Adaptive Task Allocation for Multiprocessor SoCs in Real-Time Energy Harvesting Systems

Tongquan Wei
CS Department
East China Normal University
Shanghai, China 200241
tqwei@cs.ecnu.edu.cn

Yonghe Guo
ECE Department
Michigan Tech University
Houghton, MI 49931
yongheg@mtu.edu

Xiaodao Chen
ECE Department
Michigan Tech University
Houghton, MI 49931
cxiaodao@mtu.edu

Shiyan Hu
ECE Department
Michigan Tech University
Houghton, MI 49931
shiyan@mtu.edu

***Abstract*—This paper proposes an adaptive energy efficient task allocation scheme for a multiprocessor system-on-a-chip (SoC) in real-time energy harvesting systems. The proposed scheme generates an energy efficient offline task schedule for a multiprocessor SoC energy harvesting system by balancing application workload among multiple processing elements and pushing real-time application towards their deadlines. The offline task schedule is dynamically extended to adapt to the energy availability in the runtime to improve the probability of a task to be feasibly scheduled. Simulation experiments show that the proposed scheme achieves energy savings of up to 24%, and reduces task deadline miss ratio of up to 10%.**

***Index Terms*—Allocation and scheduling, energy harvesting, multiprocessor SoC, processing element (PE), real-time embedded systems, task movement vector (TMV).**

I. Introduction

Increasing number of embedded systems are being deployed in various applications with real-time constraints. Many of these applications, such as satellite-based parallel signal processing, parallel computing in sensor networks and automated target recognition (ATR), are rich in thread-level parallelism (TLP). Advances in semiconductor technology have led to the proliferation of an embedded core based system-on-a-chip (SoC), which typically have more than one processing elements. TLP-rich applications are well suited to high-performance SoCs where multiple threads can be executed simultaneously on different processing elements to provide high computing throughput.

Due to the fast-evolving application of real-time systems in battery-powered portable devices, energy has emerged as an important design constraint. Recent research on battery-powered embedded system design has focused on either reducing system energy consumption using dynamic power management technique or increasing system energy storage by harvesting energy from ambient environment.

Dynamic power management has been an active area of research for decades and several techniques, such as Dynamic Voltage Scaling (DVS) [1]–[3], have been proposed to reduce processor energy consumption [4]–[6]. For system with multiple processing elements, such as multiprocessor SoCs, the system energy consumption depends significantly on the task-to-processor allocation strategies [7]. Recently, energy harvesting technologies have been actively explored [8]–[10]. This is due to the fact that many real-time applications are being deployed in extreme environmental conditions where replacing batteries of devices in the field is not practical. Energy harvesting is a process of deriving energy from external sources, such as ambient vibration, heat, or light, and is particularly significant for autonomous low-power embedded systems.

Several research work has been conducted in the area of low-energy design for real-time energy harvesting systems. In [11], an offline task scheduling algorithm was proposed for a frame-based real-time system with minimum and maximum energy constraints. Tasks are assumed to be independent and task preemption overhead is neglected. Rusu et al. [12] described an energy-aware scheduling scheme for multi-version tasks in rechargeable real-time systems. Tasks with worst case execution times are statically scheduled under the timing and energy constraints. In the runtime, extra energy due to variation in task execution time is redistributed among remaining tasks. A lazy scheduling algorithm was proposed in [13] that executes real-time tasks as late as possible at full processor speed. The algorithm was further improved in [14], [15] to utilize task slack time for energy savings. All the above work focuses on task scheduling on a single processor in a real-time energy harvesting system.

