Performance Effective Task Scheduling Algorithm for Heterogeneous Computing System

E. Ilavarasan P. Thambidurai and R. Mahilmannan
Department of Computer Science & Engineering and Information Technology
Pondicherry Engineering College
Pondicherry – 605014. India
Email: eilavarasan@yahoo.com

Abstract

Finding an optimal solution to the problem of scheduling an application modeled by a Directed Acyclic Graph (DAG) onto a distributed system is known to be NP-complete. The complexity of the problem increases when task scheduling is to be done in a heterogeneous computing system, where the processors in the network may not be identical and take different amounts of time to execute the same task. This paper introduces a Performance Effective Task Scheduling (PETS) Algorithm for Network of Heterogeneous system, with complexity O (v+e) (p+ log v), which provides optimal results for applications represented by DAGs. The performance of the algorithm is illustrated by comparing the schedule length, speedup, efficiency and the scheduling time with existing algorithms such as, Heterogeneous Earliest Finish Time (HEFT) and Critical-Path On a processor (CPOP) and Levelized Min Time (LMT) reported in this paper. The comparison study based on both randomly generated graphs and graphs of some real applications shows that PETS algorithm substantially outperforms existing algorithms.

1. Introduction

Heterogeneous Computing (HC) system is a suite of distributed processors interconnected by high-speed networks, thereby promising high speed processing of computationally intensive applications with diverse computing needs. A well-known strategy behind efficient execution of a huge application on HC system is to partition it into multiple independent tasks and schedule such tasks over a set of available processors. A task-partitioning algorithm takes care of efficiently dividing an application into tasks of appropriate grain size and an abstract model of such a partitioned application can be represented by a Directed A-cyclic Graph (DAG). Each task of a DAG corresponds to a sequence of operations and a directed edge represents the precedence constraints between the tasks. Each task can be executed on a processor and the directed edge shows transfer of relevant data from one processor to another. Task scheduling can be performed at compile-time or at run-time. When the characteristics of an application, which includes execution times of tasks on different processors, the data size of the communication between tasks, and the task dependencies are known a priori, it is represented with a static model. The objective function of this problem is to map the tasks on the processors and order their execution so that task precedence requirements are satisfied and a minimum overall completion time is obtained. The problem of scheduling of tasks with required precedence relationship, in the most general case, has been proven to be NP-complete [1] [2] and optimal solutions can be found only after an exhaustive search. Because of its key importance on performance, the task-scheduling problem in general has been studied extensively and various heuristics were proposed in the literature [3-12]. The motivation behind our work is to develop a new task-scheduling algorithm to deliver high performance in terms of both performance metrics (schedule length ratio, speedup, efficiency) and a cost metric (scheduling time). We have improved the work done in [5] [6] and proposed a new task scheduling algorithm.

The rest of the paper is organized as follows: In the next Section, we define the task scheduling problems. In Section 3 we present the related works, Section 4 introduces PETS algorithm and Section 5 provides results and discussions. Finally Section 6 concludes the paper with some final remarks.

2. Task Scheduling Problems

A scheduling system model consists of an application, a target computing system and criteria for scheduling. An *application* program is represented by a Directed Acyclic Graph (DAG), $G=(V, <, E)$, where $V=\{v_i, i=1...n\}$ is the set of n tasks. $<$ represents a partial order on V. For any two tasks $v_i, v_k \in V$, the existence of the partial order $v_i < v_k$ means that v_k cannot be scheduled until task v_i has been completed, hence v_i is a predecessor of v_k and v_k is a successor of v_i. The task executions of a given application are assumed to be non-preemptive. E is the set of directed edges. Data is a $n \times n$ matrix of communication data, where $data_{i,k}$ is the amount of data required to be

Proceedings of the 4th International Symposium on Parallel and Distributed Computing (ISPDC'05)
0-7695-2434-6/05 $20.00 © 2005 IEEE

transmitted from task v_i to task v_k. In a given task graph, a task without any predecessor is called an *entry task* and a task without any child is called an *exit task*. Without loss of generality, it is assumed that there is one *entry task* to the DAG and one *exit task* from the DAG. In an actual implementation, we can create a *pseudoentry* task and *pseudoexit* task with zero computation time and communication time.

