2-2008

Dynamic Energy Aware Task Scheduling for Periodic Tasks using Expected Execution Time Feedback

Sachin Pawaskar
University of Nebraska at Omaha, spawaskar@unomaha.edu

Hesham Ali
University of Nebraska at Omaha, hali@unomaha.edu

Dynamic Energy Aware Task Scheduling for Periodic Tasks using Expected Execution Time Feedback

Sachin Pawaskar and Hesham H. Ali
Department of Computer Science
University of Nebraska at Omaha
Omaha, NE 68182, USA
sachinpawaskar@msn.com | hesham@unomaha.edu

ABSTRACT

Scheduling dependent tasks is one of the most challenging problems in parallel and distributed systems. It is known to be computationally intractable in its general form as well as several restricted cases. An interesting application of scheduling is in the area of energy awareness for mobile battery operated devices where minimizing the energy utilized is the most important scheduling policy consideration. A number of heuristics have been developed for this consideration. In this paper, we study the scheduling problem for a particular battery model. In the proposed work, we show how to enhance a well know approach of accounting for the slack generated at runtime due to the difference between WCET (Worst Case Execution Time) and AET (Actual Execution Time). Our solution exploits the knowledge gained about the AET of the tasks after the first period, to come up with EET (Expected Execution Time). We then use the EET as an input for the next period to use as much slack as possible and to eliminate wastage of slack generated. This happens because WCET is used to determine if a task should be executed at runtime. Dynamically adjusting the run-queue to use EET as a feedback, which is based on the previous period's AET eliminates wastage of the slack generated. Based on the outcome of the conducted experiments, the proposed algorithm outperformed or matched the performance of the 2-Phase dynamic task scheduling algorithm and the run-queue peek algorithm all the time.

KEY WORDS

Scheduling, Energy Awareness, Heuristics, Parallel Processing, Optimal algorithms.

1. Introduction

Mobile computing has become a reality. Through the Wireless Verification Program, Intel® and leading wireless LAN service providers have verified more than 40,000 hotspots around the world, with more cropping up each day [1]. Mobile technology is continually advancing to keep up with the needs of the mobile user. But as we work to make the ideal mobile experience, we find ourselves up against an inherent struggle between extending battery life and improving mobile performance. Power consumption has been a critical design constraint in the design of digital systems due to widely used portable systems such as cellular phones and PDAs, which require low power consumption with high speed and complex functionality. The design of such systems often involves reprogrammable processors such as microprocessors, microcontrollers, and DSPs in the form of off-the-shelf components or cores. Furthermore, an increasing amount of system functionality tends to be realized through software, which is leveraged by the high performance of modern processors. As a consequence, reduction of the power consumption of processors is important for the power-efficient design of such systems.

Battery operated portable devices are widely used in mobile computing and wireless communication applications. Maximizing battery lifetime is the most important design consideration for such systems. Since the amount of energy delivered by the battery depends on the discharge current profile, the battery life can be extended by controlling the discharge current level and shape [2, 3].

Broadly, there are two kinds of methods to reduce power consumption of processors. The first is to bring a processor into a power-down mode, where only certain parts of the processor such as the clock generation and timer circuits are kept running when the processor is in an idle state. Most power-down modes have a tradeoff between the amount of power saving and the latency incurred during mode change. Therefore, for an application where latency cannot be tolerated, such as for a real-time system, the applicability of power-down may be restricted. Another method is to dynamically change the processor speed by varying the clock frequency along with the supply voltage when the required performance on the processor is lower than the maximum performance. A significant power reduction can be obtained by this method because the dynamic power of a CMOS circuit is quadratically dependent on the supply voltage [3].

In recent years there has been a significant amount of work done on studying battery characteristics and using these characteristics to shape the discharge profile. Most of the earlier work for battery-aware task scheduling has been for static tasks where complete information about the tasks is known apriori [2]. In this paper we propose an enhanced algorithm for the dynamic energy aware task scheduling problem.

2. Energy Aware Scheduling

Scheduling is a classical field with several interesting problems and results. Due to its wide range of applications, the scheduling problem has been attracting many researchers from a number of fields. A scheduling problem emerges whenever there is a choice. The choice could be the order in which a number of tasks can be performed, and/or in the assignment of tasks for processing.

