Journal of Computer Sciences 3 (2): 94-103, 2007
ISSN 1549-3636


# Low Complexity Performance Effective Task Scheduling Algorithm for Heterogeneous Computing Environments

E. Ilavarasan and P. Thambidurai
Department of Computer Science and Engineering and Information Technology
Pondicherry Engineering College, Pondicherry – 605014, India

**Abstract:** A heterogeneous computing environment is a suite of heterogeneous processors interconnected by high-speed networks, thereby promising high speed processing of computationally intensive applications with diverse computing needs. Scheduling of an application modeled by Directed Acyclic Graph (DAG) is a key issue when aiming at high performance in this kind of environment. The problem is generally addressed in terms of task scheduling, where tasks are the schedulable units of a program. The task scheduling problems have been shown to be NP-complete in general as well as several restricted cases. In this study we present a simple scheduling algorithm based on list scheduling, namely, low complexity Performance Effective Task Scheduling (PETS) algorithm for heterogeneous computing systems with complexity O (e) (p+ log v), which provides effective results for applications represented by DAGs. The analysis and experiments based on both randomly generated graphs and graphs of some real applications show that the PETS algorithm substantially outperforms the existing scheduling algorithms such as Heterogeneous Earliest Finish Time (HEFT), Critical-Path-On a Processor (CPOP) and Levelized Min Time (LMT), in terms of schedule length ratio, speedup, efficiency, running time and frequency of best results.

**Key words:** DAG, task graph, task scheduling, heterogeneous computing system, schedule length, speedup, efficiency

## INTRODUTION

A growing emphasis on concurrent processing of jobs has lead to an increased acceptance of heterogeneous computing environments and the availability of a network of processors makes a cost-effective utilization of underlying parallelism for applications like weather modeling, image processing, real-time and distributed database systems. A well-known strategy behind efficient execution of a huge application on a heterogeneous computing environment 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 Acyclic 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 of task scheduling 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] for which optimal solutions can be found only after an exhaustive search.

Efficient application scheduling is critical for achieving high performance in heterogeneous computing systems. Because of its key importance on performance, the task scheduling problem in general has been extensively studied and various heuristics have been proposed in the literature[3-17]. These heuristics are classified into a variety of categories such as list scheduling algorithms, clustering algorithms, guided random search methods and task duplication based algorithms.

In list scheduling algorithms[4-6], an ordered list of tasks is constructed by assigning priority to each task and the tasks are selected for execution based on their priority. List scheduling algorithms are generally preferred since they generate good quality schedules with less complexity. List scheduling algorithms such as Mapping Heuristic (MH)[4], Levelized Min

**Corresponding Author:** P. Thambidurai, Department of Computer Science & Engineering and Information Technology, Pondicherry Engineering College, Pondicherry – 605014, India

Time(LMT)[5], Heterogeneous Earliest Finish Time (HEFT)[6] and Critical Path On a processor (CPOP)[6] are well known task scheduling algorithms for heterogeneous system.

Clustering algorithms[7-9] try to schedule heavily communicating tasks onto the same processor, even if other processors are available, thereby trading off parallelism with interprocess communication. It is also known as three phase scheduling. In the first phase, heavily communicating tasks are grouped into a set of clusters (unbounded) using linear or nonlinear clustering heuristics and in the second phase, clusters are mapped onto the set of available processors using communication sensitive or insensitive heuristics. In the third phase cluster merging or de-clustering is done based on the available number of processors. Task Duplication based scheduling Scheme (TDS)[8] and Clustering for Heterogeneous Processors (CHP)[9] are some well known task scheduling algorithm based on clustering approach.

Genetic algorithms[10-13] are of the most widely studied guided random search techniques for the task scheduling problem. Among these algorithms, the task matching and scheduling algorithm using a genetic approach[10], Problem–Space Genetic Algorithm (PSGA)[11] and Push-Pull[12] are proposed for heterogeneous processors and an incremental genetic algorithm (GA)[13] for the homogeneous processors. Although genetic algorithms provide good quality schedules, their execution times are significantly higher than other alternatives. Extensive tests are required to find optimal values for the set of control parameters used in GA-based solutions[14].

In task duplication based algorithms[15-17], tasks are duplicated on more than one processor to reduce the waiting time of the dependent tasks. It is an interesting approach that has been blended with both list scheduling and clustering-based techniques by various researchers. Critical Path Fast Duplication (CPFD)[15], Heterogeneous Critical Node First (HCNF)[16] and Task duplication Algorithm for Network of Heterogeneous system (TANH)[17] are a few algorithms proposed in the literature for heterogeneous system using task duplication.