Heterogeneous computing system consists of a set P = {p_j: j =0, m-1} of *m* independent different types of processors fully interconnected by a high-speed arbitrary network. The bandwidth (data transfer rate) of the links between different processors in a heterogeneous system may be different depending on the kind of the network. The data transfer rate is represented by an *m x m* matrix, $R_{m \times m}$. W is a *n x m* computation cost matrix in which each W_{ij} gives the Estimated Computation Time *(ECT)* to complete task v_i on processor p_j where 0<=i<n and 1<=j<=m. The *ECT* value of a task may be different on different processor depending on the processor's computational capability. The communication cost between two processors p_x and p_y, depends on the channel initialization at both sender processor p_x and receiver processor p_y in addition to the communication time on the channel. This is a dominant factor and can be assumed to be independent of the source and destination processors. The channel initialization time is assumed to be negligible. The communication cost of the *edge(i,k),* which is for transferring data from task v_i (scheduled on processor p_x) to task v_k (scheduled on processor p_y) is defined by Eqn.(1)

$$C_{i,k} = data_{i,k} / R_{x,y} \quad (1)$$

Otherwise, $C_{i,k}$ = 0 when both the tasks v_i and v_k scheduled on the same processor. We assumed that the data transfer rate for each link is 1.0 and hence communication cost and amount of data to be transferred will be the same. A task graph with 10 tasks, and its computation cost matrix given in [6] are shown in Figure 1 and Table 1.

Table 1. Computation Cost Matrix given in [6]

Task	P_1	P_2	P_3
1	14	16	9
2	13	19	18
3	11	13	19
4	13	8	17
5	12	13	10
6	13	16	9
7	7	15	11
8	5	11	14
9	18	12	20
10	21	7	16

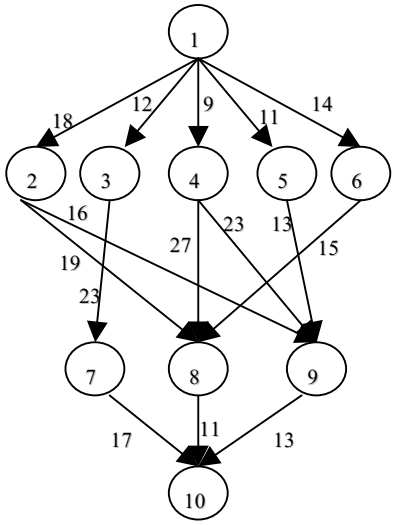


Figure 1. Task Graph given in [6]

Let $EST(v_i, p_j)$ and $EFT(v_i, p_j)$ are the Earliest Start Time and Earliest Finish Time of task v_i on p_j, respectively. For the entry task v_{entry}, $EST(v_{entry}, p_j) = 0$, and for the other tasks in the graph, the *EST* and *EFT* values are computed recursively, starting from the entry task, as shown in Eqn.(2) and (3). In order to compute the *EFT* of a task v_i, all immediate predecessor tasks of v_i must have been scheduled.

$$EST(v_i, p_j) = max \{avail[j], max (AFT(v_t + C_{t,i}))\} \quad where\ v_t \in pred(v_i) \quad (2)$$

$$EFT(v_i, p_j) = W_{ij} + EST(v_i, p_j) \quad (3)$$

where $pred(v_i)$ is the set of immediate predecessor tasks of task v_i and *avail[j]* is the earliest time at which processor p_j is ready for task execution. If v_k is the last assigned task on processor p_j, then *avail[j]* is the time that processor p_j completed the execution of the task v_k and it is ready to execute another task when we have a noninsertion-based scheduling policy. The inner max block in the *EST* equation returns the ready time, i.e., the time when all the data needed by v_i has arrived at processor p_j. After a task v_t is scheduled on a processor p_j, the earliest start time and the earliest finish time of v_t on processor p_j is equal to the actual start time $AST(v_t)$ and the actual finish time $AFT(v_t)$ of task v_t, respectively. After all tasks in a graph are scheduled, the schedule length (i.e. the overall completion time) will be the actual finish time of the exit task v_{exit}. Finally the schedule length is defined as Eqn. (4)

$$Schedule\ Length = \max\{AFT(v_{exit})\} \quad (4)$$

The objective function of the task-scheduling problem is to schedule the tasks of an application to machines such that its schedule length is minimized.

Proceedings of the 4th International Symposium on Parallel and Distributed Computing (ISPDC'05)
0-7695-2434-6/05 $20.00 © 2005 IEEE

3.Related Works

Efficient application scheduling is critical for achieving high performance in heterogeneous computing system, because of its key importance on performance, the scheduling problem has been extensively studied and various heuristics have been proposed in the literature [3-12]. These heuristics are classified into a variety of schemes such as priority-based [4,5,6], cluster-based [7], guided random search based [8] and task duplication based schemes [9,10,11].

