DEVICE-CENTRIC LOW-POWER SCHEDULING FOR REAL-TIME EMBEDDED SYSTEMS

PAO-ANN HSIUNG and HSIN-CHIEH KAO
Department of Computer Science and Information Engineering, National Chung Cheng University, 160, San-Hsing, Min-Hsiung, Chiayi, Taiwan-621, ROC
hpa@computer.org

Existing low power schedules mainly try to decrease overall system energy usage by shutting down unused devices after process scheduling. This often leads to suboptimal energy usage due to the lack of process slack time to shut down unused devices in a fixed system schedule such as that generated by a rate-monotonic or an earliest deadline first scheduler in a real-time embedded system. In this work, we try to integrate process scheduling with low power scheduling such that a low-power real-time feasible schedule is obtained. The proposed method called Low-Power Quasi-Dynamic Scheduling (LQS) was implemented and applied to some examples, including a sensor network node and Bluetooth devices, to prove its benefits.

Keywords: Device scheduling; low-power real-time embedded systems; Petri nets.

1. Introduction

In real-time embedded systems, there is a tradeoff between the satisfaction of time constraints and the optimization of energy usage because shorter schedules require more energy and vice versa. We try to characterize this tradeoff and implement a low-power real-time scheduler for the synthesis and code generation of real-time embedded software.

As an example, a real-time embedded system running on a microprocessor in a wireless sensor network node has two real-time periodic tasks $r_1$ and $r_2$. Task $r_1$ has three subtasks $t_1$, $t_2$, and $t_3$, respectively, for sending data, receiving data, and sending acknowledgement. Task $r_2$ has three subtasks $t_4$, $t_5$, and $t_6$ for moving the sensor position along the three axes, respectively. Subtasks $t_1$, $t_2$, and $t_3$ all request the same networking device $k_1$ and subtasks $t_4$, $t_5$, and $t_6$ request the same motor device $k_2$. It is necessary to schedule all subtasks to ensure their deadlines are satisfied and the power consumption is minimized. We propose a model for such behaviors and an algorithm to obtain time feasible schedules with minimized power consumption.

*This work was supported in part by research project grants from the National Science Council, Taiwan: NSC91-2313-E-194-008, NSC92-2213-E-194-003, NSC92-2218-E-194-009, and NSC93-2213-E-194-002.
Traditionally, Earliest Deadline First and Rate Monotonic Scheduling [1] are used to schedule real-time tasks but some limitations exist. For example, a set of periodic independent tasks is used as system model, whose ability to represent the behavior of an embedded system is limited. A more complex model is thus necessary. We use a model based on extended Petri Nets. In a previous work [2], Real Time Petri Nets (RTPN), were used to specify a set of concurrent periodic tasks containing subtasks that have precedence relationships among them. A subtask has a local deadline and an RTPN has a global deadline. Quasi-dynamic scheduling (QDS) [2] can be used to schedule a set of RTPN with local deadlines, global periods, and global deadlines. Data dependent branch executions are statically composed into different behavior configurations and quasi-statically scheduled. For each decomposed behavior configuration, dynamic scheduling is employed to satisfy local deadlines of all subtasks, precedence constraints, and global deadlines of each task. Here, we enhance QDS with power considerations.

The article is organized as follows. In Sec. 2, we introduce some previous work on scheduling and low-power methodologies related to the synthesis of real-time embedded software. In Sec. 3, we present our system model and give an illustration example. In Sec. 4, we present our scheduling algorithm. In Sec. 5, two examples are presented to show the performance of our algorithm. In the last section, we conclude this article.

2. Related Work

Compiler-based techniques [3, 4] have been proposed to reduce power consumption of embedded software. Voltage and frequency variable processors are also used to reduce power consumption of embedded systems through task scheduling algorithms [5, 6]. Further, device scheduling is also one of the primary techniques to reduce power consumption. Power management performed by an OS is commonly known as dynamic power management. A set of tasks is scheduled at run time such that their time constraints and power reductions are met. Predictive and stochastic models [7, 8] are often applied for I/O device dynamic power management. They are not accurate enough for a hard real-time embedded system. On-line device scheduling [9, 10] is popular but some disadvantages still exist. One of them is that the scheduler needs time to determine the schedule of a huge set of tasks. Off-line device scheduling has not been discussed as much as the methods mentioned previously. An off-line device scheduling method generates a pre-scheduled task sequence so that the time constraints are met completely. Swaminatham [11] proposed an off-line device scheduling technique for task graph models. A reachability tree is generated to find all schedules and the one with lowest power consumption is output.

