Battery Aware Dynamic Scheduling For Periodic Task Graphs

Venkat Rao[1], Nicolas Navet[1], Gaurav Singhal[2],
Anshul Kumar[3], and G.S Visweswaran[4]

[1] LORIA-INRIA
TRIO TEAM
{rao, nicolas.navet}@loria.fr

[2] University of Texas, Austin
Dept. of Electrical and Computer Engineering
g.singhal@ece.utexas.edu

[3]Indian Institute Of Technology, Delhi
Dept. of Computer Science and Engineering
anshul@cse.iitd.ernet.in

[4]Indian Institute Of Technology, Delhi
Dept. of Electrical Engineering
gswaran@ee.iitd.ernet.in

Abstract

Battery lifetime, a primary design constraint for mobile embedded systems, has been shown to depend heavily on the load current profile. This paper explores how scheduling guidelines from battery models can help in extending battery capacity. It then presents a 'Battery-Aware Scheduling' methodology for periodically arriving taskgraphs with real time deadlines and precedence constraints. Scheduling of even a single taskgraph while minimizing the weighted sum of a cost function has been shown to be NP-Hard [6]. The presented methodology divides the problem in to two steps. First, a good DVS algorithms dynamically determines the minimum frequency of execution. Then, a greedy algorithm allows a near optimal priority function [5] to choose the task which would maximize slack recovery. The methodology also ensures adherence of real time deadlines independent of the choice of the DVS algorithm and priority function used, while following battery guidelines to maximize battery lifetime. Battery simulations carried out on the profile generated by our methodology for a large set of taskgraphs show that battery life time is extended up to 23.3% as compared to existing dynamic scheduling schemes.

1. Introduction

Mobile computing has seen a tremendous growth in the last decade. Due to the continuously increasing functionality and complex applications being integrated with handheld devices, their energy consumption has seen a major upswing. However, battery energy densities have not improved as significantly over the years and this has made battery lifetime a major constraint in mobile embedded systems design.

Over the last decade a lot of effort has been put in exploring Dynamic Voltage Scaling (DVS) as a solution to the trade-off problem. In a DVS scheme, the supply voltage as well as the frequency of the processor is scaled at run time to minimize energy consumption, while satisfying the required speed constraints. Many hardware and software implementation of voltage scaling methods have been proposed. However, most of the early DVS techniques assumed the battery Subsystem to be an ideal source of energy which stores or delivers a fixed amount of energy at a constant output voltage. However, research has shown that this is not always true and designing to minimize average power consumption does not necessarily lead to optimum battery lifetime[13]. In reality, it may not be possible to extract the energy stored in the battery to the full extent as the energy delivered by a battery heavily depends on the load current profile.

Initial works in battery-aware scheduling were primarily voltage scaling algorithms trying to optimize a cost function derived from varied battery models. Both stochastic [9] and partial differential equation models [8] have been used. In [7] the authors discuss static scheduling for DAG's in a battery operated multiprocessor environment using Peukert's law. [14] describes a high level analytical battery model which has been the basis for most of the works that have followed in this field. Approaches trying to optimize a cost function derived from the battery model, however were not suitable for real time operations as they were computationally intensive and cannot be used in a dynami-

1-4244-0054-6/06/$20.00

cally changing environment. In [11], the same authors also demonstrated static scheduling techniques using voltage scaling for one-time aperiodic tasks based on heuristics derived from their earlier works. [1] proposes the extension of the idea to multiprocessor environment.

More recently, there have been attempts to use heuristics derived from battery model for dynamic scheduling of periodic tasks. [17] proposes a dynamic task scheduling algorithm *darEDF* that tries to maintain a locally non-increasing current profile for battery lifetime extension. [15] discusses the scheduling of taskgraphs with precedence constraints in multiprocessor environment and compares the work with earlier approach in [7].

This paper attempts to develop a better understanding of battery awareness by painting an intuitive picture of how the heuristics from battery models actually help in extending battery lifetime.It then addresses a very generic form of battery aware scheduling problem which aims at scheduling periodically arriving taskgraphs in a single processor environment.

