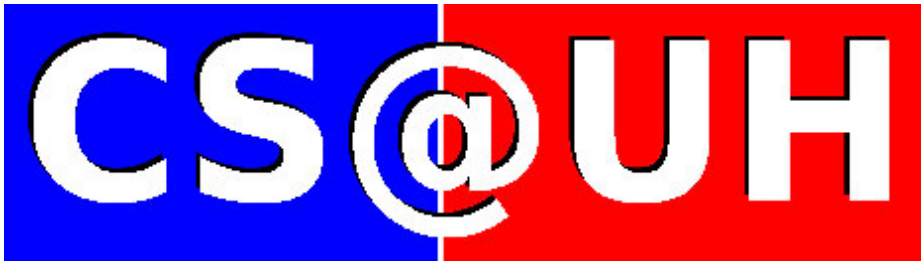

A Dynamic Voltage Scaling Algorithm for Dynamic Workloads

Albert Mo Kim Cheng and Yan Wang

Real-Time Systems Laboratory
Department of Computer Science
University of Houston
Houston, TX, 77204, USA
http://www.cs.uh.edu

Technical Report UH-CS-07-05

May 5, 2007

Keywords: Dynamic voltage scaling (DVS), power-aware computing, scheduling, dynamic workloads, real-time systems.

Abstract

Dynamic Voltage Scaling (DVS) is a promising method to achieve energy saving by slowing down the processor into multiple frequency levels in battery-operated embedded systems. However, the worst case execution time (WCET) of the tasks scheduled by DVS must be known ahead of time to ensure their schedulability. In reality, a system's workloads may change significantly without satisfying any prediction. In other words, a task's WCET may not provide useful information about its future real execution time (RET). This paper presents a novel Dynamic-Mode EDF scheduling algorithm when workloads change significantly. One of the Single-Mode, Dual-Mode, and Three-Mode frequency setting formats can be applied, based on the RET and the accumulated slack at run-time. Only one combination of the number of modes/speeds, speed-switching transition points, and the frequency scaling factor for each mode can lead to the best energy saving. Experimental results show that, given an RET pattern, our Dynamic-Mode DVS algorithm achieves an average 15% energy savings over the traditional two-mode DVS scheme on hard real-time systems. Additionally, we also consider speed-switching or energy transition overhead, and implement a preliminary test of our proposed algorithm. With a less aggressive voltage scaling strategy (fewer speed changes for each job), deadlines can still be strictly satisfied and an average of 14% energy consumption saving over a non-DVS scheme is observed.

A Dynamic Voltage Scaling Algorithm for Dynamic Workloads*

Albert Mo Kim Cheng and Yan Wang
Real-Time Systems Laboratory
Department of Computer Science
University of Houston, TX 77204-3010, USA
Corresponding Email: cheng@cs.uh.edu

Abstract

Dynamic Voltage Scaling (DVS) is a promising method to achieve energy saving by slowing down the processor into multiple frequency levels in battery-operated embedded systems. However, the worst case execution time (WCET) of the tasks scheduled by DVS must be known ahead of time to ensure their schedulability. In reality, a system's workloads may change significantly without satisfying any prediction. In other words, a task's WCET may not provide useful information about its future real execution time (RET). This paper presents a novel Dynamic-Mode EDF scheduling algorithm when workloads change significantly. One of the Single-Mode, Dual-Mode, and Three-Mode frequency setting formats can be applied, based on the RET and the accumulated slack at run-time. Only one combination of the number of modes/speeds, speed-switching transition points, and the frequency scaling factor for each mode can lead to the best energy saving. Experimental results show that, given an RET pattern, our Dynamic-Mode DVS algorithm achieves an average 15% energy savings over the traditional two-mode DVS scheme on hard real-time systems. Additionally, we also consider speed-switching or energy transition overhead, and implement a preliminary test of our proposed algorithm. With a less aggressive voltage scaling strategy (fewer speed changes for each job), deadlines can still be strictly satisfied and an average of 14% energy consumption saving over a non-DVS scheme is observed.

Keywords: Dynamic voltage scaling (DVS), power-aware computing, scheduling, dynamic workloads, real-time systems.

1. Introduction

A hard real-time system is usually characterized as static or dynamic. If the system's resource requirements do not change with the variation in the environment, it is called a static real-time system. Otherwise, if the resource requirements change significantly and unpredictably at run-time, it is called a dynamic real-time system. The characterization of a specific real-time system should be based not only on the static properties but also on the run-time execution behavior of the system.

Traditionally, in most hard real-time scheduling models, the worst-case execution time (WCET) of a task is an important factor, and should be known as a static integer in order to ensure the schedulability of the system. However, in reality, resource requirements change significantly and unpredictably in a dynamic-workload real-time system. The fluctuating execution time at run-time induces a significant difference between the real and estimated values of the Worst Case Execution Time (WCET). Tasks scheduling should be based on the run-time behavior under such system. Both pessimistic and optimistic estimation values can cause performance degradation and high power consumption in processors employing voltage scaling.

With the aim of obtaining more precise estimate of a task's real execution time (RET), giving more diversification to the scaling frequency adjustment, we present a Dynamic-Mode DVS algorithm for battery-powered real-time systems. A task is split into "three" virtual subtasks. The first two subtasks are assigned to different frequency scaling factors which are proportional to each other by a frequency variant n. It is this frequency variant n which decides the exact number of run-time frequency modes. Also, the third task is always executed with the maximum frequency. At run-time, the frequency scaling factor is adjusted based on S_k, the accumulated slack time and the previous job's RET. Obviously, the executing frequency in the dynamic-mode DVS algorithm will be changed more frequently than dual-mode, which provides much more flexible opportunity for potential energy

*Submitted to Journal of VLSI Signal Processing. A 4-page preliminary version of this work was presented at the IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS) 2005. This work is supported by the Institute for Space Systems Operations.

saving in the system. Furthermore, we attempt to obtain close-to-optimal energy saving by changing the length of different frequency zones (marked by speed change or transition points). Only a combination of a certain frequency scaling factor and the length of a frequency zone assigned to each frequency level can result in the best total energy saving.

