A Battery Aware Code-partitioning and Scheduling Scheme for Realtime Embedded Systems

Anirban Lahiri, Anupam Basu, Monojit Choudhury, Srobona Mitra, Bhargab B. Bhattacharya†

***Abstract*— The work in this paper is motivated by the growing importance of battery life in mobile embedded systems. It attempts to enhance the battery life of such devices with the help of a novel battery aware code-partitioning and scheduling scheme. The techniques introduced in this paper take into account recent findings, about battery characteristics, in order to further improve battery lifetimes. Results indicate the efficacy of the scheme in improving system battery lifetimes, while at the same time ensuring that applications meet their realtime performance objectives.**

***Index Terms*— embedded system design, battery optimization, scheduling, code-partitioning**

I. Introduction

MOBILE embedded devices have gained wide popularity over the last few decades. The functionality and performance expectations of such devices have also increased manifold over time. In order to meet this growing demand for high performance, mobile handheld devices need to pack in a greater number of system components. Further coupled with this is the requirement for small size and low power consumption.Thus posing a difficult challenge for embedded system designers.

Advances in VLSI technology, has been able to address some of these design issues through the SoC(*System-on-Chip*) design paradigm. In the SoC design paradigm, a number of components or IP(intellectual property) cores are fabricated on a single silicon die or chip. Such components may include programmable general purpose processors, ASIPs (*Application Specific Instruction-set Processors*)- like DSP processors and even RF-communication cores. This allows several co-processors, ADCs /DACs, memory cores etc. to be packaged along with the processor, thus reducing the communication delay between each of the components on the chip and resulting in a significant performance enhancement. SoCs also allow parallel execution of a number of tasks on different resources present on the chip, leading to a substantial speed-up in the applications executed on the chip.

As SoCs include more and more functional components, their power consumptions also increase correspondingly. A major portion of this power consumption may be attributed to active switching of digital circuits on the processor. This is often referred to as the dynamic power dissipation of the processor. However, a large on-chip power dissipation leads to a number of adverse effects like reduced system reliability and excessive cooling requirements. This is sometimes manifested in the form of packaging costs for the chip. Diverse approaches have been proposed in literature to address the issue of power reduction like [4], [33]. Scheduling has been found to be an effective tool in attaining low-power operation in embedded systems and its importance has has increased manifold in the SoC era. Numerous approaches for lowering peak power consumption through scheduling approaches have been proposed in literature, for instance [1], [36]. However, it may be noted that power aware real-time scheduling algorithms must ensure that all timing constraints specified on the system are always met.

In recent times, mobile devices typically rely on batteries for their operation and their usefulness is limited by their battery life. This has brought into the forefront a new design metric for portable embedded devices - battery life. It may be emphasized that low-power operation alone does not guarantee maximum battery life. Battery life depends to a great extent on a number of important properties or characteristics exhibited by batteries. Work in this area has revealed many interesting facts and phenomena about such characteristics and how they affect battery operation. Therefore it is of great import that these characteristics and phenomena be incorporated into the design flow for battery powered mobile embedded systems, in order to order maximize battery life.The various battery characteristics and how they can be suitably exploited using battery aware scheduling and code-partitioning has been shown in this paper.

The work described in this paper has been primarily motivated by real life design on the TI Innovator development platform. The TI Innovator board used for the purpose of this work is based on the OMAP 1510 processor which incorporates both a general purpose processor as well as a DSP core within the same package. This allows for parallel execution of code on more than one processor core. Furthermore special purpose cores also help in achieving significant speed-up for certain classes of applications. As more and more computation intensive applications are being ported to mobile embedded systems such architectures with multiple cores on a single embedded processor are becoming common. Examples of such processors include OMAP from Texas Instuments [15], Nomadik from ST Microelectonics [25] and Nexperia processors from Philips [29]. However, running multiple core parallely puts additional burden on the system battery and thus, battery life may be compromised. The work in this paper redeems such a scenario and addresses the issue of ensuring realtime performance of the system without reducing the battery lifetime. The mentioned work is highly relevant to the current embedded system design flows, since

† Bhargab B. Bhattacharya is with the Indian Statistical Institute, Kolkata, India

The remaining authors are with the Department of Computer Science and Engineering, Indian Institute of Technology Kharagpur, India.

the trends showing the rise in system power consumption and the growth in battery capacitiy reveal a fast widening gap between the two [18].

The techniques proposed in the paper attempts to extend battery life of portable embedded systems with the help of proper code-partitioning and scheduling.The paper first provides a brief overview of previous work in this area and a few elementary facts about batteries,which would help the reader appreciate the elegance of the solutions presented. It then describes a novel recovery based scheduling scheme that takes into consideration battery characteristics that have been brought into light as a result of recent work in the area of battery modeling [31]. This is followed by a battery aware code-partitioning technique which, in conjunction with the above scheduling technique allows for further extension of battery life. The paper presents results obtained both for random test benches as well as practical design cases. The paper concludes by a brief explanation of the obtained results followed by a summary of the contributions.

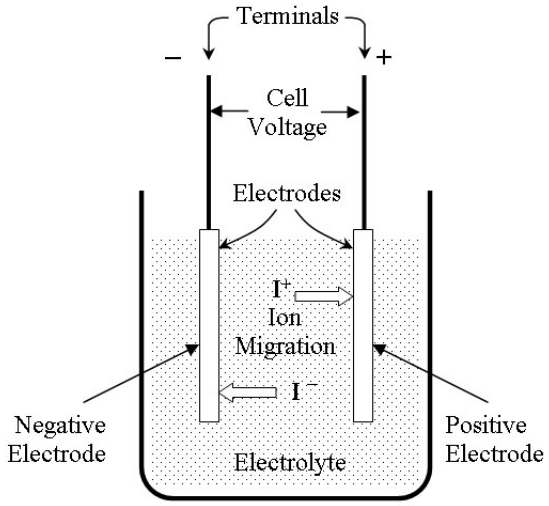

Fig. 1. Structure of an electrochemical cell

II. Related Work

Several power aware scheduling techniques for realtime systems are available in literature. Liu et. al. proposed a scheduling technique that satisfies min/max timing constraints of tasks while trying to maintain the power consumption of the system within a certain predetermined level [21]. A greedy, slack stealing, power aware scheduling algorithm for AND/OR graphs, that also considers dynamic voltage scaling (DVS) has been developed by Zhu et. al. [36]. Alvarez et. al. proposed a technique for simplifying the variable voltage scheduling problem, which may be formulated as a multiple-choice knapsack problem (MCKP), and then solve it using a heuristic algorithm [1]. Shin et. al. developed an algorithm for power conscious scheduling which takes into account that all tasks in a hard real-time system that do not take their worst case execution time (WCET) and this can be exploited to obtain power savings [32]. Luo et. al. developed a power aware joint scheduling scheme for periodic and aperiodic tasks in a real-time system [23]. However, the major objective of these scheduling techniques is to reduce the peak power consumption of the system. The power discharge profile and battery characteristics have not been taken into consideration in these scheduling strategies.

Some of the above mentioned works have been extended to successfully to achieve efficient battery aware scheduling [22]. However, new aspects of battery recovery have been suggested by recent work [31]. The current scheduling technique incorporates them in order to further enhance the system battery life. Strong heuristics have been used to put definitive bounds on an otherwise intractable problem [12].

Code-partitioning forms the other important aspect of the problem of enhancing battery life on mobile embedded systems. It may be mentioned that code-partitioning in tandem with suitable scheduling can produce significantly greater extension to battery life than a battery aware scheduling scheme [17] alone. Also, previous work has shown the advantages of functional partitioning of embedded systems for achieving both higher performances as well as reducing the system power consumption [10], [34]. However, none of these consider the effect of functional partitioning on the battery life of the system which, as mentioned previously, is becoming increasingly important. This paper proposes a battery aware functional partitioning approach which is then used along with the mentioned battery aware scheduling technique, in an iterative way, to arrive at an optimal design. The code-partitioning technique along with the battery aware scheduling shows a new design scheme, for mobile embedded systems, that considers the system battery lifetime as an important design metric.

