JCSE Journal of Computing Science and Engineering

Regular Paper
Journal of Computing Science and Engineering,
Vol. 7, No. 1, March 2013, pp. 30-43

Energy Aware Scheduling of Aperiodic Real-Time Tasks on Multiprocessor Systems

Naveen Anne and Venkatesan Muthukumar*
Department of Electrical and Computer Engineering, University of Nevada, Las Vegas, NV, USA
naveen.anne@gmail.com, Venkatesan.Muthukumar@unlv.edu

Abstract
Multicore and multiprocessor systems with dynamic voltage scaling architectures are being used as one of the solutions to satisfy the growing needs of high performance applications with low power constraints. An important aspect that has propelled this solution is effective task/application scheduling and mapping algorithms for multiprocessor systems. This work proposes an energy aware, offline, probability-based unified scheduling and mapping algorithm for multiprocessor systems, to minimize the number of processors used, maximize the utilization of the processors, and optimize the energy consumption of the multiprocessor system. The proposed algorithm is implemented, simulated and evaluated with synthetic task graphs, and compared with classical scheduling algorithms for the number of processors required, utilization of processors, and energy consumed by the processors for execution of the application task graphs.

Category: Smart and intelligent computing

Keywords: Multiprocessor scheduling; Application mapping; Energy aware scheduling; Task graphs; DVFS

I. INTRODUCTION

Various scheduling policies for real-time multiprocessors systems have been constantly evolving. Scheduling of real-time applications/tasks on multiprocessor systems often has to be combined with task to processor mapping. With the current design trends moving towards multicores and multiprocessor systems for high performance and embedded system applications, the need to develop design techniques to maximize utilization of processor time, and at the same time minimize power consumption, have gained importance. General design techniques to achieve the above goals have been fueled by the development of both hardware and software solutions. Hardware solutions to minimize power and energy consumption include: dynamic voltage and frequency scaling (DVFS) processors, dynamic power management modules (Advanced Configuration and Power Interface, ACPI), thermal management modules, intelligent energy management (IEM), and heterogeneous multicore or multiprocessor systems [1]. Software solutions for maximizing processor utilization and energy minimizations include: parallelization of instructions, threads, and tasks; effective thread/task scheduling; and mapping algorithms for multicore and multiprocessor systems.

In this work we propose a new energy aware, probability-based offline scheduling algorithm of aperiodic real-time tasks for a multiprocessor system. The proposed solution combines the traditional mapping and scheduling algorithm into a unified process. The proposed algorithm is modeled for multiprocessor (homogeneous processors) systems that support dynamic voltage scaling (DVS) pro-

Open Access http://dx.doi.org/10.5626/JCSE.2013.7.1.30 http://jcse.kiise.org

Received 9 October 2012, **Revised** 7 January 2013, **Accepted** 28 February 2013
*Corresponding Author

cessors. The algorithm is based of the force-directed scheduling (FDS) algorithm [2], most commonly used during the high-level design synthesis. The algorithm determines an optimal schedule for each task, on a processor operating at a discrete voltage level with constant frequency, which will be able to satisfy the deadline. The algorithm also maximizes the utilization of all processors, and minimizes the energy consumption of the system. For comparison purposes, this work implements four different variations of the scheduling algorithms for multiprocessor systems that support DVS processors. The first three scheduling algorithms implement the scheduling and mapping algorithms as different processes. The first scheduling algorithm, called scheduling after scaling (SaS), performs the mapping of tasks to the processors, and iteratively scales down the voltage of the processors from the maximum operable voltage, based on the slack of the tasks. Once the tasks have been mapped to the processors, voltage levels are determined, and scheduling is performed. In the second scheduling algorithm, called scheduling before scaling (SbS), the tasks are first scheduled based on the deadline (earliest deadline first, EDF) [3]. The SbS scheduling algorithm is similar to the low-energy EDF algorithm proposed by Manzak and Chakrabarti [4]. Once scheduled, the tasks with slack are assigned to processors with lower operating voltages. In the above two algorithms, processors are added to the system, to satisfy the dependency and deadline constraints. In the third scheduling algorithm, called probability based scheduling (PbS), the scheduling is based on optimizing the probability (probability of utilization) of the tasks executing in a unit time step. The probability function optimizes the utilization of each processor, and hence the utilization of the whole multiprocessor system. Mapping of the tasks and assignment of voltage levels to processors is achieved by determining the probability of utilization of each processor operating at different voltages. The final scheduling algorithm, called energy-probability based scheduling (E-PbS), combines the process of scheduling, and mapping or assignment of voltages to processors, together. The probability of utilization function incorporates the energy factor, and the scheduling algorithm determines task operation times and voltage levels of the respective processors. The developed algorithms have been implemented, simulated and evaluated for synthetic real-time applications (standard task graphs). The algorithms are compared for utilization of the processors, number of processors (resources), and energy consumed by the processors.

II. PRIOR WORK

Scheduling algorithms have been classically categorized into online or offline, priority or non-priority, pre-emptive or non pre-emptive, and hard or soft deadline scheduling algorithms. Two of the classical scheduling algorithms for uniprocessor are fixed priority rate-monotonic scheduling (RMS), and dynamic priority EDF algorithms [3]. With the advent of multicore and multiprocessor systems, where each processor is capable of dynamic performance and energy consumptions based on variable voltage and frequency operations, scheduling and mapping algorithms to optimize performance and energy have gained importance. Weiser et al. [5] were the first to propose the DVFS technique. Their approach was to reduce CPU ideal times, by dynamically adjusting DVFS levels in OS-level scheduling algorithms. Yao et al. [6] and Ishihara and Yasuura [7] proposed an optimal offline-scheduling algorithm for independent and dependent tasks, respectively, to minimize energy consumption.

Aydin et al. [8] proposed a dynamic speculative scheduling algorithm, by exploiting slack times. Zhu et al. [9] proposed scheduling algorithms to improve the above algorithms for multiprocessors for dependent and independent tasks, to share slack times. More recent works [10-14] on DVFS do not account for frequency scaling, as analysis shows performance loss on processor frequency scaling. Also, work by Dhiman and Rosing [14] has shown that memory intensive tasks have less dependency on frequency scaling, which is true with modern embedded applications. Also, issues such as task granularity and shared resource contention in caches, memories and buses have to be accounted for. Algorithms for resource contention aware or mitigations have been explored in [15-18]. Lee and Zomaya [19] proposed a DVFS energy aware scheduling algorithm for a heterogeneous distributed multiprocessor system. Also, work comparing dynamic power management and DVFS methods has been conducted by Cho and Melhem [20], and has concluded that DVFS scheduling methods provide better energy saving for multiprocessor systems. Shin and Kim [21] have considered the problem of mapping the tasks of the task graphs from real-time applications to available cores, and then determined the DVFS for each core to meet the timing deadlines, while minimizing the power consumption. The problem of determining the optimal voltage schedule for a real time system with fixed priority jobs, implemented on a variable voltage processor, is discussed in [22].

Okuma et al. [23] addressed the problem of less energy consumption, by simultaneously assigning the CPU time and a supply voltage to each task that results in low power. This algorithm uses a variable voltage processor core to schedule the tasks [7]. An online energy aware algorithm for distributed heterogeneous hard real-time systems, based on a modification of the EDF algorithm, is proposed in [8]. The energy reduction is obtained by reducing the speed of the processor when the only ready task in the system has no successor, or when the previous task has not exhausted its worst-case execution time (WCET).

In this work, we present an offline dynamic voltage scaling scheduling algorithm for dependent and non-periodic task graphs, to optimize energy consumption on a multiprocessor system. The proposed work in this article follows a similar approach to the method presented in [21], but with the following differences and advantages: 1) this work considers task graphs derived from real-time applications, as well as synthetic task graphs; 2) the process of mapping the tasks to a processor, voltage scaling of the processor and scheduling are considered together; and 3) only voltage scaling is considered in this work, for the reasons mentioned above. However, frequency scaling can also be implemented, by considering an appropriate cost function.

