Situation Aware Scheduling for Energy-Efficient Real-Time Systems

Frank Bodmann
INCHRON GmbH
Potsdam, Germany

Nina Mühleis
University of Erlangen-Nuremberg
Department of Computer Science
Erlangen, Germany

Frank Slomka
University of Ulm
Institute of Embedded Systems/Real-Time Systems
Ulm, Germany

***Abstract*—The aim of an energy-efficient real-time scheduler is to reduce power consumption by reducing the voltage and the frequency of a processor while preserving temporal correctness. Most of the research in this area is restricted to the periodic task model. In this paper we present an energy-efficient dynamic scheduling algorithm that uses the more expressive event spectrum model for the description of task activations. The additional information available through this model is used to gain a better understanding of the system's situation and allows an accurate assessment of future system activity. This enables the scheduler to adjust the processor speed more energy-efficiently than previous approaches. Descriptive examples are provided to illustrate the mechanisms and the effect of the proposed scheduling strategy.**

I. Introduction

Innovation in the field of mobile computing devices is currently limited by power consumption and battery life. For real-time systems however it is important to remember that these systems do not necessarily have to work fast, but just fast enough. For every task in the system time constraints are given which must never be violated. In order to meet these time constraints it is not always required to use the full computing power of the system. For complex real-time systems the required computing power may vary strongly during run-time depending on its task activity. By dynamically recognizing the required computation power at run-time, the processor frequency can be adapted and energy preserved based on the system's present situation.

A powerful model to describe task activations in a system is the *event spectrum model* also known as the *event stream model*. In this model a set of minimum time intervals and the respective number of task activations that may occur within them is specified. As the given intervals are minimum time intervals, task activations in the running system may occur more diffused. An online scheduler can compare past task activations to the specified intervals and conclude when future activations may occur at the earliest. From this the future processor load can be calculated and the processor frequency may be adjusted.

In addition to changing the processor frequency the operating voltage can also be adapted. In CMOS logic, used in most microprocessors today, the voltage depends on the operating frequency. When the frequency is reduced the processor can be supplied with lower voltage. This is highly effective because the systems energy consumption grows quadratically with the operating voltage. A wide range of processors across all domains offer the opportunity to adjust the operating frequency and voltage dynamically.

II. Related work

The event spectrum model has been used in the context of dynamic energy aware real-time scheduling in [1], where power consumption was deferred for the optimal use of a regenerative energy source. An overall reduction of the power consumption however was not intended in that work. In [2] the event spectrum model is used to determine the permissible slowdown factor for a real-time system. The work describes an analysis to be performed during the design phase and does not examine its use for an on-line scheduling algorithm.

A set of real-time scheduling algorithms using dynamic voltage scaling for low-power embedded systems can be found in [3]. Using a simple periodic description of task activations however leaves the scheduler with limited information, so the scheduler may be either too hesitant to slow down the processor or the saved energy by slowing down the processor early may be more than used up again as the processor speed may have to be increased later to catch up on the deferred work.

The sporadic task model is used in [4] to describe a dynamic scheduling algorithm. The sporadic task model describes the minimum time interval between two task activations whereas the event spectrum model describes the minimum time intervals for an arbitrary number of activations. A scheduling algorithm using the event spectrum model can therefore assess future activations beyond the next one more accurately.

Trying to cope with the limited information provided by the periodic task model, [5] uses a control loop to adjust to the task activity of the system as it is running. This approach considers varying execution times instead of varying task activation times.

III. Event Spectrum Model

In this section the task activation model used by this approach, introduced in [6] as the event stream model, is described. A task is activated by an event. The event spectrum model is used to describe the timing of these events. The key question this approach answers is: How many events can at most occur within any time span of a given length? This allows an abstraction from the absolute time towards a description in arbitrary time intervals. This provides a powerful way to describe the timing of events and at the same time allows an efficient analysis of the real-time properties of a system.