Energy efficient task allocation and scheduling for multiprocessor systems has been extensively explored. Gururaj et al. [16] described a mathematical programming formulation-based technique for scheduling tasks in DVS-capable multiprocessor systems. The proposed task scheduling scheme takes into account precedence constraints and variation in task execution times to produce an energy efficient task schedule in polynomial time. An energy-aware scheduling scheme for real-time multiprocessor systems with uncertain task execution time was proposed in [17]. The authors considered the probabilistic distribution in task execution time and balanced the application workload among processors using the worst-fit decreasing bin-packing heuristic for energy savings. Watanabe et al. [18] presented a pipelined task scheduling method for minimizing the energy consumption of Globally Asynchronous Locally Synchronous (GALS) multiprocessor SoCs under latency and throughput constraints. In [19], an energy-aware scheduling scheme for periodic real-time tasks in the DVS-capable multiprocessor systems was proposed by considering practical constraints such as discrete speed, idle power, and application-specific power characteristics. However, all the above work focuses on reduction of energy consumption of battery-powered devices assuming a constant energy budget.

In this paper, an energy efficient task-to-processor assignment scheme for multiprocessor SoC energy harvesting systems is proposed. The proposed offline task allocation scheme greedily assigns real-time tasks to individual processing elements to generate an initial energy efficient task schedule. The energy consumption of tasks in the initial task schedule is further reduced by balancing application workload among processing elements and pushing tasks towards their deadlines while still preserving task timing constraint and task precedence constraint. Finally, the offline task schedule is adapted to the energy availability in the runtime, and task slack time due to variations in task execution times is utilized for energy saving and energy harvesting. To the best of the authors' knowledge, this is the first work on energy-efficient task allocation for multiprocessor SoC energy harvesting systems.

The rest of the paper is organized as follows. Section II introduces the system architecture, application model, and assumptions. Section III describes the proposed adaptive task allocation scheme for multiprocessor SoCs in energy harvesting systems. Section IV presents the experimental results, and Section V concludes the paper.

II. System Architecture and Applications

Focus of the study is a real-time embedded energy harvesting system comprising three major modules: energy source module, energy storage module, and the multiprocessor SoC module. The energy source module harvests energy from one or more ambient energy sources and converts energy into electrical energy. Energy storage module stores the energy generated by the energy source module and powers the multiprocessor SoC. The multiprocessor SoC draws energy from energy storage when it executes real-time tasks and stops functioning if the energy storage is empty.

A. Energy Source and Storage

Let $P_h(t)$ denote the power generated by the energy source module at time instant t, and $E_h(t_1, t_2)$ denote the harvested energy during time interval $[t_1, t_2]$, the harvested energy fed into energy storage module can be calculated as

$$E_h(t_1, t_2) = \int_{t_1}^{t_2} P_h(t)\ dt.$$

It is assumed that the power output $P_h(t)$ of the energy source module is a function of time t. The power output can be predicted by tracing the energy source profile [20]. It is also assumed that the harvested energy is fed into energy storage without wastage and the energy demand of the multiprocessor SoC only comes from the energy storage. Let E_C denote the energy storage capacity, $E_c(t)$ denote the stored energy at time instant t, and $E_d(t_1, t_2)$ denote the processor energy dissipation, then

$$0 \leq E_c(t) \leq E_C \quad \forall t$$

and

$$E_d(t_1, t_2) \leq E_c(t_1, t_2) + E_h(t_1, t_2) \quad \forall t$$

hold. The later inequality indicates that a processing element has to stop the execution of a task when the available energy is not enough to finish the task execution. The stored energy at a time instant is the current available energy minus the processor energy dissipation, that is,

$$E_c(t_2) \leq E_c(t_1) + E_h(t_1, t_2) - E_d(t_1, t_2) \quad \forall t_1 < t_2.$$

B. Multiprocessor SoC and Real-Time Tasks

The real-time embedded energy harvesting system considered in this study contains a multiprocessor system-on-a-chip (SoC). It is assumed that processing elements of the SoC are tightly coupled such that the communication cost is small enough to be incorporated in task execution time. It is also assumed that the multiprocessor SoC has N processing elements and each processing element supports a fixed operating frequency. The operating frequency of processing element n is denoted by f_n, where $1 \leq n \leq N$. The processing element of smaller index is assumed to have lower operating frequency, that is, $f_{min} = f_1 < f_2 < \cdots < f_N = f_{max}$ holds.