III. Battery Fundamentals

A battery is necessarily an electrochemical energy storage system and is composed of one or more electrochemical cells. In each cell a positive and a negative electrode is immersed in an electrolyte as shown in figure 1 . Energy is released is due to a set of electrochemical reactions taking place at each of the two electrodes in the cell.

Compounds with higher energy content are converted into those with lower energy content as a result of these electrochemical reactions leading to charge deposition at the electrodes, thus enabling them to drive a current in an external circuit. A part of the energy is also lost in the process as Joule heating. The electrochemical reactions along with other physical specifications determine the characteristics of the battery which are represented by several parameters. These will be explained in the following subsection.

A. Battery Parameters

Each battery can be characterized by a number of parameters. These parameters indicate the charging and discharging characteristics of the battery and also its charge storing ability. Hence, such parameters are directly related to the operational life of the battery and need to be considered

carefully while selecting a battery for a particular application or device. Summarized below are some of the more important parameters [3], [16], [20]. These include:

(a) Battery Voltage

- *Open Circuit Voltage (OCV) - V_{CO}*
  It is the voltage observed across the terminals of the battery when no current is being drawn from it. Commonly, the initial open circuit voltages are specified, which is the voltage observed across the battery terminals when the battery is fully charged.
- *Cut-off Voltage - V_{cut}*
  If the voltage of the battery falls below the cut-off voltage then the battery is assumed to be fully discharged and hence rendered unsuitable for further operation. It is also known as the end-of-discharge (EOD) voltage.

(b)Battery Capacity

- *Theoretical Capacity*
  The theoretical capacity is the maximum possible amount of charge that can be stored in the battery. Hence it is implied that maximum charge that can be extracted from the battery is equal to the theoretical capacity. However, for all practical cases the amount of charge that the battery delivers is much less than the theoretical capacity.
- *Standard or Nominal Capacity*
  The amount of charge delivered by a battery, when discharged by a constant current, under standard discharge conditions is referred to as the standard or nominal capacity. The standard capacity of the battery is measured in *Ah*(Ampere-hours) as a convention. Parameters that much be specified while mentioning the standard capacity include
  - Discharge current
  - cut-off voltage
  - temperature

  Eg. 800mAh at a constant current of 100mAh at $27\,^{\circ}\mathrm{C}$
- *Actual or Available Capacity*
  The actual capacity is the amount of charge delivered by the battery for a particular discharge profile.

B. Battery Effects

Almost all mobile embedded devices today, are powered by rechargeable batteries. Rechargeable batteries belong to various types Eg. Nickel Cadmium, Nickel Metal-Hydride, Lithium Ion, Reusable Alkaline, Lithium Polymer. However, all these batteries exhibit some common properties. These include:

(a) Rate Capacity Effect The rate capacity effect indicates that if the battery is discharged at a higher rate then a lower actual capacity is obtained. This may be illustrated with the help of an example below.

Example 1 : Consider a battery which when subjected to a constant discharge current of 100mA runs for one hour. Hence the available capacity of the the battery for above discharge profile is 100mAh. Now if the same battery is subjected to a constant discharge current of 200mA then it lasts for only 22 minutes instead of the expected 30 minute discharge period. Hence it yields an available capacity of 73.33mAh instead of the 100mAh obtained in case of the first discharge profile. This can be attributed to the rate capacity effect exhibited by the battery.

The chemical process corresponding to the rate capacity effect may be described as follows. Under normal conditions, or at low discharge rates, all reaction sites on the electrodes are equally active. On the other hand at high discharge rates, most of the reaction takes place on the outermost surface of the electrodes. These reactions are often followed by the formation of insoluble or sparingly soluble compounds. Thus slightly remote reaction sites on the electrodes are permanently cut-off from any further reaction. This leads to a decrease in the overall battery capacity [9].

(b)Recovery Effect During the process of discharging if the battery is allowed certain periods of rest (i.e. zero or negligible discharge current) then it tries to recover a part of its original voltage. This in turn produces an enhanced available capacity for the battery. In other words the battery has an improved performance if its discharge is spaced out by intermittent idle periods.

The phenomenon can be explained with the help of chemistry as follows. Initially the concentration of the active ions in the electrolyte is uniform throughout the cell. Now, when the discharge process takes place at a continuous rate, then the active active ions nearer to the electrodes are depleted at a rate much faster than the rate at which they are replaced by other active ions farther away from the electrodes, through the process of diffusion. Again, if the cell is allowed an idle period then remaining active ions in the cell will reposition themselves to reach uniform distribution within the cell [11]. This often leads to a significantly longer battery life.

C. Battery models

Battery models try to emulate the behavior of practical batteries by taking into consideration the above mentioned battery effects and properties. Such battery models can be used to accurately estimate the battery life under specific load conditions. A large number of battery models have been proposed in literature [2], [9], [11], [13], [14], [24], [27], [28], [30], [31]. These can again be classified into analytical, stochastic, electrochemical, electric circuit models as suggested by [19]. Hybrid models, spanning more than one of the above types, may be considered as a separate class. The mentioned models are able capture the battery properties and effects to different extents. For the purpose of the current work, a simulation framework has been developed. This framework uses a modified Kinetic Battery Model [31] in order to perform battery life estimation in a cycle accurate manner. Hence it useful for evaluating various schedules obtained from the battery-aware scheduler. The major advantage of the mentioned battery model is that it is able to capture the aspects of recovery effect not considered in previous models. These effects have been

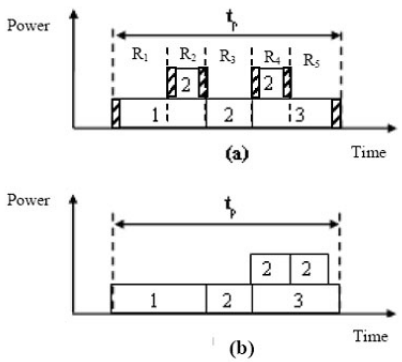

Fig. 2. Power aware Gantt chart showing (a) feasible (b) battery optimized schedules

utilized in the code-partitioning and recovery based scheduling techniques described in this work.

IV. Battery Aware Scheduling

As previously mentioned, a higher rate of discharge and variance in the discharge current reduces the available or actual capacity of the battery. It has also been noted that a pulsed discharge current can produce a longer battery life than a constant discharge current due to recovery effect [5]. These advantages of a pulsed discharge current also hold when it is superimposed on a constant current [5]. Further investigations suggest that the battery performance and hence the battery life improves with the increase in the time-period of the pulsed discharge current [31]. The effect of the above mentioned property and its relevance to battery aware task scheduling may be illustrated using an example as given below.

Example 2: Considering a set of 3 periodic tasks with a hyper-period (relative deadline) t_p. A feasible schedule and its corresponding battery aware schedule, which takes into account the above mentioned effects, for the tasks may be shown in the form of power aware Gantt charts [21] as in figure 2.

From figure 2 it may be noted that the peak power and the variance of the discharge current remains the same in both cases. However, in the second case task no. 2 has been shifted in time to obtain a battery aware schedule taking recovery effect into consideration. It has been found through simulation, using a modified Kinetic battery model [31], that the battery life in the second case can be greater than that of the first case by up to 3%. This leads to the notion of power glitches. A glitch in this case is a sudden transition in the power discharge level. Such power glitches have been shown by thick striped lines in 2(a). It has been experimentally observed that the battery life may be extended by removing or reducing such power glitches from the discharge profile. This observation also holds true in example 2 where an improvement of up to 3% is observed in the battery lifetime by the removal of just a single pair of power glitches. The scheduling algorithm described in this section reduces these power glitches in an elegant manner in order to extend battery lifetimes.