III. DEFINITIONS

Definition 1 (Task model). A task model is required as the basis for discussing scheduling. A real-time task is a basic executable entity, which can be scheduled; it can be either periodic or aperiodic, with soft or hard timing constraint. A task is best defined with its main timing parameters. This model includes the following primary parameters.

- r_i - release time of the task
- e_i - WCET of the task
- D_i - relative deadline of the task
- d_i - absolute deadline of the task
- p_i - period of the task
- g_i - start time of the task
- h_i - end time of the task
- s_i - slack of the task = $d_i - e_i$

Definition 2 (Scheduling model). For the problem of scheduling, we consider a task set or application A = $\{T_0, T_1, \ldots, T_n\} = \{T_i\}$, where each task T_i is an aperiodic task with a hard deadline d_i, and WCET of e_i and best-case execution time (BCET) of e_z. These tasks execute on a multiprocessor system consisting of processing elements PE = $\{PE_j\} = \{PE_0, PE_1, \ldots, PE_m\}$, which can operate at a prespecified set of voltages $V = \{V_1, V_2, \ldots, V_p\}$, where $i < j$ implies $V_i > V_j$. We denote a task T_i running on the processor PE_j scaled at voltage V_k as T_{ik}. In this work we also phrase the WCET of a task T_i as its execution time at maximum voltage V_1, denoted by e_{i1}. Thus, its execution time at voltage V_k is determined as:

$$e_{ik} = e_{i1} * (V_1/V_k)\,. \tag{1}$$

Also, all tasks T are subject to precedence constraints, and a partial order $T_i < T_j$ is defined on T, where $T_i < T_j$ implies that T_i precedes T_j. The problem of scheduling of an application A is denoted as:

$$\alpha(T_i) = \{t_i\} \text{ and } \alpha(A) = \{\alpha(T_1), \alpha(T_2), \alpha(T_3), \ldots, \alpha(T_n)\}. \tag{2}$$

If there exists a precedence operation between any two tasks, such that the execution of a task T_i should occur before the execution of task T_j, represented as $T_i < T_j$, then for each $T_i \in T$, $t_i + e_i \le d_i$ and $t_n + e_n \le D$, where D is the absolute deadline of the last task.

Definition 3 (Mapping and voltage scaling). Mapping is the process of assigning each task in an application to a processor, such that the processor executes the task and satisfies the task deadline, as well as other constraints (power, throughput, completion time), if any. For an application $A = \{T_1, T_2, \ldots, T_n\}$, where T_i is a task instance in the application, mapping a task, denoted as $\sigma(T_i)$, is defined as an assignment of the processor $\{P_j\} = \{PE_1, PE_2, \ldots, PE_m\}$ = PE for the task T_i:

$$\sigma(T_i) = \{PE_j\}. \tag{3}$$

Application mapping is defined as mapping of all tasks, T_i, in the application A to processors, and is denoted as:

$$\sigma(A) = \{\sigma(T_1), \sigma(T_2), \ldots, \sigma(T_n)\}. \tag{4}$$

Voltage scaling is the processes of assigning a discrete operating voltage to each processor, such that the processor executes the tasks and satisfies the task deadline, as well as other constraints (power, throughput, completion time), if any. For a set of processors PE = $\{PE_1, PE_2, \ldots, PE_m\}$ that can be scaled at discrete voltages $V = \{V_1, V_2, \ldots, V_p\}$, voltage scaling is denoted as $\delta(T_{ik})$, and is defined as an assignment of voltages (V_k) and task (T_i) to the processors:

$$\delta(T_{jk}) = \{V_k\} \text{ and } \delta(A) = \{\delta(T_{1k}), \delta(T_{2k}), \ldots, \delta(T_{nk})\}. \tag{5}$$

Definition 4 (Processor power model). Assuming that a processor PE_{ik} is scaled at a supply voltage V_k, and a frequency f_j, and a task T_i takes '*n*' clock cycles on the processor to complete its execution, the power consumption in the processor is given by:

$$P(PE_{ik}) = C_L * N_{SW} * V_k^2 * f_j, \forall\ i(tasks)\,. \tag{6}$$

where, C_L is the output capacitance, and N_{sw} is the number of switches per clock. Energy consumption by a task T_i can be computed as:

$$E_i = \frac{n * P(PE_{ik})}{f_j} = n * C_L * N_{SW} * V_k^2,\ \forall\ i(tasks)\,. \tag{7}$$

From the above equations, we can conclude that lowering the supply voltage drastically reduces the power and energy consumption of the particular processor. The supply voltage also affects the circuit delay. The circuit delay is given by $T_d = k * V_k^2/(V_k - V_t)^2$, where k is a constant that is dependent on output capacitance, and V_t is the threshold voltage.

Definition 5 (Scheduling power and energy model). The power consumed by a schedule is the total power consumed by all tasks that execute at the time period t, and is denoted as $P_\alpha(t)$,

$$P_\alpha(t) = \sum_{i=0}^{m} \mathrm{P}(PE_i(t)) . \tag{8}$$

Energy consumed by the schedule is the integral of $P_\alpha(t)$,

$$E_\alpha = \int_0^t P_\alpha(t) * \Delta t . \tag{9}$$

Definition 6 (Problem statement). Given an application A with tasks T_i, determine an optimal schedule $\alpha(A)$ with minimal energy consumption for a multiprocessor system PE, while mapping tasks to PEs, $\sigma(A)$ and voltage scaling the processors, $\delta(PE)$ in a multiprocessor system, such that all tasks are executed before their respective deadlines, and the end task before the final deadline D.

Minimize $(\sum_{i=1}^{n}(e_i))$ *and minimize* (E_α), $t_i + e_i \leq d_i$ *and* $t_n + e_n \leq D$.

IV. ENERGY AWARE SCHEDULING ALGORITHM

A. Tasks Deadlines

Since the input task graph considered has only a final hard deadline, the first step is to determine the deadlines of all the tasks.

- Find the path with the highest execution time: the path with the highest execution time is determined using a depth first search algorithm that determines all paths, from source to sink, of the task graph. The highest execution time is denoted as e_{max}.
- Find the deadlines of the tasks by backtracking: the deadlines of all the tasks are determined by backtracking to each task from the sink task. In order to determine the efficiency of the proposed algorithms the highest execution time e_{max} is relaxed to $\bar{e}_{max}$ ($D = \bar{e}_{max}$). This relaxation of the deadline allows the tasks to be operated at a lower voltage, and hence better power and energy consumption.

The deadlines of all other tasks are found by backtracking from the sink task. The deadline of a predeccessor task d_i with one sucessor task d_j is determined as $d_i = d_j - e_j$. If a task has more than one successor, then the deadline is obtained by choosing the minimum of the differences of the deadlines and execution times of the successors. For example, a task i has successors j, k and l, then d_i is given, determined as $\min(d_j-e_j, d_k-e_k, d_l-e_l)$. The slack of each task is calculated as the difference of deadline and execution time of each task, $s_i = d_i - e_i$.

B. Scheduling after Scaling Algorithm (SaS)

The proposed SaS algorithm first allocates voltages to each task, and then progressively schedules and binds the tasks to the processors.

