J Supercomput (2012) 62:967–988
DOI 10.1007/s11227-012-0771-0

Energy-efficient tasks scheduling algorithm for real-time multiprocessor embedded systems

Hwang-Cheng Wang · Isaac Woungang · Cheng-Wen Yao · Alagan Anpalagan · Mohammad S. Obaidat

Published online: 26 April 2012

Abstract In recent years, applications like multimedia, video and audio stream communications, 3D movies, to name a few, have spurred the proliferation of multiprocessor systems, particularly for real-time embedded systems. However, the complex architecture and heavy computing demands of such systems increase power consumption. Therefore, energy conservation has become a critical issue. In this paper, we propose a novel tasks scheduling algorithm for real-time multiprocessor systems. The algorithm works by reducing the workload in high speed processors with the aid of task migration so that the entire system can switch to low speed/low voltage as soon as it can reduce power consumption. The overhead of transitioning to low voltage is also analyzed and used as a criterion to determine whether the transition is beneficial. The effect of important parameters such as task granularity on the performance is also

H.-C. Wang · C.-W. Yao
Department of Electronic Engineering, National I-Lan University, I-Lan, Taiwan, ROC

H.-C. Wang
e-mail: hcwang@niu.edu.tw

C.-W. Yao
e-mail: appgra@hotmail.com

I. Woungang
Department of Computer Science, Ryerson University, Toronto, ON, Canada
e-mail: iwoungan@scs.ryerson.ca

A. Anpalagan
Department of Electrical and Computer Engineering, Ryerson University, Toronto, ON, Canada
e-mail: alagan@ee.ryerson.ca

M.S. Obaidat (✉)
Department of Computer Science and Software Engineering, Monmouth University, West Long Branch, NJ 07764, USA
e-mail: obaidat@monmouth.edu

investigated, and simulation results based on realistic processor power consumption models are shown to be promising.

Keywords Embedded system · Multiprocessor · Real time · Dynamic voltage–frequency scaling (DVFS) · Speed rate · Tasks scheduling · Task migration

1 Introduction

Embedded system applications, which typically integrate various types of multimedia applications, have become increasingly popular. The addition of digital signal processor (DSP) and multiprocessor improves the performance of new embedded systems. The enhanced hardware facilitates the processing of a large number of complex 3D graphics. Thus, a modern system meets the computing needs by using multiprocessor and incorporates multithreading for high performance and lower operating temperature. In such systems, the issue of efficient schedule of tasks in order to reduce the power consumption is an important one [1, 2]. The power management mechanism and hardware support can be traced back to 1989 when Intel unveiled CPUs that could work at lower frequencies when the system was idle. Moreover, some components can be shut down to reduce the power consumption when they are idle [3]. Together, hardware support and power saving mechanisms play an important role in reducing the power consumption.

In a real-time system, it is imperative that the system completes the execution of the task set within a given time limit. Therefore, Dynamic Voltage–Frequency Scaling (DVFS) algorithms have to perform scheduling based on the workload of each processor. The DVFS mechanism is a trade-off between performance and battery life. Since a system does not need high performance all the time, the operating frequency of the processors can be lowered to reduce the power consumption.

High-performance scheduling in a real-time system allows the tasks to be completed within a time limit and reduces the switching between states. A number of scheduling methods have been developed for embedded systems [4]. The earliest deadline first (EDF) and the least laxity first (LLF) methods [4] belong to the class of dynamic scheduling algorithms, while the rate monotonic (RM) [4] is a static scheduling algorithm. Dynamic scheduling determines the order of tasks execution at run-time whereas in static scheduling, the priority is determined at the compilation time. The EDF scheduling algorithm can guarantee the timely completion of all tasks provided the total utilization of a processor be less than 1. On the other hand, the RM scheduling algorithm can guarantee that all the deadlines will be met if the CPU utilization is below $N(2^{1/N} - 1)$, where N is the number of tasks to be scheduled. Both the EDF and RM algorithms assume that the computing time and deadlines are known in advance. Moreover, the deadline of each task is assumed to be equal to its period. The LLF method can also guarantee a processor utilization of 100 %; however, it does not assume that the rates of occurrences of tasks are known beforehand. Thus, this method is suitable for aperiodic tasks, and henceforth, it is suitable for use on homogeneous multiprocessor embedded systems since no processor affinity is assumed.

This paper introduces a scheduling strategy based on DVFS. More precisely, after a task set is assigned to the processors in a multiprocessor system, a task can be migrated to another processor based on the workload of the processors. The decision is based on the actual execution time rather than on the worst-case execution time. Practical processor power models are used to assess the performance of the proposed algorithm, showing that it can outperform the existing methods in terms of power saving.

DVFS involves transition between voltages, which incurs overhead in both time and power consumption. It is shown that under certain circumstances, it is beneficial to refrain from voltage transition when the overhead is taken into account. The criterion for the decision is derived and a new algorithm is devised accordingly. Simulations are conducted to confirm the benefits of our approach compared to some existing ones.

The rest of the paper is organized as follows. Section 2 describes some background and related work on DVFS and multiprocessor scheduling. Section 3 introduces the system model and our new DVFS algorithms. In Sect. 4, our simulation results are presented. Finally, Sect. 5 concludes the paper.

2 Background and related work

Dynamic Voltage Scaling (DVS) [5–8] and Dynamic Frequency Scaling (DFS) are two well-known approaches that have been used to reduce power consumption [9]. In practice, supply voltage and operation frequency are closely intertwined. Therefore, the term DVFS is frequently used to reflect this fact and will be used in this paper. Research in recent years has also expanded to green energy, including the conversion of solar, wind, and tide energies into electricity [10–12].

In general, an embedded system needs to be in high-performance mode only for a small fraction of time. The DVFS mechanism allows the system to lower the operation frequency. The power consumption of a processor fabricated by the CMOS technology can be divided into two categories: dynamic and static. When the processor is processing the data, the required power consumption is called the dynamic power, denoted by P_d, which is the power consumed during a time unit. P_d can be obtained as in [13], i.e.

$$P_d = C_L \times V_{DD}^2 \times f \tag{1}$$

where C_L is the load capacitance, decided at the time of hardware design and manufacturing, V_{DD} is the supply voltage and f the clock frequency.

From an energy-saving perspective, adjusting the supply voltage and the operating frequency is one of the possible ways of reducing the dynamic power consumption.

Static power is the power consumption when the processor does not perform any computing. It arises from the leakage currently incurred by the integrated circuit transistors that do not have sufficiently high insulation resistance. Therefore, the static power consumption P_s is determined by

$$P_S = I_l \times V_{DD} \tag{2}$$

where I_l is the current leakage and V_{DD} the supply voltage.

