A Group-Based Energy-Efficient Dual Priority Scheduling for Real-Time Embedded Systems

Yongqi Ge and Rui Liu

1 School of Information Engineering, Ningxia University, Yinchuan 750021, China, geyongqi@nxu.edu.cn
2 School of Agriculture, Ningxia University, Yinchuan 750021, China
* Correspondence: liurui2004_nxu@163.com

Received: 27 February 2020; Accepted: 30 March 2020; Published: 1 April 2020

Abstract: As the limitation of energy consumption of real-time embedded systems becomes more and more strict, it has been difficult to ignore the time overhead and energy consumption of context switches for fixed-priority tasks with preemption scheduling (FPP) in multitasking environments. In addition, the scheduling for different types of tasks may disrupt each other and affect system reliability. A group-based energy-efficient dual priority scheduling (GEDP) is proposed in this paper. The GEDP isolates different types of tasks to avoid the disruption. Furthermore, it also reduces context switches effectively, thus decreasing system energy consumption. As many studies ignored the context switches’ overhead in the worst-case response time (WCRT) model, and it will affect the accuracy of WCRT, thereby affecting the system schedulability. Consequently, the WCRT model is improved based on considering context switches’ overhead. The GEDP is designed and implemented in Linux, and the time overhead and energy consumption of context switches is compared in different situations with GEDP and FPP. The experimental results show that GEDP can reduce context switches by about 1% and decrease energy consumption by about 0.6% for given tasks.

Keywords: energy efficient; worst-case response time; context switches; real-time

1. Introduction

Real-time embedded systems are widely used in high-reliability application domains to ensure reliable execution of mission-critical tasks. These domains include aerospace, unmanned aerial vehicles, automobiles, and so on [1–3]. With the development of computer technology, such systems are becoming more intelligent, and the demand for long time running is getting stronger. For example, most real-time embedded systems in the unmanned aerial area are battery powered. The battery life determines the system running time. However, the battery capacity is limited. To ensure the reliability and long time running of real-time embedded system, the research is working on ways to reduce energy consumption, which can reduce system energy demands and improve the reliability and stability of energy supply [4–6].

With the ongoing focus on the energy consumption of real-time embedded systems, how to minimize the energy consumption under the premise of guaranteeing system schedulability has been gaining increased attention in recent years [7,8]. At present, the energy optimized technology for embedded systems is at different levels, including the circuit level, system level, storage level, and compile level [9]. At the system level, there are two primary energy-efficient approaches to address the energy consumption optimization: dynamic voltage and frequency scaling (DVFS) [10–12] and dynamic power management (DPM) [13,14]. These approaches are mainly the combination of task scheduling, DVFS, and DPM. According to the execution time, deadline, and period of the task, the processor frequency is dynamically adjusted under the premise of ensuring the schedulability of the task. The system energy consumption can be reduced through decreasing the processor...
frequency. However, this is at the cost of increasing the task response time. Therefore, the energy optimization for real-time systems is the trade-off between energy consumption and the task response time. The real-time systems have high reliability and hard real-time demands. Most of the above energy optimized technologies adjust the voltage and frequency of hardware resources. However, for some high-reliability real-time embedded systems, such as lunar rovers, the primary design objective is to meet the time constraint. The lunar cycle consists of 27 Earth days including the lunar day (about 13 Earth days) the lunar night (about 14 Earth days) \[15\]. During the lunar day, the moon is illuminated by the Sun. The lunar rover is powered by solar panels, and it has abundant energy to perform scientific exploration missions for 14 Earth days. However, during lunar night, the lunar rover enters the hibernation state \[16\]. The lunar rover is powered by a battery with limited capacity to keep the core systems warm enough and for processing scientific data. DVFS can save much energy by lowering the processor speed, which results in extending the task execution time. The lunar rover has hard real-time demands for scientific data processing. The voltage regulation technology will cause certain reliability problems that lead to scientific task failure. Therefore, the above energy-saving technology is difficult to adapt to the demands of high-reliability real-time systems.

For task scheduling technology, context switching is an important factor affecting the system energy consumption. With the limitation of the energy consumption of real-time embedded systems becoming stricter, it has been difficult to ignore the context switches’ overhead. The preemption threshold scheduling is an effective way to improve processor utilization and reduce context switches. However, its worst-case response time (WCRT) model ignores the context switches \[17\], thereby affecting the accuracy of WCRT. This would delay the execution of tasks and miss the deadline, thus making the system unable to be scheduled. Furthermore, the preemption threshold scheduling is running under a simulation environment, and it does not have adequate physical experiments to analyze the effect of context switches’ overhead. In addition, the scheduling for different types of tasks may cause a disruption and affect system reliability. Due to the above reasons, we present a group-based energy-efficient dual priority scheduling (GEDP) based on preemption thresholds. The contributions of this paper are as follows.

1. The GEDP scheduling algorithm is presented for isolating different types of tasks to avoid disruption and decreasing energy consumption through reducing context switches in a real-time embedded system.
2. The WCRT model is improved based on considering context switches’ overhead, and the improved WCRT model can enhance the accuracy of WCRT.
3. To verify the effectiveness of the GEDP, the context switches’ statistics methods and the tasks’ energy consumption analysis methods are discussed. Furthermore, a physical experiment is set up on the Linux 2.6.32.5 kernel, which is modified to support GEDP, and four benchmarks are taken as testing applications.

