sensors

MDPI

Article

Energy Efficient Real-Time Scheduling Using DPM on Mobile Sensors with a Uniform Multi-Cores

Youngmin Kim [1], Ki-Seong Lee [1] and Chan-Gun Lee [2,*]

[1] Da Vind College of Genneral Education, Chung-Ang University, Heuksuk-ro 84, Dongjak-gu, Seoul 06974, Korea; e010120302@cau.ac.kr (Y.K.); goory@cau.ac.kr (K.-S.L.)

[2] Department of Computer Science and Engineering, Chung-Ang University, Heuksuk-ro 84, Dongjak-gu, Seoul 06974, Korea

* Correspondence: cglee@cau.ac.kr; Tel.: +82-2-820-5829; Fax: +82-2-820-5301

Received: 31 October 2017; Accepted: 12 December 2017; Published: 14 December 2017

Abstract: In wireless sensor networks (WSNs), sensor nodes are deployed for collecting and analyzing data. These nodes use limited energy batteries for easy deployment and low cost. The use of limited energy batteries is closely related to the lifetime of the sensor nodes when using wireless sensor networks. Efficient-energy management is important to extending the lifetime of the sensor nodes. Most effort for improving power efficiency in tiny sensor nodes has focused mainly on reducing the power consumed during data transmission. However, recent emergence of sensor nodes equipped with multi-cores strongly requires attention to be given to the problem of reducing power consumption in multi-cores. In this paper, we propose an energy efficient scheduling method for sensor nodes supporting a uniform multi-cores. We extend the proposed T-Ler plane based scheduling for global optimal scheduling of a uniform multi-cores and multi-processors to enable power management using dynamic power management. In the proposed approach, processor selection for a scheduling and mapping method between the tasks and processors is proposed to efficiently utilize dynamic power management. Experiments show the effectiveness of the proposed approach compared to other existing methods.

Keywords: WSNs; mobile sensor; energy efficiency; DPM; T-Ler plane; real-time scheduling

1. Introduction

WSNs consist of a number of moblile sensor nodes which are tiny, multi-functional, and low-power. Table 1 lists mobile sensing platforms with various sensors. It is widely used in various applications to collect and process data, such as various types of physical and environment information. Recently, sensor nodes in WSNs have evolved for multimedia streaming and image processing. In response to these high performance demands, sensor nodes with multi-processors have emerged. A multi-processor sensor node platform, mPlatform, which is capable of parallel processing for computationally intensive signal processing, was proposed by Lymberopoulos et al. [1]. These platforms operate with limited batteries, as shown in Table 1. The use of a multi-cores in the sensor node makes energy consumption more serious. Power management among sensor nodes is of critical importance for several reasons: limited energy batteries and ensuring longevity [2–4], meeting performance requirements [2,5,6], inefficiency arising because of over provisioning resources [2], power challenges posed by CMOS scaling [2,7], and enabling green computing [2]. Recent advances in CMOS technology have improved the density and speeds for on-chip transistors. These trends limit the fraction of chips that can be used at maximum speeds within limited power. Therefore, power challenges in CMOS have been addressed for processor performance. Transistor performance scaling in the future may end if left unaddressed [8,9]. Battery-operated embedded systems are sensitive to high power consumption, which leads to heating and reduced battery lifetime. Thus,

energy-efficient management is essential in several embedded systems such as wearable devices. Improving energy efficiency leads to scale performance without violating the power budget. In recent advances, mobile computing devices with a multi-cores have been dynamically increased for mobile convergence applications (e.g., video streaming and web browsing). Power management in embedded systems contributes to achieve nearly 3% of the overall carbon footprint in green computing [10]. Energy efficiency scheduling algorithms for the sensor node with multi-processors are necessary. The scheduling algorithms must be able to keep battery lifetime longer while meeting the time constraints.

Table 1. Mobile sensing platforms.

Platform Name	Processor Type	Sensor Type	Battery Type
R-One [11]	ARM Cortex-M3	Accelerometer, gyroscope, bump, IR, ambient light	3.7 V lithium-ploymer battery with 2000 mAh
E-puck [12]	dsPIC 30F6014A	IR, accelerometer, microphone	Battery swappable and rechargeable with 5 Wh
MarXBot [13]	ARM11	IR, camera, accelerometer, gyroscope, RFID, 2D force, microphone	Hot-swappable battery with 38 Wh
Foot-Bot [14]	i.MX31 ARM11	IR, camera	3.7 V lithium-polymer battery with 10-Ah
CITRIC [15]	Xscale PXA-270	Camera, microphone	Four AA batteries
WolfBot [16]	ARM Cortex-A8	IR, camera, microphone, ambient light, accelerometer, magnetometer	7.4 V lithium-ion battery with 5200 mAh

The Asymmetric Multi-core Platform (AMP) is capable of parallelism with different performance levels. The examples of AMP include mobile phones, tablets, and high-end mobile sensor nodes. These devices are equipped with cores capable of handling tasks requiring high-performance processing. Note that not all tasks need the high-performance processing and energy efficient schemes are adopted even for the cores consuming low power. The problem of scheduling AMP for high-performance mobile sensors is important in terms of performance and energy efficiency. The scheduler can switch the high-performance cores to a low power state by assigning tasks to the low power cores when processing the tasks requiring low loads. It is also possible that powerful cores are changed into simpler cores to adapt the system to varying loads. ARM's big.LITTLE architecture [17,18] is a representative example. In ARM's big.LITTLE architecture, there are three modes for task migration: cluster migration, CPU migration, and global task scheduling. The scheduler improves energy efficiency by migrating tasks between big and little cores. In this paper, we discuss real-time scheduling problems in the context of AMP. We adopt a scheme using the T-Ler plane to develop energy-efficient scheduling algorithms for real-time tasks on uniform multi-core systems.

The T-Ler plane extends the T-L plane using a T-L abstraction strategy to fit uniform multi-core systems. The Voltage Frequency Scale (VFS) is exploited on energy-efficient scheduling algorithms using the T-L plane. On the other hand, there are not many studies related to Dynamic Power Management (DPM). Sensor network applications with varying loads depending on the situation can take advantage of the energy by switching the state of unnecessarily used processors. Kim et al. [19,20] proposed several T-L plane based energy-efficient algorithms using DPM for sensor nodes with identical multi-processors. However, these algorithms are not suitable for uniform multi-processor systems. In particular, it is hard to select the set of processors with the lowest power consumption among the multiple sets of processors that have the same capacity. We propose a new algorithm suitable for sensor node with uniform multi-processors, called Uniform-DPM. More specifically, we extend the previous approaches [19,20] by considering the characteristics of uniform multi-processors in terms of energy efficiency as follows:

- At the beginning of each T-Ler plane, select the processors operating with a low frequency and minimize the processing capacity as much as possible.
- Reduce the complexity of scheduling and fragments of idle time, and classify the processors and tasks into processor sets and task sets at the beginning of the T-Ler plane, respectively.
- At each event in the T-Ler plane, utilize constrained migration to reduce the complexity of scheduling and fragments of idle time.

The first extension is to reduce the power loss caused by uniform multi-processors that consist of processors with difference processing capacities. The previous approach [20], as shown in Section 2, focuses solely on minimizing the number of processors. It is not suitable for uniform multi-processors. In the case of uniform multi-processors, the processors must be selected considering the processing capacity and the frequency of each processor. The second extension is to classify processors and tasks for limited scheduling, where tasks in a set are only scheduled to processors in the according processor set. The third extension is to adjust the sets in each event and to assign tasks to the processors using the limited scheduling. These prevent the unnecessary migration of tasks and enables the collection of idle time on particular processors.

We organize this paper as follows. In Section 2, we introduce related works, including the approaches previously based on T-L plane targeting uniform multi-processors. In Section 3, we propose mechanisms to select processors and allocate tasks for energy-efficient scheduling in uniform multi-processors. We extend the proposed events in identical multi-processors to ones in uniform multi-processors. In Section 4, we perform experimental evaluations by comparing our proposed algorithms with other algorithms. Lastly, we present the conclusions and future works in Section 5.

