Power Efficient Scheduling Scheme based on PSO for Real Time Systems

Medhat H A Awadalla
Electrical and Computer Engineering Department, SQU, Oman
Communication and Computers Department, Helwan University, Egypt

ABSTRACT
Power efficient design of real-time embedded systems based on multi-processors becomes more important as system functionality is increasingly realized through heuristic approaches. This paper targets energy-efficient scheduling of tasks over multiple processors, where tasks share a common deadline. It addresses the problem of energy-aware static partitioning of periodic real time tasks on heterogeneous multiprocessor platforms. A modified Particle Swarm Optimization variant based on priority assignment and min-min algorithms for task partitioning is proposed. The proposed approach aims to minimize the overall energy consumption, meanwhile avoid deadline violations. An energy-aware cost function is proposed to be considered in the proposed approach. Extensive simulated experiments and comparisons with related approaches are conducted in order to validate the effectiveness of the proposed technique. The achieved results demonstrate that the proposed partitioning scheme significantly outperforms in terms of the number of executed iterations to accomplish a specific task in addition to the energy savings.

Keywords
Task Partitioning, Task Assignment, Heterogeneous Multiprocessors, Particle Swarm Optimization, Min-min, Priority assignment algorithm

1. INTRODUCTION
The energy usage of computer systems is becoming an important consideration, especially for battery operated systems. Nowadays, embedded systems are involved in most details of our life such as smart phones, pocket PCs, Personal Digital Assistants (PDAs), multimedia devices, ... etc. As the applications on these devices are being complicated, there is a need to increase the performance while keeping the energy consumption of these devices in accepted levels especially for the portable battery-powered ones. So, minimizing energy consumption to prolong the battery life while achieving higher performance is a critical issue in the design of portable embedded systems. Various methods for reducing energy consumption have been investigated, both at the circuit level and at the operating systems level. As the processor is one of the most important power consumers in any computing system, today's chip multiprocessor (CMP) or multiprocessor system on chip (MPSoC) platforms can deliver a higher performance at the cost of lower power consumption than uniprocessor systems. Embedded systems today are often implemented upon platforms comprised of different kinds of processing units, such as CPU's, DSP chips, graphics co-processors, math co-processors, etc., with each kind of processing unit specialized to perform a different function most efficiently. Such platforms are commonly referred to as heterogeneous platforms [1-3]. TI's OMAP™ mobile processors are good example of these heterogeneous platforms [4]. The multiprocessor scheduling of recurrent real-time tasks can be generally carried out under the partitioned scheme or under the global scheme. In the partitioned scheme, the tasks are statically partitioned among the processors and all instances (jobs) of a task are executed on the same processor and no job is permitted to migrate among processors. In the global scheme, a task can migrate from one processor to another during the execution of different jobs. Furthermore, an individual job of a task that is preempted from some processor, may resume execution in a different processor. Nevertheless, in both schemes, parallelism is prohibited, i.e., no job of any task can be executed at the same time on more than one processor. This paper considers the partitioned scheduling scheme. The main advantage of the partitioned scheduling is that after partitioning the tasks among processors, the multiprocessor scheduling problem is reduced to a set of traditional uniprocessor ones.

The problem of partitioning tasks among processors [5], sometimes referred to as Task Assignment Problem (TAP), is an intractable NP-Hard problem even if the processors are homogeneous [6]. So, approximation algorithms and heuristic techniques are used to solve this problem. This paper proposes a modified Particle Swarm Optimization (PSO) variant based on Min-min technique and priority assignment algorithm for energy-aware task partitioning on heterogeneous multiprocessor platforms. The rest of this paper is organized as follows: Section 2 reviews existing research on task partitioning upon heterogeneous platforms and related areas. Section 3 defines the problem and describes task, processor, and power models used in this paper. Section 4 presents PSO, Min-min and priority assignment techniques for task partitioning and introduces our proposed approach. Section 5 presents simulation results for the proposed algorithm and discusses these results. Section 6 summarizes our conclusions.

2. RELATED WORK
The use of heterogeneous multi-core architectures has increased because of their potential energy efficiency compared to the homogeneous multi-core architectures. The shift from homogeneous multi-core to heterogeneous multi-core architectures creates many challenges for scheduling applications on the heterogeneous multicore system.