Step 1 (Scaling the voltage of each task). Assume that the tasks can only execute at a pre-specified set of voltages, and all tasks are by default assigned the highest voltage. First the tasks are sorted with respect to their slack. Then the task with least slack is made the current task (T_i), and its execution time (e_i) is modified by a "scaling factor". The scaling factor is defined as the ratio of the current voltage (V_i) assigned to the task, to the minimum voltage (V_k) that can be assigned to the task, provided that the execution time of the task is less than the deadline of the task. The new execution time of the task is denoted as e_i. The start times and end times of all the successors of the current task are modified, until the sink. If any task misses its deadline, then the scaling factor is reduced. The new voltage assigned to the task is denoted as $V_{(k-1)}$. Again, the start and end times of the successors are modified, to see if any of the tasks misses its deadline. This procedure is repeated until none of the tasks misses its deadline, and the voltage at which the current task executes is fixed. This voltage scaling is performed on all the tasks in the graph, sorted by its slack time.

Step 2 (Assigning the tasks on processors). The tasks are then scheduled on the processors, based on the EDF scheduling algorithm, and the first come first served (FCFS) mapping algorithm. At every instance of the start time of a task, a check is done to see if any processor is idle. The task is scheduled on the idle processor, if any; else a new processor is initialized. The pseudocode of the SaS algorithm is shown in Algorithm 1.

```
Algorithm 1 Scheduling after Scaling (SaS)
Inputs: $A = \{T_1, T_2,\ldots, T_n\}$, $PE = \{PE_0, PE_1, \ldots, PE_m\}$
Outputs: $\sigma(A)$, $\delta(T_{ik})$, $\alpha(T_i)$
repeat {
    assign highest voltage to tasks;
    sort tasks based on their slacks;
    scale voltages based on slacks;
        } until (all tasks are scaled)
repeat {
    schedule tasks based on deadlines (EDF);
    assign tasks to free processors;
        } until (all tasks are scheduled)
```

C. Scheduling before Scaling Algorithm (SbS)

This algorithm performs scheduling of the tasks on the available processor, before scaling the task voltages. This algorithm is an extension of the EDF algorithm. The SbS algorithm first performs scheduling, followed by assignment of voltage to the tasks.

Step 1 (Sorting the tasks). This algorithm is different from the SaS algorithm in sorting the tasks. All the tasks are initially sorted with respect to their deadlines (similar to EDF scheduling algorithm).

Step 2 (Scheduling of tasks). The SbS algorithm proceeds from the source task to the sink task. A breadth-first search approach is adopted, to determine the task with the shortest deadline, and is scheduled first. This process is repeated, until all tasks are scheduled.

Step 3 (Assigning the tasks to processors). The tasks are then scheduled on the processors. At every instance of the start time of a task, a check is done to see if any processor is idle. The task is scheduled on the idle processor, if any; else a new processor is initialized.

```
Algorithm 2 Scheduling before Scaling (SbS)
Inputs: $A = \{T_1, T_2, \ldots, T_n\}$, $PE = \{PE_0, PE_1, \ldots, PE_m\}$
Outputs: $\sigma(A)$, $\delta(T_{ik})$, $\alpha(T_i)$
repeat {
    assign highest voltage to tasks;
    sort tasks based on their deadlines (EDF);
    assign tasks to free processors;
        } until (all tasks are scheduled)
repeat {
 for each processor {
  for each task {
    scale voltages of based on slacks and deadline;
              } }
        } until (all tasks are scaled)
```

Step 4 (Voltage scaling of the task on processors). The schedule produced from the above step is based on task deadlines only, and results in the minimum number of processors required to execute a task set. After scheduling, the voltages of the tasks are scaled, as in the SaS algorithm. The algorithm proceeds by sorting the processor, based on the tasks assigned to the processor. Voltage scaling is performed for each task on the processor. As voltage scaling is preformed on each task, the start and end times of its successor task in the same processor and any other processor, and the subsequent tasks on the same processor, are also modified. The pseudocode of the SbS algorithm is shown in Algorithm 2.

D. Probability based Scheduling Algorithm to Maximize Utilization

The proposed PbS algorithm increases the utilization of the processors, by optimizing the number of processors and energy consumed by the processors, by voltage scaling the tasks (T_i) that execute in the processor. The schedule is divided into equal time steps (Δt_j). The PbS algorithm determines the probability of utilization $P(T_i(\Delta t_j))$ of a task in each time step (Δt_j). A task is scheduled to a time step where the probability of utilization of the task is maximum. The probability of utilization is the sum of the probabilities of utilization of the current task (T_i) and its successors (T_k). Initial probabilities are calculated by assuming all tasks are operating at the highest voltage. When voltage scaling is performed on the tasks, the probability of utilization is dynamically updated, based on the time steps assigned to the voltage scaled task. Finally, the number of processors required to execute the final schedule is calculated. The following steps explain the algorithm in detail.

Step 1 (Determining the time step). All the tasks in the task set $A = \{T_1, T_2, \ldots, T_n\}$ are assumed to be executing at their BCET, i.e., all the tasks are currently running at the highest voltage. Divide the task graph into time steps, with the smallest execution time of all tasks as the unit of time steps.

Step 2 (Best-case time scheduling). The next step is to perform the best-case time (BCT) scheduling algorithm. The BCT algorithm starts with the source tasks in the task graph, and proceeds in a breadth-first order. Each task is scheduled in the earliest available time step ($BCT(T_i)$), and only if its predecessors are scheduled. A task T_i is characterized by its start time, $BCT_g(T_i)$, and end time, $BCT_h(T_i)$. This algorithm determines the earliest possible execution time of the task graph.

Step 3 (Worst-case time scheduling). The worst-case time (WCT) scheduling algorithm works exactly in the same way as the BCT algorithm, except that it starts at the sink of the task graph, and proceeds upwards. This algorithm determines the longest possible execution time for the task graph. A task T_i is characterized by its start time, $WCT_g(T_i)$, and end time, $WCT_h(T_i)$. This algorithm determines the latest possible execution time of the task graph.

Step 4 (Determining the bound limit). The algorithm proceeds by determining the bound limits (Δt_i) of each task. The bound limit of a task T_i is calculated as:

$$\Delta t_i = WCT_h(T_i) - BCT_g(T_i)\,. \tag{10}$$

where Δt_i represents the time steps within which the current task can be scheduled.

Step 5 (Determining the probability of utilization of each task in each time step). The next step in the algorithm is to determine the probability of utilization of each task in each time step, within the bound limit of the task (Δt_{ij}). The probability of utilization of a task T_i in a time step Δt_{ij} is represented as $PU(T_i(\Delta t_{ij}))$, where Δt_{ij} ranges from the earliest start time, to the latest end time of the task T_i.

Step 6 (Distributed probability of utilization). The distributed probability of utilization is the summation of probabilities of utilization of each task in each time step. It signifies the portion of a task that will be executed in a time step. The distribution probability of utilization DPU(Δt_{ij}) for a time step Δt_{ij} is given as:

$$DPU(\Delta t_{ij}) = \sum_{j} T_i(\Delta t_{ij}) . \qquad (11)$$

Step 7 (Schedule the task in a time step). The algorithm schedules the task T_i at time step Δt_{ij} that has the maximum probability of utilization. Once the task (T_i) is fixed to a time step (Δt_{ij}), the probabilities of utilization of its successors (T_k) are to be updated.

The change in the probability of utilization of a task T_k is denoted by ($\Delta PU(T_k)$). If the current task T_i is scheduled to execute in time step t_j, then the change in the probability of utilization of T_i is determined as $\Delta PU(T_i(\Delta t_{ij})) = 1 - PU(T_i(\Delta t_{ij}))$, and the change in the probability of utilization of successor task T_k is determined as $\Delta PU(T_k(\Delta t_{ij})) = -PU(T_i(\Delta t_{ij}))$.