The rest of the paper is organized in the following way. Section 2 presents the previous works related to DVS, and reviews several basic theorems used in our work. In section 3, we present our system model, including assumptions, dynamic-mode task splitting, and frequency scaling factor analysis. Our dynamic-mode DVS algorithm is then described. Section 4 shows the experimental results, and analyzes the energy consumption under different workload patterns. The conclusion and a discussion of potential improvement are given in the last section.

2. Previous Works

We review several previous key works on dual-mode voltage scaling, dynamic slack reclaiming, and feedback dynamic voltage scaling in this section.

2.1. Dual-mode Voltage Scaling

The basic source of power consumption in a digital CMOS circuit can be characterized using the following formula,

$$P_{cmos} = C_L * V^2_{DD} * f \quad (1)$$

where P_{cmos} is the dynamic power dissipation, C_L is the effective switched output capacitance, and f is the clock frequency. Power consumption increases proportionally to the processor frequency and to the square of the voltage in the CMOS circuit [7, 8]. Based on this equation, a voltage-scheduling technique was introduced to reduce CPU energy for general operating systems by Weiser, Welch, Demers, and Shenker [2], and the simulation and evaluation of Dynamic Voltage Scaling Algorithms was achieved by Pering, Burd, and Brodersen [11]. In all of these works, future workload is predicted based on past history, and appropriate voltage levels are selected based on such predictions.

The first to introduce the concept of real-time scheduling and also to apply it to a dynamic speed setting is Pering et al [12]. Combining voltage-scaling and scheduling algorithms together in real-time systems became a popular way to reduce CPU power consumptions. From that time, DVS [4, 5, 6] has become a promising method for real-time/embedded systems to save energy by allowing multiple voltage levels. In [13], they present a class of novel algorithms called real-time DVS (RT-DVS) that modify the operating system's real-time scheduler and task management service to provide significant energy savings while meeting real-time deadlines. In [3], Lee et al proposed a dual-mode assignment, Low-voltage (L)-mode and High-voltage (H)-mode, which has a flexible voltage mode setting at run-time enabling much larger energy saving. They made the assumption that the WCET of each task is known in advance.

2.2. Feedback-DVS Framework

Being aware of the highly unpredictable value of a task's worst case execution time, many works have been done to achieve a more accurate prediction of the WCET. Most of the improvements are partially related to the feedback control for the real-time scheduler.

In Zhu and Mueller's recent paper [1], they use feedback control techniques on DVS for hard real-time systems. Basically, there are 3 parts in their framework. One is the voltage frequency selector. It is introduced to calculate the error from the difference between the actual execution time of a task and the first portion of task splitting (task splitting scheme), and then choose a frequency/voltage level according to the error. Another part is a PID (Proportional Control, Integral Control, and Derivative Control) feedback controller, which is used to adjust the next assignment of the execution time for the next job. The next ready tasks are scheduled by the EDF Scheduler. After executing the job, its real execution time is fed back to the voltage frequency selector for the next job's frequency scaling decision making.

If the tasks have truly random execution times, the former jobs cannot provide any useful history information on the feedback controller. The feedback-DVS scheme does not provide any additional benefit from the feedback, thus the nature of the feedback technique has its limitation.

2.3. Why Three-Mode is Needed under Dynamic Workloads

Ishihara and Yasuura [10] proved that for a processor with few multiple discrete voltages, at most two voltages minimizes the energy consumption for a task or schedule with deadline. (Suppose V_{ideal} is the voltage which minimizes the energy consumption for this task. The task runs at a voltage below V_{ideal} up to a point which can be statically determined, and then it runs at a voltage higher than V_{ideal}.) However, this two-voltage theorem is true only for the static case in which the actual execution time of the task is known in advance. For tasks with unknown actual execution times, this theorem is not true since V_{ideal} cannot be determined before the task completes its actual execution. Therefore, our work proposes more than two voltages for running the task to better adapt to its fluctuating execution times and hence reduces energy consumption as much as possible.

We can exhibit several scenarios in which we need 3 or more voltages in order to approximate minimum energy consumption for tasks with dynamic and fluctuating actual execution times. One such scenario is given next. If we know a task's actual execution time, then according to [10], at most two voltages V_1 and V_2 are needed to minimize energy consumption, where $V_1 < V_2$. The task runs at V_1 until time instant P, and then runs at V_2 till its completion, where $V_1 < V_{ideal} < V_2$. If we do not know in advance this task's actual execution time, then only a clairvoyant scheduler would know the time instant P. However, since such a scheduler does not exist, P has to be guessed or estimated. The chosen P, denoted P', may be $< P$, $= P$, or $> P$. If $P' < P$, then the task would finish before its deadline. If we notice this after running the task using V_2, we can switch to a lower voltage after an interval to be determined, hence 3 or more voltages may be needed. If $P' > P$, then the task would miss its deadline, so we would switch to a higher voltage after a while, hence again 3 or voltages may be needed.

The recently proposed GRACE-OS [20] also allows multiple speeds (1, 2, 3 or more) for each job. It makes scheduling decisions based on the probability distribution of the tasks' cycle demands, and derives the tasks' demand distribution by online profiling and estimation. However, GRACE-OS is targeted toward soft real-time multimedia systems and misses on average 4% of the deadlines [20] whereas our approach is aimed at hard real-time systems, ensuring that all deadlines will be satisfied. An approach called PACE [4, 21] also increases the processor speed as a task progresses in its execution, but again this approach is targeted for soft real-time systems and can result in the missing of deadlines [22]. Furthermore, neither GRACE nor PACE can deal with speed-switching overhead according to [22] whereas our dynamic approach can, as shown in section 4.3.