The problem is to determine some sequences of these operations that are preferred according to certain (e.g. economic) criteria. The problem of discovering these preferred sequences is referred to as the sequencing problem. Over the years, several methods have been used to deal with the sequencing problem such as complete enumeration, heuristic rules, integer programming, and sampling methods. It is clear that complete enumeration is impractical because the problem is exponential, which means that it requires too much time, sometimes years of computation time would be required even for a small number of tasks. Hence optimal solutions cannot be obtained in real time [4, 5]. However, many heuristic methods have been used to deal with most general case of the problem. Such methods include traditional priority-based algorithms [6], task merging techniques [7], critical path heuristics [6, 8]. In addition, distributed algorithms have been designed to address different versions of the scheduling problem [9].

In general, the scheduling problem assumes a set of resources and a set of consumers serviced by these resources according to a certain policy. Based on the nature of and the constraints on the consumers and the resources, the problem is to find an efficient policy (schedule) for managing the access to and the use of the resources by various consumers to optimize some desired performance measure such as the total service time (schedule length).

Energy Aware Scheduling is a special case of the general scheduling problem in which our scheduling policy is the optimization of the energy or power of the battery. Minimizing the battery power utilization becomes the most important consideration in a system that is energy aware, at the same time one must realize that along with this there are certain parameters that must be met such as tasks meeting their deadlines.

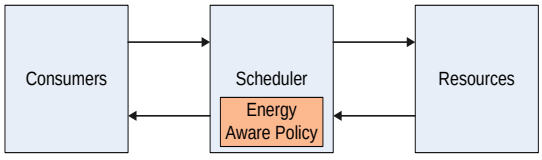

Figure 1: Energy Aware Scheduling System

Simply put an Energy Aware Scheduling System is a scheduling problem which assumes a set of resources and a set of consumers serviced by these resources according to a Energy Aware policy. Based on the nature of and the constraints on the consumers and the resources, the problem is to find an efficient policy (schedule) for managing the access to and the use of the resources by various consumers to optimize the desired performance measure which in this case is minimum amount of battery energy. Accordingly, an Energy Aware scheduling system can be considered as consisting of a set of consumers, a set of resources, and an Energy Aware scheduling policy as shown in the Figure 1 above. Clearly, there is a fundamental similarity to scheduling problems regardless of the difference in the nature of the tasks and the environment.

3. Scheduling Model

There are several models for which different algorithms have been proposed. We take look at one such model, discuss the scheduling algorithm proposed for this model, its variations and finally present our improvement for scheduling on this model.

Let us understand the basic characteristics of this Model.

1. The model assumes fixed priority scheduling.
2. The model is for a real time system, in which task deadlines must be met.

The system configuration for the battery-operated processor under consideration is described in Figure 2. The system consists of one DVS processor driven by a single battery. The battery is used to power the processor through a DC-DC converter. The DC-DC converter has an efficiency $\eta = I_{proc}*V_{proc}/I_{batt}*V_{batt}$, where V_{batt} and I_{batt} are the battery voltage and current and V_{proc} and I_{proc} are the processor voltage and current.

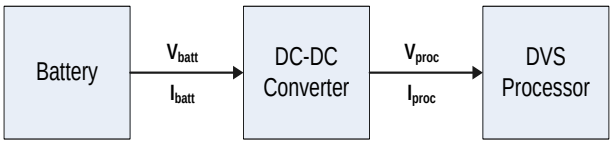

Figure 2: System Level Configuration

Non-linear properties of the battery:

There are several important properties of the battery with respect to voltage scaling that have been derived from the analytical model. We present two of the properties used for developing the real-time scheduling heuristics [10, 2]:

Property 1: *For a fixed voltage assignment (only task start times can be changed), sequencing tasks in the non-increasing order of their currents is optimal when the task loads are constant during the execution of the task.*

Property 2: *Given a pair of two identical tasks in the profile and a delay slack to be utilized by voltage down-scaling, it is always better to use the slack on the later task than on an earlier task.*

Task description: A given task ***k*** is associated with the following parameters: the current I_k , the worst case execution time $WECT_k$, the arrival time a_k, the start time t_k, the actual execution time AET_k, the deadline d_k and the period P_k. The slack associated with a task is due to two factors: (1) the inherent slack due to the difference between the deadline and the WCET and (2) the slack generated due to the actual execution time being less than the worst case execution time (Figure 3).