Among the various scheduling algorithms, list scheduling algorithms are generally preferred for task scheduling, since they produce good schedule with less time. But, the reported list scheduling algorithms in this study, such as LMT, CPOP and HEFT algorithms are complex in nature and take higher complexity. Moreover the LMT algorithm does not utilize the earliest idle time slot between two already scheduled tasks on a processor. Because of this, the schedule length generated by the LMT algorithm is not the minimum always. The HEFT algorithm uses a recursive procedure to compute the rank of a task by traversing the graph upwards from the exit task. The rank of a task is the length of the critical path from the exit task to that task. The rank of a task in the CPOP algorithm is calculated in the reverse fashion, i.e., traversing the task graph downwards from the entry task. The rank of a task is length of the critical path from the entry task to that task. The rank computation is recursive procedure and also complex in both the algorithms. 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 frequency of best results) and a cost metric (scheduling time). We proposed a new algorithm called PETS algorithm, which gives the best performance in terms of performance and cost metrics for DAG structured applications compared to the exiting scheduling algorithms such as LMT and HEFT and CPOP reported in this study.

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. The symbol $<$ 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 $v_i$. 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 transmitted from task $v_i$ to task $v_k$. In a given task graph, a task without any parent is called an *entry task* and a task without any child is called *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.

A heterogeneous computing system consists of a set $P = \{p_j : j =1,\ldots, m\}$ 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 $m \times m$ matrix, $R_{m \times m}$. W is a $n \times m$ computation cost matrix in which each $w_{ij}$ gives the Estimated Computation Time *(ECT)* to complete task $v_i$ on processor $p_j$ where 1<=i<=n and 1<=j<=m. The *ECT* value of a task may be different on different processor depending on the processors computational capability. The task executions of a given application are assumed to be non-preemptive. The communication cost between two processors $p_x$ and processor $p_y$, 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. In this study, 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 processors $p_x$) to task $v_k$ (scheduled on processor $p_y$) is defined by

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

Otherwise, $C_{i,k} = 0$ when both the tasks $v_i$ and $v_k$ are scheduled on the same processor. Further, for illustration, 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 11 tasks and its computation cost matrix given in[6] are shown in Fig. 1 and Table 1.

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.

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

$EFT(v_i,p_j) = W_{ij}+EST(v_i,p_j)$ (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 non insertion-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

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

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

**Related works:** In this section we present the related task scheduling algorithms for heterogeneous computing environment that we used for comparison with our algorithm, which are Levelized Min Time algorithm[5], Heterogeneous Earliest Finish Time algorithm[6] and Critical Path On a processor algorithm[6].

**Levelized min time (LMT) algorithm:** It is a two-phase algorithm. The first phase groups the tasks that can be executed in parallel in a level by level fashion. The second phase is a greedy method that assigns each task to the "fastest" available processor. A task in a lower level has higher priority for scheduling than a task in a higher level. Within the same level, the task with the highest average computation cost has the highest priority. If the number of tasks in a level is greater than the number of available processors, the fine-grain tasks are merged into a coarse-grain task

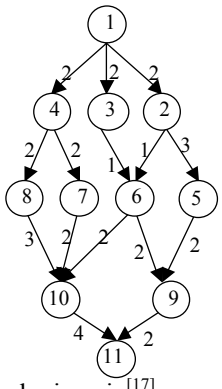



Fig. 1: Task Graph given in[17]

Table 1: Computation cost matrix (W) given in[17]

| Task $v_i$ | $P_1$ | $P_2$ | $P_3$ |
|---|---|---|---|
| 1 | 4 | 4 | 4 |
| 2 | 5 | 5 | 5 |
| 3 | 4 | 6 | 4 |
| 4 | 3 | 3 | 3 |
| 5 | 3 | 5 | 3 |
| 6 | 3 | 7 | 2 |
| 7 | 5 | 8 | 5 |
| 8 | 2 | 4 | 5 |
| 9 | 5 | 6 | 7 |
| 10 | 3 | 7 | 5 |
| 11 | 5 | 6 | 7 |

until the number of tasks is equal to the number of processors. Then the tasks are sorted in reverse order (largest task first) based on average computation time. Beginning from the largest task, each task will be assigned to the processor that minimizes the sum of computation cost of the task and the communication costs with tasks in the previous layers and does not have any scheduled task at the same level. For a fully connected graph, the time complexity is O ($v^2$ x $p^2$) when there are v tasks and p processors.

**Heterogeneous earliest finish time (HEFT) algorithm:** It is also two-phase task scheduling algorithm for a bounded number of heterogeneous processors. The first phase namely, task-prioritizing phase is to assign the priority to all tasks. To assign priority, the upward rank of each task is computed. The upward rank of a task is the critical path of that task, which is the highest sum of communication time and average execution time starting from that task to exit task. Based on upward rank priority will be assigned to each task. The second phase (processor selection phase) is to schedule the tasks onto the processors that give the earliest finish time for the task. It uses 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, should be at least capable of computation cost of the task to be scheduled and also scheduling on this idle time slot should preserve precedence constraints. The time complexity of HEFT algorithm is equal to O ($v^2$ x p) where *v* is the number of tasks in a dense graph and *p* is the number of processors.