3. System Model

Induced by the unpredictable nature of the actual execution times of general real-time tasks, a significant power consumption gap exists between the chosen schedule and the optimal energy-saving schedule. Because of this difference, if we schedule and adjust frequency based only on the estimated value of WCET, dynamic voltage scheduling will not achieve or even approach the ideal power saving result that we aim for. Under such condition, the research problem facing us is how to obtain a much closer estimate of the current real execution time given the previous job's real execution time. Is there an actual optimal solution for this problem?

This paper shows our current approach to reduce energy consumption by using a novel dynamic-mode speed adjustment method for scheduling dynamic-workload real-time systems. This is the first attempt which takes the Three-Mode Task Splitting idea into consideration in a hard real-time system where all task deadlines must be satisfied. The recently developed GRACE-OS [20] and PACE [21] also allow three or more modes for each job, but they are targeted for soft real-time multimedia systems where deadlines are allowed to be missed. Without making any prediction for the future workload, we make use of the RET of the previous job and the reclaimed slack time as a parameter to further adjust the frequency scaling factor.

The main difference between our work and the feedback EDF scheduling in [1] is:

1. Three-Mode Task Splitting is defined virtually and may not be fixed at run-time. How many subtasks the next job will be split into is totally decided by the ratio of RET and the estimated values of the previous job;
2. In our work, not only the frequency scaling factor, but also the length of each frequency zone (interval between two speed transition points) must be determined in order to minimize the total energy consumption. Only a specific assignment of the combination of the frequency scaling factor and the length of each frequency zone can lead to a proposed amount of energy saving;
3. Besides the frequency scaling factor, we also take the estimation precision of the previous job into consideration, and use it as an important factor to adjust the frequency scaling for the next job. The higher the estimation precision is, the smaller the next job frequency setting changes will be. In our proposed algorithm, a ratio is used to describe the difference between the estimation and the RET values, and we define it as n.

3.1. Assumptions

We consider (1) hard real-time systems, that is, all deadlines must be strictly satisfied. (2) Only the task execution and idle cycles are modeled. Voltage switching overhead, preemption, and task context-switch overhead are first considered negligible.

This is our original assumptions when proposing this algorithm, but since virtual three-mode task splitting is introduced, more frequent voltage scaling (speed change) is expected when compared to dual-mode, but fewer speed changes than the more-than-3 possible changes in GRACE-OS [20], PACE [21], and PPACE [22]. Even though our current algorithm is not inspired by reducing energy consumption while accounting for this speed change or energy transition overhead, a preliminary testing has been performed to show that our strategy performs better than without DVS while taking into account the speed-switching overhead.

(3) Task set: Tasks are mutually independent;

Fully preemptive: a task can be interrupted and resumed at any time;

Periodic tasks: tasks arrive at the start of each fixed period;

Arrival time: All tasks arrive at time 0; the arrival time is fixed;

(4) Supply voltage and clock speed continuously change within its operational ranges;

(5) Processor's speed can be changed continuously within a certain range. Note that the voltage and speed can be rounded up to the nearest higher discrete voltage or speed just like in GRACE [20] but unlike GRACE, our approach does not derive a frequency higher than the largest discrete frequency.

3.2. Energy Saving Strategy

The DVS technique is based on the fact that task computations usually finish before their predicted worst case workload. The period of time, beginning from the job's real completion time to the completion time predicted by its WCET, is represented as Slack (Sk). The slack time arises from the following: the utilization of the processor is less than 1; the run-time execution time deviates from WCET; and intentionally dropping tasks. DVS can utilize these slack times when adjusting the voltage levels. Energy saving can be achieved by lowering the processor speed. For this reason, the energy efficiency of a real-time DVS algorithm largely depends on the estimated value of the slack.

Dynamically monitoring and reclaiming "unused" computation time is a foundation of the DVS technique. DVS also helps to minimize the effects of conservative predictions of the actual execution time of the scheduled tasks by the WCET information [6]. There are different abstraction levels of power saving:

Task set/system level (Inter-task level) [14]: The operating system invokes the Power Management Points (PMPs) at context-switch times, taking advantage of the global knowledge such as the system-wide workload.

Task level (Intra-task level) [15]: User or compiler-inserted PMPs invoke intra-task monitoring of the execution and control the speed of a given task to improve the energy savings.

We choose to use the intra-task level DVS strategy in our proposed Dynamic-Mode DVS Algorithm. A task can use the valid unused time left by previous jobs, but the voltage scaling format is only decided by the previous job in the same task. In our algorithm, it is also necessary to minimize the frequency scaling factor as much as possible during the adjustment procedure of frequency assignment. For further analysis on how Sk affect the assignment of the frequency scaling factor, please see section 3.5 for details.

3.3. Dynamic-Mode Task Splitting

First , we use

$$\alpha = f_i / f_m \qquad (2)$$

to define the frequency scaling factor [10]: where f_i represents the scaled frequency and f_m represents the maximum frequency. We can see clearly that the frequency scaling factor should be within a range of [0,1].

Assuming that the WCET is C_i when the task is executed at the maximum frequency f_m, Sk is the slack time accumulated by only the previous finished tasks. Task splitting, shown in Figure 1, is done based on the estimated WCET plus the total reclaimed slack time.