The rest of the paper is organized as follows. Section 2 discusses the previous works in Dynamic Voltage scheduling. Section 3 discusses the battery models and the guidelines that have been derived from them. Section 4 discusses the steps involved in scheduling of conditional taskgraphs *ie* frequency setting and deciding on the next task. Section 5 describes the results of the simulations conducted to compare the various dynamic scheduling algorithms, and to highlight the utility of the proposed "Battery-aware Scheduling" methodology. Section 7 concludes the paper.

2. Dynamic Voltage Scheduling

Dynamic Voltage Scaling (DVS) has been a key technique in exploiting the hardware characteristics of processors to reduce energy dissipation by lowering the supply voltage and operating frequency. A typical system configuration for a voltage scalable processor is described in figure 1.

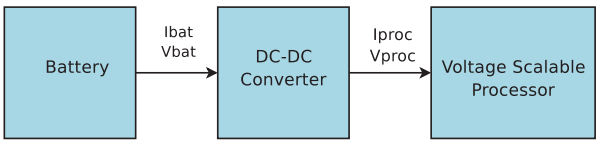

Figure 1. System Configuration for a Single Voltage Scalable Processor

Assuming that the efficiency of the DC-DC converter is η and is constant over the range of voltages, the following equation governs the voltage scaling :

$$\eta \times V_{bat} \times I_{bat} = V_{proc} \times I_{proc}$$

With the scaling of V_{proc} by a factor of s, while V_{bat} is constant, the current I_{bat} is scaled by a factor of s^3 [1]. Thus slack utilization by voltage scaling can greatly affect the load profile and battery lifetime.

Although there has been significant amount of work done in the field of DVS, for an equally diverse range of problem definitions, we use this Section to discuss the algorithms we intend to integrate in our methodology. [10] proposes "cycle conserving" *ccEDF* and "lookahead" *laEDF* DVS algorithms. While *ccEDF* adjusts the voltage and clock speed based on run-time variation in processor utilization alone, *laEDF* aggressively reduces processor frequency and voltage by estimating the minimum amount of work that needs to be completed by the next deadline while ensuring all Subsequent deadlines. Both these algorithm return frequency f_{ref} which is the minimum frequency the system needs to be operated before the current deadline so that no deadlines are violated later, but generally voltage scalable processor can run on a selected set of frequencies. So, running the processor at exact f_{ref} is not always possible. It has been shown that using a linear combination of two adjacent available frequencies($f_i < f_{ref} < f_{i+1}$) is optimal for realizing the running of processor at f_{ref} [4].

Although *laEDF* has been shown to perform better than *ccEDF*, we have used *ccEDF* in our discussion due to its simplicity and ease of implementation. In the simulation Section we have simulated using both *laEDF* and *ccEDF* and compared their performances.

3. Battery Models and Scheduling Guidelines

Perhaps the most accurate method of modeling a battery is to model the electro-chemical processes that take place within the battery. This is the approach described in [3]. The result is a numerical solution to a system of partial differential equations. The main drawbacks of this approach are the long simulation times required and the large number of parameters that need to be specified. Other approaches aimed at reducing the time complexity of low-level simulation are generally based on constructing an abstract representation of the battery like ([9],[13]).

This Section includes scheduling guidelines derived from the analysis of two different battery models: namely the Kinetic Battery Model [8] and the Diffusion Model of [14]. In order to do so, it is necessary

to prove that the battery models point in the same direction. The formal coherence of these battery models has been proved in [12] but has not been included in this paper due to space limitation.

An intuitive proof follows: The kinetic battery model can be understood as a two well model, comprising of an available charge well and a bound charge well. The available charge well supplies to the load directly while the bound charge well cannot do so. However the bound charge well contributes in the replenishment ("Recovery Effect") of the available charge well. The charge transfer between the two wells is governed by the difference in their heights and a rate constant. The battery is said to be discharged when the available charge well becomes empty.