The authors [7] have studied the energy-efficient scheduling on Intel's QuickIA heterogeneous prototype platform [8]. A regression model is developed to estimate the energy consumption on the real heterogeneous multi-core platform. The author of [9] proved that task partitioning among heterogeneous multiprocessors is intractable (strongly NP hard), represented the problem as an equivalent Integer Linear Programming (ILP) problem, and designed a 2-step approximation algorithm for solving this problem. The idea of LP relaxations to ILP problems is used in the first step to map most tasks, while in the second step the algorithm maps the remaining tasks using exhaustive enumeration. This two-step algorithm takes time polynomial in the number of tasks, and exponential in the number of processors. They used tree

partitioning in the second step instead of exhaustive enumeration to make the algorithm takes time polynomial in the number of tasks, and polynomial in the number of processors. In [9] authors have compared 11 heuristics for mapping a set of independent tasks onto heterogeneous distributed computing systems. The best one that has minimum makespan, that is defined as the maximum completion time for the whole processors, was the Genetic Algorithm (GA) followed by Min-min algorithm. In [10], Chen and Cheng (2005) applied the Ant Colony Optimization (ACO) algorithm. They proved that ACO outperforms both GA and LP-based approaches in terms of obtaining feasible solutions as well as processing time. Abdelhalim [5] presented a modified algorithm based on the Particle Swarm Optimization (PSO) for solving this problem and showed that his approach outperforms the major existing methods such as GA and ACO methods. Then, his PSO approach is developed to can further optimize the solution to reduce the energy consumption by minimizing average utilization of processors (without using any energy or power model). Finally, a tradeoff between minimizing the design makespan as well as energy consumption is obtained. In [11] Visalakshi and Sivanandam, presented a hybrid PSO method for solving the task assignment problem. Their algorithm has been developed to dynamically schedule heterogeneous tasks onto heterogeneous processors in a distributed setup. It considers load balancing and handles independent non-preemptive tasks. The hybrid PSO yields a better result than the normal PSO when applied to the task assignment problem. The results are also compared with GA. The results infer that the PSO performs better than the GA. In [12], Omidi and Rahmani used PSO for task scheduling in multiprocessor systems as an important step for efficient utilization of resources. They considered independent tasks on homogeneous multiprocessor systems. Apart from all these efforts, this paper integrates the PSO approach with a polynomial-time partitioning techniques; Min-min and priority assignment. The proposed approach takes into account energy efficiency during task partitioning among heterogeneous cores in MPSoCs.

3. SYSTEM MODEL

This paper considers the problem of power-aware task partitioning on heterogeneous multiprocessor platforms. So, models of task, processor, and power are presented. The same model in [13-14] has been used again for the sake of qualitative comparison.

3.1 Task Model

A periodic real-time task τ_i generates an infinite sequence of task instances (jobs). Each job executes for C time units at most, be generated every T time units, and has a relative deadline D time units after its arrival.

This paper considers a periodic task set $\{\tau_1,\tau_2,\ldots,\tau_n\}$ of n independent real-time tasks. A task is τ_i represented as 3-tuple (C_{ij}, D_i, T_i) where C_{ij} is the Worst-Case Execution Time (WCET) of task τ_i on processor j, D is the relative deadline, and T is the period. Implicit deadlines are considered in this paper, i.e., the relative deadline is assumed to be the same as the period. Each task τ_i has a utilization $U_{ij} = C_{ij} / T_{ij}$ on processor j. An n x m utilization matrix [15] can be defined where each row represents a task and each column represents a processor.

3.2 Processor Model

A heterogeneous multiprocessor platform with m preemptive processors based on CMOS technology is defined as $\{P_1,P_2,\ldots,P_m\}$.

This paper considers Dynamic Voltage/Frequency Scaling (DVFS) processors that supports variable frequency (speed) and voltage levels continuously, i.e., DVFS processors can operate at any speed/voltage in its range (ideal). Of course, practical DVFS processors supports discrete speed/voltage levels (non-ideal). So, the desired speed/voltage of the ideal DVFS processor is rounded to the nearest higher speed/voltage level of the practical DVFS processor supports. The time (energy) required to change the processor speed is very small compared to that required to complete a task. It is assumed that the speed/voltage change overhead, similar to the context switch overhead, is incorporated in the task execution time. In this work, it is assumed that the processor's maximum speed (frequency) is 1 and all other speeds are normalized with respect to the maximum speed. When MPSoCs platforms are considered, there are per-core and full-chip DVFS techniques [16-17]. In the per-core DVFS, each core operates at individual frequency/voltage, and has no operating frequency constraint. On the other hand, the practical full-chip DVFS designs restrict that all the cores in one chip operate at the same clock frequency/voltage. For each processor, the tasks are scheduled according to Earliest Deadline First (EDF) scheduling algorithm. So, a processor utilization U_j which is the sum of the utilizations of tasks assigned to this processor cannot exceed 1, i.e., $U_j=\sum u_{ij} \leq 1$.

