Value-based Task Scheduling for Nonvolatile Processor-based Embedded Devices

Wei-Ming Chen[1], Tai-Sheng Cheng[1], Pi-Cheng Hsiu[2], and Tei-Wei Kuo[1,2]
[1] Department of Computer Science and Information Engineering, National Taiwan University, Taiwan
[2] Research Center for Information Technology Innovation, Academia Sinica, Taiwan
d04922006@csie.ntu.edu.tw, b01902100@csie.ntu.edu.tw, pchsiu@citi.sinica.edu.tw, ktw@csie.ntu.edu.tw

***Abstract*—Energy harvesting wearable sensor nodes offer low maintenance and high mobility, but suffer from unstable power supplies and insufficient energy. Featuring low standby power and instant backup and restore operations, nonvolatile processors have emerged as one of the most promising technologies for the improvement of energy harvesting wearable devices. However, device quality of service (QoS) is still hindered by lack of sufficient energy. To address the problem, this paper studies how to schedule tasks for DVFS-enabled nonvolatile processor-based embedded devices to maximize QoS. First, we model the task scheduling problem as an optimization problem, with the objective of maximizing the total value (which represents QoS) under available harvested energy and time. We then prove the problem to be NP-hard. In addition, we propose a pseudopolynomial-time optimal algorithm based on dynamic programming, as well as an approximation algorithm that allows the trade-off between the running time and total value. To evaluate our algorithms, we applied the presented algorithms on an ultra-low-power platform produced by Texas Instruments, and conducted extensive simulations with different system models and task sets. The results demonstrate that the optimal sequences derived by the dynamic-programming algorithm can be executed on the platform, where the differences between simulation results and real traces are less than 1%. Also, compared with the dynamic-programming algorithm, our approximate algorithm can speed up about 10^5 time faster than the dynamic-programming algorithm, while the degraded value is less than 3%.**

I. Introduction

The development of technologies for use in wearable devices has accelerated significantly in recent years and annual sales of wearable devices is expected to grow to around six billion dollars by 2018 [1]. However, wearable devices suffer from limited battery life and inconvenience of recharging, raising interest in energy harvesting as a promising power source and means of improving usability. Various means of energy harvesting have been proposed and developed, with approaches harvesting energy from body motion, vibration, piezoelectric materials, and thermal, solar and wireless sources. However, such energy sources are inherently unstable, and devices relying on energy harvesting typically suffer from unstable and insufficient power, resulting in the frequent interruption of task execution. A nonvolatile processor which implements in place backup by integrating nonvolatile memory like ferroelectric flip-flops at register level, has been proposed to address the unstable power supply provided by energy harvesting [20, 32]. However, the current inability to harvest sufficient energy levels still restricts the QoS, and maximizing QoS under time and energy requirements has emerged as one of the key issues in the further development of wearable devices.

For systems with insufficient energy, an alternative is to allow the systems run under excessive workload. Systems prioritize critical applications while dropping non-critical applications according to predefined reward/value of each application. Hence, the Imprecise Computation and Increased Rewards with Increased Service were proposed to model problems to maximize QoS with insufficient resources [16, 18]. A heuristic algorithm, REW-Pack, was proposed to maximize the rewards of a DVFS-enabled processor under time and energy constraints [10]. For systems which provide periodic services, where the QoS is adjustable and evaluated in terms of values, an algorithm was presented to maximize overall value [7]. However, these solutions focus maximizing total value under fixed resource budget that they do not support energy harvesting devices.

Designing an efficient self-powered system to realize the potential benefits of harvesting energy requires an in-depth understanding of several factors [27, 28]. Harvesting-aware governing and scheduling algorithms have been presented to dynamically scale the frequency and voltage level of the systems under time and energy constraints [6, 25, 26]. The performance properties of energy-harvesting real-time network systems in Real-Time Calculus framework is analyzed and provide useful insight to design systems [22]. A DVFS strategy for self-powered systems has been proposed based on the model-based run-time prediction of future harvested energy [17]. To achieve full system energy autonomy, a global control algorithm which considers the impacts of solar panels and supercapacitors on harvested energy has been proposed [29]. Likewise, a long-term power manager was developed to improve energy efficiency based on the characteristics of supercapacitor charge redistribution [24]. The impact of system workload on the harvesting efficiency has also been investigated and used to improve energy collection [31]. Due to the limitation of insufficient harvested energy, the value-based task model was extended to energy harvesting devices. Algorithms which derive optimal and near optimal solutions were presented to maximize the QoS of energy harvesting systems with discrete QoS levels [8]. Some studies have also focused on maximizing the system QoS of environmentally powered devices under energy constraints [9, 15, 23]. However, these proposed algorithms focus on traditional energy harvesting devices which are still hindered by frequent power failure due to volatile memory.

