HIGH PERFORMANCE AND ENERGY EFFICIENT TASK SCHEDULING ALGORITHM FOR HETEROGENEOUS MOBILE COMPUTING SYSTEM

E. Ilavarasan[1] and R. Manoharan[2]

Department of Computer Science and Engineering
Pondicherry Engineering College, Puducherry, India
[1]eilavarasan @yahoo.com, [2]rmanoharan@yahoo.com

ABSTRACT

Heterogeneous Mobile Computing System (HMCS) consists of battery operated portable heterogeneous mobile nodes interconnected by wireless medium are increasingly being used in many areas of science, engineering and business. The advancements in the computing and communication technologies excel the mobile computing devices with the potential to execute larger application. However, execution of larger program is constrained by the availability of energy/power, node mobility and availability. A significant amount of work has been carried out to execute meta (independent) tasks in mobile computing system by consuming minimum energy/power and only a very few work has been carried out for the execution of larger program represented by Directed Acyclic Graph(DAG) in mobile computing system. Therefore, in this paper, the problem of scheduling the tasks of a DAG onto the mobile computing system has been explored with objectives to minimize either the schedule length or energy/power consumption or both. A new task scheduling algorithm namely, High Performance and energy efficient task Scheduling algorithm for heterogeneous Mobile computing system (HPSM) has been proposed. The performance of the algorithm is evaluated by simulation experiments using a large set of randomly generated task graphs. The experimental results show that the HPSM algorithm significantly minimizes the schedule length or the energy consumption or both.

KEYWORDS

Task scheduling, directed acyclic graph, heterogeneous mobile computing system, schedule length, energy consumption.

1. INTRODUCTION

In the recent days, Heterogeneous Mobile Computing System (HMCS) are increasingly being used in areas like battlefield, disaster management, weather modeling and complex image rendering [1]. Mobile nodes have the potential to execute larger application due to the advancements in computing and communication technologies. However, execution of larger programs in mobile nodes is constrained by the available energy/power, node mobility and availability. Energy consumption is one of the key issues in addition to task scheduling in HMCS due to its unique features such as limited, irreplaceable energy sources and lifetime requirements [2].

To alleviate the problem of energy limitation, several hardware and software based techniques have been proposed and used by today's mobile computing nodes [3], [4]. A significant amount of research has been done in the areas of power resource management in uniprocessors as well as in heterogeneous multiprocessor systems for the execution of meta tasks [5], [6]. In these researches, power/energy management is achieved through voltage scaling, computation off-loading and turning off the idle component.

The task partitioning and scheduling strategies also play an important role for achieving high performance in HMCS in addition to energy consumption. Only an efficient scheduling algorithm can utilize the resources in the HMCS and complete the program execution at the earliest. A partitioning algorithm can be employed to partition a parallel application into a set of precedence-constrained tasks represented in the form of a DAG, whereas a scheduling algorithm can be used to schedule the DAG onto the computational nodes in HMCS. The scheduling of DAG has been widely explored for Heterogeneous Computing Systems (HCS) consisting of permanent heterogeneous processors having continuous power supply. Consequently various task scheduling algorithms have been proposed in the literature and these algorithms are broadly classified into static task scheduling and dynamic task scheduling.

In static or compile-time scheduling, the characteristics of the tasks of an application such as execution times, size of data communicated between the tasks and tasks dependencies are assumed to be known apriori whereas, in dynamic or run-time scheduling this information is known at the run-time. Static task scheduling generates better schedules with lesser scheduling overhead than the dynamic scheduling algorithms. In general task scheduling problem is proven to be NP-complete and hence, several heuristic-based algorithms have been proposed in the literature [7], [8]. The heuristic based task scheduling algorithms are categorized as list-scheduling algorithms, clustering algorithms and task duplication-based algorithms.

The task scheduling algorithms developed for HCS can not be applied directly to HMCS, because the algorithms developed for HCS assume that the processors are permanently available and have continuous power supply. Whereas in the HMCS available energy/power and the node mobility are the major constraints, hence HMCS requires separate task scheduling algorithms. Execution of independent tasks with an objective to minimize the execution time and energy consumption have been explored for HMCS [9], whereas only a very few work has been carried out for the execution of dependent tasks.

The algorithms, namely Energy-Aware Duplication Scheduling algorithm (EADUS) and Time-Energy Balanced Duplication Scheduling algorithm (TEBUS), have been developed for scheduling precedence-constrained tasks on clusters of machines [10]. The EADUS is designed to save the energy by using task replicas to eliminate energy consuming messages, whereas TEBUS aims at making the best tradeoff between energy conservation and performance. These algorithms also can not be applied directly in HMCS since they are proposed for the clusters of wired processors. Six heuristics for mapping an application composed of communicating subtasks with data dependencies on to the heterogeneous ad hoc grid environment are proposed in [11]. The goal of these heuristics is to minimize the average percentage of energy consumed by the application to execute across the machines in the ad hoc grid. In [12] authors formulate an energy-aware scheduling problem for certain architecture of embedded systems and propose a heuristic to solve it. Their algorithm schedules time constrained computational tasks and communication transactions on a Network-on-Chip architecture and aims to minimize energy consumption and meet task deadlines.

However, energy-driven task allocation schemes, mentioned above, concentrate only on energy consumption while scheduling. Consequently, the length of the schedules could be very long, which is unfavorable or in some situations even not bearable. Moreover, the existing task scheduling algorithms do not address the node mobility issue. A successful execution of larger program in HMCS requires both the compile-time and run-time support. Hence, in this paper a two phase task scheduling algorithm namely, HPSM algorithm to schedule the tasks of a DAG onto the HMCS has been proposed. The HPSM algorithm minimizes either the schedule length or the total energy consumed, or both at the compile-time phase. At the run-time, HPSM algorithm reschedules the tasks assigned to a node during compile-time to the other nodes, if that node becomes unavailable due to mobility or energy exhaustion in order to successfully complete the execution.

The performance of the HPSM is compared with HEFT [7] and PETS [8] algorithms after necessary modification to suite for HMCS. The modified PETS and HEFT algorithms are referred in this paper as Performance Effective Task Scheduling algorithm for Mobile computing system (PETSM) and Heterogeneous Earliest Finish Time scheduling algorithm for Mobile computing system (HEFTM) respectively. Experiments conducted by simulation using a large set of randomly generated DAG shows that the HPSM algorithm gives better performance in terms of minimizing energy and schedule length than the HEFTM and PETSM algorithms.