3.3 Power Model

The power consumption in CMOS circuits has two main components: dynamic and static power. The dynamic power consumption which arises due to switching activity can be represented as in [10]:

$$P_{dynamic}=C_{eff} * V_{dd}^{\ 2} * f \quad (1)$$

Where Ceff is the effective switching capacitance, Vdd is the supply voltage, and f is the processor clock frequency (speed) which can be expressed in terms of a constant k, supply voltage Vdd and threshold voltage Vth as follows:

$$f=k * (V_{dd}-V_{th})^{\ 2} / V_{dd} \quad (2)$$

The static power consumption is primarily occurred due to leakage currents (I_{leak}) [2], and the static (leakage) power (P_{leak}) can be expressed as:

$$P_{leak}=I_{leak} * V_{dd} \quad (3)$$

When the processor is idle, a major portion of the power consumption comes from the leakage. Currently, leakage power is rapidly becoming the dominant source of power consumption in circuits and persists whether a computer is active or idle [10]. So, lowering supply voltage is one of the most effective ways to reduce both dynamic and leakage power consumption. As a result, it reduces energy consumption where the energy consumption is the power dissipated over time. For simplicity reasons, Eq. (1) is reduced to a simplified power model $P = f^{\ 3}$ using normalized values where f is the processor speed (frequency). Then, a simplified energy model $E= f^{\ 2}$ (using normalized values) can be used.

4. THE PROPOSED APPROACH

Before introducing our proposed approach in this paper, a background on PSO and priority assignment and min-min techniques will be presented.

4.1 PSO

In [18], Higashino et al. developed the PSO algorithm simulating the behavior of swarms in the nature, such as birds, fish, etc. In PSO, the potential solutions, called particles, fly through the problem space by following the current optimum

particles. PSO has been successfully applied in many scientific areas and there are many variants of the algorithm. A survey of PSO methods and applications could be found in [19]. At the beginning, a set (swarm) of random solutions (particles) is used to initialize the PSO algorithm that starts iterations looking for optimal solution. During every iteration, each particle is updated by two best values. The first one is the personal best pbest that the particle has achieved so far. The second is the global best gbest obtained by any particle in the swarm. After finding the two best values, the particle updates its velocity and position according to equations (4) and (5) respectively. The typical procedure of PSO is shown in figure 1.

initialize the population randomly.

DO

{

For each particle.

{

Calculate fitness value

If the fitness value is better than the best fitness value (pbest) in history then set current value as the new pbest.

}

Choose the particle with the best fitness value of all particles as the gbest.

For each particle.

{

Calculate new velocity:

$V_{new}= W. V_{o1d} +C_1. R_1. (p_{best} - X) + C_2. R_2.(g_{best}-X)$ (4)

(Where W is inertia constant, R_1 and R_2 are random values. C_1 and C_2 are constant values

and X is particle position.)

Update particle position:

$X_{new}= X_{old} + V_{new}$ (5)

}

}

Until termination criterion is met.

Fig. 1 The typical procedure of PSO.

The random numbers R_1 and R_2 are generated uniformly between 0 and 1 and the constants C_1 (self-knowledge factor) and C_2 (social-knowledge factor) are usually in the range from 1.5 to 2.5. Finally, the inertia factor W can be fixed or varied with a decreasing value as the algorithm proceeds [12], or it may be restarted as in [5]. PSO has been applied to solve the problem of task partitioning for homogeneous multiprocessor as in [12] and also for heterogeneous multiprocessors [5,11]. Considering a system consisting of m processors and n tasks. A possible solution (particle) is a vector of n elements, where each element is associated to a given task. Each element takes an integer value i where $1 \le i \le m$ and represents the processor that the task is assigned to. Thus, the search space size is m^n. There are k particles in the swarm that form swarm (population) size; these particles are initialized randomly.

4.2 Min-Min Algorithm

The Min-min algorithm is designed [9] for mapping tasks in heterogeneous computing systems. It first finds the minimum completion time of all unmapped tasks, where the completion time of a task on a machine equals task's execution time on that machine plus execution times of all tasks mapped to that machine. Next, the task which has minimum completion time is selected, similar technique called Max-min selects the task with maximum completion time, and mapped to the machine. Finally, the newly mapped task is removed and the process repeats until all tasks are mapped. To handle real-time tasks on multiprocessor system, task utilization is considered instead of execution time and completion utilization is used. Of course, tasks that make the processor's utilization exceeds 1 are unaccepted. If there is no accepted alternative, then the task set is unfeasible.

4.3 Proposed Priority Assignment Algorithm

To optimize Min-min algorithm, Priorities have been determined from Directed Acyclic Graph, DAG, and then assigned to the tasks in such way that the important task will be assigned to the processor that eventually leads to a better scheduling. Priority assignment algorithm flowchart is illustrated in Figure 2. In this paper, real-time tasks are considered.

Each task is characterized by the following parameters:

t_s (T) : is the starting time of task T of G.

t_s (T, P) : is the starting time of task T on processor P.

t_f (T) : is the finishing time of task T.

w (T) : is the processing time of task T.