Step 8 (Calculate the cumulative probability of utilization of the current task). The algorithm balances the distribution probability, by calculating the cumulative probability of utilization (CPU) of each task to time step assignment, and then selects the tasks with the smallest CPU. The CPU of a task T_i in time step Δt_{ij} is determined as $CPU(T_i(\Delta t_{ij})) = DPU(\Delta t_{ij}) * \Delta PU(T_i(\Delta t_{ij}))$. The total CPU of a task that is scheduled from time step Δt_{ij} to Δt_{ik} is calculated as:

$$CPU(T_i) = \sum_{z=j}^{k} CPU(T_i(\Delta t_z)) . \qquad (12)$$

Step 9 (Calculate the CPU of the successor tasks). Scheduling the current task (T_i) directly affects the scheduling of the successor tasks (T_j). Hence the successor effect is also considered, while determining the total cumulative probability of utilization TCPU of a task.

$$CPU(T_j) = \sum_{z=j+1}^{k} CPU(T_j(\Delta t_z)) . \qquad (13)$$

Step 10 (TCPU). The TCPU of a task determines the feasibility of scheduling the task in that particular time step. The least effect determines the step in which the task has to be executed. TCPU signifies that the task graph will use fewer resources (higher utilization). The TCPU of a task T_i is given as:

$$TCPU(T_i) = CPU(T_i) + CPU(T_j) . \qquad (14)$$

Steps 7–10 are repeated for all the tasks in the task set, and a final schedule is obtained.

Step 11 (Voltage scaling). Based on the schedule developed above, voltage scaling of the tasks are performed as in the SaS algorithm, and the FCFS mapping algorithm. At every instance of the start time of a task, a check is done to see if any processor is idle. The task is scheduled on the idle processor, if any; else a new processor is initialized. The final step in this algorithm is to find the number of processors required to execute the scheduled task set. By the end of scheduling, all the tasks are ready to be executed at a specified voltage. This algorithm finally returns a schedule in which the task voltages are scaled, making sure none of the tasks miss their deadlines, and that these tasks are executed on processors in such a way that the utilization of the processors is high. The pseudocode of the PbS algorithm is shown in Algorithm 3.

```
Algorithm 3 Probability based Scheduling (PbS)
Inputs: $A = \{T_1, T_2, \ldots, T_n\}$, $PE = \{PE_0, PE_1, \ldots, PE_m\}$
Outputs: $\sigma(A)$, $\delta(T_{ik})$, $\alpha(T_i)$
repeat {
    determine the bound limit for tasks;
    compute Probability of Utilization (PU);
    compute Distributive PU (DPU);
    compute Cummulative PU (CPU);
    compute Total CPU (TCPU)
    schedule tasks with min. TCPU;
    update PU;
      } until (all tasks are scheduled)
for each processor {
  for each task {
    scale voltages of based on slacks and deadline;
                } }
        } until (all tasks are scaled)
```

E. Energy-Probability based Scheduling Algorithm to Minimize Energy

The PbS algorithm proposed above reduces the concurrency of tasks in all time steps, to minimize the resources. However, the energy factor is not considered during scheduling. To achieve this, the energy-probability based scheduling (E-PbS) algorithm is proposed. The algorithm is similar to the PbS algorithm, but has an additional factor of voltage included in calculating the TCPU, called the TCPU-energy (TCPUE) factor.

Step 10 (TCPUE factor). Let $V = \{V_1, V_2, \ldots, V_k\}$ be the set of voltages at which a task can execute. The ($TCPUE(T_{ik})$) is calculated for different voltages V_k for the task T_i. The least $TCPUE(T_{ik})$ determines the voltage at which the task will be executed. After a task is fixed in voltage, the affected distributed probabilities of utilization are updated. This process is repeated for all the tasks, and each one is scheduled for execution at a particular voltage, and the distributed probabilities of utilization are updated dynamically, to get an optimized schedule. After determining the TCPUE, the task is scheduled in the time step where the TCPU is minimal. The TCPUE of a task T_i in time step Δt_j for each operating voltage V_k is determined as follows:

$$TCPUE(T_{ik}) = CPU(T_i) \times V_r^2 + CPU(T_j) \times V_r^2 . \quad (15)$$

The value of CPU(T_i) and CPU(T_j) for the current task and its sucessor tasks are determined as shown in steps 8 and 9, respectively, in Section IV-D. The TCPUE of each task operating at a specific volatge is calculated, by multiplying a voltage factor V_r^2 to the CPU terms. The voltage factor incorporates the energy awareness in the scheduling algorithm. The voltage factor V_r^2 is the ratio of the current operating voltage to the maximum possible operating voltage, V_k^2/V_p^2. The algorithm schedules a task at a voltage in the time step where the TCPUE is minimal. Thus in this method, the voltage scaling and mapping are implicitly integrated into the scheduling algorithm. Also, initial voltages are assigned to tasks based on their respective slacks. After a task is assigned a voltage, the affected distributed probabilities of utilization are updated. This process is repeated for all the tasks, and each one is scheduled for execution at a particular voltage, and the TCPUE is updated dynamically to get an optimized schedule. The final step in this algorithm is to find the number of processors required to execute the scheduled task set, as shown in step 11 (Section IV-D). The pseudocode of the E-PbS algorithm is shown in Algorithm 4.

Algorithm 4 Energy-Probability based Scheduling (E-PbS)

```
Inputs: A = {$T_1$, $T_2$, …, $T_n$}, PE = {$PE_0$, $PE_1$, …, $PE_m$}
Outputs: $\sigma(A)$, $\delta(T_{ik})$, $\alpha(T_i)$
repeat {
    determine the bound limit for tasks;
    compute Probability of Utilization (PU);
    compute Distributive PU (DPU);
    compute Cummulative PU (CPU);
    compute Total CPU-Energy (TCPUE)
    schedule tasks and assign processor;
    update PU;
  } until (all tasks are scheduled)
```

V. EXAMPLE

This section explains the SaS, SbS, PbS, and E-PbS algorithms, with the aid of an example. Fig. 1 shows an example task graph, consisting of ten tasks, with edges between them indicating the dependencies between the tasks. The source (task 0) and sink (task 11) tasks are redundant tasks that have zero execution time. This task graph is expressed in standard task graph (STG) format, as shown in Table 1. Only the final deadline of the end task in the STG is specified. The first step of all proposed algorithms is to determine the deadline and slack of each task, by backtracking from the sink task, as described in Section IV-A. Each task T_i is represented by parameters e_i, {T_j}, {T_k}, g_i, h_i, d_i, and s_i, which are the execution time, predecessor task set, successor task set, start time, end time, deadline and slack of the task, respectively. Also, a task executed on a processor operating at a specific voltage is simply refered to as a task operating at a specific voltage, or the voltage assigned to the task.

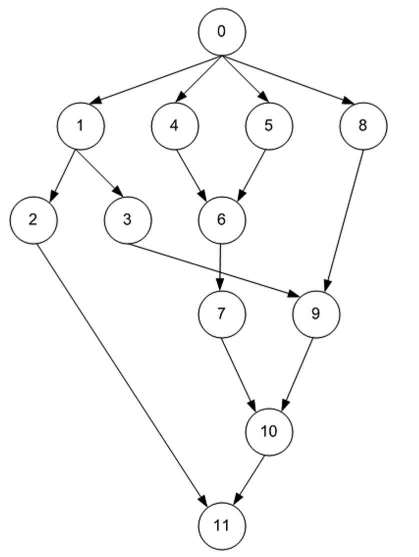

Fig. 1. Standard task graph.