Priority-based schemes [5,6,7] assume a priority for each task that is utilized to assign the tasks to the different processors. Priorities based scheduling algorithms, such as Mapping Heuristics (MH) [4], Levelized Min Time (LMT) [5], Heterogeneous Earliest Finish Time (HEFT) [6] and Critical-Path-On a Processor (CPOP) [6] have been proposed in the literature for heterogeneous systems. The complexity of MH, LMT, HEFT, and CPOP algorithms is O (v^2 x p), O (v^2 x p^2), O (v^2 x p) and O (v^2 x p) respectively. HEFT and CPOP algorithms are proved to be improvement over MH and LMT algorithms in terms of average SLR, speedup, and run time. We have chosen the recently proposed algorithms [5] [6] for improvement.

4. Performance Effective Task Scheduling (PETS) Algorithm

The proposed algorithm consists of three phases, viz., level sorting, task prioritization, and processor selection. The detailed explanation of the algorithm is given below:

In the first phase, the given DAG is traversed in a top-down fashion to sort task at each level in order to group the tasks that are independent of each other. As a result, tasks in the same level can be executed in parallel. Given a DAG *G = (V, E),* level 0 contains entry tasks. Level *i* consist of all tasks v_k such that, for all *edges*(v_j,v_k), task v_j is in a level less than *i* and there exists at least one *edge*(v_j,v_k) such that v_j is in level *i-1*. The last level comprises of some of the exit tasks.

In the second phase of the algorithm priority is computed and assigned to each task. Priority is computed based on the task communication cost and average computation cost. The Average Computation Cost *(ACC)* of a task is the average computation cost on all the available *m* processors and it is computed by using Eqn. (5)

$$ACC(v_i) = \sum_{j=1}^{m} w_{i,j} / m \quad (5)$$

Here, we have defined two kinds of communication cost for a task viz. Data Transfer Cost *(DTC)* and Data Receiving Cost *(DRC)*. *DTC* of a task is the amount of cost incurred to transfer the data to all its immediate successor tasks and it is computed as follows: The *DTC* for the exit task is 0; for all other tasks at *level i*, it is computed by using Eqn. (6)

$$DTC(v_i) = \sum_{i=1}^{x} C_{i,j}, \quad (6)$$

, x is the number of immediate successors of v_i

DRC of a task is the cost incurred to receive data from its immediate predecessor tasks. The *DRC* of a task is computed as follows: The *DRC* for the entry task is 0; for all other tasks at *level i*, it is computed by using Eqn. (7)

$$DRC(v_j) = Max\{rank(v_j)\} \quad (7)$$

where v_j *is the predecessor of* v_j

Rank of a task v_i (*rank*(v_i)) is the sum of its *DTC, DRC* and *ACC* values of that task. For the every task at each *level i*, *rank*(v_i) is computed by using Eqn. (8)

$$rank(v_i) = DTC(v_i)+DRC(v_i)+ACC(v_i) \quad (8)$$

Priority is assigned to all the tasks at each level i, based on its rank value. At each level, the task with highest rank value receives the highest priority followed by task with next highest rank value and so on. The computed ACC, DTC, DRC, rank and priority value for each of the task in Figure 1 is shown in Table 2.

In the processor selection phase, the processor, which gives minimum *EFT* for a task is selected for executing that task. It has an insertion-based policy, which considers the possible insertion of a task in an earliest idle time slot between two already scheduled tasks on a processor. At each level, the earliest start time and earliest finish time of each task on every processor is computed using Eqn. (2) and (3). Calculation of *EST* and *EFT* values for the task graph in Figure 1 is illustrated below: For example, for the task 8, *EST (8,* P_1*)*=max {40, max (40,53,50)}=53, *EFT(8,* P_1*)* = 5+53=58, *EST(8,* P_2*)* = max{70, max(59, 53, 50)}=70, *EFT(8,* P_2*)* = 11+70=81, *EST(8,* P_3*)* = max{35, max(59, 35, 35)}=59 and *EFT(8,* P_3*)* = 14+59=73. The tasks are selected for execution based on their priority value. Task with highest priority is selected and scheduled on its favorite processor (processor which gives the minimum EFT) for execution followed by the next highest priority task. Similarly all the tasks are scheduled on to suitable processor. The processors selected for executing the tasks of task graph in Figure 1 is as follows: For example, task 1 is the entry task; hence its data arrival time is 0 and P_3 gives the minimum *EFT* for task 1. Hence processor P_3 is selected for executing task 1. For task 2, the data arrival time from its predecessor (task 1 in P_3) is 9 and the *EFT* of this task on P_1, P_2 and P_3 are 40, 46, and 44. Since P_1 gives minimum *EFT*, it is selected for executing the task 2. Similarly all

Proceedings of the 4th International Symposium on Parallel and Distributed Computing (ISPDC'05)
0-7695-2434-6/05 $20.00 © 2005 IEEE

other tasks in the task graph are scheduled on to the suitable processor. The processor selected for executing each of the task in Figure 1 is shown in Table 3. The proposed algorithm is given in Figure 2.