2. Related works

2.1. Power Management Techniques

Due to the advancements in semiconductor process technologies, there have been more high-end processors available that integrate more transistors. Recently, real-time embedded systems have been increasingly adopting high-end processors. In addition, to improve the performance, real-time embedded systems are also adopting multi-processors. However, this increases the processor power consumption significantly. The power consumption of CMOS chips is as follows [21]:

$$P_{total} = P_{static} + P_{dynamic}. \tag{1}$$

P_{static} is the static power consumption which is calculated as the sum of the leakage power and short current power. $P_{dynamic}$ is the dynamic power consumption by charging and discharging of the output capacitance for processing time. It is not easy to reduce the static power consumption which depends on various parameters in the semiconductor process. Therefore, we focus on reducing the dynamic power consumption. Dynamic power is defined as:

$$P_{dynamic} = \alpha C V^2 f, \tag{2}$$

where f is the frequency, α is the switching activity factor, V is the supply voltage, and C is the capacitive load. DVFS is a method used to adjust the supply voltage and frequency of a CMOS chip by utilizing the slack time that occurs when scheduling tasks. On the other hand, DPM is a method of reducing energy consumption by switching to a low power state when slack time occurs. However, if a sufficient slack time is not guaranteed over the break-even time, the energy overhead caused by the state transition will cause loss. The break-even time BET_{sleep} is determined by Equation (3) [22].

$$BET_{sleep} = \max(t_{sw}, \frac{E_{sw} - P_{sleep} \cdot t_{sw}}{P_{idle} - P_{sleep}}) \tag{3}$$

The transition energy overhead and recovery time are denoted as E_{sw} and t_{sw}, respectively. P_{idle} denotes the idle power. The sleep power is denoted by P_{sleep}. The break-even time should be considered when developing a scheduling algorithm that not only uses the sleep mode, but also guarantees real-time responsiveness.

2.2. Global Scheduling Approaches on Multi-Processors

Scheduling disciplines can be categorized by considering the complexity of the priority mechanisms and the degree of job migration. Considering how task priorities are determined, Carpenter et al. [23] have categorized the schemes to static, dynamic but fixed within a job, or fully dynamic.

- Static: A single fixed priority is applied to all jobs for each task in the system. e.g., Rate Monotonic (RM) scheduling.
- Dynamic but fixed within a job: Different priorities may be assigned for the jobs of a task, but a job has a fixed priority at different times. e.g., Earliest Deadline First (EDF) scheduling.
- Fully dynamic: Different priorities may be assigned for a single job at different times, e.g., Least Laxity First (LLF) scheduling.

Depending on the degree of job migration, Carpenter et al. [23] have categorized the migration criterion to no migration, restricted migration, and unrestricted migration.

- No migration: The set of tasks in the system is partitioned into some subsets for available processors, a scheduler schedules a subset on a unique processor. The jobs of a task in a subset are executed on the corresponding processor.
- Restricted migration: Each job of a task must be scheduled entirely on a single processor. However, other jobs of the same task may be executed on different processors. Therefore, migrations among processors are allowed at the task-level context, but not at job boundaries.
- Unrestricted migration: Any jobs is also allowed to migrate among processors during its lifetime.

Note that our proposed scheduling algorithm supports fully dynamic and unrestricted migration.

Various global scheduling algorithms for multi-processors have been studied. In global scheduling, all eligible jobs waiting for execution are in a single priority-ordered queue shared by all of the processors in the system; the highest priority job is dispatched from this queue by the global scheduler. Most of early the studies on global scheduling extended optimal scheduling algorithms known well for a single processor, such as RM and EDF, to multi-processors. However, these extensions can result in wasted utilization of resources. The fluid scheduling model with fairness notion, where each task is always executed at a fixed rate, emerged to overcome the limitation [24]. Figure 1 compares the fluid scheduling concept and the practical scheduling. There is a gap between fluid scheduling and practical scheduling, as shown in Figure 1. There are some algorithms extending the fluid scheduling model for achieving optimality on multi-processors. Proportionate fair (P-fair) scheduling has produced a feasible schedule for periodic tasks on multi-processors, and it has shown considerable promise in multi-processor scheduling [25]. However, extensive amount of migrations and preemption are needed to follow the fluid schedule. Much effort has been made to overcome this problem in global optimal scheduling. Thereafter, Deadline Partitioning-fair (DP-fair) and Deadline Partitioning-warp (DP-wrap) algorithms were proposed, and they exhibited better performance with respect to preemption in [26]. The method of allocating tasks to the processors supported by these scheduling algorithms is not suitable for uniform multi-processors. Cho et al. [27] proposed Largest Nodal Remaining Execution-time First (LNREF) using a T-L plane abstraction and it performs well with uniform multi-processors. Funk and Meka [28] proposed a T-L plane based scheduling algorithm, U-LLREF, that extends LNREF algorithm for uniform parallel machines. In U-LLREF, a uniform multi-processors provides a condition for determining event-c. Chen et al. [29] proposed Precaution

Cut Greedy (PCG), a T-L plane based scheduling algorithm for uniform multi-processors. PCG uses a modified T-L plane, a T-Ler plane. Figure 2 shows how the PCG schedules in the first T-Ler plane. When event-c occurs, τ_3 is assigned to p_2 until the end of the T-Ler plane. Thus, in PCG, a task monopolizes a single processor, thereby preventing unnecessary task migration.

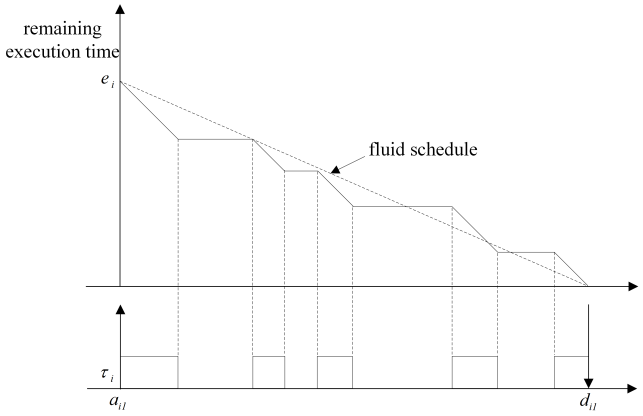


Figure 1. Fluid schedule model.

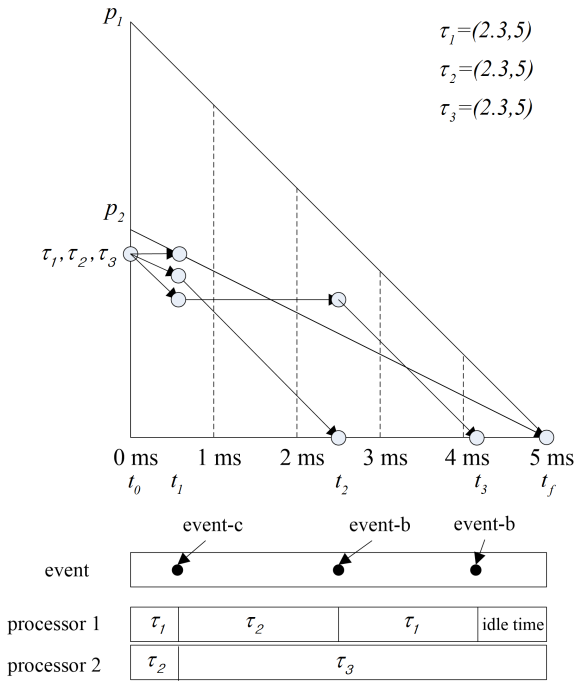


Figure 2. A scheduling example in the 1st T-Ler plane.

2.3. T-L Plane Based Energy-Efficient Global Optimal Scheduling Approaches

Energy-efficient scheduling based on the T-L plane for uniform parallel machines has been proposed due to the demand for energy efficiency. Uniform RT-SVFS [30] reduces the energy consumption by scaling the frequency of all processors with a constant rate. By scaling the height of the T-L plane, as shown in Figure 3, scheduling is enabled at the changed frequency. α_k represents the normalized frequency of the processor. In addition, energy-efficient T-L plane based scheduling algorithms for unrelated parallel machines have emerged. Independent RT-SVFS [30] determines the frequency by statically scaling each processor. This algorithm has been proposed to overcome the heavy task bottlenecks that can occur when using the frequency scaling technique. The Growing Minimum Frequency (GMF) [31], which is a state-of-the-art algorithm for T-L plane based non-uniform

frequency scaling for saving energy on VFS embedded multi-processors, has been proposed for the frequency control of multi-processors using U-LLREF, and the global optimal frequency can be determined. RT-DVFS [32] allows you to dynamically adjust the frequency of each processor in the event of scheduling.

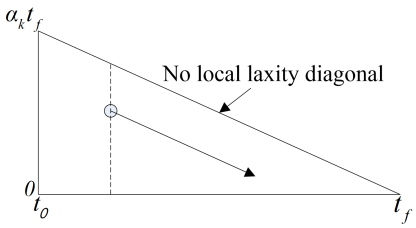


Figure 3. Transistion of the T-L plane (frequency= α_k).

It is difficult to consider DPM due to the idle time fragmentation problem that occurs when using the T-L plane based scheduling algorithm. In addition, since scheduling is performed using all processors existing in the system, a considerable energy overhead due to unnecessary state transition occurs when DPM is used. TL-DPM [19] solves the idle time fragmentation problem by using a new event to retrieve tokens, which is performed in the next plane. However, since only the token of the next plane is targeted, there is room for solving the idle time fragmentation problem. Kim et al. [20] proposed a generalized method for executing tokens to be scheduled in the later plane in the current plane in order to solve this problem. To reduce the number of state transitions, scheduling is performed using only the minimum number of processors.