The algorithm starts by assigning levels for the tasks (the root task has level 0). The level of a task graph is defined as:

This Level function indirectly conveys precedence relations between the tasks. If the task T_i is an ancestor of task T_j, then Level (T_i) < Level (T_j). If there is no path between the two tasks, then there is no precedence relation between them and the order of their execution can be arbitrary.

Secondly, the sequence of tasks' execution in each level is determined. For the root level (T_1, T_2) shown in Figure 3, if there is only one parent task, then it comes first.

If there is more than one parent task, the number of children for each parent in the next level is calculated and their parent has got a priority according to that number in a descending order. The parent with the highest number of children comes first (T_1 executed before T_2). If two or more parents have the same number of children (T_3, T_4 and T_5) then the parent that has a common child is to be executed first (T_4 and T_5 will be executed before T_3). When two parents have the same common child, they will be listed in an arbitrary order (T_4 and T_5).

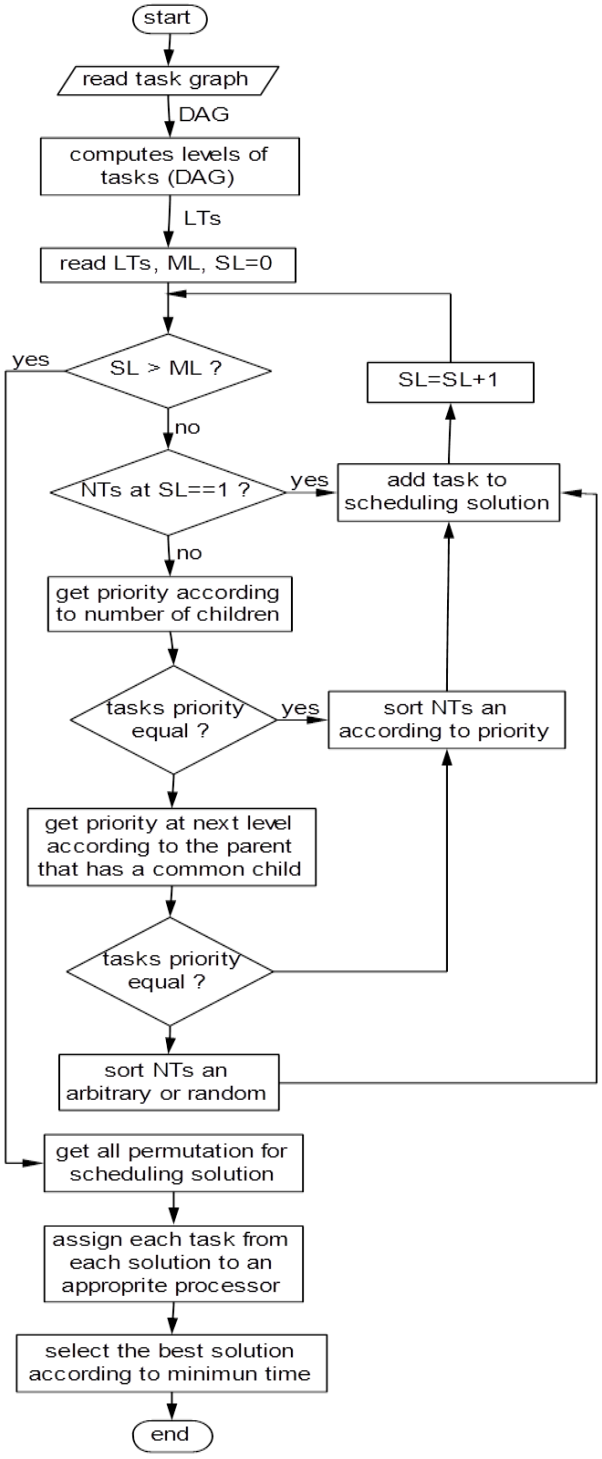

Fig. 2: Priority assignment Flowchart

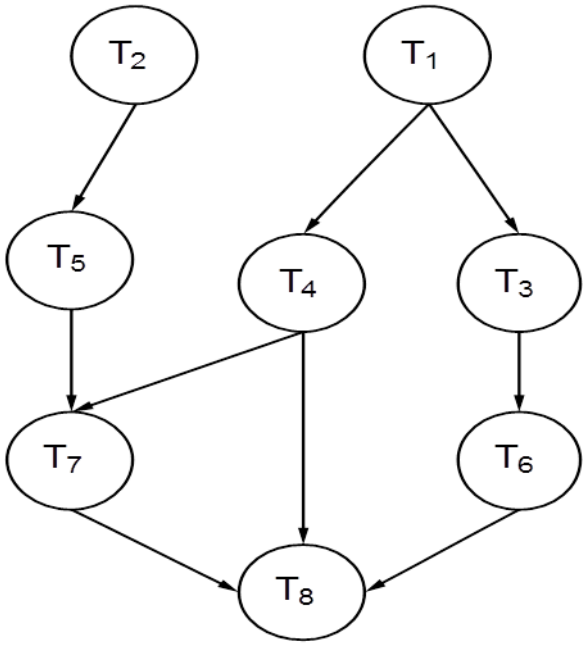

Fig. 3: Example of DAG with 8 tasks

For example, in the random generated task graph shown in figure 3, the level for each task can be determined.