1. Read the DAG, associated attributes values, and the number of processor P;
2. For all tasks at each level L_i do
3. Begin
4. Compute *DRC, DTC* and *ACC*.
5. Compute $rank(v_k) = DTC(v_k)+DRC(v_k)+ACC(v_k)$
6. Construct a priority queue using ranks;
7. While there are unscheduled tasks in the queue do
8. Begin
9. Select the first task, v_k from the queue for scheduling;
10. For each processor p_k in the processor set P do
11. Begin
12. Compute EFT (v_k,p_k) value using insertion based scheduling policy;
13. Assign the task v_k to processor p_k, which minimizes the EFT;
14. End;
15. End;
16. End.

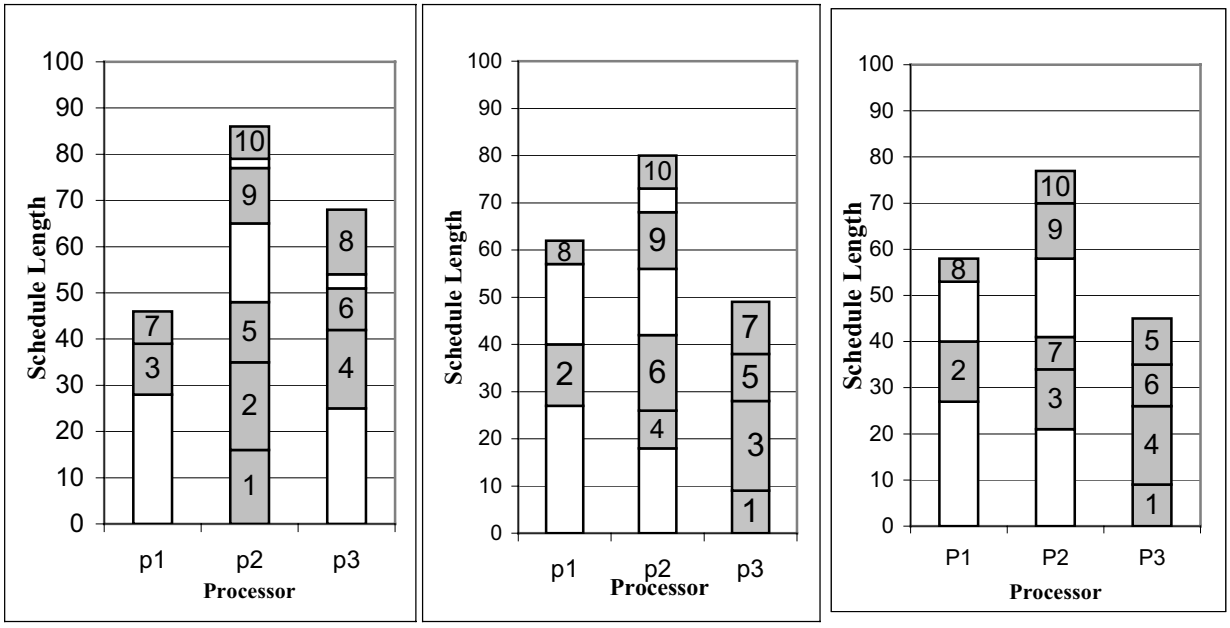
Figure 2. Proposed PETS Algorithm

Table 2. The computed ACC, DTC, DRC, rank and priority values for the tasks in Figure 1

Level	Task	*ACC*	*DTC*	DRC	rank	priority
1	1	13	64	0	77	1
2	2	16.7	35	77	128.7	2
2	3	14.3	23	77	114.3	3
2	4	12.7	50	77	139.7	1
2	5	11.7	13	77	101.7	5
2	6	12.7	15	77	104.7	4
3	7	11	17	114.3	142.3	3
3	8	10	11	139.7	160.7	2
3	9	16.7	13	139.7	169.4	1
4	10	14.7	0	169.7	184.1	1

Table 3. The Computed EST, EFT values on Processors P_1, P_2, P_3 for the tasks in Figure 1