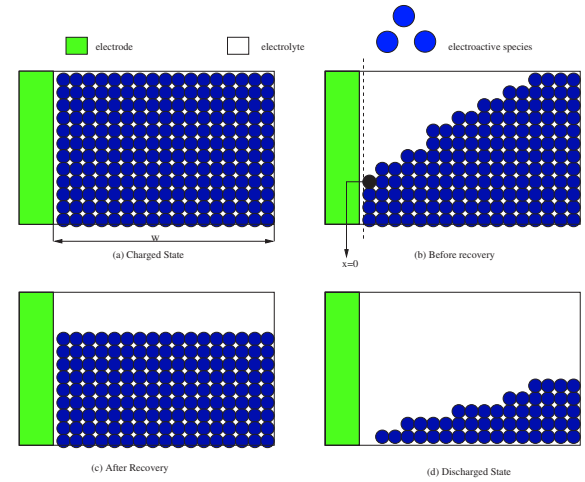

Figure 2. Diffusion Model from [14]

Correspondingly, the diffusion model can be understood as consisting of an infinite number of wells with electrolyte being consumed by reduction at the electrode. Transfer occurs between each of these wells due to diffusion and the battery is said to be discharged when the well adjacent to the electrode is empty. In both the cases it is possible to have charge left in the battery even though its discharged and its voltage has fallen below cut off.

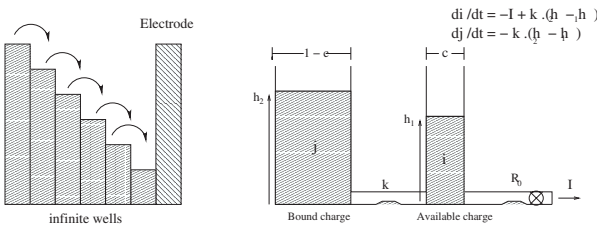

Figure 3. Diffusion Model Vs KiBaM

So in essence, the KiBaM is a coarse grain model where the wells 2-∞ of the infinite well model, have been mapped to a single well *i.e.* the bound charge well. Figure 3 illustrates the idea.Next, we discuss how the guidelines derived from the battery models allow us to achieve our goal. In most of our discussion on how the guidelines help, we would use the KiBaM model as it is easier to visualize.

Scheduling Guideline 1
A non-increasing discharge current profile is optimal for maximizing battery lifetime [14]

This guideline implies that a decreasing order of discharge current profile is optimal to maximize the charge that can be extracted from the battery. This can be better understood by taking a look at the 2-well KiBaM. As mentioned earlier, the rate of the recovery phenomenon from bound charge well to available charge well is directly proportional to the difference in the heights of the wells. A large load current leads to greater difference in the heights and the rate of recovery is higher. But if we begin with small currents and successively increase the currents, small recovery takes place initially. And with sudden large currents in the end the available charge well is emptied without time for recovery while there is still having charge left in the bound charge well. In the opposite order the initial high currents lead to depletion of the available charge well but still charge remains, while there is rapid recovery and transfer of charge from bound well. The smaller currents in the end give ample time to extract charge from the bound well as well. In fact, experiments have shown that a constant infinitesimal load current would be able to extract the maximum charge out of a battery (Rate Capacity Effect) [13].

In terms of voltage scaling, a non-increasing current profile implies that we must schedule in a non-increasing order of voltage/clock speed assignments. However such a schedule is not possible for a real time periodic tasks as we always reduce voltages to utilize slack due to tasks not taking worst case but scale up once a fresh instance of tasks arrive in order to meet deadlines in worst case. So the best that can be achieved is to have a locally non-increasing voltage/clock speed, *ie* the voltage assignments and clock speed are non-increasing within one instance of task arrival.

Scheduling Guideline 2
Given a task t to be executed before a deadline d it is better to lower the frequency and execute the task than to leave an idle slot and execute at a higher frequency. [14]

This guideline advocates for the use of any available slack time for lowering the frequency as far as possible and giving an idle slot only if it cannot be avoided. Any idle slot given would imply that later the processor would be required to run at a higher frequency and would consume more energy. This result has been proved in [14]. This can be understood by realizing that for extending battery life, it is more important to minimize net charge consumed and secondary to optimize charge distribution within the wells.

