The CSI Journal on
Computer Science and Engineering
Vol. 13, No. 2, 2016
Pages 1-22
Regular Paper

An Energy-Optimal Real-Time Scheduling Algorithm for Unrelated DVS-Enabled Parallel Machines

Mahmood Gholipour[1] Mehdi Kargahi[1, 2] Heshaam Faili[1]
Shahbaz Youssefi[3] Hadi Ravanbakhsh[1]

[1]School of Eectrical and Computer Engineering, University of Tehran,Tehran, Iran
[2]School of Computer Science, Institute for Research in Fundamental Sciences (IPM), Tehran. Iran
[3]Department of Informatics, Bioengineering, Robotics and Systems Engineering, University of Genova, Italy

Abstract

The category of unrelated parallel machines is one of the most realistic models of heterogeneous multiprocessor systems. In this model, the pair of task and machine (processor) jointly specifies the required execution time and energy to complete the task. In this paper, we consider the problem of scheduling real-time tasks on a set of unrelated parallel machines with the capability of voltage selection through Dynamic Voltage Scaling (DVS). The tasks can migrate among the processors and each processor is also permitted to switch among a set of discrete voltage levels (and thus, task-specific speed levels) to minimize the overall system energy usage. A polynomial-time Energy-Optimal Real-Time Scheduling Algorithm (EORTSA) is proposed: At first, the problem of energy-optimal task-to-machine and voltage-level-to-task assignments are formulated as a linear program and used as a polynomial-time schedule ability test for periodic tasks. Second, task sets which are passed the schedule ability test are feasibly scheduled on the machines using a matching-based algorithm. We prove that the worst number of migratory tasks is 2m, where m is the number of machines. Also, the worst-case total number of migrations, preemptions and voltage-level switches is of $O(m^2)$ in each schedule e period, which is a period of time between two arbitrary consecutive task releases in the system. Comparisons with the PCG algorithm show up to 60% energy saving and reveal that EORTSA outperforms PCG in terms of average number of task migrations and preemptions, especially in systems with large number of tasks.

Keywords: Dynamic Voltage Scaling (DVS), Energy Optimization, Migration, Real-Time Systems, Scheduling, Unrelated Parallel Machines.

1. Introduction

Real-time embedded applications need predictable computing platforms and deep information about their behavior with respect to different patterns of usage to satisfy their time-sensitive requirements. Such applications include safety-critical decision making systems such as those related to earthquake, Air Traffic Control (ATC), transportation, militarism, and many others which mostly have real-time and distributed natures. Recent advances in multiprocessor and distributed system design (multi/many-core processors, MPSoCs, clouds, etc.) have resulted great speedups through the execution of different tasks in parallel. The parallel processing elements in the mentioned systems may have either equivalent capabilities (e.g., multi-core processors or homogeneous multi-server systems) or different capabilities and characteristics (e.g., asymmetric multiprocessors or heterogeneous multi-server systems with different services as well as cloud computing systems).

Even systems which are supposed to have homogeneous processors may actually be heterogeneous due to reasons such as process variation [1] and processor aging [2]. In

overall, the diversity of available computer systems which can be connected through interconnection networks, and the distributed nature of the above-mentioned applications, confirm that most systems are inherently heterogeneous. These heterogeneities may be observed in both the computing capability and power consumption of the processing elements, with an emphasis to the fact that one major concern in the design and management of such systems is their energy usage.

As energy is an expensive resource nowadays, many studies in the context of real-time systems have been concentrated on the methods of energy saving in centralized [3, 4, 5] and distributed [6, 7] real-time systems with the consideration of the fact that the performance of such systems should remain in an acceptable level. A well-known method to do such management is to use dynamic voltage scaling (DVS) [5], which is available in most modern microprocessors [8, 9].

Meanwhile, another important property of the heterogeneous distributed real-time systems is that at a specific voltage-level of a DVS-enabled processor, the speed and energy usage are task-specific, namely they depend on the task running on the processor. This property, which is valid for unrelated parallel machines (as defined formally in Section 2), is a realistic consideration for today's computing systems. Such systems may contain different processing units (e.g., graphics co-processors, math co-processors, CPUs with different architectures, etc.), where each unit may better work for specific tasks from both aspects of computation speed and power usage. (Throughout this paper, we use the two terms machine and processor interchangeably).

In the current study, we present a scheduling algorithm for a relatively complex multiprocessor system which reflects a general behavior of the distributed real-time systems mentioned above. We consider a distributed system with periodic real-time tasks, where the system processors are heterogeneous from both aspects of processing capability and power usage. Furthermore, we suppose that speed and energy usage at a specific voltage-level of the processor are task-specific.

We consider a multi-criteria optimization problem which within the real-time tasks should meet their deadline and the energy usage should be minimized. Also, some methods will be proposed to reduce the number of task migrations as well as preemption and voltage-level switches. To the best of our knowledge, we are unaware of any previous study on a similar problem with the properties stated in this paper.

The rest of this paper is organized as follows. The next section reviews some previous work on identical, uniform, and unrelated parallel machines. Then, we present a precise definition of the problem considered in the current study in Section 3. In Section 4, the concept of a schedule period is introduced to reduce the problem to an abstract version. Next, in Section 5, the proposed scheduling algorithm and the respective schedule ability test are described, whereas its optimality and correctnessare proved. In Section 6, we discuss our method to reduce the degree of migrations as well as the number of preemptions and voltage-level switches. Section 7 reports the simulation results and Section 8 concludes the paper and indicates some future works.

2. Background

In this section, we review the properties and the respective studies of three categories of multiprocessor systems, namely identical, uniform, and unrelated parallel machines.

2.1. Identical Parallel Machines

The set of machines which are all identical and have the same computational power is called a set of identical parallel machines. In 1969, Muntz and Coffman [10] presented a level algorithm for systems with two identical processors. A strict fairness constraint for such systems, called P-fair, and a scheduling algorithm based on P-fair has been presented by Baruah et al. [11]. According to this algorithm, the resources are given to the tasks in proportion to their weights. Due to its strict fairness, this algorithm needs to reschedule tasks many times, which results in many task switches and migrations, and reduces the efficiency of the algorithm. A similar idea, namely the Fluid schedule is also given by Holman and Anderson [12].

The Fluid schedule, which is an ideal policy, assumes that each task executes at a constant rate. This algorithm is almost inefficient due to its high number of rescheduling. However, in the study, the Stagger model is also proposed which improves the P-fair algorithm.

Dertouzos and Mok [13] presented a model for expressing the relation between the laxity and remaining computation time of a task, called Laxity and Computation plane (L-C plane). The laxity of a task is a measure of urgency, which is represented on the x-axis and its remaining computation time is shown on the y-axis. Cho et al.[14] extended the idea of representing tasks as tokens in the L-C plane and introduced Time and Local Execution Time Domain Planes (T-L planes), where time and the remaining execution time are represented on the x-axis and y-axis, respectively. They proposed Largest Local Remaining Execution Time First (LLREF) scheduling algorithm based on the T-L plane and proved that it is an optimal online scheduling algorithm for identical parallel machines.

Megel et al. [57] introduced a novel approach to decrease preemption and migration count in optimal global real-time scheduling on identical multiprocessors. Their approach is composed of an offline and an online step. In the offline step, a linear program decides about placing jobs on intervals. Scheduling of associated jobs within each interval is done dynamically by an online local scheduler. Their approach suffers from severe issues: the proposed mixed integer linear programming of offline part is computationally complex since the problem is solving in a hyper period and the number of MILP variables scale with the task count, processor count and length of the hyper period.

For the sake of energy efficiency, different approaches have been investigated which usually consider continuous voltage-levels for the identical machines. Lee [15], through Dynamic Voltage and Frequency Scaling (DVFS) in a lightly-loaded multi-core system with periodic tasks, achieved up to 64% energy saving comparing to some simple methods. Chen et al. [16] devised a 1.13-approximation algorithm for scheduling migratory periodic tasks on identical multiprocessors, assuming each processor can take arbitrary operating frequencies. Later, Chen and Kuo [17]

proposed an optimal algorithm for migratory tasks and a 1.412-approximation when for non-migratory tasks, considering that the tasks have different power characteristics. Also, Chen et al. [18] proposed an algorithm with a 1.283-approximation bound for energy usage to schedule periodic tasks over identical DVS-enabled multiprocessor, considering both dynamic and static sources of power consumptions. If the overhead of turning processors on and off is not negligible, though, their algorithm is within a 2-approximation bound.

2.2. Uniform Parallel Machines

A set of processors which are solely characterized by their computation speeds is called a set of uniform parallel machines. Therefore, a job that executes for t time units on a processor with the computation speed s performs $s \times t$ units of execution. Horvath et al. [19] extended the idea of the level algorithm proposed in [10] to uniform heterogeneous multiprocessors. Gonzalez and Sahni [20] proposed an offline optimal$O(n + m)$ scheduling algorithm with no more than $2(m - 1)$ preemptions, where n is the number of tasks and m is the number of processors.

Other works have also been done on the online scheduling. Hochbaum and Shmoys [21] presented a polynomial approximation algorithm to solve the minimum makespan problem on uniform heterogeneous multiprocessors. Baruah and Goossens [22] tried to adapt the Rate Monotonic (RM) scheduling algorithm [23] to find a static-priority online scheduling algorithm for uniform multiprocessors.

They also presented a RM-feasibility test for this problem, generalizing the RM-feasibility test of identical heterogeneous multiprocessor systems. In a different study, Funk et al. [24] presented a feasibility condition for periodic task sets to be executed on uniform heterogeneous multiprocessors. This condition (which is called FGB in [25]) is based on the Earliest Deadline First (EDF) scheduling algorithm [23].

In the same study, they also presented an EDF-feasibility test; however, no efficient algorithm was found to schedule the task sets satisfying the FGB condition. Chen and Hsueh [25] proposed an optimal online $O(n^2 \log n)$ scheduling algorithm called PCG with at most n rescheduling, where n is the number of tasks. In their study, T-L_{er} plane was introduced and a greedy algorithm was proposed. They also proved the optimality of their algorithm based on the FGB condition.

In regard of energy efficiency in uniform parallel machines, Funaoka et al. [54] proposed a real-time static voltage and frequency scaling (RT-SVFS) technique based on an optimal real-time scheduling algorithm (LLREF) by means of T-L Plane transformation, which is a technique to apply processor frequency scaling to LLREF scheduling. They proposed a uniform RT-SVFS algorithm and then extended it to an independent RT-SVFS for uniform platforms.

Later, in [55] they extended this work and proposed a real-time dynamic voltage and frequency scaling (RT-DVFS) techniques based on RT-SVFS. Despite the significant run-time overhead associated to it, RT-DVFS obtains better energy saving than RT-SVFS.

2.3. Unrelated Parallel Machines

In unrelated parallel machines, there is an execution rate s_{ij} associated with each task-processor pair (T_i, P_j), with the interpretation that task T_i performs $s_{ij} \times t$ units of execution if executed on processor P_j for t time units. Two common methods for solving such scheduling problems in parallel machines are task partitioning and global scheduling. In the former method, certain tasks are only allowed to be executed on certain machines and only partial migration is allowed. Using this type of static decision has some performance advantages such as reducing the migration overhead. For example, Yu and Prasanna [26] explored minimizing the energy usage of periodic real-time tasks on unrelated parallel machines by task partitioning and proposed a polynomial-time scheduling algorithm by relaxing integer linear programming. The method of task partitioning is generally sub-optimal. Due to the NP-hard nature of such scheduling problems in their general form [12, 27], it is convenient to propose heuristics or efficient algorithms for special forms of the problems. Accordingly, the latter method, namely global scheduling is desired in many cases to feasibly schedule the task sets, in which an important criterion is to keep low the number of migrations.

Lenstra et al. [28] presented a polynomial approximation algorithm for unrelated parallel machines to find a schedule that minimizes the makespan. They guarantee that it is not longer than twice the optimal value for the case where the number of processors is fixed. They also proved that no polynomial algorithm could achieve a worst-case ratio less than 1.5 unless P = NP. For the case of two processors, Gonzalez et al. [29] introduced a linear time algorithm to construct optimal scheduling which have at most two preemptions. Jansen and Porkolab [30] gave a fully polynomial-time approximation algorithm for the problem of scheduling tasks in the case where each task can be executed only on one processor. Srivastava [31] developed a tabu search heuristic for minimizing makespan in that problem. This algorithm produces near-optimal results with reasonable amounts of computational time for problems of reasonable size. Baruah [32] proposed a polynomial-time feasibility test for the global heterogeneous multiprocessor scheduling. They also presented a scheduling solution as a proof for correctness of the test; such schedule however suffers from high preemption and migration cost.

The lack of an optimal scheduling algorithm with reasonable cost motivated us to propose an optimal scheduling algorithm; however optimality is only one of aspect of our work and we also consider the energy optimality of scheduling algorithm. Recently, Cucu-Grosjean and Goossens [56] proposed two exact schedule ability tests for task-level fixed-priority and job-level fixed-priority earliest deadline first scheduler on specific tasks executing on unrelated parallel machines. Also, for a special case of heterogeneous parallel machines which includes two type of processors, Raravi et al. [58] proposed two scheduling algorithms with time complexity of $O(n\log(n))$ for implicit deadline sporadic task set. They consider that task migration is only possible among processors of the same type.

Many works have also been done on energy-constrained unrelated parallel machines. Hsu et al. [33] and Chen and Kuo [34] tried to find a solution for cost minimization of machines and allocation, respectively under the given timing

and energy usage constraints. Chu et al. proposed a near-optimal [35] and later an optimal [36] solution to the voltage-setup problem; their work is the assignment of voltage-levels to processors with the limitation that each processor assumes one voltage-level throughout the system lifetime. Luo and Jha [37] devised both static and dynamic variable voltage scheduling algorithms to minimize the energy cost of scheduling dependent periodic and aperiodic tasks on unrelated parallel machines.

Some studies consider discrete voltage-levels for non-migratory task, and thus, they use different approaches to solve a NP-Complete problem. Ding et al. [38] approached the problem using the ant colony optimization in order to obtain energy efficient scheduling of partially dependent tasks on DVS-enabled processors, even with coarse-grained voltage modes. Lin et al. [39] addressed the problem when systems in question work on rechargeable batteries. They solve the problem based on four heuristics including genetic algorithm and ant colony optimization.