Ordered Tasks	Processors						Predecessors	Processor selected
	P_1		P_2		P_3			
	EST	EFT	EST	EFT	EST	EFT		
1	0	14	0	16	0	9	Null	P_3
4	18	31	18	26	9	26	1	P_3
2	27	40	27	46	26	44	1	P_1
3	40	51	21	34	26	45	1	P_2
6	40	53	34	50	26	35	1	P_3
5	40	52	34	47	35	45	1	P_3
9	58	76	58	70	56	76	2,4,5	P_2
8	53	58	70	81	59	73	2,4,6	P_1
7	58	65	34	41	57	68	3	P_2
10	83	104	70	77	83	99	7,8,9	P_2



(a) CPOP Algorithm (b) HEFT Algorithm (c) PETS Algorithm

Figure 3. The Schedule Length generated by CPOP, HEFT and PETS Algorithms

As an illustration, Figure 3 presents the schedules obtained by the CPOP, HEFT and PETS algorithms for the sample DAG of Figure 1. The schedule length, which is equal to 77, is shorter than the schedule lengths of the related work; specifically, the schedule lengths of HEFT, CPOP and LMT Algorithms are 80, 86, and 91 respectively. The time complexity of PETS algorithm is equal to $O(v + e)(p + \log v)$ where v is the number of tasks, e number of edges and p number of processors. For implementation, we used breadth first search for level sorting which takes O ($v + e$) time complexity. A binary heap was used to implement the priority queue, which has time complexity of O ($\log v$). Each task in the priority queue is checked with all the p processors in order to select a processor that gives the earliest finish time. Hence the complexity of the algorithm is O ($v + e$) ($p+\log v$).

5. Results and Discussion

In this section, we present the comparative evaluation of proposed PETS algorithm and the existing algorithms for heterogeneous systems such as, LMT, HEFT and CPOP for DAGs with various characteristics by simulation. For this purpose, we consider two sets of graphs as the workload for testing the algorithms: randomly generated task graphs and the graphs that represent some of numerical real world problems.

5.1 Comparison Metrics

We have used the following metrics to evaluate the performance of the algorithm and the metrics are:

Schedule Length Ratio (SLR) is the ratio of the parallel time to the sum of weights of the critical path tasks on the fastest processor.

Speedup. The speedup is the ratio of the sequential execution time to the parallel execution time.

Efficiency. The efficiency is the ratio of the speedup value to the number of processor used to schedule the graph.

Frequency of better quality of schedules. The number of times that each algorithm produced better, worse and equal quality of schedules compared to every other algorithm is counted in the experiments.

5.2 Randomly Generated Application Graphs

As part of this work we have implemented a random task graph generator that allows the user to generate a variety of test DAGs with various characteristics that depends on several input parameters and they are *number of tasks in the graph (v), out degree (β), in degree (γ), shape parameter of a graph (α)* Communication to Computation Ratio (CCR) and Range percentage of computation cost (η). By varying α value we can generate different shape of the task graph. The height of the graph is randomly generated from a uniform distribution with a mean value equal to $\sqrt{v}/\alpha$ and the width for each level is randomly selected from a uniform distribution with mean value equal to $\sqrt{v} * \alpha$. A dense graph (shorter

graph with high parallelism) and a longer graph (low parallelism) can be generated by selecting α >>1.0 and α <<1.0 respectively. CCR is the ratio of the average communication cost to the average computation cost. The computation intensive applications may be modeled by assuming CCR = 0.1, whereas data intensive applications may be modeled assuming CCR = 10.0. Range percentage of computation costs on processors (η) is basically the heterogeneity factor for processors speeds. A high percentage value causes a significant difference in a task's computation cost among the processors and a low percentage indicates that the expected execution time of a task is almost equal on any given processor in the system. The average computation cost of each task v_i in the graph, i.e., W_i, is randomly selected from a uniform distribution with range $[0, 2*W_{dag}]$, where 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 v_i on each processor p_j in the system is randomly set from the following range:

$$W_i*(1-\eta/2) <= W_{i,j} <= W_i*(1+\eta/2) \quad (8)$$

For experiments, we set the following range of values for the parameters. v = {30,40,50,60,70,80,90,100}, α = {0.5,1.0,2.0}, β = {1,2,3,4,5}, γ = {1,2,3,4,5}, CCR= {0.1,0.5,1.0,5.0,10.0} and η={0.1,0.5,1.0}.

5.3 Experimental Results

The experimental results are organized in two major test suites.

Test Suite 1: In this test suite, we evaluated the quality of schedules generated by the algorithms with respect to the graph characteristics values given in Section 5.2. We have generated around 720 random task graphs for the experiments with different characteristics and counted the number of times that each scheduling algorithm in the experiments produced better, worse, or equal schedule length compared to every other algorithm. Table 4 indicates the comparison results of the algorithm on the left with the algorithm on the top. The "combined" column shows the percentage of graphs in which the algorithm on the left gives the better, equal, or worse performance than all other algorithms combined. The ranking of algorithms, based on occurrences of best results, is {PETS, HEFT, and CPOP, LMT}.