T_1= Level 0

T_2, T_3 and T_4= level of (T_1) +1 = Level 1

T_6= level of (T_2) +1 = Level 2

T_7= Max {level of (T_1), T_2 and T_3) +1 = Level 2 Where T_1 (Level 0) < T_1 (Level 1)

T_8= Max {level of (T_2), T_3 and T_5) +1 = Level 2

Where T_2 (Level 1) = T_3 (Level 1) = T_5 (Level 1)

T_9= Max {level of (T_6), T_7 and T_8) +1 = Level 3

Where T_6 (Level 2) = T_7 (Level 2) = T_8 (Level 2)

An illustrative example

DAG shown in figure 4. Based on the priority assignment algorithm, Table 1 shows All possibilities of tasks orders based on their priorities, where RA is random order. Min-min algorithm will be invoked to determine which of these possibilities will have the minimum execution time and feeds it to PSO approach to start its optimization capability to enhance the system performance in terms of minimizing the total execution time and power saving.

Table 1 All task order possibilities based on their priorities

Possible task order				RA.	RA.	RA.	RA.	RA.		ET.
1	T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8	T_9	26
2	T_1	T_2	T_3	T_4	T_5	T_6	T_8	T_7	T_9	26
3	T_1	T_2	T_3	T_4	T_5	T_7	T_6	T_8	T_9	26
4	T_1	T_2	T_3	T_4	T_5	T_7	T_8	T_6	T_9	26

5	T_1	T_2	T_3	T_4	T_5	T_8	T_6	T_7	T_9	26
6	T_1	T_2	T_3	T_4	T_5	T_8	T_7	T_6	T_9	26
7	T_1	T_2	T_3	T_5	T_4	T_6	T_7	T_8	T_9	23
8	T_1	T_2	T_3	T_5	T_4	T_6	T_8	T_7	T_9	23
9	T_1	T_2	T_3	T_5	T_4	T_7	T_6	T_8	T_9	25
10	T_1	T_2	T_3	T_5	T_4	T_7	T_8	T_6	T_9	25
11	T_1	T_2	T_3	T_5	T_4	T_8	T_6	T_7	T_9	25
12	T_1	T_2	T_3	T_5	T_4	T_8	T_7	T_6	T_9	25

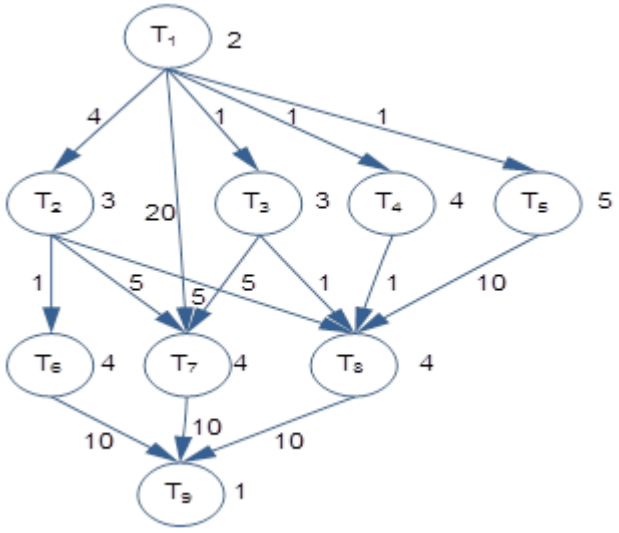

Figure 4: Example DAG with nine tasks

4.4 The Proposed Modified PSO Approach

The modified PSO approach proposed in this paper, simply modifies the initialization step in the PSO procedure by assigning priorities for each task and then incorporating a Min-min solution (particle) in the randomly generated population. This approach gives the PSO algorithm a push to start from a good solution and then the PSO goes on trying to optimize the solution resulting in the Min-min solution in the worst case, PSO in the illustrated example starts with some tasks such as order 6 or 7 as shown in table 1 because they have the minimum execution time. A cost function favoring makespan (maximum processor accumulative utilization)

minimization is proposed. Then, a penalty is added to the infeasible solutions that exceed the processing capacity of any processor. In other words, the cost is represented as follows (Chen and L. Thiele, 2009):

$$\text{Cost} = \text{Max}(U_j) + \text{Penalty} \quad \text{for } j = 1,2, \ldots, m \qquad (7)$$

$$\text{Penalty} = \text{Sum}(U_j > 1) \quad \text{for } j = 1,2, \ldots, m \qquad (8)$$

Next, the cost function is developed to incorporate energy where the proposed PSO approach tries to find energy efficient solutions. Aydin and Yang (2003) considered energy aware task partitioning for homogeneous multiprocessors and introduced some helpful proofed theorems and propositions. Some of them are presented here.