Yang et al. [7] used a fully polynomial-time approximation scheme for such problems. Yu and Prasanna [26] approximated the energy efficiency when assigning non-migratory tasks to DVS-enabled processors. They formulated the problem as an extended generalized assignment problem and also hired a heuristic to approximately solve the NP-Complete problem. They used a model in which the DVS-enabled machines have discrete voltage-levels, where both speed and energy usage of machines depend on the voltage-levels as well as the tasks they are running.

The above studies, although valuable, suffer from at least one of the following problems: (i) They do not take the advantages of DVS for runtime energy management into account [35, 36], (ii) Machine speeds are available in a continuous spectrum [35, 36, 37], (although power-supply electronic may provide systems with continuous voltage spectrum in the future, it is a fact that most of the currently available DVS-enabled microprocessors (e.g. Trans meta, Intel, and AMD) use a few discrete voltage-levels [40]), (iii) Energy usage of a machine depends either only on its specification (speed and/or workload) or only on the task that is running, rather than both [35, 36, 37, 38, 34, 7, 39, 33], and (iv) No task migration is permitted [26, 35, 36, 37, 38, 34, 7, 39, 33].

Although task migration in unrelated parallel machines offers potential energy/performance gains, there are also high costs and complexities involved in the migration process. Hence, it has not attracted considerable attention from researchers. However, according to recent researches [41, 42, 43, 44], this cost has became reasonable and task migration in unrelated parallel machines has become a feasible option. We consider task migration to better utilize the available heterogeneity of the platform as well as to achieve better power-performance efficiency.

3. System Model, Problem Definition and Conceptual Solution[1]

In this section, we introduce the model of the system. Also, we present the precise definition of the problem under study a general view to the solution method.

3.1. Task Model

We consider n periodic tasks shown as the task set $T = \{T_1, \ldots, T_n\}$. Each task T_i, i=1,…,n is independent of the other tasks and is described as (p_i,e_i), where p_i is the task period, e_i is its execution requirements and the respective relative deadline is again p_i. Additionally, there is a dummy task represented by T_0 with an execution time and period equal to the sum of the processor idle times and hyper period respectively.

3.2. Processor Model

The system consists of m unrelated parallel machines shown as the set $M = \{M_1,\ldots,M_m\}$ with no resource contentions. Machine M_j, j=1,…,m has k_j voltage-levels as $V_j = \{V_{j1},\ldots,V_{jk_j}\}$. At the l^{th} voltage-level of M_j, namely V_{jl}, l=1,…,k_j, a maximum speed S_{jl}, $(S_{i1} < S_{i2} < \cdots < S_{ik})$ is achievable and a maximum power Pow_{jl}, $(Pow_{j1} < Pow_{j2} < \ldots < Pow_{jk_j})$ may be consumed. Based on the characteristics of the running task at voltage-level V_{jl}, the attainable speed and the respective power usage may be far less than S_{jl} and Pow_{jl}, respectively.

3.3. Task-Processor Model

As also indicated in the processor model above, we consider m unrelated parallel machines (note that uniform and identical parallel machines are special cases of unrelated parallel machines). Therefore, each task T_i, i=1,…,n if assigned to a processor M_j, j=1,…,m which works at the voltage-level V_{jl}, l=1,…, k_j, can be run with a speed of S_{ijl} which certainly is a fraction of $S_{jl}(0 < S_{ijl} \le S_{jl})$. Meanwhile, the task-specific power usage of processor M_j working at voltage-level V_{jl}, when task T_i is run upon which, will be Pow_{ijl}, which again is a fraction of Pow_{jl} $(0 < Pow_{ijl} \le Pow_{jl})$. In addition, Pow_{0jl} is the power usage of processor M_j when it is idle, or equivalently executes the dummy task T_0, at voltage-level V_{jl}. In this regards, if T_i runs for duration of length t on machine M_j working at voltage-level V_{jl}, the energy usage of the task-processor pair in the duration can be calculated as:

$$E_{ijl}(t) = Pow_{ijl} \times t \tag{1}$$

To obtain the task execution times at different voltage-levels of the unrelated parallel machines, we consider the following information. Source code of each task is compiled separately for every supported processor in the set of unrelated parallel machines. The generated binary codes will be executed on the corresponding processor, and thus, the worst-case number of CPU cycles is determined. For each voltage-level of the DVS-enabled processor, a similar scenario is followed. For each task, the processor with the longest execution, which is less than or equal to the task deadline, will be considered as the base processor. We consider the execution speed of the base processors as 1 and execution speed of the task on the other processors are normalized with respect to this execution time. In our notation, we refer to the task execution time on base

processor as task execution time e_i in our task model.

Moreover, each task can run on at most one machine at each instant of time and can arbitrarily migrate across different machines during its execution. Without loss of generality in the proposed algorithm, we assume negligible overheads for task migrations, preemptions, and voltage-level switches in the current study. By the way, we propose some techniques which significantly reduce the number of such events, and thus, reduce the respective overheads.

3.4. Problem Definition

The problem considered in this paper is to schedule a feasible task set T on a given set of machines M with the goal of minimizing the power consumption. We need to divide the task instances (jobs) into job slices. A job slice is defined by a vector $(T_i, M_j, V_{jl}, t_s, t_f)$, meaning that task T_i is scheduled on machine M_j working at voltage-level V_{jl} from time t_s to time t_f. Furthermore, it is considered that $[t_s, t_f)$ is a period where the task is executed uninterruptedly, i.e., with no task migration, preemption, or voltage-level switching during that period. Equivalently, we use $(T_0, M_j, V_{jl}, t_s, t_f)$ to denote that M_j is in idle mode in the period of $[t_s, t_f)$.

Let define the average system energy usage in a time interval $[a,b)$ as:

$$\overline{E(a,b)} = \frac{(\sum_{i=0}^{n}\sum_{j=1}^{m}\sum_{l=1}^{k_j} Pow_{ijl} \times \hat{t}_{ijl})}{(b-a)} \quad (2)$$

where $\hat{t}_{ijl} \le (b-a)$ is the total length of time during time interval $[a,b)$ in which task T_i runs on machine M_j at voltage-level V_{jl}. The resulting schedule is to minimize the average system energy usage throughout the system lifetime. This can be achieved by minimizing the average system energy usage in a hyperperiod of length H, i.e., $(\overline{E(0,H)})$, where the resulting energy is shown as $\overline{E(H)}_{\min}$. This goal can be achieved by appropriate determination of the job slices along with the consideration of all other constraints.

3.5. Conceptual Solution

The proposed solution to the defined problem can be summarized as follows:

1. We first reduce the scheduling problem in a hyperperiod to scheduling the scaled tasks in Schedule Periods (SPs) of unit length (defined in Section 4) that is easier to solve.
2. Then, we model the scheduling problem in the SPs with Linear Programming (LP). If a solution to this LP exists, it specifies the required amounts of time that each task needs to be run at every voltage-level on each machine. These amounts of execution minimize the system energy usage in a manner that all deadlines are met. The algorithm for this step can be independently used for schedulability test of real-time periodic tasks on unrelated parallel machines and is explained in detail in Section 5.1.
3. Tasks are assigned to processors based on the results of Step 2. An extension to Lemma 2 of [32] is a solution for task scheduling; such a solution suffers from large number of preemptions, migrations and voltage-level switches. Hence, we use this solution only to prove the existence of a feasible schedule. Instead, we have proposed a scheduling algorithm which aims to reduce the number of migrations and preemptions. Our algorithm is composed of the following steps:
   - 3.1.1 Each SP interval is divided into several time intervals with different lengths. The procedure to obtain lengths of these intervals is explained in Section 5.
   - 3.1.2 For each interval, task to processor assignments is found by solving a custom model of minimum-cost network flow problem. This matching is selected in a way that it leads to significant decrease in the number of preemptions and migrations in the hyper period. Procedure from Steps 3.1.1 and 3.1.2 implicitly prevents the overlaps of execution of segments of a task on different processors.
   - 3.1.3 Number of migrations, preemptions and voltage-level switches is further reduced by use of mirroring technique which discussed in Section 5.2.3.

   3.2 Then, the remaining tasks, i.e., non-migratory ones, are scheduled on the machines. The main point about scheduling non-migratory tasks is that we are not concerned about the parallel execution of job slices of a task. Hence they can be scheduled using an algorithm such as EDF in the remaining free intervals of the machines which prevents excessive preemptions and voltage-level switches. Subsection 5.3 describes this step, which completes the solution.

4. Schedule Period and Feasible Scheduling

To address the aforementioned problem, in this section we introduce the concept of schedule period, which has been borrowed from Fluid scheduling method [12] and is used to convert the scheduling problem into an equivalent but simpler one. The idea of Fluid scheduling is to execute the tasks at a constant rate in a way that the tasks execution are finished exactly at their deadline [12]. In a T-L plane, as introduced in Section 2, tokens representing the tasks move over time. The trajectory of such a token during the execution of the respective task according to its Fluid schedule is called the task Fluid path.

Therefore, the Fluid path is simply a straight line between the coordinates (r, e) and $(r + p, 0)$ in the respective T-L plane, where r is the task release time, e is its execution requirement, and p is the respective relative deadline (see figure 1). The actual path is the result of concatenation of execution lines of the task. The slope of each execution line depends on S_{ijl} for task T_i when it is running on machine M_j at the voltage-level V_{jl}. This slope is zero when the task is not running on any machines.

We define a Schedule Period (SP) as a period between two arbitrary consecutive task releases in the system. Scheduling is done on a per SP basis. Consider n tasks $T_1, T_2, ..., T_n$ with their Fluid paths as shown in figure 2. In the time period between two task releases, namely in SP_k (with start time of s_k and finish time of f_k), each task T_i, according

to its Fluid schedule, executes for a certain amount of time, shown as e_i^k. This piece of a task is called a scaled task. Then the relative deadlines for the execution of all the scaled tasks located in SP_k are equal to the length of the respective SP (shown as $|SP_k|$ in this case). Accordingly, a new set of scaled tasks is resulted which we call it scaled task set. Thus, the scaled task set is shown as $\{ST_1, ST_2, ..., ST_n\}$ with each scaled task ST_i having an execution requirement of e_i^k and all the scaled tasks having a relative deadline of length $|SP_k|$.

It is proven in Theorem 1 below, which a task set is schedulable with average energy usage E, if and only if the respective scaled task set in a SP is schedulable with the same amount of average energy usage. Therefore, the problem is reduced to scheduling the scaled tasks in only one SP. The resulting schedule can then be used to schedule the scaled tasks in every SP as well as the task set in a hyper period, and as a result, throughout the system lifetime.

Theorem 1. A periodic task set is schedulable on a set of unrelated parallel machines with an average energy usage E, if and only if the respective scaled task set is schedulable in a SP with the same average energy usage on the same set of machines.

Proof. Suppose the scaled task set is schedulable in a SP, namely SP_k, with average energy usage E. Consider the tasks and their Fluid paths as in figure 2. For each task T_i, the slope of its Fluid path is e_i/p_i. In the schedule period SP_k, each scaled task ST_i has an execution requirement of:

$$e_i^k = |SP_k| \times e_i / p_i \tag{3}$$

In this regards, for another schedule period $SP_{k'}$ with the length $|SP_{k'}|$, we can scale the schedule obtained for SP_k by a factor of $|SP_{k'}|/|SP_k|$. Then we obtain a new schedule for $SP_{k'}$, within which each scaled task S T_i has a execution requirement of $e_i^{k'}$ as:

$$|SP_k| \times e_i/p_i \times |SP_{k'}|/|SP_k| = |SP_{k'}| \times e_i/p_i \tag{4}$$

In general, schedule scaling for SP_k by a factor of K means that corresponding to each job slice $(T_i, M_j, V_{jl}, s_k + t_s, s_k + t_f)$ with an average energy usage of $E(s_k + t_s, s_k + t_f) = Pow_{ijl}$ in the primary schedule, there is a job slice $(T_i, M_j, V_{jl}, s_{k'} + K \times t_s, s_{k'} + K \times t_f)$ in the scaled schedule with the average energy usage of $E(s_{k'} + K \times t_s, s_{k'} + K \times t_f) = Pow_{ijl}$. Clearly, the average energy usage does not change for any job slices in the scaled schedule and all job slices' lengths are scaled by the same factor K. Thus, the average energy usage of $SP_{k'}$ would be $\overline{E(s_{k'}, f_{k'})} = \overline{E(s_k, f_k)}$.

However, this new execution requirement (resulting from (4)) is the same as the execution requirement for task T_i in the schedule period $SP_{k'}$, i.e., $e_i^{k'}$. Therefore, to obtain a schedule for an arbitrary schedule period $SP_{k'}$ with length $|SP_{k'}|$, it is sufficient to multiply each job slice in the given scheduling of SP_k by $|SP_{k'}|/|SP_k|$.