Under maximum frequency (without scaling):

$$Ci = C_l + C_m + C_h \qquad (3)$$

Under dynamic voltage scaling (with scaling):

$$C_l/\alpha_1 + C_m/\alpha_2 + C_h/1 = Ci + Sk \qquad (4)$$

Notations:

Ci: the Worst Case Execution Time of the entire task under maximum frequency

f_m: the frequency before dynamic voltage scaling

Sk: the accumulated slack time available to task T

T_l,T_m,T_h: the subtask of task T after splitting
C_l,C_m,C_h: the WCET of three subtasks at maximum frequency

Based on the above notations, we split a task into a virtual Three-Mode Model. Each of its subtasks will be executed at a different frequency, shown in Figure 1.

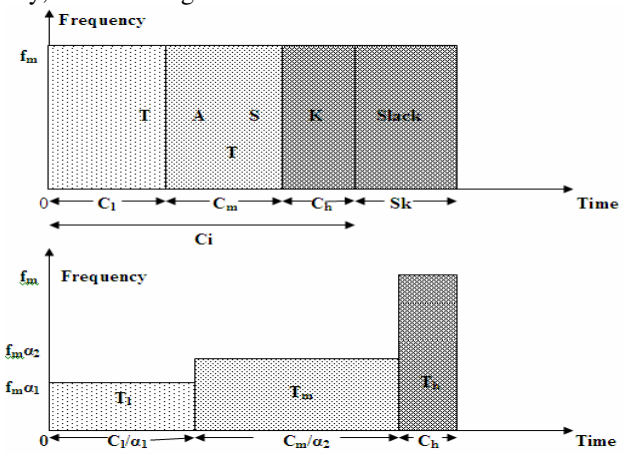

Figure 1. Dynamic-Mode Task Splitting

3.4. Frequency Scaling Factor Setting

Our idea of the frequency format setting is based on three thoughts:

1. Guarantee time constraints;
2. Introduce run-time diversification on frequency scaling;
3. Ensure scaling factor <= 1. It is a crucial definition requirement that we must take a look when solving for α in equation (4). Based on the definition of α <= 1, each scaling factor (α_1 and α_2) must have a value no larger than 1.

Having known the constraints and frequency setting thoughts, each frequency assignment format is depicted in Figure 3. All of the following conditions are based on:

$$\alpha_2 C_l + \alpha_1 C_m = \alpha_1\alpha_2 C_l + \alpha_1\alpha_2 C_m + \alpha_1\alpha_2 Sk \quad (5)$$

Equation (5) not only guarantees each job instance's deadline satisfaction, but also separates various types of frequency assignment formats.

Condition (1):

Assume that $\alpha_1 = \alpha_2 = \alpha$ in the above equation, then we can obtain

$$(C_l + C_m + S_k)\alpha^2 - (C_l + C_m)\alpha = 0$$

$$\alpha = (C_l + C_m)/(C_l + C_m + S_k) \quad (6)$$

This condition actually changes the three-mode task splitting format into the two-mode, or even the one-mode when S_k=0. That's the reason why our three-mode task splitting is virtually executed. There is a special case where a task is running under the one-mode situation: no task splitting is required, the whole task is executed at the maximum speed without doing any voltage scaling operation. When S_k is a relatively small number, we can either choose to adjust the value of α a little bit lower, or simply maintain the maximum frequency in order to gain as much slack time as possible for the rest of the job.

Condition (2): $\alpha_1 < \alpha_2$ under the constraint $\alpha_2 = n\alpha_1$ (n>1)

$$\alpha_1 = (nC_l + C_m) / (nC_l + nC_m + nS_k) \quad (7)$$

$$\alpha_2 = (nC_l + C_m) / (C_l + C_m + S_k) \quad (8)$$

Please notice the scaling factor definition constraints: $\alpha_2 = n\alpha_1 <= 1$, thus S_k must have a value not less than $(n-1)C_l$. A lower warm-up frequency is followed by a higher frequency that will reserve sufficient time for the task to finish within a certain time constraint.

Condition (3): $\alpha_1 > \alpha_2$ under the constraint $\alpha_1 = n\alpha_2$ (n>1)

$$\alpha_1 = (nC_m + C_l) / (C_l + C_m + S_k) \quad (9)$$

$$\alpha_2 = (nC_m + C_l) / (nC_l + nC_m + nS_k) \quad (10)$$

S_k must have a value not less than $(n-1)C_m$ in order to meet the constraint of $\alpha_1 <= 1$.

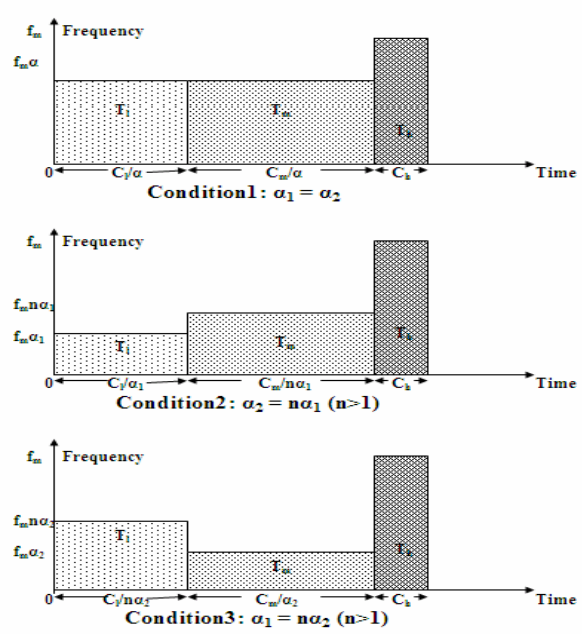

Figure 2. Frequency setting format

After analyzing the above three conditions, we declare from equations (6)-(10) that the suitable frequency scaling factor α depends on the length of the first two time zones: C_l and C_m as well as the accumulated slack time. Another question occurs: What kind of workload is each condition suitable for?

