Energy Efficient Real-Time Scheduling

Amit Sinha and Anantha P. Chandrakasan
Department of EECS
Massachusetts Institute of Technology
Cambridge, MA 02139

e-mail: sinha@mit.edu, anantha@mtl.mit.edu

***Abstract* - Real-time scheduling on processors that support dynamic voltage and frequency scaling is analyzed. The Slacked Earliest Deadling First (SEDF) algorithm is proposed and it is shown that the algorithm is optimal in minimizing processor energy consumption and maximum lateness. An upper bound on the processor energy savings is also derived. Real-time scheduling of periodic tasks is also analyzed and optimal voltage and frequency allocation for a given task set is determined that guarantees schedulability and minimizes energy consumption**.

1 Introduction

Real-time scheduling can be broadly classified into *static* and *dynamic* algorithms. Static algorithms are applicable to task sets where complete information (e.g arrival times, computation time, deadlines, precedence, dependencies etc. is available *a priori*). The Rate Monotonic (RM) algorithm one such algorithm and is optimal among all fixed priority assignments in the sense that no other fixed-priority algorithm can schedule a task set that cannot be scheduled by RM [1]. Dynamic scheduling is characterized by inherent uncertainity and lack of knowledge about the task set and its timing constraints. The Earliest Deadline First (EDF) algorithm has been shown to be an optimal dynamic scheduling algorithm [2]. However, EDF assumes resource sufficiency (i.e. even though tasks arrive unpredictably, the system resources have a sufficient *a priori* guarantee such that at any given time all tasks are schedulable) and in the absence of such a guarantee the EDF performace degrades rapidly in the presence of overload. The Spring algorithm has been proposed for such dynamic resource insufficient environments and uses techniques such as admission control and planning-based algorithms [3].

A major trend in the microprocessor industry is towards energy efficient mobile computing for maximal battery life using the concept of performance on demand. The basic idea being to run the CPU at a voltage and frequency that satisfies the current performance requirement. Dynamic voltage and frequency scaling is a very effective technique for reducing CPU energy [4][5]. Most microprocessor systems are characterized by a time varying computational load. Simply reducing the operating frequency during periods of reduced activity results in linear decrease in power consumption but does not affect the total energy consumed per task. Reducing the operating voltage implies greater critical path delays which in turn means that the peak performance is compromised. Significant energy benefits can be achieved by recognizing that peak performance is not always required and therefore the operating voltage and frequency of the processor can be dynamically adapted based on instantaneous processing requirement. Examples include the Intel Pentium III SpeedStep technology [6] which lets the user run the processor at a lower voltage and frequency when using the battery, LongRun technology from Transmeta's Crusoe [7] and PowerNOW! from AMD K6-2+ [8].

In this paper, we analyze energy efficient real-time scheduling algorithms that can exploit the variable voltage and frequency hooks available on processors for improving energy efficiency an therefore battery life of embedded systems. There has been a significant amount of work in exploiting dynamically variable voltage hardware. Efficient DC-DC conversion techniques have been developed to provide a variable power supply [9]. Scheduling strategies for non-real time systems have been proposed [10]. In [11] workload prediction schemes and performance hit metrics are analyzed. In [12] optimal *offline* scheduling techniques for variable voltage/frequency processors is analyzed for independent tasks with arbitrary arrivals. Several techniques for dynamic power management using event driven shutdown have also been proposed. The authors of [13] have proposed a set of heuristic algorithms to schedule a mixed workload of periodic and sporadic tasks.

In this paper we propose Slacked Earliest Deadline First (SEDF) algorithm and prove that it is optimal in minimizing processor energy consumption *and* maximum lateness for an independent arbitrary task set. A lower bound on energy savings through dynamic voltage and frequency scaling is also derived for all possible algorithms and arrival statistics. The SEDF algorithm is dynamic and approaches the EDF algorithm as processor utilization increases. We use the EDF algorithm as a baseline to compare the scheduling performance of SEDF. Optimal processor voltage and frequency assignments for periodic tasks is also discussed with the EDF and RM algorithms used as a baseline.