**Critical path on a processors (CPOP) algorithm:** This is similar to HEFT algorithm, but it uses different strategies in each phase. The first phase namely, task-prioritizing phase is to assign the priority to each task. In this phase, upward rank (given in HEFT algorithm) and downward rank values for all tasks are computed. The downward rank is computed by adding average execution time and communication time starting from entry task to the task excluding execution time of the task for which downward rank is computed. The sum of downward and upward rank is used to assign the priority to each task. Initially, the entry task is the selected task and marked as a critical path task. An immediate successor (of the selected task) that has the highest priority value is selected and it is marked as a critical path task. This process is repeated until the exit node is reached.

In the second phase, task with highest priority is selected for execution. If the selected task in on the critical path, then it is scheduled on the critical path processor. The critical processor is the one that minimizes the cumulative computation costs of the tasks on the critical path; otherwise, it is assigned to a processor, which minimizes the earliest execution finish time of the task. The time complexity of CPOP algorithm is equal to O ($v^2$ x p) where *v* is the number of tasks in a dense graph and *p* is the number of processors.

**Low complexity 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 each phase of the algorithm is given below:

**Level sorting phase:** In the level sorting 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 contain entry task. 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. For example, for the task graph given in Fig. 1, there are 5 l*evels*; *level 1* consists of task 1 *(entry task)*, *level 2* consists of task 2, 3 and 4, *level 3* consists of task 5, 6, 7 and 8, *level 4* consists of task 9 and 10 and *level 5* consists of task 11 *(exit task).*

**Task prioritization phase:** In the task prioritization phase, priority is computed and assigned to each task. For assigning priority to a task, we have defined three attributes namely, *Average Computation Cost (ACC), Data Transfer Cost (DTC)* and the *Rank of Predecessor Task (RPT).* The *ACC* of a task is the average computation cost on all the *m* processors and it is computed by using Eqn.(5)

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

The *DTC* of a task $v_i$ is the amount of communication cost incurred to transfer the data from task $v_i$ to all its immediate successor tasks and it is computed at each *level l* using Eqn.(6)

$$DTC(v_i) = \sum_{j=1}^{n} C_{i,j} : i < j \quad , \text{ where } n \text{ is the number of nodes in the next level}$$
$$= 0, \text{ for exit tasks} \quad (6)$$

The *RPT* of a task $v_i$ is the highest rank of all its immediate predecessor tasks and it computed using Eqn.(7)

$RPT(v_i) = Max\{rank(v_1),rank(v_2),,,...,rank(v_h)\}$
*Where* $v_1,v_2.....v_h$ are the immediate predecessors of $v_i$
= 0, for entry task (7)

Rank is computed for each task $v_i$ based on its *ACC, DTC* and *RPT* values. We have used the maximum rank of predecessor tasks of task $v_i$ as one of the parameter to calculate the rank of the task $v_i$ and the rank computation is given in Eqn. (8).

$rank(v_i) = round\{ACC(v_i)+DTC(v_i)+RPT(v_i)\}$ (8)

Priority is assigned to all the tasks at each *level l*, 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. Tie, if any, is broken using *ACC* value. The task with minimum *ACC* value receives higher priority. For example, for the task graph given in Fig. 1, the *ACC, DTC, RPT*, rank and priority values are computed as follows: For task $v_1$, there are three immediate successor tasks $v_2$, $v_3$, $v_4$ and the communication cost between $v_1$ and to these tasks are 2, 2, 2 respectively. Hence, the *DTC* of task $v_1$ is 6 (2+2+2). The RPT value of task $v_1$ is 0, since it is the entry task. The ACC value of task $v_1$ is 4 and the rank value of the task $v_1$ is 10 (4+6+0). The priority of task $v_1$ is 1, since it is the only task in level 1. Likewise the *ACC, DTC, RPT, rank* and *priority* are computed for all tasks in the task graph and the computed value is shown in Table 2.

Table 2: The *DTC, ACC, RPT, rank* and *priority* values for the tasks in Fig. 1

| Level | Task | DTC | ACC | RPT | rank | Priority |
|---|---|---|---|---|---|---|
| 1 | 1 | 6 | 4 | 0 | 10 | 1 |
| 2 | 2 | 4 | 5 | 10 | 19 | 1 |
| 2 | 3 | 1 | 5.25 | 10 | 16 | 3 |
| 2 | 4 | 4 | 3 | 10 | 17 | 2 |
| 3 | 5 | 2 | 3.75 | 19 | 25 | 2 |
| 3 | 6 | 4 | 3.5 | 19 | 27 | 1 |
| 3 | 7 | 2 | 5.75 | 17 | 25 | 3 |
| 3 | 8 | 3 | 3.5 | 17 | 24 | 4 |
| 4 | 9 | 2 | 5.75 | 26.5 | 34 | 2 |
| 4 | 10 | 4 | 4.25 | 26.5 | 35 | 1 |
| 5 | 11 | 0 | 6.5 | 34.75 | 41 | 1 |

**Processor selection phase:** In the processor selection phase, the processor, which gives minimum *EFT* for a task is selected and the task is assigned to that processor. It has an insertion-based policy, which considers the possible insertion of a task in an earliest

Table 3: The Computed EST, EFT value on processors $P_1$, $P_2$, $P_3$ for the tasks in Fig. 1