Table 1. Task graph example

T_i	e_i	{T_j}	{T_k}	g_i	h_i	d_i	s_i
0	0	0	0	0	0	0	11.5
1	9	2, 3	0	0	9	20.5	11.5
2	4	11	1	9	13	34.5	21.5
3	3	9	1	9	12	23.5	11.5
4	6	6	0	0	6	26.5	20.5
5	4	6	0	0	4	26.5	22.5
6	3	7	4, 5	6	9	29.5	20.5
7	3	10	6	9	12	32.5	20.5
8	8	9	0	0	8	23.5	15.5
9	9	10	3, 8	12	21	32.5	11.5
10	2	11	7, 9	21	23	34.5	11.5
11	0	11	2, 10	23	23	34.5	11.5

A. SaS

Initially, all processors are assumed to be operating at a maximum voltage in the voltage set V = {5 V, 3.3 V, 2.4 V}. The task graph and each task in the task graph meet their respective deadlines when operated at the maximum voltage. Another assumption is that each processor runs at a preset voltage, and voltage switching between tasks is not supported. In the first step, tasks with zero execution time are eliminated, and the tasks are sorted with respect to their slack. Next, voltage scaling is applied to the tasks. For the example, task T_1 has the least slack ($s_1 = 11.5$), and has an initial execution time, $e_1 = 9$. The new execution time $\hat{e}_1$ of the task T_1 at voltage 2.4 V is determined as $\hat{e}_1 = e_1 \times (5/2.4) = 18.75$. The start and end times ($\hat{g}_j$ and $\hat{h}_j$) of all successor tasks T_k of T_1, given as $T_k = \{T_2, T_3, T_9, T_{10}\}$, are updated with respect to the new deadline of task T_1, and evaluated to see if they meet their respective deadlines. The above process is called voltage scaling. For the voltage scaled task T_1, none of the successor tasks (T_k) misses its deadline, and hence the task T_1 is assigned a voltage of 2.4 V. If any task misses its deadline, then task T_1 is scaled to a voltage greater than 2.4 V. This process is repeated for voltages greater than 2.4 V, and checked to satisfy deadlines. This procedure is repeated for all tasks in the task graph.

Voltage scaling of all the tasks results in partitioning the tasks into three partitions, where each partition contains tasks operating at a specified voltage. An earliest deadline first scheduling is performed on the task graphs. If a task is to be scheduled at a specific voltage and the processor operating at that specific volatge is busy, a new processor is added to the resource. The schedule developed by the SaS algorithm is shown in Fig. 2. The total number of processors required by the SaS algorithm is 6.

B. SbS

The SbS algorithm performs scheduling of the tasks on the available processor, before scaling the task voltages. The algorithm proceeds by first scheduling the tasks based on the EDF scheduling algorithm. The voltage scaling succeeds scheduling by scaling each task in each processor, and determines if the following conditions are satisfied: the current task or any of its successors meet their deadlines, or any of the tasks that are executing on the same processor as the current task do not miss their deadlines. It is important to note that when a task is scaled, all the tasks on that processor are also scaled to the same voltage, because it is assumed that a processor can operate at that specified voltage. As shown in Fig. 3, the SbS algorithm starts by choosing processor P_1 and task T_1. The task T_1 is voltage scaled to the voltage 2.4 V, and checked to see if it satisfies its deadline. If the deadline of task T_1 is satisfied, the start and end times of all its successor tasks $\{T_2, T_3, T_9, T_{10}\}$ are also modified for their start time and end time. If any task in this processor misses its deadline, task T_1 is scaled to a higher voltage, and their execution times are checked, to see if all tasks satisfy their deadlines. If deadlines are satisfied, the task is operated at that voltage. This procedure is repeated on all the tasks in that processor. The total number of processors required by the EDF algorithm is 4. Fig. 3 shows the final schedule obtained using the SbS algorithm, where the processor P_1 operates at 5 V, and all other processors are scaled to operate at 2.4 V.

C. PbS

The first step in this algorithm is to determine the BCT and WCT schedules for the task graph. Since the final deadline of the task graph is $\lceil 34.5 \rceil = 34$, the whole graph is divided into 34 time steps (Δt_j). The bound limit for each task (Δt_i) is determined as shown in Equation (10).

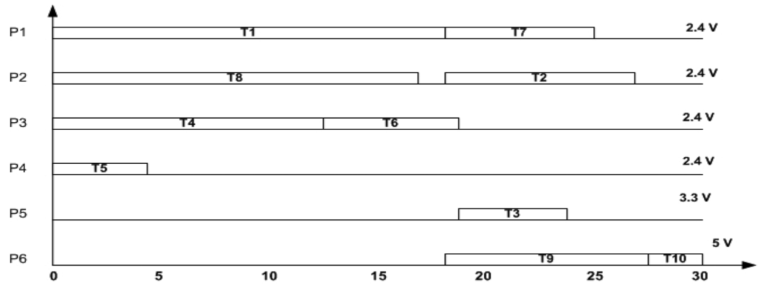

Fig. 2. Scheduling after scaling algorithm execution for the sample task graph.

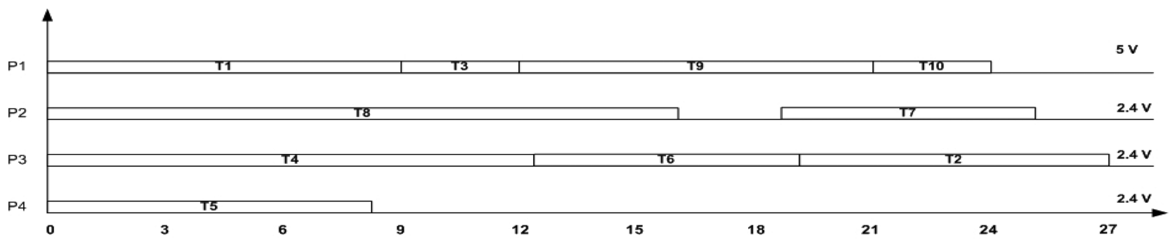

Fig. 3. Scheduling after scaling algorithm execution for the sample task graph.

Table 2a shows the bound limit of each task, along with the BCT and WCT start times (g_{BCT} and g_{WCT}) of each task. For task T_i, the task has the earliest start time $\Delta t_{1j} = 0$, and the latest start time Δt_{1j} as 11. The next step is to determine the probabilities of utilization of each task in each time step from Δt_{11} to $\Delta t_{111} + e_1 = 20 = d_1$, where $e_1 = 9$. These probabilities are determined by using Equation (11). For the example, the probabilities of utilization ($PU(T_1(\Delta t_j))$) for task T_1 within the range Δt_{1j}, j = 0 to 20 are given in Table 2b. After determining the probabilities of utilization of each task in each time step, the next step is to find the distributed probabilities of utilization ($DPU(\Delta t_{1j})$) for each time step. This is done by calculating the sum of probabilities of utilization for all task occurring in each time step. Table 2c lists the distributed probabilities of utilization for time step Δt_{i1}. Then, the $CCPU(T_i)$ of the task is determined by scheduling the task in a control step, and finding the variation in probability of utilization, as given by Equations (12) and (13). The $TCPU(T_j)$ is again the summation of all $CPU(T_i)$ of a task and its successors $CPU(T_j)$. Due to the high volume of data, the $CPU(T_i)$ and $CPU(T_k)$ are omitted. The task T_j that has the highest value of TCPU in time step Δt_{1j} is scheduled in that current time step. The probabilities of utilization of subsequent time steps and tasks are updated, and the above processes repeated for sucessor tasks. Throughout this procedure, the tasks are assumed to be running at 5 V.

The next major step is to scale the voltages of the tasks in such a way that neither the task, nor its successors, miss their respective deadlines. The schedule obtained so far assumes that the tasks are assigned 5 V. Each task in this schedule is then voltage scaled, as described in the SaS algorithm. The last and final step in this algorithm is to determine the number of processors required to accommodate all the tasks in the schedule. This is accomplished as explained in step 11 of Section IV-D. The processors are assumed to be operating at only one particular voltage, without the ability to shift voltages. The final schedule for the benchmark program 10.stg is shown in Fig. 4a.

