Scheduling of Frame-based Embedded Systems with Rechargeable Batteries

André Allavena
Computer Science Department
Cornell University
Ithaca, NY 14853
andre@cs.cornell.edu

Daniel Mossé
Department of Computer Science
University of Pittsburgh
Pittsburgh, PA 15260
mosse@cs.pitt.edu

April 13, 2001 09:06

1 Introduction

A large number of embedded real time systems operate on a cyclic basis, have a set of applications that must execute in a frame; after all applications have executed, the whole frame is repeated. This type of structure is present in communication infrastructures (where a template is often created for real-time scheduling [3, 2]) and in real-time imaging (where image magnification might include non-linear image interpolation, contrast enhancement, noise suppression and image extrapolation during each period [1]).

Due to the nature of these embedded systems, they often execute in platforms that are mobile or at least independent from any power source: many of these devices use batteries. Most of the research to date has concentrated on reducing the power consumption of such sets of applications running on devices with limited battery power. Mainly previous efforts have concentrated on predictive shutdown techniques [8] and varying speeds of processors [10, 4, 7, 5].

However, previous works on variable voltage scheduling have disregarded an important aspect of these devices, namely rechargeability of the batteries or have assumed that there will be a interval in which recharging is done which will suffice for generating enough energy for all applications being run in the embedded device. On the other hand, sometimes the recharging system (e.g., solar cells) have specific recharging characteristics that might provide more or less energy than the consuming applications.

The goal of this paper is to study scheduling in rechargeable systems with battery and deadline constraints. The rest of this paper is organized as follows. In Section 2 we describe the model. In Section 3 we present the algorithm when the speed of the CPU is fixed, and we relax this condition in Section 4. We present conclusions in Section 5.

2 Model

We consider scheduling a set of tasks on a single processor with variable voltage and frequency, that is, a variable speed CPU. We assume the tasks to be independent from each other and preemption to be free (no time or energy consumption).

2.1 Tasks

In our model, the N tasks τ_i execute within a frame and, therefore, they all have a common deadline D and they are all available at time $t_0 = 0$. Each task τ_i takes $t_i = C_i \cdot S_i$ time to run, where C_i is the worst-case number of cycles that are required by the task to execute, and S_i is the speed at which the task runs. We assume that each task τ_i consumes energy at an instantaneous constant rate p_i. In other words, the energy used by task i when it is run between t_i^1 and $t_i^2 = t_i^1 + t_i$ is $e_i = \int_{t_i^1}^{t_i^2} p_i \, dt = p_i \cdot t_i$. Because we allow preemptions, task i might be run in more than one interval. Assume the task is run on the set $\mathcal{T} = \{[t_i^1, t_i^2], \ldots, [t_i^{2k-1}, t_i^{2k}]\}$ with $t_i = \sum_{j=1}^{k} t_i^{2j} - t_i^{2j-1}$, we still have $e_i = \int_{\mathcal{T}} p_i \, dt = p_i \cdot t_i$.

2.2 Battery

Our system runs on a battery, whose energy level remains between two boundaries E_{min} and E_{max}. Let us define $\Delta E = E_{max} - E_{min}$. In our model, the recharging rate is a constant r. If the battery is fully charged, and we continue to charge it, we just waste energy. In contrast, if the battery is fully discharged ($E = E_{min}$), we just cannot run a task.

Due to the constant nature of the consumption rate p_i and the linear nature of the recharging, we can combine them. Let us define $p_i' = r - p_i$ to be the instantaneous replenishment rate of energy of the system while running task i. $p_i' > 0$ if task i consumes less than the recharging rate of the battery; $p_i' < 0$ if it consumes more.

Let us separate the tasks in two groups, namely the *recharging tasks* ($\mathcal{R} = \{\tau_i | p_i' \geq 0\}$) and the *dissipating tasks* ($\mathcal{D} = \{\tau_i | p_i' < 0\}$). In fact, we do not need to consider which group the tasks with $p_i' = 0$ belong to, since their aggregate energy behavior is null.

Let us define a weight measure on those groups: $|\mathcal{R}| = \sum_{i \in \mathcal{R}} p_i \cdot t_i$ and $|\mathcal{D}| = -\sum_{i \in \mathcal{D}} p_i \cdot t_i$. This is the amount of energy the system receives (or consumes) while running all the tasks from that group.