Firstly, when $S_k = 0$, there is a high rate of missing deadline if the frequency scaling is immediately done for the following job. So, for the next job, instead of lowering the frequency, we have the aim of accumulating S_k as much as possible. Condition 1 should be chosen under such situation that S_k is a relatively small number. Secondly, condition 2 offers 3 different frequency levels. As long as the constraint $S_k >= (n-1)C_l$ is satisfied, the frequency is gradually increased based on the factor n, and the deadline is guaranteed to be met. Similarly, condition 3 with the constraint of $S_k >= (n-1)C_m$, the frequency is dropped and then increased based on

3.5. Frequency Setting Idea

There are three cases for the RET:

1. If RET drops into the first time zone, RET and C_i usually have a large estimation difference. Condition 2 is used as the frequency setting format for the next job. At the beginning, a lower frequency is used with a higher chance of possibly shorter execution time occurring. If this is not the case, a higher frequency is used to speed-up the rest of the execution.
2. If RET is within the second time zone, the difference between RET and estimation is small. Condition 3 is used as the frequency setting format for the next job. At the beginning, a larger frequency is used with a higher chance of possibly longer execution time occurring. If this is not the case, the frequency is lowered to achieve energy saving.
3. If RET is within the last time zone, the value of RET and the estimation is very much the same. This job will leave little slack time for others. Condition 1 is used as the frequency setting format for the next job.

S_k is updated and checked upon every completion of jobs. The frequency setting format is changed to f_m as soon as S_k drops to zero. In the full speed running period, we don't care about the length for each time zone. The only purpose under this condition is to accumulate as much slack time as possible.

The definition of the frequency variant factor n is closely related to the proportion of the time zone length and the real execution time of the previous job. The bigger the difference between RET and the estimated time zone length, the bigger will n be. For example, if the previous job dropped into the first time zone, there is a big gap between RET and WCET. From the algorithm definition of n, the next job within the same task will have much longer length for the lower frequency zone. By prolonging the lower frequency zone, more energy saving is expected for the next job.

The time zone length definition should not disobey the constraint under each condition: C_l/C_m should not be larger than $S_k/(n-1)$. The larger the value of n is, the smaller the time zone length, and the larger the slack time should be compared to C_l or

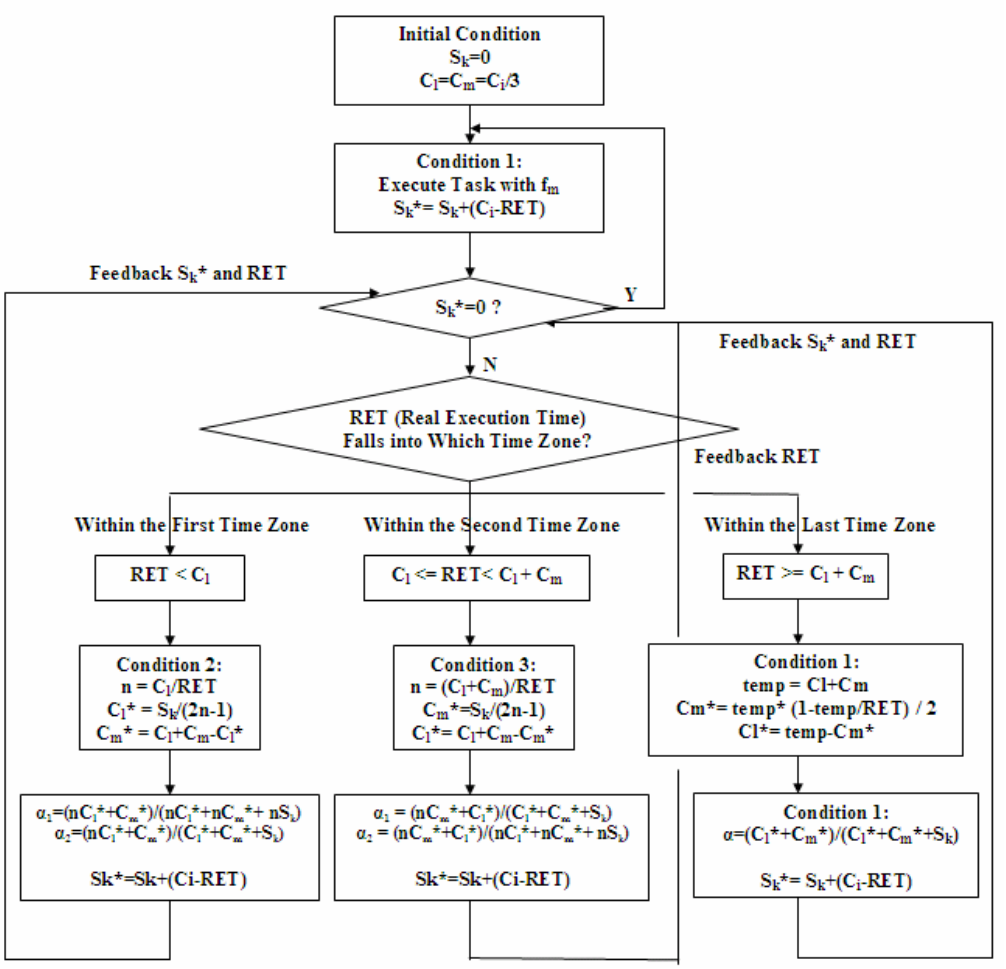

Figure 3. Algorithm Specification

C_m. In order to satisfy the constraints, as well as to make the required value not so difficult to satisfy, we choose $S_k/(2n-1)$ as the length of each required time zone. In the following experimental part, we have tested many assignments of the time zone length, and this is the best assignment format we have obtained right now.