The rest of the paper is organized as follows. The related works are summarized in Section 2. Section 3 presents the GEDP scheduling for real-time embedded systems, and the GEDP algorithm is presented based on grouping rules. Section 4 discusses the schedulability of GEDP based on WCRT analysis and the threshold priority assignment method. The physical experiments are included and discussed in Section 5. Finally, the main conclusions and future work are highlighted in Section 6.

2. Related Work

System-level energy consumption is divided into dynamic energy consumption and static energy consumption \[18\]. Since DVFS can reduce the dynamic energy consumption of the processor effectively, a large number of voltage regulation scheduling techniques are used to solve the energy consumption problems of the embedded systems. Although DVFS is an effective method to save energy, it will prolong the task execution time. For high-reliability embedded systems, the real-time performance of the task is critical to the system security. Prolonging the execution time may increase the chance of
the task missing the deadline. This will increase the probability of transient errors, thus affecting the system reliability [19,20]. Static energy consumption is caused by leakage current. When the processor frequency decreases, the threshold voltage of the CMOS circuit also becomes smaller, which leads to an increase in subthreshold leakage current [21]. When the CPU production process is 180 nm, the static energy consumption caused by the leakage current is negligible. When the CPU production process is 130 nm, the static energy consumption accounts for 10% to 30% of the system energy consumption. When the CPU production process is 70 nm, static energy consumption accounts for 50% of system energy consumption. When the CPU production process is less than 70 nm, the static energy consumption is comparable to dynamic energy consumption. Furthermore, the DVFS method reduces the dynamic energy consumption by decreasing the processor frequency, which leads to an increase in the tasks’ execution time and also increases the static energy consumption. Therefore, with the rapid development of the chip production process, the energy consumption of leakage current increase, and the space of using DVFS to reduce system energy consumption becomes smaller.

At present, many studies ignore the context switches’ energy overhead. Actually, the context switch is an important factor affecting the system energy consumption in multitasking systems. Every time a process is done on the CPU, a context switch occurs, which adds overhead to the process execution time [22]. Each context switch needs to save the current process state and restore another process state. The context switches may add up to a significant delay, and thus affect schedulability. The total time overhead of context switches would result in significant energy overhead. Many studies attribute context switches to scheduling, but context switches and scheduling are different. The processor has two operation modes in embedded systems: kernel mode and user mode. Ouni and Belleudy [23] pointed out that the scheduler switches from user mode to kernel mode through system calls. This mechanism is called the mode switch. Unlike the mode switch, the context switch refers to the process of a processor switching from one process to another, including direct overhead and indirect overhead. Direct overhead is caused by the operations that the processor must perform for process switching. Indirect overhead is the virtual address translation of process switching, which will disturb the translation lookaside buffer (TLB) [24]. At the same time, they pointed out three basic services of embedded systems: scheduling, context switch, and inter-process communication. Energy measurement experiments were carried out on an OMAP35x evaluation module (EVM) board, and the measurement of context switch time overhead and energy consumption was given. This method is mainly used to measure the average energy consumption of context switches. Based on the ARM platform, David and Carlyle [25] measured the direct and indirect overhead of the context switch on the Linux operating system. Acquaviva and Benini et al. [26] proposed a method to characterize the system energy consumption. They measured the energy consumption of the real-time system based on the prototype of the wearable device and then discussed the system energy consumption at the kernel level and the I/O driver level. Finally, the energy consumption and performance of service and context switches were analyzed. Previous studies have shown that the indirect overhead is much larger than the direct overhead. These methods focus on the characterization of the energy measurement of context switches and do not take into account the impact of scheduling policies on the energy consumption of context switches.

In view of the impact of context switches’ energy overhead, some algorithms to reduce context switches have been proposed. Behera and Mohanty [27] proposed a dynamic quantum with re-adjusted round robin algorithm to reduce context switches. Raveendran and Prasad et al. [28] proposed the variant algorithms of Rate Monotonic (RM) and Earliest Deadline First (EDF); Wang and Saksena [17] proposed a fixed priority scheduling algorithm based on a preemption threshold. Preemptive threshold scheduling can improve system utilization under the premise of maintaining system schedulability and can reduce context switches by preemptive scheduling. Then, the threshold allocation algorithm was improved in [29]; the time complexity was reduced from O(n!) to O(n^2). Experiments showed that the preemption threshold scheduling could improve the processor utilization by 15–20% compared with the preemptive scheduling. However, these algorithms are usually assumed to ignore the context
The priorities of application tasks are in the range of $[0, \text{high}]$. System tasks provide services for the operating system, such as clock interrupts, schedulers, and drivers. The priorities of system tasks are in the range of $[0, \text{low}]$. Application tasks refer to the applications that provide specific functions. In other words, when the task $\tau_i$ gets the CPU, its normal priority is raised up to its threshold priority, and it maintains priority until it finishes the execution.

3. GEDP Scheduling

3.1. Scheduling Model

The scheduling model is based on the threshold preemption. A real-time embedded system is specified using a set of independent periodic tasks, $P = \{\tau_1, \tau_2, \tau_3, \cdots, \tau_n\}$. Each task is characterized by a three-tuple $\tau_i = (C_i, T_i, D_i)$, where $C_i$ is the worst-case execution time, $T_i$ is the task period, and $D_i$ is the deadline. Each task will be given a normal priority $\pi_i \in [1, \cdots, n]$ and a threshold priority $\gamma_i \in [1, \pi_i]$.