2.3 Problem definition

Within the framework above, our problem is to find a schedule which is able to execute all the tasks within the deadline D, starting with a battery **fully charged**, ending at the **same energy level** as we started. Furthermore, to be able to reduce the cycle time of the system, we would like the schedule span (schedule length) to be as small as possible.

The motivation behind the constraint of ending at the same energy level as we started is the frame-based system environments. One must finish at the exact same level of energy or higher, otherwise the system will eventually run out of energy.

3 Fixed Speed Processor

In this section we present the first algorithm. For ease of presentation, we first consider the case where we cannot slow down the processor: all tasks are run at full speed. In the next section we will address the improvement that can be obtain with a variable speed processor.

The intuition behind this simple algorithm is to run tasks such that the battery goes to the minimum level and then recharges as much as possible, bouncing between E_{min} and E_{max}. We preempt the tasks when the energy level of the battery reaches either of the two boundary conditions. Furthermore, if the sum of power consumptions causes the energy level to finish below its original level, we insert idle time that causes the system to recharge with rate r.

Algorithm

1. If $|\mathcal{D}| > |\mathcal{R}|$, add an idle task *idle* in $\mathcal{R}$, of length $t_{idle} = \frac{|\mathcal{D}|-|\mathcal{R}|}{r}$ and of consumption $p_{idle} = 0$ (i.e., $p'_{idle} = r$). This step will make the energy consumption no larger that the energy absorption.
   [If this is not necessary, define $t_{idle} = 0$.]

2. If $\sum_{i=1}^{N} t_i + t_{idle} > D$, declare FAILURE: cannot schedule tasks within deadline and energy constraints.

3. Schedule tasks from $\mathcal{D}$ until there are no more tasks in $\mathcal{D}$ or the battery is fully discharged ($E = E_{min}$). It may be necessary to preempt task τ_i, which was the one executing when we reached $E = E_{min}$.

4. Schedule tasks from $\mathcal{R}$ until the battery is fully charged. Analogously, it may be necessary to preempt task j, which was the one executing when we reached $E = E_{max}$.

5. Iterate steps 2 and 3, scheduling first the preempted task i, preempting again if we run out of battery, or adding other tasks from $\mathcal{D}$. Then schedule task j, preempt it if needed or schedule other tasks from $\mathcal{R}$. And so on until we run out of tasks in $\mathcal{D}$.

6. At this point we have scheduled all the tasks from $\mathcal{D}$. If there are tasks left in $\mathcal{R}$, just schedule all of them. We may be wasting recharging energy (if $E = E_{max}$ at any point from now on), but we do not care since all the tasks from $\mathcal{D}$ have been scheduled.

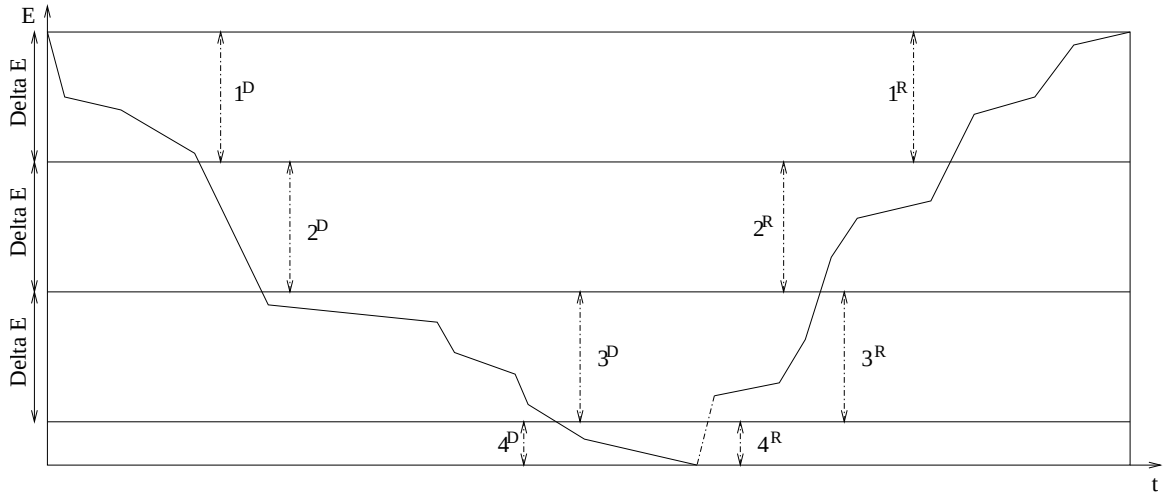


Figure 1: Graphical representation of the algorithm

In Figure 1, each segment of line represents a task. We show all the tasks from $\mathcal{D}$ first (decreasing slope), then those from $\mathcal{R}$ (increasing slope). Note that at the bottom, the first recharging task, the *idle task*, is depicted by a dashed line. The horizontal lines are the *preemption lines*; the interval between two adjacent preemption lines represents the amount of energy that can be consumed or should be recharged (i.e., $\Delta E = E_{max} - E_{min}$.