An event spectrum is a set of event elements:

$$ES = \left\{ \begin{pmatrix} p_1 \\ a_1 \end{pmatrix}, \begin{pmatrix} p_2 \\ a_2 \end{pmatrix}, ..., \begin{pmatrix} p_n \\ a_n \end{pmatrix} \right\}. \quad (1)$$

Each element represents an additional event that can occur when the observed interval has a size equal to or larger than a_i. The period p_i allows the event to be repeated every p_i time units.
If all event elements have the same period the event spectrum is called homogeneous. In this paper only homogeneous event spectra are considered.

In Fig. 1 only one event occurs concurrently so there is a maximum of one event in any interval with a size towards zero. As the interval grows, the number of events that can occur within it increases. Two events can occur in an interval sized two, etc. So the depicted sequence of events adheres to the event spectrum $ES = \{\binom{p}{0}, \binom{p}{2}, \binom{p}{10}, \binom{p}{16}, \binom{p}{21}\}$. The period p indicates that a set of events following the same constraints may occur every p time units. A sequence of events that matches this event spectrum could be more relaxed: e.g. the event at time 10 may occur at 11 instead.

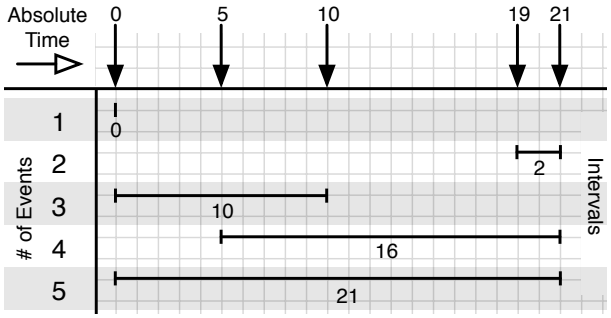


Fig. 1. Event spectrum model

The maximum number of events that can occur in time interval I can be calculated by the event bound function where n is the number of event elements describing the event spectrum:

$$E(I) = \sum_{i=1}^{n} \left\lfloor \frac{I - a_i}{p_i} + 1 \right\rfloor \quad (2)$$

The inverse function of the event bound function provides the minimum interval in which a given number of events m may occur:

$$a\,(m) = p \cdot \left\lfloor \frac{m}{n+1} \right\rfloor + a_i, \text{ with i} = \text{m modulo n} + 1 \quad (3)$$

By scaling the event bound function by the maximal computation time c each event causes, the amount of requested computation time for intervals can be determined. By reducing the size of the given interval by the deadline d the computation time of those events that have to be completely processed during the interval is retrieved. So the demand bound function $C(I)$ expresses how much demand for computation time is at most possible for any interval [7].

$$C(I) = E(I - d) \cdot c \quad (4)$$

Event spectra do not have to be specified manually, but can automatically be derived from a system description given in arbitrary programming language [8].

IV. Situation Aware Scheduling using the Event Spectrum Model

In many situations real-time systems do not need the full computing power of their processors. Situation Aware Scheduling is based on Earliest Deadline First (EDF) scheduling. After activation every task must be completed before a defined deadline is reached. If there is more than one task in the system that needs access to the processor, the task with the earliest deadline gets access to the processor first. The idea of Situation Aware

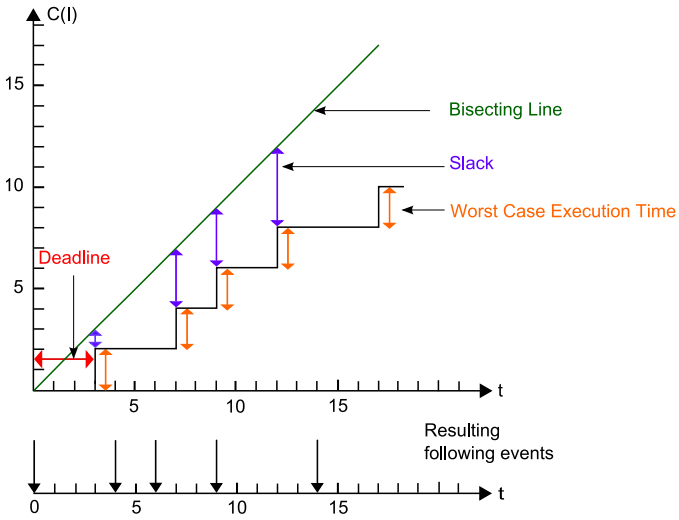


Fig. 2. Demand bound function

Scheduling is to adjust the processor's frequency so that no task violates its deadline while the system runs as slow as possible thus maximizing the load: the system can perform the tasks at hand but no more. This is done by assessing upcoming task activations.