3.2. GEDP Algorithm

For grouping tasks, tasks are sorted from the highest priority to the lowest priority. In order to avoid interference between system-critical tasks and other tasks, tasks are divided into system tasks and application tasks. The task priority range is $[0, \text{high}]$. System tasks provide system services for the operating system, such as clock interrupts, schedulers, and drivers. The priorities of system tasks are in the range of $[0, \text{low}]$. Application tasks refer to the applications that provide specific functions. The priorities of application tasks are in the range of $[\text{low}, \text{high}]$. This article uses the task flag $T\text{flag}$ to distinguish system tasks from application tasks. If $T\text{flag} = 0$, it indicates that the task is a system task. If $T\text{flag} = 1$, it indicates that the task is an application task. The system tasks are divided into one group and use the default scheduling policy. According to the definition based on $L_i$-level application tasks, the application tasks are divided into several groups. The definition of the $L_i$-level application task is given below.

**Definition 1.** The task priority subset $L_i \subseteq [\text{low}, \text{high}]$, $L_i$ cannot overlap. If the priority of task $\tau_i$ satisfies condition $\pi_i \in L_i$, the task $\tau_i$ is a $L_i$-level application task.
The grouping rules of GEDP are presented as follows:

1. **Grouping rule of system tasks**: If the priority of task \( \tau_i \) satisfies conditions \( \pi_i \in [0, \text{low}) \) and \( T_{flag} = 0 \), then the task is a system task. Add task \( \tau_i \) into the system task group, denoted as \( G_0 \).

2. **Grouping rule of application tasks**: If the priority of the task \( \tau_i \) satisfies conditions \( \pi_i \in L_i \) and \( T_{flag} = 1 \), then the task \( \tau_i \) is an application task. Add task \( \tau_i \) into the application task group, denoted as \( G_i \).

According to the grouping rules, all system tasks are grouped into one group. The number of system task groups is one. The priority range of application tasks is \([\text{low}, \text{high}]\). In extreme cases, if \( L_i = [\text{low}, \text{high}] \), all application tasks are divided into one group. That means the application tasks can be divided into one group at least. If the application tasks corresponding to each priority level in \([\text{low}, \text{high}]\) are divided into a group, the application tasks can be divided into a maximum of \( \text{high} - \text{low} + 1 \) groups. The tasks’ group number of the entire system is equal to the sum of the system tasks’ group and the application tasks. Therefore, the total system group is between \([2, \text{high} - \text{low} + 2]\).

Suppose there are \( n \) real-time tasks in the operating system, such as \( \tau_1, \tau_1, \cdots, \tau_n \). They are divided into two task groups \( G_0 \) and \( G_1 \), where tasks \( \tau_1, \tau_2, \cdots, \tau_k \) belong to the system task group \( G_0 \) and tasks \( \tau_{k+1}, \tau_{k+2}, \cdots, \tau_n \) belong to the application task group \( G_1 \). For the system task group \( G_0 \), fixed-priority preemption scheduling is used, and the normal priority is equal to the threshold priority. For the application task group \( G_1 \), the preemption threshold scheduling is taken.

Based on the above method, the GEDP algorithm designed in this paper is shown in Algorithm 1.

### Algorithm 1: GEDP algorithm.

```plaintext
1   t ← 0;
2   while true do
3       W ← \( \tau_1, \tau_2, \cdots, \tau_n \); /* The ready tasks queue is ordered by fixed priority */
4       if \( W \neq \phi \) then
5           next ← \( \tau_1 \); /* Take the highest priority task \( \tau_1 \) in the task set \( W \) */
6           // Group tasks based on priority and \( T_{flag} \)
7           if next \( \in [0, \text{low}] \) and \( T_{flag} = 0 \) then
8               \( G_0 \leftarrow \) next; /* Add tasks to group \( G_0 \) */
9           else
10              \( G_1 \leftarrow \) next; /* Add tasks to group \( G_1 \) */
11          end if
12          // Scheduler entry
13          if next \( \in G_0 \) then
14              Fixed-priority preemption scheduling;
15          else
16              Preemption threshold scheduling;
17          end if
18       end if
19       t ← t + 1;
20   end while
```

4. **Schedulability Analysis and Threshold Priority Assignment**

4.1. **WCRT Analysis**

The schedulability is determined by the WCRT of each instance of tasks. This paper analyzed the WCRT based on considering the voluntary context switches and the involuntary context switches. The response time of a task typically consists of three parts:
(1) The execution time of the task: The time overhead of voluntary context switches should be considered. Every time the task gives up the CPU, a voluntary context switch occurs.
(2) Interference from other higher priority tasks: In this case, the time overhead of involuntary context switches should be considered. Each time a task is preempted, the preemption task will be switched first and then switches back to the current task. In this case, context switches occur twice.
(3) Blocking from lower priority tasks caused by the threshold priority: If the lower priority task is running, the objective task cannot be preempted due to higher threshold priority, and this will result in blocking time.