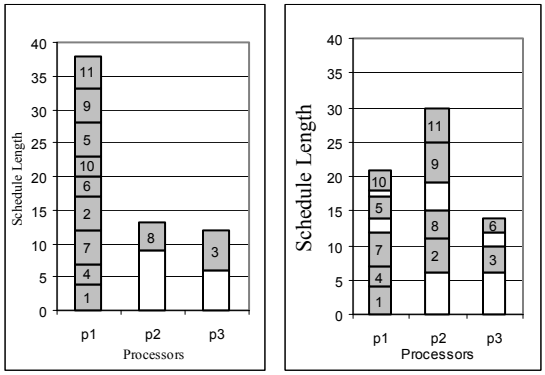
| | Processors | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | $P_1$ | | $P_2$ | | $P_3$ | | | |
| Task | EST | EFT | EST | EFT | EST | EFT | Predecessor tasks | Processor selected |
| 1 | 0 | 4 | 0 | 4 | 0 | 4 | Null | $P_1$ |
| 2 | 4 | 9 | 6 | 11 | 6 | 11 | 1 | $P_1$ |
| 4 | 9 | 12 | 6 | 9 | 6 | 9 | 1 | $P_2$ |
| 3 | 9 | 13 | 9 | 15 | 6 | 10 | 1 | $P_3$ |
| 5 | 9 | 13 | 12 | 17 | 12 | 15 | 2 | $P_1$ |
| 8 | 12 | 14 | 9 | 13 | 11 | 16 | 4 | $P_2$ |
| 6 | 12 | 15 | 13 | 20 | 10 | 12 | 2,3 | $P_3$ |
| 7 | 12 | 17 | 13 | 21 | 12 | 17 | 4 | $P_1$ |
| 10 | 17 | 20 | 19 | 26 | 19 | 24 | 6,7,8 | $P_1$ |
| 9 | 20 | 25 | 14 | 20 | 14 | 21 | 5,6 | $P_2$ |
| 11 | 22 | 27 | 24 | 30 | 24 | 31 | 9,10 | $P_1$ |

***Step 1**: read* the DAG, associated attributes values and the number of processor P;
***Step 2:** for* all tasks $v_k$ at each level *Li* do
***Step 3:** begin*
compute $ACC(v_k)$,$DTC(v_k)$ *and RPT* ($v_k$ );
$rank(v_k) = ACC(v_k) + DTC(v_k) + RPT$ ($v_k$ );
tie, if any, is broken based on *ACC* value, the task with minimum *ACC* value receives the higher priority followed by the task with next minimum *ACC* value and so on;
**Step 4:** construct a priority queue using ranks;
***Step 5:** while* there are unscheduled tasks in the queue *do*
*begin*
select the first task, $v_k$ from the priority queue for scheduling;
*for* each processor $p_k$ in the processor set P *do*
*begin*
compute *EFT* ($v_k$,$p_k$) value using insertion based scheduling policy;
assign the task $v_k$ to processor $p_k$, which minimizes the *EFT*;
*end;*
*end;*
*end;*
***Step 6:** end.*

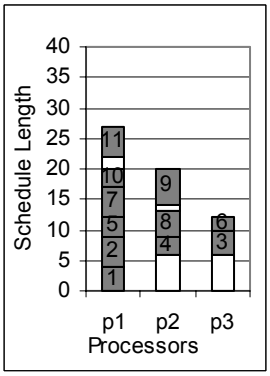
Fig. 2: Proposed PETS algorithm

idle time slot between two already scheduled tasks on a processor. At each level, the *EST* and *EFT* value of each task on every processor is computed using Eqn. (2) and (3). Calculation of *EST* and *EFT* values for the task graph in Fig. 1 is illustrated below: For example, for the task 1, $EST(1,P_1)=0$, $EFT(1,P_1)=4$, $EST(1,P_2)=0$,$EFT(1,P_2)=4$ and $EST(1,P_3)=0$, $EFT(1,P_3)=4$). For task 4, $EST(4,P_1)= max\{9, max(4)\}=9$, $EFT(4, P_1)=9+3=12$, $EST(4, P_2) = max\{0, max(6)\}=6$, $EFT(4,P_2) = 6+3=9$ and $EST(4,P_3) = max\{0,max(6)\}=6$, $EFT(4, P_3)=6+3=9$. Likewise *EST* and *EFT* values for all the tasks in the graph are computed. 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 in each level are scheduled on to the suitable processors.

The processors selected for assigning the tasks in Fig. 1 is as follows: For example, task 1 is the entry task and all the three processors $P_1$, $P_2$ and $P_3$ give the same *EFT* value, one of the processor is selected randomly. Here $P_1$ is selected for executing task 1. For task 2, the *EFT* value on three processors $P_1$, $P_2$ and $P_3$ are 9, 11 and 11. Since $P_1$ gives minimum *EFT*, it is selected for executing task 2. Similarly all other tasks in the task graph are scheduled on to the suitable processor. The processor selected for executing each of the tasks in Fig. 1 is shown in Table 3. The proposed algorithm is given in Fig. 2.