The performance of the algorithm is also evaluated by average SLR with respect to the graph structure, by varying the α value with 0.5, 1.0. and 2.0 and the results obtained by this experiment are shown in Figure 4. The results confirm that PETS algorithm substantially outperforms reported algorithms in terms of average SLR for random task graphs with various shapes.

Further, we evaluated the efficiency of the algorithms by scheduling random task graphs consists of fixed numbers of tasks (120) on to HC system consists of varying number of processors (4,8,12,16,20). For this experiment, we have used 100 numbers of randomly generated task graphs. The results obtained by this experiments are shown in Figure 5. As expected the average SLR is reduced while increasing the number of processors and at the same time PETS outperforms LMT, CPOP and HEFT algorithms.

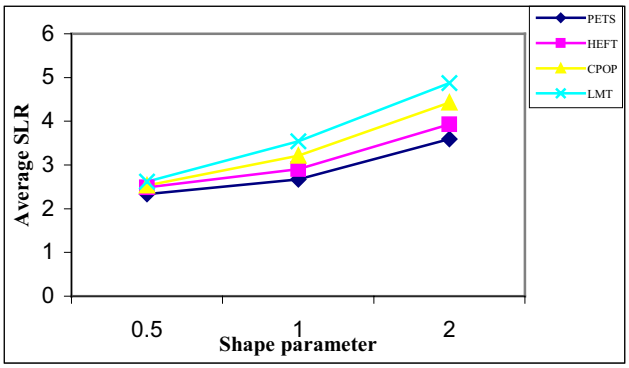


Figure 4. Average SLR for varying (α)

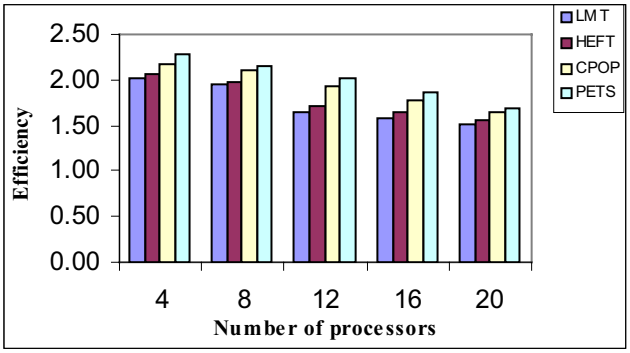
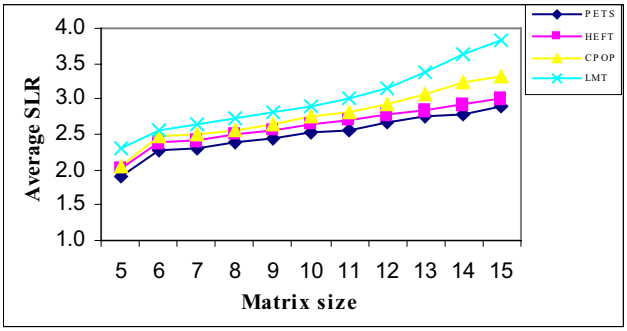


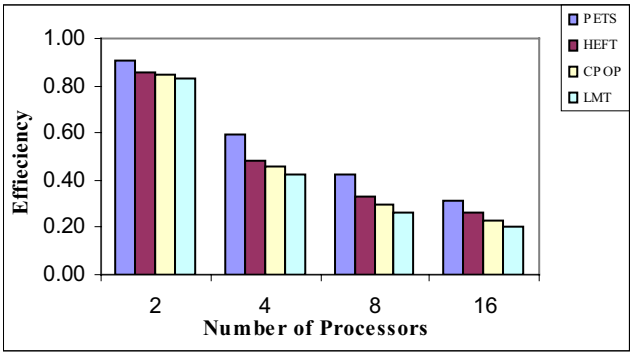
Figure 5. Efficiency

Test Suite 2: In this test suite, we considered application graphs of three real world problems such as Gauss Elimination algorithm, Fast Fourier Transformation and molecular dynamics code given in [6][12]. For the experiment of Gauss elimination applications, heterogeneous computing systems with five processors, CCR and the range percentage parameters given in Section 5.2 are used. Since the structure of the application is known, the parameters such as number of tasks, in degree and out degree are not needed. A new parameter matrix size (m) is used in place of number of tasks (v). The total number of task in a Gaussian elimination graph is equal to $(m^2+m-2)/2$. We evaluated the performance of the algorithms at various matrix sizes from 5 to 15 with an increment of one. The smallest size graph in this experiment has 14 tasks and the largest one has 119 tasks. The simulation results are given in Figure 6, which shows that PETS algorithm outperforms other reported

algorithms by average SLR and Efficiency for various matrix sizes.