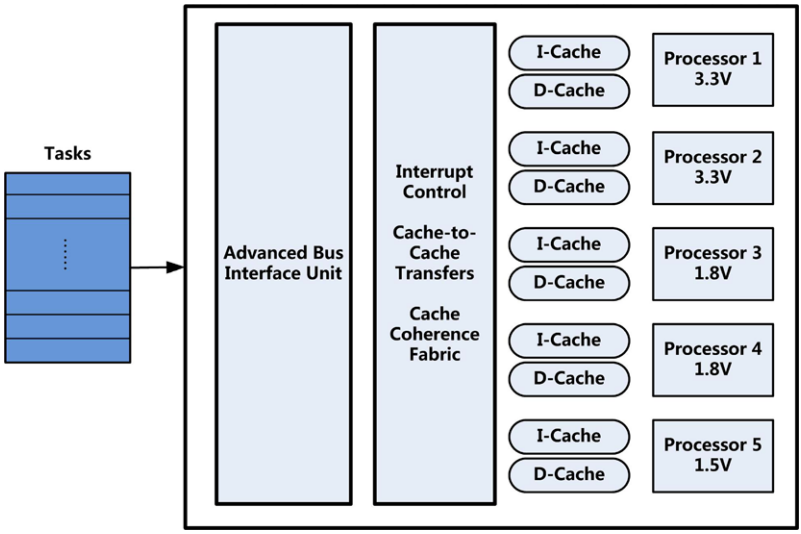

Fig. 1 A multiprocessor architecture with different supply voltages

The DVFS technique is a trade-off between performance and power saving in low-power system design. But DVFS also incurs additional overhead in energy consumption and execution time. If a system switches voltage excessively, the DVFS overhead may impair the power saving and real-time performance. Recently, much research has been devoted to the modeling of DVFS energy consumption. DVFS overhead is mainly caused by a DC–DC converter. The model proposed by Burd and Brodersen [14] considers the effect of DC–DC converter and voltage-controlled oscillator (VCO) clock generator. Based on the model, when the voltage is changed from V_1 to V_2, the transition time T_t and energy overhead E_t are given respectively by

$$T_t = \frac{2C_L}{I_{\max}} \times |V_1 - V_2|, \tag{3}$$

$$E_t = (1 - \eta) \times C_L \times \left|V_1^2 - V_2^2\right|, \tag{4}$$

where C_L is the capacitance, $I_{\max}$ the maximum current and η the efficiency of the DC–DC converter. The factor of 2 is included to account for the fact that the current is pulsed in a triangular waveform in the DVFS architecture.

Figure 1 shows an embedded system with multiple voltage levels and multiprocessors. A salient feature of a multiprocessor system is that each processor can have its own supply voltage, operation speed, I-cache, and D-cache. A multiprocessor system can shorten the computation time, thereby saving the power consumption and achieving a high performance.

Multiprocessors can be divided into two broad classes: homogeneous and heterogeneous, as shown in Fig. 2. A system consisting of multiple processors which execute different instruction sets is called heterogeneous. On the contrary, if all the processors in a system execute the same instruction set, then it is homogeneous. Homogeneous multiprocessors can also incorporate special-purpose processors such as

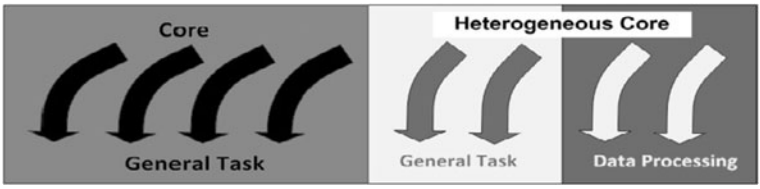

Fig. 2 Homogeneous and heterogeneous multiprocessor systems

digital signal processors (DSPs) to improve the performance of the system and reduce the workload of the main processors. A multiprocessor system can balance the load among processors using task migration at run-time [15].

Tasks scheduling in a multiprocessor system is more complex than in a single-processor system because it has to consider additional conditions in order to meet the real-time requirements. Typically, multiple voltages imply different operation speeds for each voltage. Besides, the system has to consider real-time issues. For instance, let us assume a system has two processors and two voltages, v_l and v_h. Let us also assume that there are N tasks $\tau_1, \tau_2, \ldots, \tau_N$ with deadlines $D_1, D_2, \ldots, D_N$ that need to be scheduled. All the tasks arrive at time 0 and have a common deadline T. In the two-processor system, we have $D_1 + D_2 + \cdots + D_N = 2T$. If the system is not to miss the deadline, both processors have to operate at voltage v_h between 0 and T. The tasks scheduling divides the tasks into two subsets, such that the sum of the execution time of tasks in each subset is T. This is referred to as the subset sum problem, which has been shown to be an NP-complete [16].

The two predominant approaches used for the scheduling of multiprocessor real-time systems are partitioned and global [17]. In the partitioned scheme, each task is assigned statically to one processor offline and migration is not allowed. In the global scheme, tasks can be dispatched to and executed on any available processor. The necessary and sufficient schedulability test for a task set according to the EDF scheduling algorithm requires that the sum of the worst-case utilizations be less than 1 for each processor, i.e., $C_1/P_1 + C_2/P_2 + \cdots + C_{N_i}/P_{N_i} < 1$, where P_i is the period and C_i is the computation time of the task on processor i [18].

In [19], Yao et al. proposed a simple model of job scheduling which captures several important factors for minimizing the energy consumption. It was assumed that a number of jobs arriving at a single processor with variable speeds are to be executed between the arrival time and deadline of each job. First, an offline algorithm which minimizes the energy consumption was established. Then some online algorithms were devised and their competitive ratios in relation to the optimal algorithm were analyzed through a theoretical analysis.

In [20], Anderson and Baruah showed that implementations of real-time systems with multiple processors are often more energy-efficient than their single-processor counterparts. The authors observed that there is a trade-off between energy efficiency and computing capacity that can be guaranteed as the number of processors is increased. Based on the utilization bound for fixed-priority global scheduling as well as partitioned EDF scheduling, they formulated a strategy to determine the number of processors that should be activated for a given set of periodic tasks with known maximum utilization and cumulative utilization. It was assumed that all the processors

comprising a multiprocessor system operate on the same supply voltage and possess the same computing capacity, giving rise to a system with identical processors. However, no DVFS was considered in the paper.

In [21], the co-scheduling of DVFS and non-DVFS processors was first examined. Specifically, a heterogeneous system consisting of a DVFS-capable processing element (PE) and a non-DVFS-capable processing element was considered, with tasks allowed to migrate from the former to the latter. Different characteristics of the non-DVFS PE were assumed (based on whether the power consumption is workload-independent or workload-dependent) and various approximation scheduling strategies were developed accordingly. Many interesting results were reported.

In [22], the system-level DVFS for energy saving was investigated, and a general power model that encompasses both frequency-dependent and non-frequency-dependent components was proposed. The former includes CPU and the latter may include off-chip components such as memory and IO devices. Effect of on-chip and off-chip access patterns on the power consumption was also taken into account. The problem of minimizing energy consumption with periodic tasks was formulated as a nonlinear optimization problem and a Kuhn–Tucker multiplier-based approach was applied to derive an optimal solution.

In [23], Pillai and Shin proposed a cycle-conserving DVFS for an EDF or RM system requiring real-time deadline guarantees and showed that their proposed approach can result in power saving. However, in their approach, the cycle-conserving DVFS was implemented in a single-processor system.

Following the same trend, Lin and Wang [24] proposed a tasks scheduling scheme referred to as Workload-Aware Scheduling (WAS), whose aim was to balance the workload in a multiprocessor system. However, a continuous operation speed was used in their scheme, making it less practical.

In [25], Zeng et al. introduced their so-called Adaptive Minimal Bound First-Fit (AMBFF) algorithm that schedules a real-time task set based on real power consumption models. In their algorithm, the operation speed is determined by the worst-case task execution time. It was also pointed out [25] that although many other studies have used the task worst-case execution time, this may end up degrading the DVFS performance.