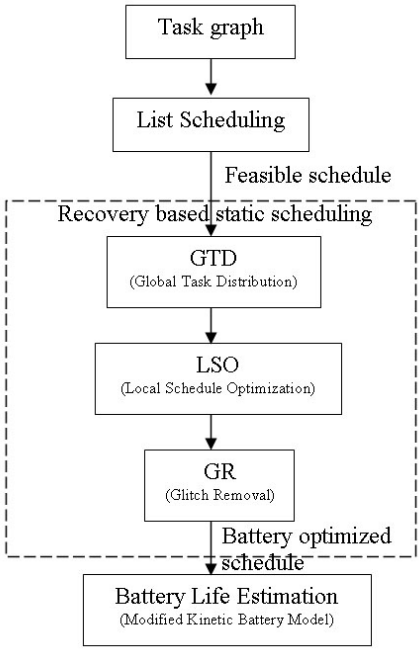

Fig. 3. Block diagram of the scheduler

A. Recovery Driven Scheduling

Real-time embedded systems typically rely on static schedulers for their operation. A discussion on the relative merits and demerits of such schedulers are beyond the scope of the current work. In this section a novel battery-aware, real-time static scheduling approach has been proposed. The scheduling technique takes into consideration hard deadlines corresponding to individual tasks as well as deadlines for the entire system as a whole. It also considers the relative interdependencies between tasks. Therefore, the scheduling approach is applicable to a large class of scheduling problems for embedded systems. Obtained results prove the efficacy of the scheduling technique as it achieves up to 29.41% further increase in battery lifetime over previous such scheduling approaches [22] and a total battery life extension of up to 37.7%.

The overall structure of the scheduler may be shown as in figure 3. The different components of the scheduling scheme have been discussed in detail in the subsequent paragraphs. The cost function used for the scheduling process has also been explained later in this section. The working of the scheduler and tasks performed by the scheduler in each step has been illustrated with the help of examples.

An input to the scheduler is in the form of a conditional task graph with hard deadlines. Each task is represented by a node in the task graph, while the edges in the task graph correspond to inter-task dependencies. It is assumed that task graph contains two special vertices - a source and a sink vertex as shown in example 3 below. If either of these vertices does not exist in the task graph then such vertices may be added to the task graph and a dummy tasks may be created corresponding to them. Each task is assumed to have a worst case execution time, a worst case power consumption,

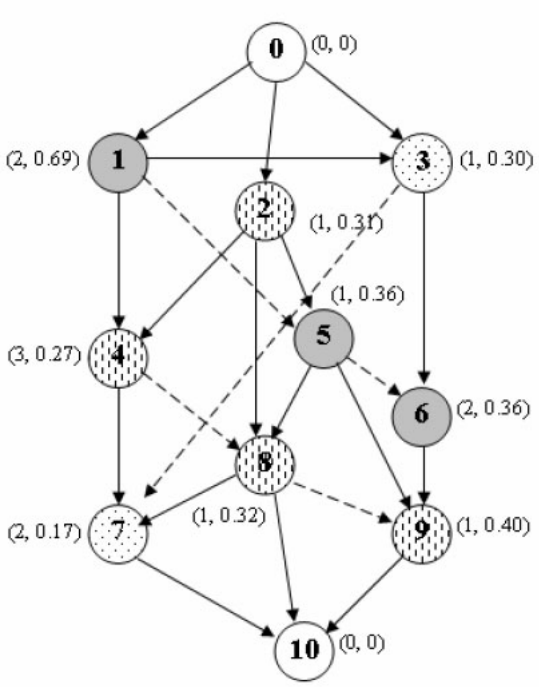

Fig. 4. Task graph for example 3

resource requirements, and deadlines if any. For the purpose of scheduling time is considered to be divided into slots of equal length referred to as "units".

Example 3 : Consider an embedded system specification given in the form of a task graph as shown in 4. Assume that the tasks have to be scheduled on 3 different processor cores or resources. The maximum execution time and maximum power requirement, in Watts, of each task is indicated respectively within a pair of parentheses alongside each task. The task graph shown has a hyperperiod of 10units. The hyper-period is considered as a hard deadline for all tasks.

A standard list scheduler [7] is used to obtain an initial feasible schedule for the task graph. It may be noted that as a result of list scheduling a relative ordering is obtained between all tasks scheduled on a processor. This can be represented using an augmented task graph as shown in figure 4. The new interdependencies arising due to list scheduling process has been shown using dotted lines in the task graph. Furthermore, a simple topological search on the augmented task graph yields the earliest and latest possible execution times of all tasks. The initial feasible schedule is then subjected to a number of distinct phases in order to obtain a battery aware schedule. The order of the phases have been shown in figure 3. The schedule finally obtained is evaluated using the cycle accurate battery life estimation framework, as mentioned previously. The different phases of the scheduling algorithm has explained in the following subsections.

B. Global Task Distribution (GTD)

In this phase tasks which are scheduled on a single processor are spaced out in time. This helps in two ways. First, the total energy consumption of the system is distributed over the entire hyper-period of the task graph. This ensures that the formation of high power consumption regions in the discharge profile is minimized. The second important advantage is that it paves the path for improving the discharge profile through local schedule transformations, which is carried out in the next phase.

The GTD phase works in an iterative manner, where tasks assigned to the same processor, by the list scheduler, are are scheduled in a single iteration.The process starts with the core having the highest utilization and proceeds in the decreasing order of the core/processor utilizations. An average spacing out time T_{AvgSpc} is calculated at the beginning of an iteration corresponding to a particular processor. T_{AvgSpc} refers to the time interval that is inserted between any two tasks on that processor and thus depends on the amount of idle time available on that processor/core. However, it may not always be possible to insert an idle period exactly equal to T_{AvgSpc} between any two tasks due to realtime constraints. Such realtime constraints are indicated by the earliest and latest possible start times of each task, which must be obeyed at all times during the scheduling process. In such cases T_{AvgSpc} must be recomputed after the task is scheduled in order to reflect the remaining available idle time corresponding to the processor. If at any time, T_{AvgSpc} assumes a fractional value then each time it is used, a random decision is made either to use a floor or a ceiling function to round it off to an integral number of time slots, provided the earliest and latest execution times are not violated for the immediately following task. It may be emphasized that the GTD phase allows for greater power savings than a threshold based scheme [22] for distributing tasks. Also, no human intervention is required for fixing such thresholds. The working of the GTD phase has been illustrated through example 4.

Example 4 : Considering the embedded system specified in example 3 as the input to the scheduling algorithm. The power aware Gantt charts representing the schedule after the first phase has been shown in figure 5.

C. Local Schedule Optimization (LSO)

The primary objective of this phase is to improve the discharge profile by lowering the local maxima for different regions and eliminating glitches across the entire power discharge profile. This is achieved by through a process of local schedule optimization using an iterative task shifting scheme. The schedule obtained at the end of this phase minimizes the battery losses as a result of rate-capacity effect and improve the benefits from recovery effect shown by the battery.

An empirical cost function has been developed for the purpose of estimating the effect schedule transformations on the battery life. In order to use the cost function the entire power discharge profile may be considered as a dynamic set of rectangles {R1, R2,..., Rn}, where the area bounded by each rectangle represents the corresponding energy consumption as shown in 5. The reasoning behind choosing an empirical cost

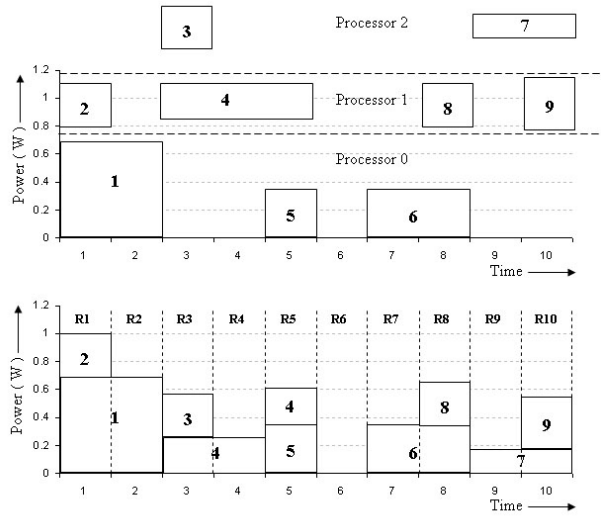

Fig. 5. Power-aware Gantt chart showing schedule after GTD phase

function instead computing the actual energy drawn from the battery, during the scheduling process is as follows.