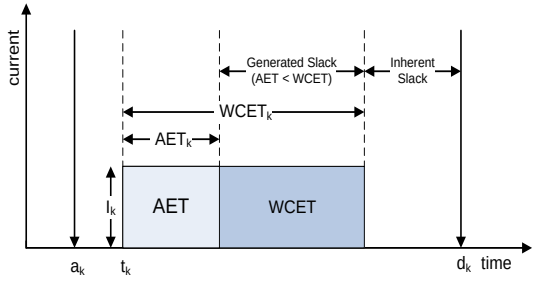

Figure 3: Task Description

Power-Down Modes:
In most embedded systems, a processor often waits for some events from its environment, wasting its power. To reduce the waste, modern processors are often equipped with various levels of power modes. In the case of the PowerPC 603 processor [11], there are four power modes, which can be selected by setting the appropriate control bits in a register. Each mode is associated with a level of power saving and delay overhead. For example, in *sleep mode*, where only the PLL and clock are kept running, power consumption drops to 5% of full power mode with about 10 clock cycles delay to return to full power mode.
In the conventional approach employed in most portable computers, a processor enters power-down mode after it stays in an idle state for a predefined time interval. Since the processor still wastes its energy while in the idle state, this approach fails to obtain a large reduction in energy when the idle interval occurs intermittently and its length is short. In [12, 13], the length of the next idle period is predicted based on a history of processor usage. The predicted value becomes the metric to determine whether it is beneficial to enter power-down modes or not. This method focuses on event driven applications such as user-interfaces because latency, which arises when the predicted value does not match the actual value, can be tolerated. However, we need an exact value instead of a predicted value for the next idle period when we are to apply the power-down modes in a hard real-time system, which is possible in the LPFPS.

4. Overview of Previous Work

C. Chakrabarti and J. Ahmed [2] enhanced the algorithm proposed by Y.Shin and K. Choi [3] by extending the algorithm to account for the slack generated at runtime due to the difference between WECT and AET (Actual Execution Time). They proposed an algorithm which had 2 Phases. The basic idea of the algorithms in this model is to exploit the slacks generated to reduce the voltage levels of the tasks, so that the battery charge consumed or the drop in voltage is minimized. The algorithm operates in two phases.

1. **Phase I:** Off-line task scheduling algorithm using WCET.
2. **Phase II:** On-line algorithm using AET.

In Phase I the tasks are assumed to be executed at their WCETs. A schedule is determined for one hyper-period (defined as the least common multiple of the periods of all the tasks in the task set).
In Phase II (on-line), the slack generated due to the AET being less than the WCET, is used to further scale the voltage levels of the tasks.

Phase I: The off-line scheduling algorithm is based on a paper presented by the same co-authors [10], it determines the task ordering and the voltage level of each instance of a task in a hyper-period. Applying WCETs in this phase guarantees that the tasks meet their deadline. This is done in two steps.
Step 1: Obtain a feasible schedule by using the earliest deadline first algorithm.
Step 2: Utilize the available slack by voltage down scaling as much as possible starting from the end of the profile.

Phase II: During operation of the system, the AET of a task could be a lot smaller than its WCET. It is suggested that it is best to use the slack as late as possible, which is achieved by a process called as slack forwarding. Slack forwarding is based on the observation that slack generated by early completion of a task can be made available to a later task if the later task is released prior to the time at which the slack originated.

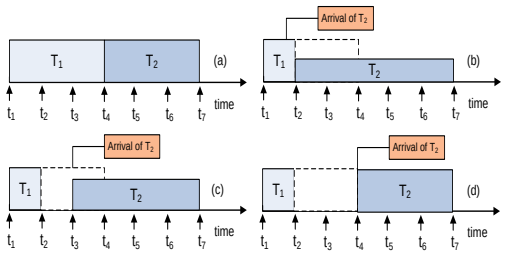

Figure 4: (a) WCET Schedule. (b) WCET Schedule with full slack forwarding. (c) WCET Schedule with partial slack forwarding. (d) WCET Schedule with no partial slack forwarding.

