Real-Time Task Scheduling for Energy-Aware Embedded Systems[1]

Vishnu Swaminathan and Krishnendu Chakrabarty
Dept. of Electrical & Computer Engineering
Duke University
Durham, NC 27708
{*vishnus,krish*}*@ee.duke.edu*

ABSTRACT

We present a new approach for scheduling workloads containing periodic tasks in real-time systems. The proposed approach minimizes the total energy consumed by the task set and guarantees that the deadline for every periodic task is met. As embedded software becomes a larger component of system-on-a-chip design, energy management using the operating system becomes increasingly important. We present a mixed-integer linear programming model for the NP-complete scheduling problem and solve it for moderate-sized problem instances using a public-domain solver. For larger task sets, we present a novel low-energy earliest-deadline-first (LEDF) scheduling algorithm and apply it to two real-life task sets.

1 Introduction

A number of energy-constrained embedded and mobile systems are designed for real-time use. These systems must be designed to meet both functional *and* timing requirements [2]. Thus, the correct behavior of these systems depends not only on the accuracy of computations but also on their timeliness. While energy minimization for embedded and mobile computing is of great importance, energy consumption must be carefully balanced against the need for real-time responsiveness.

One approach to conserve energy is to employ low-power design methodologies. These methods, however, are limited in that they can be used only in systems with a fixed supply voltage. Most modern computers are equipped with a voltage supply that is capable of providing at least two different voltages to the system, and the above methods are comparatively less effective than ones that are tailored to make use of this feature.

In embedded systems with variable-speed processors, the operating system (OS) can reduce energy consumption by scheduling tasks appropriately. Energy minimization by adjusting CPU speed was first studied in [12]. For real-time systems, optimal preemptive off-line scheduling algorithms have also been provided [13]. Hong et al. presented a heuristic for the preemptive model with a limit on the number of speed changes [4]. Although the scheduling methods cited above are very efficient, most of them make the assumption that the CPU can operate at several different voltage levels (and hence different clock frequencies) which can be varied continuously. In addition, a number of these methods are aimed at the synthesis of low-power designs and they do not address energy minimization during field operation.

In this paper, we present an on-line scheduling algorithm for real-time systems that attempts to minimize the energy consumed by a periodic task set. This is based on the well-known earliest-deadline-first (EDF) algorithm [8]. We consider a practical scenario where a single CPU executes a set of periodic non-preemptable tasks. The voltage, and consequently the clock speed, of the CPU may be switched between two or more values dynamically at run-time through OS system calls. This option is available in most modern computers, which provide at least two different operating speeds [5]. We attempt to find the voltage at which each task must be executed such that the energy consumed by the entire set of periodic tasks is minimized and generate a schedule for the task set such that the release time requirements are satisfied and the deadlines for every task is met.

2 Preliminaries

In this section, we present our notation and the underlying assumptions. We are given a set of n periodic tasks $R = \{r_1, r_2, \ldots, r_n\}$. Associated with each task $r_i \in R$ are (i) its *release* (or arrival) time a_i, (ii) its deadline d_i, (iii) its length l_i (represented in number of instruction cycles), and (iv) its period p_i. Each task is released at time $t = a_i$. In other words, each task can begin execution only at or after time $t = a_i$. Release times are arbitrary in that a task may be released at any time before its deadline. We assume, without loss of generality, that all tasks have identical periods. If the periods of tasks are different, we can perform a polynomial transformation to this scenario through the application of the LCM theorem [7]. In such a case, there may be a need to execute a given task several times within the

[1]This research was supported in part by DARPA under grant no. N66001-001-8946, and in part by a graduate fellowship from the North Carolina Networking Initiative.

"hyperperiod". All tasks must complete execution before their deadlines.

We assume that the CPU can operate at one of two voltages: V_1 or V_2. Depending on the voltage level, the CPU speed may take on two values: s_1 or s_2.The model can be easily extended to handle more than two voltages (speeds). The supply voltage to the CPU is under OS control and the OS may dynamically switch the voltage during runtime with relatively low overhead. We restrict ourselves to two speeds out of practical considerations. CPU speeds are specified in terms of the number of instructions executed per second. Each task r_i may be executed at a voltage v_i, $v_i \in \{V_1, V_2\}$, and correspondingly, at a speed x_i, $x_i \in \{s_1, s_2\}$. Tasks are not preemptable, i.e., once a task starts executing, no other task can execute until it completes execution. There is also no *inserted idle time*. This means that the scheduling algorithm does not allow the processor to be idle if there is a task that has been released but has not completed execution.