4. Scheduling of taskgraphs with precedence constraints

This Sections deals with solving a general formulation of the scheduling problem for a single processor environment. The goal is to schedule periodically arriving taskgraphs while optimizing battery lifetime. Each taskgraph is a directed acyclic graph (DAG) in which each node is associated with a task and each edge defines the precedence relation between the two tasks. Task graphs considered here are periodic with deadlines equal to their periods, and all tasks within a taskgraph should be finished by the deadline of the taskgraph. The problem can be defined as:

Problem Definition: *To find a battery efficient schedule for a given a set of periodic tasks graphs* $(T_1, T_2, ...T_n)$ *which have corresponding deadlines* $(D_1, D_2, ...D_n)$*, where a taskgraph* T_i *comprises of any m interdependent nodes, each of which are in themselves tasks with given worst case computations* $(wc_{i1}, wc_{i2}, ...wc_{im})$*.*

In our methodology, we divide the problem into two aspects. One deals with the global frequency selection (which also determines the current profile) and the other deals with choosing the local order of tasks so as to optimize the slack recovery, while maintaining the precedence constraints and deadlines.

The following subsections discuss the individual aspects of the problem and the choice of solution.

4.1. Selecting Frequency

There are many algorithms based on EDF that can be employed to select the frequency that would ensure meeting of all Subsequent deadlines. For the current discussion we have chosen *ccEDF* from [10] as its an efficient algorithm that is simple and ensures locally non-increasing voltage(and hence current) assignments.

The *ccEDF* works by initially calculating the utilization U by summing up the WC_i/D_i's for all released taskgraphs. These WC_i's are themselves a sum of all wc_j's of the nodes of the i^{th} taskgraphs. A frequency f_{ref} is selected such that $f_{ref} = U \times f_{max}$. At the arrival of new taskgraph or at the end of a node in the running taskgraph the WC_i's are updated, U is recalculated and a corresponding frequency is selected again. The pseudocode for the algorithm is described below.

Algorithm 1 Pseudocode for Frequency setting using *ccEDF*

upon_release(Taskgraph T_i *)*
1: $WC_i = \sum wc_{ij}$
2: select_frequency()

upon_endofnode(Taskgraph T_i*,* τ_{ij} *)*
1: $WC_i = WC_i + ac_{ij} - wc_{ij}$
2: select_frequency()

select_frequency ()
1: $U = \sum WC_i/D_i$
2: $f_{ref} = U \times F_{max}$ return f_{ref}

In cases where the j^{th} node of a particular taskgraph T_i takes less that worst case computation we update the WC_i by removing the worst case wc_{ij} component and replacing it by the actual computation taken ac_{ij} so as to utilize the knowledge of actual arrival to allow further lowering of the frequency. This is done as long as the new instance of the taskgraph T_i is not released, whereupon we switch back to the worst case specification. Although the algorithms in [10] were meant to schedule individual tasks with no precedence constraint, we have extended them to handle taskgraphs with precedence constraints.

4.2. Choice of order within EDF

Preemptive Earliest Deadline First(EDF) is an efficient scheduling policy for real time periodic tasks that ensures a feasible schedule even up to 100% processor utilization. However given tasks with identical deadlines, like in the taskgraphs, its possible to have better slack recovery and utilization for a specific order of execution. Fig 4 describes a motivational example of the same. The example describes the execution trace of two tasks with identical deadline=10. Task1 has *wc*=4 whereas task2 has *wc*=6. In the case 1 (A and B) the actual computation is 40% and 60% of *wc* specifications for tasks 1 and 2 respectively. And in case 2 (A and B) they are 60% and 40% of *wc* specifications respectively. The traces set A and B compare the Largest

Task First(LTF) and Shortest Task First(STF) heuristics. In case 1 STF gives better slack recovery and utilization, while in case 2 LTF is better.

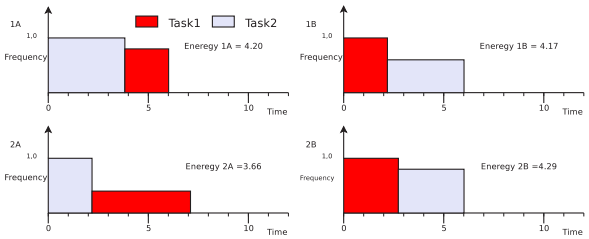

Figure 4. Example of order affecting slack recovery

Many such heuristics exist that try to solve this problem. [5] describes a near optimal UBS heuristic that allows the choosing of the order of tasks to minimize energy cost function for independent tasks with common deadline and given worst case specifications.