Unlike previous works, this paper proposes a Workload-Aware Task Migration scheduling algorithm (our so-called WATM), for homogeneous real-time multiprocessor systems. WATM is a global scheduling algorithm, which considers practical processor power models to improve the workload balance among processors. Moreover, the task migration in WATM is based on the actual number of processor cycles or execution time at run-time. The key features of our WATM algorithm are as follows:

- Power-saving: WATM takes into account the relationship between the practical processor power models and the task utilization in determining the tasks scheduling and operation speed.
- Task migration: WATM migrates a task from one processor to another with a lower operation speed than the current one, which may improve the power saving.
- DVFS overhead reduction: When using DVFS, a processor incurs two types of overhead: time and energy, which can be quantified. Moreover, our modified WATM algorithm (so-called WATM + RTO—for WATM with Reduction of Time

Overhead) avoids excessive transitions in the operation speeds, thereby, reducing the time and energy overhead associated with DVFS.

3 Proposed scheduling algorithm

3.1 System model

Many embedded systems are usually real-time systems. In this paper, we consider a preemptive real-time system and we assume that the EDF scheduling is used in which the tasks are periodic and independent in execution. In a real-time system, there is a task set $\Lambda = \{r_1, r_2, \ldots, r_N\}$ of N real-time tasks that need to be executed periodically. In the treatments of these schedulers, we assume that the DVFS and task migration overheads are negligible. Each task is denoted by a 3-tuple $\tau_i = \langle P_i, W_i, D_i \rangle$ with attributes P_i standing for period, W_i for worst-case execution time (WCET), and D_i for deadline which is assumed to be equal to P_i. The utilization of a task μ_i is obtained as follows:

$$\mu_i = \frac{W_i}{D_i}, \tag{5}$$

with each μ_i satisfying the condition

$$\mu_i \leq MTU \quad \text{for } i = 1, 2, \ldots, N, \tag{6}$$

where MTU is the maximal task utilization which is a system-wide parameter.

Table 1 gives the power consumptions for XScale and PowerPC. XScale is a microprocessor core implementation of the ARMv5 architecture by Intel and Marvell and has been in a wide range of embedded systems. PowerPC CPUs also have RISC architecture and are popular as embedded system and high-performance processors. The two processors both allow for DVFS and their distinct characteristics serve to illustrate the difference in power saving achieved by the proposed algorithm. Shown in the table are the operation frequencies and the corresponding voltages and power consumptions. The speed rate for a particular frequency is obtained by dividing the frequency by the highest frequency available on a processor:

$$S_i = f_i / f_L, \quad f_i \in \{f_1, \ldots, f_L \mid f_1 < \cdots < f_L\}. \tag{7}$$

Our proposed algorithm also uses the relationship between the speed rates depicted in Table 1 and the utilization of the tasks, in order to choose a suitable operation frequency. The choice of the operation frequency affects the power consumption of a processor directly. The power consumption for a processor operating at speed rate S_i is the weighted sum of the run mode power $P(S_i)$ and the idle power P_{idle}:

$$\text{Power} = \frac{U_{\text{tot}}}{S_i} P(S_i) + \left(1 - \frac{U_{\text{tot}}}{S_i}\right) P_{\text{idle}} \tag{8}$$

where U_{tot} is the total utilization of all the tasks on a processor. The values of $P(S_i)$ and P_{idle} are also given in Table 1. It should be noted that the value of P_{idle} is independent of the selected speed rate.

Table 1 Processor power models and parameters

Processor type	XScale [26]					PowerPC 405LP [27]			
Speed rate	0.15	0.4	0.6	0.8	1.0	0.1	0.3	0.8	1.0
Frequency (MHz)	150	400	600	800	1000	33	100	266	333
Voltage (V)	0.75	1.0	1.3	1.6	1.8	1.0	1.0	1.8	1.9
Run mode power (mW)	80	170	400	900	1600	19	72	600	750
Idle power (mW)	40 [28]					12			

3.2 Proposed scheduling algorithm

The rationale for developing our proposed scheduling algorithm (so-called Workload Adaptive Task Migration, or WATM) is the observation that in a multiprocessor system, utilizations falling within the same range of speed rates have the same corresponding power consumption. For instance, in XScale, the same run mode power of 400 mW is consumed when the utilization is 0.5 and 0.6 because the system operates at the same speed rate (i.e., 0.6) for the two values of utilization. By exploiting this at run-time, WATM can help reducing the power consumption.

WATM consists of two phases (Figs. 3 and 4). In Fig. 3, the procedures of the first phase are described. Key notations used in the algorithm are summarized in Table 2. The inputs to the WATM algorithm consist of the set of tasks to be scheduled, the utilization of each task, and the set of discrete speed rates available on the processors in increasing order. At first, the processors are set to the lowest speed rate. The first task is examined to see if it can be assigned to processor 1. If so, the assignment is made and the load on processor 1 is increased by that of the task. Otherwise, an attempt is made on processor 2 and the subsequent processors until one is found that can accommodate the task using the current speed rate. If no such processor can be found, the speed rate is pushed up and the search process starts over from processor 1. After task 1 is scheduled, task 2 is processed as described above. This process is repeated until all the tasks are scheduled or until failure is declared.

Through the simple schedulability test of the EDF scheduling, we get

$$\sum_{i \in TS(k)} u_i \leq S_k, \tag{9}$$

where S_k is the smallest value of speed rate on processor k which satisfies (9).

After phase 1 of the WATM algorithm is completed, the tasks allocated to each processor are sorted in descending order of utilization. This allows the first few tasks to be executed at high speed rates. Afterwards, the speed rates of the individual processors can be lowered as the utilization of tasks goes down. When DVFS is implemented, a system will slow down its operation speed when a task is finished and the workload is reduced. Many algorithms schedule the tasks based on WCET [29]. However, the execution of tasks typically takes less time than WCET under most circumstances. Therefore, WATM uses the actual execution time or processor cycles in the second phase to migrate tasks when a processor has completed the execution of a task. This gives more time for the system to switch to a lower speed rate with

```
Input M, N, L, u[1], u[2], ..., u[N], SR[1], SR[2], ..., SR[L]
Initialization
   sg = 1;
   TS[i] = φ for i = 1 to M
   UT[i] = 0 for i = 1 to M
   Sridx[i] = sg for i = 1 to M
for j = 1 to N
        k = 1;
   loop:
      if u[j] + U[k] < SR[sg]
         TS[k] = TS[k] ∪ {j};
         UT[k] = u[j] + UT[k];
         Sridx[k] = sg;
      else
         k = k + 1;
         if (k > M)
            sg = sg + 1;
            if (sg > L) return 'failure';
            k = 1;
         end
         goto loop
      end
end
```

Fig. 3 First phase of the WATM algorithm

Table 2 Notations used in Fig. 3

Notation	Meaning
M	Number of processors
N	Number of tasks
L	Number of discrete speed rates
TS	Subset of tasks assigned to a processor
UT	Total utilization on a processor
Sridx	Speed rate index on a processor
SR	Speed rates available on a particular processor type in a multiprocessor system

the goal to reduce the power consumption. This also allows the system to lower its operation frequency for the remaining tasks while attempting to meet the real-time requirement.