A possible way to gain information about future task activations is the event spectrum model. This model provides the event bound function (Equation 2) that characterizes the maximum number of events that may occur in any time interval. If past task activations of a running system are taken into consideration, the event bound function can be used to calculate the worst possible time for future events. In other words time intervals provided by the event bound function are mapped to absolute time. So the processor load for the future can be calculated as it depends on these future events, their deadlines and their Worst Case Execution Times (WCET). The processor load can be expressed by the demand bound function of the event spectrum model (Equation 4).

Consider a task with expected task activations as shown in Fig. 2. A step in the demand bound function occurs when the deadline of a task activation is reached because up to this point in time the computation time must have been made available. The height of the step is equal to the WCET of the task. The bisecting line represents time available for calculations. The minimal distance from the function to the bisecting line is equivalent to time that is not used by EDF scheduling but could have been used as not all deadlines have passed while the processor is already idle. This time is called slack.

A. Determining the remaining computation time

In order to calculate the maximal expected processor usage, it is necessary to keep track of the remaining calculation times from past activations. At the moment an event occurs, the complete WCET of the task is its remaining computation time. Whenever the task gets access to the processor this time is decremented.

B. Determining the lists of future events

For every task the scheduler sustains a sorted list of earliest points in time t_j when the next, 2nd to next, 3rd to next,... activation may occur. It is initialized by the event bound function, with: $t_j = a_j$. All times are stored in relation to a point in time t_{old}. The list has to be updated whenever an event occurs and whenever the future processor load is calculated. The length of this list depends on the length of the event spectrum (n). The list always contains one more element than the set of elements of the event spectrum $(n+1)$.

1) Updating the list of future events of task T_i: An occurring event implicates that the first element in the list may be deleted as the occurring event was represented by that element. All following elements move up accordingly: Before the update times t_1 to t_{n-1} were relative the moment in time when the list was last updated (t_{old}). Since the last update $t_p = t_{now} - t_{old}$ time units have passed. So the new relative points in time are $t'_j = t_{j+1} - t_p$. In Fig. 3 this is shown in the first line. When updating the list, the information available in the list has to be again combined with the constraints provided by the event bound function. The inverse function $a(m)$ of the event bound function represents the minimum interval in which m activations may occur. Thus the following events can occur at the earliest at:

$$t'_j = max((t_{j+1} - t_p),\ a(j))\ \forall\ 0 \leq j < n \qquad (5)$$

The process is illustrated in Fig. 3.

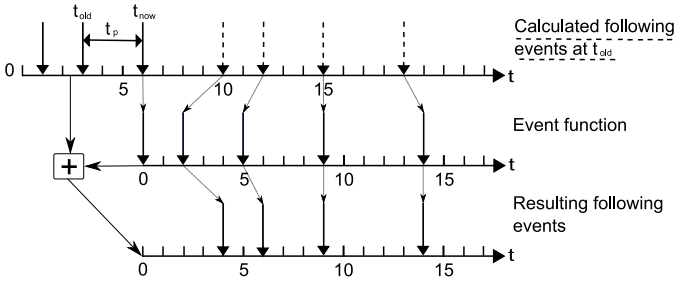


Fig. 3. Update following Events

The new value for the last element t_n of the list has to be determined from scratch. As it represents the $(numberOccuredEvents + n + 1)$ths activation of the task it can be computed as:

$$t_n = a(numberOccuredEvents + n + 1) - t_{now} \quad (6)$$

If fewer than the allowed number of events occur within a period, the event bound function can be reset to its origin at the start of the new period.