Nonvolatile processors have emerged as a candidate solution to handle the unstable power output of energy harvesting systems [20, 32]. A checkpoint scheme that ensures correctness for system checkpoints has been proposed [21]. Furthermore, to realize the benefit of this technology, several nonvolatile processor configurations and various architectural level designs have been evaluated and explored in order to maximize forward progress of applications [20]. A machine learning-based scheme was presented to dynamically select nonvolatile microarchitecture to use the harvested energy for maximizing forward progress [19]. The nature of nonvolatile processors also offers a range of methods for task scheduling and system governance. Taking advantage of instant sleeps and wakeups, a framework has been proposed to reduce energy consumption by shutting down CPU cores during short-term idle phases [30]. Also, it has been shown that the power switching overhead of nonvolatile processors influences system performance, and a performance-aware scheduling technique has been proposed [13]. Targeting at reducing deadline miss rate and improving energy utilization, several scheduling algorithms for solar-powered nonvolatile sensor nodes have recently been proposed [11, 12]. However, no previous studies has focused on maximizing total value of a self-powered nonvolatile processor.

In this paper, we study value-based task scheduling for an energy harvesting nonvolatile processor, and validate the proposed scheduler on MSP430FR5969 [4] even though this may not be the final platform. The contributions of this paper are as follows. First, we model the problem of task scheduling for a self-powered nonvolatile processor. The objective of the problem is to maximize the total system value which is the sum of all completed tasks, provided that the energy consumed does not exceed the harvested energy and the execution time does not exceed the scheduling period. We then derive the fundamental result on the NP-hardness of the problem. In addition, we propose a pseudopolynomial-time optimal algorithm based on dynamic programming to resolve the problem, and analyze the properties of the proposed algorithm. Furthermore, to reduce the running time, we present an approximation algorithm based on the dynamic-programming algorithm. We also discuss the performance of the approximation algorithm, and show the trade-off between running time and total value from a mathematical perspective. For the performance evaluation, we use real system parameters of a device developed by Texas Instruments, MSP430FR5969 , measuring the real task execution patterns, and conduct extensive simulations with the different task sets. To provide useful insights, we validate the proposed algorithms on the device. The results show that the differences between simulation results and real traces are ranged from 0.1% to 0.6% under different scenarios.

The rest of this paper is organized as follows. In Section II, we present the system model and define the problem formally. In Section III, we propose an algorithm to resolve the problem and derive an approximate result based on the algorithm. In Section IV, we present the simulation results and discuss the performance of the proposed algorithms. In Section V, we conclude this paper.

II. System Model and Problem Definition

A. System Model

Integrating nonvolatile memory makes nonvolatile processors a promising solution for the development of energy harvesting wearable devices and sensor nodes [20]. We explore value-based task scheduling for a system equipped with a DVFS-enabled single-core nonvolatile processor with the following characteristics. **a)** Frequent sleeps and wakeups: In a traditional volatile system, the system state, which resides in volatile memory has to be saved before the system can enter a sleep mode (i.e., turning off SRAM). Thus, the sleep and wakeup transitions usually require considerable time and energy because the data have to be dumped to and restored form a secondary storage device (e.g, SSD or HDD). However, the non-volatile design drastically reduces the time and energy overhead [32]. **b)** Various sleep modes: Compared to traditional processors, non-volatile system enables additional sleep modes for idling the system by backing up and turning off memories at different levels. For example, several low power modes are proposed and implemented as the system state with the help of non-volatile memory [4]. **c)** Low standby power: To retain data in volatile memory, traditional devices have to maintain a steady power supply to volatile memory. In contrast, nonvolatile processors can shut off the power supply to memory after the data is backed up to non-volatile memory. Therefore, the standby power of the nonvolatile processor is nearly zero when all volatile memories are turned off [14]. Through these advantages, nonvolatile processors allow for computers to be normally-off, with hardware aggressively shut down except when required.