3. Proposed Energy Efficient Approach on Uniform Multi-Processors

3.1. Feasibility Conditions

Theorems 1 and 2 represent the conditions that must be met to obtain schedules satisfying the time constraints when uniform multi-processors are used for scheduling the given task set.

Theorem 1. *(Horvath et al. [33]) The level algorithm constructs a minimal length schedule for the set of independent tasks with service requirements $e_1 \geq e_2 \geq ... \geq e_n$ on the processing system $\pi = (c_1 \geq c_2 \geq ... \geq c_m)$, where $m \leq n$. The schedule length is given by*

$$\max\{\max_{1 \leq i \leq m}(\frac{e_i}{c_i}), \frac{e_n}{c_m}\}. \tag{4}$$

Theorem 2. *(Funk et al. [34]) Consider a set $\tau = \{\tau_1, ..., \tau_n\}$ of periodic tasks indexed according to non-increasing utilization (i.e., $u_i \geq u_{i+1}$ for all i, $1 \leq i \leq n$, where $u_i = e_i/p_i$). Let $U_i = \sum_{j=1}^{i} u_i$ for all i, $1 \leq i \leq n$. Let π denote a system of $m \leq n$ uniform processors with capacities c_1, c_2, ..., c_m, $c_i \geq c_{i+1}$ for all i, $1 \leq i \leq m$. Periodic task system τ can be scheduled to meet all deadlines on the uniform multi-processor platform π if and only if the following constraints hold:*

$$U_n \leq c_m, \tag{5}$$

where $U_k \geq c_k$, for all $k = 1, 2, ..., m$.

Selecting processors for the scheduling tasks at the beginning of each T-L plane is divided into the case where tasks are allocated to the processors with the same capacity as the utilization, and the case where they are not.

3.2. Processor Selections and Classification

3.2.1. Simple Case: Exact Match

Table 2 shows some examples of processor selections for scheduling the task set shown in Table 3. In a CMOS chip, power consumption is determined by the operating frequency and supply voltage. The relationship between the power consumption and the supply voltage in the processor is as follows.

$$P_{dynamic} \propto V^2. \tag{6}$$

In addition, according to the relationship between the supply voltage and the operating frequency in a processor, shown in Equation (7), a processor operating at a higher frequency requires a higher supply voltage than that operating at a lower frequency. Therefore, as shown in Table 2, a processor with a higher supply voltage will have a higher capacity.

$$f \propto \frac{(V - V_{th})^{\beta}}{V}, \tag{7}$$

where V_{th} is the threshold voltage of transistors and β is a measure of the velocity saturation in COMS transistors.

Table 2. An example of the available processor sets.

	S_1	S_2	S_3
p_1 (voltage = 1.4 v, freq. = 600 MHz, capacity = 1)	O	X	O
...	X	X	X
p_5 (voltage = 1.2 v, freq. = 300 MHz, capacity = 0.5)	X	O	O
p_6 (voltage = 1.2 v, freq. = 300 MHz, capacity = 0.5)	X	O	X
...	X	X	X
p_{n-1} (voltage = 1 v, freq. = 150 MHz, capacity = 0.25)	O	O	X
p_n (voltage = 1 v, freq. = 150 MHz, capacity = 0.25)	O	O	X
...	X	X	X
total capapcity	1.5	1.5	1.5

Table 3. Task properties.

Task	Period	WCET	Utilization
τ_1	5 ms	2.5 ms	0.5
τ_2	10 ms	5 ms	0.5
τ_3	10 ms	1.25 ms	0.25
τ_4	20 ms	2.5 ms	0.25

S_1, S_2, and S_3 shown in Table 2 satisfy Theorems 1 and 2 presented above. Since the processing capacity of S_1, S_2, and S_3 is equal to the total utilization of the task set, there is no idle time when the task set is scheduled. However, since the number and capacity of processors is not the same in each processor, the power consumed by S_1, S_2, and S_3 is different. The energy consumption for scheduling the task set in Table 3 on S_1, S_2, and S_3 is shown in Table 4. The lowest power consumption can be observed on S_2, where each task is independently assigned to a processor whose capacity is equal to the utilization of each task in the task set. If the total capacity of a processor set is equal to the total utilization of a task set, then there is no idle time because all processors always perform their tasks. Therefore, the power consumption of each processor is dependent on the processed workload. High-capacity processors can handle more work in terms of processor workloads. Equation (8) shows the power consumption $E_e(V_k)$ needed to process e_i in a processor whose operating frequency and supply voltage are f_k and V_k, respectively.

$$E_e(V_k) = \alpha C V_k^2 f_k(\frac{e_i}{f_k}) = \alpha C V_k^2 e_i, \tag{8}$$

where V_k and f_k denote the voltage and capacity of the *k-th* processor, respectively. Lemma 1 shows the power consumption characteristics of processor sets whose total capacity is equal to the total utilization of a task set.

Table 4. Dynamic power consumption of some feasible processor sets.

	S_1	S_2	S_3
Dynamic power consumption	$2.46\alpha C$	$1.94\alpha C$	$2.68\alpha C$

Lemma 1. *If $U_{total} = c_n = c_i + c_j$, when scheduling a task set with U_{total} on two processor sets $S_1 = \{c_n\}$ and $S_2 = \{c_i, c_j\}$, the power consumption satisfies $\alpha C V_n^2 e_n > \alpha C V_i^2 e_i + \alpha C V_j^2 e_j$.*

Proof of Lemma 1. According to Equation (8), the power consumption measures of S_1 and S_2 are $\alpha C V_n^2 e_n$ and $\alpha C V_n^2 e_i + \alpha C V_i^2 e_j$ respectively. Since $c_n = U_{total}$ and $c_i + c_j = U_{total}$, there is no idle time when the tasks are scheduled. In addition, $V_n^2 e_n = V_n^2(e_i + e_j) > V_i^2 e_i + V_j^2 e_j$, where $e_n = e_i + e_j$ and $V_n > V_i, V_j$. Hence, $\alpha C V_n^2 e_n > \alpha C V_i^2 e_i + \alpha C V_j^2 e_j$. □

According to Lemma 1, selecting the 0.8 capacity processor for scheduling the 0.6 and 0.2 utilization tasks in the task set, as shown in Table 1, will result in higher power consumption than selecting the 0.6 and 0.2 capacity processors for the scheduling the tasks. Therefore, assigning each task to the processor sets whose capacity is equal to its utilization is the most energy-efficient way when there are enough processors. Under the condition of $c_i \leq u_i$, the processor whose capacity is equal to u_i shows the lowest power consumption to execute the task with the utilization u_i. Lemma 2 shows these characteristics.

Lemma 2. *When a task with utilization u_i is executed on two processors under the condition of $c_n > c_j = u_i$, their power consumption for processing the allocated workload during the task period is $E_e(V_n) > E_e(V_j)$.*

Proof of Lemma 2. When two processors with capacities of c_n and c_j perform the workload e_i during the period p_i, their power consumption measures are $E_e(V_n)$ and $E_e(V_j)$ respectively. If $c_n > c_j$, $V_n > V_j$ is satisfied by Equation (7). Hence, $E_e(V_n) > E_e(V_j)$ is satisfied by Equation (8). □

When a task set with the total time of $\sum_{i=1}^{n} e_i$ is scheduled on n processors whose capacity is different, the power consumption required for processing the allocated workload on n processors is shown in Equation (9). e_1, e_2, ..., e_n represents the workload assigned to each processor.

$$\begin{aligned} E_e &= \alpha C V_1^2 f_1(\frac{e_1}{f_1}) + \alpha C V_2^2 f_2(\frac{e_2}{f_2}) + ... + \alpha C V_n^2 f_n(\frac{e_n}{f_n}) \\ &= \alpha C V_1^2 e_1 + \alpha C V_2^2 e_2 + ... + \alpha C V_n^2 e_n \\ &= \sum_{i=1}^{n} E_e(V_i). \end{aligned} \tag{9}$$

If the total capacity of n processors is greater than the total utilization of a task set to be scheduled, there will be idle time during task scheduling. This means that the power consumption during the idle time should be taken into account to measure the processors' power consumption required for scheduling the task set. The power consumption of n processors is shown in Equation (10). α_i denotes the power consumption of the i-th processor during the idle time.

$$E_d = \sum_{i=1}^{n} (E_e(V_i) + \alpha_i). \tag{10}$$

Lemma 3 and Theorem 3 show the power consumption required for scheduling a task set on a set of n processors with different capacities.