The second phase of the WATM algorithm is outlined in Fig. 4. It is performed after the execution of a task on a processor k is finished. Here, we consider whether the last task on the processor k can be migrated to another processor. If another processor, say processor j, has a slower operation speed than the processor under consideration (processor k) and if accepting the last task will not affect the operation speed of pro-

```
Input M, u[1], u[2], ... u[N], SR[1], SR[2], ..., SR[L], Sridx[1], Sridx[2], ...,
  Sridx[M]

repeat
  success = false;
  for j = 1 to M,
    if Sridx[j] > Sridx[k] and u[lask(k)] + UT[j] < SR[Sridx[j]]
      TS[k] = TS[k] − {last(k)};
      TS[j] = TS[j] ∪ {last(k)};
      success = true;
      break;
    end
  end
  if (success == false) return(last(k));
  last(k) = prev(last(k));
until last(k) = NULL;
choose minimum spt(k) which satisfies (9) based on tasks remaining on processor k
```

Fig. 4 Second phase of the WATM algorithm

cessor j, then the last task of processor k will be migrated to processor j. After doing so, the second last task on processor k will become the last task and the process is repeated. For convenience, suppose there is a function *last()* which returns the last task on the processor and a function *prev()* which returns the previous task of the one under consideration. In this way, the system can balance the workload among processors. Moreover, the operation speed of the current processor can be lowered earlier after some of the remaining tasks have been migrated to other processors. Thus, the second phase of WATM serves the dual purpose of balancing workload and saving power. A processor will choose its own new speed rate according to the reduced workload in the last step based on the criterion in (13).

Figure 5 illustrates the effect of the task migration for a system with two processors and 11 tasks. The horizontal line represents the speed rate of each processor. Before task 6 is migrated, the processors have chosen the speed rates 0.9 and 0.6, respectively. The processors maintain the speed rates 0.9 and 0.6 until tasks 1 and 7 are completed. After that, processor 2 has the capability of executing task 6. After task 6 migrates to processor 2, processor 1 can lower its speed rate from 0.9 to 0.6. Thus, power saving is accomplished by switching to a lower operation speed.

3.3 DVFS overhead and its reduction

In an actual situation, the DVFS overhead is closely related to the power saving and the real-time execution. First, we calculate the energy overhead of XScale using (3) for each of the voltage transitions, as shown in Table 3. On a prototype system, the typical parameter values are $C = 100$ μF, $I_{\mathrm{MAX}} = 1$ A and $\eta = 90$ %. From a power saving point of view, DVFS can help reducing the power consumption because the

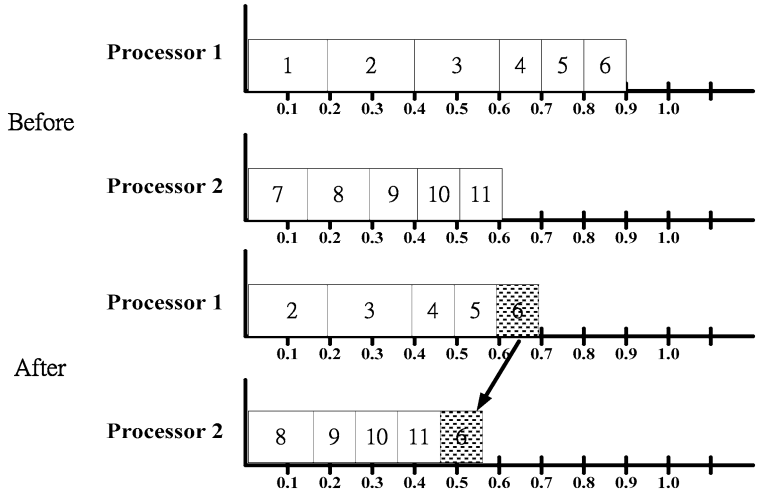

Fig. 5 Effect of task migration

Table 3 DVFS overhead for XScale processor

Voltage transition (V)	Energy overhead (μJ)	Time overhead (μs)
$1 \rightarrow 0.8$	3.6	40
$1 \rightarrow 0.6$	6.4	80
$1 \rightarrow 0.4$	8.4	120
$1 \rightarrow 0.15$	9.775	170
$0.8 \rightarrow 0.6$	2.8	40
$0.8 \rightarrow 0.4$	4.8	80
$0.8 \rightarrow 0.15$	6.175	130
$0.6 \rightarrow 0.4$	2	40
$0.6 \rightarrow 0.15$	3.375	90
$0.4 \rightarrow 0.15$	1.375	50

energy overhead is very low (under 10 μJ.) But for a real-time system, the time overhead can be an important parameter. Table 3 shows the time overhead of XScale. Similarly, the energy and time overhead results for the PowerPC are summarized in Table 4. From these tables, it can be observed that it is advantageous to reduce the frequency of DVFS.

However, lowering the frequency of voltage transitions should not hamper the reduction in power consumption. The data in Table 1 indicate that the power saving in relation to speed rate is smaller for the lowest speed rate than the others in both processors. Hence, the transition to that speed rate should be avoided. This can be accomplished by modifying the algorithm presented in Fig. 4. Indeed, some additional conditions are introduced which prevent the transition to the lowest speed rate when such a transition is unfavorable, yielding our so-called WATM + RTO (Reduction of Time Overhead) scheme. Criteria for determining when RTO should be performed will be given in Sect. 4. In order to reduce the time overhead, the cooperation be-

Table 4 DVFS overhead for PowerPC 405LP processor

Voltage transition (V)	Energy overhead (μJ)	Time overhead (μs)
$1 \rightarrow 0.8$	3.6	40
$1 \rightarrow 0.3$	9.1	140
$1 \rightarrow 0.1$	9.9	180
$0.8 \rightarrow 0.3$	5.5	100
$0.8 \rightarrow 0.1$	6.3	140
$0.3 \rightarrow 0.1$	0.8	40

tween WATM and RTO is used to reduce the power consumption and the number of transitions in the operation speed.

3.4 Time complexity of the WATM algorithm

The first phase of the WATM algorithm is a first-fit algorithm. For each task, WATM tries to place it on the first processor that can accommodate the task. If no such processor can be found, it increases the processor speed to the next higher level and tries again. If the system has M processors with N tasks to be allocated, and there are L discrete speeds, the time complexity is $O(MNL)$. The time taken by the sorting operation can be neglected because all the existing EDF algorithms also need to sort the tasks. In the second phase of the WATM, the processors are interrogated in sequence in order to determine where the last task on the processor under consideration should be migrated to. Therefore, the time complexity of the task migration is $O(M)$.

Task migration overheads are implementation-dependent. In this respect, the system architecture and communication paradigm play a major role. In distributed-memory multiprocessor systems, task migration involves the transfer of the task memory content, which may include the processor state, the user level, the kernel level context, and the address space. In [30], Acquaviva et al. showed that the migration at the middleware/OS level is feasible. The authors also improved the energy efficiency of multimedia applications on multiprocessor systems. Furthermore, migration overhead in terms of quality of service (QoS) can be hidden by the data buffer present in the inter-processor communication queues. In our work, the task migration overhead is neglected since the main focus is on the comparison of the task mapping policies as well as determining how the run-time task migration may result in energy consumption saving.

