	13	1
2	2	16.667	1	18	34.667	1
2	3	14.333	1	12	26.333	3
2	4	12.667	1	9	21.667	5
2	5	11.667	1	11	22.667	4
2	6	12.667	1	14	26.667	2
3	7	11	3	23	34	3
3	8	10	2,4,6	27	37	2
3	9	16.667	2,4,5	23	39.667	1
4	10	14.667	7,8,9	17	31.667	1

Stage 2 – Task Selection: This is the second stage of Task Prioritization phase in which all tasks at each level are sorted in non-increasing order of their ECT value, which is computed in the first stage and priority is assigned to them. A task which is having the highest ECT is given higher priority and the tasks are selected by their priority. Thus the task sequence arrived by the proposed algorithm for the given graph is $t_1 - t_2 - t_6 - t_3 - t_5 - t_4 - t_9 - t_8 - t_7 - t_{10}$.

B. Processor Selection Phase

In this phase, the selected task is assigned to a processor in the set of processors that minimizes its finish execution time using the insertion-based scheduling policy [1]. When a processor m_j is assigned to a task t_i, the insertion based scheduling policy considers the possible insertion of the task in an earliest idle time slot between two already-scheduled tasks on that processor. This must be done without violating the precedence constraints among tasks. An idle time slot on processor m_j is defined as the difference between execution start time and finish time of two tasks that were consecutively scheduled on the processor m_j and should be greater than or equal to the computation cost of the task to be scheduled. The search starts from a time equal to the ready time of t_i on m_j, and proceeds until it finds the first idle time slot with a sufficient large time space to accommodate the computation cost of t_i on m_j. If no such idle time slot is found, the insertion-based scheduling policy inserts the selected task after the last scheduled task on m_j. If two processors are producing same EFT for a selected task, then the following selection strategies can be followed:

- Select processor randomly
- Select processor that is lightly loaded

Using the above processor selection strategy, all tasks of the given graph have been scheduled in the order they are selected for execution. The mapping of tasks and their processor pair is shown in TABLE III.

TABLE III: TASKS AND PROCESSORS PAIR

Selected Task	Selected Processor
t_1	M_3
t_2	M_3
t_6	M_3
t_3	M_1
t_5	M_2
t_4	M_2
t_9	M_2
t_8	M_2
t_7	M_1
t_{10}	M_2

According to equation (3) the earliest starting time of the entry task on any processor is 0. When compared with processors M_1 and M_2, processor M_3 has less computation time. So task t_1 is scheduled on M_3. Similarly, for the next task t_2, the earliest finish time (EFT) is calculated by equation (5). The task t_2 is assigned to processor M_3 which minimizes its EFT compared to other processors. Likewise, the selected task sequence is assigned to the available processors in such a way to obtain minimum schedule length.

V. EXPERIMENTAL RESULTS AND ANALYSIS

The effectiveness of the proposed algorithm is proved theoretically for the given graph and experimentally for the real application graph, Fast Fourier Transformation.

A. Comparison Metrics

The following metrics have been taken to evaluate the proposed algorithm.

1) Schedule Length Ratio (SLR) is defined as the normalized schedule length to the lower bound of the schedule length. It is calculated using the following formula.

$$SLR = \frac{makespan}{\sum_{n_i \in CP_{MIN}} min_{p_j \in Q}\{w_{i,j}\}} \quad (10)$$

The CP_{MIN} is the critical path of the unscheduled application DAG based on the computation cost of tasks on the fastest processor p_j. The denominator is equal to the sum of computation costs of tasks located on CP_{MIN} when they are scheduled on p_j.

2) The Speedup of a schedule is defined as ratio of the schedule length obtained by assigning all tasks to the fastest processor, to the parallel execution time (makespan of the output schedule) of the task schedule.

$$Speedup = \frac{min_{p_j \in Q}\{\sum_{n_i \in V} w_{i,j}\}}{makespan} \quad (11)$$

B. Comparative Analysis

The sample DAG is shown in Fig.2 and its computation cost matrix is given in Table I. The stepwise trace of the proposed algorithm is given in Table II. For this graph the generated schedule length of the ECTS algorithm is shown in Fig.3.

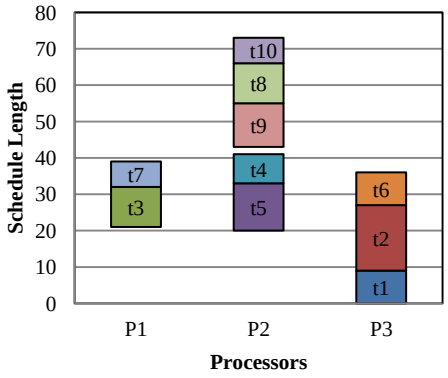

Fig. 3. Schedule length of ECTS algorithm

The schedule length (makespan) comparison of the three algorithms is shown in Fig.4. The schedule length of ECTS algorithm is 73, which is shorter than the schedule length generated by PETS and HEFT algorithms which are 77 and 80 respectively. It is observed that the proposed ECTS algorithm is producing better schedule compared to the other two algorithms.