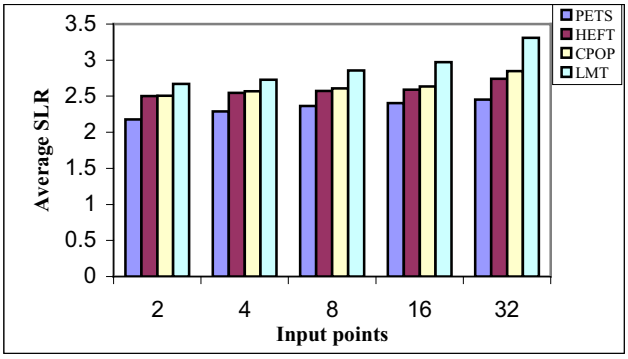
(a) Average SLR



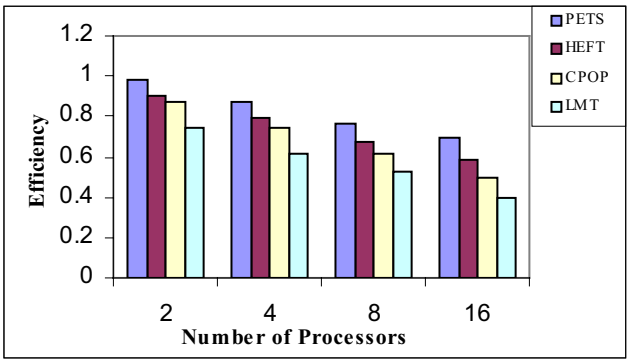
(b) Efficiency

Figure 6. Gaussian Elimination graphs

For FFT related experiments, only the CCR and range percentage parameters given in Section 5.2 were used. Since the structure of the application is known, other parameters such as number of tasks, in degree and out degree are not needed. The number of data points in FFT is another parameter in our experiments, which varies from 2 to 32 incrementing powers of 2. Figure 7(a) presents the average SLR values for FFT graphs at various sizes of input points. Figure 7(b) presents the efficiency values obtained for each of the algorithms with respect to various numbers of processors with graphs of 32 data points.



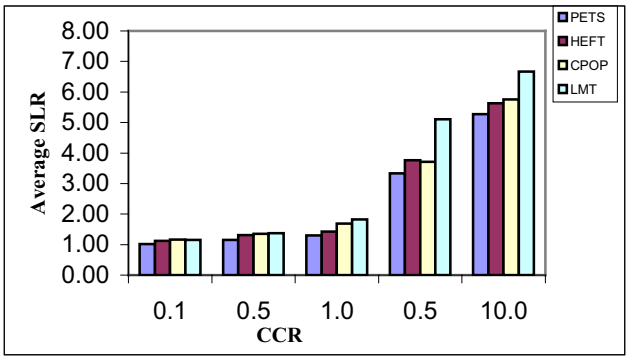
(a) Average SLR



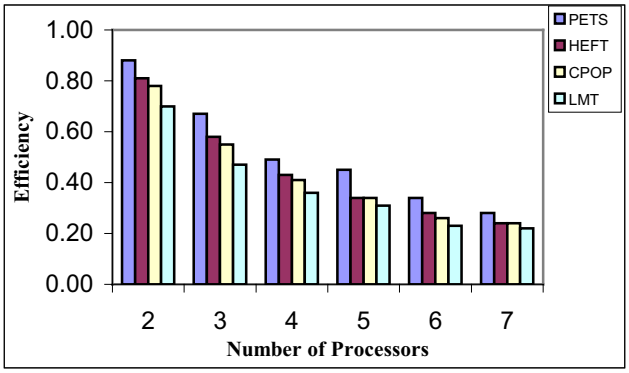
(b) Efficiency

Figure 7. FFT Application graphs

The task graph of the molecular dynamics code given in [6],[12] is also part of our experiment since it has an irregular task graph. Since the number of task is fixed in the application and the structure of the application is known, only the values of CCR and the range percentage parameters given in Section 5.2 are used in the experiments. Figure 8 shows the performance of the algorithms (Average SLR and Efficiency) with respect to five different CCR values when the number of processors is equal to seven. The simulation results show that PETS algorithm substantially outperforms HEFT, CPOP and LMT algorithms.