2) Updating the lists of future events of all other tasks: When an event occurs, also the list of the following events of the tasks on the same processor that the event did not belong to have to be updated. If the next possible events are still in the future, their expected arrival times are simply decremented by t_p. If one or more expected events of the task did not occur at the earliest possible time, a new event can occur at any time. The distance to the following event is 0. For the second, third,... event the algorithm must again ensure that the event bound function is not violated. So the following events can occur at $t'_j = max(t_j,\ a(j))$.

C. Calculating the Processor Frequency

The minimal processor frequency allowed without violating a deadline depends on the predicted future events for every task in the system, their deadlines, WCETs and the remaining computation time of past task activations. The demand bound function of the whole system is the sum of the individual demand bound functions of every task. In order to find the minimal slack, the slack for every step in the demand bound function of the system must be calculated. Due to different relative deadlines of individual tasks the order of occurring events may not match the order of steps in the individual demand bound functions. In order to reduce the complexity of examining the demand bound function a sorted list of steps in the function is used. The steps occur at $s_j = t_j + D_i$. When examining the function for every task it is known which step in the function occurs next. This way the next step in the function can be found by comparing the next steps of all tasks to each other in their demand bound function. The complexity is linear to the number of tasks in the system.

The slack has to be computed at every point in time a step occurs. Occurring steps mean that the value of the demand bound function is increased by the WCET of the task the step belongs to. The value of the demand bound function at time$_k$ is $demand_k = demand_{k-1} + wcet$ and thus $slack_k = time_k - demand_k$. Remaining computation times from previous task activations are added to the demand bound function at their accompanying deadlines.

An implementation for building the demand bound function is shown in listing 1.

Listing 1. Code for slack calculation

```
Time slack(stepCount){
    Time slack = MAX_TIME;
    Time demand = 0;
    for(k = 0; k < stepCount; k++){

        Time time_k = timeOfNextStep();
        Time wcet = wcetOfNextStep();

        // Value of demand bound
        // function at time_k
        demand = demand + wcet;
        slack = min(slack, time_k-demand);
    }
    return slack;
}
```

When computing the minimal slack a time t_{term} is needed at which the calculation can be terminated safely. At this time the processor speed must be inferred from the slack with the guarantee that no deadline will be violated before or after. This guarantee can be given if all activations before t_{term} have to be completed before t_{term}. This condition can be expressed as: $\forall\ t_i\ < t_{term}$: $s_i \leq t_{term}$. For an event spectrum with n elements $n+1$ steps are considered for slack calculation. The last saved step of task T$_i$ occurs at $t_{i\_last} = p_i + D_i$. The sum of steps for every task within the smallest t_{i_last} can be used as the maximum number of steps to be considered. If no termination situation is found within the maximum number of steps, the slack must be assumed to be zero.

The processor time used by the scheduler itself can be accounted for as additional steps in the demand bound function. The minimum processor frequency for task T$_i$ depends on the slack at the point in time when the task gets access to the processor the first time and the WCET of the task T$_i$. The scale factor for the processor frequency is:

$$FrequencyScale\ =\ \frac{wcet}{wcet\ +\ slack}. \quad (7)$$

This formula can in turn be used to select the appropriate frequency form a finite set of working points of the processor. Reducing the tasks frequency implicates a longer execution time for the task. The initial remaining computation time for a task is its WCET.

$$ComputationTimeRemaining' = \frac{ComputationTimeRemaining}{FrequencyScale} \quad (8)$$

V. Evaluation

We have developed a test environment to evaluate the effect of Situation Aware Scheduling. The load of the system is compared to the maximal load possible and to that generated by an EDF scheduler. The maximal load possible is lower than 100% when there are times when no task can work: all deadlines have passed while all following activations are still in the future.