Lemma 3. *When the task set is scheduled with the processor set S_1, the lowest power is consumed, where the total capacity of S_1 is $\sum_{\forall \tau_i \in \tau} u_i = \sum_{\forall p_i \in S_1} c_i$.*

Proof of Lemma 3. If $\sum_{\forall \tau_i \in \tau} u_i = \sum_{\forall p_i \in S_1} c_i$, scheduling involves no idle time, so the processor power consumption is $\sum_{\forall p_i \in S_1} E_e(V_i)$. If $\sum_{\forall \tau_i \in \tau} u_i < \sum_{\forall p_i \in S_1} c_i$, scheduling involves some idle time, so the processor power consumption based on Equation (10) is $\sum_{\forall p_i \in S_1} (E_e(V_i) + \alpha_i)$. Hence, if $\sum_{\forall \tau_i \in \tau} u_i = \sum_{\forall p_i \in S_1} c_i$, then the lowest power consumption will be observed. □

Theorem 3. *Independently assigning each task in the task set τ to processors whose capacity is equal to the utilization of the task u_i shows the lowest power consumption for scheduling a set of tasks.*

Proof of Theorem 3. This is easily proven by Lemmas 1–3. □

Therefore, selecting processors whose capacity is equal to the utilization of each task shows the lowest power consumption for scheduling a set of tasks.

3.2.2. Generalized Solution

When not assignable to a processor with the same capacity as the task's utilization, it is necessary to select a processor set available for scheduling with the limited processors. Table 5 shows the characteristics of processors used for task scheduling. Table 6 shows the processor sets selected from the processors in Table 5 for scheduling the task set shown in Table 7.

Table 5. Processor properties.

	p_1	p_2	p_3	p_4
Supply voltage	1.4 V	1.2 V	1.0 V	1.0 V
Operating frequency	600 MHz	300 MHz	150 MHz	75 MHz
Processing capacity	1	0.5	0.25	0.125

Table 6. Selecting processors for scheduling a task set.

	S_1	S_2	S_3	S_4	S_5	S_6	S_7	S_8	...
p_1	O	O	O	O	X	O	O	X	...
p_2	O	O	O	X	O	O	X	O	...
p_3	O	O	X	O	O	X	O	O	...
p_4	O	X	O	O	O	X	X	X	...
Total capacity	1.875	1.75	1.625	1.375	0.875	1.5	1.25	0.75	...

Since the processor sets S_1, S_2, S_3, and S_6 shown in Table 6 satisfy Theorems 1 and 2, they can be used for task scheduling. However, since the processor sets are differently configured, the idle time during the task scheduling and the difference in their supply voltages result in their different power consumption. Therefore, the following two strategies should be considered to select energy-efficient processors.

Table 7. Task properties.

Task	Period	WCET	Utilization
τ_1	5 ms	4.5 ms	0.9
τ_2	10 ms	4.25 ms	0.425

Selecting a processor set for task scheduling in consideration of all the problems presented above is a NP-hard problem. Therefore, in this paper, we propose a heuristic method for selecting an energy-efficient processor set. In the proposed method, if the size of the current plane is smaller than C_{sleep}, the processor in active mode is added to a processor set for task scheduling because it cannot be switched to sleep mode at the end of the previous plane. If the preferentially selected processors are not enough for scheduling the given task set, additional processors will be selected. Processors for scheduling are selected in terms of the local utilization of tasks from highest to lowest. Selecting processors for scheduling depends on the difference between the total local utilization of tasks in τ_{ready} at the start time t_0 in each plane $\sum_{\tau_j \in \tau_{ready}} r_j(t_0)$ and the total capacity $\sum_{p_j \in P_{selected}} c_j$ of the processors in $P_{selected}$. The selected processors are moved to $P_{selected}$. The following describes how to select processors.

- If $0 \leq \sum_{\tau_j \in \tau_{ready}} r_j(t_j) - \sum_{p_j \in P_{selected}} c_j < r_i(t_0)$, the processor with the smallest capacity is selected for scheduling in the given processor set, $\{p_j | c_j \geq r_i(t_0) - \sum_{\tau_j \in \tau_{ready}} r_j(t_0) - \sum_{p_j \in P_{selected}} c_j$ *where* $p_j \in \{P_{all} - P_{selected}\}\}$.
- If $\sum_{\tau_j \in \tau_{ready}} r_j(t_0) - \sum_{p_j \in P_{selected}} c_j \geq r_i(t_0)$, the previously selected processor is used for scheduling without selecting an additional processor.

P_{all} is the set of all the processors in the system. $P_{selected}$ is the set of the selected processors for task scheduling. Algorithm 1 shows how to select processors for scheduling at the beginning of each plane. The function *getMinimumCapacityProcessor(availableCapacity, τ, P_{temp})* takes the available capacity (*availableCapacity*) of the previously selected processor into account to return the lowest capacity processor for scheduling the task set τ from the given processor set P_{temp}. The function *add*() adds elements to the set, and the function *erase*() removes elements from the set. The processors in P_{sleep} indicate the processor in the sleep state in the plane. It is necessary to ensure a break-even time longer than the idle time in order to use DPM techniques for switching the state of a processor. To ensure the idle timeis long enough to enter the sleep mode, the idle time in the plane should be generated as much as possible on a single processor. To prevent unnecessary power consumption, a task is assigned to the selected processor whose capacity is the lowest for scheduling the task. For this reason, in the proposed method, the processors in the selected processor set are classified into the following categories: processors that can be used to the maximum extent in the plane and processors that can be used exclusively by a single task in the plane. That is, the processors in $P_{selected}$ are classified into the following categories: P_{fixed}, P_{max}, and P_{slack}. The processors in P_{fixed} represent a set of processors exclusively used by a single task. P_{max} is the set of processors used to the maximum extent in the plane. P_{slack} is the set of processors that may result in idle time in the plane during task scheduling. The tasks to be executed on the classified processor sets are divided into the following categories: τ_{fixed}, τ_{max}, and τ_{slack}. Tasks assigned to a processor set cannot be moved to another processor set. The following describes how to classify the processor sets.

Algorithm 1 Processor selection of the beginning time of a T-L plane

Input : $P_{all}, P_{sleep}, \tau_{all}, psize$
Output : $P_{selected}, \tau_{ready}$
psize—Size of the T-L plane
P_{all}—The set of processors in the system
P_{sleep}—The set of processors to be sleep mode
$P_{selected}$—The set of processors selected for scheduling tasks
P_{temp}—The temporary set of processors
τ_{all}—The set of all tasks in the system
τ_{ready}—The set of ready tasks
τ—Temporary variable for tasks
p—Temporary variable for processors
availableCpacity—Temporary variable for available capacity
for $\forall p \in P_{all} - P_{sleep}$ **do**
  if $psize < p.C_{sleep}$ **then**
    $add(p, P_{temp})$;
  end if
end for
for $\forall \tau \in \tau_{all}$ **do**
  if $\tau.e > 0$ **then**
    $add(\tau, \tau_{ready})$;
  end if
end for
repeat
  $\tau = getFirstLocalUtilizationTask(\tau_{ready})$;
  $availableCapacity = \sum_{p_i \in P_{selected}} p_i.c - \sum_{\tau_i \in \tau_{ready}} \tau_i.r(t_0)$
  if $availableCapacity \geq \tau.r(t_0)$ **then**
    contitue;
  else
    $p = getMinimumCapacityProcessor(availableCapacity, \tau, P_{temp})$;
    if p is null **then**
      $p = getMinimumCapacityProcessor(availableCapacity, \tau, P_{sleep})$;
      if $p.c > p_1.c$ **then**
        $erase(p, P_{sleep})$
      end if
      $erase(p, P_{sleep})$;
    else
      $erase(p, P_{temp})$;
    end if
    $add(p, P_{selected})$;
    $p_1 = p$;
  end if
until τ is not null
return $P_{selected}, \tau_{ready}$

1. To select a processor for scheduling a task τ_i in τ_{ready} where the difference between the total local utilization of the tasks in τ_{slack} at t_0 ($\sum_{\tau_j \in \tau_{slack}}(t_0)$) and the total capacity of the processors in P_{slack} ($\sum_{\tau_j \in \tau_{slack}} r_j$) is greater than zero: $\sum_{\tau_j \in \tau_{slack}} r_j(t_0) - \sum_{p_j \in P_{slack}} c_j > 0$.
   - If $\sum_{\tau_j \in \tau_{slack}} r_j(t_0) - \sum_{p_j \in P_{slack}} c_j \geq r_i(t_0)$, the task is additionally assigned to a previously selected processor without selecting an additional processor. The assigned task is moved from τ_{slack} to τ_{ready}.
   - If $\sum_{\tau_j \in \tau_{slack}} r_j(t_0) - \sum_{p_j \in P_{slack}} c_j < r_i(t_0)$, the task is additionally assigned to a previously selected processor without selecting an additional processor. The assigned task is moved from τ_{slack} to τ_{ready}.

2. If $\sum_{\tau_j \in \tau_{slack}} r_j(t_0) - \sum_{p_j \in P_{sl}} c_j = r_i(t_0)$,

   - All processors and tasks in P_{slack} and τ_{slack} are moved to P_{max} and τ_{max}.
   - The processor whose capacity is the lowest for scheduling a task τ_i, is selected from the following processor set, $\{p_j | c_j \geq r_i(t_0) where p_j \in P_{selected}\}$. If the capacity of the selected processor is equal to the local utilization ($r_i(t_0)$) of the task t_i, the processor and the task are moved to P_{fixed} and τ_{fixed}, respectively. Otherwise, they are moved to P_{slack} and τ_{slack}, respectively.