Consider two tasks T_1 and T_2 and let us assume WCET for the tasks. Task T_1 starts at t_1 and finishes at t_4 and T_2 starts at t_4 and finishes at t_7, as shown in Figure 4(a). Suppose T_1 actually finishes earlier at time t_2, generating

a slack of (t_4-t_2). All of this slack is available to T_2 if its arrival time is at t_2 or before, as depicted in Figure 4(b). If the task T_2 was released at t_3, only a part of the generated slack is available to T_2, as shown in Figure 4(c). If the task T_2 was released at t_4 none of the generated slack is available to T_2 as shown in the Figure 4(d). Thus the decision of slack forwarding can be made by inspecting the arrival time of the subsequent task to be executed.

The purpose of this algorithm is to readjust the voltage level of the task based on additional slack. The basic steps are as follows. After the completion of a task, the scheduler gets the next task from the run queue. The finish time of the task is estimated based on the voltage level determined in Phase I. If the finish time is before the release time of the next task in the queue, the voltage level of the task is readjusted.

Example:

Consider the three tasks given in Table 1 which is reproduced below. Rate monotonic priority assignment is a natural choice because periods (*Pi*) are equal to deadlines (*Di*). Priorities are assigned in row order as shown in the fifth column of the Table 1. Note that this is the same example from the original algorithm 1 by Y.Shin and K. Choi [3], which is being adapted to show the incremental improvement done by Chakrabarti and J. Ahmed [2].

Table 1: Example Task Set

	P_i	D_i	C_i	Priority
T_1	50	50	10	1
T_2	80	80	20	2
T_3	100	100	40	3

Let us consider the task set in [3] represented by the Table above. There are three tasks with periods 50, 80 and 100 minutes. The hyper-period is 400 minutes (L.C.M of 50, 80 and 100).The set of operating voltages considered during voltage scaling is S_v = {3.3, 3.0, 2.7, 2.5, 2.0} volts. Figure 5(c) shows the final task profile with the improved algorithm after each phase as well as that generated with the low power fixed priority algorithm in [3].

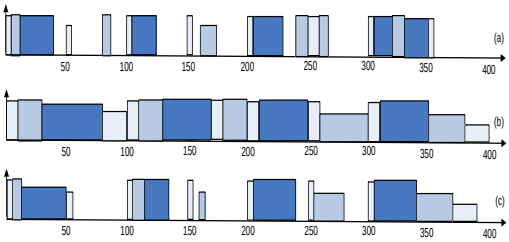

Figure 5: Task scheduling using LPFPS algorithm in [3] versus enhancements in [2]

S. Pawaskar and H. Ali [14] enhanced the algorithm proposed by C. Chakrabarti and J. Ahmed [2] by exploiting the fact that even though some tasks become available based on the actual periodicity of a task they are not executed because the run queue is determined by the schedule generated in the offline phase I of the algorithm using the conservative EDF (Earliest Deadline First) algorithm. S. Pawaskar and H. Ali [14] peek at the task run-queue to find such tasks and schedule them for execution if possible based on the knowledge of the available slack and the arrival on the next task. This helps in minimizing the wastage of the generated slack.

Considering the same set of tasks as described in [2, 3] and shown here in Table 1., this waste of slack can be observed at time t=80 even though T_2 becomes available as per the periodicity of the task it is not executed because the run queue determined by the Offline phase has T_1 as the next task. We also notice that T_2 can be easily completed before T_1 whose next earliest start time is t=100, because T_2 has WCET execution time of 20 and since it starts at time t=80 we have a timeframe of (100 – 80) = 20 available for execution.

A similar yet slightly different situation occurs at time t=240, where even though T_2 becomes available as per the periodicity of the task it is not executed in [2] because the run queue determined by the Offline phase has T_1 as the next task at t=250. We also notice that T_2 cannot be easily completed before T_1 whose next earliest start time is t=250, because T_2 has WCET execution time of 20 and since it starts at time t=240 we have a timeframe of (250 – 240) = 10 available for execution. But a simple task look –ahead shows that to execute both T_1 and T_2 we have a total time of (240-300) = 60 and the WCET for each is 10 and 20 respectively, a total duration WCET of 30, which tells us that scheduling T_2 now will not cause us to miss the deadline for T_1 and that both tasks can be executed within the available time of 60.