4 Performance evaluation

To evaluate the power saving performance of the proposed algorithm in a multiprocessor real-time system, we have developed a simulator in Java using the practical processor power consumption models described in Sect. 3. The simulator takes as input a task set comprising tasks independent of each other. Two parameters, namely, the average utilization and the maximal task utilization, were considered to investigate their effects on the performance. The simulator used randomly generated tasks

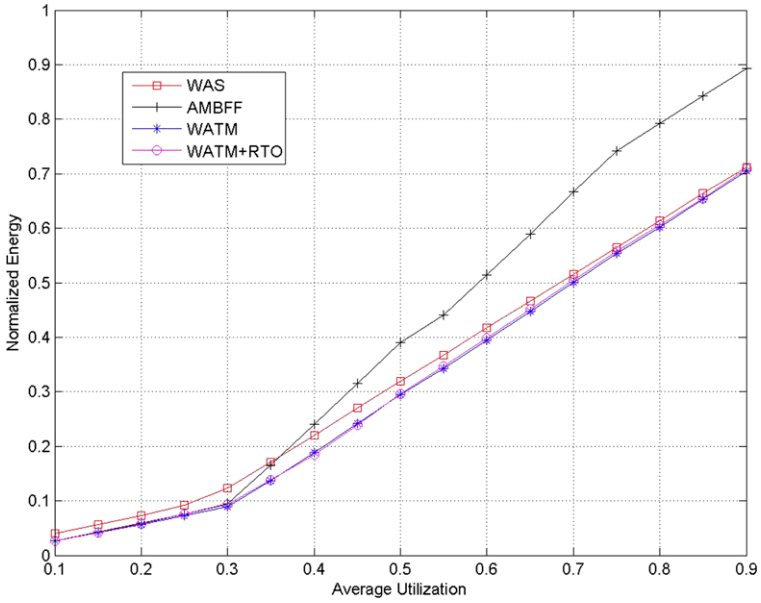

Fig. 6 PowerPC 405LP with MTU = 0.1

with the period of each task in the interval [2, 1000] and its utilization uniformly distributed in the range $[0.001, MTU]$. The total utilization of tasks was set within $(0.1, 0.9M)$. Each setting with an average utilization from 0.1 to 0.9 and a step size of 0.05 was repeated 100 times for $M = 8$.

In the sequel, we compare our WATM algorithm against the WAS [17] and the AMBFF algorithms [18] chosen as benchmarks, using the same simulation environment. It is well known that WAS can lead to more balanced workload than other algorithms. On the other hand, the AMBFF algorithm, which is based on the practical discrete speeds, is a static voltage-scaling algorithm which only performs voltage scaling once at the time of task assignment.

Figure 6 shows the normalized energy by dividing the energy consumption of the three algorithms by that for execution at full speed (i.e., no DVFS was performed) on PowerPC 405LP. All the three algorithms (WAS, AMBFF, and WATM) show similar performance when the average utilization is low. When the average utilization is high, the WATM algorithm outperforms the WAS and AMBFF algorithms in terms of power saving because of its capability to migrate tasks. The difference in power saving is significant when compared to the AMBFF scheme. Figure 7 shows that XScale can have greater power saving than PowerPC 405LP. This can be attributed to the fact that XScale has more discrete speed levels than PowerPC 405LP.

In Figs. 6 and 7 it can be observed that the WAS has poor power saving performance at low utilization compared to the WATM and AMBFF algorithms. This can be justified by the fact that the WAS does not consider the practical processor power models and static power consumption. But WAS which uses active DVFS exhibits good performance for high utilization. From the simulation results we observe the general trend of WAS as follows. As utilization increases gradually, WAS and

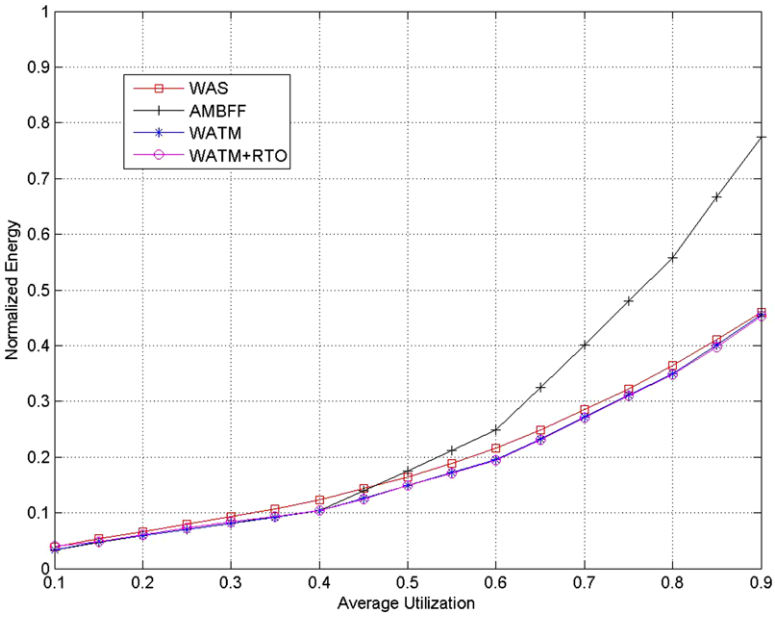

Fig. 7 XScale with MTU = 0.1

WATM demonstrate greater capability of power saving by DVFS. On the other hand, AMBFF does not perform DVFS at run-time, making its power saving performance worse than that of the WAS and WATM algorithms for high utilization. WATM uses task migration as well as active DVFS to reduce the power consumption and has sound performance in most situations.

The value of MTU can also affect the performance as illustrated in Figs. 8 and 9. For these two figures, the value of MTU is 0.5, whereas in Figs. 6 and 7, the MTU value is 0.1. MTU is closely related to the granularity (size) of individual tasks. It becomes increasingly difficult for a system to allocate and migrate tasks with large values of MTU or coarser task granularity. The performance of WAS and WATM is more severely deteriorated than that of the AMBFF. This can be justified by the fact that for small values of MTU, a processor can quickly complete the execution of tasks, allowing it to more quickly switch to lower speeds in WAS and WATM. On the other hand, when the MTU value is large, the switch to lower speeds is postponed, thus hampering the performance of the two algorithms.

In Fig. 10, the power consumption results were normalized with respect to the AMBFF algorithm in order to highlight the above observations. It can be observed that the performance of the WAS algorithm is relatively poor when the utilization is low. The difference between the WAS, WATM, and AMBFF algorithms dwindles when the average utilization is high and the MTU value is large.

When the average utilization of a system equals one of the discrete speed rates, the WATM algorithm shows a worse performance. This can be observed in Fig. 10 where the average utilization used is 0.8. The WATM algorithm migrates the tasks from a processor of high speed rate to other processors of low speed rate. If the aver-

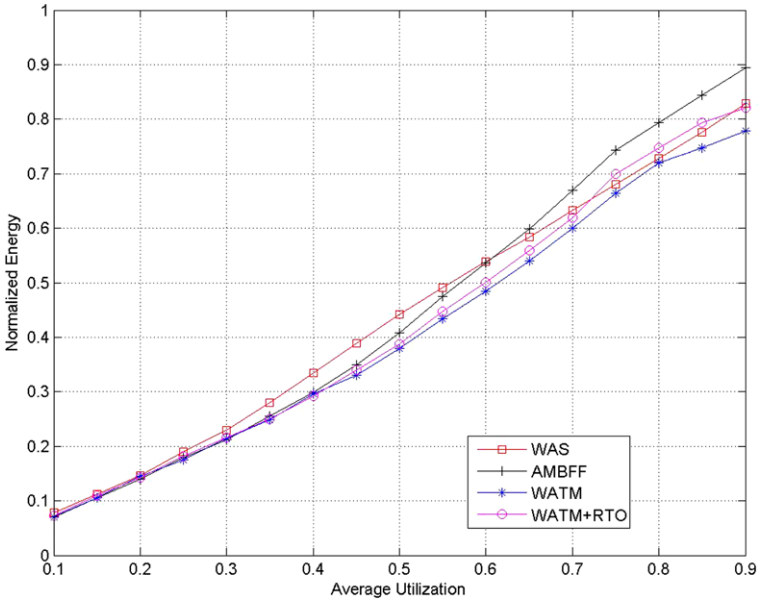

Fig. 8 PowerPC 405LP with MTU = 0.5

age utilization equals one of the discrete speeds, all of the processors will choose the same operation speed, nullifying the power-saving effort. With more discrete speeds, XScale is better suited for choosing the appropriate operation speed. Therefore, XScale also offers greater opportunities for migrating tasks in order to reduce the power consumption.