Consider a real-time task set Γ consisting of M periodic tasks with precedence constraints: $\{\Gamma | \tau_1, \tau_2, \cdots, \tau_M\}$. A partial-order relation $<$, called a precedence relation, is utilized to specify the precedence constraints among tasks. The notation $\tau_i < \tau_j < \tau_k$ indicates that task τ_i is an immediate predecessor of task τ_j, which is in turn an immediate predecessor of task τ_k. In other words, task τ_j is ready for execution when task τ_i is completed, and task τ_k is ready for execution when both task τ_i and task τ_j are completed.

For the sake of easy presentation, it is assumed that the precedence graph of each task is a chain. However, the proposed scheme can deal with tasks whose precedence graphs are tress and forests. It is assumed that the task of smaller index is the predecessor of the task of larger index, that is, $\tau_i < \tau_j$ holds for $i < j$.

The timing characteristics of the task τ_m for $1 \leq m \leq M$ are defined as a tuple $\tau_m = \{T_m, D_m, wcet_m\}$, where T_m is the period, D_m is the deadline, and $wcet_m$ is the worst case execution time in cycles. It is assumed that the period of a task equals its deadline and all tasks in a task set share a common deadline, that is, $T_m = D_m = T = D$. Each task is assumed to be ready at the very beginning and can be assigned to one and only one processing element. Task τ_m for $1 \leq m \leq M$ is also characterized by the start time st_m and finish time ft_m.

III. Energy Efficient Task Allocation Algorithm

This section describes the proposed adaptive task-to-processing element assignment scheme for multiprocessor SoCs in real-time embedded energy harvesting systems. The proposed task allocation scheme statically assigns tasks in a given task set to processing elements to minimize energy consumption of the tasks, and adapts the static task schedule to the runtime behavior of tasks and energy availability. This algorithm aims at statically minimizing energy consumption of tasks in a given task set and then dynamically improving the percentage of feasible tasks by utilizing accumulated slack time in the runtime. Section III-A describes the offline task allocation scheme and section III-B presents the runtime adaptation of the offline task schedule.

A. Generate an Energy Efficient Offline Task Allocation

Tasks in a given task set are first assigned to individual processing elements in a way that timing constraints and precedence constraints of all tasks are satisfied, and task energy consumption is greedily minimized. The produced task schedule is called an initial schedule. The energy consumption of tasks in the initial schedule is further reduced by moving tasks to specific processing elements without compromising the schedule feasibility and violating task precedence constraint. The output of the algorithm at this step is an energy efficient offline task schedule.

Generate an initial task schedule: It has been shown that, for a given task, the processing element operating at the single frequency f_{low} consumes less energy than the processing element operating at the single frequency f_{high} if the timing constraint of the task is satisfied, where $f_{low} < f_{high}$ [3]. Therefore, the initial task schedule is generated by assigning tasks in the given task set to the processing element with lower operating frequency such that the energy consumption of the output task schedule is greedily minimized.

Since precedence graph of a task is assumed to be a chain, the precedence constraint of the task can be satisfied by executing the task after its earliest start time and before its latest finish time. The earliest start time of a task is defined to be its earliest start time on the processing element with the lowest operating frequency, while the latest finish time of a task is defined to be its latest finish time on the processing element with the highest operating frequency. For task τ_m, its earliest start time is

$$est_m = \frac{wcet_{pm}}{f_{min}}, \tag{1}$$

and its latest finish time is

$$lft_m = D - \frac{wcet_{sm}}{f_{max}}, \tag{2}$$

where $wcet_{pm}$ denotes the sum of the worst case execution time of all predecessors of task τ_m, and $wcet_{sm}$ denotes the sum of the worst case execution time of all successors of task τ_m. f_{min}/f_{max} is the speed of the precessing element with lowest/highest operating frequency in the multiprocessor SoC system. Therefore, the precedence constraint of task τ_m is met if

$$est_m \le st_m, \quad ft_m \le lft_m \quad 1 \le m \le M \tag{3}$$

holds. For a task without a predecessor, its earliest start time is 0, while for a task without a successor, its latest finish time is its deadline D. Similarly, for an independent task, its earliest start time is 0 and its latest finish time is its deadline D.