Power supply instability is one of the major challenges for designing an energy harvesting device because the amount of power harvested depends on many environmental factors and may vary significantly [28]. For example, when a traditional processor loses power, the system suffers from considerably increased overhead to back up data to secondary storage or the loss of the unfinished computing progress. However, nonvolatile processors can be used to preserve computing progress even if a power failure occurs [20]. In addition, to increase system reliability, a small energy storage device, such as a supercapacitor, is commonly applied to the energy harvesting system so that extra energy can be saved for future use [20, 27]. However, due to the lack of a stable power supply, the harvested energy might not be sufficient to execute all tasks. Consequently, some tasks have to be dropped in factor of more important tasks. To deal with these issues, we consider a value-based task model to maximize system QoS under the harvested energy constraint by exploiting the advantages of non-volatile processors.

B. Problem Definition

In this paper, we are interested in maximizing the total system value on the condition that the total energy consumed and time required do not exceed the available amount where the available energy is harvested over time. For example, given

Period	Initial energy	Harvested energy	Frequency levels	Sleep modes
D = 40	E_0 = 200	$h(19)$ = 100	M = 1	K = 2

	Power	Overhead
Sleep 1	$p(1)$ = 2	$o(1)$ = 1
Sleep 2	$p(2)$ = 1	$o(2)$ = 2

	Value	Required time	Required energy	Ready time
Task 1	v_1 = 5	$t(1,1)$ = 11	$e(1,1)$ = 120	$r(1)$ = 0
Task 2	v_2 = 10	$t(2,1)$ = 23	$e(2,1)$ = 90	$r(2)$ = 10
Task 3	v_3 = 8	$t(3,1)$ = 21	$e(3,1)$ = 130	$v(3)$ = 18

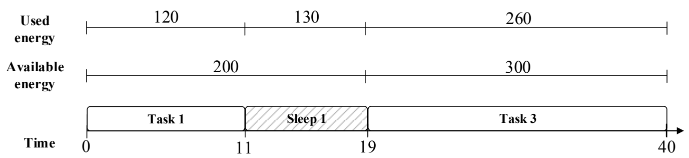


Fig. 1: An Example

an input instance, the objective is to derive an optimal sequence such as the one shown in Fig. 1. The system and task model under our considerations can be formulated as follows.

We assume a frame-based task model. In the system, there are N non-preemptive tasks, and each task i is ready at $r(i)$. The tasks set is denoted by $T = \{t_1, t_2, ..., t_N\}$. All task deadlines are equal to the scheduling period, which is denoted by D. Tasks can be dropped or scheduled in our model. The tasks are executed by a nonvolatile processor which can adjust its operating frequency and voltage dynamically. The processor is equipped with M frequency levels, $F = \{f_1, f_2, ..., f_M\}$, and every task can run at an available speed f_k, where $1 \leq k \leq M$. In addition, the worst case execution time and energy consumption for executing task i at frequency k are known, and respectively denoted by $t(i,k)$ and $e(i,k)$. Note that we ignore the switching overhead of the frequency scaling because the overhead is negligible compared to the execution time and scheduling period. Once a task i is completed, the total system value increase by v_i which represents the importance of the task.

In our model, the system is equipped with a set of sleep modes denoted by $Z = \{z_1, z_2 ..., z_K\}$. Different sleep modes correspond to the decision of how many hardware components (e.g., CPU, volatile memory and hardware clocks) are turned off. When the system enters a sleep mode z_j, the power consumption is $p(z_j)$, and the switch overhead is $o(z_j)$. Hence, for example, the energy consumption of the system sleeping t time unit in mode z_j is $p(z_j) \times t + o(z_j)$. Each task can enter any sleep mode for an arbitrary time unit after the task is executed, and zm_i represents the sleep mode into which task i enters. Moreover, the system is assumed to harvest energy over time. Initially, the available energy is E_0 at time 0, and the system gains $h(i)$ energy from the environment at time i.