However, scheduling becomes much more complex for tasks graphs. Scheduling of even a single taskgraph with precedence constraints while minimizing the weighted sum of a cost function is known to be NP-Hard [6]. In order to simplify, its possible to solve the problem without taking into account interdependencies. Our reduced problem now is to find the schedule for some m tasks having specified worst case computations with a common deadline minimizing the total amount of energy consumed. This problem reduces to the one in [5] which defines a p_{ubs} priority function and the task having the least value of this function is scheduled next.

$$p_{ubs}(o, \tau_k) = \frac{\overline{X_k}}{s_o^2 - s_{o,k}^2}$$

where $\overline{X_k}$ is the estimate of the amount of CPU cycles that task(τ_k) is actually going to require, s_o is the speed after the given partial order and $s_{o,k}$ is the speed after adding task τ_k at the end of the partial order o. The partial order o at any point is order of tasks that have been executed up till now.

$\overline{X_k}$ is used to calculate the priority function, and even if the estimate is wrong no deadlines are violated. However, the accuracy of the estimate, definitely determines the optimality of the schedule. If the estimate is very accurate then the schedule obtained will be near optimal (less than 1% of optimal [5]) , but if the estimate is bad then the schedule will be more like a random schedule. One can use various techniques for accurate estimates of $\overline{X_k}$, one of which is to keep history of previous instances of each task.

After scheduling near optimally for independent tasks with a simple priority function, the idea can be extended to handle tasks with interdependencies. It would be optimum to run the various tasks of a single taskgraph according to the above mentioned p_{ubs}, but since all tasks are not ready to be scheduled, finding an optimum schedule would be a very hard problem. One option is to use this priority function only on the ready list of tasks. So, when selecting the next task to be executed out of tasks in the ready list, we choose the task having the minimum value of p_{ubs}.

This is brings us to the issue of what tasks should comprise of the ready list.

Ready list comprising of nodes of one graph only

A trivial option is to populate the ready list with tasks from a single taskgraph with the most imminent deadline (which have no precedence constraint or whose constraints have been satisfied). In other words, the Ready list at any point in time would comprise of the independent tasks from the taskgraph with the earliest deadline amongst all released taskgraphs. The task that has the smallest value of the UBS priority function amongst the ones in Ready list is executed first. After the finishing of that task the Ready list is updated. This is a simple solution that allows all deadlines to be met since we always follow EDF. Simulations show that this solution yields good results besides being very simple to implement. However this solution limits the choice of tasks to be scheduled and is not optimum in slack recovery. A more greedy approach that allows nodes from other released graphs also in the Ready lists is mentioned below.

Ready list comprising of nodes of all released graphs

This solution allows the independent tasks from all taskgraphs which have been released, to be part of the Ready list. The tasks are scheduled according to the UBS priority function like in the previous approach. However since the Ready list does no longer follows the EDF policy, we need to introduce an additional feasibility check before scheduling any particular task to ensure all deadlines are met. The number of conditions that are needed to be checked before scheduling any task out-of-EDF-order depends on its position in the EDF order. For example no checks are required for scheduling a task that belongs to the taskgraph with the most imminent deadline. And for a task that belongs to taskgraph occupying k^{th} position in the EDF order, k-1 conditions have to be checked. This can be understood by considering the simple fact that executing a task that belongs to taskgraph occupying the k^{th} position in the EDF order can only jeopardize the

meeting of the deadlines of k-1 taskgraphs before it, since after them the k^{th} taskgraph becomes most imminent and hence would be executed even in case of simple EDF ordering. So k-1 checks are required to ensure that the execution of this task would not violate the deadlines for the taskgraphs which come before it. Each check is a condition that ensures that the amount of computations required to be completed in the worst case to meet the next deadline is less than or equal to the time left multiplied by the current f_{ref}.

The next task scheduled is the one with the minimum p_{UBS} in the Ready list which satisfies the feasibility check. The checks are conducted in the increasing order of p_{UBS} value and stopped as soon as a valid candidate is found. Use of f_{ref} in these checks ensure that the we are not forced to run at higher frequencies even if tasks take their worst case (locally non-increasing voltage assignments). Described below is the pseudocode for the feasibility check.