Algorithm 2 shows how to classify the processor set $P_{selected}$ into the following categories: P_{fixed}, P_{max}, and P_{slack}. The task with the highest local utilization is considered first to classify the processor set and the task set. The function *getFirstLocalUtilizationTask*(τ_{ready}) returns the task with the highest local utilization in τ_{ready}. The function *getMinimumCapacityProcessor*(*availableCapacity*, τ, $P_{selected}$) takes *availableCapacity* into account to return the processor whose capacity is the lowest for scheduling a task τ in $P_{selected}$. If the capacity of the returned processor is equal to the local utilization of the task, the processor and the task is moved to P_{slack} and τ_{slack}, respectively. If *availableCapacity* is 0, the processors in P_{slack} and the tasks in τ_{slack} are moved to P_{max} and τ_{max}.

3.3. Scheduling Strategy

In the paper written by Chen et al. [29] , there are two suggested methods for scheduling on a uniform multi-processors. However, event-t, event-s, and event-r presented above are not taken into account in these scheduling methods. In this section, we propose a new T-L plane based scheduling method in which event-t, event-s, and event-r are used to reduce the power consumption of a uniform multi-processors. When the τ_{fixed}, τ_{max}, and τ_{slack} tasks are scheduled with the P_{fixed}, P_{max}, and P_{slack} processor sets, the tasks cannot be moved from one processor set to another in order to generate no idle time on the processors in P_{fixed} and P_{max}. The remaining part shows the movement of elements between task sets and processor sets and the processor assignment when a rescheduling event occurs. Since event-t as defined above targets identical multi-processors is not suitable for uniform multi-processors, it is redefined as in Definition 1.

Definition 1. *An event-t in uniform multi-processors occurs at t_t if the following conditions are met.*

- $t_f - t_t \geq C_{sleep}$.
- $\sum_{\tau_i \in \tau_{active}} r_i(t_t) = (\sum_{p_i \in P_{slack}} c_i) - c_j$ where $p_j \in P_{slack}$.

Algorithm 3 shows the process of assigning tasks to processors when a rescheduling event occurs. All the tasks in the set τ_{active} are moved to the set τ_{ready}, and all the tasks in the set τ_{active} are deleted. The function *eraseAll*(τ_{active}) removes all elements in the set τ_{active}. Tasks are assigned to processors in each processor set in the following order: P_{slack}, P_{max}, and P_{fixed}. The function *getMaximumLocalUtilizationTask*($p.c$, τ_{fixed}, τ_{ready}) returns the task with the highest local utilization in τ_{fixed} and τ_{ready} where the task can be performed on the processor with the capacity of $p.c$. The function *getFirstLocalUtilizationTask*(τ_{fixed},τ_{ready}) returns the task with the highest local utilization in τ_{fixed} and τ_{ready}. The function *allocateTaskToProcessor*(τ, p) assigns the task τ to the processor p.

Algorithm 2 Classification of selected processors for scheduling

Input : $P_{selected}, \tau_{ready}$
Output : $P_{fixed}, P_{max}, P_{slack}, \tau_{fixed}, \tau_{max}, \tau_{slack}$
P_{fixed}—The set of processors fixed by a task
P_{max}—The set of processors having maximum utilization
P_{slack}—The set of processors to be able to have slack time
τ_{fixed}—The set of tasks fixed to a processor on on P_{fixed}
τ_{max}—The set of tasks scheduled on P_{max}
τ_{slack}—The set of tasks scheduled on P_{slack}
τ_1—Temporary variable for tasks
τ_2—Temporary variable for tasks
p_1—Temporary variable for processors
p_2—Temporary variable for processors
repeat
    $\tau_1 = getFirstLocalUtilizationTask(\tau_{ready})$;
    $availableCapacity = \sum_{p_i \in P_{slack}} p_i.c - \sum_{\tau_i \in \tau_{slack}} \tau_i.r(t_0)$;
    $p_1 = getMinimumCapacityProcessor(availableCapacity, \tau 1, P_{selected})$;
    if $p_1.c = \tau_1.r(t_0)$ **then**
        add(p_1, P_{fixed});
        add(τ_1, τ_{fixed});
    else if $availableCapacity = 0$ **then**
        for $\forall \tau_2 \in \tau_{slack}$ **do**
            add(τ_2, τ_{max});
            erase(τ_2, τ_{slack});
        end for
        for $\forall p_2 \in P_{slack}$ **do**
            add(p_2, P_{max});
            erase(p_2, P_{slack});
        end for
        add(τ_1, τ_{slack});
        add(p_1, P_{slack});
    else
        add(τ_1, τ_{slack});
        add(p_1, P_{slack});
    end if
    erase($p_1, P_{selected}$);
until τ_1 is not null
return $P_{fixed}, P_{max}, P_{slack}, \tau_{fixed}, \tau_{max}, \tau_{slack}$

Algorithm 4 shows the movement of the elements between processor sets and task sets. When an event-b occurs, all the tasks which have triggered an event-b are moved to τ_{done} and are removed from τ_{active}. The function *getEventTasks*() returns all the tasks that have triggered the event-b. When an event-c or an event-f occurs, all the tasks that have triggered the event are moved to τ_{fixed}, and the processors that have triggered the event are moved to P_{fixed}. The function $getProcessor(\tau.r(t_0), P_{max})$ returns the processor with the capacity $\tau.r(t_0)$ in P_{max}. When an event-t occurs, the processors which can be switched to sleep mode are moved to P_{sleep} and are removed from P_{slack}. When an event-s or an event-r occurs, all the tasks that have triggered the event are moved to τ_{done} and are removed from τ_{active}. The function *reallocateProcessorTime*() assigns the available processing time to a task with remaining execution time in τ_{done}. The assigned task is moved to τ_{ready}.

Algorithm 3 Assignment of tasks to processors at rescheduling

Input : $P_{fixed}, P_{max}, P_{slack}, \tau_{fixed}, \tau_{max}, \tau_{slack}$
Output : $\tau_{fixed}, \tau_{max}, \tau_{slack}$
for $\forall \tau \in \tau_{active}$ **do**
    $add(\tau, \tau_{ready})$;
end for
$eraseAll(\tau_{active})$;
for $\forall p \in P_{slack}$ **do**
    $\tau = getMaximumLocalUtilizationTask(p.c, \tau_{slack}, \tau_{ready})$;
    if τ is null **then**
        $\tau = getFirstLocalUtilizationTask(\tau_{slack}, \tau_{ready})$;
    end if
    $allocateTaskToProcessor(\tau, p)$;
    $erase(\tau, \tau_{ready})$;
    $add(\tau, \tau_{active})$;
end for
for $\forall p \in P_{max}$ **do**
    $\tau = getMaximumLocalUtilizationTask(p.c, \tau_{max}, \tau_{ready})$;
    if τ is null **then**
        $\tau = getFirstLocalUtilizationTask(\tau_{max}, \tau_{ready})$;
    end if
    $allocateTaskToProcessor(\tau, p)$;
    $erase(\tau, \tau_{ready})$;
    $add(\tau, \tau_{active})$;
end for
for $\forall p \in P_{fixed}$ **do**
    $\tau = getMaximumLocalUtilizationTask(p.c, \tau_{fixed}, \tau_{ready})$;
    $allocateTaskToProcessor(\tau, p)$;
    $erase(\tau, \tau_{ready})$;
    $add(\tau, \tau_{active})$;
end for
return $\tau_{fixed}, \tau_{max}, \tau_{slack}$

Figure 4 shows the scheduling in the first plane from the proposed method when scheduling the tasks of Table 8 on the processors listed in Table 9. Algorithm 2 is used to categorize the processor sets and ready tasks selected by Algorithm 1 at t_0. Task τ_5 that has triggered an event-c at τ_1 and the processor p_3 whose capacity is equal to the local utilization of τ_5 are moved to τ_{fixed} and P_{fixed}, respectively. At the same time, the processor p_4 is moved to P_{sleep} by event-t. Task τ_1 that has triggered an event-b at t_2 is moved to τ_{done}. Task τ_3 that has triggered an event-b at t_3 is moved to τ_{done}. At the same time, task τ_3 that has triggered an event-c and the processor p_1 whose capacity is equal to the local utilization of τ_3 are moved to τ_{fixed} and P_{fixed}, respectively. Table 10 shows the elements added to the processor and task sets by Algorithm 4 at each event in the *1st* plane. Tasks are assigned to processors by Algorithm 3. As shown in Figure 4, tasks assigned to processors move diagonally along the slope of the processor capacity and tasks unassigned to processors will move horizontally.