(a) Average SLR



(b) Efficiency

Figure 8. Molecular Dynamics Structure

Proceedings of the 4th International Symposium on Parallel and Distributed Computing (ISPDC'05)
0-7695-2434-6/05 $20.00 © 2005 IEEE

Table 4. Pair-Wise Comparison of the Scheduling Algorithms

Algorithm		PETS	HEFT	CPOP	LMT	COMBINED
PETS	Better	*	441	498	607	71%
	Equal		198	140	72	19%
	Worse		81	82	41	10%
HEFT	Better	81	*	556	657	61%
	Equal	198		38	27	12%
	Worse	441		126	36	27%
CPOP	Better	82	126	*	634	39%
	Equal	140	38		40	10%
	Worse	498	556		46	51%
LMT	Better	41	36	46	*	7%
	Equal	72	27	40		9%
	Worse	607	657	634		84%

6. Conclusion

The task scheduling algorithm PETS proposed here has been proven to be better for scheduling DAG structured applications onto heterogeneous computing system in terms of performance matrices (average schedule length ratio, speedup, efficiency, frequency of best results) and scheduling time. The performance of the PETS algorithm has been observed experimentally by using large set of randomly generated task graphs with various characteristics and application graphs of several real world problems such as Gaussian Elimination, Fast Fourier Transformation and Molecular Dynamics code. The simulation results confirm that PETS algorithm substantially better that of the existing algorithms such as LMT, CPOP and HEFT in terms performance matrices. The complexity of PETS algorithm is O ($v + e$) ($p+log\ v$), which is less when compared with other scheduling algorithms reported in this paper. We have planned to extend this algorithm for arbitrary-connected networks and also for the dynamic networks.

References

[1] R.L. Graham, L.E. Lawler, J.K. Lenstra, and A.H. Kan, "Optimization and Approximation in Deterministic Sequencing and Scheduling: A Survey", *Annals of Discrete Mathematics*, pp. 287-326, 1979.

[2] T. Cassavant and J.A. Kuhl, "Taxonomy of Scheduling in General Purpose Distributed Memory Systems", *IEEE Trans. Software Engineering*, vol. 14, no. 2, pp. 141-154, 1988.

[3] C.C. Hui and S.T. Chanson, "Allocating Task Interaction Graphs to Processors in Heterogeneous Networks", *IEEE Trans. Parallel and Distributed Systems*, vol. 8, no. 9, pp. 908-926, Sept. 1997.

[4] H.EI-Rewini and T.G.Lewis, " Scheduling Parallel Program Tasks onto Arbitrary Target Machines,", *Journal of parallel and Distributed Computing*, vol.9, pp.138-153, 1990.

[5] M.Iverson, F.Ozguner and G.Follen, "Parallelizing Existing Applications in a Distributed Heterogeneous Environments", *Proc. Heterogeneous Computing Workshop*, pp.93-100,1995

[6] H. Topcuglou, S. Hariri and M.Y. Wu, "Performance Effective and Low-Complexity Task Scheduling for Heterogeneous Computing", IEEE Trans. on Parallel and Distributed Systems, vol 13, No.3, Feb' 2002.

[7] M. Kafil and I. Ahmed, "Optimal Task Assignment in Heterogeneous Distributed Computing Systems,", *IEEE Concurrency*, vol. 6, no. 3, pp. 42-51, July-Sept. 1998.

[8] M.K. Dhodhi, I.Ahmad, A. Yatama, "An Integrated Technique for Task Matching and Scheduling onto Distributed Heterogeneous Computing Systems", *Journal of parallel and distributed computing* ", 62, pp. 1338-1361, 2002.

[9] Atakan Dogan and Fusun Ozguner, "LDBS: A Duplication Based Scheduling Algorithm for Heterogeneous Computing Systems", *Proc. Int'l conf. Parallel Processing (ICPP'02).*

[10] Sanjeev Basker and Prashanth C.SaiRanga, "Scheduling Directed A-cyclic Task Graphs On Heterogeneous Network of Workstations to Minimize Schedule Length", *Proc. ICPPW*, 2003.

[11] Rashmi Bajaj and D.P. Agrawal, "Improving Scheduling of Tasks in a Heterogeneous Environments,", *IEEE Trans. on Parallel and Distributed Systems*, vol. 15, No.2, Feb' 2004.

[12] S.J. Kim and J.C. Browne, "A General Approach to a Mapping of Parallel Computation upon Multiprocessors Architectures,", *Proc. int'l conf. parallel processing*, vol. 2, pp. 1-8,1988.