To avoid this waste, the algorithm is enhanced such that the original start time for each periodic task is fed to the algorithm as input. Figure 6 shows the final task profile with the run-queue peek algorithm [14] as well as those generated by [2] and with the low power fixed priority algorithm [3]. Since the algorithm further scale down of the voltage and make more use of online slack the run-queue peek algorithm [14] performs better compared to [2] and [3].

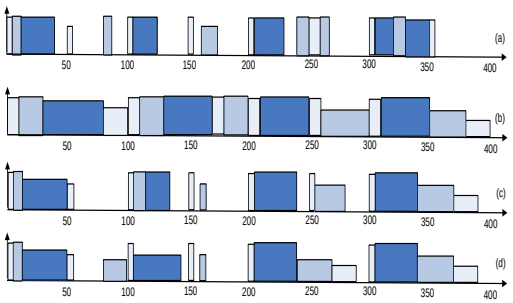

Figure 6: Task scheduling using LPFPS [3], 2-Phase algorithm [2] and Run-Queue peek [14]

5. Proposed Solution: Expected Execution Time Feedback

We realized that some online slack could be potentially wasted in the algorithm proposed by C. Chakrabarti and J. Ahmed [2] and S. Pawaskar and H. Ali [14] due to the fact that WCET is used to determine the scheduling for every periodicity even though after the first and subsequent execution of the tasks we are aware of the AET and can rationally compute expected execution time which allows us to better utilize the slack generated and hence the improve on the overall utilization of energy. The calculation of expected execution time can be done in one of two ways, either conservatively or in a risky manner.

Motivation: Our solution exploits the fact that even though we have knowledge of the AET of the tasks after the first period, it is not used in the determination of the task scheduling for the subsequent periods. Dynamically adjusting the run-queue based on the previous periods AET is obviously going to be much more efficient than using a static run queue that is determined by the schedule generated in the offline phase I of the algorithm using the conservative EDF (Earliest Deadline First) algorithm. We dynamically adjust the task run-queue by calculating EET based on the knowledge of WCET and the AET of the previous period. Tasks are then scheduled for execution if possible based on the knowledge of the available slack and the arrival on the next task. This helps in minimizing the wastage of the generated slack.

Most of the Energy Aware Scheduling Algorithms designed so far use WCET to compute the workloads in the offline phase. In general most tasks complete between BCET and WCET. In fact, it is a well known that most tasks complete well before WCET. We propose to exploit this knowledge to our advantage and propose that instead of computing workload at WCET, we use information regarding expected execution time (EET).

Expected Execution Time (EET) may be computed in several ways, one way to compute this would be based on Actual Execution Time (AET) in the previous hyper-period, another approach could be average of all previous AET for that task, so on and so forth. An important aspect of this approach is that at runtime depending on AET we may have some tasks completing in time greater than EET and some less than EET. This could potentially lead to deadline violations, which we need to resolve.

Approaches to compute Expected Execution Time

1. **Conservative Approach:** Expected Execution Time is computed conservatively so that it is closer to WCET. This approach has a lower propensity for deadline violations, which need to be resolved. Accordingly, we form the following equations [Eq1].

$$Slack = (WCET - AET)$$
$$EET = WCET - \alpha(WCET - AET)$$
$$EET = WCET - \alpha(Slack) \ where\ 0 < \alpha < 1$$

2. **Risky Approach:** Expected Execution Time is computed quite generously so that it is closer to BCET. This approach has a higher propensity for deadline violations, which need to be resolved. Accordingly, we form the following equations [Eq2].

$$rSlack = (AET - BCET)$$
$$EET = WCET - \alpha(WCET - AET) - \beta(AET - BCET)$$

Since we first utilize all the regular slack, α is 1.

$$EET = AET - \beta(AET - BCET)$$
$$EET = AET - \beta(rSlack)\ where\ 0 < \beta < 1$$