An inter-task level slack reclaiming policy is used in the algorithm with time complexity O(N), where N is the number of jobs to be scheduled. S_k is updated upon completion of every previous job and after the arriving of the current job. Its value is defined as the previous slack time plus the difference between the WCET and the RET. If the current job uses up all its reserved time as well as the accumulated slack time, S_k will drop to zero. Under this situation, a full speed execution will be used on the next job. If the job uses only part of its reserved time, S_k should be the previous job's slack time plus the unused time. The value of (C_i-RET) can be positive, negative, or zero depending on the RET at run-time.

At every time instant of execution, the run-time scheduling complexity for our Dynamic-Mode DVS Algorithm is O(n), where n is the number of tasks in the task set. A task can arrive and finish upon every time instant. We have to keep a record of the rest of the task's earliest arrival time and deadline; the worst case occurs when all the tasks arrive at the same time.

4. Experimental Results

During the experimental part, we attempt to test our algorithm under different criteria. In order to make energy consumption calculation clear and easy, we use the following simplified formula $E = fV^2t$. Our ongoing work studies other energy models. The normalized processor energy consumption is used as a characterized variable to evaluate the performance of the Dynamic-Mode DVS algorithm under the following predefined RET patterns. In all the following experiments, we assume that a fixed amount of energy consumption is used for a given frequency. The processor model for scaling is also simplified as shown in Table 1:

Frequency	Voltage
25%	2 V
50%	3 V
75%	4 V
100%	5 V

Table 1. Processor Model for Scaling

4.1. Robustness/Stability under Different Workloads

In this section, the robustness/stability of the Dynamic-Mode DVS Algorithm is tested under either light or heavy workload. In order to do this, we performed simulation under different ranges both on number of task sets and the number of tasks in the task sets.

Figure 4 shows the normalized energy consumption under different numbers of task sets, which varies from 3 to 30. Each task set contains the following 5 tasks: T1={2,24}, T2={3,36}, T3={2,5}, T4={2,14}, T5={1,4}. The dynamic variant workload is characterized by three RET patterns. In Sin Pattern RET=C_{random}*|Sin(t+JI/2)|, each time value of C_{random} is generated by a random function, and has a value within the range of [50%WCET,100%WCET]. As for the same method, Cos Pattern is characterized as RET=C_{random}*|Cos(t+JI/2)|. Another Constant Pattern is also used, which gives a constant RET value for all tasks. From Figure 4, we can see that our algorithm gives almost all the same amount of energy saving under a large number of task set variants [5,100]. Notice that there is a slight reduction on the energy consumption under each large number of task set, especially from 35.

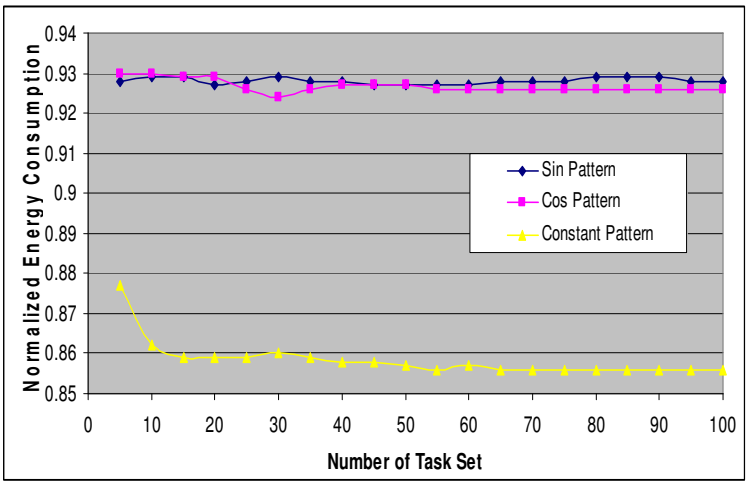

Figure 4. Energy Consumption under Different Number of Task Sets (Worst Case Utilization=0.96)

Next, we adjust the number of tasks in the task set, and let it changes from 3 to 10 using only the Sin Pattern. The amount of energy consumption is decreased with the number of tasks in a task set. Here, we only compare the variant of the Worst Case Utilization within [0.5,1.0]. The reason is, during the experiment, we noticed that our Dynamic-Mode DVS Algorithm has a dramatic energy saving especially under heavy system workload.

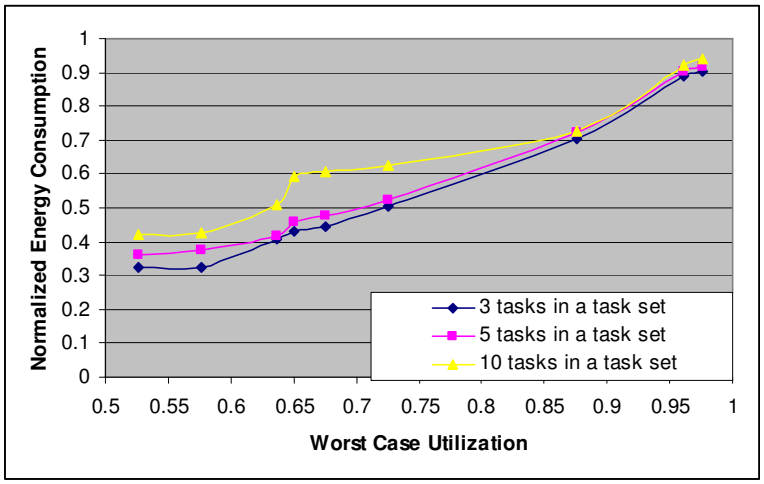

Figure 5. Energy Consumption for Different Number of Tasks (Worst Case Utilization= [0.5 , 1.0])

All task deadlines are met in all these experiments. From the above experimental result, we conclude that our algorithm is robust and stable under both light and heavy workloads.