It is well-known that power consumption in CMOS circuits has a quadratic dependence on the CPU voltage. Therefore, the energy E_i consumed by task r_i of length l_i is $E_i \propto v_i^2 \cdot l_i$. Thus, energy consumption of task r_i varies quadratically with its assigned processor voltage v_i and linearly with its length l_i.

The low-energy scheduling problem can be shown to be NP-complete. Nevertheless, it can be solved exactly for moderate-sized instances using mixed-integer linear programming (MILP).

3 Mixed-integer linear programming model

The optimization problem we address is to minimize the total energy consumed by the set of n tasks by optimally determining their start times $t_1, t_2, \ldots, t_n$, their voltages and corresponding execution speeds.

The following constraints need to be modeled: (i) CPU speeds are limited to one of two values—s_1 or s_2, (ii) The deadline for each task must be met, (iii) Tasks are non-preemptable, and (iv) A task may start only after it has been released.

We observed in Section 2 that the energy consumed by task r_i is proportional to $v_i^2 l_i$.

Hence, the objective function is **Minimize** $\sum l_i v_i^2$.

The modeling of the constraints and their subsequent linearization are omitted here due to space limitations.

The MILP model is too computationally-intensive to be used for large task sets. However, it is helpful in providing a lower bound on the amount of energy consumed by a given task set. In our MILP formulation, a priori knowledge of the release times has been inherently assumed. We observe that energy can be minimized to a greater extent in the off-line case than in the on-line one. This justifies use of MILP as a comparison tool for providing lower bounds on energy consumption.

4 The LEDF heuristic

Although MILP is a useful and optimal solution method for small problem instances, it cannot be used for large test cases. In order to solve large problem instances, we have developed a heuristic algorithm to generate near-optimal solutions in polynomial time.

The **low-energy earliest deadline first** heuristic, or simply LEDF, is an extension of the well-known earliest deadline first (EDF) algorithm. The operation of LEDF is as follows:

LEDF maintains a list of all released tasks, called the "ready list". When tasks are released, the task with the nearest deadline is chosen to be executed. A check is performed to see if the task deadline can be met by executing it at the lower voltage (speed). If the deadline can be met, LEDF assigns the lower voltage to the task and the task begins execution. During the task's execution, other tasks may enter the system. These tasks are assumed to be placed automatically on the "ready list". LEDF again selects the task with the nearest deadline to be executed. As long as there are tasks waiting to be executed, LEDF does not keep the processor idle. This process is repeated until all the tasks have been scheduled. Figure 1 gives the algorithm in pseudocode form.

```
Procedure LEDF()
begin
Repeat forever
{
Are tasks waiting to be scheduled in ready list?
    yes
    Sort deadlines in ascending order
    Schedule task with earliest deadline
    Check if deadline can be met at lower speed (voltage)
    If deadline can be met, schedule task to execute at lower voltage (speed)
    If deadline cannot be met, check if deadline can be met at higher speed (voltage)
    If deadline can be met, schedule task to execute at higher voltage (speed)
    If deadline cannot be met, task cannot be scheduled. Call exception handler!
    no
    do-nothing
}
end
```

Figure 1. The LEDF algorithm

Our algorithm has a computational complexity of $O(n \log n)$. The worst-case scenario occurs when all n tasks are released at time $t = 0$. This involves sorting n tasks in the ready list and then selecting the task with the earliest deadline for execution. Given that all n tasks have already arrived and that they are already sorted by deadline, we no longer need to perform sorting on the task set.

Task	Release a_i	Deadline d_i	Length l_i (x 10^6)	$l_i/300$ (x 10^6)	$l_i/400$ (x 10^6)
t_1	3	7	800	2.66	2.0
t_2	9	21	750	2.5	1.875
t_3	0	5	1600	5.33	4.0
t_4	18	25	1000	3.33	2.5
t_5	14	16	600	2.0	1.5
t_6	7	10	1200	4.0	3.0
t_7	20	27	1100	3.66	2.75
t_8	14	20	1600	5.33	4.0
t_9	11	14	500	5.0	3.75
t_{10}	30	35	1400	4.66	3.5
t_{11}	27.5	30	800	2.66	2.0
t_{12}	40	42	600	2.0	1.5
t_{13}	34	39	1600	5.33	4.0
t_{14}	40	46	1200	4.0	3.0
t_{15}	44	50	1400	4.66	3.5
t_{16}	44	55	2000	6.66	5.0
t_{17}	40	43	300	1.0	0.75

Table 1. A simple task set consisting of 17 tasks.