First, the exact power drawn from the battery need not be calculated at each step during the scheduling process in order to arrive at a battery aware schedule. Second, the computational cost required for calculating the exact power drawn from the battery at each step can be avoided. A further advantage of such a cost function is that it can be computed only for a portion of the discharge profile under consideration instead of the entire profile.

The use of the cost function has been explained in detail in the following paragraph.

$$F_s = \alpha \sum_{i=1}^{n} (1 + H_i)^2 L_i + (1 - \alpha) \sum_{i=1}^{n} \Delta H_i \qquad (1)$$

Where,
α = coefficient of recovery [0, 1]
H_i = height of the rectangle R_i
L_i = length of the rectangle R_i
$\Delta H_i = max(H_i - H_{i-1}, 0) + max(H_i - H_{i+1}, 0)$
ΔH_i may be referred to as the power glitch for rectangle R_i.

The cost function is the sum of two terms, which represent rate capacity effect and recovery effect respectively. The factor α is used to adjust the relative weights attached to each of the two effects and has been referred to as the coefficient of recovery. The value of α is determined by the parameters of the battery used. For all practical purposes α assumes values between 0.5 and 1. As previously mentioned, the battery consumption is super-linear in terms of the discharge current or power drawn, due to rate capacity effect [28]. This fact can also be confirmed from Peukerts formula [20]. In order to capture this property H_i must raised to a power greater than unity. Thus to reduce the associated computational load the cost function has been made quadratic in terms of H_i. Simulations have proved that the cost function achieves its

```
LSO( )
{
    Lock(τ) = FALSE ,∀ τ ∈ tasks
    P = {set of rectangles constituting the power profile}
    repeat
    {
        R = max_height(r : r∈ P and ∃ τ ∈ r and Lock(τ) = FALSE);
        T = max_slack_time{τ : τ ∈ R and Lock(τ) = FALSE };
        result = Shift_Task(R, T);
        if(result = NULL)
            Lock(T) = TRUE;
        Else
            Compute_Power_Profile(P);
    }until(Lock(τ ) = TRUE, ∀ τ ∈ tasks)
}

Shift_task(R, T)
{
    R′ = min_distance(r, R){r : r ∈P, height(r) + height(T) ≤ height(R)}
    ΔF = Contemplate_Move(R, T, R′)
    If (ΔF < 0)
        Lock(T) = TRUE;
    Else if (ΔF > 0)
    {
        Make_Contemplated_Move(R, T, R′)
        Release_All_Locks( );
    }
}
```

Fig. 6. Pseudocode for the LSO phase

intended goal of reflecting the battery life. The cost function is used extensively both in the current phase as well as the next phase of the scheduling algorithm in order to evaluate the result of various local schedule transformations.

The working of this phase can be summarized as follows. The algorithm at each step determines the rectangle with the highest height. It then tries to lower this height by local shifting of tasks. It also keeps track of whether the value of the cost function is reduced as a result of these transformations. This enables the algorithm to consider both rate capacity and recovery effect simultaneously. The cost function as specified previously takes into consideration new aspects of recovery effect. This allows for further extension of battery life. The algorithm for LSO phase may be represented in pseudo-code as given in figure 6.

The function $Comtemplate\_Move$ found in the pseudocode given in figure 6 above determines the amount by which the selected task can to be moved into the newly selected rectangle so as not violate any of its constraints and calculates the corresponding reduction in the cost function (denoted by ΔF). It may be mentioned that the cost function is recomputed only for the portion of the schedule that has been altered. This significantly increases the efficiency of the algorithm. If an improvement in the schedule is detected as a result of this contemplated task movement then the task is actually shifted using the $Make\_Contemplated\_Move$ function.

Example 5 : The schedule obtained, in example 4, after the first phase of the algorithm is taken as input to the second

```
Remove_Glitch( )
{
        H = Compute_Glitch(∀r : r∈ P)
        R = max_H{r : r∈P Λ Lock(R) = FALSE}
        R′ = min_distance(r,R){r : r_H > 0}
        T = first { τ : τ∈R Λ τ∉precedes(R) Λ τ∉succede(R) Λ Lock(τ) = FALSE }
        If (T = NULL)
                Lock(R);
        If (t_R > t_R′)
                ΔF = ComtemplateMove(R,T,succede(R′))
        else
                ΔF = ComtemplateMove(R,T,precede(R′))

        If (ΔF > 0)
        {
                Make_Contemplated_Move(R,T,R′)
                Release_All_Locks( );
        }
        else
        {
                Lock(T)
                If (Lock(τ) = TRUE,∀τ∈R)
                        Lock(R)
        }
}
```

Fig. 7. Pseudocode for the GR phase

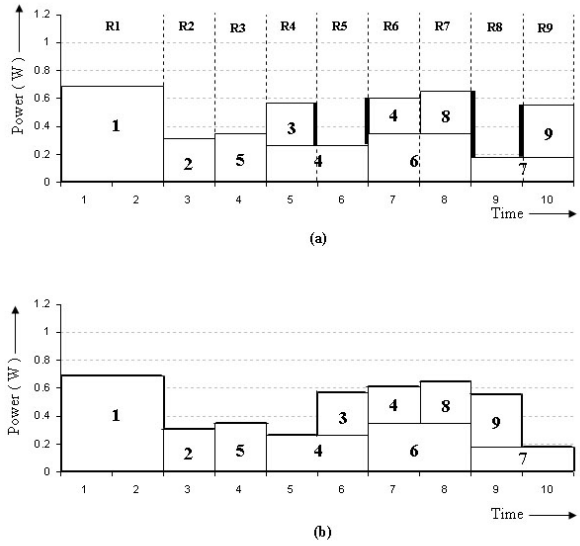

Fig. 8. Schedule (a)after the LSO phase (b)after the GR phase

phase. The schedule obtained at the end of the LSO phase is shown in figure 8(a).

D. Glitch Removal (GR)

A glitch in this case, as mentioned before, refers to an abrupt change in the power drawn by the system from the battery. It has been found through experiments that such glitches are deterimental to the battery life. The objective of this phase of the algorithm is to reduce the total amount of glitching the system undergoes during each hyperperiod. The procedure followed during this phase is shown in figure 7 in the form of pseudocode.

Example 6 : The schedule obtained from the previous phase of the scheduler, as shown in example 5 is taken as input to the third phase. It may be observed from figure 8(a) that the tasks 9, 1 and 3 produce significant power glitch, shown by thickened lines. However since task 1 cannot be shifted, tasks 3 and 9 are shifted to reduce the glitch as shown in figure 8(b). This is the battery aware scheduled that is finally obtained.

V. Battery Aware Code-partitioning

As mentioned before the use of multi-core processors in the embedded systems domain is increasing rapidly. This owes itself to the fact that, application specific cores on the processor contributes in achieving significant performance enhancements. DSP oriented applications like speech processing including speech synthesis and recognition, image/video decoding or enhancement, object recognition and tracking, implementation of security protocols, graphics rendering are some of the classes of applications that benefit immensely from such hardware acceleration. It may be emphasized that often the power consumption of such hardware accelerators exceed that of the on-chip general purpose processor by up to a few orders in magnitude. In such cases code-partitioning along with scheduling can help achieve performance gains without affecting the battery life. System power reduction with the help of functional partitioning has been studied previously [10], [34]. However, the effect of functional partitioning on battery life has not been considered by them. The current work analyzes how functional partitioning can ensure high performance as well as extend battery life.

A. Processor Architectures

Detailed knowledge about the multi-core processor architecture is essential in order to perform code-partitioning targeted towards that processor. This includes information about which cores on the processor can be used to execute a specific task. In order to perform battery aware code-partitioning further information is required about the worst case execution time and maximum power consumption that is associated when a given task is executed on a particular processor core.