The remainder of the paper is organized as follows. Section 2 defines the task scheduling problem for HMCS and the related terminology. Section 3 presents the proposed task scheduling algorithm HPSM. The complexity of HPSM algorithm is presented in section 4. The phases of the HPSM algorithm are exemplified with an illustration in section 5. Section 6 presents the results of the study and finally conclusions and suggestions for future work are presented in Section 7.

2. TASK SCHEDULING PROBLEM

The task scheduling problem consists of three major components namely, a task graph model (representation of application), a target system and a scheduling algorithm.

An application (parallel program) is divided into set of tasks with some precedence relationships among them and can be represented by a DAG also called a *task graph.* A DAG is represented by the five-tuple $G\ (V, E, P, T, C)$ where:

- V is the set of *vertices* v, and each vertex $v_i \in V$ represents an application task, which is a sequence of instructions that must be executed serially on the same processor in non preemptive manner.
- E is the set of communication edges $e(v_i, v_j)$, and each $e(v_i, v_j) \in E$ represents an edge from task v_i to task v_j. The edge from task v_i to task v_j also represents the precedence constraints such that task v_j cannot start until v_i finishes and sends its data to v_j.
- P is the set $\{p_i : i = 1,\dots, p\}$ of p heterogeneous mobile processors.
- T is the set of computation costs $T\ (v_i, p_j)$, which represent the estimated computation times of task v_i on processor p_j.
- C is the set of communication costs $c(v_i, v_j)$, which represent the communication cost associated with the edges $e(v_i, v_j)$.

Task v_p is a predecessor of task v_i if there is a directed edge originating from v_p and ending at v_i. Likewise, task v_s is a successor of task v_i if there is a directed edge originating from v_i and ending at v_s. Let $pred(v_i)$ be the set of all predecessors of v_i and $succ(v_i)$ be the set of all successors of v_i. Without loss of generality, it can be assumed that there is one entry task and one exit task for a DAG. If there are multiple entry or exit tasks, the multiple tasks can always be connected through a dummy task which has zero computation cost and zero communication cost edges. A sample task graph is given in Figure 1 and the computation cost matrix of the tasks is given in Table 1.

Target computing systems HMCS consists of set P of p heterogeneous mobile computing nodes/processors equipped with wireless communication and networking facilities. There is a centralized access point or the control module, through which every node can communicate directly with any other node within the range of access point and, a node can also communicate indirectly with those nodes outside the range of access point. With indirect communication, other access points are used to relay (forward) data from source to destination. The HMCS considered in this work assumes that the nodes are heterogeneous and they have computing power on par with the other computers. The nodes are close enough to each other so that singlehop communication is possible. Every node submits its profile information such as the type of processor, duration of availability, etc., to the control module (scheduler) when it joins the network.

Similarly a node has to notify to the scheduler whenever it leaves the network. It is assumed that a node performs computation and communication at the same time and once it started executing a task, it should complete task execution and send the output to all the successor tasks scheduled onto other nodes. Any initial data is preloaded before the actual execution of application task begins, and a node consumes no energy if it is idle.

Table 1. Computation Costs of the Task Graph in Figure 1.

Task	P_1	P_2	P_3
v_1	14	16	9
v_2	13	19	18
v_3	11	13	19
v_4	13	8	17
v_5	12	13	10
v_6	13	16	9
v_7	7	15	11
v_8	5	11	14
v_9	18	12	20
v_{10}	21	7	10

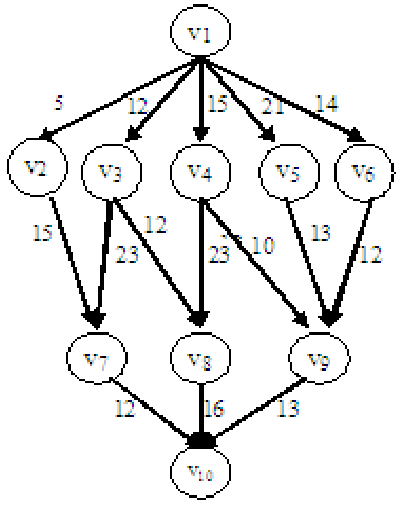


Figure 1. A random task graph

The bandwidth (data transfer rate) of the links between different processors in a heterogeneous system may be different depending on the kind of network. The data transfer rate is represented by a $p \times p$ matrix $R_{p \times p}$. 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 and can be assumed to be independent of the source and destination processors. The communication startup costs of processors are given in a p-dimensional vector *S*.

Let $Data(v_{i,}, v_k)$ be the amount of data to be transferred from task v_i to task v_k (the weight of the edge). The communication cost of the edge $e(v_i, v_k)$, 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

$$c(v_i, v_{,k}) = S_x + Data(v_{i,}, v_k)/ R[p_x, p_y]$$

$$= 0, \text{ otherwise when } x = y. \qquad (1)$$

where S_x is the communication start-up cost of processor p_x.

In order to define the objective of the task scheduling problem for HMCS the following attributes such as Earliest Start Time(EST) and Earliest Finish Time (EFT) are defined first. The *EST* of task v_i on processor p_j is represented as $EST(v_i, p_j)$. Likewise the *EFT* of task v_i on processor p_j is represented as $EFT(v_i, p_j)$. Let $EST(v_i)$ and $EFT(v_i)$ represent the earliest start time upon any processor and the earliest finish time upon any processor, respectively. For the entry task v_{entry} , the $EST(v_{entry}) = 0$, for other tasks in the task graph the *EST* and *EFT* values are computed starting from the entry task to exit task by traversing the task graph from top to bottom. To compute the *EST* of a task v_i all immediate predecessor tasks of v_i must have been scheduled. The calculation of EST and EFT of task is mathematically in Eq. (2).

$$EST(v_i, p_j) = max\{P\_available[v_i, p_j], max(EFT(v_p, p_k)+C(v_p, v_i))\} \text{ , where } v_p \in pred(v_i),$$

$$C(v_p, v_i) = 0 \text{ when } k = j,$$

$$EFT(v_i, p_j) = T(v_i, p_j) + EST(v_i, p_j). \quad (2)$$

P_*Available*[v_i, p_j] is defined as the earliest time that processor p_j will be available to begin executing task v_i. The inner *max* clause in the *EST* equation finds the latest time that a predecessor's data will arrive at processor p_j. If the predecessor finishes earliest on a processor other than p_j, communication cost must also be included in this time. $EST(v_i, p_j)$ is the maximum time at which processor p_j becomes available and the time at which the last message arrives from any of the predecessors of v_i . The attributes used for minimizing the energy consumption in HMCS are defined as follows:

$B(j)$ be the initial battery energy of the node p_j.