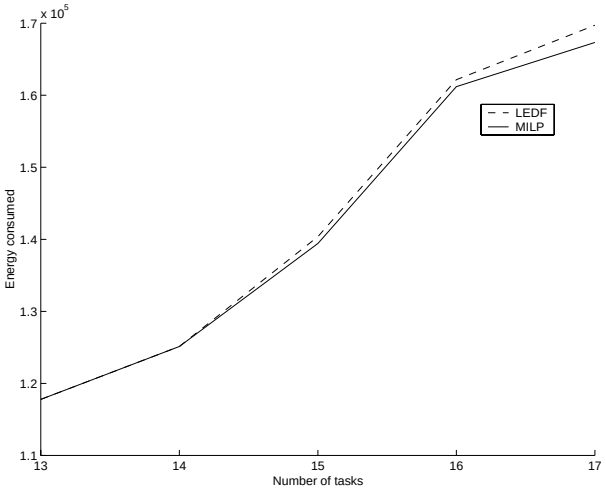

Figure 4. Comparison of schedules generated by MILP and LEDF.

Scheduling a task for execution has $O(1)$ complexity. This results in a worst-case execution time of $O(n \log n)$.

We now present our experimental results. First we show the results of the MILP simulations and LEDF for a task set of seventeen tasks.

Our example task set is given in Table 1. It consists of seventeen tasks r_1 to r_{17}. Each task has a release time, a deadline and length. We assume the two processor speeds to be 300 million instructions per second (MIPS) at 2.47 V and 400 MIPS at 3.3 V.

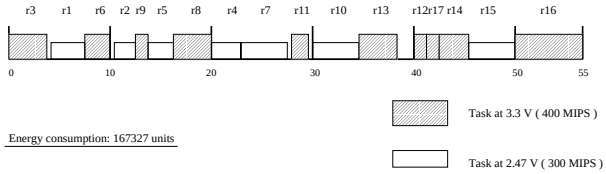

Figure 2. Schedule generated using MILP

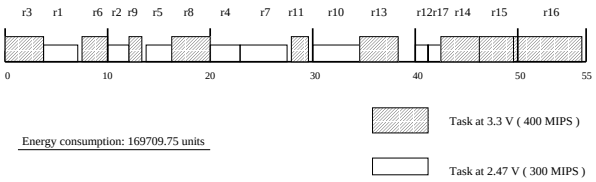

Figure 3. Schedule generated using LEDF

The execution times for the tasks at the two different speeds are also shown. Figure 2 shows the schedule generated using MILP. The energy consumed by the schedule generated through MILP is 167327 units (measured by the sum of the $v_i^2 l_i$ values).

Figure 3 shows the corresponding LEDF schedule for the same task set. The energy consumed by this schedule is 169709.75 units. We observe that the energy consumed here is marginally greater than that for the MILP schedule. Nevertheless, our results show that our algorithm can generate near-optimal schedules with an energy increase of around 1%.

We observe that the order in which tasks are executed remains the same in both the MILP and LEDF schedules. The increased energy consumption of LEDF arises due to the fact that LEDF does not possess knowledge of the release times a priori. Our MILP model, which does not have such a restriction, executes tasks r_{12} and r_{17} at a higher speed (voltage) even though both could have met their respective deadlines by executing at a lower speed (voltage). We showed in Section 2 that energy consumed by a task is proportional to its length. Since the length of task r_{15} is greater than the lengths of both r_{12} and r_{17}, we see that by executing r_{12} and r_{17} at a higher speed (voltage), we can execute r_{15} at a *lower* speed (voltage) and thus reduce the effect of the length of r_{15} on energy. This, in fact, does result in an optimal schedule. This characteristic of the task set cannot be utilized by LEDF, or any other on-line scheduling algorithm, because no scheduling algorithm has prior knowledge of release times. In order to evaluate the effectiveness of LEDF, we generated energy-optimal schedules using MILP and the corresponding schedules using LEDF for task sets containing upto seventeen tasks. The results, plotted in Figure 4, show that LEDF produces optimal schedules for upto fourteen tasks. Beyond that, it slowly starts to diverge from the optimal schedule.