Whether a task can be feasibly scheduled on a processing element of a multiprocessor SoC energy harvesting system depends significantly on the finish time of the last task on the processing element. Let $idlt_n$ denote the time instant on processing element n and call it the **idle instant**. Task τ_m can be feasibly scheduled on processing element n if

$$\max(idlt_n, est_m) + \frac{wcet_m}{f_n} \le D$$

$$1 \le n \le N, 1 \le m \le M \tag{4}$$

holds.

```
Require: Task set $\Gamma$
1.  for $\tau_m, 1 \le m \le M$ do
2.    derive $est_m$ and $lft_m$ using equation (1) and (2)
3.    $st_m = est_m$ and $ft_m = lft_m$
4.  end for
5.  for $n = 1$ to $N$ do
6.    $idlt_n = 0$
7.  end for
8.  $feasible = 0$
9.  for $\tau_m, 1 \le m \le M$ do
10.   for $PE_n, 1 \le n \le N$ do
11.     if inequality (5) holds then
12.       update $st_m, idlt_n$ and $ft_m$ acc. to equation (6)
          {assign task $\tau_m$ to $PE_n$}
13.       $feasible = 1$; break
14.     else
15.       $feasible = 0$
16.     end if
17.   end for
18.   if $feasible == 0$ then
19.     print "Infeasible Schedule"
20.     exit(1) {Exit when infeasible}
21.   end if
22. end for
```

Fig. 1: Generate an initial task schedule.

Combining inequality (3) and inequality (4) gives the below condition for task τ_m to be feasibly scheduled on processing element n without compromising the precedence constraint of the task.

$$\max(idlt_n, est_m) + \frac{wcet_m}{f_n} \le lft_m \le D$$

$$1 \le n \le N, 1 \le m \le M \tag{5}$$

Fig. 1 shows the algorithm to generate an initial task schedule. Lines 1 to 4 derive the earliest start time and latest finish time of tasks in the given task set using equation (1) and (2), and initialize the start time and finish time of each task to its earliest start time and latest finish time, respectively. Lines 5 to 7 reset the idle instant $idlt_n$ ($1 \le n \le N$) of all processing elements to 0. A flag $feasible$ is introduced to indicate the feasibility of a task allocation. The $feasible$ flag is initialized to 0 in line 8. For task τ_m and processing element n, if the inequality (5) holds, then the task can be feasibly allocated to processing element n and its timing constraint is satisfied. The $st_m, idlt_n$, and ft_m are updated in turn, that is,

$$\begin{aligned} st_m &= \max(idlt_n, est_m) \\ idlt_n &= st_m + \frac{wcet_m}{f_n} \\ ft_m &= idlt_n. \end{aligned} \tag{6}$$

The flag $feasible$ is set to 1 and the algorithm proceeds to allocate the next task. Otherwise, the task can not be feasibly allocated to the processing element n. The $feasible$ is reset to 0, and the algorithm examines the next processing element for feasibly allocating the task. If the task can not be feasibly allocated to any of the N processing elements, the tasks in the given task set can not be feasibly scheduled, and the algorithm exits. This process continues until all tasks are feasibly assigned to processing elements, or a task can not be feasibly scheduled on any processing element, whichever comes first. This process is demonstrated from lines 9 to 22 in Fig. 1. The output of the algorithm is an initial task schedule of tasks in a given task set. The energy consumption of the initial task schedule is greedily minimized.

Migrate tasks to further reduce energy consumption: It has been shown that balancing system workload among multiple processing elements minimizes system energy consumption [7], and executing a task as late as possible increases the chance of the task to meet its deadline in energy harvesting systems [13]. The proposed task movement algorithm (TMA) further minimizes energy consumption of the initial task schedule and improves the percentage of schedulable tasks in the runtime when available energy $E_c(t)$ is small by balancing workload among processing elements and pushing tasks towards their deadlines.

For the sake of easy presentation, the concept of energy profit is introduced. The energy profit of a task movement is defined to be the energy consumption of the task on the processing element of the initial task schedule minus the energy consumption of the task on the processing element to be moved to. As a result, for a system with N processing elements, there will be N energy profits for a single task. To achieve an energy efficient task schedule, tasks in the initial task schedule are to be migrated among processing elements such that the energy profit of the resultant task schedule is maximized.