Interprocessor communication among the multiple processor cores forms another important aspect that needs to be considered during code-partitioning and scheduling. Dedicated bus systems and shared memory paradigms are commonly used for implementing interprocessor communication. This may be further supplemented by a scheme where the sending processor notifies the receiving processor, about the message, by raising an interrupt. Mailboxes act as another important layer of abstraction over the actual communication infrastructure. Different processors use various combinations of these schemes, APIs(Application Programming Interfaces) are usually provided to simplify the task of programming. In the following paragraph the architecture of one such commercially available multi-core embedded processor, the TI OMAP 1510 has been briefly discussed. There are two primary reasons for the mentioned discussion. First, it provides a practical perspective as to how the mentioned code-partitioning and

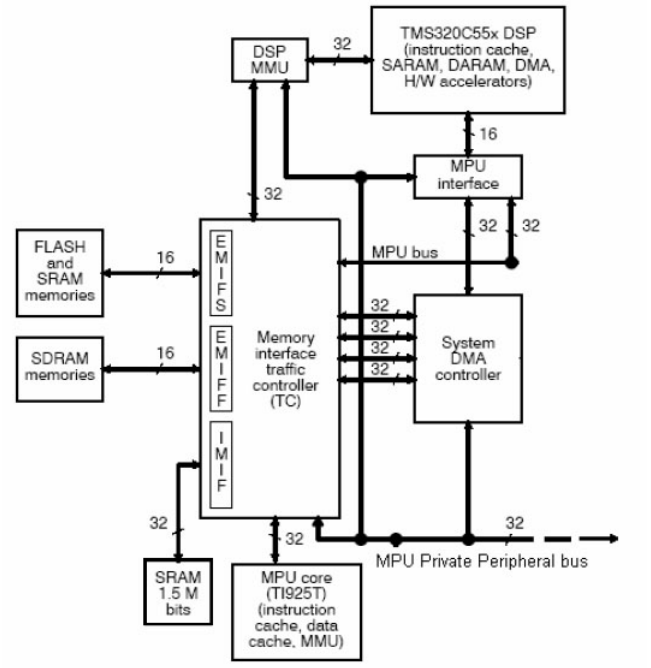

Fig. 9. Relevant portions of the OMAP architecture (courtesy Texas Instruments)

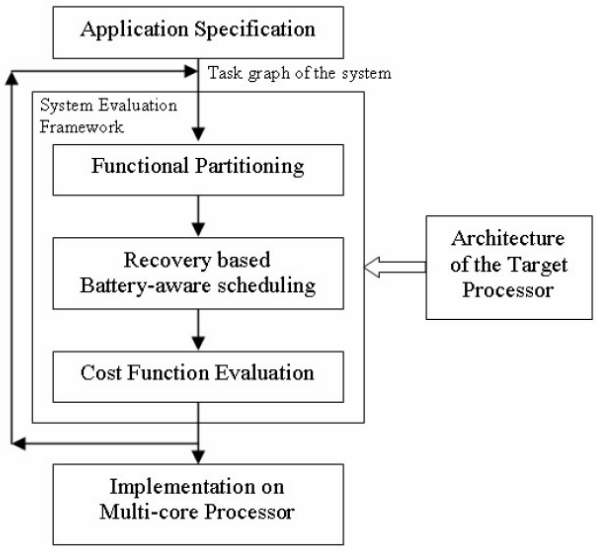

Fig. 10. The system design flow

scheduling scheme may be used in real-life systems. Second, the experiments given in this paper include a practical design scenario where, the code-partitioning and scheduling scheme has been used for a TTS(text-to-speech) engine to be implemented on the TI OMAP 1510 platform.

The TI OMAP 1510 architecture is identical to that of the TI OMAP 5910. Relevant portions of the OMAP architecture has been shown in figure 9. The TI OMAP 1510/5910 is a dual core processor with an ARM9 [6] and a TMS320C55x DSP [15] core respectively. The OMAP provides two facilities interprocessor communication. The first being shared memory while the second is a set of dedicated mailbox registers. When a value is written to one of the mailbox registers by a processor, an interrupt is generated on the other processor. The processor also provides three separate memory interfaces, as shown in figure 9 to reduce memory access conflicts between the processors. The advantages of such a memory controller with multiple interfaces will become apparent when the design decisions with respect to the mentioned practical scenario is discussed in the section on experimental results.

B. Functional Partitioning

The partitioning strategy used in this work is similar to previous functional partitioning approaches [10], [34]. An overview of the complete design flow including functional partitioning and the scheduling is shown in figure 10. The partitioning scheme uses strong heuristics to drastically reduce the search space. For a more comprehensive search through the solution space a GA(genetic algorithm) based scheme may be employed, which has been earmarked as a future work.

The input to the functional partitioning algorithm is in the form of a task graph. Apart from the task interdependencies, the following information is assumed to be known a priori for each task:

- the set of processor cores which can execute a particular task
- worst case execution time of a task when executed on a certain processor
- worst case power consumption for a task, processor pair

Let $\mu_p = \{\mu_{p_1}, \mu_{p_2}, ..., \mu_{p_m}\}$ be the set of *m* processors or cores and $T = \{T_1, T_2, ..., T_n\}$ be the set of tasks specified by the task graph. Nodes in the task graph are considered iteratively by the partitioning algorithm, in a breadth first manner, starting with the source node. A cost function is used for determining the processor to which task T_i must be assigned. When task T_i is being considered for assignment, the cost function is computed for each processor $\mu_i \in \mu_p$, which can possibly execute T_i. The cost function may be written in a *"max-form"* as :

$$F_s = w_1 \cdot \frac{t_{max} - t}{t_{max}} + w_2 \cdot \frac{P_{max} - P}{P_{max}} + w_3 \cdot (1 - \mu_{util}) + w_4 \cdot (1 - \alpha) \cdot B_{pred,i} \quad (2)$$

Where,

t_{max} = maximum, worst case execution time taken by any processor $\mu_j \in \mu_p$ for executing T_i

P_{max} = maximum, peak power taken by any processor $\mu_j \in \mu_p$ for executing T_i

t = worst case execution time when T_i is executed on μ_i

P = peak power consumed by μ_i while executing T_i

α = coefficient of recovery [0, 1]

w_k = weight factor corresponding to the k^{th} term

$$B_{pred,i} = \begin{cases} 1 & \text{if both task } T_i \text{ and one of its predecessor is scheduled on } \mu_i \\ 0 & \text{otherwise} \end{cases}$$

Alternatively, the cost function may be expressed in the *"min-form"* as follows :

$$F_s = w_1 \cdot \frac{t}{t_{max}} + w_2 \cdot \frac{P}{P_{max}} + w_3 \cdot (\mu_{util}) + w_4 \cdot (\alpha - 1) \cdot B_{pred,i} \quad (3)$$

The processor yielding the maximum (minimum) value of the cost function expressed in *max-form*(*min-form*) for task T_i is selected for executing that task. It may be noted that functional partitioning is followed by battery-aware scheduling as shown in the flow diagram in figure 10.Finally, the obtained design is evaluated using a cycle accurate battery life estimation framework, mentioned in a previous section. If the system fails to come up with a design that meet the the real-time constraints or achieve the desired battery life, the design flow may be repeated after suitably changing the various weight factors. Thereby, a trade-off may be reached between the execution speed and the system battery lifetime.

VI. Experiments and Results

Two different classes of experiment were performed for the code-partitioning and scheduling scheme. The first involves randomly generated datasets using a standard benchmarking tool, while the second is based on a practical design scenario. The results corresponding to each of the above experiments is given in the following sub-sections.

A. Randomly Generated Task-graphs

A substantial number of experiments have been carried out on random tasks graphs generated using a standard randomized task graph generator - TGFF [8]. The results obtained from the code-partitioning and scheduling systems were evaluated using a modified Kinetic Battery Model [31] for a battery having a maximum capacity of 2000mAh.

The battery aware scheduling scheme alone yielded a battery life extension in the range of 19 to 34% over a standard list scheduling approach, for task graphs with 5 to 70 nodes. Simulations show that that the battery life extension was greater by up to 21% for systems with a large number of tasks and processor cores. This may be attributed to the increased level of parallelism and flexibility in scheduling options in the case of a larger number of cores. Table I shows the achieved battery life extensions for a representative set of the observations using battery aware scheduling alone. However, it is interesting to note that when the same input task graph was first subjected to the battery aware code-partitioning scheme followed by the scheduling algorithm a further improvement of up to 31% was observed in the battery life. This is over and above the gains achieved by battery-aware scheduling [17] alone. A comparison of the two has been shown in figure 12 for the same data-set.