According to the algorithm above, we should preempt a task at the point where it crosses the line. The tasks between two consecutive of those lines are the tasks being considered and scheduled by one iteration of the algorithm, namely 1 and $1'$, 2 and $2'$, etc, the plain number being tasks from

$\mathcal{D}$, the primed ones from $\mathcal{R}$. The algorithm actually schedules the groups of tasks in that order. Note the added idle time is not concentrated all at the end of the schedule; in this example, it is at the end of the recharging task sets $3'$ and $4'$.

3.1 Discussion

1. We never run out of battery (that is, we never dispatch tasks when there is no energy); this is obvious from the algorithm that does not allow tasks to run after E_{min}.

2. We insert as little idle time as needed; this is also clear from the algorithm, since we only insert the minimum time needed to recharge and finish ($t_{idle} = \max(\frac{|\mathcal{D}|-|\mathcal{R}|}{r}, 0)$). This also leads to the following observation:

3. We end up as early as possible, and the length of the schedule is $T = \sum_{i=1}^{N} t_i + t_{idle}$.

4. We only waste recharging power when there are no dissipating tasks left to use it.

5. In reasoning about the algorithm, one might consider 2 different deadlines, namely one for the execution of the tasks, D_{tasks}, and the second for recharging (i.e., for bringing the energy to the same level as the start), D_{recharge}. Clearly, there are also two corresponding relevant spans of the schedule $T_{\text{recharge}} = \sum_{i=1}^{N} t_i + t_{idle}$ and $T_{\text{tasks}} = \sum_{i=1}^{N} t_i + \max(\frac{|\mathcal{D}|-|\mathcal{R}|-\Delta E}{r}, 0)$. One just needs schedule enough idle time to do the last pass without running out of energy, in order to achieve those optimal values. We fail if either $D_{\text{tasks}} < T_{\text{tasks}}$ or $D_{\text{recharge}} < T_{\text{recharge}}$.

6. One can dynamically narrow the two boundaries of the battery by simply adjusting the E_{min} and E_{max} constants when the battery begins to age.

7. The algorithm is very efficient and easy to implement. The complexity of the algorithm is $O(n)$ if $\forall i, e_i = p_i \cdot t_i \leq k \cdot \Delta E$, for some constant k.

3.2 Minimizing the Number of Preemptions

We do not address the problem of minimizing the number of preemptions within the energy and deadline constraints, since the following proposition shows that it is a complex problem.

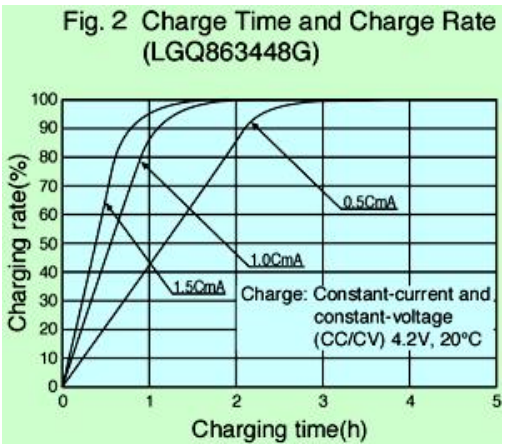
Proposition 1 *Finding a schedule which is able to schedule all the tasks, stay within energy boundaries and minimize the number of preemption is $\mathcal{NP}$-complete.*