Proposition 1: For a single processor system and a set of periodic real-time tasks with total utilization ≤ 1. The optimal speed to minimize the total energy consumption while meeting all the deadlines is constant and equal to total utilization (Aydin, and Q. Yang, (2003)).

Proposition 2: A task assignment that evenly divides the total load among all the processors, if it exists, will minimize the total energy consumption for any number of tasks. So, minimizing the makespan will minimize energy consumption especially when full-chip DVFS multiprocessor platforms are considered, the makespan cost function, Eq. (9), will be used as all processors on the chip have to operate at the same frequency which is the maximum processor utilization, Aydin, and Q. Yang (2003). On the other hand, if per-core DVFS multiprocessor platforms are assumed, an energy-aware cost function needs to be proposed. An energy-aware cost function depends on average utilization of processors, but it does not give an accurate measure for energy consumption. Then, a tradeoff between average and maximum utilization is introduced. This paper introduces an energy-aware cost function considering simplified energy model as follows:

$$\text{Cost} = \text{Sum}(U_j^2) / m + \text{Penalty for } j\ 1,2, \ldots, m \qquad (9)$$

When applying PSO, the parameters used are the swarm size k = 100, No. of iterations=100, $C_1 = C_2 = 2$ [5], and the inertia W = 1 that, according to the PSO variant used, may be fixed or may decrease linearly until reaching 0 or it may be then restarted (re-excited) to 1 to decrease linearly again.

5. EXPERIMENTS AND DISCUSSION

The approaches have been implemented using MATLABTM. Utilization matrices have been uniformly generated of light tasks with utilization ranges from 0.05 to 0.25 and medium tasks with utilization ranges from 0.25 to 0.5. The implemented approaches are Min-min, Max-min, PSO with fixed inertia (PSO-fi), PSO with varied inertia (PSO-vi), PSO with re-excited inertia (PSOre), and our proposed Min-min based PSO approach (PSOm). With relatively small search spaces, all PSO variants show good results with reasonable number of iterations. But, when search spaces grow, so much iterations are needed to get good results using PSO approaches.

PSO variants using variable inertia such as PSO-vi and PSO-re show better performance than PSO with fixed inertia (PSO-fi) with the same number of iterations and the same problem instances. Figures 5 and 6 below show comparisons among Min-min, Max-min, and PSO variants with 200 iterations for light tasks scheduled on 4 and 10 cores respectively.

Our proposed approach gives the PSO algorithm a push toward the best solution using a particle (solution)

obtained by Min-min. This makes PSO gives better results with reasonable number of iterations slightly more than in [ ].

In the worst case, our proposed approach gives Min-min performance if it could not optimize the solution.

Figures 7 and 8 show the performance of our proposed

approach with 100 iterations and light tasks assigned to 4 and 10 cores respectively. It is obvious that our proposed approach behaves so better when the search space grows.

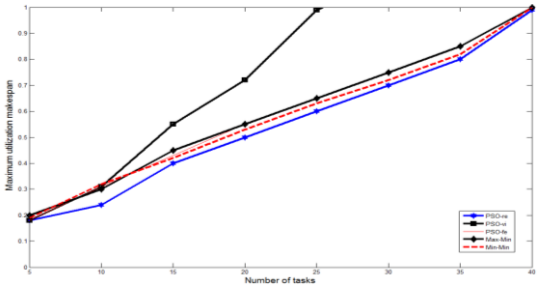

Fig. 5: A comparison of partitioning methods with light tasks partitioned upon 4 processors

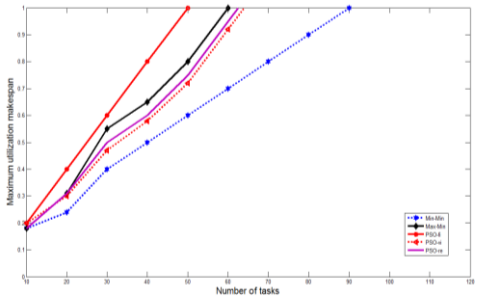

Fig. 6: A comparison of partitioning methods with light tasks partitioned upon 10 processors.

When medium tasks are used, the proposed approach behaves the same way and shows better performance especially with large search spaces. Figures 9 and 10 show the case when medium tasks are partitioned on 8 and 16 processors respectively.