(a) CPOP Algorithm (b) HEFT Algorithm




(c) PETS Algorithm

Fig. 3: The schedule length generated by CPOP, HEFT and PETS algorithms

The time complexity of PETS algorithm is equal to $O(e)(p + \log v)$, where $v$ and $e$ are the number of tasks and edges respectively and $p$ is the number of processors.

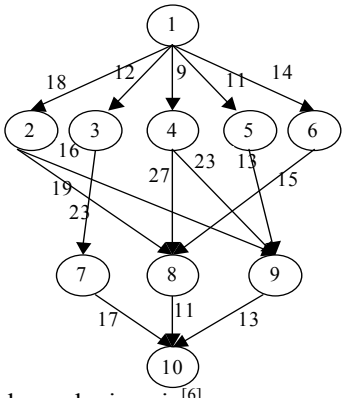
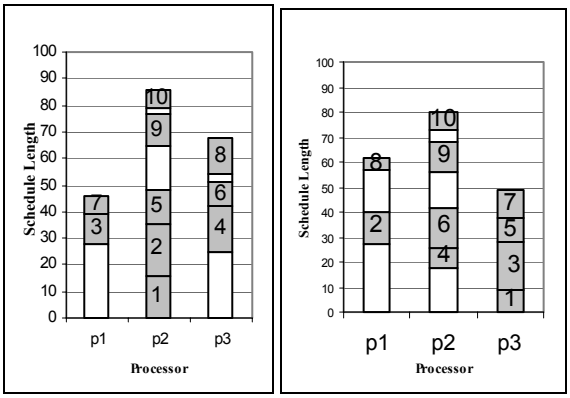


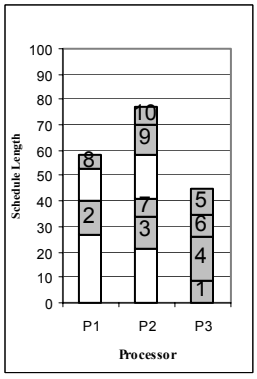

Fig. 4: Task graph given in[6]

Table 4: Computation cost matrix (W) 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 |




(a) CPOP Algorithm (b) HEFT Algorithm




(c) PETS Algorithm

Fig. 5: The schedule length generated by CPOP, HEFT and PETS algorithms

We used breadth first search for level sorting which takes *O(e)* time complexity only, since the implementation is done by adjacency lists. If the adjacency matrix is used for implementation the complexity is $O(v^2)$. 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 overall complexity of the algorithm is *O*(*e*) (*p*+*log v)*. As an illustration, Fig. 3 presents the schedules obtained by the CPOP, HEFT and PETS algorithms for the sample DAG of Fig. 1. The schedule length generated by PETS algorithm is 27, is shorter than the schedule length produced by HEFT and CPOP algorithms which are 30 and 38 respectively.

We have also obtained the schedule generated by the PETS, HEFT and CPOP algorithms for the task graph given in Fig. 4 and its computation cost matrix given in Table 4. Figure 5 presents the schedule generated by CPOP, HEFT and PETS algorithms. The schedule length generated by PETS algorithm is 77, is shorter than the schedule length produced by HEFT and CPOP algorithms which are 80 and 87 respectively.

## PERFORMANCE ANALYSES AND DISCUSSION

In this section, we present the comparative evaluation of proposed PETS algorithm and the existing algorithms for heterogeneous system 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. We have used Intel Xeon processors with 1 GHz speed for our experiments.

**Comparison metrics:** We have used the following metrics to evaluate the proposed algorithm.

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

**Speedup:** Speed up is the ratio of the sequential execution time to the parallel execution time.

**Efficiency:** Efficiency is the ratio of the speedup value to the number of processor used to schedule the graph.

**Number of occurrences of the better quality of schedules:** The numbers of times that each algorithm produced better, worse and equal quality of schedules compared to every other algorithm.

**Running time of the algorithms:** The running time (the scheduling time) of an algorithm is its execution time for obtaining the output schedule of a given task graph.

**Randomly generated application graphs:** A random task graph generator has been developed, which 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. If a DAG's CCR value is very low, it can be considered as a computation intensive application. Range percentage of computation costs on processors (η). It 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*Wdag], where Wdag 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:

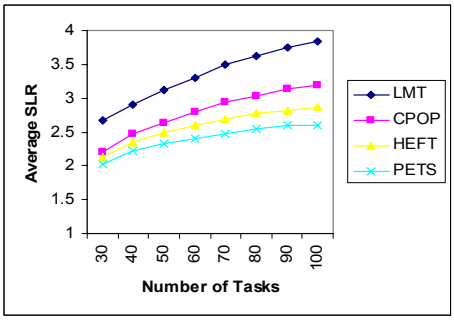
$$W_i*(1-\eta/2)<=W_{i,j}<=W_i*(1+\eta/2) \quad (9)$$

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}.