2 Variable Voltage Processing

Using simple first order CMOS delay models it has been shown in [15] that the energy consumption per sample is given by

$$E(r) = CV_0^2 T_s f_{ref} r\left[\frac{V_t}{V_0} + \frac{r}{2} + \sqrt{r\frac{V_t}{V_0} + \left(\frac{r}{2}\right)^2}\right]^2 \qquad (1)$$

where C is the average switched capacitance per cycle, T_s is the sample period, f_{ref} is the operating frequency at V_{ref}, r is the normalized processing rate i.e. $r = f / f_{ref}$ and $V_0 = (V_{ref} - V_t)^2/V_{ref}$ with V_t being the threshold voltage. The normalized workload in a system is equivalent to the processor utilization. The operating system scheduler allocates a time-slice and resources to various processes based on their priorities and state. Often no process is ready to run and the processor simply idles. The normalized workload, w, over an interval is simply the ratio of the non-idle cycles to the total cycles, i.e. w = (*total_cycles* - *idle_cycles*) / *total_cycles*. The workload is always in reference to the fixed maximum supply and maximum processing rate. In an ideal DVS system the processing rate is matched to the workload so that there are no idle cycles and utilization is maximum. Figure 1 shows the plot of normalized energy versus workload as described by Equation 1, for an ideal DVS system. The important conclusions from the graph are, (i) Averaging the workload and processing at the mean workload is more energy efficient because of the convexity of the $E(r)$ graph and Jensen's inequality: $\overline{E(r)} \geq E(\bar{r})$. (ii) A small number of discrete processing rate levels (i.e supply voltage, V_{dd}, and operating frequency, f) can give energy savings very close to the savings obtained from arbitrary precision DVS. Also shown in the graph is a simple quadratic $E(r)$ model (i.e. $E(r) = r^2$). As can be seen a simple quadratic model can be used to approximate Equation 1 quite accurately. Although we have used the exact form in all our simulations we will use the quadratic form wherever possible for analytical tractability.

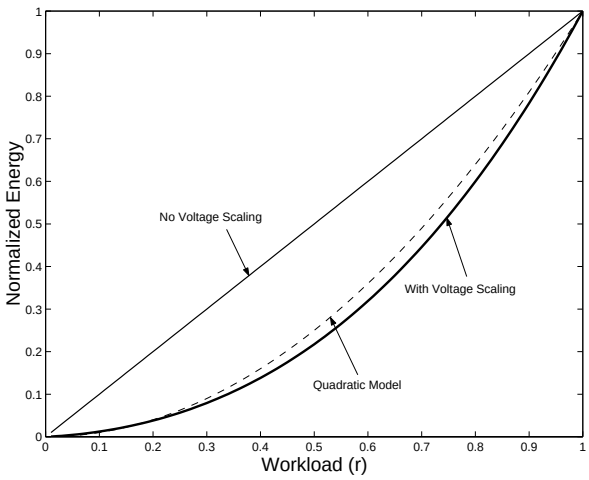


Fig. 1. Energy consumption as a function of workload

3 Aperiodic Task Scheduling

3.1 The Earliest Deadline First Algorithm

Let the task set to be scheduled be denoted by

$$\vartheta = \{\tau_i(a_i, c_i, d_i), 0 < i \leq N\} \quad (2)$$

where a_i, c_i and d_i represent the arbitrary arrival time, computation time and deadline of the i^{th} task from a set of N tasks. We assume that the tasks are independent i.e. they do not have any dependence constraints, the system consists on one processor and preemption is allowed. Under these conditions, the following theorem holds.

Theorem 1: [2] *Given a set on N independent tasks with arbitrary arrival times, any algorithm that at any instant executes the task with the earliest absolute deadline among all the ready tasks is optimal with respect to minimizing the maximum lateness.*

This theorem, known as the *Earliest Deadline First* (EDF) algorithm, was first posed by Horn [2] and was proved to be optimal by Dertouzos [14]. Maximum lateness is defined as

$$L_{max} = \max_i(f_i - d_i) \quad (3)$$

where f_i is the finish time of the i^{th} task. L_{max} is a good criteria for hard-real time algorithms as it upper-bounds the time by which any task misses its deadline. We will use L_{max} as a metric to evaluate our scheduling algorithm.