Proof: This problem reduces to Subset Sum: given positive numbers $(w_1, w_2, \ldots, w_n)$, and a target W, find a subset $\mathcal{P}$ of $[1..N]$ such that $\sum_{i\in\mathcal{P}} w_i = W$. Let us define $S = \sum_{i=1}^{N} w_i$. It is strictly equivalent to consider the target sum of W or $S - W$. So, we assume $W \geq S/2$. Let us define the following $N + 2$ tasks: for $i = 1, 2, \ldots, N$, $t_i = w_i$ and $p_i = 1$, task $N + 1$ has $t_{N+1} = W$ and $p_{N+1} = -1$ and last task $N + 2$ has $t_{N+2} = S - W$ and $p_{N+2} = -1$. Let us choose $\Delta E = W$. A schedule with no preemption will schedule task $N + 1$, the tasks from $\mathcal{P}$, task $N + 2$ and then the tasks from $[1..N] - \mathcal{P}$ (or the opposite: task $N + 2$ and the tasks from $[1..N] - \mathcal{P}$, then task $N + 1$ and the tasks from $\mathcal{P}$). Anyhow, it will solve Subset Sum.

3.3 Addressing the limitations of the model

3.3.1 If the battery has a non constant recharging rate:

Although we have considered a constant recharging function, as shown in Figure 2 [9], most batteries have a constant recharging rate r on the middle part (0 to 80 %, and a recharging rate which continuously decreases from r to 0 (usually exponentially) above that. In this case, we run the algorithm with $E_{min} = 0$ and $E_{max} = 0.8$ (that is on their linear recharging slope only). If we run recharging tasks while above E_{max}, and there are still tasks in set $\mathcal{D}$, it means we are spending more time in order to recharge the same amount of energy, or to recharge less for the same amount of time. Anyhow you are wasting energy from the battery's point of view since, considering the length of the schedule is fixed, and assuming you do not have extra recharging tasks, staying always in the linear part means you would have received more energy. And if you have extra recharging tasks, better run them at the end anyway to return a "above the expected maximum" recharged battery.



3.3.2 Variable Power Consumption Functions within a Single Task

If the p_i functions are not constant, but they do not vary much during the execution of the tasks, one can narrow the energy boundaries to set safe limits. There are many ways of doing this, depending on what one knows about the task.

For example, let $\bar{p_i} = \frac{1}{t_i} \int_{\mathcal{T}} p_i(t)\, dt$, that is $\bar{p_i}$ is the average power consumption. If $|p_i(t) - \bar{p_i}| < \alpha$ for all t and for all tasks i, then choosing $E_{max} = E_{max} - \alpha \cdot \max(t_i)/2$ and $E_{min} = E_{min} + \alpha \cdot \max(t_i)/2$ are safe limits. Doing this for each task and not for all simultaneously will provide smaller safety margins.

4 Variable-Speed CPU

It has been shown elsewhere that using variable voltage and frequency CPUs allows for quadratic energy savings [10, 4]. In this paper, we are not concerned with the energy savings per se, but our goal is to meet the deadline and energy constraints imposed by the system (deadline imposed by the user and energy constraints imposed by the battery and recharging subsystem).

Therefore, our goal is to save as much energy as needed in order to save time. Recall that in our frame-based system, we want to finish as early as possible. Clearly, if $|\mathcal{D}| = |\mathcal{R}|$, there is no need to insert idle time and the schedule span is minimal. The question is, thus, how to take advantage of the variable voltage CPU to reduce the span when $|\mathcal{D}| \neq |\mathcal{R}|$.

First, we note that when $|\mathcal{D}| < |\mathcal{R}|$, we just run tasks at full speed and waste recharging energy.

Thus, the only case we can improve is when $|\mathcal{D}| > |\mathcal{R}|$: we are "wasting" time while running our idle task in order to recharge. Perhaps, if we did not consume so much energy, we would not need to wait and recharge. We show below that if we can slow down the processor, we shall consume

less, and we shall see that even if some tasks take longer to run, we are able to reduce the span of the schedule.

To facilitate the presentation, let us consider the tasks in a system without boundaries on the battery energy and with their original power consumption modes (functions p_i, not p'_i). In this case, it is as if we schedule all the tasks at full speed in any order, and then make the CPU wait for $t = t_{idle}$ for the system to recover the "lost" energy. This waiting period is analogous to running the processor at speed $S = 0$.

When a task τ_i is scheduled at full speed, it will execute for t_i and consume $e_i(t_i)$. The system recovers energy at a rate r; in any interval of length t, it recovers $t \cdot r$. However, due to the quadratic saving of energy with linear increase in delay [10, 4], slowing down a task instead of waiting (i.e., spending Δt more on running a task instead of waiting for Δt) will always save some energy, provided the energy function is positive, decreasing, differentiable, and convex. We show this graphically, with the help of Figure 3.