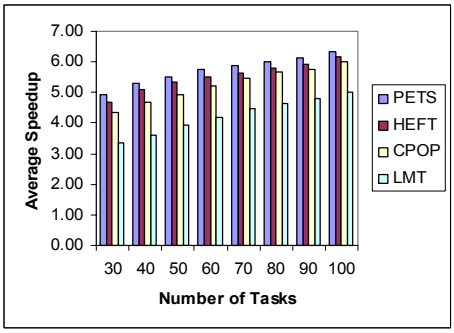
## RESULTS

The experimental results are organized in two major test suites.

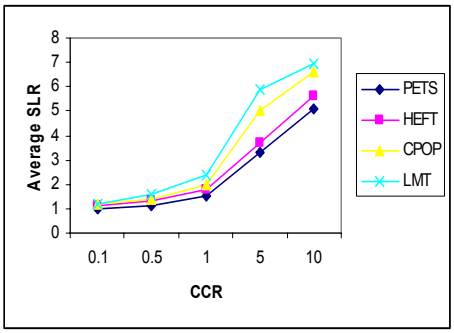
**Test suite 1:** In this test suite, we evaluated the quality of schedules generated by each of the algorithms for random task graphs with respect to various graph characteristics values. We have generated a large set of random task graphs with different characteristics and scheduled these task graphs on to a heterogeneous computing system consists of 15 processors. The average SLR and speedup generated by each of the algorithm are plotted and are shown in Fig. 6a and Fig. 6b. Each data point in the reported graph is the average of the data obtained in 240 experiments.



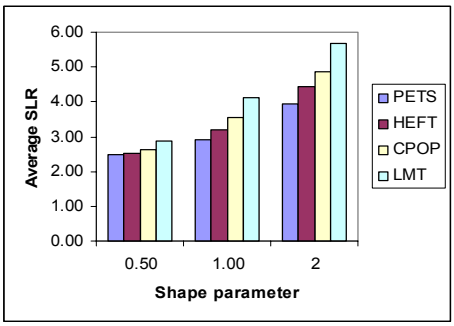

(a) Average SLR



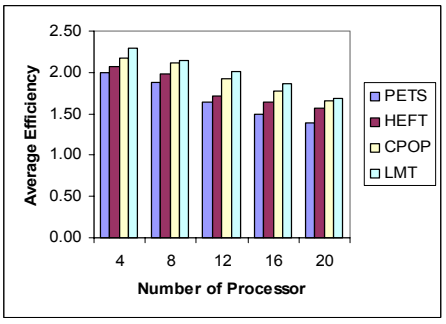

(b) Average Speedup



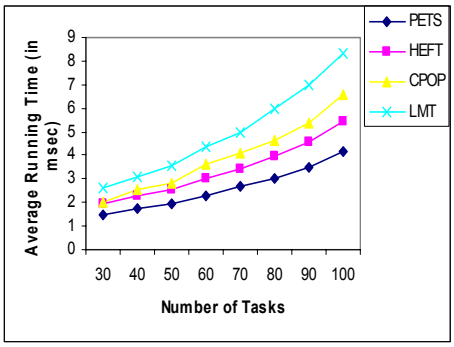

(c) Average SLR for varying CCR




(d) Average SLR for different shape parameter




(e) Average Efficiency



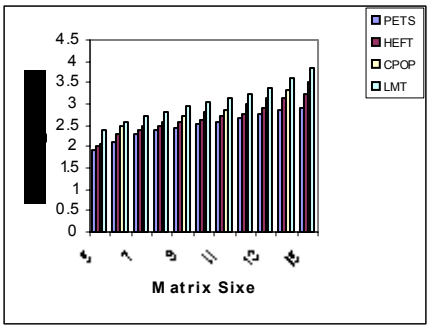

(f) Average Running Time

Fig. 6: Performance of the algorithm for random task graphs

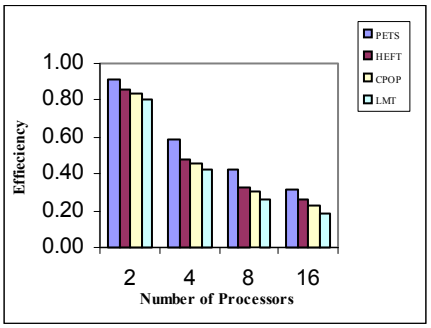
The average SLR value based ranking (starting with minimum ending with maximum) of the algorithms is {PETS, HEFT, CPOP and LMT} and the Speedup value based ranking (starting with maximum and ending with minimum) of the algorithms is {PETS, HEFT, CPOP and LMT}. The average SLR value of the PETS algorithm on all generated graphs is better than the HEFT algorithm by 8%, the CPOP algorithm by 17% and the LMT algorithm by 40%.