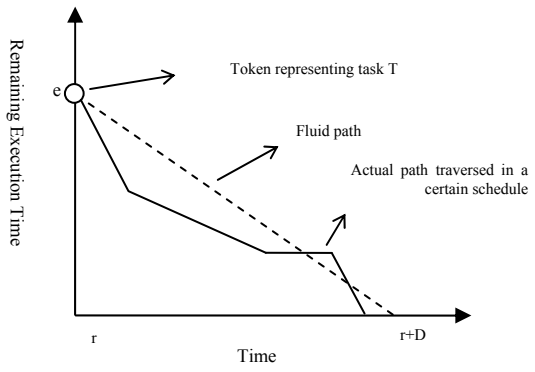


Figure 1. A sample T-L plane

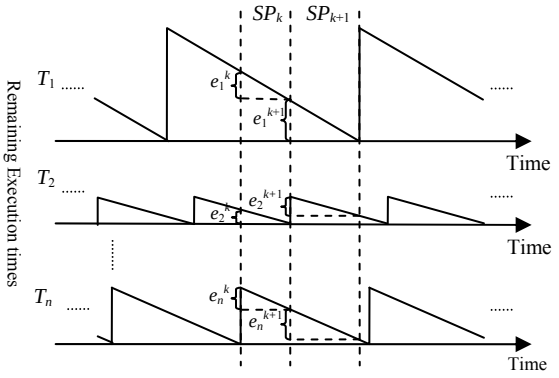


Figure 2. Schedule periods

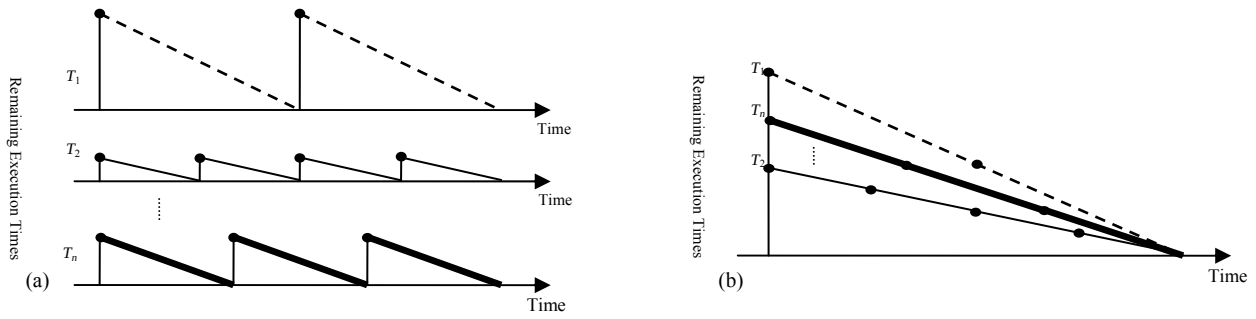


Figure 3. Relaxing the problem of scheduling in a SP to one hyper period: a) Fluid schedule of periodic tasks in one hyper period, b) Cumulated execution times of the tasks in the hyper period

In this way, the task set would be schedulable in all the SPs. Scheduling the scaled tasks in each SP results in each scaled task token in the respective T-L plane to meet the task's Fluid path at the end of the SP. Since all the scaled task deadlines are located at the end of their respective SPs, all deadlines are met by individual scheduling of each SP. Besides, average energy usage in all the SPs is the same and equal to E. Thus, the average energy usage in a hyper period would be the same. Thus, the periodic task set is schedulable with the average energy usage E.

To prove the theorem in the inverse direction, suppose the task set is schedulable. Consider the length of the tasks hyper period is H and the average energy usage is E. In this regards, task T_i is executed H/p_i times in the hyper period, each with the length of e_i. Therefore, in each hyper period, task T_i has a cumulative execution requirement of length $H \times e_i / p_i$. The problem of scheduling tasks with $H \times e_i / p_i$, i=1,…,n execution requirements, all having a relative deadline equal to H, is a relaxed version of the original problem (see figure 3).

In other words, we can scale down the scheduling in a hyper period by factor $|SP_k|/H$ to address the problem of scheduling the respective scaled task set in SP_k for an arbitrary value of k. The reason is that each task T_i must be executed for $|SP_k| \times e_i / p_i$ which is equal to $H \times e_i / p_i \times |SP_k|/H$. Besides, the average energy usage remains equal to E, because scaling a scheduling does not change its average energy usage. Thus, the proof is complete.

Let $\overline{E(SP)}_{\min}$ show the minimum value of the average energy usage of the scaled task set in a SP that can be obtained through proper scheduling. Then we have:

Corollary 1. The minimum average energy usage for a set of periodic tasks on a set of unrelated parallel machines is E if and only if the minimum average energy usage for scheduling the corresponding scaled task set in an arbitrary SP (e.g., SP_k) on the same set of machines is E, or equivalently, $\overline{E(SP)}_{\min} = \overline{E(H)}_{\min}$.

Proof. We use contradiction for the proof. First, consider that there is a scheduling with an average energy usage $\overline{E(SP)}_{\min}$ ($< \overline{E(H)}_{\min}$) for the scaled task set in a SP. However, according to Theorem 1, we were able to schedule the periodic task set with the average energy usage $\overline{E(SP)}_{\min}$, which is in contradiction to the optimality of $\overline{E(H)}_{\min}$. Similarly, for the inverse direction, consider that there is a scheduling with average energy usage $\overline{E(H)}_{\min}$ ($< \overline{E(SP)}_{\min}$) for the periodic task set. Again, according to Theorem 1, we were able to schedule the scaled task set with the average energy usage $\overline{E(H)}_{\min}$ in the SP, which is in contradiction to the optimality of $\overline{E(SP)}_{\min}$.

In the following, we provide a scheduling algorithm for the scaled task set in a SP of unit length to minimize the energy usage. Then we use the obtained schedule to construct a global schedule with minimal average energy usage for the task set.

5. Energy-Optimal Task Scheduling on Unrelated Parallel Machines

The proposed Energy-Optimal Real-Time Scheduling Algorithm (EORTSA) has three steps. First, to assure that all the deadlines can be met and the energy usage is minimized, we find the required amounts of time that each task needs to be run on each machine, as well as the length of time that it needs to spend at each voltage-level on that machine. The algorithm for this step, explained in detail in Section 5.1, can be independently used for schedule ability test of real-time periodic tasks on unrelated parallel machines. Second, we schedule the tasks which need migration, namely migratory tasks, on the appropriate machines. The details of this step are discussed in Section 5.2. Finally, the remaining tasks, i.e., non-migratory ones, are scheduled on the machines determined in the first step, only in the remaining free intervals of the machines, which completes the solution as discussed in Section 5.3.

5.1. Schedule Ability Test

In this subsection, we use linear programming to assign the system tasks to the unrelated parallel machines and set their speeds in a manner that the average system energy usage is minimized and all the deadlines are met. This assignment can then be used as a schedule ability test for the system. Given n tasks and m machines, the amounts of time that each task T_i, i=1,…,n needs to be run on machine M_j, j=1,…,m, at voltage-level V_{jl}, l=1,…,k_j, namely t_{ijl} in a SP of unit length is derived. The objective function here is to minimize the average energy usage in a SP, which is equivalent to minimizing the total energy usages of the machines (based on Corollary 1). Thus, the optimization problem is formulated as follows (notice that the length of SP is considered 1):

Minimize

$$\overline{E} = \sum_{j=1}^{m} \left(\sum_{i=0}^{n} \left(\sum_{l=1}^{k_j} Pow_{ijl} \times t_{ijl} \right) \right) \quad (5)$$

Subject to

$$\sum_{j=1}^{m} \sum_{l=1}^{k_j} s_{ijl} \times t_{ijl} = e_i / p_i, \quad \text{i=1,...,n}$$

$$\sum_{i=0}^{n} \sum_{l=1}^{k_j} t_{ijl} = 1, \quad \text{j = 1,...,m}$$

$$\sum_{j=1}^{m} \sum_{l=1}^{k_j} t_{ijl} \le 1, \quad \text{i = 1,…,n}$$

$$t_{ijl} \ge 0, \quad \text{i = 1,...,n, j = 1,...,m, l = 1,...,}k_j$$

The first constraint above indicates that the total normalized execution time of task T_i with respect to the speed levels of the machines at different voltage-levels should be equal to its normalized expected execution requirement. The second constraint indicates that the idle times of machine M_j and the total execution time of the job slices assigned to that machine should be equal to the length of the respective SP. Also, the third constraint mentions that the total execution time of task T_i on different machines

working at different voltage-levels should not violate the scaled task deadline, which is again the length of the SP (i.e., the task cannot be executed for longer than the SP length). The last constraint simply indicates that all the times are nonnegative. Solving this LP problem, the t_{ijl}s, i = 1,…,n, j = 1,…,m, l = 1,…,k_j, are obtained which specify how long each task T_i should be executed on machine M_j at voltage-level V_{jl} to minimize the overall system energy usage.

In other words, we model the energy-optimal scheduling as a LP problem. This LP model could be used as a schedule ability test as follows. If there is not any solution for scheduling the given task set on the given set of parallel machines, then the LP problem does not have any solution. In addition, it is guaranteed that if the task set is schedulable, then the LP problem certainly has a solution.

According to the optimization, if there are two voltage-levels l and l' for machine M_j in an order that $t_{0jl} > 0$ and $t_{0jl'} > 0$ while $Pow_{0jl'} \geq Pow_{0jl}$, the energy usage can be reduced by setting t_{0jl} to $t_{0jl} + t_{0jl'}$ and $t_{0jl'}$ to zero, which has no effect on the satisfaction of the other constraints. Applying this method iteratively, we can reach to a situation in which there are no two voltage-levels for a machine M_j so that $t_{0jl} > 0$ and $t_{0jl'} > 0$, and therefore, machines use only one voltage-level when they are idle. Example 1 illustrates how this optimization works.

Example 1: Suppose a machine set $M = \{M_j \mid 1 \leq j \leq 4\}$, where M_1 and M_4 have two voltage-levels and the other machines work in only one voltage-level. The execution requirements and periods of the tasks of $T = \{T_i \mid 1 \leq i \leq 7\}$ are listed in table 1. Given the task power consumptions (Table 2) and execution speeds (Table 3) on different voltage-levels, the non-zero variables have been obtained by solving the LP, as shown in table 4.

Corresponding to each $t_{0jl} > 0$ $(i > 0)$, we define a task segment (TS_{ijl}) of length t_{ijl}, with the interpretation that there is a segment of task $T_i$to be executed upon machine M_j at voltage-level V_{jl}. For the SP under discussion, we divide these segments into two categories: segments which belong to the tasks running over more than one machine in that SP, referred to as segments of migratory tasks, and segments running over only one machine during the SP, referred to as segments of non-migratory tasks.

In Sections 5.2 and 5.3, respectively, scheduling of migratory and non-migratory task segments over the processors is discussed.

Table 1. Execution requirements and periods of tasks

Task Characteristics	**T_1**	**T_2**	**T_3**	**T_4**	**T_5**	**T_6**	**T_7**
E	40	90	40	20	75	8	10
P	40	100	50	25	100	10	12

Table 2. Power usage of different tasks on different voltage-levels

Voltage-level	**T_0**	**T_1**	**T_2**	**T_3**	**T_4**	**T_5**	**T_6**	**T_7**
V_{11}	1	5	7	2	2	2	3	3
V_{12}	2	7	9	3	4	4	6	5
V_{21}	1	2	4	6	3	4	5	3
V_{31}	1	5	6	2	2	2	11	5
V_{41}	1	2	2	2	2	2	4	4
V_{42}	2	4	5	5	5	3	6	6

Table 3. Execution speed of different tasks on different voltage-levels

Voltage-level	**T_1**	**T_2**	**T_3**	**T_4**	**T_5**	**T_6**	**T_7**
V_{11}	1	0.5	1	0.5	0	1	0.5
V_{12}	2	1	2	1	0	2	1
V_{21}	0.5	2	0	0	0	2	2
V_{31}	2	1	0.5	2	2	0	0.5
V_{41}	0	1	1	0	0	0.5	1
V_{42}	0	2	2	0	0	1	2

Table 4. List of non-zero variables at optimal poin

Variable	**Value**	**Type**
t_{112}	$0.16\overline{904761}$	Segment of a migratory task
t_{121}	$0.4\overline{238095}$	Segment of a migratory task
t_{131}	0.225	Segment of a migratory task
t_{241}	0.9	Segment of a non- migratory task
t_{312}	0.35	Segment of a migratory task
t_{341}	0.1	Segment of a migratory task
t_{431}	0.4	Segment of a non- migratory task
t_{531}	0.375	Segment of a non- migratory task
t_{611}	$0.4\overline{809523}$	Segment of a migratory task
t_{621}	$0.15\overline{952380}$	Segment of a migratory task
t_{721}	$0.41\overline{6}$	Segment of a non- migratory task

5.2. Scheduling Migratory Tasks

In this subsection, we discuss the method of scheduling segments of migratory tasks in a SP of unit length. Without loss of generality, we perform the scheduling in the interval [0,1). All the segments of migratory tasks, i.e., TS_{ijl}s should be scheduled on their corresponding machines (M_j) with no parallel execution of different job slices of the same task. This scheduling is done through an iterative algorithm, namely Algorithm 1, as shown in Algorithm 1.

Algorithm 1. Scheduling algorithm for migratory tasks

1.	For each t_{ijl} ; i=1,…,n;j =1,…,m; l=1,…,k_i
1.1.	if T_i is a migratory taskthent'$_{ijl}$=t_{ijl}
1.2.	else t'$_{ijl}$=0
2.	$\gamma = 1$
3.	prevX = ∅
4.	nextX = ∅
5.	while$\gamma > 0$
5.1.	U = {urgenttasks}
5.2.	F = {fullmachines}
5.3.	if (U ≠ ∅ or F ≠ ∅)
5.3.1	nextX= matching (prevX)
5.4.	determine value of σ// σ is the time that is scheduled in this iteration
5.5.	for each $(T_i, M_j) \in$ nextX
5.5.1	an arbitrary task segment TS_{ijl} is scheduled in the interval of $[\gamma-\sigma,\gamma)$
5.5.2	t'$_{ijl}$ = t'$_{ijl}$– σ
5.6.	prevX = nextX
5.7.	$\gamma = \gamma - \sigma$

In the beginning of each iteration, all machines have idle periods in the range of $[0,\gamma)$. In each iteration, some part of scheduling is done in the time interval $[\gamma-\sigma,\gamma)$, where the length of such interval, i.e. $\sigma > 0$, is specified through the determination of some task-machine pairs and their respective voltage-levels. Before going further, we introduce some notations and terminologies. Having the value of γ at the beginning of an iteration and t'_{ijl}s as defined in Algorithm 1, we define machine M_j as full if

$$\sum_{i=1}^{n}\sum_{l=1}^{k_j} t'_{ijl} = \gamma \quad (6)$$

which means machine M_j should execute migratory tasks for all the remaining time and has no other time to be idle or execute non-migratory tasks during the interval $[0,\gamma)$. We use F to refer to the set of such full machines. Similarly, we call a migratory task T_i as urgent if

$$\sum_{j=1}^{m}\sum_{l=1}^{k_j} t'_{ijl} = \gamma \quad (7)$$

namely, task T_i should be executed continuously in the remaining period $[0,\gamma)$ in order to be completed on time. We use U to refer to the set of urgent tasks. In each iteration, full machines and urgent tasks are selected to be scheduled. This decision is made to prevent unnecessary preemptions and migrations. For this purpose, in each iteration, we need to find a bijective function $\chi: D \rightarrow R$, where D and R are set of tasks and machines, respectively, satisfying conditions $U \subseteq D \subseteq T$ and $F \subseteq R \subseteq M$. More precisely, χ is a matching that maps each urgent task to a (possibly full) machine and some other tasks to the remaining full machines.