A Pentium 4 desktop PC with 512 MB of primary memory was used for the purpose of executing the scheduling and code partitioning scheme. While the code-partitioning scheme is of linear order, the execution time taken by the scheduling algorithm is also comparatively much less than that for other approaches, especially an exhaustive search which has an exponential time order, as seen in figure 13.

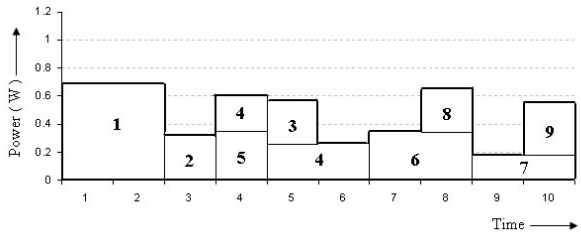

Fig. 11. Schedule without considering specified aspects of recovery effect

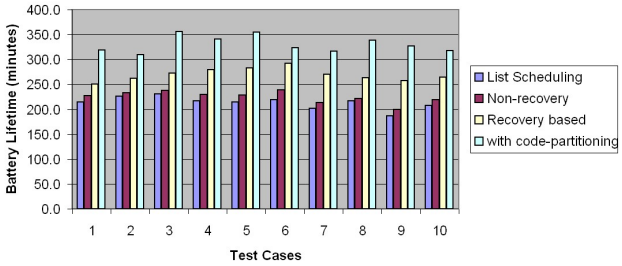

Fig. 12. Battery lifetimes corresponding to different scheduling schemes

It has been further observed that the proposed scheduler also reduces the variance of the power discharge profile as a result of glitch reduction, which has been a major objective in previous battery aware schedulers [22]. The reduction in power glitch may be visualized as the decrease in the geometric perimeter of the maximum power envelope. This length may be measured in any arbitrary units and is used only for comparison between two power aware Gantt charts with the same units. In order to illustrate this figure 11 shows the output of a non-recovery based scheduler corresponding to the task graph in example 3. The recovery based schedule shown in figure 8(b) achieves a battery liftime extension of about 17.49% more than the schedule shown in figure 11, although both schedules have the same peak power for equal amounts of time.

B. Battery Aware TTS Design

As compute intensive applications like those involving speech processing, are being ported to mobile handheld de-

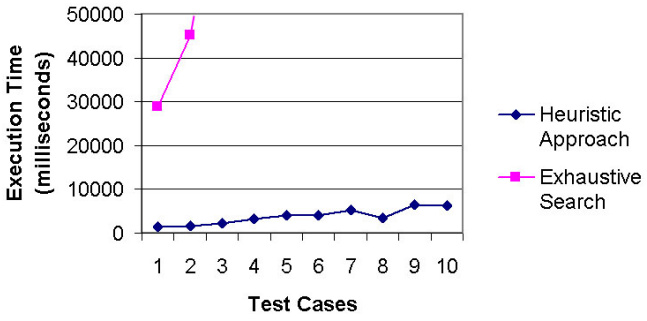

Fig. 13. Execution time for scheduling approaches

#Test	#Tasks	#Cores	Execution Time (milliseconds)	Battery Lifetime (minutes)			Increase in Battery Life (%)	
				Non-optimized	Non-recovery	Recovery-based	Over Non-recovery based scheme	Over Non-optimized scheduling
1	5	2	1376	214.0	227.0	250.8	10.48	17.15
2	7	3	1591	226.4	233.4	262.2	12.34	16.81
3	11	4	2212	231.3	237.6	271.9	14.42	17.71
4	23	5	3178	217.2	229.6	279.2	21.57	28.65
5	23	7	4129	214.7	228.6	283.4	23.96	31.79
6	27	9	3965	218.9	238.7	291.9	22.28	33.28
7	35	9	5192	201.5	213.0	270.2	26.83	34.44
8	37	4	3439	216.61	221.9	262.78	18.42	21.31
9	49	11	6387	187.2	199.1	257.5	29.41	37.71
10	54	10	6266	207.57	219.3	264.76	20.73	27.55

TABLE I
COMPARISON OF SCHEDULING SCHEMES

vices, the performance requirements on the such systems continue to rise. The challenge here is to extend the battery life of such handheld devices without affecting its performance. The experiment discussed in this section is the result of a such a requirement.

It was observed that when Shruti [26] - a concatenative TTS(Text-to-Speech) system was ported from a desktop platform, running at a few GHz, to an embedded ARM processor running, at only a few hundred MHz, the response time of the resulting system was unacceptable. In order to investigate the cause for such enormous delay and explore ways of improving the system response, the application was profiled using the Valgrind toolset [35]. Profiling revealed that the application consisted of two major types of tasks - namely NLP and DSP tasks as shown by the call graph in figure 14. The call graph obtained through Valgrind also indicates that the DSP tasks take up more than thrice the computation power of the CPU than all other tasks taken together. The TI OMAP dual-core processor was selected for this reason, since it has both a general purpose ARM processor as well as a TMS320C55x DSP processor core. However, a supplementary DSP processor would put additional overhead on the system battery. Hence, an efficient code-partitioning and scheduling scheme is required to ensure that both hard-realtime performance constraints are met and battery-lifetime is not compromised.

In order to implement the battery aware code-partitioning scheme the worst case execution times of each task, when executed on a particular processor core, was found through numerous simulations using the TI Code-Composer Studio [15]. The execution cost for the DSP tasks were significantly less on the TMS320C55x DSP processor than on the ARM. Hence, battery aware functional code-partitioning indicated that all DSP tasks, especially filtering (which takes about 60% of the computational time on ARM), be executed on the TMS320C55x DSP processor, while the NLP tasks be executed on the ARM. However, the task graph for the application showed that the DSP tasks were data dependent on the NLP tasks, thus ruling out the scope for parallel execution.

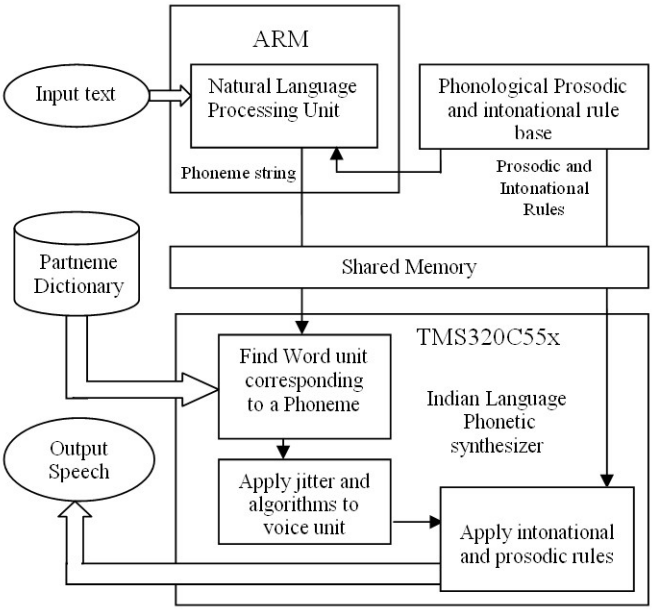

Fig. 15. Functional block diagram of the TTS showing code-partitioning

Interestingly, even in such a case parallel execution of NLP and DSP task instances from overlapping hyper-periods is possible, thus leading to a pipelined execution mechanism. Figure 15 shows the functional block diagram of the system arrived at, after code-partitioning.