$RcompE(p_i)$ be the rate at which the node p_i consumes energy for executing a task, per execution time unit. Then the energy consumed for executing the task v_i on the node p_i, $EcompE(v_i, p_i)$ is computed using Eq. (3)

$$EcompE(v_i, p_i) = T(v_i, p_i) \times RcompE(p_i). \quad (3)$$

$RcommE(p_i)$ be the rate at which the node p_i consumes energy for transmitting one byte of data, per communication time unit.

The *Total Communication Time (TCT)* involved for the processor p_i for receiving input data from predecessor task of v_i and sending output data to the successor task of v_i is computed using Eq. (4)

$$TCT(v_i, p_i) = C(v_i, v_p)+ C(v_i, v_s), \text{ where } v_p \in pred(v_i) \text{ and } v_s \in succ(v_i) \quad (4)$$

The total *Energy Consumed for Communication (ECC)* by a processor p_i for sending and receiving data for task v_i is computed using Eq. (5)

$$ECC(v_i, p_i) = TCT(v_i, v_j) \times RcommE(p_i). \quad (5)$$

The *Total Energy Consumed (TEC)* by a processor for computation and communication for the task v_i is computed using Eq. (6)

$$TEC(v_i, p_i) = Ecomp(v_i, p_i) + ECC(v_i, p_i) \text{ such that } TEC(v_i, p_i) < B(i). \quad (6)$$

Minimization of *schedule length* and *energy consumption* is equally important for achieving better performance in HMCS. Hence, two system attributes λ and γ, such that λ + γ = 1, representing the relative weights of timing requirements (performance) and energy consumption respectively are introduced to fix the objective function. This decides the trade-off between schedule length and energy consumption. The λ and γ values are also used to map time units and energy units to generic cost unit. The problem is mathematically formulated and given in Eq. (7)

$$cost = \lambda \times EFT(v_i, p_j) + \gamma \times TEC(v_i, p_j), \ \forall v_i \in V \text{ and } p_j \in P. \quad (7)$$

The objective of the task scheduling algorithm is to minimize the cost. i.e., the schedule length or the energy consumption or both based on the values of λ and γ. The $EFT(v_i, p_j)$ is computed using the Eq. (2) and $TEC(v_i, p_j)$ is computed using the Eq. (6). When $\lambda = 1$ or $\gamma = 0$, then the cost function is minimization of *EFT* of the exit task, which is nothing but the schedule length of the application.

cost = EFT(v_i, p_j), where v_i is the exit task.

When $\gamma = 1$ or $\lambda = 0$, then the cost function is minimization of total energy consumed by all the processors involved in the program execution and is given in the Eq. (8)

$$cost = \sum_{i=1}^{v} min \{TEC(v_i, p_j)\}, \ \forall p_j \in P. \tag{8}$$

The relative weight to minimize the schedule length or energy consumption can be altered by varying the values of λ from 0 to 1, or γ from 1 to 0.

3. THE PROPOSED HPSM ALGORITHM

The HPSM algorithm has two phases namely, compile-time phase and run-time phase. The compile-time phase of the algorithm has three stages, such as level sorting, task prioritization and processor selection.

In the level sorting phase, the given DAG is traversed in a top-down fashion to sort tasks 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 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 some of the *exit tasks*. For implementation, it is assumed that there is one *entry task* and one *exit task* for a DAG. If there are multiple *entry* or *exit tasks*, the multiple tasks can be connected to a dummy task with zero computation cost and zero communication cost edges.

In the task prioritization phase, priority is computed and assigned to each task of the task graph. The attributes used to assign the priority to a task are the *Down Link Cost (DLC)*, the *Up Link Cost (ULC)*, the *Link Cost (LC)* and the *Average Computation Cost (ACC)* of the task. The DLC of a task is the maximum data (input) received by a task from all its immediate predecessor tasks. The *DLC* for all tasks at *level 0* is 0 and for all other tasks at *level l*, the *DLC* of a task v_j is computed using the Eq. (9)

$$DLC(v_j) = Max\{Data(v_i, v_j)\}, \ \ where \ v_i \in pred(v_j) \tag{9}$$

The *ULC* of a task is the maximum data (output) to be transferred from a task to all its immediate successors. The *ULC* for *exit task* is 0 and for all other tasks at *level l*, it is computed using the Eq. (10)

$$ULC(v_j) = Max\{Data(v_j, v_k)\}, \ \ where \ v_k \in succ(v_j). \tag{10}$$

The *LC* of a task is the sum of *DLC, ULC* and maximum *LC* of its immediate predecessor tasks. The *LC* of a task is calculated using the Eq. (11)

$$\begin{aligned} LC(v_j) &= 0, \textit{for entry task, otherwise} \\ &= max\{LC(v_k)\}+ULC(v_j)+DLC(v_j), \ \ \textit{for all } v_k \in pred(v_j). \end{aligned} \tag{11}$$

The *Average Computation Cost (ACC)* of a task v_i is the average of computation cost on all the *m* available processors and it is computed using the Eq. (12)

$$ACC(v_i) = \sum_{j=1}^{m} T(v_i, p_j) \tag{12}$$

Priority is assigned to all the tasks at each *level i*, based on its LC value. At each level, the task with the highest *LC* value receives the highest priority followed by the task with next highest

LC value and so on in the same level. While assigning priority, if two tasks are having the same *LC* value, then the tie is broken based on the *ACC* value. The task with maximum *ACC* value receives higher priority than the task with the lower *ACC* value.

In the processor selection stage, a processor is selected for executing a task based on the value of λ and γ. When $\lambda = 1$ (or $\gamma = 0$), the algorithm gives maximum weight to minimize the schedule length and when $\lambda = 0$ (or $\gamma= 1$) maximum weight is given to energy minimization. By ranging the values of λ from 0 to 1 (or $\gamma = 1$ down to 0), the weight assigned to minimize the energy consumption or schedule length can be changed. For energy minimization, the algorithm considers only the energy consumed by the processors for computations and communications. The energy consumption of the other components in the node is being assumed to negligible.