Given a set of tasks, $T = \{t_1, t_2, ..., t_N\}$, a *schedule* is a set of selected tasks and configuration assignments, $(S, sf_i, st_i, zt_i, z_i, zm_i)$, which indicates that for each task $i \in S$, the operating frequency level of the task is sf_i; the scheduled time of it is st_i; the task sleeps at zt_i; the task sleeps z_i time; the task enters zm_i sleep mode. A schedule is *feasible* if it satisfies the following three constraints. The *energy*

TABLE I: Summary of Notations

Notation	Description
Input	
D	Global scheduling period
E_0	Initial energy budget in the system
N	Number of available tasks
K	Number of sleep modes
M	Number of speed levels
T	Set of tasks
F	Set of frequency levels
Z	Set of sleep modes
$r(i)$	Ready time of task i
$v(i)$	Value of task i
$t(i,k)$	Time used by task i under speed level k
$e(i,k)$	Energy used by task i under speed level k
$p(j)$	Power consumption at sleep mode j
$o(j)$	Backup overhead when sleep mode is j
$h(i)$	Energy harvested by the system at time i
Output	
S	Set of selected tasks
sf_i	Speed level of task i
st_i	Scheduled time of task i
zt_i	Sleeping Start time of task i
z_i	Sleep time of task i
zm_i	Sleep mode of task i

constraint in Equation (1) imposes a limitation so that the total energy consumed not exceeds the cumulative available energy harvested by the system at any time.

$$\begin{gathered} \forall\ t,\ where\ 1 \leq t \leq D \\ \sum_{i \in S, st(i) \leq t} e(i, sf_i) + \sum_{j \in S, zt_j \leq t} z_j * p_{zm_j} + o(z_{zm_j}) \\ \leq E_0 + \sum_{l \leq t} h(l) \end{gathered} \tag{1}$$

Equation (2) ensures that total execution time of scheduled tasks and sleep time does not exceed than the scheduling period.

$$\sum_{i \in S} t(i, sf_i) + zt_i \leq D \tag{2}$$

Equation (3) ensures that the tasks will not be scheduled before they are ready.

$$\forall i \in S,\ st_i \leq r(i) \tag{3}$$

We define the *nonvolatile processor task scheduling (NVPTS) problem* as follows.

Instance: A device processor equipped with a set of frequency levels, $F = \{f_1, f_2, ..., f_M\}$, and a set of sleep modes, $Z = \{z_1, z_2 ..., z_K\}$, where the switch overhead and power consumption of each sleep mode z_i are $o(z_i)$ and $p(z_i)$ respectively; a set of tasks $T = \{t_1, t_2, ..., t_N\}$, where each task t_i is associated with a ready time $r(t_i)$, a value v_{t_i}, required energy $e(t_i, k)$ and required time $t(t_i, k)$ under an operating frequency k; the initial energy is E_0, and harvested energy $h(t)$ at time t; a scheduling period D.

Objective: To derive a feasible schedule $\lambda(i) = \{(S, sf_i, st_i, zt_i, z_i, zm_i) | i = 1, 2, ..., N\}$ such that $\sum_{i \in S} v_i$ is maximized.

III. Value-based Nonvolatile Processor Task Scheduling

In this section, we consider the NVPTS problem. First, in section III-A, we show that the problem is NP-hard. Second, in section III-B, we present a dynamic-programming algorithm which resolve the problem in pseudo-polynomial time. Finally, in section III-C, we propose an approximation algorithm based on the dynamic-programming algorithm by rounding input data, where the value difference between optimal solution and approximate solution is bounded.

A. Problem Hardness

In this section, we first show that a NP-hardness of NVPTS by a reduction from the NP-hard problem, which maximize rewards while guaranteeing time and energy constraints (MRGTE) [10].