For each $(T_i, M_j) \in \chi$, there is at least one segment of a migratory task TS_{ijl} while $t'_{ijl} > 0$. This means that task T_i still must be executed on machine M_j for some amounts of time. Our proposed approach for finding the matching will be discussed in detail in Subsection 5.2.1.

Two approaches are followed in the matching algorithm to reduce the number of migrations, preemptions, and voltage-level switches:

1. In each iteration, to find the matching χ, the algorithm gets feedback from the task-to-processor assignments of the previous iteration. In other word, we intend to select the same edges that have been selected in previous iteration, or equivalently, assign tasks to the same processors to which they have been assigned in the previous iteration. This strategy leads to improvement in the number of migrations and preemptions as well as voltage-level switches.
2. We postpone scheduling tasks and processors as late as their equivalent nodes become critical. In each iteration, critical nodes must participate in scheduling, or otherwise, corresponding tasks or processors to critical nodes cannot satisfy their timing constraints, namely constraints (5.1) and (5.3). By postponing task scheduling, idle time segments of processors in schedule period, which are used to schedule non-migratory tasks, will connect together, leading to minimum preemption and voltage-level switches for non-migratory tasks.

The first two lines of Algorithm 1 force the schedule to be composed exclusively from migratory task segments. In each iteration, prevX contains the edges from the previous iteration and nextXholds the current set of edges. Each edge in the sets indicates the mapping of one task from migratory task set to one processor from the processor set. At the beginning of each iteration, current full machines and urgent tasks is identified. Then, if there is no urgent task or no full machine, the task to processor assignment is postponed as late as possible. Otherwise, based on Algorithm 2, a task to processor assignment with the minimum possible changes (with respect to the immediate previous iteration) will be obtained.

After finding χ, for each $(T_i, M_j) \in \chi$, an arbitrary task segment TS_{ijl} is scheduled in the interval $[\gamma-\sigma,\gamma)$, i.e., task T_i is scheduled on machine M_j at voltage-level V_{jl} from γ – σ to γ. σ defines the interval length, and respective procedure to calculate it is discussed in Section 5.2.2. At the end of each iteration, we change the remaining range of scheduling by setting γ to γ – σ; decrement each t'$_{ijl}$ by σ for the scheduled task segments TS_{ijl}s; and replace the set of edges used in the previous iteration with the set of edges obtained in the current iteration to determine the matching χ in the next iteration. Iterations will be stopped when γ becomes zero. Based on this approach, we select TS_{ijl} which was scheduled in the immediate previous iteration. This strategy leads to less preemptions, migrations and voltage-level switches. Next section describes the implementation of the approach used to find the matching.

5.2.1. Finding the Tasks to Machines Assignment in Each Iteration

To find task to machine assignment (called matching χ in

Algorithm 1), in each iteration, we define a graph with m′vertices for full machines and the machines that are adjacent to urgent tasks and n′ vertices for the urgent tasks and adjacent tasks to full machines. We refer to the vertices corresponding to full machines as well as urgent tasks as critical vertices. In the graph, vertex of each task T_i is adjacent to vertex of machine M_j if there is a segment of a migratory task TS_{ijl} for some values of l while $t'_{ijl} > 0$. Therefore, matching functionχ corresponds to a minimum weighted bipartite matching which covers all critical vertices on the resulting bipartite graph by using as much edges as possible from previous iteration matching. Now, we describe the procedure to find this matching.

First, the adjacency matrix for migratory tasks and machines is initialized. Using the adjacency matrix, a bipartite graph is built which its vertices are full machines and machines which are adjacent to urgent tasks in one part and urgent tasks and adjacent tasks with full machines at the other part. Then, in Lines 13 and 14 from Algorithm 2, we add edges to set E if there is adjacency between the corresponding vertices. Then a network flow graph will be built from the bipartite graph. For this purpose, two additional nodes are inserted in the bipartite graph node set. The first node, namely the source node will be connected to all task nodes. Similarly, the second node, i.e. the sink node will be connected to all processor nodes. Positive supply value of 1 will be assigned to the nodes which fall in the category of urgent tasks (supply node).

Also, negative supply value of 1 is assigned to each full machine nodes (demand nodes). To keep our special minimum cost network flow model solvable, the sum of supply and demand flows must remain equal. In this regard, additional flow, which is equal to the difference of supply flows and demand flows, will be added to the source or sink nodes. All arcs have capacity of one except arcs between source node and urgent task nodes and arcs between full machine nodes and the sink node, which their capacity is set to 0. Arcs which are connected to source and sink have the cost of 1. The costs of other arcs are determined by the algorithm presented in figure 6.

This algorithm assigns cost a to arcs which are selected in the previous iteration and have the most importance. It assigns cost b to second most importance arcs, the arcs for which one of its adjacent nodes belongs to category of urgent tasks and the other is a full machine. The costs of all remaining arcs which have the least importance are set to c. Although the exact values of a, b, and c depend on the size of the problem, but the relation $a < b < c$ holds among them. These values should be selected in such a way that enables the matching to select as much as possible arcs with higher priority while finding the minimum total cost. Obviously, it is possible to leave some arcs with costs of a or b uncovered in the matching due to the problem constraints which prevent overlapped execution of difference segments of each task.

As shown in Algorithm 3, cost function is used for assigning edge cost and is computed according to the set of input parameters including E, U, F, and prev X. By solving minimum cost network flow problem, if flow of every urgent task and full machine nodes are zero, then it represents a matching that covers all urgent task and all full machine nodes. Otherwise, in each iteration, flow value of source node will be incremented by 1 and flow value of sink node will be decremented by 1. This loop will be terminated by finding a matching that covers all the full machines and urgent tasks. In such a matching, flow of every node equals to zero. Proof of the fact that such matching always exists and so the loop will be stopped eventually is an extension to what proposed in [32] and is described below.

Algorithm 2. Matching χ

1.	Matching χ :
2.	{
3.	Input: prevX
4.	Output: X // a set of edges
5.	Matrix $T_{n' \times m}$ // adjacency matrix where n′and m are the numbers of migratory tasks and processors
6.	If $\sum_{l=0}^{k} t'_{ijl} > 0$ then T(i, j)=1
7.	Else T(i, j)=0
8.	// create a bipartite graph
9.	VT = {V_{T_i} | $T_i$isurgenttaskoratask thatadjacenttoatleastonefullmachi
10.	VP = {V_{P_j} | $p_j$isfullmachineoraprocessorthatadjacenttoatleastoneur
11.	V = VT ∪ VP // a set of vertex
12.	E=∅ ; // a set of edge
13.	For $T_i \in U$
13.1.	For all P_j Processors
13.1.1.	IfT(i, j)=1 then $E = E \cup \{(V_{T_i}, V_{P_j})\}$ and Capacity $((V_{T_i}, V_{P_j}))=1$
14.	For $p_j \in F$
14.1.	For all T_i tasks
14.1.1.	If T(i, j)=1 then $E = E \cup \{(V_{T_i}, V_{P_j})\}$ and Capacity$((V_{T_i}, V_{P_j}))=1$
15.	// convert bipartite graph to network flow
16.	$V = V \cup \{V_S\} \cup \{V_d\}$; // add source and sink nodes to graph
17.	For all V_{T_i} : $E = E \cup \{(V_S, V_{T_i})\}$
17.1.	If $(V_{T_i} \notin U)$ Capacity$(V_S, V_{T_i}) = 1$
18.	For all V_{P_i} : $E = E \cup \{(V_{P_i}, V_d)\}$
18.1.	If $(V_{P_i} \notin F)$ Capacity$(V_{P_i}, V_d) = 1$
19.	// Edge cost definition
20.	Assign edge cost // cost function in Algorithm 3
21.	IF$(V_{T_i} \in U)$ Flow$[V_{T_i}]$=1
22.	IF$(V_{P_i} \in F)$ Flow$[V_{P_i}]$=-1
23.	IF(|U|> |F|) Flow$[V_d]$=|F|-|U|
24.	Else Flow$[V_S]$=|F|-|U|
25.	$E' = \emptyset$; // a set of edge
26.	while (true)
26.1.	$E' = Solution(Flow, V, E, C)$ // Solution function returns set of edges of the minimum cost network flow
26.2.	If ($\forall t_i \in U$, exist one edge $e=(V_{T_i}, V_{P_j}) \in E'$) And ($\forall p_j \in F$, exist one edge $e=(V_{T_i}, V_{P_j}) \in E'$)
26.2.1.	X={ (V_{T_i}, V_{P_j}) | $(V_{T_i}, V_{P_j}) \in E'$}
26.2.2	Return X
26.3	Else
26.3.1.	Flow$[V_S]$++
26.3.2.	Flow$[V_d]$--
27.	}

Algorithm 3. Cost Function

1.	Cost Function :
2.	Input : E, U, F, prevX
3.	Output: C[ ] // The matrix C indicates the cost of edges
4.	For each $(V_{T_i}, V_{P_j}) \in E$
4.1.	If $(V_{T_i}, V_{P_j}) \in$ prevX then $C(V_{T_i}, V_{P_j}) = a$
4.2.	Else if $(T_i \in U)$ and $(P_j \in F)$ then $C(V_{T_i}, V_{P_j}) = b$
4.3.	Else if $(T_i \in U)$ then $C(V_{T_i}, V_{P_j}) = c$
4.4	Else if $(P_j \in F)$ then $C(V_{T_i}, V_{P_j}) = c$

As we know, a matching which covers all critical vertices can be obtained through the following steps:

- Find a matching from all urgent tasks to machines, namely a bijection $\chi_T: U \to R_U, R_U \subseteq M$, using standard

matching algorithms for bipartite graphs,

- Find a matching from all full machines to tasks, namely a bijection $\chi_M: F \rightarrow R_F, R_F \subseteq T$, using standard matching algorithms for bipartite graphs,
- Mixing χ_T and χ_M using the algorithm shown in Algorithm 2 (known as Algorithm 2) to obtain a matching that covers all critical vertices.

Algorithm 4. Finding χ Using χ_T and χ_M

1.	$\chi = \phi$; // Domain(χ) = Range(χ) = ϕ
2.	For each $T_i \in U - Domain(\chi) - Range(\chi_M)$
2.1.	Add $(T_i, \chi_T(T_i))$ to χ
2.2.	Remove $(\chi_T(T_i), X)$ from χ_M, if there exists
3.	For each $(X,Y) \in \chi_M$
3.1.	Add (Y, X) to χ

It can be shown that for each arbitrary $S \subseteq U$, $|N(S)| \geq |S|$, where N(S) is the set of machines, which their corresponding vertices are neighbors of vertices of S in the bipartite graph. Thus, Hall's condition [45] holds and therefore χ_T and χ_M exist. In order to prove that $|N(S)| \geq |S|$, we consider the set of all the remaining parts of task segments TS_{ijl}, i.e. t'_{ijl}, so that $T_i \in S$. Since each $T_i \in S$ is urgent, we can conclude:

$$|S| \times \gamma = \sum_{T_i \in S} (\sum_{M_j \in N(S)} \sum_{l=1}^{k_j} t'_{ijl}) \quad (8)$$

Also, it is trivial that

$$\sum_{T_i \in S} (\sum_{M_j \in N(S)} \sum_{l=1}^{k_j} t'_{ijl}) \leq \sum_{M_j \in N(S)} (\sum_{T_i \in T} \sum_{l=1}^{k_j} t'_{ijl}) \quad (9)$$

Considering that no machine can execute tasks with total execution times greater than γ, we can conclude:

$$\sum_{M_j \in N(S)} (\sum_{T_i \in N(N(S))} \sum_{l=1}^{k_j} t'_{ijl}) \leq |N(S)| \times \gamma \quad (10)$$

From (8), (9), and (10), we find that $|N(S)| \geq |S|$.

To mix χ_T and χ_M, in each iteration of Line 2 of Algorithm 4, an urgent task and its paired machine in χ_T, namely $(T_i, \chi_T(T_i))$, is added to χ, and therefore, T_i is added to Domain(χ) and a task will be removed from U–Domain(χ) –Range(χ_M) for the next iteration of the algorithm. Moreover, considering a task X, removing $(\chi_T(T_i), X)$ from χ_M, if it exists, removes X from Range(χ_M). If X is an urgent task, anew member is added to U–Domain(χ)–Range(χ_M) for the next iteration. This ensures that we do not ignore the urgent task X and thus, this task will be checked in further iterations.

In Line 3 of Algorithm 4, full machine vertices are covered either in χ or in χ_M. It can be described as follows: considering a machine X, if pair (X, Y), where Y is a task, is removed from χ_M, a pair (Z, X), where Z is also a task, is added to χ. Furthermore, no urgent task vertex remains uncovered in χ unless it has been covered in χ_M owing to the condition of $T_i \in U - Domain(\chi) - Range(\chi_M)$ in line 1 of Algorithm 2. Therefore, through adding task-machine pairs of χ_M to χ, we can claim that χ covers all full machine as well as all urgent task vertices.

For clarifying the matching algorithm, figure 4 displays snapshots of execution of matching for Example 1 at the end of fifth and sixth iterations. Red nodes represent critical nodes. Each edge is labeled 'X/Y' where X and Y denote the capacity and cost of the edge, respectively. Edges which are chosen to carry flow are drawn in red. The set of red edges between machines and tasks, which are passed across an ellipse as well, compose our desired matching.

As it has shown in figure 4, after selecting the edges (T_1, P_2), (T_3, P_4), (T_6, P_1) as a final solution in iteration 5, the cost of edges (T_3, P_4), (T_6, P_1) has decreased to increase the probability of their selection in the current iteration. It should be noted that selecting the edges which were selected in the previous iteration leads to lower number of preemptions, migrations and voltage-level switches.