The upper bound of blocking time for task $\tau_i$ is shown as Equation (1), where $B(\tau_i)$ is the blocking time from lower priority tasks and $C_j$ is the execution time of blocking tasks.

$$B(\tau_i) = \max_j \{ C_j : \gamma_j \leq \pi_i < \pi_j \} \quad (1)$$

The busy period of task $\tau_i$ begins from the critical time. All instances of higher priority tasks arrive simultaneously, and the task contributed maximum blocking time is executed before the critical time. In addition, in order to get the WCRT, all the tasks come at their maximum rate. The Level-I busy period $W(\tau_i)$ can be computed by Equation (2), and $C_{nv}$ is the maximum time overhead of involuntary context switches.

$$W(\tau_i) = B(\tau_i) + \sum_{\forall j, \pi_j < \pi_i} \left( \left\lfloor \frac{W(\tau_i)}{T_j} \right\rfloor \right) \cdot (C_j + 2C_{nv}) \quad (2)$$

All higher priority task instances that come before the start time and any earlier instances of tasks should be finished before the start time of the $q$th instance of $\tau_i$. Therefore, the start time $S_i(q)$ of the $q$th instance of $\tau_i$ can be computed iteratively using Equation (3), and $C_v$ is the maximum time overhead of voluntary context switch.

$$S_i(q) = B(\tau_i) + (q - 1) \cdot (C_i + C_v) + \sum_{\forall j, \pi_j < \pi_i} \left( 1 + \left\lfloor \frac{S_i(q)}{T_j} \right\rfloor \right) \cdot (C_j + 2C_{nv}) \quad (3)$$

During the execution of task $\tau_i$, only the task that has higher priority than $\gamma_i$ can obtain the processor before the task $\tau_i$ finishes. The worst-case finish time $F_i(q)$ can be computed using Equation (4).

$$F_i(q) = S_i(q) + C_i + C_v + \sum_{\forall j, \pi_j < \pi_i} \left( \left\lfloor \frac{F_i(q)}{T_j} \right\rfloor - \left( 1 + \left\lfloor \frac{S_i(q)}{T_j} \right\rfloor \right) \right) \cdot (C_j + 2C_{nv}) \quad (4)$$

The WCRT of task $\tau_i$ is equal to the max response time of all instances in the busy period interval. The WCRT can be calculated by Equation (5).

$$R_i = \max_{q \in \{1, 2, \ldots, m\}} (F_i(q) - (q - 1) \cdot T_i) \quad (5)$$

In order to analyze the effect of the improved WCRT model, we take the time overhead of context switching as 100 ms. We give a set of tasks including four tasks, as shown in Table 1, where $W_i$ is the worst-case response time of preemption threshold scheduling. $IW_i$ is the improved worst-case response time based on the considering context switches. The task set can be scheduled in $W_i$, but cannot be scheduled in $IW_i$. Because the $IW_i$ takes into account the overheads of context switches, the WCRT would be increased, leading the missed deadline of $\tau_1$ and $\tau_3$. Therefore, the improved WCRT model can enhance the accuracy due to considering context switches’ time overhead.
Table 1. Tasks’ attributes.

| Tasks | $C_i$ | $T_i$ | $D_i$ | $\pi_i$ | $W_i$ | $IW_i$ |
|-------|-------|-------|-------|--------|-------|-------|
| $\tau_1$ | 6     | 32    | 17    | 10     | 17    | 18    |
| $\tau_2$ | 4     | 48    | 32    | 21     | 12    | 17    |
| $\tau_3$ | 2     | 48    | 8     | 20     | 8     | 13    |
| $\tau_4$ | 5     | 40    | 17    | 43     | 6     | 11    |

4.2. Threshold Priority Assignment

The threshold priority assignment algorithm based on the preemption threshold [17] is shown as Algorithm 2. First, initialize the attributes of the tasks, such as $C_i$, $D_i$, and $T_i$. Second, sort the tasks from low to high priority according to the normal priority using the quick sort algorithm. Third, calculate the threshold for each task. The threshold priority is initialized to normal priority, and then, calculate the WCRT under the current threshold. If the WCRT exceeds the deadline, increase the threshold priority. If the threshold priority is higher than the highest priority, the task set cannot be scheduled. Otherwise, the WCRT is recalculated cyclically.

Algorithm 2: Threshold priority assignment.

1. Initialize task properties;
2. Use the quick sort algorithm to sort tasks from low to high priority;
3. for $i \leftarrow 0$ to taskNum do
4.   $\gamma_i = \pi_i$; /* Initial thresholds’ priorities */
5.   // Calculate WCRT at the current threshold
6.   Compute $R_i$ by Formulas (1)–(5);
7.   while $R_i > D_i$ do /* WCRT exceeds deadline */
8.     $\gamma_i--; /* Increase threshold priority */$
9.     if $\gamma_i < 0$ then
10.    return false;
11.   end if
12.   recalculate $R_i$;
13. end while
14. end for
15. return true;

4.3. Schedulability Analysis Tool

In order to provide a simple way to compute WCRT and threshold priority, we developed an open-source tool called the schedulability analysis tool (SAT) based on Debian 6.0 under the QT integrated development environment. The source code for SAT is available on GitHub at https://github.com/geyongqi/SAT. In the current version, the main functions of SAT are WCRT analysis, threshold calculation, and task running statistics. The following introduces the interface, functions, and operation demonstration of SAT briefly. The main GUI is shown in Figure 1.
SAT includes eight functional modules as shown in red numbers in Figure 1. Module 1 is the menu bar, which can mainly calculate the WCRT and analyze the schedulability. Module 2 is the toolbar corresponding to the menu bar. Module 3 is the task attribute input box. Module 4 is a task list box that displays the task attributes added to the tool. Module 5 is a threshold calculating the switch and determines whether the thresholds need to be calculated or not. Module 6 is a threshold priorities display box, which displays the threshold priorities of each task. Module 7 is a task running statistics display box, which displays the task context switches' time overhead and other information. It captures the statistics of a specific task at runtime, including the task ID, the total number of context switches, the number of voluntary and involuntary context switches, etc. Module 8 is the task running information, showing whether the task is running or not. It can capture the task runtime and resource occupation information, including task ID, parent task ID, task name, normal priority, memory usage, and current running status. The following is the operation demonstration of SAT.