The pseudo-code for our EET feedback algorithm is shown in the Figure 7 below. The algorithm is similar in nature to [2, 14] but has key distinctions, we first initialize the EET of each task to the WCET since we can only compute EET after the execution of the tasks in the first hyper-period. We also get the initial scaling level of the tasks from the Phase I schedule for the first period. After the task is executed we then perform some key steps, first we check to see if the AET > EET, this means that we could potentially run into deadline violations and need to adjust slack disbursement accordingly, otherwise AET < EET and we need to update the scaling level to absorb the additional slack. Finally we calculate the EET of the task for the next hyper-period based on the current AET of that task.

```
Input: Phase I schedule and original task periodicity
Initialize EET for all Tasks to WCET (This is to account for the first period)
Initialize the Scaling level of the tasks from the Phase I schedule for first period.
Repeat for Every Task
Get the scaling level of the next task Ti
If the task is not available (Current time < Task Ti start time)
{
        if ( (original task periodicity shows a task To is available earlier) and
            (start time of Ti – To >= EET of To ) or (Ti+2 - To >= EET Ti + EET To) )
                Schedule task To and remove it from Phase I schedule
        else
                Wait
}
Else
{
        If (finish time of task Ti < release time of task Ti+1)
                Update the scaling level to absorb the slack
}
Execute the task
If (AET > EET) // we need to check for possible deadline violations.
{
        CheckandAdjustforDeadlineViolation();
}
Update Scaling Level to absorb the additional slack if any
Compute EET ( Ti , AET, Alpha, Beta )
```

Figure 7: Pseudo-code of proposed EET feedback algorithm

6. Implementation and Results

To show the effectiveness of our algorithm we ran experiments on the proposed algorithm and the algorithm in [2] and [14]. Multiple task sets were used, which we shall call each task set as a test case. The tasks were randomly generated in a set of 3-tuple. Each task has a periodicity between 1 and 10 units. The deadlines of the tasks were made equal to that of their periods. The WCET for a task was randomly chosen between 0 and the period of the task. All the test cases where the task set was not schedulable were dropped. We assume that the AET of the tasks is drawn from a random Gaussian distribution with mean, denoted by μ, and standard deviation denoted by σ, given by the following equation [Eq3] and where BCET is assumed to be 0.1 time the WCET.

$$\mu = \frac{wcet+bcet}{2} \text{ And } \sigma = \frac{wcet-bcet}{6} \rightarrow \text{Eq3}$$

We assumed a continuous operating voltage for the system. The set of operating voltages considered during voltage scaling is S_v = {3.3, 3.0, 2.7, 2.5, 2.0} volts. We then ran the experiment taking a conservative approach (using Eq1), Figure 8 below show performance improvements of our proposed approach as α was varied from 0 to 1. It is clear from the plot that as we moved the EET closer to AET we consistently gained in a better battery performance for most cases, in no case does it perform worse. Note that we use a similar technique as in [2, 3] to generate our tasks, to have a high degree of confidence in our conclusions.

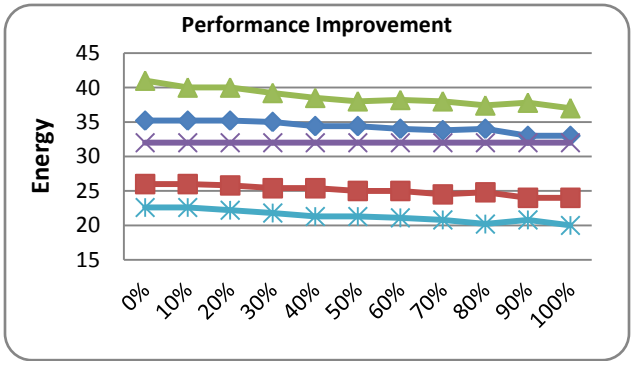

Figure 8: Performance of conservative approach

We also ran our experiment against the algorithm proposed in [2] and [14] and the Figure 9 below show that the enhanced EET feedback algorithm consistently performed better. It is clear from the plot that as we moved the EET closer to AET we consistently gained in a better battery performance for most cases, in no case does it perform worse. Note that we use a similar technique as in [2, 3, 14] to generate our tasks, to have a high degree of confidence in our conclusions.