Algorithm 4 Movement of elements during rescheduling in the T-L plane

Input : $P_{fixed}, P_{max}, P_{slack}, \tau_{fixed}, \tau_{max}, \tau_{slack}$
Output : $P_{sleep}, \tau_{ready}, \tau_{active}, \tau_{done}$
τ_{active}—The set of tasks to be excuted
τ_{done}—The set of tasks to be done
if event-b **then**
    $T = getEventbTasks()$;
    for $\forall \tau \in T$ **do**
        $add(\tau, \tau_{done})$;
        $erase(\tau, \tau_{active})$;
    end for
else if event-c | event-f **then**
    $T = getEventcOrEventfTasks()$;
    for $\forall \tau \in T$ **do**
        $add(\tau, \tau_{fixed})$;
        if $\tau \in \tau_{max}$ **then**
            $p = getProcessor(\tau.r(t_0), P_{max})$;
            $erase(\tau, \tau_{max})$;
            $erase(p, P_{max})$;
        else
            $p = getProcessor(\tau.r(t_0), P_{slack})$;
            $erase(\tau, \tau_{slack})$;
            $erase(p, P_{slack})$;
        end if
        $add(p, P_{fixed})$;
    end for
else if event-t **then**
    $capacity = \sum_{p_i \in P_{slack}} p_i.c - \sum_{\tau_i \in \tau_{slack}} \tau_i.r(t_0)$
    $p = getProcessor(capacity, P_{slack})$;
    $add(p, P_{sleep})$;
    $erase(p, P_{slack})$;
else if event-s | event-r **then**
    $T = getEventsOrEventrTasks()$;
    for $\forall \tau \in T$ **do**
        $add(\tau, \tau_{done})$;
        $erase(\tau, \tau_{active})$;
    end for
    $reallocationProcessorTime()$;
    for $\forall \tau \in \tau_{done}$ **do**
        if $\tau.l(t_{cur}) = 0$ **then**
            continue;
        else
            $add(\tau, \tau_{ready})$;
            $erase(\tau, \tau_{done})$;
        end if
    end for
end if
return $P_{sleep}, \tau_{ready}, \tau_{active}, \tau_{done}$

Table 8. Task properties.

Task	Period	WCET	Utilization
τ_1	5 ms	4 ms	0.8
τ_2	5 ms	2.5 ms	0.5
τ_3	10 ms	3 ms	0.3
τ_4	10 ms	2 ms	0.2
τ_5	20 ms	2 ms	0.1

Table 9. Processor properties.

	p_1	p_2	p_3	p_4
Supply voltage	1.4 V	1.2 V	1.0 V	1.0 V
Processing capacity	1	0.5	0.25	0.25

Table 10. Example of sets at events in the plane.

Set	Element			
	t_0	t_1	t_2	t_3
τ_{fixed}	τ_2	τ_2, τ_5	τ_2, τ_4	τ_2, τ_4, τ_5
P_{fixed}	p_2	p_2, p_3	p_2, p_3	p_1, p_2, p_3
τ_{max}	.	.	.	.
P_{max}	.	.	.	.
τ_{slack}	τ_1, τ_3, τ_4, τ_5	τ_1, τ_3, τ_4	τ_3, τ_4	.
P_{slack}	p_1, p_3, p_4	p_1	p_1	.
τ_{done}	.	.	τ_1	τ_1, τ_3
P_{sleep}	.	p_4	p_4	p_4

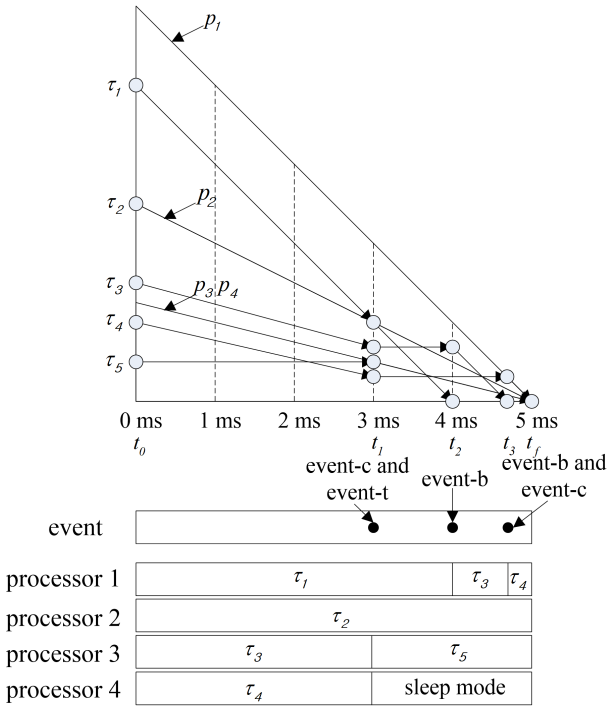


Figure 4. A scheduling in the first plane.

4. Energy Efficiency on Uniform Multi-Processors

In this section, the performance of the proposed algorithm is compared with the major algorithms previously developed for power management. We implemented a simulator operating in Windows 10 using the Ruby language (version 2.4.1) for the experiments. Figure 5 illustrates the architecture of the simulator. The results of the simulation show the energy consumption for task executions, as well as the energy overheads associated with the state transitions.

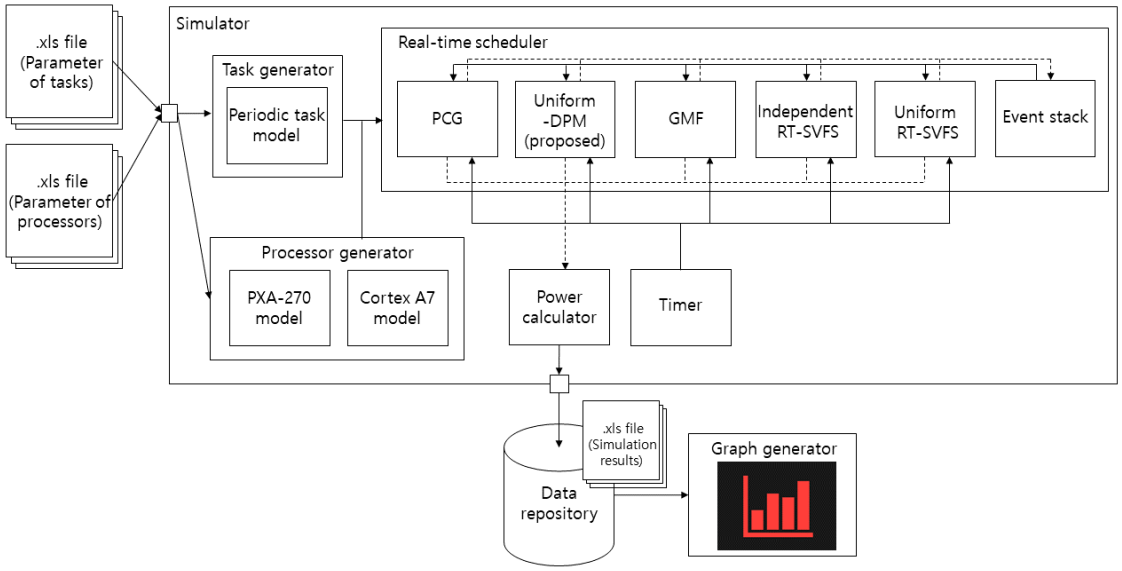


Figure 5. The architecture of the simulator.

4.1. Experiment Environment

The characteristics of the cortex-A7 core in Marvell's MV78230, which is the Multi-Core ARMv7 system based on the chip processor, is used to set the experimental parameters of the processor in the simulator. This core supports dynamic frequency scaling and dynamic power down options. Tables 11 and 12 show that cortex-A7 supports six frequency levels and five processor states. Run thermal is used in the stress test of the CPU. The deep idle and sleep modes consume the same energy with respect to the CPU. We consider the run typical, idle, and sleep modes in Table 12 for our experiment. WolfBot [16], which is a distributed mobile sensing platform, has ARMv7 based cortex processors.

Table 11. Frequency levels of the cortex-A7 core.

Parameter	Level 1	Level 2	Level 3	Level 4
Frequency (MHz)	800	1066	1333	1600
Run typical power (W)	3.3	3.6	4	4.9

Table 12. Power states of the cortex-A7 core.

States	Power (Watts)
Run Thermal	5.9
Run Typical	4.9
Idle	2.4
Deep Idle	0.07
Sleep	0.07

To confirm the scalability of the proposed algorithm, we change the number of available processors within the range 8–32. Then, we use the Emberson procedure to construct 100 task sets on each available processor. The total utilization of the task set is equal to 8, and the task has a utilization within 0.01–0.99. The period of each task is evenly distributed within 10–150 and simulated for 1000 system units.

4.2. Experiment Results and Analysis

Table 13 shows the platform type and power management technique of the algorithm to be simulated. The algorithm's platform type is called "non-uniform" when the associated frequency of each processor is independently adjustable, and is called "uniform" when it can change all the frequencies at a constant rate when scaling the frequency of the processor. It is possible for each processor among the uniform multi-processors to operate at a different frequency. A job has a different execution time depending on which processor is allocated. These platforms are otherwise called "unrelated".