Step 1: Enter the attributes of the task;
Step 2: Click the “Add Task =>” button to add the task to the task list. If all tasks have been added, continue to Step 3; otherwise, repeat Step 1 to continue adding tasks;
Step 3: If you are doing a non-threshold experiment, go to Step 4 directly. If you are doing a threshold experiment, check the “Open Threshold” check-box and click the “Comp Threshold =>” button to calculate the threshold priorities;
Step 4: If the threshold calculation fails, click the “Clear” button to clear all the information and return to the first step to restart the operation; otherwise, click the “RUN” button to run all tasks;
Step 5: Enter the task name, and click the “START” button to capture the statistical information when the task is running;
Step 6: Click the “START” button to capture the tasks running information in the system. Repeat Step 5 and Step 6 to get the running information of the tasks at different times.

![Figure 1. Worst-case response time (WCRT) analysis tool.](image-url)
5. Results and Discussion

5.1. Experiment Setup

The time overhead and energy overhead of context switches is mainly related to the saving and restoring registers, cleaning the CPU pipeline and cache disruption. It is not directly related to other system peripherals. We modified the Linux 2.6.32.5 kernel to support threshold priority and took it as the runtime environment for real-time tasks. The context switches statistics function was implemented in the user space of Linux through netlink sockets. Time acquisition functions were inserted before and after context switches to calculate the time overhead. Four benchmarks were designed as four tasks running in the user space [30], including Linpack_bench, Memory_test, Whetstone, and Mxm. The attributes of the benchmarks are list in Table 2. The energy measurement was based on the Chroma 66200 Digital Power Meter, which was connected to the testing host (Dell Vostro 3450 installed Debian 6.0).

Table 2. The attributes of four benchmarks

| Tasks       | $C_i$ | $T_i$ | $D_i$ | $\pi_i$ | $\gamma_i$ | $W_i$ |
|-------------|------|------|------|--------|----------|------|
| Linpack_bench | 34   | 165  | 160  | 53     | 53       | 155  |
| Memory_test  | 60   | 245  | 243  | 70     | 53       | 241  |
| Whetstone    | 26   | 190  | 185  | 62     | 45       | 181  |
| Mxm          | 59   | 160  | 100  | 45     | 45       | 86   |

According to WCRT analysis, we needed to get the maximum time overhead per context switch for each benchmark to compute the threshold priorities. The context switches’ time overhead was closely related to the internal environment of the operating system. Different tasks had different context switch time overhead. Even if the same task was switched at different frequencies, the context switch time overhead may be different. Therefore, we designed the experiment to run four benchmarks according to the attributes in Table 2 for about 30 min. The context switch time overhead of the four benchmarks is shown in Figures 2–5. Memory_test had 96 context switches, and the context switch time overhead was between $[0.025, 108]$ µs. Linpack_bench had 24 context switches, and the context switch time overhead was between $[0.178, 16.6]$ µs; Mxm had 28 context switches, and the context switch time overhead was between $[0.023, 8.88]$ µs. Whetstone had 30 context switches, and the context switch time overhead was between $[0.023, 24.5]$ µs.

Figure 2. Context switch time overhead of Memory_test.
5.2. Experiment 1: The Number of Context Switches Analysis

In this experiment, due to the influence of the physical experiment environment, the number of context switches in each experiment was different. In order to analyze the trend of the number of context switches comprehensively, we performed ten experiments, and the experimental data are shown in Table 3.
Table 3. The ten experiments’ data. FPP, fixed-priority tasks with preemption scheduling; GEDP, group-based energy-efficient dual priority scheduling.

| Times | Linpack_bench | Memory_test | Whetstone | Mxm |
|-------|---------------|-------------|-----------|-----|
|       | FPP | GEDP | FPP | GEDP | FPP | GEDP | FPP | GEDP |
| T1    | 798 | 803 | 592 | 582 | 580 | 569 | 1496 | 1509 |
| T2    | 802 | 793 | 594 | 579 | 576 | 561 | 1511 | 1504 |
| T3    | 807 | 801 | 587 | 576 | 567 | 566 | 1532 | 1501 |
| T4    | 797 | 797 | 600 | 578 | 567 | 566 | 1532 | 1501 |
| T5    | 804 | 806 | 594 | 588 | 559 | 565 | 1539 | 1509 |
| T6    | 801 | 822 | 576 | 573 | 569 | 527 | 1518 | 1524 |
| T7    | 814 | 803 | 585 | 576 | 561 | 521 | 1533 | 1539 |
| T8    | 812 | 804 | 592 | 579 | 587 | 570 | 1592 | 1545 |
| T9    | 802 | 804 | 582 | 580 | 575 | 570 | 1538 | 1530 |
| T10   | 796 | 800 | 592 | 586 | 571 | 528 | 1522 | 1527 |

Take the experimental data T2 as an example: we compared the voluntary and involuntary context switches under fixed-priority tasks with preemption scheduling (FPP) and GEDP. The trend of context switches of the four benchmarks is shown in Figures 6–13. The number of voluntary context switches of Linpack_bench changed little, and the number of involuntary context switches under GEDP was less than FPP. The number of voluntary context switches of Memory_test and Whetstone did not change, and the trend of involuntary context switches decreased significantly. The number of voluntary context switches for Mxm did not change, and the number of involuntary context switches had almost no change at all. According to the task attributes in Table 2, since the threshold priority was to prevent preemption of high-priority tasks, the GEDP mainly affected the involuntary context switches and did not affect voluntary context switches. The normal priority and threshold priority of Linpack_bench were the same, and the involuntary context switches should be unchanged. Memory_test had a normal priority of 70 and a threshold priority of 53 to prevent Linpack_bench and Whetstone from preemption. Therefore, under GEDP, the number of involuntary context switches of Memory_test should be reduced. Whetstone had a normal priority of 62 and a threshold priority of 45, which could prevent Mxm from preemption. Therefore, under GEDP, the number of involuntary context switches of Whetstone should be reduced. The normal priority of Mxm was consistent with the threshold priority, and the number of context switches should not change.