Theorem 1. *The NVPTS problem is NP-hard.*

Proof: We first describe the input for MRGTE. Given a global deadline D, an energy budget E_{max} and a set of non-preemptive tasks, $T = \{t_1, t_2, ..., t_N\}$, which executing task i at frequency k requires $t_{i,k}$ time unit and $e_{i,k}$ energy unit. The system value increases by v_i when task i is completed. The problem is to find a set of tasks $S \subseteq T$ and a configuration of sf_i for all $i \in S$ in order to maximize system value while subject to energy and time constraints. The problem can be formally rewritten as

$$\begin{aligned} \text{Maximize} \quad & \sum_{i \in S} v_i \\ \text{Subject to} \quad & \sum_{i \in S, j \in Z} t(i, s_j) \leq D \\ & \sum_{i \in S, j \in Z} t(i, s_j) \leq E_{max} \end{aligned} \tag{4}$$

Given an instance $I = \langle T, D, E_{max}, F, t(i,k), e(i,k) \rangle$ of the MRGTE problem, we now construct an instance $I' = \langle T', D', E_0', F', Z', r'(i), t'(i,k), e'(i,k), p'(j), o'(j), h'(i) \rangle$ of our problem in polynomial time. The construction procedure is described as follow. We assign $T', D', F', t'(i,k), e'(i,k)$ and E_0 as $T, D, F, t(i,k), e(i,k)$ and E_{max} respectively. Without loss of generality, we assume that each task of I' is ready at the beginning ($r(i) = 0$). Similarly, the harvested energy of I' is assigned as zero ($h(i) = 0$). There is only one sleep mode (i.e, $|Z| = 1, p(1) = 1$ and $o(1) = 1$). To complete the proof, we now present a polynomial time procedure f such that any solution S to the MRGTE problem is optimal if and only if $f(S)$ is an optimal solution to the NVPTS problem.

Given an optimal solution S to the MRGTE problem, an algorithm which is denoted by f constructs an optimal solution $f(S)$ to the NVPTS problem correspondingly. f assigns the scheduled time and the speed level for each task as the configuration of S, and let sleep time of all tasks be zero. Obviously, f is a polynomial time algorithm. Next, we prove that S is optimal if and only if $f(S)$ is optimal. First of all, we suppose that S is optimal, and $f(S)$ is not optimal. If $f(S))$ is not a optimal solution, there exists a set of task $M \nsubseteq f(S)$ can replace a set of tasks $L \subseteq f(S)$ so that system value of $f(S) - L + M$ is greater than the system value of $f(S)$, and constraints of the NVPTS problem sustain. If this being the case, S will also not be optimal because $S - L + M$ will also be greater than S and the time and energy constraints of the MRGTE problem will hold for NVPTS problem. This contradicts the supposition. On the other hand, we suppose that $f(S)$ is optimal, and S is not optimal. Similarly, there exists a set of task M and a set of task L such that system value of $S - L + M$ is greater than the system value of S. The supposition is also violated because $f(S) - L + M$ is greater than $f(S)$, and all the constraints of the NVPTS problem hold since all sleep time and ready time of tasks are set to zero. Since both the construction procedure and f can be executed in polynomial time, we conclude that the NVPTS problem is NP-hard.

■

B. A Pseudopolynomial-time Optimal Algorithm

In this section, we present a dynamic-programming algorithm which runs in pseudo-polynomial time. Then, we analyze its time complexity, and prove its optimality. Finally, we provide a simple example to better explain the algorithm.

1) Algorithm Description: The algorithm is based on the recursive formula in Equation (5). $V(i, j)$ is defined as the maximal value while i time unit and j energy unit are used without violating the constraints of the NVPTS problem. Note that every $V(i, j)$ corresponds to a combination of output, and the output of $V(i, j)$ is derived from a intermediate result $V(m, n)$. Our objective is to find the maximal $V(i, j)$ among all possible values for i and j. Before explaining the formula, we define some term. Let $O_{i,j,m,n}$ denotes a set of possible scheduling operations such as executing a task or entering a sleep mode while all constraints sustain. Specifically, executing a task k with the frequency s_k is included in $O_{i,j,m,n}$ if **a)** executing the task requires $i - m$ time and $j - n$ energy; **b)** the task must not be selected in the intermediate sequence $V(m, n)$; **c)** the ready time of the task $r(k)$ must not be later than m, which is the time to execute the task; **d)** $j \leq E_0 + \sum_{l \leq m} h(l)$ because the consumed energy can not exceed the harvested energy. On the other hand, entering the sleep mode z at time m is included in $O_{i,j,m,n}$ if $j - n = p(z) * (i - m) + o(z)$ and $j \leq E_0 + \sum_{l \leq m} h(l)$ according to the constraints on energy consumption and time. Note that the values of sleep operations are set to zero, and a task can only enter a sleep mode under the condition that the previous intermediate result, $V(m, n)$ is not sleeping. Besides $O_{i,j,m,n}$, let q_u be the reward of each scheduling operation $u \in O_{i,j,m,n}$. Given the i, j, m, n, Equation (6) returns the operation with maximum reward among all instances of $O_{i,j,m,n}$.