For the investigated system with M real-time tasks and N processing elements, a single task has N possible destination processing elements including the current processing element where the task is allocated, and there are $M \times N$ task movement options in total. A task movement vector (TMV) is then introduced to describe a task movement option. The i^{th} ($1 \leq i \leq M \times N$) task movement option is denoted by TMV_i and is defined as $TMV_i = \{tid, st_{tid}, ft_{tid}, PE_{n1}, PE_{n2}, ep, flg\}$, where tid is the index of the involved task, τ_{tid}. The start time and finish time of the task are st_{tid} and ft_{tid}, respectively, and the source processing element and the destination processing element of the movement is denoted by PE_{n1} and PE_{n2}, respectively. The ep is the energy profit of the task movement, and flg is the flag to indicate if the movement has been taken. Let $TMV_i{\cdot}tid$ indicate the member of the index tid. Other members of TMV_i can be represented in the same way.

Assume task τ_{tid} is to be moved from processing element n_1 to n_2. If the inequality

$$\max(idlt_{n2}, est_{tid}) + \frac{wcet_{tid}}{f_{n2}} \leq lft_{tid} \leq D$$
$$1 \leq n \leq N, 1 \leq tid \leq M \qquad (7)$$

```
**Require:** initial task schedule
1.  for $\tau_m, 1 \leq m \leq M$ do
2.    for $PE_n, 1 \leq n \leq N$ do
3.      generate & initialize task movement vectors (TMVs)
4.    end for
5.  end for
6.  sort $M \times N$ TMVs in the order of energy profit from high
    to low
7.  for $i = 1$ to $M \times N$ do
8.    $feasible = 1$
9.    if $TMV_i{\cdot}flg_i$ == 1{task $TMV_i{\cdot}tid$ has been moved}
      then
10.     $feasible = 0$
11.   end if
12.   if inequality (7) holds then
13.     $feasible = 1$ {task $TMV_i{\cdot}tid$ can be moved}
14.   else
15.     $feasible = 0$ {task $TMV_i{\cdot}tid$ can not be moved}
16.   end if
17.   if $feasible == 1$ then
18.     move task $TMV_i{\cdot}tid$ from $TMV_i{\cdot}n_1$to $TMV_i{\cdot}n_2$
19.     update $st_{tid}, idlt_{n2}$, and $ft_{tid}$ using equation (8)
20.     update $idlt_{n1}$ using equation (9)
21.     $TMV_i{\cdot}flg_i$ = 1
22.   end if
23. end for
```

Fig. 2: Task movement algorithm (TMA).

holds, then task τ_{tid} can be feasibly moved to processing element n_2. The $st_{tid}, idlt_{n2}$, ft_{tid} are updated to

$$st_{tid} = \max(idlt_{n2}, est_{tid})$$
$$idlt_{n2} = st_{tid} + \frac{wcet_{tid}}{f_{n2}}$$
$$ft_{tid} = idlt_{n2}, \qquad (8)$$

and the idle instant of processing element n_1 is updated to

$$idlt_{n1} = idlt_{n1} - \frac{wcet_{tid}}{f_{n1}}. \qquad (9)$$

As is shown in Fig. 2, the task movement algorithm (TMA) takes as input the initial task schedule. Similar to the algorithm in Fig.1, a flag $feasible$ is introduced to indicate the feasibility of a task schedule. Lines 1 to 5 generate $M \times N$ task movement vectors. For TMV_i, the members of tid, st_{tid}, and ft_{tid} are initialized by the initial task schedule, the energy profit ep is calculated, and the flg is reset to 0, indicating that all tasks have not been moved. Line 6 of the TMA algorithm sorts all task movement vectors in the order of energy profit from high to low. This arrangement of task movement vectors ensures that energy consumption of the resultant task schedule is less than that of the initial task schedule.