Algorithm 2 Pseudocode for the Feasibility Check

feasibility-check$(\tau_{i,j}, timet, f_{ref})$

1: $flag \leftarrow true$
2: **for** $j = 1$ to edf-position$(\tau_{i,j})$-1 **do**
3: $sumWC \leftarrow 0$
4: **if** $sumWC + WC_j + wc_{ij} > f_{ref} \cdot (D_j - t)$ **then** {tasks in the ready-list are indexed according to the EDF order}
5: $flag \leftarrow false$
6: **return** $flag$
7: **end if**
8: **end for**
9: **return** $flag$

This solution allows more candidates to be reviewed for the priority function, which in turns allows better slack recovery and a more efficient schedule. An example is presented in figure 5. It compares the the execution trace of a set of 3 taskgraphs where taskgraphs T1 has only a single task with wc=5 and D_1=20, taskgraph T2 also has a single task with wc=5 and D_2=50, and taskgraph T3 has 3 tasks each having wc=5 and D_3=100. All tasks are released at t=0. The utilization is 0.5 and hence the f_{ref} is set to 0.5 f_{max}. All tasks take their wcet and hence the f_{ref} does not change during the trace. We assume that tasks from taskgraph3 > taskgraph2 > taskgraph1 according the p_{UBS} priority function. The trace depicts how the methodology allows the execution of released tasks according to priority function without violating deadlines and exceeding f_{ref} by using the feasibility check. Although in the example, we have assumed that all tasks take their wcet, in reality the priority function chooses tasks that are most likely to take much less than wcet and would allow maximum slack recovery and scope for energy minimization.

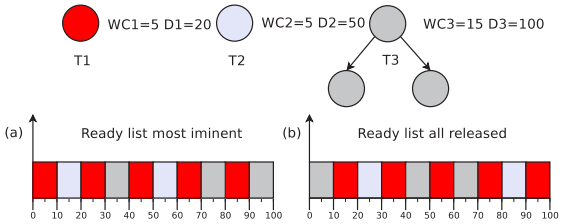

Figure 5. (a)Trace using Canonical EDF ordering, (b)Trace using p_{UBS} based ordering using Feasibility check .

Throughout the discussion we have used *ccEDF* for frequency setting and p_{UBS} priority function for determining the optimal candidate in terms of slack recovery. However, under certain conditions, other DVS schemes or priority function may be more suitable or efficient. The methodology presented here can be used with little or no changes with any frequency setting algorithm and any priority function without deadline violation. The next Section discusses the simulations carried out and results that underline the utility of our methodology.

5. Simulations

C simulations were conducted to compare the various dynamic scheduling algorithms. The DVS enabled processor simulated supports the following three frequency-voltage tuples [(0.5GHz,3 V), (0.75GHz,4V), (1.0GHz,5V)]. Task graphs were generated from TGFF [2] with random dependencies and the worst case computation of each node was chosen randomly following a uniform distribution. Utilization of the system was kept to 70%. Actual computation of a task is assumed to be chosen at random between 20% and 100% of the WCET. Different execution profiles for the taskgraphs were generated by the various scheduling algorithms.

Stochastic battery model from [13] was used to estimate battery life for the profiles generated by various scheduling algorithms. A 1.2V Panasonic AAA NiMH battery with a maximum capacity of 2000 mAh was considered for simulation. This maximum capacity also defines the theoretical bound on performance of the battery and should not be confused with the nominal capacity (which is around 1600mAh for this battery).

The maximum capacity of the battery is defined as the charge delivered by it under infinitesimal load [13].

Similarly the charge in the available well (discussed in Section 3) is defined as the charge that would be delivered if we were to draw infinite current. We can evaluate these values by plotting a load vs delivered capacity curve for the battery and extrapolating the ends (see Figure 5).

The first set of simulations compare the performance of P_{UBS} priority function with the LTF based heuristic presented in [16] in scheduling single DAG's generated from Princeton's TGFF program [2]. The simulation also compare random schedules, that were generated by picking up a task randomly every time from the ready list, and the optimal schedule (in terms of energy consumption) calculated using exhaustive search. The results were averaged over a number of single DAG's with varying number of nodes and have been presented in Table 1.