In this figure, each curve represents the energy consumed by a task, either alone (i.e., p functions) or agregated with the recharging (i.e., p' functions, represented by the unlabeled lines in the graph). A slowed down task will always end up having cousumed less than the same task run at full speed followed by a waiting time. Therefore, the agreagate of the slowed down task and the recharging will always be higher in energy than the full speed one, since we are adding the same value (the recharging) to both.

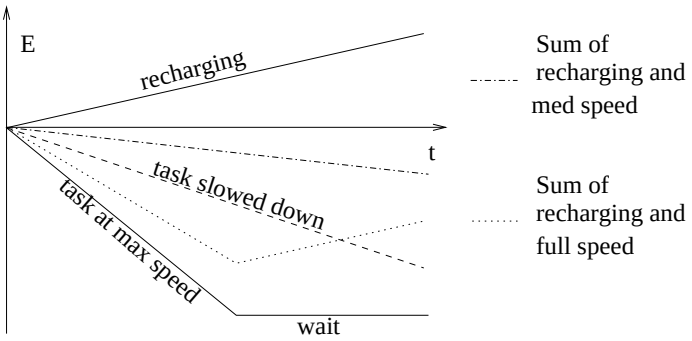


Figure 3: Why slowing down a task may yield shorter schedules.

4.1 Decreasing the Schedule Span

¿From Section 3, we know that the schedule span is $T = \sum_{i=1}^{N} t_i + t_{idle}$, where $t_{idle} = \frac{|\mathcal{D}|-|\mathcal{R}|}{r}$. During T we gain $E_{in}(T) = T \cdot r$ by recharging, and we spend $E(T) = \sum_{i=1}^{N} e_i(t_i)$. Let $E_0(T)$ be the minimum achievable energy spent within interval T when considering variable speed CPU. What we are really interested in is minimizing T such that $E_{in}(T) \geq E(T)$.

Proposition 2 *T is minimal if and only if $E_{in}(T) = E_0(T)$.*

Proof ($\Rightarrow$)
For this part of the proof, we consider two cases (remember the idle task is included in the tasks we run):

$E_{in}(T) < E_0(T)$ at the end of time T. In this case, we have not reached our goal to end up at the same level of energy as we started. Therefore, we would need to increase the amount of idle time and consequently increase the span of the schedule.

$E_{in}(T) > E_0(T)$ at the end of time T. Since we assume that at the beginning of the schedule $|\mathcal{R}| < |D|$ and we ended up with $E_{in}(T) > E_0(T)$, it must be true that we slowed down some task(s) to fit T. Now we can speed up at least one task and finish earlier (i.e., decreasing T). Clearly, this will cause spending more energy, but no more than $E_{in}(T) - E_0(T)$.

Proof ($\Leftarrow$)
Here we prove that $E_{in}(T) = E_0(T) \Rightarrow T$ is minimized. Since $E_0(T)$ is a decreasing function of T and $E_{in}(T)$ is a strictly increasing function of T, they intersect in a unique point. This point is T_{min}.

4.2 How to achieve T_{min} such that $E_{in} = E_0$

The total amount of energy $E(T) = \sum_i e_i$ used by the tasks is minimum when (using Lagrange multipliers [6]) $\frac{dE}{dT}(T) = 0$, which means that for a constant λ, $\forall i, j : \frac{de_i}{dt_i}(t_i) = \frac{de_j}{dt_j}(t_j) = \lambda$. In other words, if we could choose the task speeds[1], we would choose $\lambda = -r$, so that the rate of consumption of each task would be the same as the rate of recharging, r.

Let us consider a processor running at maximum speed, and in particular the recharging tasks (those in $\mathcal{R}$, that is, with a $p_i'(t_i) > 0$ or $p_i(t_i) < r$). From the above, we know that the optimal theoretical solution is such that all tasks consume energy at rate r. Since the recharging tasks are already running at full speed, they cannot be sped up to comply with rate r, and therefore there will be a surplus of energy.

According to the optimal solution, the dissipating tasks (those in $\mathcal{D}$, that is, with a $p_i'(t_i) < 0$ or $p_i(t_i) > r$) should be slowed down, to match the consumption rate r. However, we can use the surplus from above to run these tasks so that they consume energy at a rate higher than the optimal r. In other words, we are able to run dissipating tasks faster.