In the following a set of three tasks is analyzed with these characteristics:

$$\begin{aligned}
\text{Task 1: } ES_1 &= \left\{ \binom{40}{0}, \binom{40}{9}, \binom{40}{20} \right\} \\
wcet_1 &= 2\ ms, deadline_1 = 7\ ms \\
\text{Task 2: } ES_2 &= \left\{ \binom{20}{0}, \binom{20}{6}, \binom{20}{13} \right\} \\
wcet_2 &= 2\ ms, deadline_2 = 4\ ms \\
\text{Task 3: } ES_3 &= \left\{ \binom{10}{0}, \binom{10}{5} \right\} \\
wcet_3 &= 1\ ms, deadline_3 = 2\ ms
\end{aligned}$$

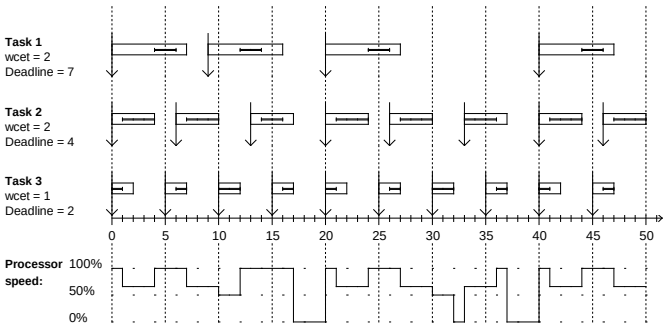
Thus the earliest possible activations [ms] are:

- Task 1: 0, 9, 20, 40, 49, 60, 80,...
- Task 2: 0, 6, 13, 20, 26, 33, 40,...
- Task 3: 0, 5, 10, 15, 20,...

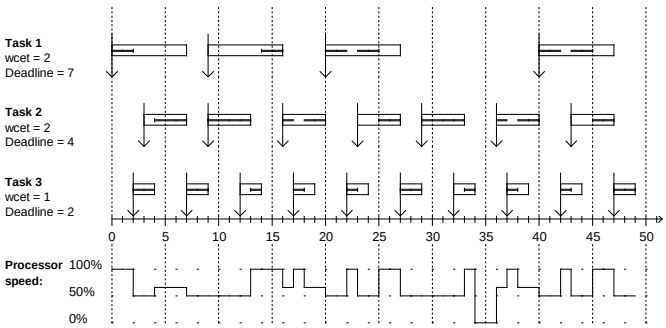
The maximum number of steps for calculating the slack was set to 5. Fig. 4 shows the simulation results of several activation patterns. In Table I the load of the system in these cases when using the algorithm for Situation Aware Scheduling (SAS) is compared to the load that is reached by a common EDF scheduler and the maximum load possible. The best results are achieved when events occur as expected. In the majority of these cases the maximal possible load is reached. The least significant improvements are achieved if an event occurs significantly later than expected. In this case the algorithm must expect an event for this task at any moment. Enough computation time must be remaining so no deadline will be violated if the event finally occurs.

We compared these results to dynamic voltage scaling algorithms using the periodic task model. However performing a real-time analysis on a task system that has been transferred from a description by event spectra to a description by periodic tasks, usually results in a predicted system utilization of more than 100%. Consequently algorithms such as the Look-Ahead RT-DVS [3] in fact try to set the processor to frequencies well above 100%, rendering comparisons like these meaningless.

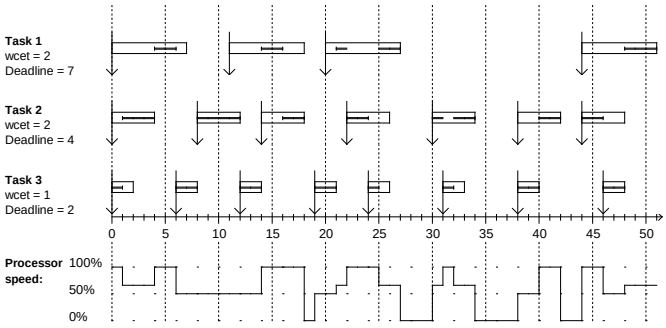
Case 1: All tasks are activated at $t = 0$, all following events occur as soon as possible



Case 2: The first activations of the tasks are dephased, all following events occur as soon as possible