We have also evaluated the performance of the algorithm with respect to various CCR and graph structure values and the outcomes of these results are shown in Fig. 6c and Fig. 6d. Each data point in the reported graph is the average of the data obtained in 220 experiments. These experiments also confirm that PETS algorithm substantially outperforms reported algorithms for various CCR value and for different graph structure. Further, we evaluated the efficiency of the algorithms by scheduling task graphs consisting of fixed number of tasks (120) on to heterogeneous computing system consisting of varying number of processors (4,8,12,16,20). For this experiment, we have used 1000 numbers of randomly generated task graphs. The results obtained by this experiment are shown in Fig. 6e. 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. The average running time of each of the algorithms is calculated for the above experiments and it shown in Fig. 6f. The graph shows that the PETS algorithm is the fastest one and the LMT algorithm is the slowest one. On average the PETS algorithm is faster than the HEFT algorithm by 23%, the CPOP algorithm by 39% and the LMT by 48%.

**Test suite 2:** In this test suite, we considered application graphs of three real world problems, such as LU decomposition, Fast Fourier Transformation (FFT) and molecular dynamics code given in[6,15]. For the experiment of LU decomposition, heterogeneous computing systems with five processors, CCR and the range percentage of computation cost values are used.




(a) Average SLR




(b) Efficiency

Fig. 7: Performance of algorithms for LU Decomposition graphs

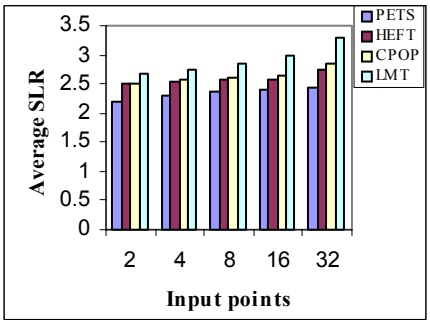
Since the structure of the application is known, the parameters such as number of tasks, in degree and out degree are not needed. A new parameter matrix size (n) is used in place of number of tasks ($v$). The total number of task in a LU decomposition graph is equal to $(n^2+n-2)/2$. We evaluated the 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. Figure 7a gives the average SLR values of the algorithms at various matrix sizes from 5 to 15 with an increment of one, when the number of processors is equal to five.

For the efficiency comparison, the number of processors used in our experiments is varied from 2 to 16; the CCR and range percentage parameters have the same set of values. Figure 7b gives efficiency comparison for LU decomposition graphs when matrix size if 15. The experiments conducted using Gaussian Elimination graphs confirm that PETS outperforms other reported algorithms in terms of average SLR and efficiency.

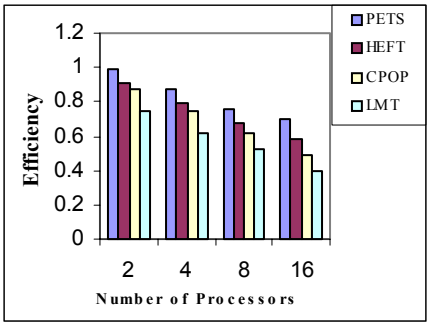
For FFT related experiment the graph characteristics such as CCR and the range percentage of computation cost values are used. Since the structure of the application is known, other parameters such as number of tasks, in degree and out degree are not needed. The number of data points in FFT is another parameter in our experiments, which varies from 2 to 32 incrementing powers of 2. Figure 8a presents the average SLR values for FFT graphs at various sizes of

Table 5: Pair-wise comparison of the scheduling algorithms

| | | PETS | HEFT | CPOP | LMT | COMBINED |
|---|---|---|---|---|---|---|
| PETS | Better | | 8565 | 9467 | 10927 | 79% |
| | Equal | * | 2452 | 1112 | 735 | 12% |
| | Worse | | 1233 | 1671 | 588 | 9% |
| HEFT | Better | 1233 | | 9579 | 11147 | 60% |
| | Equal | 2452 | * | 613 | 491 | 10% |
| | Worse | 8565 | | 2058 | 612 | 30% |
| CPOP | Better | 1671 | 2058 | | 10927 | 40% |
| | Equal | 1112 | 613 | * | 735 | 7% |
| | Worse | 9467 | 9579 | | 588 | 53% |
| LMT | Better | 588 | 612 | 588 | | 5% |
| | Equal | 735 | 491 | 735 | * | 5% |
| | Worse | 10927 | 11147 | 10937 | | 90% |

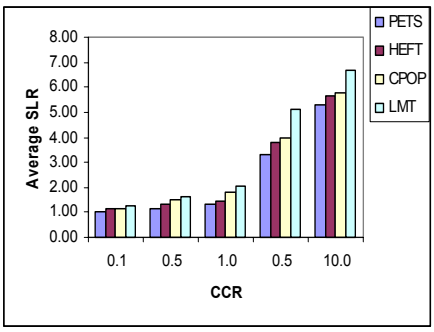



(a) Average SLR

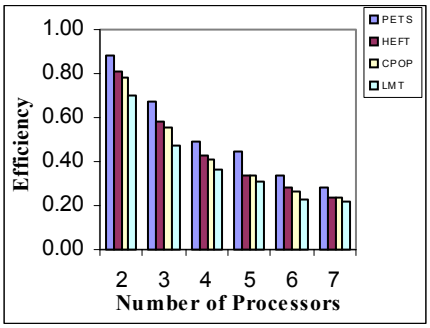



(b) Efficiency

Fig. 8: Performance of the algorithms FFT application graphs




(a) Average SLR




(b) Efficiency

Fig. 9: Performance of the algorithm for molecular dynamics structure

input points. Figure 8b presents the efficiency values obtained for each of the algorithms with respect to various numbers of processors with graphs of 32 data points

The task graph of the molecular dynamics code given in[6] 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, the graph characteristics CCR and the range percentage of computation cost values are alone used. Figure 9a shows the performance of the algorithms (Average SLR) with respect to five different CCR values when number of processors is equal to six. The efficiency comparison of the algorithms is given in Fig. 9b. The experiments conducted using molecular dynamics graphs also show the supremacy of the PETS algorithm over the HEFT, CPOP and LMT algorithms.