As mentioned earlier, when full-chip DVFS is considered the makespan cost function is used. If per-core DVFS is considered, the introduced energy-aware cost function, Eq. (8), is taken into account. Figures 11 and 12 show the case of partitioning light tasks on per-core DVFS platforms of 4 and 10 cores respectively. It is clear that using makespan cost function, Eq. (6), increases the feasibility (schedulability) of the task set more than using Eq. (8) as a cost function which is more energy efficient.

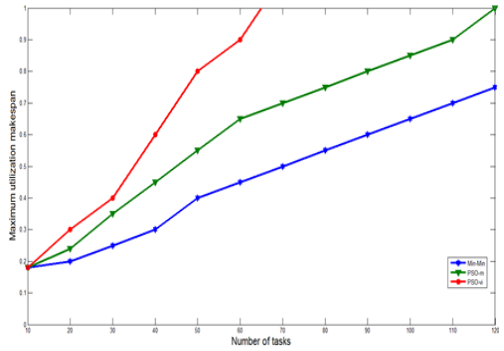

Fig. 7: A comparison among Min-min, PSO-vi, and PSO-m techniques with light tasks partitioned upon 4 processors.

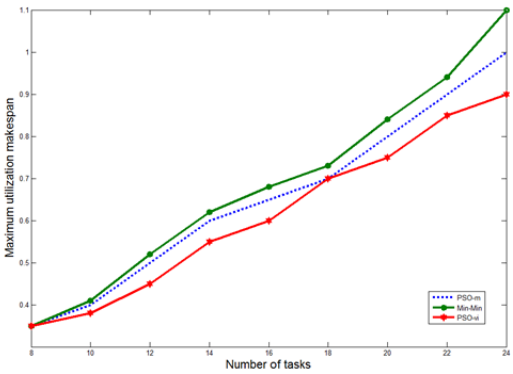

Fig. 8 A comparison among Min-min, PSO-vi, and PSO-m techniques with medium tasks partitioned upon 8 processors.

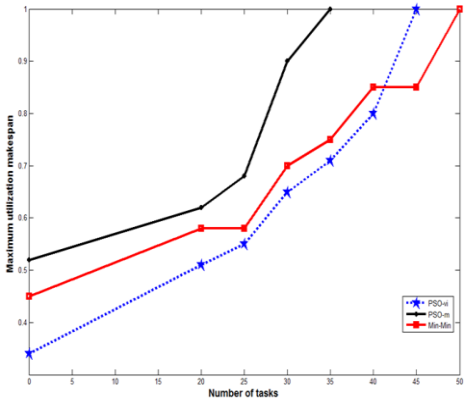

Fig. 9: A comparison among Min-min, PSO-vi, and PSO-m techniques with medium tasks partitioned upon 16 processors.

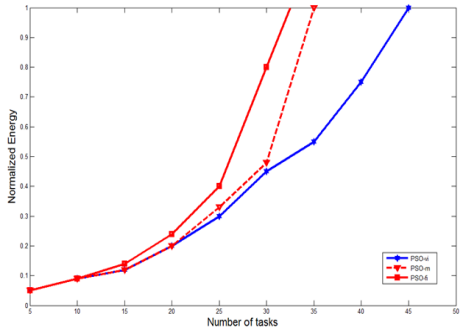

Fig. 10 A comparison among PSO-fi, PSO-vi, and PSO-m techniques with light tasks partitioned upon 4 processors.

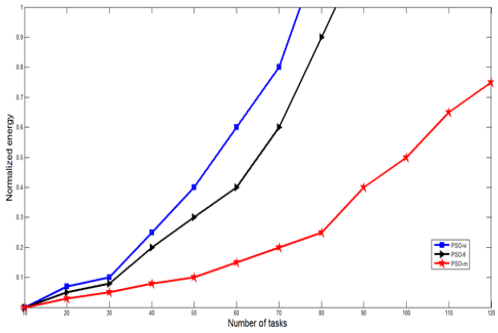

Fig. 11 A comparison among PSO-fi, PSO-vi, and PSO-m techniques with light tasks partitioned upon 10 processors.

It is worth to be noted that another Max-min particle (solution), in addition to Min-min particle, may be added to the population in the initialization step when the task set nature requires that, i.e., when Max-min gives better solutions than Min-min. This occurs when task utilizations are diverse, e.g., when there is a long task in a short-task task set.

6. CONCLUSIONS

This paper considered the problem of power-aware task partitioning on heterogeneous multiprocessor platforms. The paper proposed a modified PSO variant based on Min-min and priority assignment algorithms that outperformed its counterparts in less number of iterations for the same problem instance. Also, the energy-aware cost function is addressed in this paper and it differentiated between the full-chip and per-core DVFS processors. As a future work, any verified polynomial-time partitioning technique can be added as a particle to the population in the initialization step to give the PSO algorithm a forward push to get better solutions.