5.2.2. Finding the Value of δ

The remaining part of the scheduling for an iteration of Algorithm 1 is to specify the length of time in which this part of scheduling is done. Based on the work in [32], value of σ is the largest value satisfying the following three conditions:

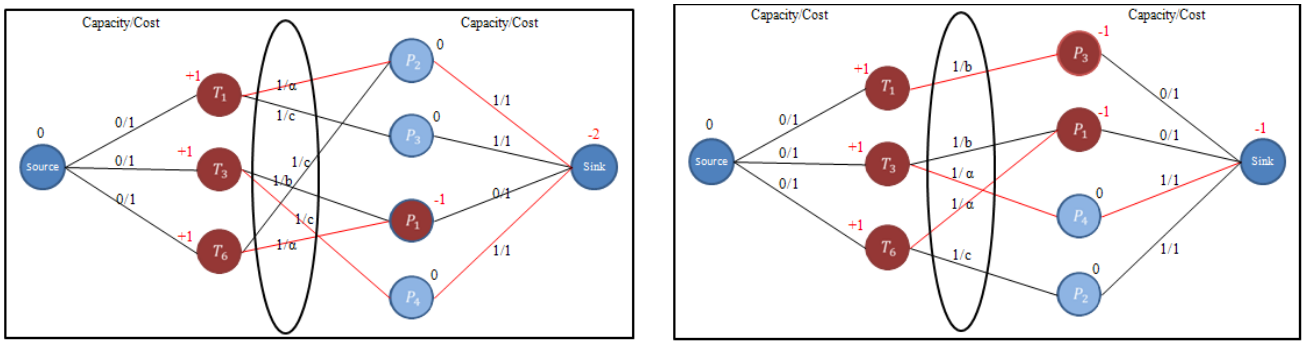
1. For each assigned task segment TS_{ijl}, we need to have:

$$\sigma \leq t'_{ijl} \quad (11)$$

This ensures that T_i has enough work to be done on M_j at voltage-level V_{jl} for σ amounts of time in $(\gamma - \sigma, \gamma)$ and

$$t'_{ijl} \geq 0 \quad (12)$$

holds for all task segments in the next iteration.



(a) Iteration 5

(b) Iteration 6

Figure 4. Snapshots of matching algorithm (iterations 5 and 6 from example 1)

2. For each migratory task T_i that have not been mapped, we need

$$\sigma \le \gamma - \sum_{j=1}^{m} \sum_{l=1}^{k_j} t'_{ijl} \quad (13)$$

This condition ensures that T_i remains non-urgent throughout interval $(\gamma - \sigma, \gamma)$ and

$$\sum_{j=1}^{m} \sum_{l=1}^{k_j} t'_{ijl} \le \gamma \quad (14)$$

holds for T_i in the next iteration, i.e., the required amounts of time for executing the remaining parts of task segments of T_i are available.

3. For each machine M_j which have not been allotted, we need

$$\sigma \le \gamma - \sum_{i=1}^{n} \sum_{l=1}^{k_j} t'_{ijl} \quad (15)$$

This ensures that M_j remains non-full throughout interval $(\gamma - \sigma, \gamma)$ and

$$\sum_{i=1}^{n} \sum_{l=1}^{k_j} t'_{ijl} \le \gamma \quad (16)$$

holds for M_j in the next iteration, i.e., in the iteration that the required amounts of time for executing the remaining parts of segments of migratory tasks on M_j are available.

In other word, scaled tasks can be scheduled on the allocated machines unless one of the following Types of Changes occurs in the status of the system:

- Type A, shown as A(TS_{ijl}): A task segment TS_{ijl} is assigned to the proper machine at the proper voltage-level for the length of t'_{ijl},
- Type B, shown as B(M_j): A machine M_j becomes full, which is equivalent to adding a critical vertex corresponding to the machine in the bipartite graph,
- Type C, depicted as C(T_i): A task T_i becomes urgent, which is equivalent to adding a critical vertex corresponding to the task in the bipartite graph.

At the end of each iteration of Algorithm 1, at least one type of the changes occurs. Thus, we need to find a new set of task segments to be scheduled for the next iteration.

It should be noted that when a change of Type A (e.g., A (TS_{ijl})) occurs as the sole change at the end of an iteration, if there is a task segment $TS_{ijl'}$ the previous matching can be used with only a voltage-level switching, e.g. from V_{jl} to $V_{jl'}$, for the new iteration. Doing so, we can effectively reduce the number of preemptions. However, if there is no other task segment $TS_{ijl'}$, an edge is removed from the bipartite graph and we need to find a new matching.

When t'_{ijl} is decremented by the amount of σ for each assigned task segment TS_{ijl}, it can be concluded for each migratory task T_i that

$$\sum_{j=1}^{m} \sum_{l=1}^{k_j} S_{ijl} \times t'_{ijl} = e'_i \quad (17)$$

where e'_i is the remaining execution requirements of task T_i for the next iteration. In addition, with this reduction in the values of t'_{ijl}s, it is trivial that (14) also holds for urgent tasks and (16) holds for full machines. In fact, (12), (14), (16), and (17) are corresponding to Constraints (5.4), (5.3), (5.2) and (5.1), respectively. This suggests that, in each iteration, it is ensured that all corresponding constraints to primitive constrains on migratory tasks and machines are held. Since at least a Change of Type A, B or C occurs at the end of each iteration, Algorithm-I terminates because of finiteness of the number of thesechanges. Next,in the last iteration, the remaining execution requirements of each migratory tasks gets zero. The reason is that, according to (12), (14) and (16), all t'_{ijl}s eventually turn into zero. Thus, it is verified that Algorithm 1 schedules the segments of migratory tasks in a proper manner.

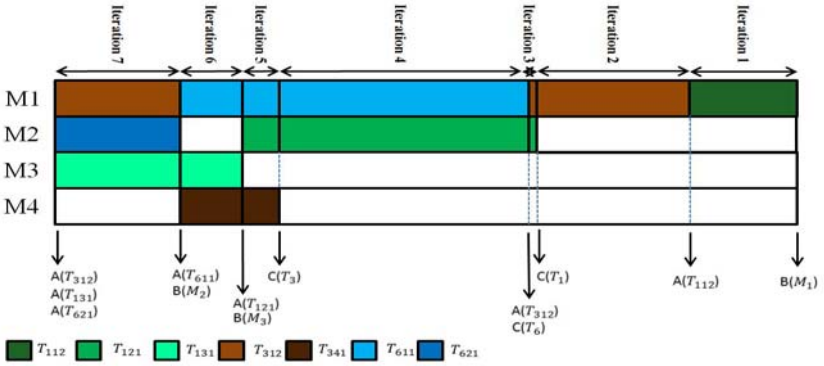


Figure 5. Assigning migratory tasks in the boundaries of a SP

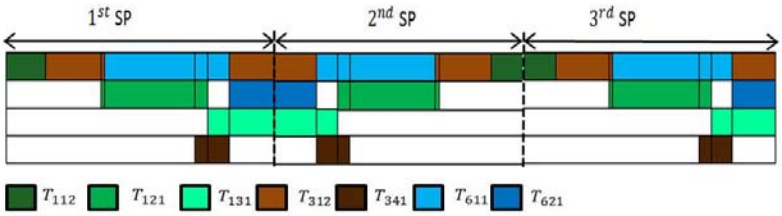


Figure 6. Migratory task segment placement

Example 2. After obtaining the task segments in Example 1, here figure 5 shows how segments of migratory tasks are assigned to the parallel machines in the boundaries of a SP. The change types occurring at the end of iterations are also shown.

5.2.3. Schedule Period Placement

In the next step, we schedule migratory tasks in entire hyper period. For further improvement in the number of migration and preemption and voltage-level switches, algorithm 5 as shown in Algorithm 5 is proposed.

Algorithm 5. Schedule Period Placement

```
1.      SP_U: Schedule set for a SP with unit length
2.      for i = 1 to N  // N=number of all SPs in hyperperiod
2.1.    if (i is Odd)
2.1.1.  SP'_U : mirror(SP_U)
2.1.2.      Scale SP_U to SP_i
2.2     else
2.2.1.      Scale SP_U to SP_i
```

In algorithm 5, SP_U is a data structure that holds the scheduling of migratory tasks. Each SP_U object specifies a processor, an execution interval, and a voltage-level for a single task. For any schedule period, the proportionally scaled unit SP is repeated. Function mirror (SP) will reverse the scheduling order for the given SP and is applied to schedule periods of odd index which merges free spaces of adjacent SPs while decreases the number of preemptions and voltage-level switches for non-migratory tasks. It also makes the migratory tasks at the end of SP to repeat in the start of next SP which causes reducing the number of migrations, preemptions and voltage-level switches. A similar technique is used in [46] to decrease the number of preemptions.

Example 3: Figure 6 shows the placement of migratory task segments to the parallel machines after applying the mirror procedure on the schedule of Example 2.

Briefly, we use the scaled version of the obtained schedule of a SP of unit length by the factor of $|SP_k|$ beside its mirror to schedule migratory tasks in SP_k, and thus, achieve a feasible scheduling for the migratory tasks in a hyper period. In this regard, each machine will have some idle intervals in which some non-migratory tasks can be scheduled. In the next subsection, we describe the method of scheduling segments of non-migratory tasks within these idle intervals. As a matter of fact, it is not vital to schedule migratory and non-migratory tasks through different algorithms, or even in different phases like the approach discussed in [32]. We could use Algorithm 1 to schedule both migratory and non-migratory tasks simultaneously. Such possible scheduling suffers from high overhead; hence, in order to reduce the number of preemptions as well as voltage-level switches, we schedule segments of non-migratory tasks separately after all migratory task segments have been scheduled. More details are presented in the following section.

5.3. Scheduling Non-Migratory Tasks

The main point about scheduling non-migratory tasks is that we are not concerned about the parallel execution of job slices of any such task. In addition, each task segment TS_{ijl} of non-migratory tasks can be executed in SP_k during the machine idle periods for $t_{ijl} \times |SP_k|$ after scheduling migratory tasks, as the LP constrains (Constraint (5.2)). By scheduling in this way, each task token would meet its fluid path at the end of each SP, and therefore, no non-migratory task misses its deadline, i.e. there is a feasible scheduling for them in idle times of each machine. However, in order to reduce the number of preemptions and voltage-level switches, we use a different algorithm for this part. As a result from [53], algorithms with work-conserving property have the advantage to avoid unnecessary task preemptions. Hence, we consider this property for scheduling of non–migratory tasks that significantly decreases the number of preemptions and voltage-level switches. Corresponding to each segment TS_{ijl} of non-migratory tasks, we define a release-based-task-segment ($RBTS_{ijl}$) with size τ_{ijl} which indicates the amount of time that task T_i should be executed on machine M_j at voltage-level V_{jl} in the task period of length p_i, where $\tau_{ijl} = t_{ijl} \times p_i$. We schedule RBTSs per task period instead of TSs per SP. Also, we use EDF to schedule the RBTSs (i.e., for non-migratory tasks) in the idle times throughout each hyper period. Of course, with employing scheduling algorithms with lower preemption overheads than EDF (e.g., EDZL), it is possible to decrease the number of preemptions even more. The sum of execution requirements of all RBTSs of a task T_i ($RBTS_{ijl}$) in a period p_i is equal to the exact execution requirements of task T_i in the period, i.e. e_i. According to Constraint (5.1), we have

$$\sum_{j=1}^{m} \sum_{l=1}^{k_j} S_{ijl} \times \tau_{ijl} = e_i \quad (18)$$

Thus, even if the scheduling strategy is changed, it will not cause any deadline to be missed while the job slices of a certain task instance are executed within their respective period (i.e., their relative deadline). Furthermore, according to the optimality of the EDF algorithm [23], it is straightforward to show that EDF gives a feasible scheduling in the idle times of each machine, if such a schedule exists. In this regard, as the existence of a feasible schedule for the task set is verified and confirmed in Step 1 of the scheduling (i.e., through the LP), it would be sufficient to use the EDF algorithm while considering each $RBTS_{ijl}$ as a task with execution requirement τ_{ijl} and period p_i. In this way, all the RBTSs, and equivalently, all the non-migratory tasks are scheduled. It is worthwhile to note how this method of scheduling results in fewer preemptions and voltage-level switches: in the case where two RBTSs, namely $RBTS_{ijl}$ and $RBTS_{ijl'}$ exist for a certain task T_i and no new task with an earlier deadline arrives before the RBTSs deadline, EDF selects $RBTS_{ijl'}$ (which has the same deadline as $RBTS_{ijl}$) for execution instead of random selection among the other admissible tasks. In this regard, one voltage-level switch occurs instead of a preemption.

Example 4. As shown in figure 7, RBTSs corresponding to segments of non-migratory tasks of Example 1 which listed in table 1, are scheduled after scheduling segments of migratory tasks in Example 2. Migratory tasks are shown in gray color, while RBTSs are represented in other colors.

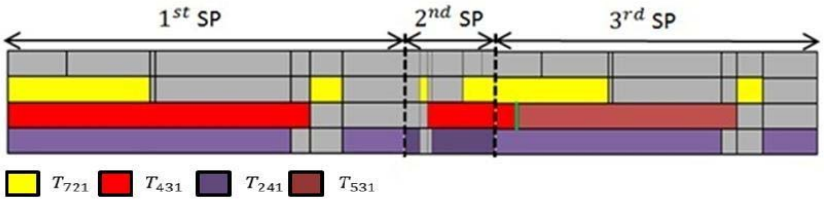


Figure 7. Complete scheduling schema over SPs

Theorem 2. The Linear Programming (LP) problem mentioned in Step 1 has a solution with minimal value of E if and only if the set of unrelated parallel machines has a feasible scheduling with minimal average energy usage E.

Proof: It is trivial to show that if there is a scheduling for a task set, the constraints of the LP would be met; therefore, the LP has a solution with minimal average energy usage E if there is a feasible scheduling. By Corollary 1, it is inferred that minimal average energy usage for scheduling the task set in a hyper period ($\overline{E(H)}_{\min}$), and therefore the system lifetime, is also the minimal average energy usage for scheduling the scaled task set in a SP of unit length, which is the minimum value of $\overline{E}$ in the LP.

To prove the other direction, consider a solution to the LP. We described how we can use this solution to construct a schedule for the tasks upon the machines, such that all deadlines are met. Specifically, for segments of migratory tasks, we will construct a schedule over the interval $[0,1)$. By Theorem 1, a schedule in which all task instance deadlines are met may be obtained by repeating the scaled versions of this schedule over all SPs. Then we have scheduled non-migratory tasks in the remained idle times. It is mentioned that non-migratory tasks can be scheduled by the EDF algorithm without missing a deadline.