The performance of the WAS, AMBFF, WATM, and WATM + RTO algorithms in terms of the normalized energy when the DVFS overhead is taken into account is shown in Figs. 7–10. For instance, if the average utilization on one of the XScale processors has 0.4 under WATM + RTO, the power consumption using (8) is obtained as

$$\frac{0.4}{0.4} \times 170 + 0 = 170. \tag{10}$$

On the other hand, for the WATM algorithm, the power consumption is obtained as

$$\frac{0.15}{0.15} \times 80 + \frac{0.25}{0.4} \times 170 + 0 = 186.25. \tag{11}$$

Thus, the WATM + RTO algorithm outperforms the WATM algorithm on XScale. To justify this, let us assume two speed rates, $S_1 < S_2$, and task utilization $\mu = \mu_1 + \mu_2$. The condition to justify the transition from S_2 to S_1 is determined as

$$\frac{\mu}{S_2} P(S_2) > \frac{\mu_1}{S_1} P(S_1) + \frac{\mu_2}{S_2} P(S_2), \tag{12}$$

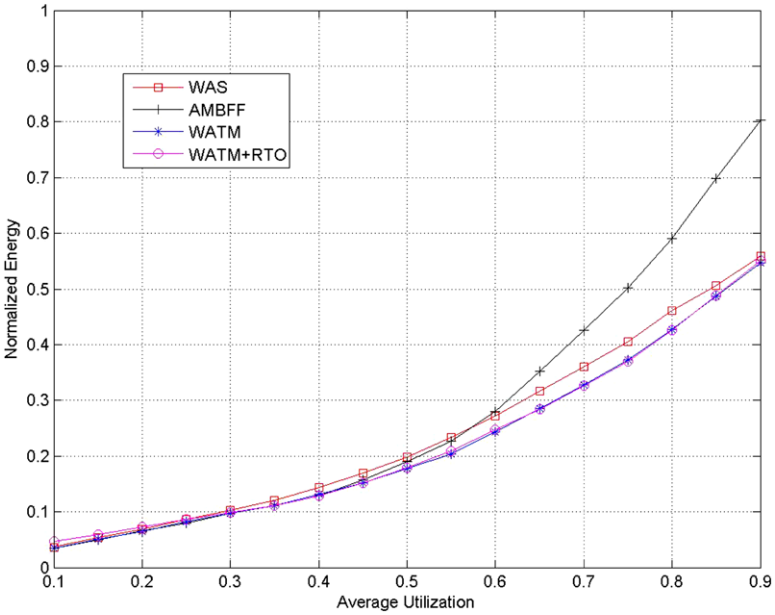

Fig. 9 XScale with MTU = 0.5

which is equivalent to

$$\frac{P(S_2)}{S_2} > \frac{P(S_1)}{S_1}. \tag{13}$$

For the lowest two speed rates 0.15 and 0.4 on XScale, the corresponding power consumption is 80 and 170 mW, respectively. Thus (12) is not satisfied and the RTO leads to a better performance. Now, considering the lowest two speed rates 0.1 and 0.3 on PowerPC 405LP, the corresponding power consumption is 19 and 240 mW, respectively. Thus (12) is satisfied and the RTO gives inferior results in power saving. These observations are reflected in Figs. 6 and 7. The difference in the normalized energy between WATM and WATM + RTO on PowerPC 405LP is obvious for MTU = 0.5.

The WAS algorithm will allocate a task to the processor of minimum workload in a system to balance the workload among processors. As a result, it tends to activate more processors than the AMBFF and WATM schemes, as reflected in Table 5. For low average utilization, it is not necessary to activate all the processors in a system. The WATM and AMBFF schemes consider the relationship between utilization and discrete speeds. In Fig. 11, WAS allocates a task to the other processor on XScale to balance the workload. WAS and WATM both select the 0.4 speed rate and have the same power consumption. But WATM can use a smaller number of active processors than WAS, which is helpful in saving the static power.

Figures 12 and 13 show the DVFS time overhead of WATM and WAS on the XScale for MTU = 0.1 and 0.5. The WATM + RTO scheme does not allow transition in speed after utilization = 0.4 on the XScale as this is prohibited by the RTO. The time overhead for MTU = 0.5 is smaller compared to that for MTU = 0.1 because a

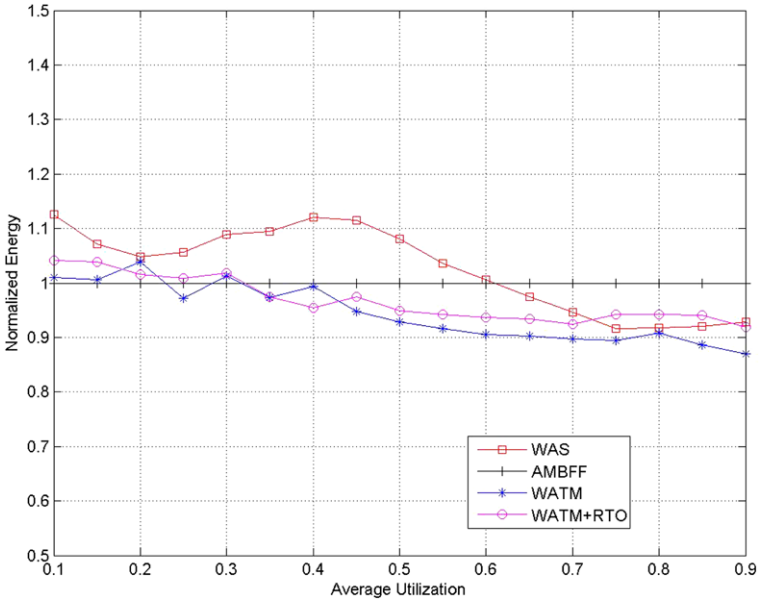

Fig. 10 Same as Fig. 8 with power normalized against AMBFF

Table 5 Number of active processors on XScale when MTU = 0.1

Algorithm	Avg. utilization		
	0.1	0.2	0.3
WATM	3	5	7
AMBFF	3	5	7
WAS	8	8	8

larger granularity of individual tasks will cut down the number of transitions in the operation speed.

The DVFS time overhead of WATM and WAS on PowerPC 405LP for MTU = 0.1 and 0.5 is depicted in Figs. 14 and 15. The plateau in the curve for WAS in Fig. 14 stems from the fact that PowerPC 405LP has fewer discrete speeds than XScale. Also, the time overhead of WATM + RTO increases rapidly after utilization = 0.8 on PowerPC 405LP because 0.8 is one of the discrete speed rates for the processor. Again, the time overhead for MTU = 0.5 is smaller than that for MTU = 0.1. In Figs. 12–15 it can be observed that WATM + RTO outperforms WAS in terms of time overhead.