Finally we present the frequency of quality schedule produced by each of the algorithms. To obtain the frequency of quality schedule produced by each of the algorithm, we counted the number of times that each scheduling algorithm in the experiments conducted in *test suite 1* and *test suite 2* (around 12250 experiments*)* produced better , worse, or equal schedule length compared to every other algorithm. Table 5 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, CPOP and LMT}.

## 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 average schedule length ratio, speedup, efficiency, running time and frequency of best results. 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 three world problems such as LU decomposition, Fast Fourier Transformation and Molecular Dynamics code. The simulation results confirm that PETS algorithm is substantially better than that of the existing algorithms such as LMT, CPOP and HEFT. The complexity of PETS algorithm is O (*e*) (*p+log v)*, which is less when compared with other scheduling algorithms reported in this study. We have planned to extend this algorithm for arbitrary-connected networks.

## REFERENCES


1. Graham, R.L., L.E. Lawler, J.K. Lenstra and A.H. Kan, 1979. Optimization and approximation in deterministic sequencing and scheduling: A survey. Ann. Discrete Math., pp: 287-326.
2. Cassavant. T. and J.A. Kuhl, 1988. Taxonomy of scheduling in general purpose distributed memory systems. IEEE Trans. Software Engg., 14: 141-154.
3. Hui, C.C. and S.T. Chanson, 1997. Allocating task interaction graphs to processors in heterogeneous networks. IEEE Trans. Parallel and Distributed Systems, 8: 908-926.
4. EI-Rewini, H. and T.G. Lewis, 1990. Scheduling parallel program tasks onto arbitrary target machines. J. Parallel and Distributed Computing, 9: 138-153.
5. Iverson, M., F. Ozguner and G. Follen, 1995. Parallelizing existing applications in a distributed heterogeneous environments. Proc. Heterogeneous Computing Workshop, pp: 93-100.
6. Topcuoglu, H., S. Hariri and M.Y. Wu, 2002. Performance effective and low-complexity task scheduling for heterogeneous computing. IEEE Trans. on Parallel and Distributed Systems, 13: 3.
7. Kafil, M. and I. Ahmed, 1998. Optimal task assignment in heterogeneous distributed computing systems. IEEE Concurrency, 6: 42-51.
8. Ranaweera, A. and D.P. Agrawal, 2000. A task duplication based algorithm for heterogeneous systems. Proc. IPDPS, pp: 445-450.
9. Cristina Boeres, Jos´e Viterbo Filho and Vinod E. F. Rebello, 2004. A cluster-based strategy for scheduling task on heterogeneous processors. Proc. 16th Symp. on Computer Architecture and High Performance Computing (SBAC-PAD).
10. Wang, L., H.J. Siegel, V.P. Rowchoudhry and A.A. Maciejewski, 1997. Task matching and scheduling in heterogeneous computing environments using a genetic algorithm-based approach. J. Parallel and Distributed Computing, 47: 8-22.
11. Dhodhi, M.K., I. Ahmad, A. Yatama, 2002. An integrated technique for task matching and scheduling onto distributed heterogeneous computing systems. J. Parallel and Distributed Computing, 62: 1338-1361.
12. Kim, S.C. and S. Lee, 2005. Push-pull: Guided search DAG scheduling for heterogeneous clusters. Proc. Intl. Conf. Parallel Processing (ICPP'05).
13. Annie, S.W., H. Yu, S. Jin, K.-C. Lin, 2004. An incremental genetic algorithm approach to multiprocessor scheduling. IEEE Trans. on Parallel and Distributed Systems, 15: 824-834.
14. Braun, T.D., H.J. Siegel, N. Beck and L.L. Boloni *et al*., 1999. A comparison study of static mapping heuristics for a class of meta-tasks on heterogeneous computing systems. Proc. 8th Workshop on Heterogeneous Processing, pp: 15-29.
15. Ahmed, I. and Y. Kwok, 1998. On exploiting task duplication in parallel program scheduling. IEEE Trans. on Parallel and Distributed Systems, 9: 872-892.
16. Basker, S. and P.C. SaiRanga, 2003. Scheduling directed a-cyclic task graphs on heterogeneous network of workstations to minimize schedule length. Proc. ICPPW.
17. Bajaj, R and D.P. Agrawal, 2004. Improving scheduling of tasks in a heterogeneous environments. IEEE Trans. on Parallel and Distributed Systems, 15: 107-118.