![Figure 6. Voluntary context switches of Linpack_bench.](image-url)
Figure 7. Involuntary context switches of Linpack_bench.

Figure 8. Voluntary context switches of Memory_test.

Figure 9. Involuntary context switches of Memory_test.
Figure 10. Voluntary context switches of Whetstone.

Figure 11. Involuntary context switches of Whetstone.

Figure 12. Voluntary context switches of Mxm.
As discussed above, the number of context switches for each benchmark was related to its normal priority and threshold priority. Normal priority affects voluntary context switches, and threshold priority affects involuntary context switches. Even if each benchmark kept its own normal priority and threshold priority in multiple experiments, the number of context switches would be different. Combining the experimental results with the above theoretical analysis, the experimental results were basically consistent with the theoretical analysis.

In order to quantify the gains of GEDP relative to FPP on context switching, we defined four analysis indicators. $R_c$ is the context switch reduction rate of a task in one experiment, shown as Equation (6); $R_v$ is the average context switch reduction rate of a task in multiple experiments, shown as Equation (7); $R_t$ is the total context switch reduction rate for all tasks in one experiment, shown as Equation (8); $R_{vt}$ is the average context switch reduction rate for all tasks in multiple experiments, shown as Equation (9).

\[
R_c(\tau_i) = \left[1 - \frac{C_{GEDP}(\tau_i)}{C_{FPP}(\tau_i)}\right] \times 100\% \tag{6}
\]

\[
R_v(\tau_i) = \left[\frac{\sum_{j=1}^{n} R_c(\tau_i)}{n}\right] \times 100\% \tag{7}
\]

\[
R_t(i) = \left[1 - \frac{T_{GEDP}(i)}{T_{FPP}(i)}\right] \times 100\% \tag{8}
\]

\[
R_{vt}(n) = \left[\frac{\sum_{i=1}^{n} R_t(i)}{n}\right] \times 100\% \tag{9}
\]

$C_{FPP}(\tau_i)$ is the number of context switches for the task $\tau_i$ under FPP in one experiment. $C_{GEDP}(\tau_i)$ is the number of context switches for task $\tau_i$ under GEDP in one experiment. $T_{FPP}(i)$ is the number of context switches for all tasks under FPP in the $i$th experiment. $T_{GEDP}(i)$ is the number of context switches for all tasks under GEDP in the $i$th experiment, and $n$ is the number of experiments. The $R_c$ of Memory_test was between 0.3% and 2.5%. The $R_v$ of Memory_test was about 1.3%. These findings showed that the context switches of Memory_test were reduced significantly. The $R_c$ of Whetstone was between $-1.1\%$ and 7.5%. The $R_v$ of Whetstone was about 2.9%. The results indicated that the context switches of Whetstone were significantly reduced. The $R_c$ and $R_v$ of Linpack_bench and Mxm were relatively small, and the changes were not obvious. From the overall analysis of the four benchmarks, $R_I$ was between 0.1% and 2.1%. The average reduction rate of the ten experiments was approximately...
Based on the above analysis, it can be seen that the number of context switches of Memory_test and Whetstone was significantly reduced due to the threshold priority.

5.3. Experiment 2: System Energy Consumption Analysis

In this experiment, we studied the effect of GEDP compared with FPP on energy consumption. Experiments were performed on GEDP and FPP, respectively. Each experiment lasted 10 hours, and the energy consumption was sampled per 1000 seconds. The energy consumption and context switches under GEDP and FPP are shown in Table 4. For FPP, the energy consumption in 10 hours was about 964,448.50 J, and the number of context switches was 38,771. For GEDP, the energy consumption was about 958,537.53 J, and the number of context switches was 38,386. The experiment results showed that:

(1) Compared with FPP, the GEDP reduced energy consumption by 5910.98 J and reduced context switches 385 times. The energy consumption and the number of context switching under the GEDP were significantly reduced compared to FPP. This showed that GEDP was efficient to reduce context switches to optimize the system energy consumption.

(2) The savings of energy consumption tended to increase as the number of context switches decreased. This was because GEDP reduced more context switches, thus producing cumulative effects on saving energy.

(3) For given benchmarks, the experiment results showed that GEDP provided gains (0.6% energy consumption and 1% context switches) over the previous scheduling FPP. It proved that reducing the number of context switches could decrease system energy consumption. However, affected by the physical experiment environment and different task attributes, different benchmarks may have different gains. We plan to study the relationship between benchmarks and gains in future work.

(4) According to the results for total system energy consumption and context switches in Table 4, we could calculate that the average energy overhead of a context switch was about 15 J. The context switches’ energy overhead was related to memory accesses in saving/restoring the task context, as well as due to the additional cache misses resulting from a context switch. The context switch energy overhead was not constant.