We now explain the recursive Equation (5) and discuss how to find the maximal value.

$$V(i,j) = \begin{cases} 0, & \text{if } i = j = 0 \\ \max\limits_{0 \le m < i, 0 \le n < j} \{V(m,n) + mv(i,j,m,n)\} & otherwise \end{cases} \tag{5}$$

$$mv(i,j,m,n) = \max_{u \in O_{i,j,m,n}} (q_u) \tag{6}$$

1. If $i = j = 0$, the $V(0,0)$ is the base case of our problem so that the value of $V(i,j)$ is set as 0 because of no tasks placed initially.
2. Otherwise, we try to find the the maximal value for $V(i,j)$ derived by either executing a task or entering a sleep mode from $V(m,n)$. We need to go through all possible tasks and sleep modes while all the constraint mentioned above are satisfied. Thus, the possible values for m, n, i and j are 0 to $i-1$, 0 to $j-1$, 0 to D and 0 to $E_0 + \sum\limits_{l \le m} h(l)$ respectively. The recursive formula find the maximum value of $V(m,n) + mv(i,j,m,n)$ among the values so that all possible operations deriving $V(i,j)$ are examined. By considering all of them, the value of $V(i,j)$ is derived.

Algorithm 1 implements a dynamic programming based on the recursive formula which is shown in Equation 5, and two three-dimensional tables, $S[i,j,\tau]$ and $F(i,j,\tau)$, represent the scheduling time and the speed of tasks respectively according to the schedule sequence of each subproblem. In order to keep some important information, some variables are used to derived scheduling sequences. MAX and COM indicate the value and constraints of the optimal solution. In the beginning of the algorithm, MAX and COM are initialized to $-\infty$ and $\emptyset$ respectively because there is no feasible schedule found yet. In the for loop (Line 3-9), the algorithm initializes S and F, and then finds the maximum value by invoking $V(i,j)$ all possible values. Once another maximal value is found, MAX and COM are updated to keep the track of the optimal solution. In Procedure $V(i,j)$, $V[i,j]$ denotes the solution of $V(i,j)$ to each subproblem, and $V[i,j]$ is initialized to $-\infty$ because no answer is found yet. The procedure simply returns 0 if $i = j = 0$ because this is the base case of our problem. Otherwise, the procedure derives the returned value of $V(i,j)$ according to the presented recursive functions. When another maximum value is found, the procedure keep updating the value (Line 20) and sequence (Line 21-25), including how the task is run or how the system sleep. In each iteration, procedure $mv(i,j,m,n)$ is called to find the scheduling operation constructing $V[i,j]$ from $V[m,n]$ with maximum reward. Note that we update all the variables related to the result since we do not know what kind of the operation is returned from $mv(i,j,m,n)$. Procedure $mv(i,j,m,n)$ implements Equation (6) which returns the value of the optimal operation. In the beginning of the procedure, the value, denoted by $opmax$, is initialized to $-\infty$; the selected task, $optasks$, is initialized to 0; the speed of the task, $taskf$, is initialized to 0. Then, in order to find the optimal operation such that the presented constraints are satisfied, the procedure searches all possible task execution (Line 31-37) and system sleep (Line 38-41). Finally, the optimal operation, $opmax$, is returned.