Scheduling in this manner results in executing each task segment TS_{ijl} (segments of both non-migratory and migratory tasks) obtained from the LP, for $t_{ijl} \times H$ amounts of time during a hyper period. Therefore, energy usage of the scheduling is $\overline{E} \times H$ in a hyper period of length H, which means the average energy usage of the scheduling in that period, is equal to $\overline{E}$. Furthermore, based on Corollary 1, average energy usage of the system could not be less than the minimum average energy usage of the scaled task set in a SP of unit length. Thus, this algorithm gives a scheduling with the minimum energy usage if the system has a feasible solution.

6. Discussion

In this section, we first determine the number of migratory tasks based on some properties of the linear programming, and then, we specify an upper bound on the number of preemptions and voltage-level switches.

6.1. Linear Programming and the Degree of Migration

The cost of migration might be inevitable in some distributed systems since the whole task state must be migrated as well. However, the proposed algorithm schedules most tasks on only one processor, and bounds the number of migratory tasks. The total number of edges in the bipartite graph at the first iteration of Algorithm 1 is a measure to find out how much the schedule is global rather than partitioned (degree of migration); a schedule in which the number of segments of migratory tasks is equal to zero is a purely partitioned schedule [32]. This measure can determine the degree of migration as discussed in the following.

As shown in [47] and [48], there are polynomial-time algorithms for solving LPs as well as polynomial-time algorithms (e.g., see [49]) for obtaining a basic solution, given a non-basic optimal solution to a LP problem. Therefore, a basic solution can be obtained in polynomial-time. Considering a LP with n variables, I inequality constraints and E equality constraints, the basic solution to the LP problem lies at a vertex point in which n–E inequalities satisfy as equalities.

Returning back to our problem, we have

$$N = (n+1) \times \sum_{i=1}^{m} k_i \tag{19}$$

variables and inequalities (Constraint (5.4)) in the LP. Also, we have n and m equalities for Constraints (5.1) and (5.2), respectively, and, n inequalities for Constraint (5.3). However, according to the m equalities for Constraint (5.2), we can conclude

$$\sum_{j=1}^{m} (\sum_{i=1}^{n} \sum_{l=1}^{k_i} t_{ijl}) \le m \tag{20}$$

and thus,

$$\sum_{i=1}^{n} (\sum_{j=1}^{m} \sum_{l=1}^{k_i} t_{ijl}) \le m \tag{21}$$

Consequently, at most m number of the tasks can be urgent (satisfy inequalities of Constraint (5.3) with equality). According to the mentioned property of LP, N–n–m inequalities would be marginally satisfied. As shown above, at most m inequalities for Constraint (5.3) can satisfy as equalities. Thus at least N–n–2m of inequalities of Constraint (5.4) is held in the form of equalities.

This means that there are at most n+2m non-zero variables at the optimal point. On the other hand, because each task T_i ($i > 0$) must have at least one non-zero variable t_{ijl} for some values of j and l, we can conclude that n variables of n+2m non-zero variables belong to different tasks; therefore, at most 2m extra task segments exists. These extra segments can result in at most 2m migratory tasks. Also, there are at most 2m+2m=4m segments of migratory tasks.

6.2. Number of Preemptions, Migrations and Voltage-Level Switches

Regarding the issue that there is no need for voltage-level switches when a machine is idle (as described in Section 5), the upper bound on the number of migrations, preemptions and voltage-level switches can be obtained as follows. The number of migrations and voltage-level switches for migratory tasks will be at most equal to the number of iterations of Algorithm 1 in each machine; because there are no interruptions in the interval of iteration and the task can run continuously. The number of iterations of Algorithm 1 is at most equal to the number of times that one of the change types A, B, or C occur. There are at most 6m changes, namely 4m changes of Type A, which is less than the number of task segments belonging to the migratory tasks, m changes of Type B, which is less than the number of full machines, and m changes of Type C, which is less than the number of urgent tasks.

Therefore, 6m is an upper bound on the number of migrations and voltage-level switches in a SP for migratory tasks on each machine, and thus, $6m^2$ is an upper bound for the entire system model. On the other hand, using the EDF algorithm, non-migratory tasks will be preempted only when a change of Types A, B or C occurs or when a new RBTS arrives or its execution finishes, considering RBTSs as tasks. We refer to these two latter cases as Change Type D (times in which changes of Type D occur, are shown in red color in Algorithm 4). The number of SPs in a hyper period, which is at most equal to the number of task releases, is

$$\sum_{i=1}^{n} \frac{H}{p_i} \tag{22}$$

Therefore, the total number of preemptions for all machines will be equal to the number of Type A, B and C changes, which is

$$(6m^2) \times \sum_{i=1}^{n} \frac{H}{p_i} \tag{23}$$

plus the number of Type D changes that is

$$2 \times \sum_{RBTS_{ijl}} \frac{H}{p_i} \tag{24}$$

Furthermore, since there are at most 2m extra task segments (as mentioned above), the number of RBTSs is not more than n + 2m, and at least one RBTS for each non-migratory task exists. This means that:

$$\sum_{RBTS_{ijl}} \frac{H}{p_i} \le \sum_{i=1}^{n} \frac{H}{p_i} + 2m \frac{H}{\min(p_i)} \le (2m+1) \sum_{i=1}^{n} \frac{H}{p_i} \tag{25}$$

Thus, the number of preemptions and voltage-level switches in a SP using the proposed algorithm is $2 \times (2m+1) + 6m^2 = O(m^2)$ in the average case. This is independent of n and much fewer than the preemptions in the available online algorithms, even for identical platforms where $n \gg m$ (e.g. [14]).

7. Experimental Results

This section presents the evaluation results of the proposed algorithm for synthetic task sets on different configurations of parallel machines. Since, to the best of our knowledge, there is no optimal scheduling algorithm for unrelated parallel machines that support task migration, we compare EORTSA algorithm with PCG [25] which is introduced for real-time scheduling on uniform parallel machines.

Most of the optimal scheduling algorithms for heterogeneous multiprocessors are based on fairness [25, 32, 50], so the order of the number of preemptions and migrations of these algorithms is almost the same. These algorithms proportionally assign processors to the tasks according to the utilization rates. Among them, PCG is an optimal scheduling algorithm which is insensitive to the energy usage. PCG has the least number of migrations and preemptions in the class of optimal scheduling algorithms in uniform parallel machines which make it a good choice to compare with preemptions and migrations of EORTSA. On the other hand, since EORTSA scheduling algorithm is optimal regarding the energy usage, we have omitted the comparison of EORTSA with other existing algorithms which intend to reduce the energy.

Although the proposed algorithm supports all types of machines, it was restricted to uniform parallel machines in experiments. We have considered four uniform parallel machines with 2, 4, 8 and 16 XScale PX270 processor cores. Since this processor supports multiple DVS voltage-levels, it is possible to create a uniform parallel machine consisting of cores with different execution speed by assigning different voltage-levels to cores accordingly. We have used data from table 6 to create such parallel machine in which every core works at a fixed random voltage and corresponding frequency. Also, it can be observed from table 6, there is a linear dependence between the speed and the power, meaning that by doubling the speed, power consumption will increase approximately twice.

With such restriction, it is hard to reveal the potentials of our algorithm in reducing power consumption during the evaluations. Our algorithm achieves much more energy saving for unrelated platforms rather than uniform ones, as power consumption relates to execution speed through a nonlinear function and is task-dependant. To reflect the nonlinear relation between power consumption and execution speed, which is most common in unrelated platforms, a custom uniform platform with nonlinear power-speed relation is modeled using the following assumptions:

- Tasks are assumed to have the same functionality but with different input sets,
- Task execution times are calculated for different processors and are normalized based on the execution speed of the slowest processor,
- Task execution time on different processors is based on E3S benchmark suite [51].

Since we only have one type of the task on the system, this unrelated platform can be considered as a uniform platform. Now, PCG algorithm can schedule the tasks in this new platform. Tables 6 and 7 report the average powers and relative speeds of some processor cores for different operating frequencies. Although actual power and relative

speed values vary depending on the tasks, these values can be used as a guideline for our constructed platform.

Linear program and minimum weighted bipartite matching problem (via minimum cost network flow) are modelled and solved in ILOG Cplex solver [52].

Table 5. Specifications of the Intel XScale PXA270 processor [7] for different speeds

Speed (MHz)	104	208	312	416	520	624
Normalized speed	0.5	1	1.5	2	2.5	3
Active power (watts)	0.116	0.279	0.390	0.570	0.747	0.925
Idle power (watts)	0.064	0.129	0.154	0.186	0.222	0.260

Table 6. Speed and power of executing text processing benchmark on different processors according to E3S benchmark suites [47]

Processor model	Speed	Active power (watts)	Idle power (watts)
AMD ElanSC520-133 MHz	3.0	1.6	0.16
AMD K6-2E 400MHz/ACR	1	10	1
AMD K6-2E+ 500MHz/ACR	1.75	14	1.4
AMD K6-IIIE+ 550MHz/ACR	2	16	1.6
IBM PowerPC 405GP - 266 MHz	1.75	2	0.2
NEC VR5432 - 167 MHz	0.16	2.5	0.25

We conducted several different experiments to evaluate the average power, number of preemptions and number of migrations of the above-mentioned scheduling algorithms.

As stated before, the evaluation has been performed on synthetic task sets. These tasks have been generated randomly in the following manner. First the utilization value of each task has been generated randomly such that the system utilization reaches to a target value determined in the experiment setup. In order to avoid unaffordable hyper period length in simulation, we use the following approach for generating of task periods. First an integer number which has at least 150 divisors is randomly selected as the hyper period value. After that, the task period will be chosen from the set of those divisors. In this way, we have restricted the periods to integer numbers. And finally the execution time is calculated using $e_i = p_i \times u_i$.

In each experiment, system utilization has been considered in the range of 10% to 100% (with step 10%). Each experiment setup has been repeated 100 times. Then, the impact of utilization on the performance metrics has been observed. These performance measures are Average Number of Migrations and Preemptions per job (ANMP) and the average energy usage as reported in the other researches in the literature.

Figure 8 shows ANMP as the vertical axis according to different system utilizations (horizontal axis). Also in this figure, the comparison of the number of preemptions and migrations for different task set sizes (10, 20, 30 and 40) has been reported. In this figure, when there are 10 tasks in the system and the utilization is higher than or equal to 60%, PCG algorithm has less preemptions and migrations than EORTSA (it is about 33% of EORTSA). In fact, since EORTSA aims at minimizing the energy, it may lead to higher ANMP in comparison with non-energy minimizing and general scheduling algorithms. Consequently, in the loads near 100% in which the opportunity of saving energy is limited, EORTSA is less effective than non-energy-minimizing scheduling algorithms such as PCG. However, when the number of tasks becomes larger (e.g., 20, 30 or 40), ANMP of EORTSA decreases. The reason is that with larger number of tasks, each task most likely has smaller portion in the total utilization of the system so the chance that it is preempted or migrated becomes slighter. Consequently, the performance of EORTSA improves when there are a large number of tasks in the system.

Another important feature of EORTSA is that it keeps the number of preemptions and migrations under a fixed bound, i.e. $O(m^2)$, that only depends on the number of processors in the system. Figure 8 reveals this fact especially when the number of tasks increases. In a similar fashion, PCG has considerable amounts of ANMP (see figure 8(d)). Because it makes proportional assignments on each schedule period, in the higher task counts, it encounters many task switches. Besides, EORTSA schedules RBTSs per task period instead of task segments per schedule period for non-migratory tasks and always preserves the number of migratory tasks under twice the number of processors (2m). Figure 9 shows comparison of power consumption of scheduled task set for PCG and our algorithm.

Figure 9 (a) presents power consumption for 4 processors for different system utilizations. EORTSA consumes less power at different utilizations than PCG. However, when the utilization is 100%, there will be no difference between the power consumption of PCG and EORTSA (because all processors are always busy at that utilization). For EORTSA, the maximum power saving in comparison with PCG is achieved when the utilization is around 30% to 70% (see figure 9(a)) which was about 60% for utilization 30% and 40%.

In the next experiment (Figure 10), we have considered different number of processors (2, 8 and 16), and for each setup, two task set sizes have been considered. Similar to our discussion for the 4-processor setup, by increasing the number of tasks, EORTSA outperforms PCG in ANMP. Figure 9(b) shows the average power consumption of each experiment for 2, 4, 8 and 16 processors. With the increase of the number of processors, the gap between EORTSA and PCG also increase because when EORTSA finds more opportunities to distribute tasks over the processors, it might find better options to enhance power consumption of the tasks.

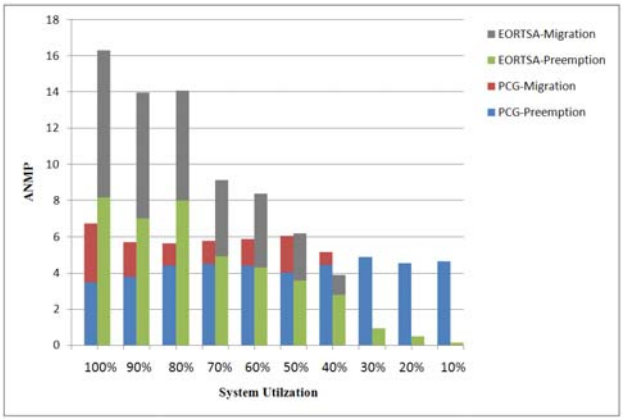
Further, a version of PCG algorithm for unrelated platform which is aware of different execution speed of tasks on different processors has been implemented. The modified PCG has lower schedule ability bound which is depending on the underlying platform configuration. The reason of non-optimal assignment of tasks to processors in this extension of PCG is its greedy assignment which is based on local view of the system.