3.2 Real-Time Systems with Variable Processing Rate

With variable voltage/frequency systems two things need to be determined at every scheduling interval - (i) The task to be scheduled, and (ii) The relative processing rate. A simple greedy algorithm that sets the processing rate such that the scheduled task just meets its deadline will not work. This can be illustrated by a simple example shown in Fig. 2.

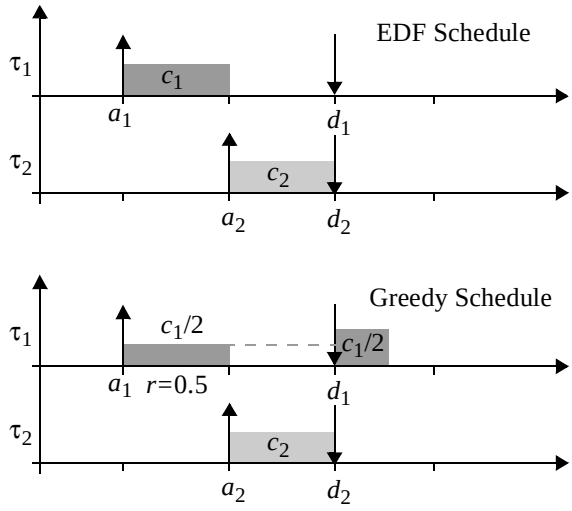


Fig. 2. Greedy scheduling of processing rate

The two tasks have the same deadlines and one comes in after the first one. EDF is able to schedule the two tasks such that both of them meet their respective deadlines. A greeedy algorithm that sets the processing rate, r, based on information about the current task's deadline is not able to schedule the tasks. At time $t = a_1$, the greedy scheduler sees only task τ_1 with deadline d_1 and sets the processing rate to $r = c_1/(d_1-a_1)$, such that the task occupies the complete time available. At time $t = a_2 < d_1$, τ_2 arrives with the same deadline $d_1 = d_2$, and even though the rate is set back to $r =$

1 and τ_2 meets its deadline, τ_1 fails to complete before its deadline d_1. Therefore any algorithm that modifies processing rate must do so in an intelligent way such that it approaches the EDF algorithm is L_{max} optimality.

3.3 The Slacked Earliest Deadline First (SEDF) Algorithm

In this section we propose the SEDF algorithm and show that it is optimal is minimizing processor energy and maximum lateness. In fact, the SEDF, algorithm approaches the EDF algorithm in an asymptotic way as the processor is fully utilized.

Theorem 2: *Given a set of independent tasks with arbitrary arrival times, computation times and deadlines, any algorithm that at every scheduling instant t_i executes the task with the earliest absolute deadline among all the ready tasks and sets the instantaneous processing rate to $r_i(S_i, U_i)$, where U_i is the processor utilization upto time t_i and S_i is the available slack for the scheduled task, is optimal with respect to minimizing the maximum lateness and processor energy. This optimum processing rate is approximated by*

$$r_i(S_i, U_i) = \begin{cases} S_i + (1 - S_i)U_i, & 0 < S_i \le 1 \\ 1, \text{ otherwise} \end{cases} \quad (4)$$

Proof: Let the scheduling intervals be Δt such that a decision as to which task will be allocated the processor during the interval (t_i, t_{i+1}) is made at discrete time instant $t_i = i\Delta t$. The problem we want to solve is : Which task should be allocated the processor during the interval (t_i, t_{i+1}) and what should be the relative processing rate, $0 \le r_i \le 1$? We assume that the scheduling decision takes negligible time compared to the scheduling interval Δt and that the computation times are integral multiples of the scheduling interval.