During the run-time, if any node leaves the HMCS or the power/energy of any node exhausted, then the HPSM algorithm reschedules the pending tasks to be executed in that node to the other nodes in the HMCS which have similar or higher profile (processor speed, rate of data transfer etc.,) than the leaving node or the energy exhausted node. The pseudo code of the HPSM algorithm is given in Figure 2.

```
// HPSM Algorithm //
Input:
    Number of tasks: $v$ and the computation cost matrix of the DAG: $T (v \times v)$
    Amount of data to be transferred between the tasks: $D (v \times v)$
    Number of processors in the systems: $p$
    Rate of data transfer between the processors: $R (p \times p)$
    Initial energy available in each processor vector, $B(p)$
    Rate of energy consumed by the processors per unit time execution, $Rcomp(p)$
    Rate of energy consumed by the processors to send or receive 1 byte, $Rcomm(p)$
    Values for $\lambda$ or $\gamma$ represents weight value to minimize schedule length and energy
    consumption.
Output:
    Minimum schedule length or energy consumption or both.
Phase I (compile-time)
1.  begin
2.    read the DAG, associated attributes values, and the number of processor $P$;
3.    level sort the given DAG;
4.    for all task $v_k$ in the DAG do
5.     begin
6.        compute $ULC$, $DLC$ and $ACC$ values for the task $v_k$;
7.        compute $LC(v_k) = max\{LC(v_j)\} + ULC(v_k) + DLC(v_k)$, where $v_j \in pred(v_k)$;
8.       insert the task into the priority queue based on the $LC$ value such that the tasks in lower
         level are placed in the priority queue first than the tasks in the higher level and tie if
         any, is broken using the $ACC$ value;
9.     end;
10.   while there are unscheduled tasks in queue
```

```
11.    begin
12.     select the highest priority task $v_j$ from the queue for scheduling;
13.    for each processor $p_k$ in the processor set $P$
14.    begin
15.     compute $EFT(v_j, p_k)$ using isetion-based scheduling policy;
16.     compute $TEC(v_j, p_k)$;
17.     if $(TEC(v_j, p_k) < B(p_k))$ then
18.      begin
19.        compute cost = $\lambda\ (EFT(v_j, p_k)) + \gamma\ (TEC(v_j, p_k))$;
20.        assign the task $v_j$ to the processor $p_k$, which minimizes the cost;
21.        $B(p_k) = B(p_k) - TEC(v_j, p_k)$;
22.        $TECP = TECP + TEC(v_j, p_k)$;
23.      end
24.    else
25.      select the next processor $p_{k+1}$ in the processor set $P$;
26.   end;
27.  end;
28. end.
```

Phase II (Run-time)

```
1.  begin
2.   let b_list be the list of tasks to be processed in the processor $p_l$, which leaves
     the network;
3.   find a suitable processor in the network which have the similar or higher profile
     (characteristics) than the leaving processor $p_l$;
4.   If suitable processor $p_a$ is found then
5.       schedule all the tasks in the b_list onto the processor $p_a$
6.   else
7.       remove the highest priority task from the b_list and schedule it to the existing
         processor which gives minimum EFT for the task;
8.   repeat the steps 2 - 6 until b_list becomes empty;
9.  end.
```

Figure 2 Pseudo code of the HPSM Algorithm

4. Complexity of the HPSM Algorithm

The time complexity of the HPSM is computed for two phases (compile-time and run-time) as follows. In the compile-time phase, level sorting takes $O(e+v)$ time complexity. The prioritization of the tasks (steps 4-9) takes $O(v \log v)$ time complexity. During the *processor selection,* to the *EFT* value for a task v_j, the algorithm searches for a free slot in between any two already scheduled tasks on the same processor p_k, and the search continues until a first free slot that is capable of holding the computation cost of task v_j is found. If no free slot is available then the finish time of the last assigned task in p_k is considered as the start time of the task v_j. The algorithm also computes the *TEC* value for task v_j on every processor p. The time complexity of this phase of the algorithm is $O((e\times p)+(e\times p))$ or $O(e\times p)$, where e is the number of edges. Thus the time complexity of the compile-time phase of the HPSM algorithm is $O((v+e)+v \log v+(e\times p)+(e\times p))$ which is equal to $O(e\times p)$. For a dense graph the number of *edges* is proportional to $O(v^2)$ and hence the time complexity of the *processor selection phase* of the algorithm is $O(pv^2)$.

In the run-time, when a processor p_l leaves the network, at the worst case maximum of v tasks scheduled in p_l at the compile-time are to be rescheduled using insertion based scheduling policy to the remaining $p-1$ processors in the HMCS or to any other suitable processor joins the network after the task allocation is made at the compile-time. Thus the time complexity of the run-time phase of the algorithm is $O(pv^2)$.

5. Illustration of the HPSM Algorithm

The HPSM algorithm is illustrated using the graph given in Figure 1 and its computation costs shown in Table 1. For illustration it is assumed that HMCS consists of 3 processors p_1, p_2 and p_3 and the energy consumed by these processors per unit time execution is 0.6, 0.8 and 0.7 energy units respectively. Further it is assumed that the energy consumed by these processors per unit time communication is 0.2, 0.1 and 0.2 energy units respectively.

5.1.1. Compile-time Phase

The compile-time phase of the HPSM algorithm is illustrated for two scenarios namely, *scenario 1* and *scenario 2*. *Scenario 1* exemplifies minimization of schedule length and *scenario 2* exemplifies minimization of energy consumption. The tradeoff between minimizing the schedule length and the energy consumption is also studied by varying the weight values of λ from 0 to 1, or $\gamma =$ from 1 down to 0.

Scenario 1: The HPSM algorithm minimizes the schedule length when $\lambda = 1$ or $\gamma = 0$. The HPSM algorithm initially level sorts the tasks in the task graph. For example, for the task graph given in Figure 1, after level sorting, level 1 consists of only task v_1, level 2 consists of tasks v_2, v_3, v_4, v_5 and v_6. Level 3 consists of tasks v_7, v_8 and v_9 respectively. Finally level 4 consists of task v_{10}. After the level sorting process, the tasks at each level are scheduled to the suitable processors based on the priority of the task. Priority is assigned to each task in the task graph based on the attributes such as *DLC, ULC, LC* and *ACC*. Table 2 presents the calculated *DLC, ULC, LC, ACC* values and the priority for the task graph shown in Figure 1.

Once the priority for tasks are assigned, a priority queue is constructed in such a way that task at level 1 is placed in the queue first followed by the tasks in the next higher level and so on. The task with highest priority is selected from the priority queue and is assigned to the processor which gives the minimum *EFT* value computed using the Eq. (2). Similarly all the tasks in the queue are scheduled to the suitable processors. Table 3 presents the calculated *EST, EFT* values and the processors selected for executing the tasks in the task graph shown in Figure 1. The shaded value in the cell indicates the earliest finish time of the task on a particular processor.

Table 2. *DLC, ULC, LC* and the other *v*alues computed for the task graph shown in Figure 1.