Here, we propose a quasi-dynamic device scheduling technique that uses quasi-dynamic scheduling instead of the earliest deadline first scheduling as used in most previous works. Some advantages compared to previous works include the simultaneous analysis of multiple resources, namely time, memory, and power, complex system model with finer granularity, and the generation of schedules with lower power consumptions.
3. System Model and Target Problem

To model the complex behaviors of low power real-time embedded software, we propose Power-Aware Real-Time Petri Net (PARTPN) as a system model.

**Definition 1.** Power-Aware Real-Time Petri Nets (PARTPN)

A Power-Aware Real-Time Petri Net is a 7-tuple \((P, T, F, M_0, T, p, d)\), where \(P\) is a finite set of places, \(T\) is a finite set of transitions, \(P \cup T \neq \emptyset\), \(P \cap T = \emptyset\), \(F: (P \times T) \cup (T \times P) \rightarrow N\) is a weighted flow relation between places and transitions, represented by arcs, \(M_0: P \rightarrow N\) is the initial marking (assignment of tokens to places), \(\tau: T \rightarrow N \times (N \cup \infty) \times K\), i.e., \(\tau(t) = (\alpha, \beta, L)\), where \(t \in T\), \(\alpha\) is the transition execution time (WCET), \(\beta\) is the local deadline of transition \(t\), and \(L \subseteq K\) is the list of devices required for the execution of the piece of code represented by transition \(t\). \(K\) is the set of all system devices that can be scheduled. We will use the abbreviations \(\tau_{\alpha}(t)\), \(\tau_{\beta}(t)\), and \(\tau_{L}(t)\) to denote the transition execution time, local deadline, and requested device set, respectively, \(p \in N\) is the global period, and \(d \in N\) is the global deadline of the PARTPN task model.

A set of two PARTPNs modeling the sensor network node example introduced in Section 1, is presented in Fig. 1. The attributes of two schedulable I/O devices are as shown in Table 1.

**Table 1. Devices used in Sensor Network Node.**

<table>
<thead>
<tr>
<th>(k_i)</th>
<th>(\delta_{\alpha})</th>
<th>(\delta_{\beta})</th>
<th>(\delta_{\alpha_p})</th>
<th>(\delta_{\alpha_d})</th>
<th>(\delta_{\alpha_{\infty}})</th>
<th>(\delta_{\alpha_d})</th>
</tr>
</thead>
<tbody>
<tr>
<td>(k_1)</td>
<td>5</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>(k_2)</td>
<td>5</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Fig.1. PARTPN Models of Sensor Network Node.

The target problem we would like to solve is to generate a piece of software code which meets the requirements described in the PARTPN system model and satisfies all user-given constraints, including global and local deadlines, available global memory, and upper-bound on global energy usage.

4. Low-Power Quasi-Dynamic Scheduling

To solve the above real-time embedded software synthesis problem, a Low-Power Quasi-Dynamic Scheduling (LQS) method is proposed. Similar to QDS and other static scheduling techniques [11], LQS also uses a reachability tree constructed in a depth-first search manner from the given set of PARTPN models. The front-end processing of LQS is similar to QDS. The schedules that satisfy all non-time constraints are called Extended Quasi-Static Schedules (EQSS) [12]. QDS generates an EQSS schedule that has the minimum total schedule time, whereas LQS generates an EQSS schedule that has the minimum total power consumption. Basic power calculations and tree pruning rules to efficiently find feasible LQS schedules are given in the rest of this section.
For each tree node, the system resource usages, including total time, memory, and power, must be calculated. The initial marking (root node) has zero time, memory, and power usages. The time and memory calculations are straightforward. Given a node \( M \), the power usage at a successor node \( M' \) that is obtained after firing a transition \( t \) is as follows.

(i) Initially, all devices are set in working state. The set of working devices \( D_w(M_0) \) is set as \( K \), where \( M_0 \) is the initial marking.

(ii) The maximum device wakeup (reset) time \( \varepsilon(t, M) \) required by a transition \( t \) at marking \( M \) is calculated as follows.

\[
\varepsilon(t, M) = \max_{k \in K} \delta_{\text{wup}}(k) + \delta_{\text{ad}}(k) \times \max\{0, (\tau_\alpha(t) + \varepsilon(t, M) - \delta_{\text{wup}}(k))\} + \\
\delta_{\text{sd}}(k) \times (\tau_\alpha(t) + \varepsilon(t, M)) + \\
\sum_{k \in K \setminus D_w(M)} \delta_{\text{sd}}(k) \times (\tau_\alpha(t) + \varepsilon(t, M))
\]

where \( D_R(t, M) \) is the set of devices required to be working (i.e., either kept working or newly awakened) for firing \( t \) at marking \( M \), as defined later in this section.

Some rules to decide which transition should be fired first and what devices are to be awakened are proposed and implemented in LQS as follows.