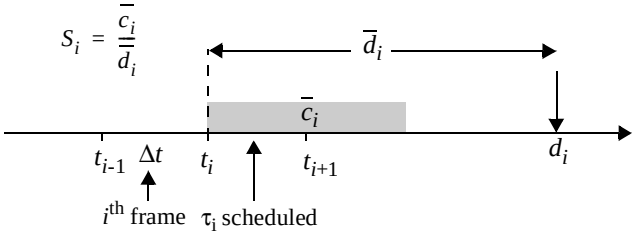


Fig. 3. Illustrating the parameters involved in SEDF

Let τ_i be the task with the earliest deadline at time instant t_i and let $\bar{c}_i$ denote the residual computation time ($\bar{c}_i = c_i$ if the task τ_i was never scheduled before). The computation time is always with reference to the maximum processing rate i.e. $r = 1$. Let U_i denote the processor utilization upto time t_i. U_i is simply the ratio of the number of idle frames (where no tasks were ready for scheduling) to the total number of scheduling frames, *i*. Let $\bar{d}_i = d_i - t_i$, be the maximum number scheduling slots available to task τ_i to complete before its deadline. If $\bar{d}_i < 0$, the task has missed its deadline and there is no positive slack available. The available slack for task τ_i is the ratio of $\bar{c}_i$ to $\bar{d}_i$ and is denoted by S_i. All these variables have been shown in Fig. 3. If $S_i > 1$, the task will miss its deadline no matter what. If $S_i < 0$, the task has already missed its deadline. Under both these circumstances, minimizing maximum lateness requires that the task be finished as soon as possible and so the processing rate r_i is set to 1. Note that $S_i = 0$ is not possible since that would mean that the task has 0 residual computation time i.e. it is already completed.

For the case where $0 < S_i \le 1$, the analysis is as follows. Assuming that the processor utilization is stationary over the next $\bar{d}_i$ slots, the probability that the task will finish before its deadline at the maximum processing rate is given by

$$\text{Prob}[\tau_i \text{ finishes}] = P(r = 1) = \sum_{k = \bar{c}_i}^{\bar{d}_i} {}^{\bar{d}_i}C_k (1 - U_i)^k U_i^{\bar{d}_i - k} \quad (5)$$

which follows from the fact at the maximum processing rate there are $\bar{c}_i$ slots required to complete the task out of a maximum of $\bar{d}_i$ slots and the probability of any particular slot being occupied is U_i. The probability of completing at any processing rate r is therefore given by

$$P(r) = \sum_{k = \left\lceil \frac{\bar{c}_i}{r} \right\rceil}^{\bar{d}_i} {}^{\bar{d}_i}C_k (1 - U_i)^k U_i^{\bar{d}_i - k} \quad (6)$$

where the number of required slots simply scales with the reduced processing rate. The energy savings at any processing rate *r*, for a given task is given by

$$E_{save}(r) = 1 - r^2 \quad (7)$$

based on our discussion in the previous section. Let us define

$$\xi(r) = P(r) \cdot E_{save}(r) \quad (8)$$

where $\xi(r)$ can be interpretted as weighted energy savings where $E[\xi(r)]$ is the expected energy savings given the task completed before its deadline. To maximize $\xi(r)$ we set the partial derivative, with respect to *r* equal to zero,

$$\frac{\partial}{\partial r}\xi(r) = 0 \quad \Rightarrow \quad \frac{2r}{1 - r^2} = \frac{P'(r)}{P(r)} \quad (9)$$

The optimum *r* cannot be obtained analytically since $P(r)$ is not differentiable in the entire range $0 \le r \le 1$. Fig. 4 shows the completion probability, the weighted energy savings as a function of the processing rate, *r*, and the optimum processing rate as a function of the processor utilization (for a slack $S_i = 0.1$). The graphs are somewhat intuitive. As *r* increases, the computation slots required decreases and the probability $P(r)$ of completion *increases*. The increase is faster with lower processor utilization. The energy savings on the other hand *decreases* with increased processing rate. The weighted energy savings therefore has an optimum processing rate where it is maximized. Fig. 5 shows the optimized processing rate, *r*, as a function of the processor utilization, *U*, and the available slack, *S*. This is an exact numerical solution for Equation 9. Also shown in Fig. 5 is the optimum

processing rate as a linear function of U and S as represented by Equation 4.