Level	Task	DLC	ULC	LC	ACC	Priority
1	v_1	0	21	21	13	1
2	v_2	5	15	41	17	5
2	v_3	12	23	56	14	2
2	v_4	15	23	59	13	1
2	v_5	21	13	54	12	3
2	v_6	14	12	47	13	4
3	v_7	23	12	91	11	2
3	v_8	23	16	98	10	1
3	v_9	13	13	80	17	3
4	v_{10}	16	0	114	13	1

The schedule length generated by the HPSM, PETSM and the HEFTM algorithms for the task graph shown in Figure 1 and its computation costs given in Table 1 are 80, 83 and 88 respectively. The schedule length generated by the HPSM is lesser than the schedule length generated by the PETSM and the HEFTM algorithms.

Table 3. The *EST* and *EFT* values Computed for the Task Graph Shown in Figure 1 Using HPSM Algorithm

Task	Processors						Processor
	p_1		p_2		p_3		
	EST	EFT	EST	EFT	EST	EFT	
v_1	0	14	0	16	0	**9**	p_3
v_4	24	37	24	32	9	**26**	p_3
v_3	21	**32**	21	34	26	45	p_1
v_5	32	44	30	43	26	**36**	p_3
v_6	32	45	23	**39**	36	45	p_2
v_2	32	**45**	39	58	36	54	p_1
v_8	49	**54**	49	60	44	58	p_1
v_7	54	**61**	60	75	60	71	p_1
v_9	61	79	49	**61**	51	71	p_2

v_{10}	74	95	73	**80**	74	84	p_2

Scenario 2: In this scenario the energy reduction mechanism of the HPSM algorithm for completing an application is exemplified. The HPSM algorithm minimizes the energy consumption when the maximum weight is given to energy minimization (i.e., $\lambda = 0$ or $\gamma = 1$). When $\lambda = 0$, the HPSM algorithm selects the highest priority task from the ready queue and schedules it to a processor which consumes minimum energy for executing the task. The total energy consumed by a processor for executing a task is equal to the sum of energy consumed by the processor for executing the task and the energy consumed for communication. The energy consumed by a processor for communication is equal to the energy consumed to receive the input from the predecessor tasks and the energy consumed to send the output to the successor tasks. The communication energy is assumed to be zero when both the predecessor and the successor tasks are scheduled in the same processor. The HPSM algorithm computes the energy consumption by a task using the Eq. (6). The computation of energy consumption for the tasks shown in Figure 1 and the corresponding computation costs given in Table 1 is as follows:

For example, for the task v_1 in the Figure 1, the estimated computation time on processors p_1, p_2 and p_3 are 14, 16 and 9 respectively. The energy utilization by these processors for executing the task v_1 is respectively 8.4, 12.8 and 6.3. The energy consumption for sending 67 bytes of data (sum of data to be transferred from task v_1 to v_2, v_3, v_4, v_5 and v_6) by the processors p_1, p_2 and p_3 are 13.4, 6.7 and 13.4 respectively. The total energy utilized by the processors p_1, p_2 and p_3 are 21.8, 19.5 and 19.7 respectively. Since p_2 utilizes lesser energy than the p_1 and p_3, task v_1 is assigned to p_2. The estimated energy consumption may be reduced further if the child's tasks or the immediate successor's tasks of the task v_1 are placed on the same processor. Similarly energy consumption of each processor for executing the tasks given in Figure 1 is computed. The computed energy value and the processor selected are shown in Table 4. The values in bold denote the minimum energy consumption for executing a task by a particular processor.

Table 4. Energy Consumed by the Processors P_1, P_2 and P_3 and the Processor Selected for Executing the Tasks Shown In Figure 1

Task selected for execution based on priority	Energy consumed by p_1	Energy consumed by p_2	Energy consumed by p_3	Processor selected
v_1	21.8	**14.7**	19.7	p_2
v_4	17.4	**8.7**	21.5	p_2
v_3	16	**13.9**	22.7	p_2
v_5	14	**10.4**	13.8	p_2
v_6	13	15.4	**11.5**	p_3
v_2	**8.8**	17.2	16.6	p_1
v_8	**13.2**	13.9	20	p_1
v_7	**11.2**	17	17.7	p_1
v_9	20.4	**10.8**	23.6	p_2
v_{10}	20.8	**7**	15.2	p_2

The total energy consumed by all the three processors for completing the application given in Figure 1 is 110.2, 111.6 and 11.6 energy units for HPSM, PETSM and HEFTM algorithms respectively. From the illustrations it is evident that on an average the HPSM minimizes the schedule length or energy consumption significantly compared to the other two algorithms PETSM and HPSM.

5.1.2 Run-time Phase

The run-time phase of the HPSM algorithm is exemplified as follows: Suppose, during the execution of the program, at time interval 46, if a node p_1 or p_2 or p_3 leaves the network with notification to the scheduler, then the scheduler reschedules the incomplete tasks scheduled earlier in p_1 or p_2 or p_3 to the newly joined processor or among the existing processors in the network. This situation is illustrated for two scenarios i.e., when the processors p_1 and p_3 leave the network. Suppose when the processor p_1 leaves the network at time unit 46, there are two remaining tasks (task v_8 and v_7) to be executed in p_1. The HPSM algorithm reschedules task v_8 and v_7 to the processor p_3, since p_3 gives minimum EFT for these tasks by meeting the precedence constraints. As expected, the rescheduling of tasks will increase the schedule length generated by the HPSM algorithm. The PETSM reschedule task v_8 onto p_2 and the HEFTM algorithm reschedules the task v_9 onto the processor p_2. The final schedule length generated by HPSM, PETSM and HEFTM are 93, 83 and 88 respectively. Similarly, when the processor p_2 leaves the network at time unit 48, the HPSM algorithm reschedules the tasks 9 and 10 onto processor p_1. The PETSM reschedules the tasks 9 and 10 in p_2 onto p_1 and the HEFTM algorithm reschedules the task 10 onto p_3. The schedule lengths generated by the HPSM, PETSM and HEFTM, after task rescheduling, are 100, 99 and 88 respectively. The rescheduling of tasks at run-time shows that the HEFTM algorithm is better than the HPSM and PETSM algorithms because it has enough holes (free slots) in processor p_2 to accommodate other tasks.

6. EXPERIMENTAL RESULTS AND DISCUSSION

Experiments have been conducted by simulation to evaluate the performance of the HPSM algorithm in terms of schedule length and the energy consumption. A large number of random task graphs have been generated and used for the simulation experiments. The parameter used for the generation of random task graph is discussed below.

6.1. Generation of Random Task Graphs

The simulations are performed by creating a set of random DAGs which are input to the proposed as well as the existing task scheduling algorithms. The method used to generate random DAGs is similar to that presented in [7]. The following input parameters are used to create the DAG.