# of tasks	Random[1]	LTF[1]	p_{UBS} [1]
5	1.32	1.25	1.05
6	1.41	1.29	1.14
7	1.33	1.27	1.17
8	1.56	1.44	1.25
9	1.52	1.26	1.21
10	1.35	1.21	1.09
11	1.66	1.53	1.28
12	1.58	1.39	1.31
13	1.57	1.51	1.22
14	1.44	1.37	1.29
15	1.55	1.51	1.32

Table 1. Energy consumption (normalized w.r.t optimal schedule) by various scheduling policies for different number of tasks in a taskgraph

We have not considered taskgraphs with more than 15 tasks because it takes prohibitively long time to find the optimal schedule by exhaustive search on all feasible schedules.

The second set of simulations are aimed at highlighting the utility of allowing independent tasks from all released taskgraphs to form the Ready list. They compare the resulting energy consumption of the various ordering schemes were tested in scheduling increasing number of taskgraphs with nodes varying from 5 to 15. All schemes employed *laEDF* for frequency setting. The results have been normalized with respect to near optimal schedule obtained by removing precedence constraints within the taskgraphs. It can be seen from Figure 6 that as the number of taskgraphs were increased, the ordering schemes start diverging from the near optimal. However, the scheme selecting the next task using p_{UBS} on all released independent tasks, performs closest to the near optimal among all schemes compared.

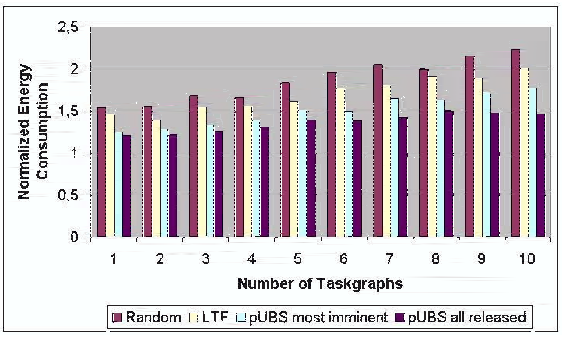

Figure 6. Energy Consumption for Ordering Schemes Normalized wrt Near-Optimal

The final set of simulations compare the battery lifetimes and charged delivered by the combination of various scheduling schemes. The scheduling schemes are defined by the DVS algorithm used for frequency setting, the priority function used for choosing the next task and the Ready list used(ie if it is populated by independent tasks of the most imminent taskgraph only or by all released taskgraphs). The table below describes the schemes and EDF scheduling without any DVS, *laEDF* and *ccEDF* [2] [10] both with random ordering, 'Battery-aware Scheduling'(BAS-1) with the use of a p_{UBS} for ordering among independent tasks from the most imminent taskgraph and 'Battery-aware Scheduling'(Bas-2) with the use of a p_{UBS} for ordering among independent tasks from all released taskgraphs.

Results were obtained by averaging performance of the various algorithms over a large number (100) of random taskgraph sets. It can be observed from the results in Table 2 that scheduling without any DVS algorithm, consumes a lot of energy per task, therefore battery finishes earlier giving very little efficiency. *ccEDF* and *laEDF* [2] [10], while being energy efficient are not able to fully utilize the battery capacity, therefore results in low battery lifetime. The BAS-1 uses a better priority function making it more energy efficient, and hence is able to extract a larger amount of charge from the battery, extending battery lifetime and performance. But BAS-2 allows more tasks to be available for consideration for the priority function making it more effective and resulting in the best battery performance.

[1]Normalized w.r.t optimal schedule

[2]Extended to handle Task Graphs

Scheme	DVS Algo.	Priority fct	Ready list	Charge Delivered (mAh)	Battery Life (min)
EDF	None	Random	most imminent	1567	74
Cycle Conserving	*ccEDF*	Random	most imminent	1608	101
Look Ahead	*laEDF*	Random	most imminent	1607	120
BAS-1	*laEDF*	p_{UBS}	most imminent	1723	137
BAS-2	*laEDF*	p_{UBS}	all released	1757	148

Table 2. Results of Simulations for different scheduling algorithms when utilization was kept at 70%