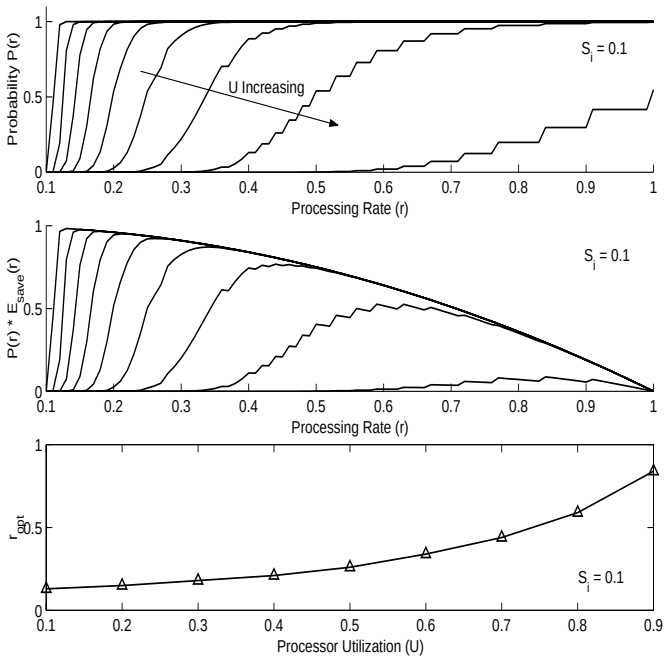


Fig. 4. Completion probability, weighted energy savings and optimum processing rate

A closed form expression for optimum r can be obtained if we let $\Delta t \to 0$ and in the limit, the function P(r) actually becomes continuous. Using Stirling's approximation,

$$n! \approx \sqrt{2\pi n} n^n e^{-n} \tag{10}$$

in Equation 6, the limit of the sum actually becomes an integral and for U around 0.5, the probability of completion is given by the gaussian integral (the error function)

$$P(r) = \frac{1}{\sqrt{2\pi}} \int_a^1 e^{-\frac{x^2}{2}} dx \qquad a = \frac{\frac{S_i}{r} - (1-U)\overline{d}_i}{\sqrt{\overline{d}_i (U(1-U))}} \tag{11}$$

for values of U close to 0, the function tends to a poisson integral. Although these equations can be solved exactly, the simple linear function shown in Equation 4 is quite adequate as we have shown in Fig. 5 and will show in our results.

3.4 Results

Fig. 6 shows an example of EDF and SEDF scheduling on a set of 10 tasks characterized by a uniform random process. While EDF meets all deadlines (the preemptive nature is obvious from tasks 3 and 7) SEDF is not able to meet all deadlines, the L_{max} being equal to 3 time units. The energy savings is 53%. The changing height of the computation time bars indicates reduced processing rate which is shown along with the evolving processor utilization at the bottom of the graph in Fig. 6(b). Since SEDF is stochastically optimal, the maximal lateness and energy savings improve as the learning time and task set increases.

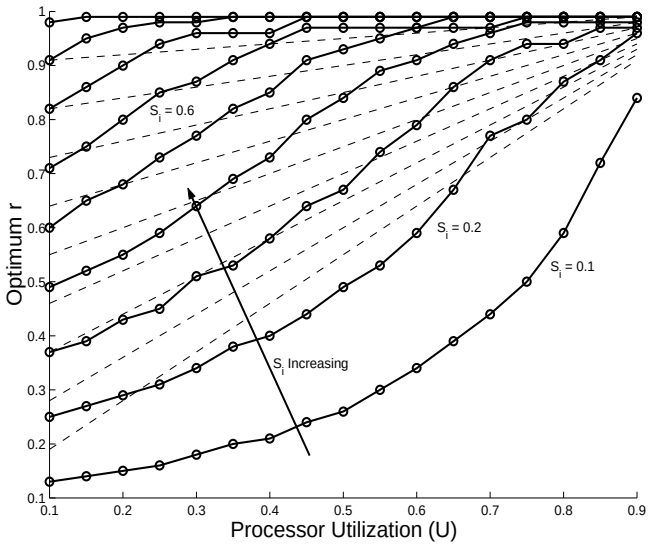
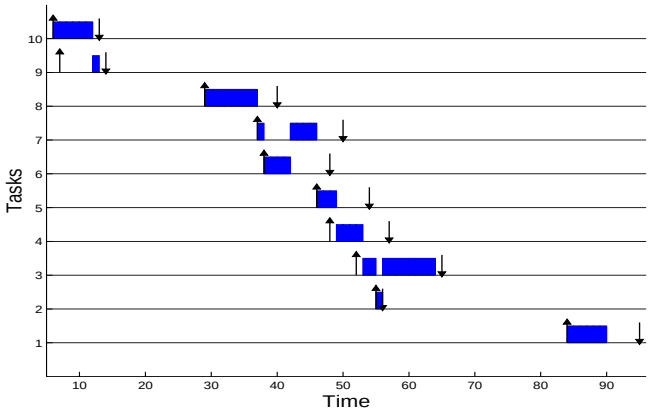
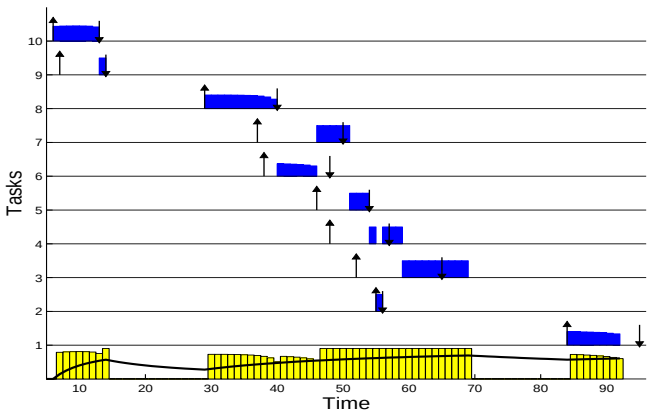