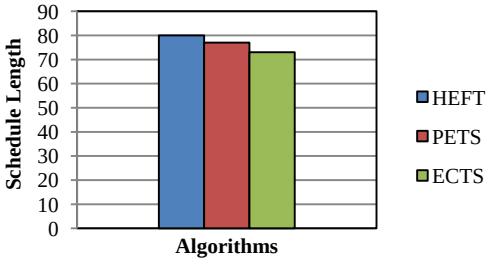

Fig. 4. Schedule length comparison of three algorithms

C. Fast Fourier Transformation

A task graph of Fast Fourier Transformation [1] (FFT) algorithm with respect to four data points is given in Fig.5.

FFT is characterized by the size of the input vector (data points).

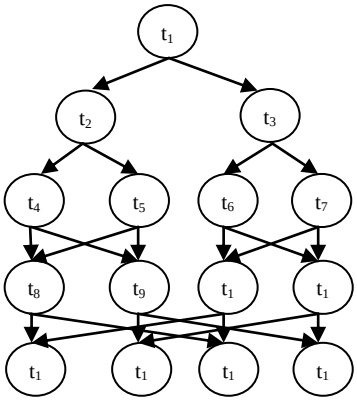

Fig .5. FFT graph with four data points

For an input vector of size M, the total total number of nodes in the task graph is equal to $(2 * M - 1) + (M * \log_2 M)$. The parameters such as number tasks, in degree and out degree are not needed because the structure of the FFT application graphs is already known. Communication to computation cost ratio and range percentage of computation cost (heterogeneity factor) are taken from the data sets given in [1]. The number of data points is taken as input, which various from 2 to 16 incrementing powers of 2.

- Heterogeneity factor β, where $\overline{w_i}$ is selected randomly from a uniform distribution with range$[0, 2 * \overline{w_{DAG}}]$, where $\overline{w_{DAG}}$ is the average computation cost of the given graph, which is set randomly in the algorithm. Then the computation cost of each task t_i on each processor m_j is randomly set from the following range:

$$\overline{w_i} * \left(1 - \frac{\beta}{2}\right) \leq w_{i,j} \leq \overline{w_i} * \left(1 + \frac{\beta}{2}\right) \quad (12)$$

- Communication to computation ratio (CCR) is defined as the ratio of the average communication cost to the average computation cost.

D. Performance Results

The algorithm is implemented using C language in Intel core i7-620M processor. For this experimentation around 885 numbers of FFT graphs have been generated. The performance of the algorithm is compared with respect to different number of data points, which varies from 2 to 16 incrementing powers of 2, and the result is shown in Fig.6. Further the efficiency of the algorithm is compared for various number of processors (2, 4, 8, 16, 32) and is shown in Fig.7 and Fig.8.

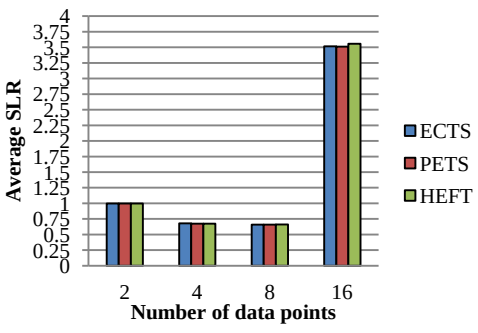

Fig. 6. Average SLR for different data points

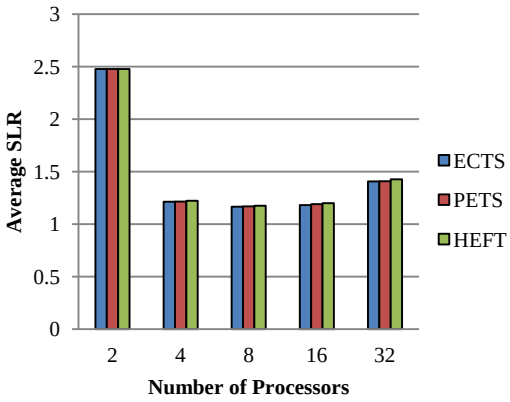

Fig. 7. Average SLR for different processors

The average SLR is reduced while increasing number of processors and number of tasks for the case of ECTS algorithm. The average speedup obtained for each of the algorithms with respect to varying numbers of processors is shown in the Fig.8. The ECTS algorithm is faster than other algorithms for large number of tasks and processors as shown from the graphs.

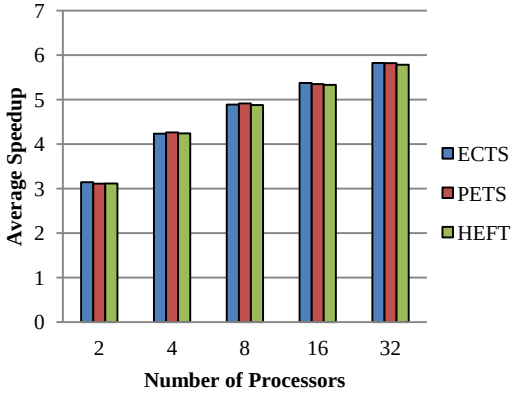

Fig. 8. Average speedu.