D. E-PbS

The CPU for each task and its sucessor in time step Δt_{i1} are evaluated as shown in step 10 of Section IV-D. The TCPUEs of a task at 5.0 V (V_1), 3.3 V (V_2) and 2.4 V (V_3) for the task T_i are determined as shown in Equation (16), and are represented as $TCPUE(T_{iV1})$, $TCPU(T_{iV2})$, and $TCPUE(T_{iV3})$, respectively. The voltage factor V_r^2 for the above three cases is given as V_1^2/V_1^2, V_2^2/V_1^2, and V_3^2/V_1^2, respectively. The algorithm schedules a task at a voltage in the time step where the TCPUE is minimal. After a task is assigned a voltage, the affected distributed probabilities of utilization are updated. This process is repeated for all the tasks, and each one is scheduled for execution at a particular voltage; and the TCPUE is

Table 2a. BCT and WCT start time and bound limit of each task

T_i	e_i	WCT	BCT	Δt_i	d_i
[1]	9	1	1	12	20
[2]	4	31	10	22	34.5
[3]		1	10	12	23.5
[4]	6	21	1	21	26.5
[5]	4	23	1	23	26.5
[6]	3	27	7	21	29.5
[7]	3	30	10	21	32.5
[8]	8	16	1	16	23.5
[9]	9	24	13	12	32.5
[10]	2	33	22	12	34.5

WCT: worst-case time, BCT: best-case time.

Table 2b. Distribution probabilities for task T_1 within the bound limit Δt_{1j}

Δt_{1j}	PD
1	0.08
2	0.17
3	0.25
4	0.33
5	0.42
6	0.50
7	0.58
8	0.67
9	0.75
10	075
11	0.75
12	0.75
13	0.67
14	0.58
15	0.50
16	0.42
17	0.33
18	0.25
19	0.17
20	0.08

Table 2c. $CCPU(T_1)$ for task T_1 at time step Δt_{i1}

T_i	Probability
[1]	0.08
[4]	0.05
[5]	0.04
[8]	0.06
CCPU	0.24

CCPU: cumulative probability of utilization.

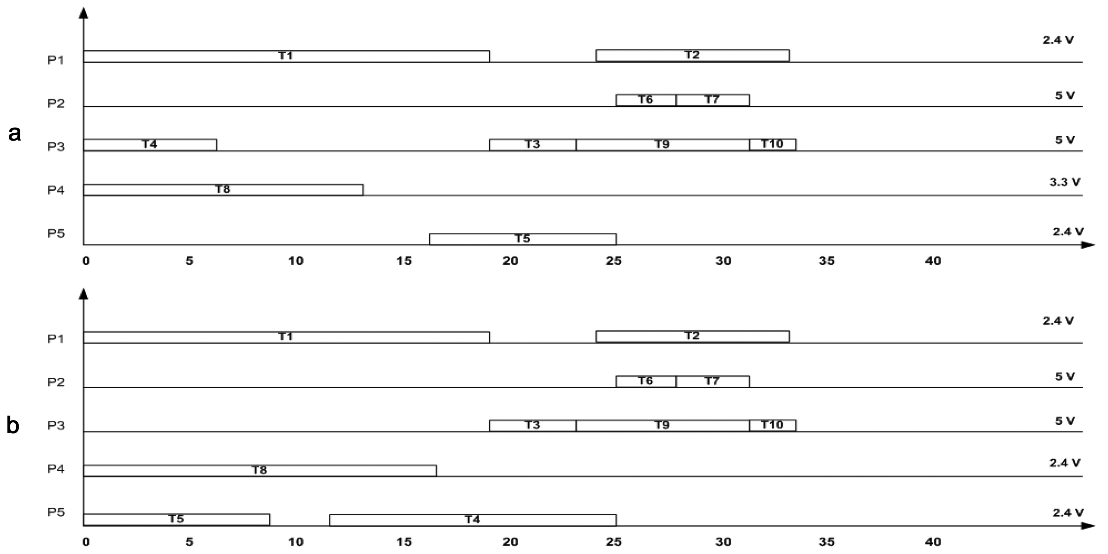

Fig. 4. (a) Probability based scheduling algorithm (PbS) execution for the sample task graph and (b) energy-PbS algorithm execution for the sample task graph.

updated dynamically, to get an optimized schedule. The final schedule obtained by E-PbS is shown in Fig. 4b.

VI. RESULT

Processors supporting several supply voltages are available. Various supply voltages result in different energy consumption levels for a given task T_i. The voltage at which a task is run can be decided before the execution of the task, or sometimes, when the task is executing. In the former method, a particular voltage is assigned to a task, and it is run at that voltage on the processor. In the latter case, the switching of the voltage while the tasks are running on the processor is controlled by the OS. These variable voltage processors operate at different voltage ranges, to achieve different levels of energy efficiency.

All the four scheduling algorithms have been implemented in C++. The STG set were used to evaluate the performance of the above algorithms. The STG benchmark suite contains both synthetic, and practical real-time application task graphs. The assumptions followed during evaluation are as below:

- All the processors operate at a pre-specified set of voltages.
- All the processors operate at a constant and identical frequency.
- Processors cannot shift from one voltage level to another while executing a task set. This constraint allows us to ignore overheads, such as processor voltage transition times, transition energy, states, and steps [24] during scheduling.
- The switching capacitance of the processors is constant, and is the same for all the processors.
- All the tasks are assumed to be non-preemptable, aperiodic, dependent tasks.

Table 3. Number of processors required for scheduling

Benchmark	#T	SaS				SbS				PbS				E-PbS			
		5 V	3.3 V	2.4 V	Total	5 V	3.3 V	2.4 V	Total	5 V	3.3 V	2.4 V	Total	5 V	3.3 V	2.4 V	Total
7.stg	7	1	1	3	5	1	0	3	4	2	2	2	6	2	1	3	6
10.stg	10	1	1	4	6	1	0	3	4	2	1	2	5	2	0	3	5
50.stg	50	2	1	13	16	1	0	14	15	7	3	6	16	6	3	7	16
sparse.stg	96	1	1	62	64	1	0	61	62	12	3	44	59	14	0	47	61
100.stg	100	3	5	46	54	2	3	41	46	19	7	15	41	20	3	18	41
300.stg	300	1	0	120	121	1	0	122	123	49	15	57	121	53	8	64	125
500.stg	500	6	4	205	215	2	2	201	205	79	33	87	199	91	20	96	207

SaS: scheduling after scaling, SbS: scheduling before scaling, PbS: probability based scheduling, E-PbS: energy-probability based scheduling.

Table 4. Average power consumed by each processor in nWatts for the scheduling algorithms

Benchmark	#T	SaS				SbS				PbS				E-PbS			
		5.0 V	3.3 V	2.4 V	Total	5.0 V	3.3 V	2.4 V	Total	5.0 V	3.3 V	2.4 V	Total	5.0 V	3.3 V	2.4 V	Total
7.stg	7	125.00	181.17	68.00	102.03	450.00	-	64.00	160.50	150.00	130.68	46.08	108.92	150.00	152.46	55.68	103.25
10.stg	10	275.00	49.50	101.64	121.84	575.00	-	112.00	227.75	325.00	141.57	106.56	200.94	250.00	-	128.64	177.18
50.stg	50	487.50	372.90	156.41	211.32	1375.00	-	177.43	257.27	471.43	254.10	182.40	322.29	475.00	214.17	201.60	306.48
sparse.stg	96	1500.00	370.26	345.79	364.20	3050.00	-	356.85	400.29	1387.50	471.90	325.70	549.09	1107.14	-	338.25	514.71
100.stg	100	333.33	65.34	116.06	123.43	600.00	348.26	126.44	161.50	360.53	272.25	137.09	263.71	338.75	119.79	177.92	252.12
300.stg	300	1325.00	-	259.90	268.71	2350.00	-	291.15	307.89	838.78	294.76	247.48	492.79	663.21	201.47	298.44	446.90
500.stg	500	600.00	193.30	254.00	262.52	2075.00	1115.90	294.63	320.01	906.33	337.92	243.38	522.24	689.29	209.63	318.60	471.03
Average power (PE)		663.69	205.41	185.97	**207.72**	1496.43	732.08	203.21	**262.17**	634.22	271.88	184.10	**351.43**	524.77	179.50	217.02	**324.53**

SaS: scheduling after scaling, SbS: scheduling before scaling, PbS: probability based scheduling, E-PbS: energy-probability based scheduling.