Data is transferred between the processors using a combination of shared memory and mailbox facilities provided by the OMAP architecture, as mentioned previously. Writing to one of the mailboxes generates an interrupt to the other processor. however, this mechanism is inefficient for transferring large amount of data. To overcome this, the data is first written to a location in the shared memory. The address of the location is then passed to other processor using the mailbox register. This also helps in establishing a synchronization mechanism between the processors. Furthermore memory access conflicts

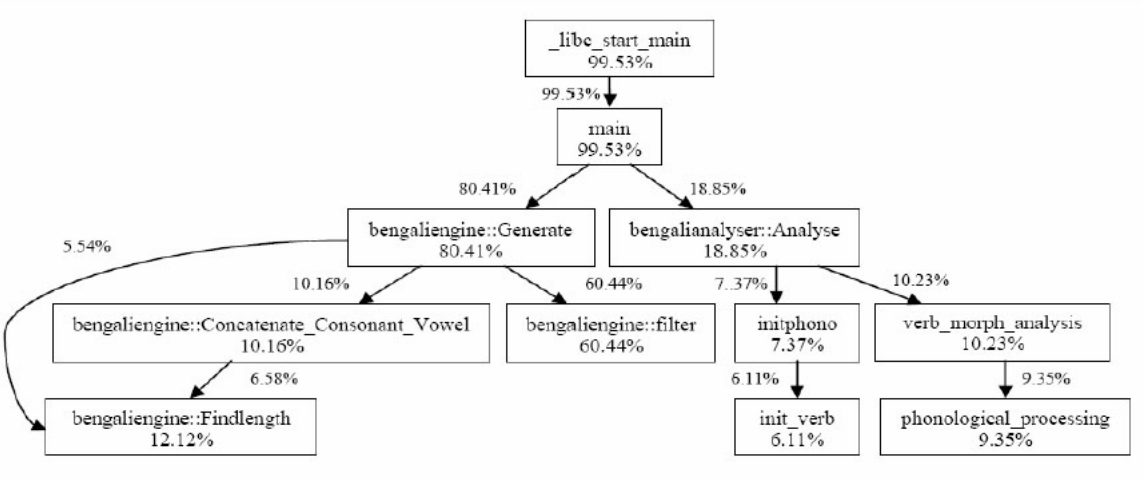

Fig. 14. Call graph for the Shruti text to speech engine

between the two processors have been greatly reduced by exploiting the multiple memory interfaces provided by the OMAP architecture. Resources, like exception lists, lexicons, which are frequently accessed by the NLP tasks running on the ARM9 core are are stored in the internal SRAM shown in figure 9. On the other hand the relatively large (3.5MB) speech dictionary, which is accessed by the DSP processor, is stored in the external memory and is accessed through the external memory interface labeled as EMIFF in figure 9. The memory interface traffic controller allows both memory accesses to take place simultaneously without any conflicts, thus resulting in a greater speedup.

In order to realize the pipelined execution of the TTS, the text input is split into word clusters before processing. A task graph is formed by the tasks corresponding to a word cluster. The deadline or hyper-period of the for the entire task graph should be such that no discontinuity is observed in the speech output, i.e. the system should not produce a jittery speech output. The pipelined execution scheme also helps in reducing the response time of the system. This means that due to its small size the first word cluster will produce an output much faster compared to the response time of the system, if the entire text was processed as a single unit.

Determining the size of the word cluster is another important aspect of the pipelined execution. Abrupt changes in the power discharge profile leads to a lower battery efficiency as stated previously. Increasing the word cluster length helps in reducing such abrupt changes or glitches as shown in figure 16. The figure shows the power-aware Gantt charts for two different cluster lengths. It may be observed that a larger cluster length eliminates the glitches shown by thickened lines in figure 16(a). However, the cluster length may not be increased arbitrarily. Since each cluster is treated as an unit, tasks will require more memory space to process a larger cluster. Hence, a trade-off must be reached between the memory requirement of the system and the possible battery savings.

It has been assumed that the power consumption of other components of the system, like memory, buses, etc., is either

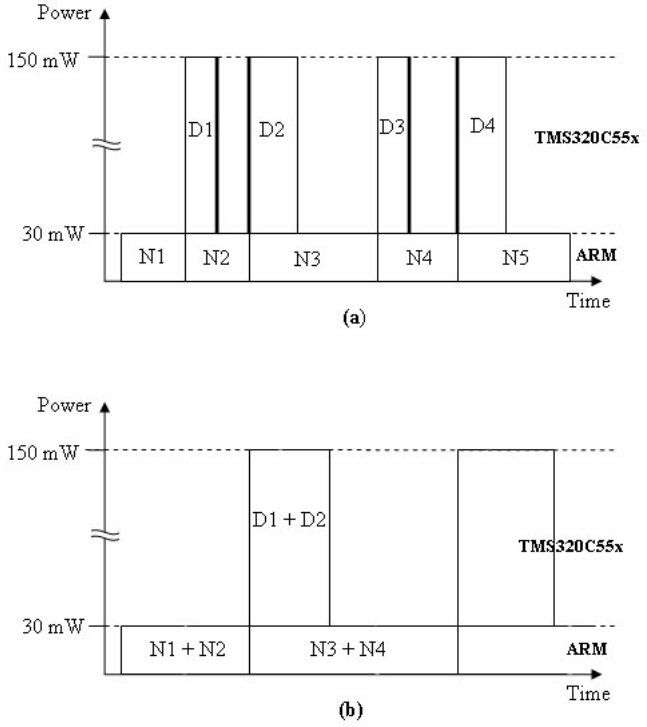

Fig. 16. Power aware Gantt charts showing schedule for (a) five word (b) ten word clusters

constant or can be attributed to the task been executed. Thus, the Gantt may either be shifted long the Y-axis or may be scaled suitably to take into account the power consumptions of other system components. However, in this case only the power consumption of the processor cores have been considered. It may again be emphasized that though the two schedules in figure 16 have the same peak power, their corresponding battery lifetimes may be significantly different. In this case , it has been determined empirically that the battery

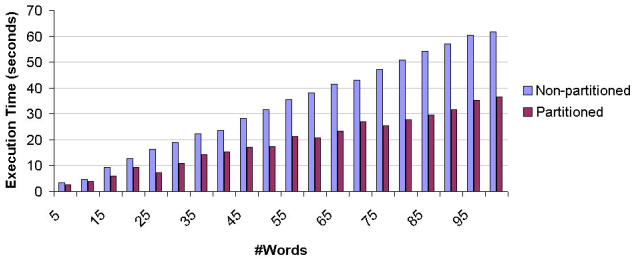

Fig. 17. Comparison of non-partitioned and partitioned TTS execution times

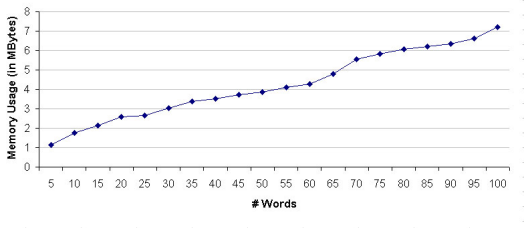

Fig. 18. Graph showing the rise in memory usage with increase in word cluster lengths

life of the second schedule can exceed that of the first by up to 6%. The large gaps in DSP activity, obtained after scheduling can be further exploited using DVS(Dynamic Voltage Scaling) and DPM(Dynamic Power Management) features provided by the processor to further improve battery life [15].

The tools used for experimentation primarily include the Code Composer Studio and Innovator Kit from Texas Instruments. A cycle accurate framework based on a modified Kinetic Battery Model [31] was developed for estimating the battery lifetime under various load conditions.The battery used for the purpose of simulation is assumed to have a nominal capacity 2000mAh at 3.7V.

Simulations showed an average reduction of 39.36% in the execution time over a non-partitioned implementation. The average execution times for different word cluster lengths have been shown in figure 17 for non-partitioned implementation on ARM and the code-partitioned implementation on the OMAP. The technique achieved a maximum speed-up of up to 1.86 and a battery life extension up to 11%. It may be observed from figures 17 and 18 that the system memory requirement increases almost linearly with an increase in the word cluster length. On the other hand the improvement in battery lifetime tends to saturate beyond a certain word cluster length. Obtaining the full theoretic capacity of the battery is nearly impossible. Hence, a balance must be struck between the battery life and the system memory usage while selecting the word cluster length. Phrases may be considered instead of words for an intonated speech synthesis system.