Table 13. Summary of the energy-efficient scheduling algorithms.

Algorithm Name	Platform Type	Power Management
PCG	Uniform	-
Uniform-DPM (proposed)	Uniform	DPM
GMF	Non-uniform	SVFS
Independent RT-SVFS	Non-uniform	SVFS
Uniform RT-SVFS	Uniform	SVFS

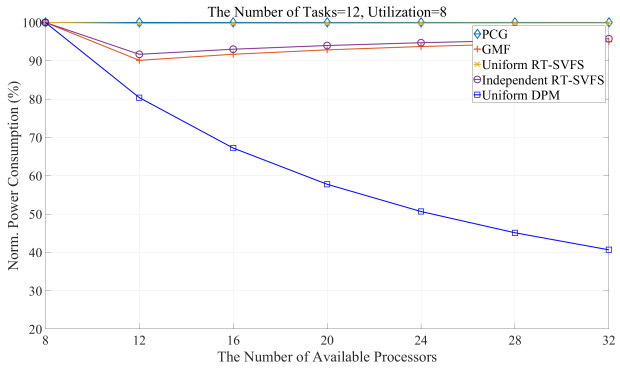
Figure 6 shows the power efficiency obtained by simulating the five algorithms mentioned in Table 13 while varying the number of available processors and the number of tasks. We implement our proposed algorithm as well as the following models: PCG, the original uniform algorithm without any power management [29]; Uniform-DPM, our proposed scheduling algorithm for DPM-embedded uniform multi-processors; GMF [31]; Independent RT-SVFS [30]; and Uniform RT-SVFS [30]. The x-axis of Figure 6 represents the number of available processors, and the y-axis represents the normalized power consumption (NPC). The power consumption consumed by the PCG is measured by the reference consumption and the power consumption rate of each algorithm. Figure 6 show the results when the number of tasks composing a task set is 12, 16, 20, and 24, respectively. All of the algorithms to be simulated is global optimal scheduling. Thus, since the total utilization of the task set used in the simulation is fixed at 8, the power efficiency of all algorithms shows 100% energy consumption in all scheduling using eight processors. As shown in Figure 6, the GMF and RT-SVFS algorithms change the power efficiency according to the number of tasks, while the proposed algorithm, Uniform-DPM, consumes the same a mount of power. This is because they always generate the same idle time. In addition, in the case of many available processors, the proposed algorithm shows high power efficiency by preventing unnecessary processor activation and idle time fragmentation, and by preventing frequent state transitions of the processor. GMF and independent RT-SVFS have similar power efficiencies because they determine the frequency of each processor independently. GMF finds a global optimal solution in the search spaces, but not Independent-SVFS. Thus, GMF is better than Independent-SVFS, as shown in Figure 6. Uniform RT-SVFS adjusts the frequency of all processors to a certain ratio, so if the number of tasks is small, the energy efficiency is not good because the work can be concentrated on some processor and the frequency of the processor cannot be lowered. However, as the number of tasks increases, the number of tasks can be divided and processed simultaneously by multiple processors, which can reduce the frequency of the processor. Tables 14 and 15 show the energy efficiency characteristics of this proposed algorithm. Table 14 shows that Uniform-DPM always shows constant energy efficiency regardless of the number of tasks. Table 15 shows that the energy efficiency increases as the number of processors increases.

Table 14. Summary of the experimental results by varying the number of tasks.

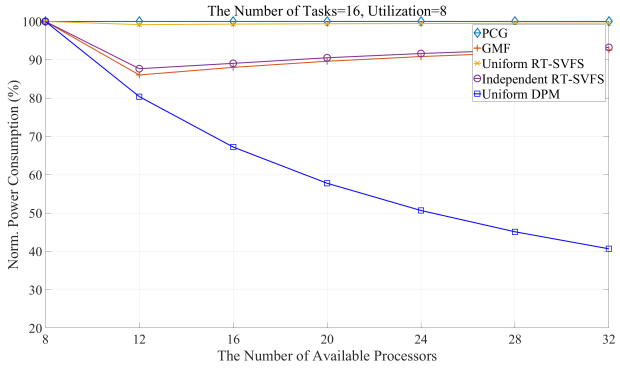
		Saved Norm. Power Consumption (%)			
# of Processors	**# of Tasks (Total Untilization)**	**Uniform-DPM**	**GMF**	**Independent RT-SVFS**	**Uniform RT-SVFS**
12	12 (8)	19.6	9.9	8.3	0.3
12	16 (8)	19.6	14	11	0.8
12	20 (8)	19.6	16.4	14.6	1.7
12	24 (8)	19.6	18.6	17	3.6

Table 15. Summary of the experimental results by varying the number of uniform processors.

		Saved Norm. Power Consumption (%)			
# of Processors	**# of Tasks (Total Untilization)**	**Uniform-DPM**	**GMF**	**Independent RT-SVFS**	**Uniform RT-SVFS**
12	24 (8)	19.6	18.4	17.1	3.6
16	24 (8)	32.8	17.6	16.4	3.3
20	24 (8)	42.2	15	14.3	2.7
24	24 (8)	49.4	14.5	12.9	2.5
28	24 (8)	54.9	12.1	11.6	2.2
32	24 (8)	59.3	11	10.6	2

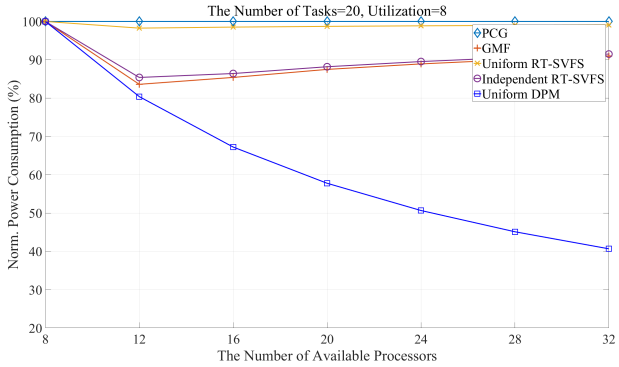


(**a**)

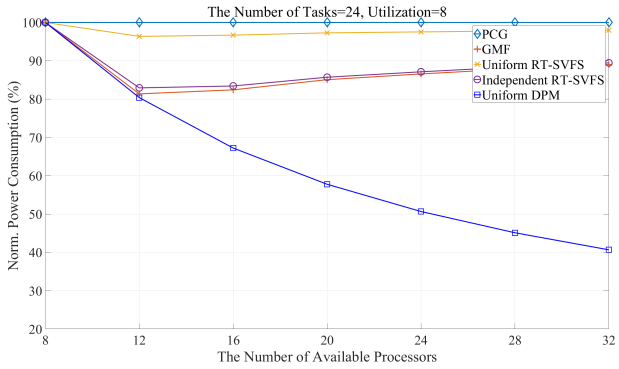


(**b**)

Figure 6. *Cont.*



(**c**)



(**d**)

Figure 6. Comparing the energy consumption of an energy-efficient approach while varying the number of tasks: (**a**) 12; (**b**) 16; (**c**) 20; and (**d**) 24.

5. Conclusions and Future Works

The lifetime of WSNs is closely related to the management of sensor nodes operating at limited energy. In this paper, we propose a power management method for sensor nodes supporting DPM-enabled uniform multi-processors. In the proposed approach, the selection of processors to process a set of tasks and the assignment of tasks to the selected processors have been proposed in terms of energy efficiency. In addition, we implement a simulator to measure the power consumption of various scheduling algorithms. The experimental results show that the proposed algorithms provide better scalability to the number of available processors than DVFS-based approaches. Currently, our proposed algorithms can handle periodic tasks with implicit deadlines. In future work, we plan to extend our algorithms to handle sporadic tasks with time constraint. We are very interested in combining the DVFS and DPM approaches for T-L plane abstraction as well. In addition, studies on trade-offs between the power usage and computational complexity, as well as performance evaluations on overloaded situations, would be interesting problems for potential future research.

Acknowledgments: This work was supported by the National Research Foundation of Korea (NRF) grant (NRF-2017R1D1A1B03029552, NRF-2017R1E1A1A01075803) funded by the Korea government (MSIP).