### Table 4. System energy consumption comparison.

| Sampling | Context Switches | Energy Consumption (J) |
|----------|------------------|------------------------|
|          | FPP   | GEDP   | FPP    | GEDP   |
| S0       | 1029  | 1026   | 30,278 | 30,222 |
| S4       | 1056  | 1047   | 30,042 | 30,280 |
| S8       | 1049  | 1047   | 26,082 | 25,763 |
| S12      | 1074  | 1079   | 26,143 | 25,958 |
| S16      | 1054  | 1045   | 26,101 | 25,837 |
| S20      | 1087  | 1069   | 26,336 | 26,209 |
| S24      | 1094  | 1085   | 26,295 | 26,100 |
| S28      | 1095  | 1098   | 26,788 | 26,454 |
| S32      | 1072  | 1058   | 26,223 | 25,797 |
| S36      | 1100  | 1073   | 26,624 | 25,949 |

6. Conclusions and Future Works

Considering that the time overhead and energy overhead of context switching can affect system reliability and energy consumption, the purpose of this study was to reduce context switches and isolate different types of tasks to avoid disruption. A group-based energy-efficient dual priority scheduling for a real-time embedded system was proposed. In addition, the worst-case response time was improved by considering context switches for analyzing schedulability. The experiment
results demonstrated that the GEDP scheduling ensured efficiencies to decrease the system energy consumption by reducing context switches for real-time embedded systems in this work.

In this paper, the gains included two aspects: the number of context switches and system energy consumption. In terms of the number of context switches, the number of context switches for each benchmark was related to its normal priority and threshold priority. Normal priority affects voluntary context switches and threshold priority affects involuntary context switches. Even if each benchmark kept its own ordinary priority and threshold priority in multiple experiments, the number of context switches would be different. In terms of the system energy consumption, the system energy consumption was directly affected by the number of context switches. The more context switches the scheduling saved, the more the energy decreased. In addition, affected by the physical experiment environment and different task attributes, different benchmarks had different gains. Therefore, further research is still needed to develop more benchmarks to study the correlation between the gains.

The WCRT computation will waste a large number of resources. Because the WCRT in this article was calculated offline, it did not affect the task execution. In general, the real execution time of the task was much shorter than the WCRT, which meant the WCRT was not accurate enough. We will continue to study the highly efficient WCRT computing method or explore other methods. In addition, the GEDP can be extended to other scheduling algorithms and compared with other proactive and reactive real-time scheduling. The SAT tool can be improved to support more scheduling algorithms and benchmarks. Furthermore, possible future work would be generalizing this scheduling to be applicable to other kinds of systems.

Author Contributions: Conceptualization, R.L.; data curation, Y.G.; funding acquisition, Y.G.; methodology, Y.G.; resources, R.L.; software, R.L.; validation, Y.G.; writing, original draft, Y.G.; writing, review and editing, R.L. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported in part by the National Natural Science Foundation of China under Grant No. 61862049, in part by the Ningxia Key Research and Development Project (special talent) under Grant No. 2018BBB04020, in part by the Young Scholar in Western China of the Chinese Academy of Sciences under Grant No. XAB2018AW12, and in part by the Ningxia Higher Education Research Project under Grant No. NGY2018-229 and No. NGY2017031.

Conflicts of Interest: The authors declare no conflict of interest. The funders had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript, or in the decision to publish the results.

References

1. Norouzi, R.; Kosari, A.; Sabour, M.H. Real time estimation of impaired aircraft flight envelope using feedforward neural networks. Aerosp. Sci. Technol. 2019, 90, 434–451.

2. Kato, S.; Tokunaga, S.; Maruyama, Y.; Maeda, S.; Hirabayashi, M.; Kitsukawa, Y.; Monroy, A.; Ando, T.; Fujii, Y.; Azumi, T. Autoware on board: Enabling autonomous vehicles with embedded systems. In Proceedings of the ACM/IEEE 9th International Conference on Cyber-Physical Systems, Porto, Portugal, 11–13 April 2018; pp. 287–296.

3. Ikura, M.; Miyashita, L.; Ishikawa, M. Real-time Landing Gear Control System Based on Adaptive 3D Sensing for Safe Landing of UAV. In Proceedings of the IEEE/SICE International Symposium on System Integration (SII), Honolulu, HI, USA, 12–15 January 2020; pp. 759–764.

4. Malewski, M.; Cowell, D.M.J.; Freear, S. Review of battery powered embedded systems design for mission-critical low-power applications. Int. J. Electr. 2018, 105, 893–909.

5. Xu, H.; Li, R.; Pan, C.; Keqin, L. Minimizing energy consumption with reliability goal on heterogeneous embedded systems. J. Parallel Distrib. Comput. 2019, 127, 44–57.

6. Tzilis, S.; Trancoso, P.; Sourdis, I. Energy-Efficient Runtime Management of Heterogeneous Multicores using Online Projection. ACM Trans. Archit. Code Optim. (TACO) 2019, 15, 1–26.
7. Wägemann, P.; Dietrich, C.; Distler, T.; Ulbrich, P.; Schroder-Preikschat, W. Whole-system worst-case energy-consumption analysis for energy-constrained real-time systems. In Proceedings of the Leibniz International Proceedings in Informatics, Barcelona, Spain, 3–6 June 2018; pp. 24–36.

8. Li, J.; Shu, L.C.; Chen, J.J.; Li, G. Energy-efficient scheduling in nonpreemptive systems with real-time constraints. *IEEE Trans. Syst. Man Cybern. Syst.* 2012, 43, 332–344.

9. Dick, R.P.; Lakshminarayana, G.; Raghunathan, A.; Jha, N.K. Analysis of power dissipation in embedded systems using real-time operating systems. *IEEE Trans. Computer-Aided Des. Integr. Circuit Syst.* 2003, 22, 615–627.