In line 8, the $feasible$ flag is set to 1, indicating that the current task schedule is feasible. Each task can only be moved at most once, as is shown from lines 9 to 11. If the current task $TMV_i{\cdot}tid$ satisfies the inequality 7, then it can be moved and $feasible$ flag is set to 1 (lines 12-13). Otherwise, the task can not be moved and the $feasible$ flag is set to 0 (lines 14-15). For the current task that can be moved, line 18 moves the task from PE_{n1} to PE_{n2}, line 19 updates the $st_{tid}, idlt_{n2}$, and ft_{tid} according to equation (8), line 20 update $idlt_{n1}$ using equation (9), and line 21 set the flag $TMV_i{\cdot}flg_i$ to 1, indicating that the task has been moved. The process shown from lines 7 to 23 repeats until all task movement vectors have been examined, or the energy profit of a task movement vector is less than or equal to 0, whichever comes first. The output of the algorithm is an energy efficient offline task schedule.

B. Dynamic Adaptation of the Offline Task schedule to Energy Availability

The available energy of a energy harvesting system fluctuates with time and is limited by the energy storage capacity. When the offline task schedule is generated, the energy constraint is not considered. While in the runtime, if the available energy is not enough for a task to finish the execution, the processing element of the task has to stop the task execution before the task is completed. As a result, the task and its successors may miss their deadlines and violate their precedence constraints, and the offline task schedule is invalidated. Hence, the offline task schedule needs to be tailored in the runtime based on the energy information of the energy harvesting system.

The proposed scheme aims to improve the percentage of tasks that can be feasibly scheduled in an energy harvesting system. To achieve this, the offline task schedule is generated so that the total energy consumption of the offline task schedule is minimized, which in fact increases the chance of a task to finish execution when available energy varies. In addition, when the offline task schedule is generated, tasks in the given task set are pushed towards their deadlines, which avoids spending scarce energy on tasks too early when the available energy $E_c(t)$ is small.

In the runtime, few tasks executes up to their worst case execution time. This feature of task execution time is utilized to maintain the feasibility of runtime task schedule in two aspects. On one hand, the variation in task execution time reduces the energy consumption of a task, and hence conserve the scarce resource of energy. On the other hand, energy from ambient environment is harvested during the slack due to the variation in task execution time, which further improve the availability of energy.

Assume task τ_{m-1} and task τ_m are assigned on processing element n, and the execution of task τ_{m-1} is finished at $ft_{m-1} - slk_{m-1}$, where slk_{m-1} is the accumulated slack time of preceding tasks due to variation in execution times. Task τ_m starts execution at st_m if enough energy is available to finish the execution, that is, the inequality

$$E_c(ft_{m-1} - slk_{m-1}) + E_h(ft_{m-1} - slk_{m-1}, ft_m) \geq E_d(st_m, ft_m) \quad (10)$$

holds. In inequality (10), $E_c(ft_{m-1} - slk_{m-1})$ is the available energy at time instant $ft_{m-1} - slk_{m-1}$, $E_h(ft_{m-1} - slk_{m-1}, ft_m)$ is the energy harvested during the period from the actual finish time of task τ_{m-1} to the nominal finish time of task τ_m, and $E_d(st_m, ft_m)$ is the energy demand of the task. Task τ_m is discarded if inequality (10) does not hold. Hence, energy is conserved for the execution of remaining tasks. Note that a task is not rescheduled to execute early even if its processing element is idle before its start time. This policy enables the algorithm avoid spending energy on the task too early when system available energy storage is small.

IV. Experimental Results

Extensive simulation experiments were performed to validate the proposed task allocation scheme for energy efficiency and feasibility performance. The proposed task allocation algorithm was implemented in C, and tested on an Asus machine with Core Duo processor of 1.66 GHz and DDR memory of 2.5 GB. Since, to the best of the authors' knowledge, this is the first work on energy efficient task allocation for multiprocessor SoC energy harvesting system, the proposed scheme was validated by comparing the initial task schedule and the optimized task schedule in energy consumption and task deadline miss ratio when available energy is small. Alg:initial schedule

Simulations were carried out over 1000 task sets of varying sizes to account for stochastic anomalies. The number of tasks in a task set ranges from 10 to 60 and the tasks were generated by assuming a common deadline. Task execution times in cycles were generated such that all tasks in a task set can be feasibly scheduled at a certain processor speed. Precedence constraint is applied to randomly selected tasks in a task set such that the precedence of a single task is a chain.