Fig. 5. Optimum processing rate as a function of processor utilization and slack



(a) EDF Scheduling



(b) SEDF Scheduling

Fig. 6. Example of EDF and SEDF scheduling

We have compared the SEDF algorithm to the EDF algorithm based on random task sets where the arrival times, computation times and deadlines are characterized by uniform, gaussian and

poisson processes. In each case, the maximum lateness and the energy consumption were compared. The energy savings averaged over 3×10^6 experiments was about 60% while the degradation in maximal lateness was less that 10%. The results of all the experiments have been summarized in Fig. 7 where each bar represents the average of 10^5 experiments. The increased energy savings from gaussian characterized task sets can be attributed to the fact that arrivals and computations are more clustered (i.e. within $\pm2\sigma$ mostly) and so the predicted slack is better.

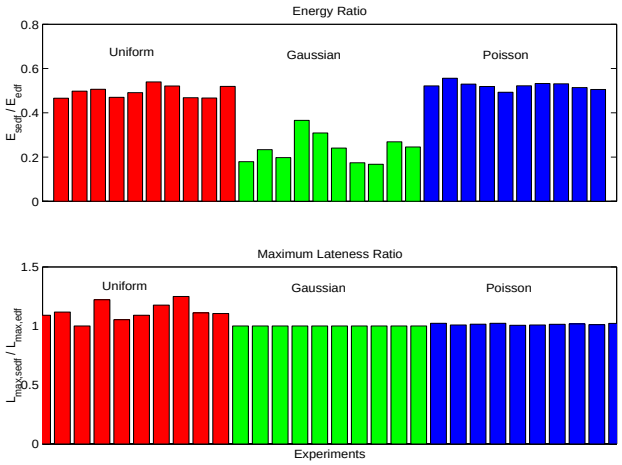


Fig. 7. Comparing the performance of EDF and SEDF

Finally, Fig. 8 shows the ratio of the energy consumption of the SEDF to the EDF case as a function of processor utilization. As the utilization increases, the slacking gets harder and the SEDF schedule tends to the EDF schedule with processing rate increasingly being set to 1.

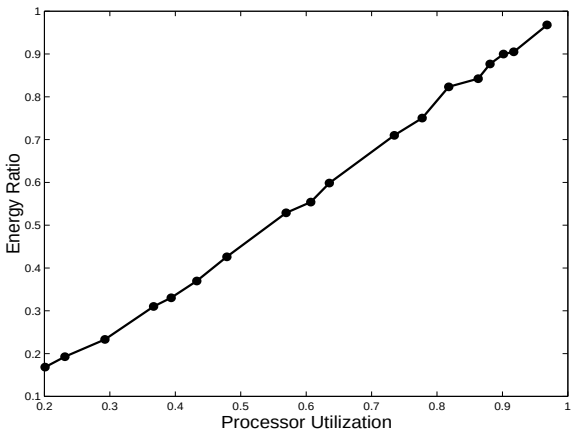


Fig. 8. Energy ratio as a function of processor utilization

3.5 Upper Bound on Energy Savings

Theorem 3: *Given a set of independent tasks with arbitrary arrival times, computation times and deadlines, the maximum energy savings possible by the any dynamic voltage and frequency setting algorithm that produces a schedule that meets all deadlines is bounded by* $E_{save}(r_{min})$*, where the processing rate*