VII. Conclusions and Future Work

A battery aware code-partitioning and scheduling scheme has been proposed in this paper. The work indicates the importance of code-partitioning and scheduling in order to improve battery lifetimes as well achieve better system performance in embedded multiprocessor systems. It may be emphasized that, though scheduling alone results in significant improvement in battery life, code-partitioning provides scope for even greater improvement. Results prove the efficacy of the proposed scheme. The techniques discussed in the paper are highly relevant to design of battery powered mobile embedded devices and finds application in the increasing number of multi-core embedded processors. The mentioned code-partitioning and scheduling schemes use a heuristic search technique for finding a battery aware solution in the design space. A genetic algorithm with similar cost functions may be used to perform a more thorough search of the design space. Finally, it may be concluded that using DVS and DPM techniques in combination with the above mentioned scheme could yield further improvements to battery life.

References

[1] P. M. Alvarez, E. Levner, and D. Mosse, "Adaptive scheduling server for power-aware real-time tasks," *Trans. on Embedded Computing Sys.*, vol. 3, no. 2, pp. 284–306, 2004.

[2] L. Benini, G. Castelli, A. Macii, E. Macii, M. Poncino, and R. Scarsi, "A discrete-time battery model for high-level power estimation," in *DATE '00: Proceedings of the conference on Design, automation and test in Europe*. New York, NY, USA: ACM Press, 2000, pp. 35–41.

[3] I. Buchmann, "Batteries in a portable world," http://www.cadex.com. [Online]. Available: http://www.cadex.com

[4] T. D. Burd and R. W. Brodersen, "Energy efficient cmos microprocessor design," in *HICSS '95: Proceedings of the 28th Hawaii International Conference on System Sciences (HICSS'95)*. Washington, DC, USA: IEEE Computer Society, 1995, p. 288.

[5] C. F. Chiasserini and R. R. Rao, "Pulsed battery discharge in communication devices," in *MobiCom '99: Proceedings of the 5th annual ACM/IEEE international conference on Mobile computing and networking*. New York, NY, USA: ACM Press, 1999, pp. 88–95.

[6] A. Corp., http://www.arm.com. [Online]. Available: http://www.arm.com

[7] R. P. Dick and N. K. Jha, "Mocsyn: multiobjective core-based single-chip system synthesis," in *DATE '99: Proceedings of the conference on Design, automation and test in Europe*. New York, NY, USA: ACM Press, 1999, p. 55.

[8] R. Dick, D. Rhodes, and W. Wolfe, "Tgff: Task graphs for free," in *Workshop Hardware / Software Codesign*, 1998, pp. 97–101.

[9] M. Doyle, T. Fuller, and J. Newman, "Modeling of galvanostatic charge and discharge of lithium polymer insertion cell," *Journal of Electrochemical Society*, vol. 140, pp. 1526–1533, June 1993.

[10] Y. Fei and N. K. Jha, "Functional partitioning for low power distributed systems of systems-on-a-chip," in *ASP-DAC '02: Proceedings of the 2002 conference on Asia South Pacific design automation/VLSI Design*. Washington, DC, USA: IEEE Computer Society, 2002, p. 274.

[11] T. Fuller, M. Doyle, and J. Newman, "Relaxation phenomena in lithium-ion insertion cells," *Journal of Electrochemical Society*, vol. 141, pp. 982–990, April 1994.

[12] M. R. Garey and D. S. Johnson, *Computers and Intractability; A Guide to the Theory of NP-Completeness*. New York, NY, USA: W. H. Freeman & Co., 1990.

[13] S. Gold, "A pspice macromodel for lithium-ion batteries," in *Battery Conference on Applications and Advances*, 1997, pp. 35–41.

[14] W. Gu and C. Wang, "Thermal-electrochemical modeling of battery systems," *Journal of Electrochemical Society*, vol. 147, no. 8, pp. 2910–2922, 2000.

[15] T. Instuments, http://www.ti.com. [Online]. Available: http://www.ti.com

[16] H. Kiehne, *Battery Technology Handbook*. New York, NY, USA: Marcel Dekker Inc., 2003.

[17] A. Lahiri, S. Agarwal, B. B. Bhattacharya, and A. Basu, "Recovery-based real-time static scheduling for battery life optimization," in *VLSID '01: Proceedings of the The 19th International Conference on VLSI Design (VLSID '06)*. Washington, DC, USA: IEEE Computer Society, 2006.

[18] K. Lahiri, "On-chip communication: System-level architectures and design methodologies," Ph.D. dissertation, Department of Electrical and Computer Engineering, University of California, San Diego, 2003.

[19] K. Lahiri, S. Dey, D. Panigrahi, and A. Raghunathan, "Battery-driven system design: A new frontier in low power design," in *ASP-DAC '02: Proceedings of the 2002 conference on Asia South Pacific design automation/VLSI Design*. Washington, DC, USA: IEEE Computer Society, 2002, p. 261.

[20] D. Linden, *Handbook of Batteries and Fuel Cells*. New York, NY, USA: McGraw-Hill, 1984.

[21] J. Liu, P. H. Chou, N. Bagherzadeh, and F. Kurdahi, "Power-aware scheduling under timing constraints for mission-critical embedded systems," in *DAC '01: Proceedings of the 38th conference on Design automation*. New York, NY, USA: ACM Press, 2001, pp. 840–845.

[22] J. Luo and N. K. Jha, "Battery-aware static scheduling for distributed real-time embedded systems," in *Proc. of DAC*, 2001, pp. 444–449.

[23] ——, "Power-conscious joint scheduling of periodic task graphs and aperiodic tasks in distributed real-time embedded systems," in *ICCAD '00: Proceedings of the 2000 IEEE/ACM international conference on Computer-aided design*. Piscataway, NJ, USA: IEEE Press, 2000, pp. 357–364.

[24] J. F. Manwell, J. G. McGowan, U. Abdulwahid, and K. Wu, "Improvements to the hybrid2 battery model," in *American Wind Energy Association Windpower 2005 Conference*, 2005.

[25] S. Microelectronics, http://www.st.com. [Online]. Available: http://www.st.com

[26] A. Mukhopadhyay, S. Chakraborty, M. Choudhury, A. Lahiri, S. Dey, and A. Basu, "Shruti - an embedded text-to-speech system for indian language," *IEE Proceedings on Software Engineering*, 2005, to appear.

[27] D. Panigrahi, S. Dey, R. Rao, K. Lahiri, C. Chiasserini, and A. Raghunathan, "Battery life estimation of mobile embedded systems," in *VLSID '01: Proceedings of the The 14th International Conference on VLSI Design (VLSID '01)*. Washington, DC, USA: IEEE Computer Society, 2001, p. 57.

[28] M. Pedram and Q. Wu, "Design considerations for battery-powered electronics," in *DAC '99: Proceedings of the 36th ACM/IEEE conference on Design automation*. New York, NY, USA: ACM Press, 1999, pp. 861–866.

[29] Philips, http://www.philips.com. [Online]. Available: http://www.philips.com

[30] D. Rakhmatov and S. Vrudhula, "Energy management for battery-powered embedded systems," *Trans. on Embedded Computing Sys.*, vol. 2, no. 3, pp. 277–324, 2003.

[31] V. Rao, G. Singhal, A. Kumar, and N. Navet, "Battery model for embedded systems," in *VLSID '05: Proceedings of the 18th International Conference on VLSI Design held jointly with 4th International Conference on Embedded Systems Design (VLSID'05)*. Washington, DC, USA: IEEE Computer Society, 2005, pp. 105–110.

[32] Y. Shin and K. Choi, "Power conscious fixed priority scheduling for hard real-time systems," in *DAC '99: Proceedings of the 36th ACM/IEEE conference on Design automation*. New York, NY, USA: ACM Press, 1999, pp. 134–139.

[33] V. Tiwari, S. Malik, and A. Wolfe, "Power analysis of embedded software: a first step towards software power minimization," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 2, no. 4, pp. 437–445, 1994.

[34] Vahid, Le, and Hsu, "A comparison of functional and structural partitioning," in *ISSS '96: Proceedings of the 9th international symposium on System synthesis*. Washington, DC, USA: IEEE Computer Society, 1996, p. 121.

[35] Valgrind, http://valgrind.org/. [Online]. Available: http://valgrind.org/

[36] D. Zhu, D. Mosse, and R. Melhem, "Power-aware scheduling for and/or graphs in real-time systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 15, no. 9, pp. 849–864, 2004.