Solar energy was selected as the energy source in the simulation experiments. The power $P_h(t)$ of the solar source [21] is given by

$$P_h(t) = |F \cdot N(t) \cos(\frac{t}{70\pi}) \cos(\frac{t}{120\pi})|,$$

where F is a constant scaling unit, and $N(t)$ is a random variable that is normally distributed with mean 0 and variance 1.

Table I compares the initial task schedule with the task schedule that is optimized using the proposed task movement algorithm (TMA). In the table, the E_1 denote the average energy consumption of the optimized task schedule, E_2 denote the average energy consumption of the initial task schedule, and CPU denotes the average runtime of the TMA algorithm. The common deadline of tasks is denoted by $Deadline$, and the average schedule length is denoted by $SchdLength$. $(E_2 - E_1)/E_2$ is utilized to denote the energy savings of the optimized task schedule when compared to the initial task schedule.

It can be derived from Table I that the task schedule optimized using the proposed task movement algorithm achieves energy savings of up to 24% when compared to the energy consumption of the initial task schedule. This is because the task movement algorithm greedily balances application workload among processing elements, which effectively reduces system energy consumption [7]. Due to the load balancing property of the TMA algorithm, the schedule length of the

TABLE I: Compare initial task schedule and optimized task schedule. CPU refers to the algorithm runtime in seconds

# of tasks	Optimized task schedule				Initial task schedule				
in a task set	E_1	CPU(Sec)	$SchdLength$	$Deadline$	E_2	CPU(Sec)	$SchdLength$	$Deadline$	$(E_2 - E_1)/E_2$
10-19	138.86	0.96	26.14	56.00	183.41	0.98	55.04	56.00	24.29%
20-29	211.37	1.03	36.24	88.00	279.45	0.98	87.11	88.00	24.35%
30-39	350.77	1.01	60.26	134.00	434.94	1.04	133.91	134.00	19.25%
40-49	416.68	0.94	65.25	170.00	532.16	0.97	169.87	170.00	21.70%
50-59	510.61	0.98	56.16	203.00	632.33	1.00	202.28	203.00	19.35%

optimized task schedule is shorter as compared to that of the initial task schedule. For example, for task sets that have 10-19 tasks, the schedule length of the optimized task schedule is 26.14 while the schedule length of the initial task schedule is 55.04. The runtime overhead of the task movement algorithm is around 1 Second, and could be negligible when compared to task execution times.

TABLE II: Comparison of task deadline miss ratio

# of tasks	Initial task schedule	Optimized task schedule
in a task set	miss ratio	miss ratio
10	20%	10%
20	15%	10%
30	20%	10%
40	20%	10%
50	10%	8%

Table II compares the task deadline miss ratio of the initial task schedule with that of the optimized task schedule. Experimental results for five test cases are presented. It is shown in the table that the task deadline miss ratio of the optimized task schedule is up to 10% lower when compared to that of the initial task schedule. This is because the task movement algorithm pushes tasks towards their deadlines and schedules tasks to execute as late as possible. This policy enables the optimized task schedule avoid spending scarce energy on tasks too early when system available energy is small.

V. Conclusions

In this paper, an energy efficient task allocation scheme for multiprocessor SoCs is proposed for real-time embedded energy harvesting systems. The proposed scheme first generates an energy efficient initial task schedule. The energy consumption of the task schedule is further reduced by balancing application workload among processing elements of a multiprocessor SoC and pushing tasks in a given task set towards their deadlines without compromising schedule feasibility and violating task precedence constraints. The offline task schedule is then extended in the runtime to adapt to the variation in task execution time for further energy savings. Extensive simulation experiments show that the proposed energy efficient task allocation scheme achieves energy savings of up to 24%, and reduces task deadline miss ratio of up to 10%.