(1) Number of nodes (tasks) in the graph, v.

(2) Shape parameter of the graph, α. The height of the DAG is randomly generated from a uniform distribution with mean value equal to $\sqrt{v/\alpha}$. The width for each level in the DAG is randomly selected from a uniform distribution with mean equal to $\alpha \times \sqrt{v}$. If $\alpha = 1.0$, then the DAG will be balanced. A dense graph (shorter graph with high parallelism) and a longer graph (low parallelism) can be generated by selecting $\alpha \gg 1.0$ and $\alpha \ll 1.0$ respectively.

(3) Out degree of a node, *out_degree*: Each node's out degree is randomly generated from a uniform distribution with mean value equal to *out_degree*.

(4) *Communication to Computation Ratio* (*CCR)*: If the DAG has a low CCR, it can be considered as a computation-intensive application and if CCR is high, it is a communication-intensive application.

(5) *Average Computation Cost (ACC)* in the graph: Computation costs are generated randomly from a uniform distribution with mean value equal to ACC.

(6) Range percentage of computation costs on processors, β: A high β value causes a wide variance between a task's computations across the processors. A very low β value causes a task's computation time on all processors to be almost equal. The average computation cost T_i of task v_i in the task graph is selected randomly from a uniform distribution with range [0, $2*T_{dag}$], where T_{dag} is the average computation cost of the given graph, which is set randomly in the algorithm. The computation cost of v_i on any processor p_j will then be randomly selected from the range $[T\times(1-\beta/2)]$ to $[T\times(1 + \beta/2)]$.

The input parameters described above were varied with the following values:

$v = \{20, 30, 40, 50, 60, 70, 80, 90, 100\}$,

$CCR = \{0.1, 0.5, 1.0, 5.0, 10.0\}$,

$\alpha = \{0.5, 1.0, 2.0\}$,

$out\_degree = \{1, 2, 3, 4, 5\}$,

$\beta = \{0.1, 0.25, 0.5, 0.75, 1.0\}$,

$m = \{0.25, 0.5, 1.0\}$.

For simulation of experimental results the following systems attributes were assumed.

Input parameters	Values
Number of nodes in the HMCS	Ranges from 3 to 12
Energy consumption rate of the nodes for computations	Randomly generated between 0.1 to 2.0

	Joule
Energy consumption rate of the nodes for communication	Randomly generated between 0.1 to 2.0 Joule
Initial available energy	2000 to 10000 Joule

The performance of the HPSM algorithm is compared with two other algorithms namely, PETSM and HEFTM algorithms. Two types of experiments have been conducted for the evaluation of algorithms. In the first experiment, the performances of the algorithms are evaluated based on the schedule length (by setting λ = 1 or γ = 0) generated by each of the algorithm for the randomly generated task graphs. In the second experiment, the performances of the algorithms are evaluated based on the average energy consumption of the nodes (by setting λ = 0 or γ = 1) to complete the program for the randomly generated task graphs.

In the first experiment, random task graphs are generated with the number of tasks in the task graph ranges from 10 to 50 in step size of 10, using the graph characteristics values given in section 6.1 and these task graphs are scheduled onto the HMCS consisting of four nodes. The average schedule length generated by each of the algorithm in this experiment is plotted and is shown in Figure 2**.** Each data point in the reported graph is the average of the data obtained in 60 experiments. The experimental results show that HPSM generated lesser schedule length than the PETSM and HEFTM algorithms. The average schedule length based ranking of the algorithms is HPSM, PETSM and HEFTM.

Further, for the experiments conducted above, the average energy consumed by all the processors in the HMCS to complete the program execution have been determined and plotted and is shown in Figure 3. The result shows that the HPSM algorithm consumes lesser energy than the HEFTM and the PETSM algorithms. The average energy consumption based ranking of the algorithm is HPSM, PETSM and HEFTM.

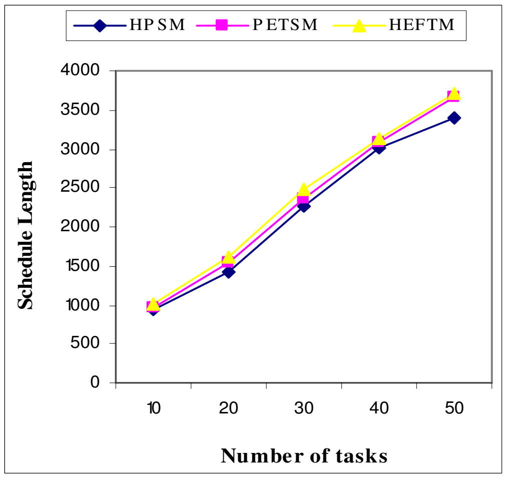


Figure 2. Average SL generated by HPSM, PETS and HEFTM algorithms for the random task graphs when (λ = 1 or γ = 0)

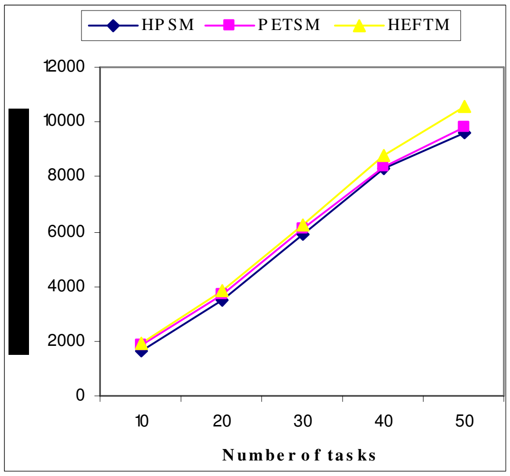


Figure 3. Average energy consumption four processors by HPSM, PETSM and HEFTM algorithms for the random task graphs when (λ = 1 or γ = 0)

Another set of experiments are conducted to evaluate the energy consumption. For this experiment, random task graphs are generated with number of tasks in the task graph tasks ranges from 10 to 50 in step size of 10, using the graph characteristics values given in section 6.1 and these graphs scheduled onto the HMCS consisting of four nodes with an objective to minimize the energy consumption (by setting $\lambda = 0$ or $\gamma = 1$). The total energy consumption of all the four nodes to complete the program is computed and plotted in Figure 4**.** Each data point in the reported graph is the average of the data obtained in 60 experiments. On an average the energy consumed by all the nodes to complete the execution of an application is less when the HPSM algorithm is used instead of PETSM and HEFTM algorithms. The average energy consumption based ranking of the algorithms is HPSM, PETSM and HEFTM.