There are two ways to achieve the minimum T, namely, (a) bring all the dissipating tasks to comply with rate r and then raise their energy consumption accordingly to consume the surplus, or (b) slow down the highest energy consumers until all the surplus has been absorbed. Solution (a) complies with the intuition presented above, but represents an extra step in comparison to solution (b), which we describe below.

Algorithm

Each task i has a running time associated with it, t_i, which is initialized to the time it takes to run task τ_i at maximum speed.

1. Consider the task(s) with the highest $\frac{de_i}{dt_i}(t_i)$ and slow it (them) down so that $\frac{de_i}{dt_i}(t_i) = \max_{\text{other tasks}} \frac{de_j}{dt_j}(t_j)$. Update t_i values to be the time it takes to run these tasks at their lower speeds.
2. Update $T = \sum_{i=1}^{N} t_i$, $E_0(T)$ and $E_{in}(T)$.
3. If $E_0 = E_{in}$ stop, if $E_0 > E_{in}$ iterate, go back to 1.
4. If $E_0 < E_{in}$ we have gone too far. Take all the tasks that were slowed down in the last iteration (all other tasks cannot be sped up), and run them faster so that $E_0 = E_{in}$. Note that all the $\frac{de_i}{dt_i}(t_i)$ of these tasks are the same.

Scheduling Now that we know the different speed settings, we can run the algorithm from Section 3 to schedule the tasks (running them at the speed computed above) and stay within the energy boundaries. We have $|\mathcal{R}| = |\mathcal{D}|$ (no need for an idle task of course), and the span of the schedule is as small as possible.

[1]Each task will run at a constant speed throughout its execution.

5 Conclusion

Several current and future embedded devices (and their applications) require real-time properties while carrying out power management together. Since most previous work has not concentrated on rechargable systems, we have proposed a simple and efficient way of scheduling tasks in a frame-based system with maximum and minimum energy constraints. We started by describing an algorithm for fixed-speed CPUs, and (although counter intuitive) showed that a variable speed processor allows to **reduce the length** of the frame by **slowing down** tasks.

Our work can be extended in the case of a fixed number of identical frames, or in the case of different set of frames (do not necessarily need to finish absolutely fully charged).

References

[1] L. D. Nguyen and A. M. K. Cheng. An Imprecise Real-Time Image Magnification Algorithm. In *International Conference on Multimedia Systems*, 1996.

[2] L. Dong, D. Mosse, and R. Melhem. Scheduling Algorithms for Dynamic Message Streams with Distance Constraints in TDMA protocol. *Proceedings of EuroMicro Conference on Real-Time Systems*, June 2000.

[3] C. C. Han, K. J. Lin, and C. J. Hou. Distance-Constrained Scheduling and Its Applications to Real-Time Systems. *IEEE Trans. on Computers*, 45(7):814–826, 1996.

[4] I. Hong, G. Qu, M. Potkonjak and M. Srivastava. Synthesis Techniques for Low-Power Hard Real-Time Systems on Variable Voltage Processors. In *Proceedings of 19th IEEE Real-Time Systems Symposium (RTSS'98)*, Madrid, December 1998.

[5] C. M. Krishna and Y. H. Lee. Voltage Clock Scaling Adaptive Scheduling Techniques for Low Power in Hard Real-Time Systems. In *Proceedings of the 6th IEEE Real-Time Technology and Applications Symposium (RTAS'00)*, Washington D.C., May 2000.

[6] D. Luenberger, *Linear and Nonlinear Programming,* Addison-Wesley, Reading Massachusetts, 1984.

[7] Y. Shin and K. Choi. Power Conscious Fixed Priority Scheduling for Hard Real-Time Systems. In *Proceedings of the 36th Design Automation Conference, DAC'99*, pp. 134-139.

[8] M. Srivastava, A. P. Chandrakasan and R. W. Brodersen. Predictive System Shutdown and other Architectural Techniques for Energy Efficient Programmable Computation. *IEEE Transactions on VLSI Systems*, 4(1): 42-55, 1996.

[9] Toshiba Battery Company, Lithium Ion Rechargeable Batteries specifications, `http://www.tbcl.co.jp/TB_e/liion.htm`

[10] F. Yao, A. Demers and S. Shankar. A Scheduling Model for Reduced CPU Energy. *IEEE Annual Foundations of Computer Science*, pp. 374 - 382, 1995.