5 Conclusion

In this paper, we introduced a workload-aware task migration scheduling algorithm (WATM) for real-time multiprocessor systems, in which task management (for task

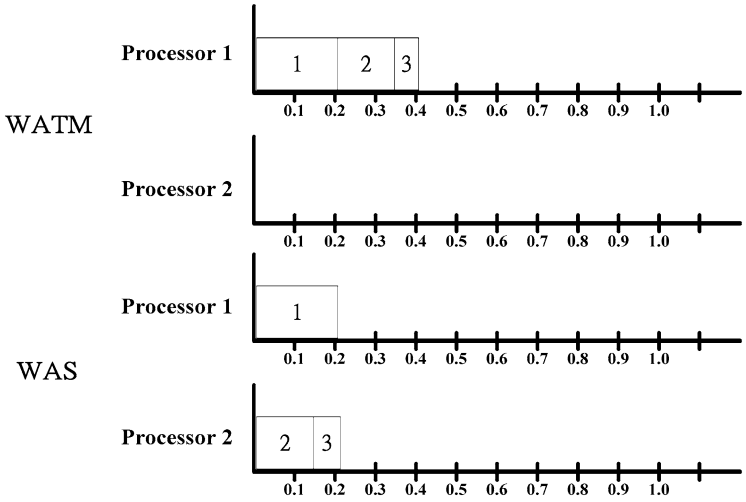

Fig. 11 Number of active processors for three tasks

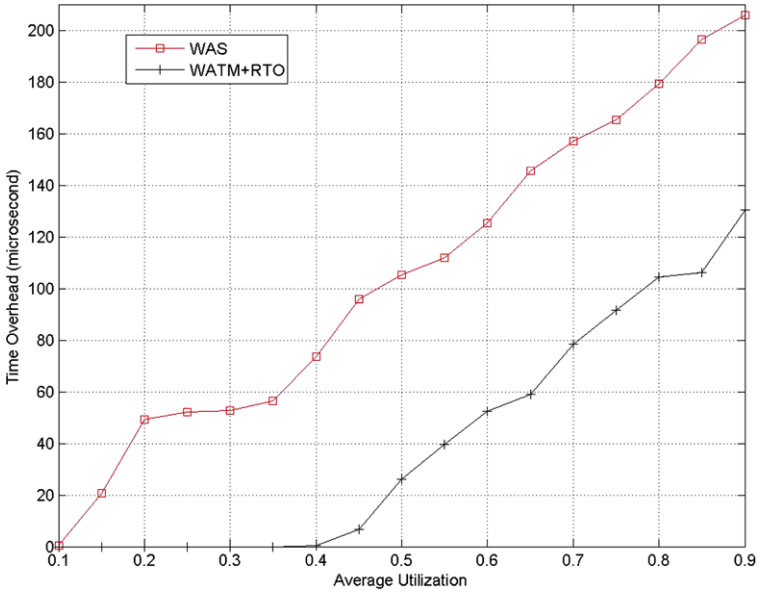

Fig. 12 Time overhead on XScale with MTU = 0.1

migration) and real-time scheduler (for real-time condition) cooperate with each other. Simulation results based on practical processor power models demonstrated the following.

(1) Our WATM algorithm can achieve a better energy saving compared to the WAS, AMBFF algorithms, chosen as benchmarks;

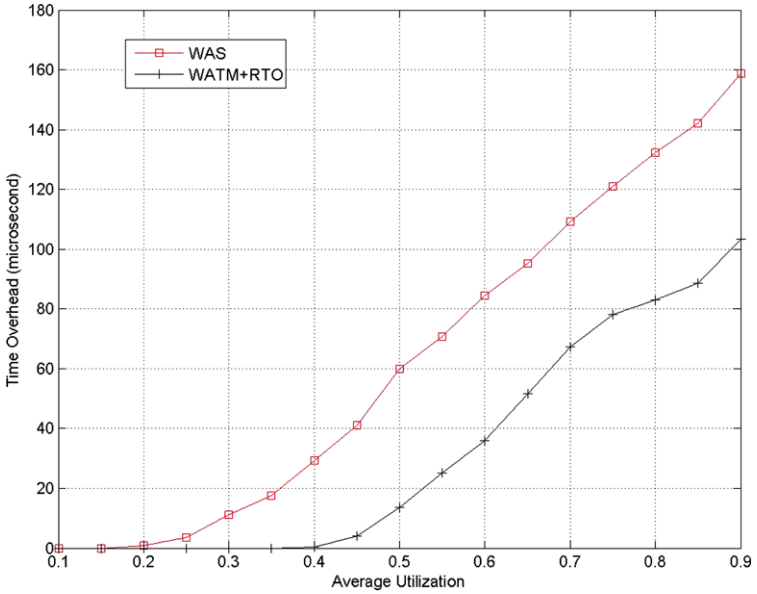

Fig. 13 Time overhead on XScale with MTU = 0.5

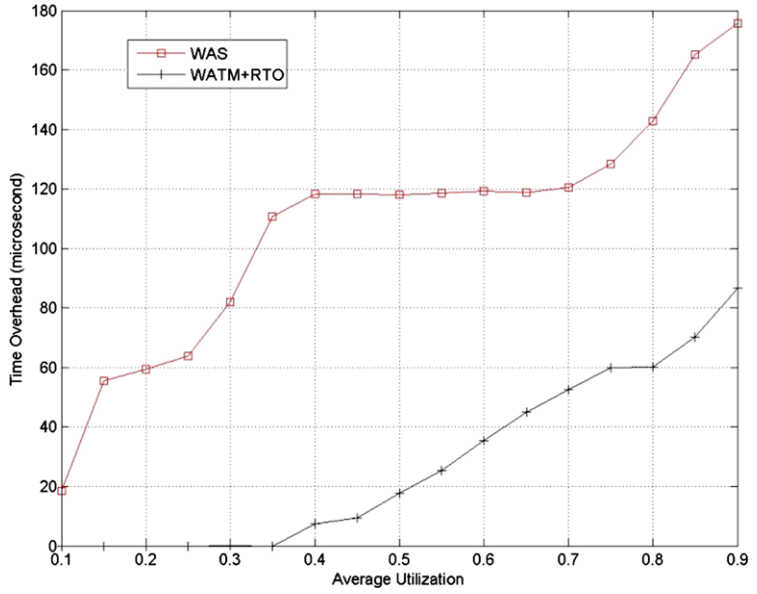

Fig. 14 Time overhead on PowerPC 405LP with MTU = 0.1

(2) Our modified WATM algorithm (so-called WATM + RTO) can achieve an average of 17.4 and 16.1 % in terms of energy reduction compared to the AMBFF scheme for MTU = 0.1 in the case of PowerPC and XScale, respectively, and 4

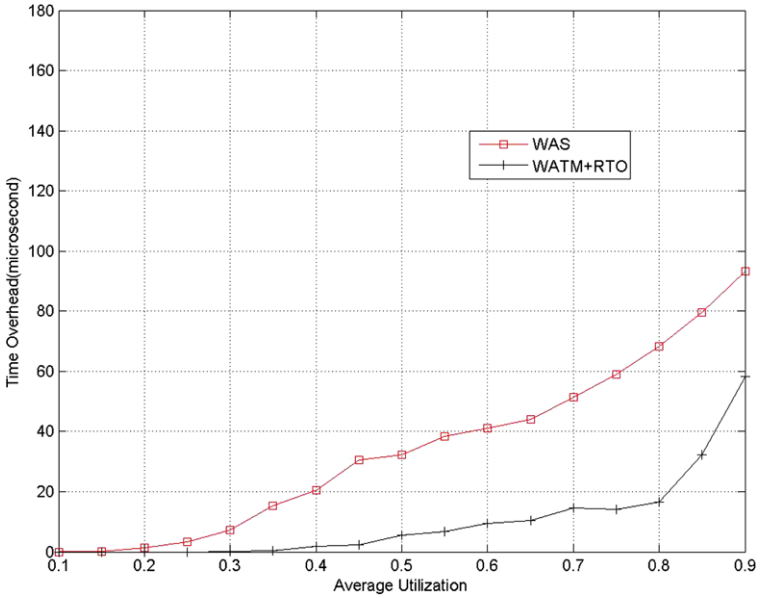

Fig. 15 Time overhead on PowerPC 405LP with MTU = 0.5

and 7 % reduction for MTU = 0.5. It is also shown that this performance can be improved with higher utilization and the number of discrete speeds available;

(3) We have analyzed the overhead associated with the voltage transition and we have provided the conditions for when such transition should be performed. Indeed, for MTU = 0.1, WATM + RTO was shown to achieve an average of 79.9 and 74.9 % DVS time overhead reduction compared to the WAS for PowerPC and XScale, respectively.