Case 3: All tasks are activated at $t = 0$, following activations may arrive later than possible



Case 4: Some activations do not occur within their periods (skipped activations)

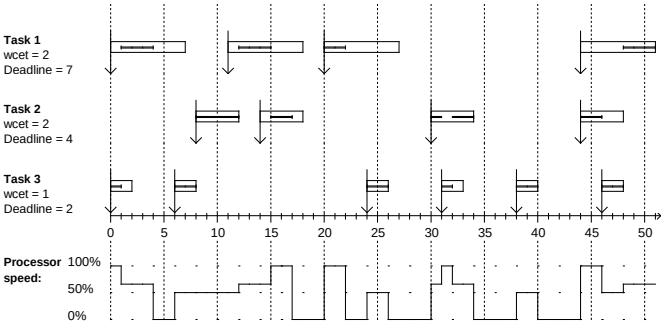


Fig. 4. Example with three tasks

TABLE I
ANALYSIS RESULTS

Case	Execution Time [ms]		Load max.	Load	Load	Average Processor Speed
	EDF	SAS		EDF	SAS	SAS
1	26	33	82,5%	65%	82.5%	80%
2	33	45	95,7%	70%	95.7%	68.6%
3	30	41	80,4%	58%	80.4%	80.4%
4	21	33	76,4%	41%	64.7%	78.4%

VI. CONCLUSION

In this paper we have shown how a dynamic real-time scheduler can gain an improved situation awareness allowing improved dynamic voltage scaling. This situation awareness is achieved by matching comprehensive information on possible task activation patterns against actual past task activations. As a result the scheduler has an insight into possible future task activations and can adjust the processor speed in order to minimize power consumption while ensuring that all real-time guarantees are met.

We demonstrated the effectiveness of the Situation Aware Scheduler in a simulation. The degree of energy savings depends on the actual task activations. Best results are achieved when events occur alongside their initial specification.

Currently the objective of the algorithm is to minimize the required working frequency of the processor. With the increasing importance of leakage in current processors, in some situations power consumption can be further reduced by temporarily deactivating the processor. Future work should consider extending the scheduler to find these situations and drive the processor accordingly.

REFERENCES

[1] C. Moser, L. Thiele, L. Benini, and D. Brunelli, "Real-Time Scheduling with Regenerative Energy," *Proceedings of the 18th Euromicro Conference on Real-Time Systems (ECRTS)*, pp. 261–270, 2006.

[2] H. Lipskoch, K. Albers, and F. Slomka, "Fast calculation of permissible slowdown factors for hard real-time systems," *Integrated Circuit and System Design. Power and Timing Modeling, Optimization and Simulation*, pp. 495–504, 2007.

[3] P. Pillai and K. Shin, "Real-time dynamic voltage scaling for low-power embedded operating systems," *Proceedings of the eighteenth ACM symposium on Operating systems principles*, pp. 89–102, 2001.

[4] A. Qadi, S. Goddard, and S. Farritor, "A dynamic voltage scaling algorithm for sporadic tasks," *Real-Time Systems Symposium, 2003. RTSS 2003. 24th IEEE*, pp. 52–62, 2003.

[5] Y. Zhu and F. Mueller, "Feedback EDF scheduling exploiting dynamic voltage scaling," *Real-Time and Embedded Technology and Applications Symposium, 2004. Proceedings. RTAS 2004. 10th IEEE*, pp. 84–93, 2004.

[6] K. Gresser, "An event model for deadline verification of hard real-time systems," in *5th Euromicro Workshop on Real-Time Systems*, Finland, 1993.

[7] S. K. Baruah, D. Chen, S. Gorinsky, and A. K. Mok, "Generalized multiframe tasks," *Real-Time Systems*, vol. 17, no. 1, pp. 5–22, 1999.

[8] F. Bodmann, K. Albers, and F. Slomka, "Analyzing the timing characteristics of task activations," *Proceedings of the first IEEE Symposium on Industrial Embedded Systems (SIES)*, 2006.