8. Conclusions and Future Works

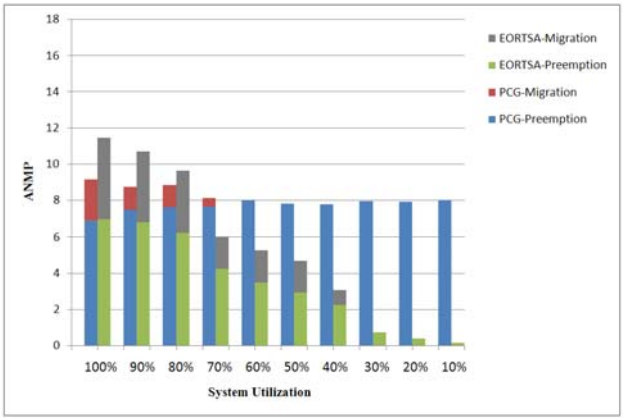
In this paper, we have studied the problem of scheduling a set of periodic real-time tasks on unrelated parallel machines among which task migration is permitted. Each processor (machine) in the system has a few discrete speed levels and

can employ DVS for voltage and speed scaling. At each voltage-level, the processor has a maximum speed as well as a maximum power usage. Further, there exist task-processor specific speeds which are fractions of the maximum speeds as well as task-processor specific power usages which are fractions of the mentioned maximum power usages. A polynomial-time optimal scheduling algorithm is proposed which feasibly schedules the set of tasks on the machines while minimizes the overall system energy usage. We introduce the first part of the algorithm as a schedule ability test for unrelated platforms. Also, with multiple improvements the number of task migrations and preemptions and voltage-level switches are reduced to avoid extra overheads. It is mentioned that the number of migratory tasks is at most 2m, where m is the number of processors. Also, it is shown that the total number of migrations, preemptions and voltage-level switches in each schedule period is in the order of $O(m^2)$.

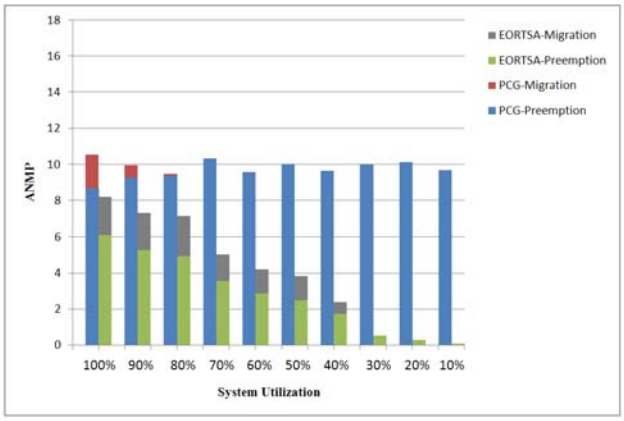
As an idea for the further work, we intend to extend our energy model to include leakage power and consider overheads of migration, preemption, and voltage level switches in terms of energy.



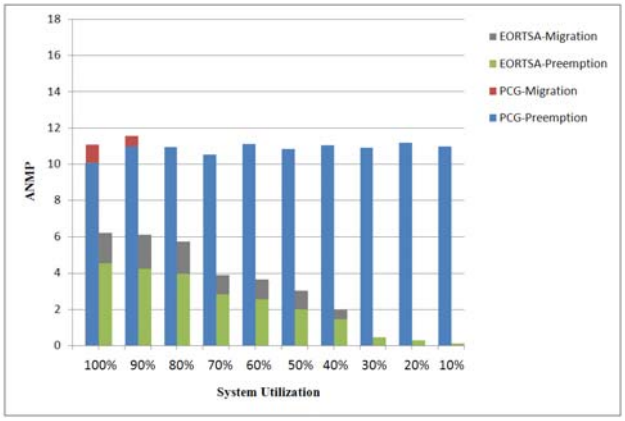
(a) n=10, m=4



(b) n=20, m=4

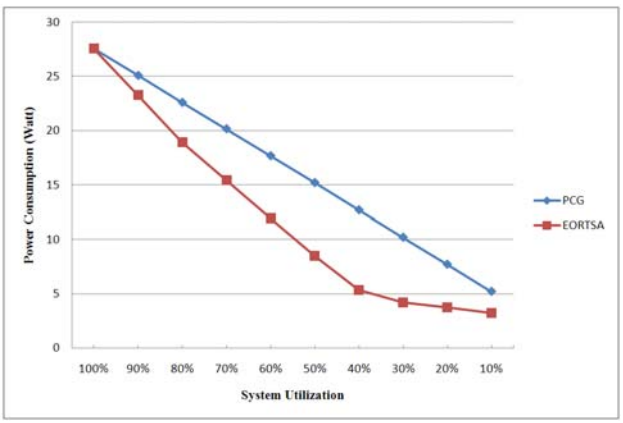


(c) n=30, m=4

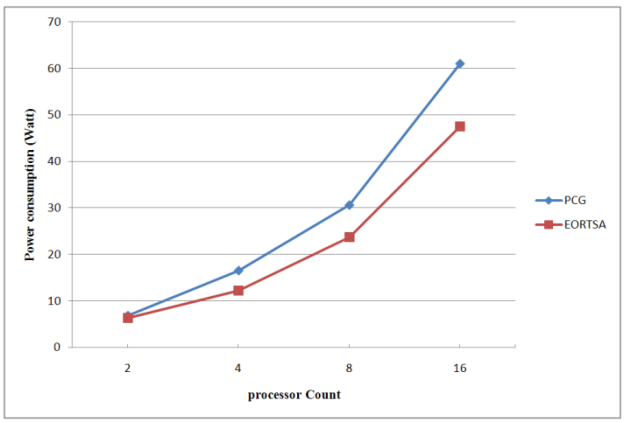


(d) n=40, m=4

Figure 8. Average Number of preemptions and migrate onsper job for 4 processors

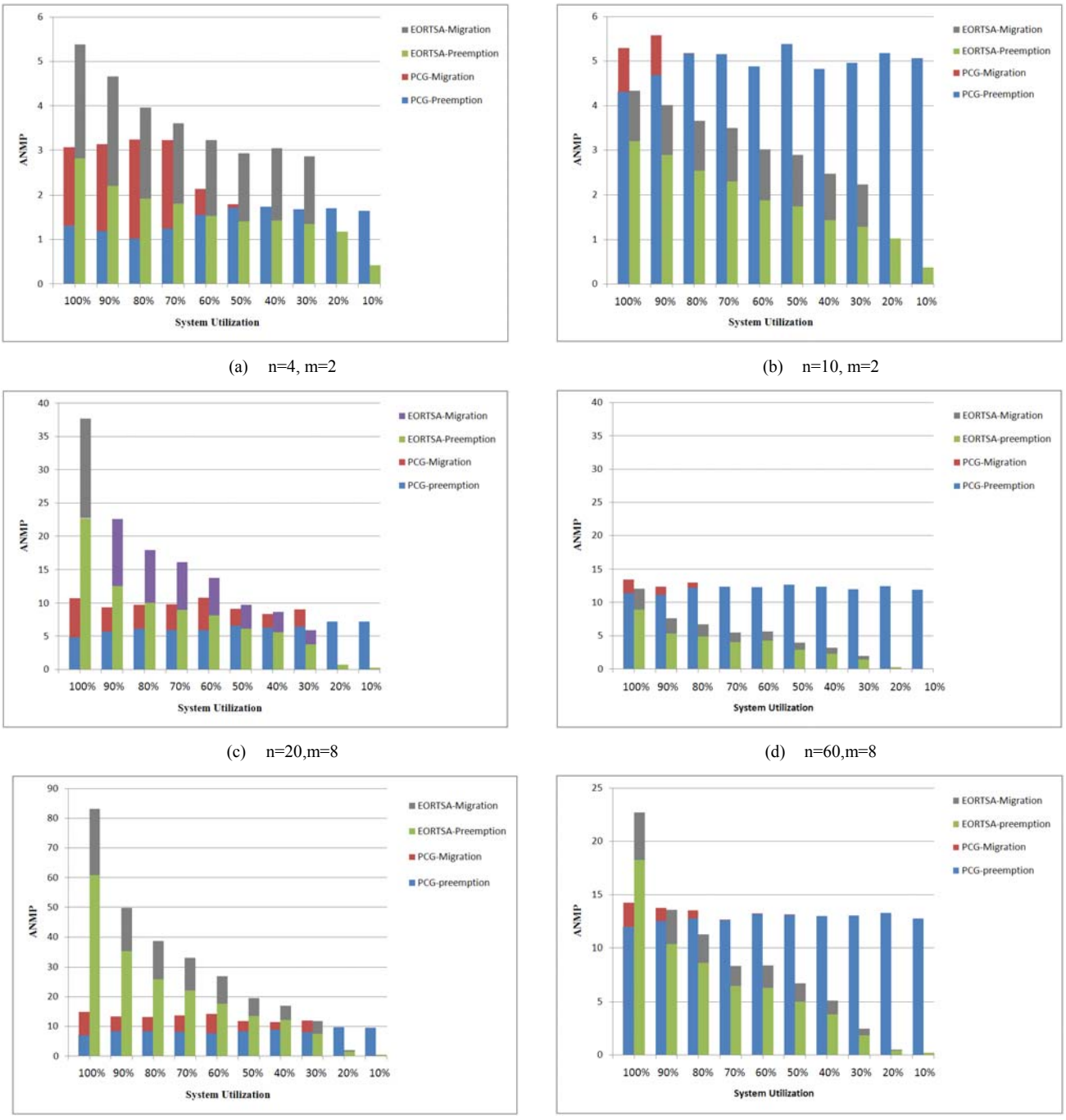


(a) For 4 Processors



(b) Average of all of utilization

Figure 9. Power consumption



(a) n=4, m=2

(b) n=10, m=2

(c) n=20,m=8

(d) n=60,m=8

(e) n=40,m=16

(f) n=120,m=16

Figure 10. Number of preemptions and migrations per job for different processor and task count

References

[1] K. K. Rangan, M. D. Powell, G.-Y. Wei, and D. Brooks, Achieving uniform performance and maximizing throughput in the presence of heterogeneity, in: High Performance Computer Architecture (HPCA), 2011 IEEE 17th International Symposium on, pp. 3-14, 2011.

[2] U. R. Karpuzcu, B. Greskamp, and J. Torrellas, The BubbleWrap many-core: popping cores for sequential acceleration, in: 42nd Annual IEEE/ACM International Symposium on Microarchitecture, pp. 447-458, 2009.

[3] E. Bini, G. Buttazzo, and G. Lipari, Minimizing CPU energy in real-time systems with discrete speed management, ACM Transactions on Embedded Computing Systems (TECS), vol. 8, no. 4, pp. 31, 2009.

[4] M. A. Haque, H. Aydin, and D. Zhu, Energy-aware Standby-Sparing Technique for periodic real-time applications, in: 29th International Conference on Computer Design (ICCD), IEEE, pp. 190-197, 2011.

[5] M. Kargahi, and A. Movaghar, Performance optimization based on analytical modeling in a real-time system withconstrained time/utility functions, IEEE Transactions on Computers, vol. 60, no. 8, pp. 1169-1181, 2011.

[6] J.-J. Chen, A. Schranzhofer, and L. Thiele, Energy minimization for periodic real-time tasks on heterogeneous processing units, in: International Symposium on Parallel & Distributed Processing (IPDPS), pp. 1-12, 2009.

[7] C.-Y. Yang, J.-J. Chen, T.-W. Kuo, and L. Thiele, An approximation scheme for energy-efficient scheduling of real-time tasks in heterogeneous multiprocessor systems, in: Proceedings of the Conference on Design, Automation and Test in Europe, pp. 694-699, 2009.

[8] A. M. A. Athlon, processor model 6 CPGA data sheet, On the World Wide Web at http://www.amd. com/products/cpg/athlon/techdocs/pdf/24319. pdf, 2001.

[9] Intel Crop, Intel PXA270 processor electrical mechanical, and thermal specification data sheet, 2004, Available from: <http://www.phytec.com/pdf/datasheets/PXA270_DS.pdf>.

[10] R. R. Muntz, and E. Coffman, Optimal preemptive scheduling on two-processor systems, IEEE Transactions on Computers, vol. 100, pp. 1014-1020, 1969.

[11] S. K. Baruah, N. K. Cohen, C. G. Plaxton, and D. A. Varvel, Proportionate progress: A notion of fairness in resource allocation, Algorithmica, vol. 15, no. 6, pp. 600-625, 1996.

[12] P. Holman, and J. H. Anderson, Adapting Pfair scheduling for symmetric multiprocessors, Journal of Embedded Computing, vol. 1, no. 4, pp. 543-564, 2005.

[13] M. L. Dertouzos, and A. K. Mok, Multiprocessor online scheduling of hard-real-time tasks, IEEE Transactions on Software Engineering, vol. 15, no. 12, pp. 1497-1506, 1989.

[14] H. Cho, B. Ravindran, and E. D. Jensen, An optimal real-time scheduling algorithm for multiprocessors, in: 27th IEEE International Real-Time Systems Symposium (RTSS), pp. 101-110, 2006.

[15] W. Y. Lee, Energy-Saving DVFS Scheduling of Multiple Periodic Real-Time Tasks on Multi-core Processors, in: Proceedings of the 13th IEEE/ACM International Symposium on Distributed Simulation and Real Time Applications, pp. 216-223, 2009.

[16] J.-J. Chen, H.-R. Hsu, K.-H. Chuang, C.-L. Yang, A.-C. Pang, and T.-W. Kuo, Multiprocessor energy-efficient scheduling with task migration considerations, in: Proceedings of the 16th Euromicro Conference on Real-Time Systems (ECRTS), pp. 101-108, 2004.

[17] J.-J. Chen, and T.-W. Kuo, Multiprocessor energy-efficient scheduling for real-time tasks with different power characteristics, in: International Conference on Parallel Processing (ICPP), pp. 13-20, 2005.

[18] J.-J. Chen, H.-R. Hsu, and T.-W. Kuo, Leakage-aware energy-efficient scheduling of real-time tasks in multiprocessor systems, in: Proceedings of the 12th Real-Time and Embedded Technology and Applications Symposium, pp. 408-417, 2006.

[19] E. C. Horvath, S. Lam, and R. Sethi, A level algorithm for preemptive scheduling, Journal of the ACM (JACM), vol. 24, no. 1, pp. 32-43, 1977.

[20] T. Gonzalez, and S. Sahni, Preemptive scheduling of uniform processor systems, Journal of the ACM (JACM), vol. 25, no. 1, pp. 92-101, 1978.

[21] D. Hochbaum, and D. Shmoys, A polynomial approximation scheme for machine scheduling on uniform processors: using the dual approximation approach, in: Foundations of Software Technology and Theoretical Computer Science, pp. 382-393, 1986.