Again, for the above experiments, the average schedule length generated by each of the algorithm has been determined and plotted and is shown in Figure 5. The result shows that the HPSM algorithm generates a more minimum schedule length than HEFTM and PETS algorithms. The ranking of the algorithm based on the average schedule length for this experiment is HPSM, PETSM and HEFTM.

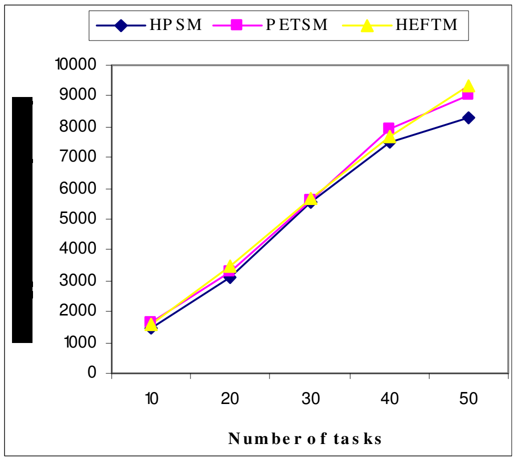


Figure 4 Average energy consumption of four processors by HPSM, PETSM and HEFTM algorithms for random task graphs

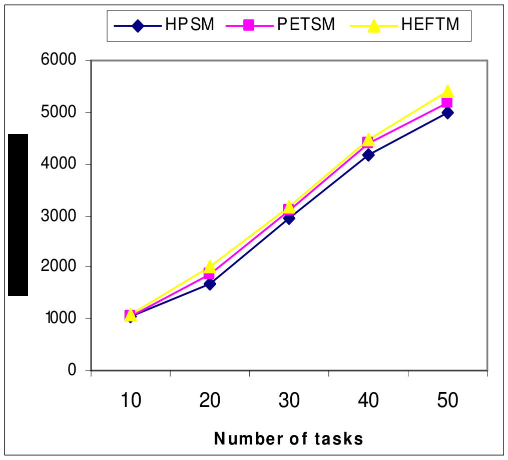
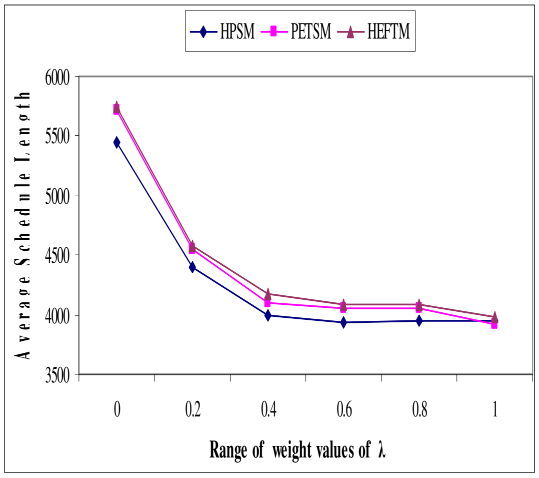


Figure 5 Average SL generated by HPSM PETSM and HEFTM algorithms for the random task graphs when (λ = 0 or γ = 1)

The first and the second experiments show that, when the minimization of the schedule length is the primary objective of the algorithms, then the energy consumed by the node increases correspondingly. On an average the schedule length generated by the HPSM, PETS and the HEFTM algorithm are 2208, 2328 and 2390 by consuming 5794, 5974 and 6258 units of energy respectively. When minimization of the energy consumption is the primary objective of the algorithms, then the schedule length generated by each of the algorithms increases correspondingly. On an average the total energy consumed by all the four processors by the HPSM, PETS and the HEFTM algorithms are 5203, 5433 and 5569 energy units with the schedule length of 3023, 3118 and 3236 respectively.

In another experiment, the performances of the algorithms are studied with respect to the schedule length and energy consumption by varying the λ value from 0 to 1 in step size of 0.2. For this experiment, random task graphs with 50 tasks have been generated and scheduled onto a HMCS consisting of 4 nodes. The other graph characteristics are as given in section 6.1. The schedule length generated by each of the algorithm is plotted and shown in Figure 6. Each data point in the reported graph is the average of the data obtained in 60 experiments. For the same experiment, the energy consumed by the processors is determined and plotted in Figure 7. This experiment confirms that when the schedule length decreases energy consumption increases and vice versa.



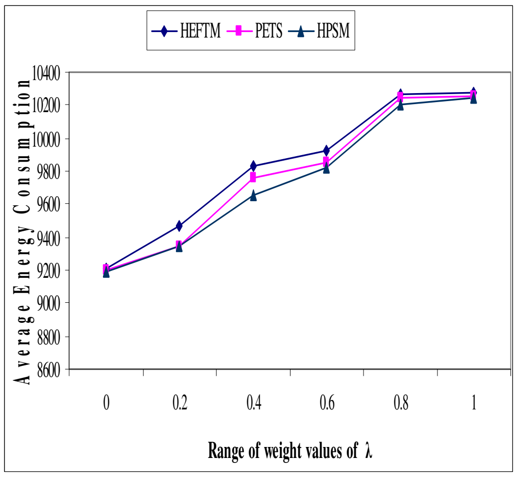


Figure 6 Average SL generated by HPSM, PETSM and HEFTM algorithms for different weight values of λ

Figure 7 Average energy consumption of the processors for different weight of λ by HPSM, PETSM and HEFTM algorithms

Finally, the performances of the algorithms are studied with respect to the node mobility. For the experimental purpose, it is assumed that nodes in HMCS have limited mobility and if any node wishes to leave the network, it has to send notification to the scheduler and can leave only after getting the acknowledgement from the scheduler. For this experiment, random task graphs with 100 tasks have been generated using the graph characteristic given in section 6.1 and scheduled onto the twelve nodes. Whenever a node leaves the network or when the node becomes unavailable due to energy exhaustion, the incomplete tasks in the leaving node or the energy exhausted node are rescheduled to the other node which has similar or higher profile than the leaving node or the energy exhausted node. The performance of the algorithms in terms of schedule length when none of the nodes join the network is studied and plotted in Figure 8.