4.2. Comparison with Traditional DVS Algorithm

Figure 6 compares the performance of four algorithms. The analysis of the experimental results shows that although the deadline constraint of every task is met, a fixed mode DVS cannot obtain the optimal energy saving

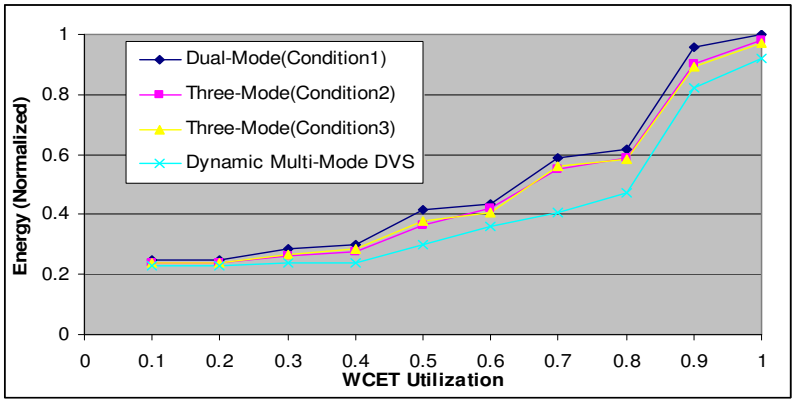

Figure 6. Comparison of Different Mode DVS Algorithms

whereas our Dynamic-Mode DVS provides a novel idea to whereas our Dynamic-Mode DVS provides a novel idea to further energy saving. Comparing the results, the three-mode DVS saves 5.35% energy consumption compared to dual-mode. Dynamic-Mode DVS achieves 17% more energy saving over Dual-Mode as well as 12.5% more energy saving over Three-Mode DVS. Our Dynamic-Mode DVS achieves an average of 15% more energy saving over the dual-mode DVS when speed-switching or energy transition overhead is ignored. We next account for this overhead and study the performance of the proposed Dynamic-Mode DVS algorithm.

4.3. Accounting for Speed-Switching Overhead

During the experiment, we noticed that most published on-line DVS algorithms make the assumption that the overhead during the whole scheduling procedure is negligible. The recent on-line DVS algorithm presented in [16] is the first to explicitly account for speed-change or transition overhead in a preemptive fixed-priority scheduler. The practical PACE algorithm [22] is another recently introduced algorithm to account for speed-change overhead and processor idle power. Transition time overhead can complicate the DVS algorithm by reducing the available slack time. It is very complex for an on-line voltage scheduling algorithm to guarantee task deadlines when this overhead must be considered. Basically, there are several kinds of overheads that are being considered: time and energy

transition overhead as well as leakage energy dissipation, overhead when scheduling voltage levels, and checkpoint times for fault-tolerant systems. Extra energy consumption must be considered, because both the voltage transition energy itself and the transition time within which no jobs can be executed count. Since our algorithm uses a three-mode virtual task splitting method, there are potentially more voltage scaling changes than previous dual-mode DVS.

In our preliminary experimental result, we account for this speed-change or transition overhead. With the rapid development of microprocessor nowadays, less and less overhead occurs in switching speed/voltage. The Strong ARM SA-1100 processor can change voltage in less than 140 microseconds [17] and newer microprocessors have even shorter speed-switching time. The "ultra-low power" IBM PowerPC 405LP processor offers both asynchronous as well as conventional (synchronous) frequency switching. Asynchronous switching even allows a task to run during frequency/voltage transitions and is expected to cut the effective switching time to below 20 microseconds. The following is our preliminary test model accounting for the speed-change overhead.

Time/Energy Overhead	**Cost**
interrupt (preemption) response	sub-200-nanosecond
Context Switch Overhead	sub-microsecond
Transition Time Overhead [16]	100 μs
Transition Energy Overhead [18]	$\Delta E=\eta * C_{DD} * \lvert V1^2-V2^2\rvert$ ($\eta=0.9$ and $C_{DD} = 5\mu F$)

Table 2. Energy Transition Overhead Model

We test our Dynamic-Mode DVS Algorithm under the above energy transition overhead model. The normalized energy saving under different worst case utilizations is shown in Figure 7. During the experiment, a pessimistic judgment on energy transition overhead is added before every application of voltage scaling in order to meet the tasks' strict time constraint as well as avoiding more energy consumption caused by the voltage scaling. If the energy transition overhead is bigger than the expected energy saving, no voltage scaling is used. Otherwise, the proposed voltage scaling is executed, and then a certain amount of transition overhead is added to the original energy consumption. In this testing, our proposed algorithm is used less aggressively in order to avoid transitions that would increase energy instead of reducing it. The performance of our Dynamic-Mode DVS algorithm under energy transition overhead is shown in Figure 7. Performing pessimistic energy saving judgment guarantees all task deadlines, but the amount of energy saving is dramatically reduced.

However, even accounting for the energy transition overhead, our dynamic-mode DVS algorithm can still achieve an average of 14% energy saving over a strategy which has no voltage scaling strategy. Since our original purpose is not directed toward energy saving with transition overhead, it has less benefit right now. However, we expect to derive potential improvements to make our algorithm more suitable when counting energy transition overhead. Also, our algorithm's energy-saving efficiency increases as newer microprocessors' speed-change overhead decreases.

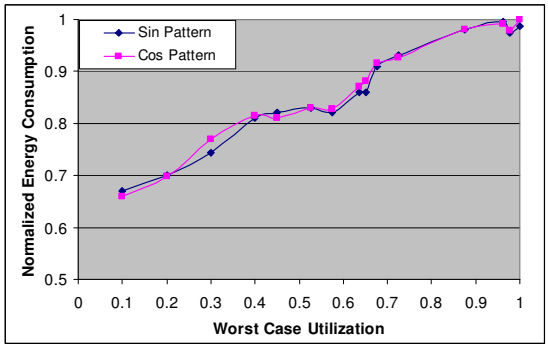

Figure 7. Normalized Energy Consumption with the Consideration of Energy Transition Overhead

5. Conclusion and Future Work

This paper presents a DVS scheduling approach for hard, periodic real-time systems. A novel idea called Three-Mode Task Splitting is introduced and verified, which is essential especially under a dynamic workload. The experimental results show that our algorithm can achieve an average 15% energy consumption saving over the state-of-the-art dual-mode algorithm in a robust/stable way for a variety of task workloads. Furthermore, an average 14% energy saving over a non-DVS scheme can be expected while accounting for the voltage transition overhead, and this energy saving increases as the advancing microprocessor technology decreases the voltage transition overhead.