6. Conclusion

Energy-autonomous embedded systems have an attached finite-capacity energy source - a battery, that must be relatively small and light for the embedded system to be mobile. Consequently, the system energy budget is severely limited, and efficient energy utilization becomes one of the key problems in the context of battery-powered embedded computing. In this paper we have presented a Battery-aware Scheduling Methodology that facilitates the combining of a good DVS algorithm with a heuristic based priority function for scheduling of conditional taskgraphs. Simulations carried out on random tasks graphs and the profiles generated by various algorithms illustrate that our methodology performs up to 47% better than *ccEDF* and upto 23.3% better than *laEDF* scheduling schemes in terms of battery lifetime. It can result in up to 100% improvement in battery lifetime over systems with no DVS. Since the simulated taskgraphs are periodic, this is also a good measure of the amount of work that was done by the system before the battery was discharged. The methodology presented has also been shown to be generic, and can be used with little or no changes with any frequency setting algorithm and any priority function without deadline violation.

References

[1] P. Chowdhury and C. Chakrabarti. Battery aware task scheduling for a system-on-chip using voltage/clock scaling. *IEEE Workshop on Signal Processing Systems (SIPS '02)*, Oct. 2002.

[2] R. Dick and W. Wolf. Task graphs for free (tgff). *http://helsinki.ee.princeton.edu/ dickrp/tgff/*.

[3] T. F. Fuller, M. Doyle, and J. S. Newman. Modeling of galvanostatic charge and discharge of the lithium polymer insertion cell. *J. Electrochem Soc.*, 140:1526–1533, 1993.

[4] B. Gaujal, N. Navet, and C. Walsh. Shortest path algorithms for real-time scheduling of fifo tasks with optimal energy use. *ACM Transactions on Embedded Computing Systems*, 4(4), 2005.

[5] F. Gruian. *Energy Centric Scheduling for Real-Time Systems*. PhD thesis, Lund University, Lund, Sweden, 2002.

[6] E. L. Lawler. Sequencing jobs to minimize total weighted completion time. *Annals of Discrete Mathematics*, pages 75–90, 1978.

[7] J. Luo and N. K. Jha. Battery-aware static scheduling for distributed real-time embedded systems. In *DAC'01: Proceedings of the 38th conference on Design automation*, 2001.

[8] J. F. Manwell and J. G. McGowan. Extension of the kinetic battery model for wind/hybrid power systems. *Proceedings of EWEC*, pages 284–289, 1994.

[9] D. Panigrahi, C. Chiasserini, S. Dey, R. Rao, A. Raghunathan, and K. Lahiri. Battery life estimation of mobile embedded systems. *In Proceedings of International Conference on VLSI Design*, pages 55–63, January 2001.

[10] P. Pillai and K. G.Shin. Real time dynamic voltage scaling for low powered embedded systems. *Operating Systems Review*, 35:89–102, October 2001.

[11] D. Rakhmatov and C. Chakrabarti. Battery conscious task sequencing for portable devices including voltage/clock scaling. *DAC 2002.*, 2002.

[12] V. Rao and G. Singhal. Integrated power management for embedded systems. *Bachelors Thesis, Indian Institute of Technology, Delhi*, 2005.

[13] V. Rao, G. Singhal, A. Kumar, and N. Navet. Battery model for embedded systems. *In Proceedings of International Conference on VLSI Design*, pages 105–110, January 2005.

[14] S. Vrudhula and D. Rakhmatov. Energy management for battery powered embedded systems. *ACM Transactions on Embedded Computing Systems*, pages 277–324, August 2003.

[15] I. P. Y. Chai, S.M Reddy and B. Al-Hashimi. Battery aware dynamic voltage scaling in multiprocessor embedded system. *IEEE ISCAS, Japan.*, 2005.

[16] D. Zhu, R. Melhem, and B. Childers. Scheduling with dynamic voltage/speed adjustment using slack reclamation in multi-processor real-time systems. *IEEE trans. on Parallel and Distributed Systems*, pages 686–700, 2003.

[17] J. Zhuo and C. Chakrabarti. An efficient dynamic task scheduling algorithm for battery powered dvs systems. In *ASPDAC'05: Asia and South Pacific Design Automation Conference*, 2005.