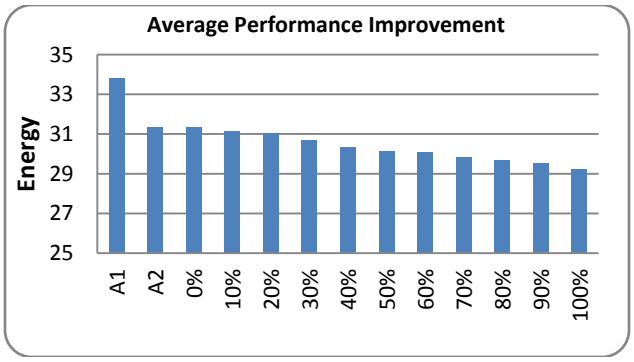

Figure 9:Avg. Peformance of conservative approach

We then decided to run the experiments using the risky approach (using Eq2), Figure 10 below show performance improvement of our proposed approach as β was varied from 0 to 1. Note that for all the cases the performance was slightly better just beyond AET for some low values of β {0.1, 0.2, and 0.3} and then progressive got worse as depicted by the upward curve depending on the Task set. However beyond a certain value of β typically 0.4 or higher we ran into deadline violations that could not be resolved. This suggests that we can squeeze some amount of performance beyond the AET (using EET < AET).

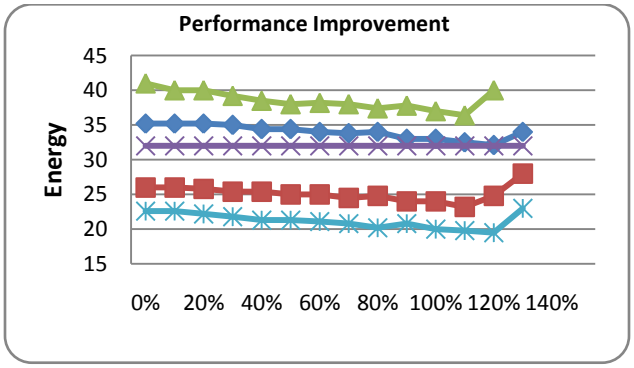

Figure 10: Performance with risky approach

This was also reflected in the plot for average performance improvement over all test cases as shown in Figure 11 below.

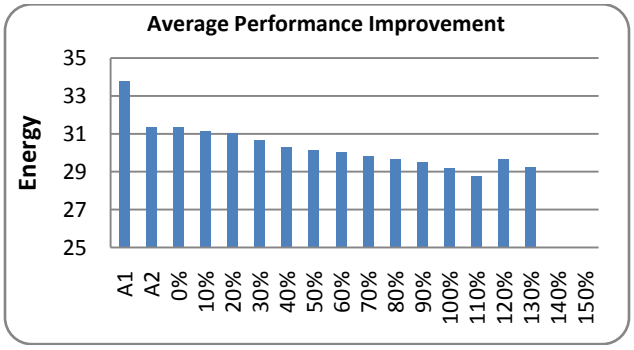

Figure 11: Avg. Performance with risky approach

We also calculated the average energy utilized for all the test cycles and the plot below (Figure 12) clearly suggests that the enhanced EET feedback algorithm performs better than the algorithms in [2] and [14]. We get an average reduction of approximately 7.4% as compared with the algorithm in [14] and 15.7% as compared with the algorithm in [2]. Note that we use a similar technique as in [2, 3, 14] to generate our tasks, to have a high degree of confidence in our conclusions.