We also applied the LEDF algorithm to a real-time task

set consisting of twenty-four tasks. This task set was used for the development of an application-specific integrated circuit for an avionics application [10]. The processor speeds were assumed to be 100 MIPS and 260 MIPS, with corresponding voltages 1.2 V and 3.3 V respectively. The energy consumed by this task set was 203803.55 units. If all tasks are scheduled at the higher speed, the energy consumed is as high as 360938 units, i.e. 77% higher. On the other hand, if only the lower speed is used, as many as seven tasks miss their deadlines.

Finally, we applied LEDF to a 39-task example. This task set comes from an embedded signal processing application for an anti-submarine warfare (ASW) system [1]. The task set runs on a Mercury PowerPC 6U VME board with a 200 MHz 603e processor. We have assumed that the processor is capable of operating at 100 MHz and that the processor speeds are dynamically switchable. We have also assumed that the CPU operates at 200 MHz at 3.3 V and at 100 MHz at 1.65 V. Furthermore, in order to emphasize energy minimization, the deadlines we have used are tighter than the actual deadlines for the actual task set. For this example, LEDF generated a schedule that consumed 323680.78 units of energy. For this task set, if only the higher speed (voltage) is used, the energy consumed is 477482 units, i.e, 47.5% higher. However, if only the lower speed is used, seventeen tasks miss their deadlines.

The MILP model took prohibitively large amounts of time for scheduling task sets consisting of more than seventeen tasks running on a dual-processor Sun Ultra-1 with a 256 MB memory capacity. For an eighteen task data set, MILP took over a day to generate the optimal solution. On the other hand, LEDF took under a second to generate a near-optimal solution for the thirty-nine task example.

References

[1] J. Anderson, private communication, March 2000.

[2] G. C. Buttazzo, *Hard Real-time Computing Systems: Predictable Scheduling Algorithms and Applications*, Kluwer Academic Publishers, Norwell, MA, 1997.

[3] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, Freeman and Co., New York, NY, 1979.

[4] I. Hong, G. Qu, M. Potkonjak, and M. B. Srivastava, "Synthesis techniques for low-power hard real-time systems on variable-voltage processors", *Proc. Real-time Systems Symposium*, pp. 178–187, 1998.

[5] Intel Pentium III Processor Brief, *http://www.intel.com*

[6] K. Jeffay, D. F. Stanat and C. U. Martel, "On non-preemptive scheduling of periodic and sporadic tasks with varying execution priority," *Proc. IEEE Real-Time Systems Symposium*, pp. 129–139, December 1991.

[7] E. L. Lawler and C. U. Martel, "Scheduling periodically occurring tasks on multiple processors", *Information Processing Letters*, vol. 12, no. 1, pp. 9–12, 1981.

[8] C. L. Liu and J. Layland, "Scheduling algorithms for multiprogramming in a hard real-time environment", *Journal of the ACM*, vol. 20, pp. 46–61, 1973.

[9] J. R. Lorch and A. J. Smith, "Scheduling techniques for reducing processor energy use in MacOS", *Wireless Networks,*, vol. 3, pp. 311–324, 1997.

[10] J. H. Wensley, K. N. Levitt, M. W. Green, J. Goldberg and P. G. Neumann, " Design of a fault tolerant airborne digital computer. Volume 1: Architecture," Final Report, Stanford Research Institute, Menlo Park, CA, October 1973.

[11] J. M. Rabaey and M. Pedram, *Low Power Design Methodologies*, Kluwer Academic Publishers, Norwell, MA, 1996.

[12] M. Weiser, B. Welch, A. Demers and S. Shenker, "Scheduling for reduced CPU energy," *Proc. Symposium on Operating System Design and Implementation*, pp. 13–23, 1994.

[13] F. Yao, A. Demers and S. Shenker, "A scheduling model for reduced CPU energy", *Proc. IEEE Annual Foundations of Computer Science*, pp. 374–382, 1995.