7. REFERENCES

[1] Dawei, L. and Wu, J., (2012).Task Partitioning Upon Energy-Aware Scheduling for Frame-Based Tasks on Heterogeneous Multiprocessor Platforms. 41st Int. Conf. on Parallel Processing, pp. 430 – 439.

[2] Kong, F. Yi, W. and Deng, Q. (2012). Energy-Efficient Scheduling of Real-Time Tasks on Cluster-Based Multicores. In DATE'11, pp. 1-6.

[3] Chen, J.-J. and Thiele, L. (2009). Task partitioning and platform synthesis for energy efficiency. In the 15th IEEE Int. Conf. on Embedded and Real-Time Computing Systems and Applications, pp. 393-402.

[4] Texas Instruments (TI), OMAP™ Mobile Processors. Available at: http://www.ti.com/general/docs/gencontent.tsp?content Id=46946 [last accessed 15/2/2012].

[5] Abdelhalim, M. B. (2008). Task assignment for Heterogeneous Multiprocessors using Re-Excited Particle Swarm Optimization. In 2008 Int. Conf. on Computer and Electrical Engineering, pp. 23-27.

[6] Aydin, H. and Yang, Q. (2003). Energy-Aware Partitioning for Multiprocessor Real-Time Systems. In IPDPS, pp. 1- 9.

[7] Cong J. and Yuan B., Energy-efficient scheduling on heterogeneous multi-core architectures. In ISLPED '12 Proceedings of the 2012 ACM/IEEE international symposium on Low power electronics and design, 2012, pp. 345-350.

[8] Chitlur N., (2012) QuickIA: Exploring heterogeneous architectures on real prototypes. HPCA '12, pp. 1–8.

[9] Baruah, S. (2004). Partitioning real-time tasks among heterogeneous multiprocessors. ICPP, Montreal, Quebec, Canada, pp. 467-474.

[10] Braun, T. D., Siegel, H. J., Beck, N., Boloni, L. L. Maheswaran, M. Reuther, A. I., Robertson, J. P., Theys, M. D., Yao, B., Hensgen, D. and Freund, R. F. (2001). A Comparison of Eleven Static Heuristics for Mapping a Class of Independent Tasks onto Heterogeneous Distributed Computing Systems. Journal of Parallel and Distributed Computing, vol. 61, 2001, pp. 810-837.

[11] [9] Chen, H. and Cheng, A. M. K. (2005). Applying Ant Colony Optimization to the partitioned scheduling problem for heterogeneous multiprocessors. ACM SIGBED Review, Vol. 2 , No. 2, 2005, pp. 11-14.

[12] Visalakshi, P. and Sivanandam, S. N. (2009). Dynamic Task Scheduling with Load Balancing using Hybrid Particle Swarm Optimization. Int. J. Open Problems Compt. Math, Vol. 2, No. 3, pp. 475 – 48.

[13] Abdullah E., Shalan M., Awadalla M., Saad E. M. (2014). Energy-efficient task allocation techniques for asymmetric multiprocessor embedded systems. ACM Transactions on Embedded Computing Systems (Impact Factor: 1.18).

[14] Saad, E. M., Awadalla, M.A., Shalan, M., and Elewi, A. 2012. Energy-Aware Task Partitioning on Heterogeneous Multiprocessor Platforms. IJCSI International Journal of Computer Science Issues, Vol. 9, Issue 2, No 1, ISSN (Online): 1694-0814, Pp. 176-183.

[15] Omidi, A. and Rahmani, A. M. (2009). Multiprocessor Independent Tasks Scheduling Using A Novel Heuristic PSO Algorithm. pp. 369 – 373.

[16] Chen, J. and Kuo, C. (2007). Energy-efficient scheduling for real time systems on dynamic voltage scaling (DVS) platforms. 13th IEEE Int. Conf., RTCSA, pp. 28-38.

[17] Koufaty, D. Reddy, D. and Hahn, S. (2010). Bias Scheduling in Heterogeneous Multicore Architectures. In Proceedings of the 5th ACM European Conference on Computer Systems (EuroSys), pp. 125 - 138.

[18] Higashino, W. Capretz, M. A. M. and Toledo, M. B. F. (2014). Evaluation of Particle Swarm Optimization Applied to Grid Scheduling. Proc. Of 23rd IEEE WETICE Conf., Parma, Italy, pp. 1-6.

[19] Chou Q. , Ge D, and Zhang R., (2014). PSO Based Optimization of Testing and Maintenance Cost in NPPs. Hindawi Publishing Corporation Science and Technology of Nuclear Installations, Volume 2014, pp. 1-9.