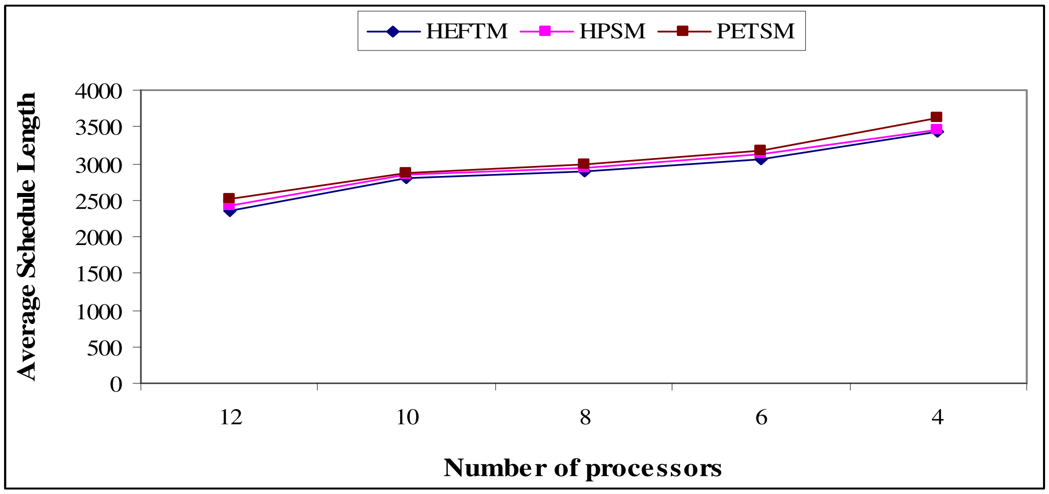


Figure 8 Average SL generated by HPSM, PETSM and HEFTM algorithms for different number of processors in HMCS

The experiment results show that the schedule length is increased whenever the nodes leave the network and the tasks in the leaving nodes are rescheduled to other nodes. Each data point in the figure is the average of 50 experiments. The rescheduling of tasks, at run-time, shows that the HEFTM algorithm is marginally better than the HSPM and PETSM algorithms.

7. CONCLUSION

The scheduling of DAG-structured application onto the HMCS is explored in this paper with an objective to minimize the schedule length or energy consumption or both, based on the weights given for minimizing the schedule length or energy. As the execution of an application on HMCS demands both the compile-time and runt-time support, a new two-phase task scheduling algorithm namely, HPSM has been proposed and developed, which considers either the minimization of the schedule length or the energy utilization or both, at the compile-time based on the needs and rescheduling of tasks whenever a node leaves the HMCS or the energy of the node exhausted at the run-time. The performance of the HPSM algorithm is compared with the PETSM and HEFTM, the modified versions of the PETS and HEFT algorithms respectively. Simulation experiments have been conducted to study the effectiveness of the algorithms in terms of the schedule length and energy consumption using a large set of randomly generated task graphs. The experimental results show that the HPSM algorithm significantly performs better than the PETSM and the HEFTM algorithms.

ACKNOWLEDGEMENTS

The authors gratefully acknowledge Sandeep and Sriram the past students of the Department of Computer Science and Engineering for the helps rendered in implementing the existing and the proposed algorithms and also for conducting large number of experiments.

REFERENCES

[1] Lee, S.hyun. & Kim Mi Na, (2008) “This is my paper”, ABC *Transactions on ECE*, Vol. 10, No. 5, pp120-122.

[2] Gizem, Aksahya & Ayese, Ozcan (2009) *Coomunications & Networks*, Network Books, ABC Publishers.

[1] Talukder A. K., and R. R. Yavagal, *Mobile Computing: Technology, applications and service creation*, Tata McGraw Hill, 2006.

[2] Zeng H., C. Ellis, A. Leveck and A. Vahdat. Ecosystem: Managing energy as a first class operating system resource. *Proceedings of the Architectural Support for Programming Languages and Operating Systems (ASPLOS),* 2002, pp. 123-132.

[3] Flinn J., D. Narayanan and M. Satyanarayanan. Self-Tuned remote execution for pervasive computing,” *Proceedings of the 8th workshop on Hot topics in Operating Systems (HotOS-VIII)* 2001.

[4] Weiser M., B. Welch, A. Demers and S. Shenker. Scheduling for reduced CPU energy,” *Proceedings of the 1st symposium on operating systems design and implementation,* 1994, pp. 13-23.

[5] Hong I., D. Kirovski, G. Qu, M. Potkonjak and M.B. Srivastava. Power optimization of variable-voltage core-based systems. *IEEE Transactions on Computer-Aided Design Integrated Circuits Systems* 1999, pp. 1702–1714.

[6] Mishra R., N. Rastogi, Z. Dakai, D. Mosse and R. Melhem. Energy aware scheduling for distributed real-time systems. *Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS'03)* 2003, pp. 22–26.

[7] Topcuoglu H., S. Hariri and M. Y. Wu. Performance effective and low-complexity task scheduling for heterogeneous computing. *IEEE Transactions on Parallel and Distributed Systems,* 2002*,* 13(3): 274.

[8] Ilavarasan E. and P. Thambidurai. Low complexity performance effective task scheduling algorithm for heterogeneous computing environments. *Journal of Computer Science*, Science Publications, 2007, 3(2): 94-103.

[9] Lu X., H. Hassanein and S. Akl. Energy aware dynamic task allocation in mobile ad hoc networks. *Proceedings of the International Conference on Wireless Networks, Communications and Mobile Computing (WirelessCom'05),* 2005, pp. 534-539.

[10] Zong Z., A. Manzanares, B. Stinar and X. Qin. Energy-Aware Duplication Strategies for Scheduling Precedence-Constrained Parallel Tasks on Clusters. *Proceedings of the IEEE International Conference on Cluster Computing,* 2006*,* pp. 1-8.

[11] Shivle S., R. Castain, H. J. Siegel, A. A. Maciejewski, T. Banka, K. Chindam, S. Dussinger, P. Pichumani, P. Satyasekaan, W. Saylor, D. Sendek, J. Sousa, J. Sridharan, P. Sugavanam and J. Velazco. Static mapping of subtasks in a heterogeneous ad hoc grid environment. *Proceedings of the Parallel and Distributed Processing Symposium*, 2004, pp. 110-115.

[12] J. Hu and R. Marculescu, "Energy-Aware Communication and Task Scheduling for Network-on-Chip Architectures under Real-Time Constraints," in Proceedings of the Design, Automation and Test in Europe Conference and Exhibition, Feb. 2004.

Authors

E. Ilavarasan is currently working as Assistant Professor in the Department of Computer Science and Engineering at Pondicherry Engineering College, Puducherry, India. He has published more than twenty research papers in the International Journals and Conferences. His area of specialization includes Parallel and Distributed Systems, Design of Operating Systems and Web Technology.

R. Manoharan is currently working as Assistant Professor in the Department of Computer Science and Engineering at Pondicherry Engineering College, Puducherry, India. He has published more than fifteen research papers in the International Journals and Conferences. His area of specialization includes High Speed Networks and Mobile Computing