References

[1] G. Quan and X. Hu, "Energy efficient fixed-priority scheduling for real-time systems on variable voltage processors," *Proceedings of the DAC*, 2001.

[2] Y. Shin, K. Choi, and T. Sakurai, "Power optimization of real time embedded systems on variable speed processors," *Proceedings of the ICCAD*, 2000.

[3] T. Ishihara and H. Yasuura, "Voltage scheduling problem for dynamically variable voltage processors," *Proceedings of the ISLPED*, 1998.

[4] A. Chandrakasan, S. Sheng, and R. Brodersen, "Low-power cmos digital design," *IEEE Journal of Solid-State Circuits*, 1992.

[5] J. Lorch and A. J. Smith, "Software strategies for portable computer energy management," *IEEE Personal Communication Magazine*, vol. 5, no. 3, pp. 60–73, 1998.

[6] L. Benini, A. Bogliolo, and G. Micheli, "A survey of design techniques for system-level dynamic power management," *IEEE Transactions on VLSI Systems*, 2000.

[7] T. Wei, P. Mishra, K. Wu, and H. Liang, "Fixed-priority allocation and scheduling for energy-efficient fault-tolerance in hard real-time multiprocessor systems," *IEEE Transactions on Parallel and Distributed Systems*, 2008.

[8] S. Roundy, D. Steingart, L. Frechette, P. Wright, and J. Rabaey, "Power sources for wireless sensor networks," *Proceedings of Wireless Sensor Networks*, pp. 1–17, 2004.

[9] V. Raghunathan, A. Kansal, J. Hsu, J. Friedman, and M. Srivastava, "Design considerations for solar energy harvesting wireless embedded systems," *International Symposium on Information Processing in Sensor Networks*, pp. 457– 462, 2005.

[10] X. Jiang, J. Polastre, and D. Culler, "Perpetual environmentally powered sensor networks," *International Symposium on Information Processing in Sensor Networks*, pp. 463– 468, 2005.

[11] A. Allavena and D. Mosse, "Scheduling of frame-based embedded systems with rechargeable batteries," *Workshop on Power Management for Real-Time and Embedded Systems*, 2001.

[12] C. Rusu, R. Melhem, and D. Mosse, "Multi-version scheduling in rechargeable energy-aware real-time systems," *Journal of Embedded Computing*, 2005.

[13] C. Moser, D. Brunelli, L. Thiele, and L. Benini, "Lazy scheduling for energy harvesting sensor nodes," *Working Conference on Distributed and Parallel Embedded Systems*, 2006.

[14] S. Liu, Q. Qiu, and Q. Wu, "Energy aware dynamic voltage and frequency selection for real-time systems with energy harvesting," *Proceedings of the DATE*, 2008.

[15] S. Liu, Q. Wu, and Q. Qiu, "An adaptive scheduling and voltage/freqency selection algorithm for real-time energy harvesting systems," *Proceedings of the DAC*, 2009.

[16] K. Gururaj and J. Cong, "Energy efficient multiprocessor task scheduling under input-dependent variation," *Proceedings of the DATE*, 2009.

[17] C. Xian, Y. Lu, and Z. Li, "Energy-aware scheduling for real-time multiprocessor systems with uncertain task execution time," *Proceedings of the DAC*, 2007.

[18] R. Watanabe, M. Kondo, M. Imai, H. Nakamura, and T. Nanya, "Task scheduling under performance constraints for reducing the energy consumption of the gals multi-processor soc," *Proceedings of the DATE*, 2007.

[19] G. Zeng, T. Yokoyama, H. Tomiyama, and H. Takada, "Practical energy-aware scheduling for real-time multiprocessor systems," *15th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, 2009.

[20] A. Kansal, J. Hsu, S. Zahedi, and M. Srivastava, "Power management in energy harvesting sensor networks," *ACM Transactions on Embedded Computing Systems* (in revision).

[21] C. Moser, D. Brunelli, L. Thiele, and L. Benini, "Real-time scheduling for energy harvesting sensor nodes," *MICS Scientific Conference and SNF panel Review*, 2006.