Finally, a comparison is made with respect to the quality of schedules generated by each of the algorithms. On all generated graphs, ECTS algorithm gives better or equal performance than the PETS algorithm by 77.2% and the HEFT algorithm by 80.4%.

From the above results, it can be observed that the proposed ECTS algorithm outperforms when compared with HEFT algorithm and gives competitive performance when compared with PETS algorithm. Moreover, ECTS algorithm gives better performance when the number of tasks and processors were increased.

VI. Conclusion

In this paper, a non-preemptive static task scheduling algorithm is proposed for heterogeneous computing systems. The ECTS algorithm is evaluated for the real world FFT application graphs. The scheduling performance of the algorithm is compared with the existing HEFT and PETS algorithms. The experimental results show that the performance of ECTS algorithm increases with respect to increase in the number of tasks. It can be tested for more real applications graphs and enhancement can be made in terms of time complexity and efficiency which are to be considered as

future course of work. Also the effect of task duplication in reducing schedule length can be considered in future.

REFERENCES

[1] H. Topcuoglu, S. Hariri, and M.Y. Wu, "Performance-effective and low-complexity task scheduling for heterogeneous computing," *IEEE Trans. Parallel and Distributed Systems*, vol. 13, no.3, pp. 260-274, March 2002.

[2] L. Lee, H. Chang, K. Liu, G. Chang, and C. Lien, "A dynamic scheduling algorithm in heterogeneous computing environments," *IEEE W4B-4, ISCIT*, pp. 313-318, 2006.

[3] C. Yang, P. Lee, and Y. Chung, "Improving static task scheduling in heterogeneous and homogeneous computing systems," in *International Conf. on Parallel Processing*, 2007, pp. 45.

[4] E. Illavarasan and P. Thambidurai (February 2007). Low complexity performance effective task scheduling algorithm for heterogeneous computing environments. *J. of Computer Sci.* [Online]. 3(2). pp. 94-103. Available: http://thescipub.com/issue-jcs/3/2

[5] M. Wu and D. Gajski, "Hypertool: A Programming aid for message passing systems," *IEEE Trans. on Parallel and Distributed systems*, vol.1, pp.330-343, July 1990.

[6] C. Boeres, J. V. Filho, and V. E. F. Rebello, "A cluster based strategy for scheduling task on heterogeneous processors," in *Proc. of 16th Symp. on Computer Architecture and High Performance Computing (SBAC-PAD)*, 2004, pp.214-221.

[7] S. Baskiyar and C. Dickinson (May 2005). Scheduling directed a-cyclic graph on a bounded set of heterogeneous processors using task duplication. *J. Parallel Distributed Computing*, [Online]. 65. pp. 911-921. Available: http://www.sciencedirect.com/science/article/pii/S0743731505000067

[8] S. Ranaweera and D. P. Agrawal, "A task duplication based scheduling algorithm for heterogeneous systems," in *Proc. International Parallel and Distributed Processing Symposium*, 2000, pp. 445–450.

[9] A. Agarwal and P. Kumar, "Economical duplication based task scheduling for heterogeneous and homogeneous computing systems," in *IEEE International Advance Computing Conference*, 2009, pp.6-7.

[10] S. C. Kim and S. Lee, "Push-pull: Guided search DAG scheduling for heterogeneous clusters," in *International. Conf. on Parallel Processing*, 2005, pp.603-610.

[11] R. Eswari and S. Nickolas, "Expected completion time based scheduling algorithm for heterogeneous processors," in *Proc. 2011 International Conf. Information Communication and Management*, IPCSIT vol.16 2011, pp.72-77.

R. Eswari, obtained M.E. in the field of Computer and Communication from Anna University, Chennai, Tamilnadu, India, and B.E. in the field of Computer Science and Engineering from Bharathidasan Univeristy, Tiruchirappalli, Tamilnadu, India. She is pursuing Ph.D. degree from the department of Computer Applications, National Institute of Technology, Tiruchirappalli, Tamilnadu, India. She secured first rank in her M.E. course. She is currently working as Assistant Professor in the Department of Computer Applications, National Institute of Technology, Tiruchirappalli, She is a life member of Computer Society of India, and a member of IACSIT. Her research interest includes scheduling in distributed systems, database management systems and data structures.

S. Nickolas is currently working as Associate Professor in the Department of Computer Applications, National Institute of Technology, Tiruchirappalli, Tamilnadu, India. He obtained his M.E. in the field of Computer Science from Regional Engineering College, Tiruchirappalli, and Ph.D. from National Institute of Technology, Tiruchirappalli, in the year 2007 Dr. S. Nickolas has published 20 papers in International Conferences and 10 papers in International Journals. He is life member of ISTE, CSI, IE and also member of IACSIT. His area of interest includes Database systems, Data mining, Software metrics and Distributed systems.