(i) Before generating a successor node by firing a transition \( t_1 \) at marking \( M \), schedulability is first checked for all other enabled transitions \( t_2 \) in \( G \), where \( G \) is the set of concurrently enabled transitions corresponding to \( M \). A transition \( t_1 \) in \( G \) is said to be firable if for all other transitions \( t_2 \) in \( G \):

\[
\rho_\beta(t_2) - (\rho_\alpha(t_1)) + \varepsilon(t_1, M) \geq \rho_\alpha(t_2),
\]

where \( \rho_\beta(t_2) \) represents the amount of time left before reaching the local deadline of \( t_2 \), \( \rho_\alpha(t_1) \) represents the amount of execution time left for \( t_1 \) to finish execution.

(ii) When a transition \( t_1 \) is chosen for firing at marking \( M \), the devices that are kept working may be more than the devices, \( \tau_\alpha(t_1) \), required by \( t_1 \). Specifically, the devices required by all transitions \( t_2 \) that satisfy Eq. (4) must be kept working.

\[
\rho_\alpha(t_2) \leq \rho_\beta(t_2) - (\rho_\alpha(t_1)) + \varepsilon(t_1, M) < \rho_\alpha(t_2) + \varepsilon(t_2, M')
\]

where \( M' \) is the marking reached after firing \( t \) from \( M \). The set of devices required to be kept awake when firing a transition \( t \) at marking \( M \) is, thus, \( D_w(t, M') = \tau_\alpha(t) \cup \tau_\alpha(t') \), where \( t' \) satisfies Eq. (4).

5. Application Example

To illustrate LQS, the real-time embedded software in a sensor network node, introduced in Section 1 and modeled by two concurrent PARTPNs in Sec. 3 as shown in Fig. 1 and
Table 1, is used as an example. On application of LQS, a reachability tree is generated as shown in Fig. 2. The details of the reachability tree construction are omitted.

Fourteen schedules are generated, from which the schedule \(<t_1t_2t_3t_4t_5t_6t_7t_8t_9t_{10}>\) has the lowest power consumption of 66 units and time 10 units, so it is selected as the final result. The software code is then synthesized according to this schedule.

For this example, the generated reachability tree has 34 nodes after applying all the proposed pruning techniques. If a complete reachability tree is generated without the proposed pruning techniques, then there will be 69 nodes in the un-pruned reachability tree. There may be much more nodes required when a larger system is synthesized.

To show the benefits of our proposed LQS algorithm, we compare it with the Energy-Optimal Device Scheduler (EDS), which tries to find a low-power schedule by constructing a reachability tree. The comparison between LQS and EDS is shown in Table 1, where we see LQS being superior to EDS in terms of smaller sizes of the reachability trees and resulting schedules with lower power consumptions.

<table>
<thead>
<tr>
<th>Schedule</th>
<th>Time</th>
<th>Memory</th>
<th>Power</th>
<th>Nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>LQS (&lt;t_1t_2t_3t_4t_5t_6t_7t_8t_9t_{10}&gt;)</td>
<td>10</td>
<td>2</td>
<td>66</td>
<td>34</td>
</tr>
<tr>
<td>EDS (&lt;t_1t_2t_3t_4t_5t_6t_7t_8t_9t_{10}&gt;)</td>
<td>14</td>
<td>N/A</td>
<td>78</td>
<td>371</td>
</tr>
</tbody>
</table>

The other example, we applied LQS to, was the Master/Slave role switching in the wireless Bluetooth communication protocol. The procedures for two Bluetooth devices \(A\) and \(B\) to switch their roles are modeled using four PARTPNs. Due to page-limits, the models are omitted here. The role switch must be communicated between the host layer and the host-control/link manager layer, the models of which have, respectively, 4 and 9 untimed EQSS schedules. Thus, there are totally 36 system behaviors for which 36 reachability trees are constructed. It is found that only one of them is LQS schedulable. The generated reachability tree for the schedulable behavior has 34 nodes and six feasible schedules. The schedule with the lowest power consumption requires 24 time
units and consumes 189 power units. Without the proposed pruning techniques, the full reachability tree will have 8191 nodes.

6. Conclusions

We proposed a device-centric low-power scheduling algorithm for real-time embedded software. A model called **Power-Aware Real-Time Petri Net** (PARTPN) was also proposed to describe finer behaviors of an embedded system. Low-Power Quasi-Dynamic Scheduling was then proposed to find a low-power schedule for this model. It ensures that the generated schedule satisfies all local deadlines of subtasks, global deadlines of tasks, memory and power constraints. The schedule also has minimum energy consumption. In comparison with an offline I/O device scheduling method, the efficiency and capability are shown to be more superior.

References