Algorithm 1

```
Input: $I = (T, F, e(), v(), t(), r(), Z, o(), p(), h(), E_0), D$
Output: The value, energy and time of an optimal schedule $\lambda$
 1: $MAX \leftarrow -\infty$
 2: $COM \leftarrow -\emptyset$
 3: for $i \leftarrow 0$ to $D$ do
 4:     for $j \leftarrow 0$ to $E_0 + \sum_{l \le i} h(l)$ do
 5:         $S[i,j] \leftarrow -1$
 6:         $F[i,j] \leftarrow 0$
 7:         if $MAX < V(i,j)$ then
 8:             $MAX \leftarrow V[i,j]$
 9:             $COM \leftarrow i, j$
10: return $MAX$,$COM$
```

```
Procedure $V(i,j)$
11: $V[i,j] = -\infty$
12: if $i = j = 0$ then
13:     $V[i,j] = 0$
14: else
15:     $E \leftarrow E_0$
16:     for $m \leftarrow 0$ to $i-1$ do
17:         $E \leftarrow E + h(m)$
18:         for $n \leftarrow 1$ to $j-1$ do
19:             if $E > j \wedge V[i,j] < V[m,n] + mv(i,j,m,n)$ then
20:                 $V[i,j] \leftarrow V[m,n] + mv(i,j,m,n))$
21:                 $S[i,j] \leftarrow S[m,n]$
22:                 $S[i,j,optask] \leftarrow m$
23:                 $F[i,j] \leftarrow F[m,n]$
24:                 $F[i,j,optask] \leftarrow taskf$
25:                 $Z[i,j] \leftarrow sleep$
26: return $V[i,j]$
```

```
Procedure $mv(i,j,m,n)$
27: $opmax \leftarrow -\infty$
28: $optask \leftarrow 0$
29: $taskf \leftarrow 0$
30: $sleep \leftarrow 0$
31: for $\tau \leftarrow 1$ to $|T|$ do
32:     if $S[m,n,t] = 0 \wedge r(\tau) \le m$ then
33:         for $f \leftarrow 1$ to $|F|$ do
34:             if $j - n = e(\tau,f) \wedge i - m = t(\tau,f) \wedge opmax < v_(\tau)$
                then
35:                 $opmax = v_(\tau)$
36:                 $optask \leftarrow \tau$
37:                 $taskf \leftarrow f$
38: for $z \leftarrow 1$ to $|Z|$ do
39:     if $opmax < 0 \wedge Z[m,n] = 0 \wedge j - n = p(z) * (i-m) + o(z)$
        then
40:         $opmax \leftarrow 0$
41:         $sleep \leftarrow 1$
42: return $opmax$
```

We now describe how to construct an optimal sequence λ, after the S, F and V have been derived. Let λ be an empty set. First, the optimal value for i and j can be derived from COM, which locates the optimal value for $V[i,j]$. Second, each task τ, where $F[i,j,\tau] > 0$, is executed at frequency level $F[i,j,\tau]$ at time $S[i,j,\tau]$. Finally, for each time slot where the operation of it is not determined (i.e., the time between task execution), a task sleeping fitting in the time slot is found and added to λ. Through the above procedure, the optimal solution is derived, and the procedure requires $O(N)$ time to arrange task executing; also, the procedure requires $O(NZ)$ time to find the sleeping operations. To sum up, it takes $O(NZ)$ to construct λ based on tables S, F and V.

2) Properties: Now, we analyze the time complexity and prove the optimality of the proposed algorithm. To simply the representation, we let E_t denote the total energy which is $E_0 + \sum\limits_{l \leq D} h(l)$.

Lemma 1. *The time complexity of Algorithm 1 is $O([DE_t]^2[NM+K])$.*

Proof: Before analyzing the time complexity of Algorithm 1, we discuss the time complexity of Procedure $V(i,j)$ and Procedure $mv(i,j,m,n)$. The time complexity of Procedure $mv(i,j,m,n)$ is $O[NM+K]$, because the procedure tries to schedule a task by trying all combination of tasks and available frequency levels; also, the procedure tries to find a sleep mode such that the system can sleep from m time to i time. Procedure $V(i,j)$ derives the solution to $V[i,j]$ from the solutions of subproblems. The number of all subproblems are $O(DE_t)$ because $i \leq D$ and $j \leq E_t$, and finding a solution to a subproblem takes $O[NM+K]$. Therefore, the time complexity of Procedure $V(i,j)$ is $O([DE_t][NM+K])$. At last, by searching all possible values for i and j, Algorithm 1 returns the optimal solution. Given the time complexity of $V(i,j)$ is $O([DE_t][NM+K])$, we conclude that the time complexity of Algorithm 1 is $O([DE_t]^2[NM+K])$. ■

Theorem 2. *Algorithm 1 is a pseudopolynomial-time optimal algorithm for the NVPTS problem.*