Our on-going work includes evaluating the algorithm under a real embedded environment (Intel PXA255 development platform), finding more aggressive slack reclaiming strategies which are more suitable to account for speed-switching/voltage transition overheads and processor idle power. In addition, we will compare the performance of our approach with the best dual-mode schemes as well as the recently proposed PPACE approach [22] by accounting for the processor idle power and the speed change overhead.

References

[1] Yifan Zhu, Frank Mueller, "Feedback EDF Scheduling Exploiting Dynamic Voltage Scaling", IEEE RTAS, 2004.

[2] M. Weiser, B. Welch, A. Demers, S. Shenker, "Scheduling for reduced CPU energy," Processings of 1st USENIX Symposium on Operating Systems Design and Implementation (OSDI'94), pp.13-23.

[3] Yann-Hang Lee, Yoonmee Doh, C. M. Krishna, "EDF Scheduling Using Two-Mode Voltage-Clock-Scaling for Hard Real-Time Systems", ACM CASES'01, Atlanta, Georgia, USA, November 2001.

[4] J. Lorch, "Operating Systems Techniques for Reducing Processor Energy Consumption," PhD thesis, University of California, Berkeley, 2001.

[5] D. Shin, J. Kim and S. Lee, "Intra-task voltage scheduling for low-energy hard real-time applications," IEEE Design and Test of Computers, 18:(2), March-April 2001.

[6] Rami Melhem, Nevine AbouGhazaleh, Hakan Aydin, Daniel Mosse, Chapter 7 Power management points in power-aware real-time systems, University of Pittsburgh.

[7] T. Ishihara and H. Yasuura, "Voltage scheduling problem for dynamically variable voltage processors," Proceedings of the 1998 international symposium on Low power electronics and design, pages 197-202. ACM Press, 1998.

[8] T. D. Burd, R. W. Brodersen, "Energy efficient CMOS microprocessor design," Proceedings of the 28th Annual Hawaii International Conference on System Sciences. Volume1: Architecture, T. N. Mudge and B.D Shriver, Eds., IEEE Computer Society Press, pp.288-297, Jan. 1995.

[9] C. Liu and J. Layland, "Scheduling algorithms for multiprogramming in a hard real-time environment," J. of the Association for Computing Machinery, 20(1): 46-61, Jan. 1973.

[10] T. Ishihara and H. Yasuura, "Voltage Scheduling Problem for Dynamically Variable Voltage Processors," ACM ISLPED 98, Aug. 1998.

[11] T. Pering, T. Burd, R. Brodersen, "The Simulation and Evaluation of Dynamic Voltage Scheduling Algorithms," Proceedings of International Symposium on Low Power Electronics and Design (ISLED'98), pp.76-81.

[12] T. Pering and R. Brodersen, "Energy Efficient Voltage Scheduling for Real-Time Operating Systems," The 4th IEEE Real-Time Technology and Applications Symposium, Work in Progress Session, 1998.

[13] Padmanabhan Pillai and Kang G. Shin, "Real Time Dynamic Voltage Scaling for Low power Embedded Operating Systems" in Proc. of 18th Symposium on Operating Systems Principles, Banff, Canada, October, 2001.

[14] N. AbouGhazaleh, D. Mosse, B. Childers and R. Melhem, "Toward the Placement of Power Management Points in Real Time Applications," Proc. Workshop on Compilers and Operating Systems for Low Power, September 2001.

[15] D. Shin, J. Kim and S. Lee, "Intra-task voltage scheduling for low-energy hard real-time applications," IEEE Design and Test of Computers, 18:(2), March-April 2001.

[16] Bren Mochocki, Xiaobo Sharon Hu and Gang Quan, "Practical On-line DVS Scheduling for Fixed-Priority Real-Time Systems," Real-Time and Embedded Technology and Applications Symposium, San Francisco, CA, March 2005.

[17] J. Pouwelse, K. Langendoen, and H. Sips, "Dynamic Voltage Scaling on a Low-Power Microprocessor," Proc. 7th International Conference on Mobile Computing and Networking, Rome, Italy, July 2001.

[18] T.D. Burd, “Energy-Efficient Processor System Design,” PhD thesis, University of California, Berkeley, CA, May 2001.
[19] C.-C. Chu and A. M. K. Cheng, “Static and Dynamic Methods to Improve Total Reward of Tasks in Battery-Powered Devices,” Proc. WIP Session, IEEE-CS Real-Time and Embedded Technology and Applications Symposium, 2004.
[20] W. Yuan and K. Nahrstedt, “Energy-Efficient Soft Real-Time CPU Scheduling for Mobile Multimedia Systems,” Proc. ACM SOSP '03, October 2003.
[21] J. Lorch, and A. J. Smith, “Improving Dynamic Voltage Scaling Algorithms with PACE,” Proceedings of the ACM SIGMETRICS 2001 Conference, pp. 50–61, Cambridge, MA, June 2001.
[22] Ruibin Xu, Chenhai Xi, Rami Melhem and Daniel Mossé, “Practical PACE for Embedded Systems,” Proc. 4th ACM International Conference on Embedded Software (EMSOFT '04), Pisa, Italy, September, 2004.