In the future, we intend to explore the task migration overhead model in other types of multiprocessor real-time systems, and investigate how this can affect the deadline and power consumption. We also plan to combine the WATM algorithm with other scheduling policies such as rate monotonic scheduling (RMS) in order to increase its performance, for instance by implementing it into the operating system kernel of a real multiprocessor embedded system.

References

1. Kim H, Hong H, Kim HS, Ahn JH, Kang S (2008) Total energy minimization of real-time tasks in an on-chip multiprocessor using dynamic voltage scaling efficiency metric. IEEE Trans Comput-Aided Des Integr Circuits Syst 27(11):2088–2092
2. Dorsey J, Searles S, Ciraula M, Johnson S, Bujanos N, Wu D, Braganza M, Meyers S, Fang E, Kumar R (2007) An integrated quad-core opteron processor. In: Proc of intl solid state circuits conference, Sept 2007, pp 102–103
3. Benini L, Hodgson R, Siegel P (1998) System-level power estimation and optimization. In: Proc of IEEE intl symposium on low power electronics and design, Aug 1998, pp 173–178

4. Cottet F, Delacroix J, Kaiser C, Mammeri Z (2002) Scheduling in real-time systems. Wiley, New York
5. Mudge T (2001) Power: A fist class architectural design constraint. IEEE Comput 34(4):52–58
6. Tiwari V, Malik S, Wolfe A (1994) Compilation techniques for low energy: an overview. In: Proc IEEE symposium on low power electronics, digest of technical papers, Oct 1994, pp 38–39
7. Valluri M, John LK (2001) Is compiling for performance = compiling for power? In: Lee G, Yew PC (eds) Interaction between compilers and computer architectures. Kluwer Academic, Norwell, pp 101–105. Chap 6
8. Chakrapani LN, Korkmaz P, Mooney VJ III, Wong WF (2001) The emerging power crisis in embedded processors: what can a poor compiler do? In: Proc of intl conference on compilers, architecture, and synthesis for embedded systems. ACM, New York, pp 176–180
9. Zhang Y, Qian L, Lu Q, Qian P, Zhao L (2009) A dynamic frequency scaling solution to DPM in embedded Linux systems. In: Proc of IEEE intl conference on information reuse and integration, Aug 2009, pp 256–261
10. Agarwal A, Rajput S, Pandya AS (2006) Power management system for embedded RTOS: an object-oriented approach. In: Proc. of electrical and computer engineering, CCECE '06. Canadian conference, May 2006, pp 2305–2309
11. Jiang X, Polastre J, Culler DE (2005) Perpetual environmentally powered sensor networks. In: Proc of intl symposium on information processing in sensor networks, June 2005, pp 463–468
12. Roundy S, Steingart D, Frechette L, Wright P, Rabaey J (2004) Power sources for wireless sensor networks. In: Proc of European workshop on wireless sensor networks, Jan 2004, pp 1–18
13. Raghunathan V, Kansal A, Hsu J, Friedman J, Srivastava M (2005) Design considerations for solar energy harvesting wireless embedded systems. In: Proc of intl symposium on information processing in sensor networks, June 2005, pp 457–462
14. Burd TD, Brodersen RW (2000) Design issues for dynamic voltage scaling. In: Proc of low power electronics and design, pp 9–14
15. Almida GM, Varyani S, Busseuil R, Sassaelli G, Benoit P, Torres L (2010) Evaluation the impact of task migration in multiprocessor systems-on-chip. In: Proc of the 23rd symposium on integrated circuits and system design, pp 73–78
16. Cormen TH, Leiserson CE, Rivest RL, Stein C (2009) Introduction to algorithms, 3rd edn. MIT Press, Cambridge, Chap 34
17. Qu G (2007) Power management of multicore multiple voltage embedded systems by task scheduling. In: Proc of intl conference on parallel processing workshops
18. Liu CL, Layland JW (1973) Scheduling algorithms for multiprogramming in a hard real-time environment. J ACM 20(1):46–61
19. Yao F, Demers A, Shenker S (1995) A scheduling model for reduced CPU energy. In: Proc of IEEE annual symposium on foundations of computer science, Milwaukee, USA, Oct 23–25, 1995, pp 374–382
20. Anderson JH, Baruah SK (2004) Energy-efficient synthesis of periodic task systems upon identical multiprocessor platforms. In: Proc of IEEE intl conference on distributed computing systems, pp 428–435
21. Hung CM, Chen JJ, Kuo TW (2006) Energy-efficient real-time task scheduling for a DVS system with a non-DVS processing element. In: Proc of IEEE RTSS, pp 303–312
22. Aydin H, Devadas V, Zhu D (2006) System-level energy management for periodic real-time tasks. In: Proc of IEEE real-time systems symposium, pp 313–322
23. Pillai P, Shin KG (2001) Real-time dynamic voltage scaling for low-power embedded operation systems. In: Proc of the 18th ACM symposium on operating systems principles, Oct 2001
24. Lin MH, Wang K (2008) Energy efficient workload-aware DVS scheduling for multi-core real-time embedded systems. Master thesis, Institute of Network Engineering, National Chiao Tung University, Taiwan, ROC
25. Zeng G, Yokoyama T, Tomiyama H, Takada H (2009) Practical energy-aware scheduling for real-time multiprocessor systems. In: Proc of the 15th IEEE intl conference on embedded and real-time computing system and applications, pp 383–392
26. Intel XScale Microarchitecture: Benchmarks (2005) http://web.archive.org/web/20050326232506/developer.intel.com/design/intelxscale/benchmarks.htm (Last accessed Nov 7, 2011)
27. Rusu C, Xu R, Melhem R, Mosse D (2004) Energy-efficient policies for request-driven soft real-time systems. In: Proc of EuroMicro conference on real-time systems, July 2004, pp 175–183

28. Xu R, Xi C, Melhem R, Mosse D (2004) Practical PACE for embedded systems. In: Proc of intl conference on embedded software (EMSOFT), pp 54–63
29. Lehoczky J, Thuel S (1994) Algorithms for scheduling hard aperiodic tasks in fixed-priority systems using slack stealing. In: Proc of IEEE real-time systems symposium
30. Acquaviva A, Alimonda A, Carta S, Pittau M (2008) Assessing task migration impact on embedded soft real-time streaming multimedia applications. EURASIP J Embed Syst. doi:10.1155/2008/518904