[22] S. K. Baruah, and J. Goossens, Rate-monotonic scheduling on uniform multiprocessors, IEEE Transactions on Computers, vol. 52, no. 7, pp. 966-970, 2003.

[23] C. L. Liu, and J. W. Layland, Scheduling algorithms for multiprogramming in a hard-real-time environment, Journal of the ACM (JACM), vol. 20, no.1, pp. 46-61, 1973.

[24] S. Funk, J. Goossens, and S. Baruah, Online scheduling on uniform multiprocessors, in: Proceedings 22nd Real-Time Systems Symposium (RTSS), pp. 183-192, 2001.

[25] S.-Y. Chen, and C.-W. Hsueh, Optimal dynamic-priority real-time scheduling algorithms for uniform multiprocessors, in: Real-Time Systems Symposium (RTSS), pp. 147-156, 2008.

[26] Y. Yu, and V. K. Prasanna, Resource allocation for independent real-time tasks in heterogeneous systems for energy minimization, Journal of Information Science and Engineering, vol. 19, no.3, pp. 433-450, 2003.

[27] J. D. Ullman, NP-complete scheduling problems, Journal of Computer and System sciences, vol. 10, no. 3, pp. 384-393, 1975 .

[28] J. K. Lenstra, D. B. Shmoys, and É. Tardos, Approximation algorithms for scheduling unrelated parallel machines, Mathematical programming, vol. 46, no. 1, pp. 259-271, 1990.

[29] T. Gonzalez, E. L. Lawler, and S. Sahni, Optimal preemptive scheduling of two unrelated processors, ORSA Journal on Computing, vol. 2, no. 3, pp. 219-224, 1990.

[30] K. Jansen, and L. Porkolab, Improved approximation schemes for scheduling unrelated parallel machines, Mathematics of Operations Research, vol. 26, no. 2, pp. 324-338, 2001.

[31] B. Srivastava, An effective heuristic for minimising makespan on unrelated parallel machines, Journal of the Operational Research Society, vol. 49, no. 8, pp. 886-894, 1998.

[32] S. Baruah, Feasibility analysis of preemptive real-time systems upon heterogeneous multiprocessor platforms, in: Proceedings of the 25th International Real-Time Systems Symposium (RTSS), pp. 37-46, 2004.

[33] H.-R. Hsu, J.-J. Chen, and T.-W. Kuo, Multiprocessor synthesis for periodic hard real-time tasks under a given energy constraint, in: Proceedings of the conference on Design, automation and test in Europe (DATE), (European Design and Automation Association, pp. 1061-1066, 2006.

[34] J.-J. Chen, and T.-W. Kuo, Allocation cost minimization for periodic hard real-time tasks in energy-constrained DVS systems, in: Proceedings of the 2006 IEEE/ACM international conference on Computer-aided design (ICCAD), ACM, pp. 255-260, 2006.

[35] T.-Y. Huang, Y.-C. Tsai, and E.-H. Chu, A near-optimal solution for the heterogeneous multi-processor single-level voltage setup problem, in: International Parallel and Distributed Processing Symposium (IPDPS), pp. 1-10, 2007.

[36] E.-H. Chu, T.-Y. Huang, and Y.-C. Tsai, An optimal solution for the heterogeneous multiprocessor single-level voltage-setup problem, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 28, pp. 1705-1718, 2009.

[37] J. Luo, and N. K. Jha, Static and dynamic variable voltage scheduling algorithms for real-time heterogeneous distributed embedded systems, in: Proceedings of the 2002 Asia and South Pacific Design Automation Conference (ASP-DAC), pp. 719, 2002.

[38] D. Ding, L. Zhang, and Z. Wei, A novel voltage scaling algorithm through ant colony optimization for embedded distributed systems, in: International Conference on Integration Technology (ICIT), pp. 547-552, 2007.

[39] J. Lin, A. M. Cheng, and R. Kumar, Real-time task assignment in heterogeneous distributed systems with rechargeable batteries, in: International Conference on Advanced Information Networking and Applications (AINA), pp. 82-89, 2009.

[40] M. Kargahi, and A. Movaghar, Stochastic DVS-based dynamic power management for soft real-time systems, Microprocessors and Microsystems, vol. 32, no. 3, pp. 121-144, 2008.

[41] M. DeVuyst, A. Venkat, and D. M. Tullsen, Execution migration in a heterogeneous-ISA chip multiprocessor, in: Proceedings of the 17th international conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), pp. 261-272, 2012.

[42] T. Li, P. Brett, R. Knauerhase, D. Koufaty, D. Reddy, and S. Hahn, Operating system support for overlapping-ISA heterogeneous multi-core architectures, in: 16th International Symposium on High Performance Computer Architecture (HPCA), pp. 1-12, 2010.

[43] V. Nollet, P. Avasare, J.-Y. Mignolet, and D. Verkest, Low cost task migration initiation in a heterogeneous mp-soc, in: Proceedings of the International Conference on Design, Automation and Test in Europe (DATE), pp. 252-253, 2005.

[44] H. Shen, and F. Pétrot, Novel task migration framework on configurable heterogeneous MPSoC platforms, in: Proceedings of the Asia and South Pacific Design Automation Conference (ASP-DAC), pp. 733-738, 2009.

[45] J. A. Bondy, and U. S. R. Murty, Graph theory with applications, Elsevier, New York, 1976.

[46] B. Andersson, and E. Tovar, Multiprocessor scheduling with few preemptions, in: Proceedings of the 12th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA), pp. 322-334, 2006.

[47] N. Karmarkar, A new polynomial-time algorithm for linear programming, in: Proceedings of the 16th annual ACM symposium on Theory of computing, pp. 302-311, 1984.

[48] J. Renegar, A polynomial-time algorithm, based on Newton's method, for linear programming, Mathematical Programming, vol. 40, no. 1, pp. 59-93, 1988.

[49] A. Schrijver, Theory of linear and integer programming, John Wiley and Sons, New York, 1986.

[50] S. Funk, and V. Nanadur, LRE-TL: An Optimal Multiprocessor Scheduling Algorithm for Sporadic Task Sets, in: 17th International Conference on Real-Time and Network Systems (RTNS), pp. 159-168, 2009.

[51] R.Dick, Embedded System Synthesis Benchmarks Suite(E3S), Available from: <http://ziyang.eecs.umich.edu /~dickrp/e3s/>.

[52] IBM ILOG CPLEX 12.1, User's Manual, Available from: <http://www.ilog.com/products/cplex>, 2010.

[53] K. Funaoka, S. Kato, and N. Yamasaki, Work-conserving optimal real-time scheduling on multiprocessors, in: Euromicro Conference on Real-Time Systems (ECRTS), pp. 13-22, 2008.

[54] K. Funaoka, S. Kato, and N. Yamasaki, Energy-efficient optimal real-time scheduling on multiprocessors, in: 11th International Symposium on Object Oriented Real-Time Distributed Computing (ISORC), pp. 23-30, 2008.

[55] K. Funaoka, A. Takeda, S. Kato, and N. Yamasaki, Dynamic voltage and frequency scaling for optimal real-time scheduling on multiprocessors, in: International Symposium on Industrial Embedded Systems (SIES), pp. 27-33, 2008.

[56] L. Cucu-Grosjean, and J. Goossens, Exact schedulability tests for real-time scheduling of periodic tasks on unrelated multiprocessor platforms, Journal of Systems Architecture, vol. 57, pp. 561-569, 2011.

[57] T. Megel, R. Sirdey, and V. David, Minimizing task preemptions and migrations in multiprocessor optimal real-time schedules, in: 31th Real-Time Systems Symposium (RTSS), pp. 37-46, 2010.

[58] G. Raravi, B. Andersson, K. Bletsas, and V. Nelis, Task assignment algorithms for two-type heterogeneous multiprocessors, in: 24th Euromicro Conference on Real-Time Systems (ECRTS), pp. 34-43, 2012.

Appendix

List of Symbols

Symbols	Present
T	Set of all tasks
T_i	The i^{th} task in T
p_i	Period and relative deadline of task T_i
e_i	Execution requirement of task T_i
N	Number of tasks
M	Set of all machines
M_j	The j^{th} machine in M
m	Number of machines
k_j	Number of voltage-levels of machine M_j
V_j	Set of all voltage-levels of machine M_j
V_{jl}	The l^{th} voltage-level of machine M_j
S_{jl}	Maximum speed of machine $M_j$at voltage-level V_{jl}
Pow_{jl}	Maximum power consumption of machine M_j at voltage-level V_{jl}
T_0	Dummy task; (its execution is equivalent to be idle)
Pow_{0jl}	Power consumption of machine M_j at voltage-level V_{jl} when idle
S_{ijl}	Speed of machine M_j at voltage-level V_{jl} when it is executing task T_i (i> 0)
Pow_{ijl}	Power consumption of machine M_j at voltage-level V_{jl} when it is executing task T_i (i> 0)
$E_{ijl}(t)$	Energy consumed by machine M_j at voltage-level V_{jl} when it is executing task T_i (i> 0)for t amount of time
$(T_i, M_j, V_{jl}, t_s, t_f)$	A job slice, that means Task T_i is scheduled on machine M_j at voltage-level V_{jl} in time interval $[t_s, t_f)$
$\overline{E(a,b)}$	Minimum average energy usage of M in time interval $[a,b)$
H	Size of a hyper period
$\overline{E(H)}_{min}$	Minimum average energy usage of T on M in a hyper period
SP	Schedule Period
SP_k	The k^{th} SP (the time interval between the k^{th} and $(k + 1)^{th}$ task releases)
$\lvert SP_k \rvert$	Length of SP_k
ST	Scaled task
ST_i	Scaled task corresponding to task T_i
e_i^k	Execution time of the scaled task corresponding to task T_i in SP_k
s_k	Start time of SP_k
t_k	Finish time of SP_k
$\overline{E(SP)}_{min}$	Minimum average energy usage of scaled task set corresponding to Tin a SP, on M
t_{ijl}	The amount of time needed by task T_i to be executed on machine M_j at voltage-level V_{jl}(LP variables)
TS_{ijl}	Task segments corresponding tot_{ijl}> 0wherei> 0
t'_{ijl}	Size of non-scheduled part of task segment TS_{ijl} throughout Algorithm 1
e'_i	The remaining execution requirement of migratory task T_i in the ongoing iteration of Algorithm 1
γ	the length of available interval in which the rest of scheduling takes place in each iteration of Algorithm 1
σ	The length of available interval in which scheduling of current iteration is going to be done in Algorithm 1
F	Set of full machines in each iteration of Algorithm 1
U	Set of urgent tasks in each iteration of Algorithm 1
X	A matching between machines and tasks that covers all urgent task and full machines
χ_T	A matching from all urgent tasks to machines
χ_M	A matching from all full machines to tasks
R_U	The range of χ_T
R_F	The range of χ_M

Change of Type A	A task segment TS_{ijl} is assigned to the proper machine at the proper voltage-level for the length of t'_{ijl}
Change of Type B	A machine M_j becomes full
Change of Type C	A task T_i becomes urgent
Change of Type D	A new RBTS arrives or its execution finishes
$A(TS_{ijl})$	A Change of Type A on task segment TS_{ijl}
$B(M_j)$	A Change of Type B on machine M_j
$C(T_i)$	A Change of Type C on task T_i
$RBTS_{ijl}$	Released-based task-segment corresponding to task segment TS_{ijl}
τ_{ijl}	The length of $RBTS_{ijl}$
S	A set of machines that is used in Algorithm 4
N(S)	Set of all neighbours of node S in a graph
prevX	Set of selected edges from the previous iteration
nextX	Set of edges in current edges
U	Set of urgent tasks
F	Set of full machines
$T_{n'\times m}$	adjacency matrix where $n'$and m are the numbers of migratory tasks and processors
VT	Set of nodes corresponding to urgent tasks or each task that adjacent to at least one full machine
VP	Set of nodes corresponding to full machines or each machine that adjacent at least one urgent task
V_S	Source node
V_d	Sink node

Mahmood Gholipour received his Master's degree in Computer Engineering from the School of Electrical and Computer Engineering, University of Tehran. In 2010, he received hisBachelor's degree in Computer Engineering from Arak University, Arak, Iran.

E-mail: m.gholipour@ut.ac.ir

Mehdi Kargahi received his B.S. degree in computer engineering from Amir-Kabir University in 1998 and the M.S. and Ph.D. degrees in computer engineering from Sharif University of Technology in 2001 and 2006, respectively. He is currently an associate professor in the Department of Electrical and Computer Engineering at University of Tehran, Iran. He has been a researcher at the Institute for Studies in Theoretical Physics and Mathematics (IPM) from 2003. His research interests include distributed systems, performance modeling and dependable real-time systems

E-mail: kargahi@ut.ac.ir.

Heshaam Faili received his B.S. and M.S. degrees in Software Engineering and his PhD in Artifical Intelligence from Sharif University of Technology on 1997, 1999 and 2006 respectively. He joined to the Artificial Intelligence and Robotic group of the School of Electrical and Computer Engineering, University of Tehran at 2008. He is now an associate professor, and his main research interests include approximation and randomized algorithms, AI and statistical approaches, text processing and mining methods.

E-mail: hfaili@ut.ac.ir

Shahbaz Youssefi received his B.S. degree in Computer Engineering from the School of Electrical and Computer Engineering, University of Tehran, Iran. He has worked on real-time scheduling theory in the Dependable Real-Time Systems (DRTS) Research Laboratory. He is now a researcher in at the University of Genova, Italy.

E-mail: shahbaz.youssefi@unige.it

Hadi Ravanbakhsh received his B.S. degree in Computer Engineering from the Department of Electrical and Computer Engineering at University of Tehran, Iran. His project for graduation has been on energy-aware scheduling of real-time tasks on unrelated parallel machines which is done in the Dependable Real-Time Systems (DRTS) Laboratory. He is now a PhD candidate at Department of Computer Science, University of Colorado-Boulder.

E-mail: hadi.ravanbakhsh@colorado.edu

Paper Handling Data:

Submitted: 10.11.2016
Received in revised form: 20.12.2016
Accepted: 02.01.2017
Corresponding author: Dr. Mehdi Kargahi,
School of Eectrical and Computer Engineering, University of Tehran, Tehran, Iran.

[1]A complete list of the symbols used throughout this paper is presented in the appendix at the end of this paper.