$$r_{min} = \frac{\sum c_i}{\max(d_i)} \tag{12}$$

Proof: The denominator term of r_{min} is simply the maximum possible total time, T, allowed to finish all the tasks before their respective deadlines, i.e. $r_{min} = (\sum c_i)/T$. It is obvious that minimum energy results when tasks are slacked such that the entire time frame T is used up (i.e. processor utilization is 1). Assume that there exists an algorithm Λ, which is able to meet all deadlines and it schedules processing rates and tasks in each scheduling interval as shown in Fig. 3. A particular task τ_k, might get scheduled in different slots with different processing rates. Let the average processing rate seen by τ_k be $\bar{r}_k$. The actual computation time of τ_k is therefore $c_k/\bar{r}_k$, and therefore the absolute best case occurs when $T = \sum c_k/\bar{r}_k$. We will now show that minimum energy consumption occurs when all the processing rates are equal. We begin with the following inequality (which can be readily verified using the Cauchy-Schwarz inequality)

$$\left(\sum_k \frac{c_k}{\bar{r}_k}\right)\left(\sum_k c_k\bar{r}_k\right) \geq \left(\sum_k c_k\right)^2 \tag{13}$$

Rearranging the terms we get

$$\sum_k c_k\bar{r}_k \geq \left(\sum_k c_k\right)\left(\frac{\sum_k c_k}{T}\right) = \sum_k c_k r_{min} \tag{14}$$

The total normalized energy consumption is

$$E_{tot} = \sum_k \frac{c_k}{\bar{r}_k} E(\bar{r}_k) = \sum_k c_k\bar{r}_k \tag{15}$$

where we have substituted the quadratic energy consumption model of Fig. 1. The left-hand side of Equation 14 is the energy consumption for the schedule produced by Λ, while the right-hand side is the energy consumption of a schedule where all tasks have the same processing rate, r_{min}. Therefore, it can be concluded that minimum energy consumption (or maximum energy savings) occurs when all tasks have the same averaged processing rate. Using a similar argument, it can be shown that within a task, minimum energy consumption occurs when each of the different scheduled processing rates are equal to the average processing rate, $\bar{r}_k$.

The maximum savings, for example, with the task set shown in Fig. 6 is 74.5% (with $r_{min} = 0.5$). The savings by the SEDF algorithm was 53%. However, the comparison is not completely fair since the SEDF algorithm did not meet all the deadlines.

4 Periodic Task Scheduling

In this case our task set to be scheduled is denoted by

$$\vartheta = \{\tau_i(\phi_i, T_i, c_i), 0 < i \leq N\} \tag{16}$$

where ϕ_i is the phase, T_i is the time period and c_i is the computation time of the i^{th} task from a set of N periodic tasks. Once again, we assume that the tasks are independent, the system consists on one processor and preemption is allowed. Every task has to be executed once in each of its periods with the relative deadline being equal to the time period, T_i.

4.1 Dynamic Priority Assignments

Once again EDF is an optimal dynamic scheduling policy, reason being that EDF did not make any assumptions about tasks being periodic or aperiodic. EDF being intrinsically preemptive, the currently executing task is pre-empted whenever another periodic instance with a earlier relative deadline becomes active. Since the task set, ϑ, is completely determined *a priori*, the processing rate can also be determined completely and does not have to be adaptive. In fact the following theorem holds.

Theorem 4: *A set of periodic tasks is guaranteed to be schedulable with maximum energy savings iff the processing rate is*

$$r_{min} = \sum_i \frac{c_i}{T_i} \tag{17}$$

Proof: It has been shown in [1] that a periodic task set is guaranteed to be schedulable by EDF *iff* $\sum_i c_i/T_i \le 1$. Let $T = T_1T_2...T_N$, be an observation time frame. Using a line of reasoning exactly similar to the proof of **Theorem 3**, it can be shown that minimum processor energy consumption occurs when all tasks are slacked by the same amount, to the maximum allowable limit such that

$$\sum_i \frac{(c_i/r)}{T_i} \le 1 \Rightarrow r_{min} = \sum_i \frac{c_i}{T_i} \tag{18}$$

4.2 Static Priority Assignments

It has been shown also shown in [1] that the Rate-Monotonic algorithm is an optimal fixed priority algorithm. RM schedules tasks based on their periods, with priorities statically assigned to be inversely proportional to the task periods (i.e. the highest priority being assigned to task having the smallest period and so on). Since priorities are statically assigned, a ready task with a lower period will preempt another task with a higher period despite the fact that its relative deadline is earlier. With such a fixed priority assignment, the following theorem holds.