Table 5. Utilization of the processors for the scheduling algorithms

Benchmark	#T	SaS						SbS						PbS						E-PbS					
		5.0 V	Util	3.3 V	Util	2.4 V	Util	5.0 V	Util	3.3 V	Util	2.4 V	Util	5.0 V	Util	3.3 V	Util	2.4 V	Util	5.0 V	Util	3.3 V	Util	2.4 V	Util
7.stg	7	1	0.99	1	0.62	3	0.44	1	0.67	0	0.00	3	0.41	2	1.00	2	0.80	2	0.67	2	1.00	1	0.61	3	0.78
10.stg	10	1	0.98	1	0.29	4	0.51	1	0.67	0	0.00	3	0.56	2	0.65	1	0.39	2	0.68	2	0.84	0	0.00	3	0.67
50.stg	50	2	0.76	1	0.70	13	0.33	1	0.67	0	0.00	14	0.38	7	0.47	3	0.52	6	0.42	6	0.72	3	0.79	7	0.43
sparse.stg	96	1	0.63	1	0.31	62	0.33	1	0.67	0	0.00	61	0.34	12	0.54	3	0.31	44	0.90	14	0.95	0	0.00	47	0.90
100.stg	100	3	0.82	5	0.52	46	0.43	2	0.52	3	0.69	41	0.47	19	0.57	7	0.65	15	0.76	20	0.92	3	0.95	18	0.85
300.stg	300	1	0.95	0	0.00	120	0.32	1	0.67	0	0.00	122	0.36	49	0.57	15	0.42	57	0.75	53	0.99	8	0.95	64	0.76
500.stg	500	6	0.40	4	0.30	205	0.31	2	0.58	2	0.72	201	0.36	79	0.61	33	0.47	87	0.69	91	0.94	20	0.89	96	0.78

SaS: scheduling after scaling, SbS: scheduling before scaling, PbS: probability based scheduling, E-PbS: energy-probability based scheduling.

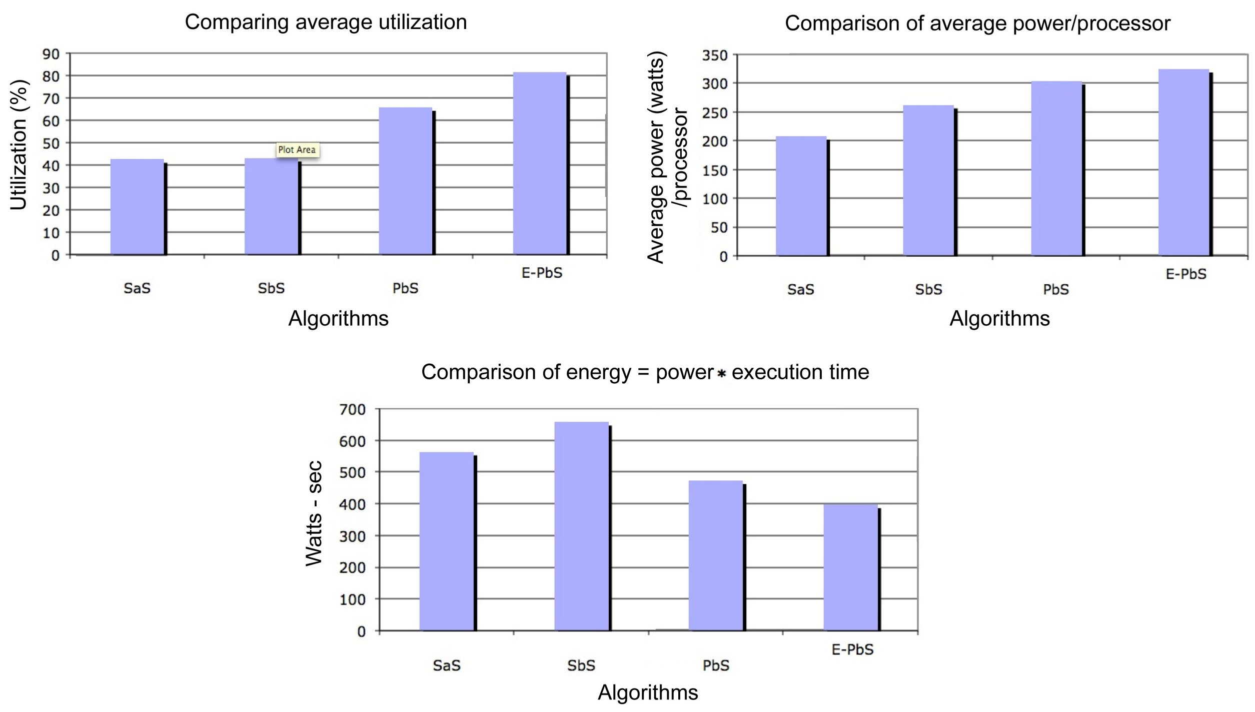

Fig. 5. Comparison of utilization, average power/processor consumption, and total average energy consumption for the SaS, SbS, PbS and E-PbS algorithms.

The performance of the SaS, SbS, PbS, and E-PbS is summarized below, based on the results obtained for the various benchmarks, as shown in Tables 3–5. From Table 3, we conclude that the SbS algorithm results in the smallest required processors to complete the task graphs. However, the SaS and SbS algorithms have more processors operating at 2.4 V (lowest operation voltage), and hence will have larger completeion times for the task graphs. Almost 90% of the processors operate at this voltage. On the other hand, the PbS and E-PbS algorithms have processors equally operating at all voltages (distributed almost uniformally). Table 4 shows the average processor power consumed by each algorithm (SaS = 208, SbS = 262, PbS = 351, E-PbS = 324), for each benchmark. The following conclusions are derived from Table 4: the SaS and SbS algorithms have lesser average processor power consumption (≈25% less) compared to the PbS and E-PbS algorithms. Table 5 shows the processor utilization (SaS = 42%, SbS = 43%, PbS = 65%, E-PbS = 81%) achieved by the algorithms for the various benchmarks. In this case, the PbS and E-PbS algorithms outperform (≈35%) the SaS and SbS algorithms. Typically, the SaS algorithm results in lower power consumption and higher completion times. This is evident by the large number of processors and tasks operating at 2.4 V (Table 3). Similarly, the SbS algorithm results in a smaller number of processors, and better completion times. The proposed PbS and E-PbS algorithms results in higher processor utilizations, but with smaller completion times (Table 5). Hence, the PbS and E-PbS algorithms results in smaller energy consumption (average processor power × completion times (or utilization)), as shown in Fig. 5. In this section we provide a quick time complexity analysis of the various algoirthms implemented in this work. Let '*n*' be the number of tasks, '*m*' the number of processors, and '*k*' the number of voltage levels of the processors. Also, for the PbS and E-PbS scheduling algorithms, let 'Δt' be the fixed time interval for which the probability distibutions are determined. Given the above variables, the worst case complexity of the SaS and SbS algorithms can be derived as $O(knlog_2n)$ and $O(knm)$, respectively. Similarly, the complexity of the PbS and E-PbS algorithms can be determined as $O(\Delta tn^3)$ and $O(\Delta tk^2n^3)$, respectively.