Author Contributions: Youngmin Kim and Chan-Gun Lee conceived and developed the algorithm; Ki-Seong Lee performed the experiments; Youngmin Kim and Ki-Seong Lee analyzed the data; Youngmin Kim and Chan-Gun Lee verified the results and finalized the paper.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Lymberopoulos, D.; Priyantha, N.B.; Zhao, F. mPlatform: A reconfigurable architecture and efficient data sharing mechanism for modular sensor nodes. In Proceedings of the 6th International Conference on Information Processing in Sensor Networks, Cambridge, MA, USA, 25–27 April 2007.
2. Mittal, S. A survey of techniques for improving energy efficiency in embedded computing systems. *Int. J. Comput. Aided Eng. Technol.* **2014**, *6*, 440–459.
3. Tutuncuoglu, K.; Yener, A. Optimum transmission policies for battery limited energy harvesting nodes. *IEEE Trans. Wirel. Commun.* **2012**, *11*, 1180–1189.
4. Zhang, Y.; He, S.; Chen, J.; Sun, Y.; Shen, X.S. Distributed sampling rate control for rechargeable sensor nodes with limited battery capacity. *IEEE Trans. Wirel. Commun.* **2013**, *12*, 3096–3106.
5. Tan, Y.K.; Panda, S.K. Energy harvesting from hybrid indoor ambient light and thermal energy sources for enhanced performance of wireless sensor nodes. *IEEE Trans. Ind. Electron.* **2011**, *58*, 4424–4435.
6. Wang, Y.; Liu, X.; Yin, J. Requirements of quality of service in wireless sensor network. In Proceedings of the International Conference on Systems and International Conference on Mobile Communications and Learning Technologies (ICN/ICONS/MCL 2006), Morne, Mauritius, 23–29 April 2006.
7. Krishnarnurthy, R.; Alvandpour, A.; De, V.; Borkar, S. High-performance and low-power challenges for sub-70 nm microprocessor circuits. In Proceedings of the IEEE Custom Integrated Circuits Conference, Orlando, FL, USA, 15 May 2002.
8. Venkatesh, G.; Sampson, J.; Goulding, N.; Garcia, S.; Bryksin, V.; Lugo-Martinez, J.; Swanson, S.; Taylor, M.B. Conservation cores: Reducing the energy of mature computations. In *ACM SIGARCH Computer Architecture News*; ACM: New York, NY, USA, 2010; pp. 205–218.
9. Esmaeilzadeh, H.; Blem, E.; St Amant, R.; Sankaralingam, K.; Burger, D. Dark silicon and the end of multicore scaling. In *ACM SIGARCH Computer Architecture News*; ACM: New York, NY, USA, 2011; pp. 365–376.
10. Smarr, L. Project Greenlight: Optimizing cyber-infrastructure for a carbon-constrained world. *Computer* **2010**, *43*, 22–27.
11. McLurkin, J.; Lynch, A.J.; Rixner, S.; Barr, T.W.; Chou, A.; Foster, K.; Bilstein, S. A low-cost multi-robot system for research, teaching, and outreach. In *Distributed Autonomous Robotic Systems*; Springer: New York, NY, USA, 2013; pp. 597–609.
12. Mondada, F.; Bonani, M.; Raemy, X.; Pugh, J.; Cianci, C.; Klaptocz, A.; Magnenat, S.; Zufferey, J.C.; Floreano, D.; Martinoli, A. The e-puck, a robot designed for education in engineering. In Proceedings of the 9th Conference on Autonomous Robot Systems and Competitions, Castelo Branco, Portugal, 7 May 2009.
13. Bonani, M.; Longchamp, V.; Magnenat, S.; Rétornaz, P.; Burnier, D.; Roulet, G.; Vaussard, F.; Bleuler, H.; Mondada, F. The marXbot, a miniature mobile robot opening new perspectives for the collective-robotic research. In Proceedings of the 2010 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2010), Taipei, Taiwan, 18–22 October 2010.
14. Brutschy, A.; Pini, G.; Decugniere, A. *Grippable Objects for the Foot-Bot*; IRIDIA Technical Report, Technical Report TR/IRIDIA/2012-001; Université Libre de Bruxelles: Brussels, Belgium, 2012.
15. Chen, P.; Ahammad, P.; Boyer, C.; Huang, S.I.; Lin, L.; Lobaton, E.; Meingast, M.; Oh, S.; Wang, S.; Yan, P.; et al. CITRIC: A low-bandwidth wireless camera network platform. In Proceedings of the Second ACM/IEEE International Conference on Distributed Smart Cameras (ICDSC 2008), Stanford, CA, USA, 7–11 September 2008.
16. Betthauser, J.; Benavides, D.; Schornick, J.; O'Hara, N.; Patel, J.; Cole, J.; Lobaton, E. WolfBot: A distributed mobile sensing platform for research and education. In Proceedings of the 2014 Zone 1 Conference of the American Society for Engineering Education (ASEE Zone 1), Bridgeport, CT, USA, 3–5 April 2014.
17. Chung, H.; Kang, M.; Cho, H.D. Heterogeneous Multi-Processing Solution of Exynos 5 Octa with ARM® big. LITTLE™ Technology. Samsung White Paper. 2012. Available online: https://www.arm.com/files/pdf/Heterogeneous_Multi_Processing_Solution_of_Exynos_5_Octa_with_ARM_bigLITTLE_Technology.pdf (accessed on 31 October 2017).
18. Kamdar, S.; Kamdar, N. big. LITTLE architecture: Heterogeneous multicore processing. *Int. J. Comput. Appl.* **2015**, *119*, 1.
19. Youngmin, K.; Ki-Seong, L.; Byunghak, K.; Chan-Gun, L. TL Plane Based Real-Time Scheduling Using Dynamic Power Management. *IEICE Trans. Inf. Syst.* **2015**, *98*, 1596–1599.

20. Kim, Y.; Lee, K.S.; Pham, N.S.; Lee, S.R.; Lee, C.G. TL Plane Abstraction-Based Energy-Efficient Real-Time Scheduling for Multi-Core Wireless Sensors. *Sensors* **2016**, *16*, 1054.
21. Jan, M.R.; Anantha, C.; Borivoje, N. *Digital Integrated Circuits—A Design Perspective*; Pearson Publishing: London, UK, 2003.
22. Chen, G.; Huang, K.; Knoll, A. Energy optimization for real-time multiprocessor system-on-chip with optimal DVFS and DPM combination. *ACM Trans. Embed. Comput. Syst.* **2014**, *13*, 111.
23. Carpenter, J.; Funk, S.; Holman, P.; Srinivasan, A.; Anderson, J.H.; Baruah, S.K. A Categorization of Real-Time Multiprocessor Scheduling Problems and Algorithms In *Handbook on Scheduling Algorithms, Methods, and Models*; Chapman Hall/CRC: Boca Raton, FL, USA, 2004; Volume 19, pp. 1–30.
24. Holman, P.; Anderson, J.H. Adapting Pfair scheduling for symmetric multiprocessors. *J. Embed. Comput.* **2005**, *1*, 543–564.
25. Anderson, J.H.; Srinivasan, A. Pfair scheduling: Beyond periodic task systems. In Proceedings of the 2000 Seventh International Conference on Real-Time Computing Systems and Applications, Cheju Island, Korea, 12–14 December 2000.
26. Levin, G.; Funk, S.; Sadowski, C.; Pye, I.; Brandt, S. DP-FAIR: A simple model for understanding optimal multiprocessor scheduling. In Proceedings of the 2010 22nd Euromicro Conference on Real-Time Systems (ECRTS), Brussels, Belgium, 6–9 July 2010.
27. Cho, H.; Ravindran, B.; Jensen, E.D. An optimal real-time scheduling algorithm for multiprocessors. In Proceedings of the 27th IEEE International Real-Time Systems Symposium (RTSS'06), Rio de Janeiro, Brazil, 5–8 December 2006.
28. Funk, S.H.; Meka, A. U-LLREF: An optimal scheduling algorithm for uniform multiprocessors. In Proceedings of the 9th Workshop on Models and Algorithms for Planning and Scheduling Problems, Abbey Rolduc, The Netherlands, 29 June–3 July 2009.
29. Chen, S.Y.; Hsueh, C.W. Optimal dynamic-priority real-time scheduling algorithms for uniform multiprocessors. In Proceedings of the Real-Time Systems Symposium, Barcelona, Spain, 30 November–3 December 2008.
30. Funaoka, K.; Kato, S.; Yamasaki, N. Energy-efficient optimal real-time scheduling on multiprocessors. In Proceedings of the 2008 11th IEEE International Symposium on Object Oriented Real-Time Distributed Computing (ISORC), Orlando, FL, USA, 5–7 May 2008.
31. Moreno, G.A.; De Niz, D. An optimal real-time voltage and frequency scaling for uniform multiprocessors. In Proceedings of the 2012 IEEE 18th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA), Seoul, Korea, 19–22 August 2012.
32. Funaoka, K.; Takeda, A.; Kato, S.; Yamasaki, N. Dynamic voltage and frequency scaling for optimal real-time scheduling on multiprocessors. In Proceedings of the International Symposium on Industrial Embedded Systems (SIES 2008), Le Grande Motte, France, 11–13 June 2008.
33. Horvath, E.C.; Lam, S.; Sethi, R. A level algorithm for preemptive scheduling. *J. ACM* **1977**, *24*, 32–43.
34. Funk, S.; Goossens, J.; Baruah, S. On-line scheduling on uniform multiprocessors. In Proceedings of the 22nd IEEE Real-Time Systems Symposium (RTSS 2001), London, UK, 3–6 December 2001.