10. Kada, B.; Kalla, H. An Efficient Fault-Tolerant Scheduling Approach with Energy Minimization for Hard Real-Time Embedded Systems. *Cybern. Inf. Technol.* 2019, 19, 45–60.

11. Carvalho, S.A.L.; Cunha, D.C.; Silva-Filho, A.G. Autonomous power management in mobile devices using dynamic frequency scaling and reinforcement learning for energy minimization. *Microprocess. Microsyst.* 2019, 64, 205–220.

12. Ahmed, I.; Zhao, S.; Trescases, O.; Betz, V. Automatic application-specific calibration to enable dynamic voltage scaling in FPGAs. *IEEE Trans. Computer-Aided Des. Integr. Circ. Syst.* 2018, 37, 3095–3108.

13. Zhou, M.; Cheng, L.; Antonio, M.; Wang, X.; Bing, Z.; Ali Nasseri, M.; Huang, K.; Knoll, A. Peak Temperature Minimization for Hard Real-Time Systems Using DVS and DPM. *J. Circuits Syst. Comp.* 2019, 28, 1950102.

14. Khan, H.; Bashir, Q.; Hashmi, M.U. Scheduling based energy optimization technique in multiprocessor embedded systems. In Proceedings of the International Conference on Engineering and Emerging Technologies, Edinburgh, UK, 16–19 July 2019; pp. 1–8.

15. Zuniga, A.F.; Turner, M.F.; Rasky, D. Building an Economical and Sustainable Lunar Infrastructure to Enable Lunar Industrialization. In Proceedings of the AIAA SPACE and Astronautics Forum and Exposition, Orlando, FL, USA, 12–14 September 2017; p. 5148.

16. Luhua, X.; Xianfeng, C.; Yuan, X. The Operations of China’s First Lunar Rover. In Proceedings of the SpaceOps 2012 Conference, Stockholm, Sweden, 11–15 June 2012; pp. 1–8.

17. Wang, Y.; Saksena, M. Scheduling fixed-priority tasks with preemption threshold. In Proceedings of the Sixth International Conference on Real-Time Computing Systems and Applications, Hong Kong, China, 13–15 December 1999; pp. 328–335.

18. Hosseinimotlagh, S.; Khunjush, F.; Hosseinimotlagh, S. A cooperative two-tier energy-aware scheduling for real-time tasks in computing clouds. In Proceedings of the 22nd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing, Turin, Italy, 12–14 February 2014; pp. 178–182.

19. Davis, R.I.; Burns, A. Priority assignment for global fixed priority pre-emptive scheduling in multiprocessor real-time systems. In Proceedings of the 30th IEEE Real-Time Systems Symposium, Washington, DC, USA, 1-4 December 2009; pp. 398–409.

20. Zhu, D.; Melhem, R.; Mossé, D. The effects of energy management on reliability in real-time embedded systems. In Proceedings of the IEEE/ACM International conference on Computer-aided design, San Jose, CA, USA, 7–11 November 2004; pp. 35–40.

21. Deepaksubramanyan, B.S.; Nunez, A. Analysis of subthreshold leakage reduction in CMOS digital circuits. In Proceedings of the 50th Midwest Symposium on Circuits and Systems, Montreal, QC, Canada, 5–8 August 2007; pp. 1400–1404.

22. Fataniya, B.; Patel, M. Survey on different method to improve performance of the round robin scheduling algorithm. *Int. J. Res. Sci. Eng. Technol.* 2018, 6, 69–77.

23. Ouni, B.; Belleudy, C.; Senn, E. Accurate energy characterization of OS services in embedded systems. *EURASIP J. Embedded Syst.* 2012, 2012, 1–16.

24. Liu, F.; Guo, F.; Solihin, Y.; Kim, S.; Eker, A. Characterizing and modeling the behavior of context switch misses. In Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques, Toronto, ON, Canada, 25–29 October 2008; pp. 91–101.

25. David, F.M.; Carlyle, J.C.; Campbell, R.H. Context switch overheads for Linux on ARM platforms. In Proceedings of the Workshop on Experimental Computer Science, San Diego, CA, USA, 25–26 June 2007; pp. 1–7.

26. Acquaviva, A.; Benini, L.; Riccò, B. Energy characterization of embedded real-time operating systems. In *Compilers and Operating Systems for Low Power*; Springer: Cham, Switzerland, 2003; pp. 53–73.

27. Behera, H.S.; Mohanty, R.; Nayak, D. A New Proposed Dynamic Quantum with Re-Adjusted Round Robin Scheduling Algorithm and Its Performance Analysis. arXiv 2011, arXiv:1103.3831.
28. Raveendran, B.K.; Prasad, K.D.; Balasubramaniam, S.; Gurunarayanan, S. Variants of priority scheduling algorithms for reducing context-switches in real-time systems. In Proceedings of the 8th International Conference on Distributed Computing and Networking, Guwahati, India, 27–30 December 2006; pp. 466–478.

29. Wang, Y.; Saksena, M. Scheduling Fixed-Priority Tasks with Preemption Threshold An Attractive Technology? Montreal: Concordia University. 1999. Available online: http://www.cs.utah.edu/~regehr/reading/open_papers/wang_saksena_attractive.pdf (accessed on 10 January 2020).

30. John Burkardt. C Software. Available online: https://people.sc.fsu.edu/~jburkardt/c_src/ (accessed on 12 January 2020).