VII. CONCLUSIONS

Scheduling of applications or task graphs in multiprocessor or multicore systems is an important research question that needs to be addressed, in light of today's emerging embedded system demands. It is often necessary to develop scheduling and task mapping algorithms that optimize processor utilization and processor power consumption. This work proposes an integrated scheduling and mapping algorithm, called PbS, that optimizes the above requirements. The proposed PbS algorithm, and its variation, called E-PbS algorithm, are based on the force directed scheduling algorithm used in high-level synthesis of digital circuits. The algorithms were implemented and tested with synthetic and real application task graphs (benchmarks). The proposed algorithms are compared with traditional scheduling approaches like FCFS (SaS) and EDF (SbS). The SaS and SbS algorithms also implement an explicit mapping, or voltage scaling algorithms, after and before scheduling, respectively. The algorithms are compared for processor utilization, average power consumed per processor, and the number of processors required to implement the given set of task graphs. Based on the results discussed in Section VI, we can conclude that the PbS and E-PbS algorithms provide better processor utilization and lower energy consumption, compared to the SaS and SbS algorithms.

REFERENCES

1. S. Zhuravlev, J. C. Saez, S. Blagodurov, A. Fedorova, and M. Prieto, "Survey of energy-cognizant scheduling techniques," *IEEE Transactions on Parallel and Distributed Systems*, 2012. http://doi.ieeecomputersociety.org/10.1109/TPDS.2012.20.
2. P. G. Paulin and J. P. Knight, "Force-directed scheduling in automatic data path synthesis," in *Proceedings of the 24th ACM/IEEE Design Automation Conference*, Miami Beach, FL, 1987, pp. 195-202.
3. J. W. S. Liu, Real-Time Systems, Upper Saddle River, NJ: Prentice Hall, 2000.
4. A. Manzak and C. Chakrabarti, "Variable voltage task scheduling algorithms for minimizing energy/power," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 11, no. 2, pp.270-276, 2003.
5. M. Weiser, B. Welch, A. Demers, and S. Shenker, "Scheduling for reduced CPU energy," in *Proceedings of the 1st USENIX Conference on Operating Systems Design and Implementation*, Monterey, CA, 1994, pp. 13-23.
6. F. Yao, A. Demers, and S. Shenker, "A scheduling model for reduced CPU energy," in *Proceedings of the 36th Annual Symposium on Foundations of Computer Science*, Milwaukee, MI, 1995, p. 374-382.
7. T. Ishihara and H. Yasuura, "Voltage scheduling problem for dynamically variable voltage processors," in *Proceedings of the International Symposium on Low Power Electronics and Design*, Monterey, CA, 1998, pp. 197-202.
8. H. Aydin, R. Melhem, D. Mosse, and P. Mejia-Alvarez, "Dynamic and aggressive scheduling techniques for power-aware real-time systems," in *Proceedings of the 22nd IEEE Real-Time Systems Symposium*, London, UK, 2001, pp. 95-105.
9. D. Zhu, R. Melhem, and B. R. Childers, "Scheduling with dynamic voltage/speed adjustment using slack reclamation in multiprocessor real-time systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 14, no. 7, pp. 686-700, 2003.
10. Y. Shin, K. Choi, and T. Sakurai, "Power optimization of real-time embedded systems on variable speed processors," in *Proceedings of IEEE/ACM International Conference on Computer Aided Design*, SanJose, CA, pp. 365-368, 2000.
11. G. Quan and X. Hu, "Energy efficient fixed-priority scheduling for real-time systems on variable voltage processors," in *Proceedings of Design Automation Conference*, Las Vegas, NV, 2001, pp. 828-833.
12. Y. Lee, Y. Doh, and C. M. Krishna, "EDF scheduling using two-mode voltage-clock-scaling for hard real-time systems," in *Proceedings of the International Conference on Compilers, Architecture, and Synthesis for Embedded Systems*, Atlanta, GA, 2001, pp. 221-228.
13. C. Isci, A. Buyuktosunoglu, C. Y. Chen, P. Bose, and M. Martonosi, "An analysis of efficient multi-core global power management policies: maximizing performance for a given power budget," in *Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture*, Orlando, FL, 2006, pp. 347-358.
14. G. Dhiman and T. S. Rosing, "Dynamic voltage frequency scaling for multi-tasking systems using online learning," in *Proceedings of the ACM/IEEE International Symposium on Low Power Electronics and Design*, Portland, OR, 2007, pp. 207-212.
15. M. Kondo, H. Sasaki, and H. Nakamura, "Improving fairness, throughput and energy-efficiency on a chip multiprocessor through DVFS," *ACM SIGARCH Computer Architecture News*, vol. 35, no. 1, pp. 31-38, 2007.
16. N. Takagi, H. Sasaki, M. Kondo, and H. Nakamura, "Cooperative shared resource access control for low-power chip multiprocessors," in *Proceedings of the 14th ACM/IEEE International Symposium on Low Power Electronics and Design*, San Francisco, CA, 2009, pp. 177-182.
17. R. Watanabe, M. Kondo, H. Nakamura, and T. Nanya, "Power reduction of chip multi-processors using shared resource control cooperating with DVFS," in *Proceedings of the 25th International Conference on Computer Design*, Lake Tahoe, CA, 2007, pp. 615-622.
18. G. Dhiman, G. Marchetti, and T. Rosing, "vGreen: a system for energy efficient computing in virtualized environments," in *Proceedings of the 14th ACM/IEEE International Symposium on Low Power Electronics and Design*, San Francisco, CA, 2009, pp. 243-248.
19. Y. C. Lee and A. Y. Zomaya, "Energy conscious scheduling for distributed computing systems under different operating conditions," *IEEE Transactions on Parallel and Distributed Systems*, vol. 22, no. 8, pp. 1374-1381, 2011.
20. S. Cho and R. G. Melhem, "Corollaries to Amdahl's law for energy," *IEEE Computer Architecture Letters*, vol. 7, no. 1, pp. 25-28, 2008.
21. D. Shin and J. Kim, "Power-aware scheduling of conditional task graphs in real-time multiprocessor systems," in *Proceedings of the ACM/IEEE International Symposium on Low Power Electronics and Design*, Seoul, Korea, 2003, pp. 408-413.
22. M. A. Moncusi, A. Arenas, J. Labarta, " Energy aware EDF scheduling in distributed hard real time systems," in *Work-in-Progress session of Real-Time Systems Symposium*, Cancun, Mexico, 2003.
23. T. Okuma, T. Ishihara, and H. Yasuura, "Real-time task scheduling for a variable voltage processor," in *Proceedings of the 12th International Symposium on System Synthesis*, San Jose, CA, 1999, pp. 24-29.
24. L. Shang, L. S. Peh, and N. K. Jha, "Dynamic voltage scal-

ing with links for power optimization of interconnection networks," in *Proceedings of the 9th International Symposium on High-Performance Computer Architecture*, Anaheim, CA, 2003, pp. 91-102.

Naveen Anne

Naveen Anne received the B.E. honors degree from Malaviya National Institute of Technology, INDIA, M.S. degree in EE from University of Nevada, Las Vegas, and Master of Engineering Management from Duke University in 2004 and 2006, respectively. He is currently a senior consultant at BMC Software specializing in designing, implementing and configuring data center automation software solutions for clients of BMC Software.

Venkatesan Muthukumar

Venkatesan Muthukumar received the B.E. degree from College of Engineering, Anna University, INDIA, and the M.S. and Ph.D. degrees from Monash University, Australia, in 1996 and 2000, respectively. He is currently an Associate Professor with the Department of Electrical and Computer Engineering, University of Nevada, Las Vegas. His current research interests include network on chip, high performance computing on system on chips, multicores and multiprocessor systems.