Theorem 5: *A set of N periodic tasks is guaranteed to be schedulable using fixed priority assignments with maximum energy savings if the processing rate is*

$$r_{min} = \frac{\sum_i \frac{c_i}{T_i}}{N(2^{1/N} - 1)} \tag{19}$$

Proof: It has been shown in [1] that RM guarantees that an arbitrary set of N periodic tasks is schedulable if the total processor utilization, $U = \sum_i c_i/T_i \le N(2^{1/N} - 1)$. The processing rate in Equation 19 can be derived exactly as shown in the proof of **Theorem 4**.

5 Conclusions

We have analyzed energy efficient scheduling algorithms for arbitrary independent periodic and aperiodic task sets characterized by real-time deadlines using variable voltage and frequency assignments on a single processor. The Slacked Earliest Deadline First (SEDF) algorithm has been proposed and it has been shown that SEDF is optimal in minimizing maximum lateness and processor energy consumption. A bound on the maximum energy savings possible with any algorithm, for a given task set, is also derived. Energy efficient scheduling for periodic task sets is also considered (both static and dynamic priority assignments) and optimal processing rate assignments have been derived under a guaranteed schedulability criteria.

ACKNOWLEDGEMENTS

This research is sponsored by the Defense Advanced Research Projects Agency's (DARPA) Power Aware Computing/Communication Program and Air Force Research Laboratory, Air Force Materiel Command, USAF, under agreement number F30602-00-2-0551.

REFERENCES

[1] C. L. Liu and J. W. Layland, "Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment", Journal of the Association for Computing Machinery, 1973, vol. 20, no. 1, pp. 46-61

[2] W. Horn, "Some Simple Scheduling Algorithms", Naval Research Logistics Quaterly, 21, 1974

[3] K. Ramamritham and J. A. Stankovic, "Dynamic Task Scheduling in Distributed Hard Real-Time Systems", IEEE Software, July 1984, vol. 1, no. 3

[4] T. Burd, et. al., "A Dynamic Voltage Scaled Microprocessor System", ISSCC 2000, pp. 294-295

[5] T. Pering, T. Burd and R. Broderson, "The Simulation and Evaluation of Dynamic Voltage Scaling Algorithms", International Symposium on Low Power Electronics and Design, 1998, pp. 76-81

[6] http://www.intel.com/mobile/pentiumIII/ist.htm

[7] http://www.transmeta.com/crusoe/lowpower/longrun.html

[8] http://www.amd.com/products/cpg/mobile/powernow.html

[9] G. Wei and M. Horowitz, "A Low Power Switching Power Supply for Self-Clocked Systems", International Symposium on Low Power Electronics and Design, 1996, pp. 313-318

[10] K. Govil, E. Chan and H. Wasserman, "Comparing Algorithms for Dynamic Speed Setting of a Low-Power CPU", Proceedings of the ACM International Conference on Mobile Computing and Networking, 1995, pp. 13-25

[11] A. Sinha and A. Chandrakasan, "Dynamic Voltage Scheduling Using Adaptive Filtering of Workload Traces", 14th International Conference on VLSI Design, Bangalore, Jan 2001

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

[13] I. Hong, M. Potkonjak and M. B. Srivastava, "On-Line Scheduling of Hard Real-Time Tasks on Variable Voltage Processor", Proceedings of ICCAD, 1998, pp 653-656

[14] M. L. Dertouzos, "Control Robotics: The Procedural Control of Physical Processes", Information Processing, vol. 74, 1974

[15] V. Gutnik and A. P. Chandrakasan, "An Embedded Power Supply for Low-Power DSP", IEEE Transactions on VLSI Systems, vol. 5, no. 4, Dec. 1997, pp. 425-435