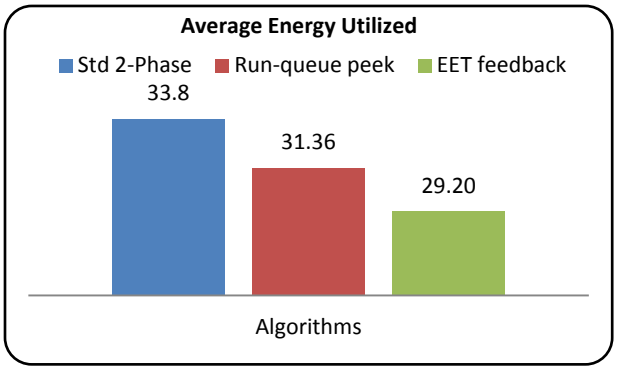

Figure 12: Average Energy Utilized

7. Conclusions

In this paper we proposed an enhanced dynamic task scheduling algorithm using expected execution time feedback for battery operated DVS systems that further maximize the residual charge and the battery voltage. This algorithm has a better battery performance compared to the other algorithms. Our proposed enhancement provides on average an improvement of approximately 15% over the original approach [2]. The performance gains vary from 6% to 20% over the all the test cases. An important consideration in real time systems is time complexity of the additional steps to get these performance gains. Our proposed solution steps have an overall time complexity which is constant [O (1)] and hence adds only negligible processing time.

Our future research will focus on using various techniques of calculating expected execution time (EET). In the proposed solution above we calculate EET after every period. We need to investigate if that helps in reducing energy utilization compared with using the EET after the first hyper-period. We intend to further explore both the suggested approaches of computing EET namely conservative and risky and study their performance relative to each other and understand when it would be reasonable to use one approach over the other. Another investigative thread is can we update the Phase I schedule to use EET instead of WCET to calculate schedule after the first hyper-period. Our future research will explore the application of these approaches and others in a real world application such as a high performance grid computing environment where management of overheating nodes is an important consideration and in wireless sensor networks where devices have energy utilization as a critical operating parameter.

References

[1] Intel. (2005). "The battery life challenge – balancing performance and power", Retrieved Dec 4th 2005 from http://www.intel.com/products/centrino/enablingbatterylIfe.pdf

[2] Jameel Ahmed and Chaitali Chakrabarti, "A Dynamic Task Scheduling Algorithm for Battery Powered DVS Systems" Proc. ISCAS, 813-816, 2004.

[3] Youngsoo Shin and Kiyoung Choi, "Power Conscious Fixed Priority Scheduling for Hard Real-Time Systems, 2005

[4] J. Ullman, NP-complete scheduling problems, Journal of Computer and System Sciences, 10, 384-393, 1975.

[5] E. G. Coffman, R. L. Graham, J. L. Bruno, W. H. Kohler, R. Sethi, K. Steiglitz, and J. D. Ullman: Computer and Job-Shop Scheduling Theory, John Wiley & Sons, A Wiley-Inter-Science publication, 1976.

[6] Hesham El-Rewini, Theodore G. Lewis, Hesham H. Ali: Task Scheduling in Parallel and Distributed Systems, PTR Prentice Hall, Inc. Englewood Cliffs, New Jersey 07632. 1994.

[7] Peter Aronsson and Peter Fritzson: Task Merging and Replication using Graph Rewriting, Tenth International Workshop on Compilers for Parallel Computers, Amsterdam, the Netherlands, Jan 8-10, 2003

[8] A. A. Khan, C. L. McCreary and M. S. Jones, A Comparison of Multiprocessor Scheduling Heuristics, International Conference on Parallel Processing, 1994.

[9] Rong Xie, Daniela Rus and Cliff Stein: Scheduling Multi-Task Agents. In Proceedings of the Fifth IEEE International Conference on Mobile Agents, pages 260-276, Atlanta, Georgia, December, 2001.

[10] P. Chowdhury and C. Chakrabarti, "Battery-aware task scheduling for a system-on-a-chip using voltage/clock scaling," *Proc.SiPS*, 2002.

[11] S. Gary, "PowerPC: A microprocessor for portable computers," *IEEE Design & Test of Computers*, pp. 14–23, Dec. 1994.

[12] M. B. Srivastava, A. P. Chandrakasan, and R. W. Brodersen, "Predictive system shutdown and other architectural techniques for energy efficient programmable computation," *IEEE Trans. on VLSI Systems*, vol. 4, pp. 42–55, Mar. 1996.

[13] C. Hwang and A. Wu, "A predictive system shutdown method for energy saving of event-driven computation," in *Proc. Int'l Conf. on Computer Aided Design*, pp. 28–32, Nov. 1997.

[14] S. Pawaskar and H. Ali, "Dynamic Energy Aware Task scheduling using run-queue peek" in *Proc. Int'l Conf. on Parallel and Distributed Computing and Networks"*, Feb. 2007.