Proof: Proving the theorem equals to proving the correctness of the dynamic-programming formula $V(i,j)$. We prove the formula to be correct through mathematical induction on index i, j. The induction basis is $V(0,0)$, and the $V(0,0) = 0$ according to Equation (5). The basis is correct because the number of scheduled task is 0 initially. Next, we suppose that the formula to be correct for $V(m,n)$ where $0 \leq m < i$ and $0 \leq n < j$. We show that $V(i,j)$ is also correct. Without loss of generality, we assume that there exists an optimal schedule sequence $\lambda = \{k_1, k_2, ..., k_{|\lambda|}\}$ where the scheduled time of k_a is later than the scheduled time of k_{a-1}; the value of scheduling sequence λ is V_λ; moreover, executing all $k \in \lambda$ requires i time and j energy. Let $\lambda' = \{k_1, k(2)...., k(|\lambda|-1)\}$ such that the value of sequence λ' is $V_{\lambda'}$, and executing all $k \in \lambda'$ requires m time and n energy. Obviously, $0 \leq m < i$ and $0 \leq n < j$. According to the hypothesis, we know that $V(m,n)$ is correct, and thus $V_{\lambda'} \leq V(m,n)$. In addition, $v_{k_\lambda} \leq mv(i,j,m,n)$ because $mv(i,j,m,n)$ examines all the possible operations using $i-m$ time and $j-n$ energy. Considering all discussion above, The value difference between V_λ and $V(i,j)$ is $V_\lambda - V(i,j) = V_{\lambda'} + v_{k_\lambda} - V(m,n) - mv(i,j,m,n) \leq 0$. This implies that the recursive formula is correct since the value of constructed schedule derived by running Algorithm 1 is greater or equal than the optimal result. Thus, Algorithm 1 is optimal for the NVPTS problem. ■

3) A Simple Example: In this section, we provide a simple example to better explain Algorithm 1. Consider the input instance given in figure 1, we now show how Algorithm 1 derive the optimal sequence shown in figure 1. Figure 2 shows the table V completed by Algorithm 1 for the example. Let us consider three entries, $V[11,120]$, $V[19,130]$ and $V[40,260]$. After each table entry is initialized as $-\infty$, $V[0,0]$ is assigned to 0 since it is the base case of our problem. Then, the algorithm finds the maximum value for each entry. $V[11,120] = 5$ can be derived as follows. We begin with the table entry $V[0,0]$ and examine each entry $V[m,n]$, where $0 \leq m < i$ and $0 \leq n < j$, and find the operation, o, such that $V[m,n] + v_o$ is maximized. Thus, task 1 is scheduled at $V[0,0]$, and $V[11,120] = V[0,0] + v_1 = 5$. Similarly, for entry $V[19,130]$, the value of it is derived from $V[11,120]$ by entering sleep mode 2 from time 11 to 19. At last, considering $V[19,130] = 5$, an maximum value of $V[40,260]$ is derived by scheduling task 3 at time 19 since the total energy is enough by harvesting energy at time 19. Through examining all entries, the optimal sequence shown in 1 is found by Algorithm 1.

j \ i	0	1		11		19		40	
0	0	-∞	⋯	-∞		-∞		-∞	
1	-∞	-∞		-∞	⋯	-∞		-∞	
	⋮		⋱	…		…		…	
120	-∞	-∞	…	5		-∞	⋯	-∞	⋯
		⋮			⋱	…		…	
130	-∞	-∞	…	-∞	…	5		-∞	
				⋮			⋱	…	
260	-∞	-∞	…	-∞	…	-∞	…	13	
					⋮				⋱

Fig. 2: Table V constructed by Algorithm 1

C. A Rounding Approximation Algorithm

In this section, we present an approximation algorithm, Algorithm 2, based on the presented Algorithm 1. The reason why Algorithm 1 is a pseudopolynomial-time algorithm is that the number of subproblems is related to D and E_{max}. Thus, in order to improve the efficiency of the dynamic-programming algorithm, we prune all variables which is related to time and energy, to the multiple of an integer R so that number of subproblems are reduced. We describe Algorithm 2 as follows, and then analyze the time complexity and approximation ratio of the algorithm. At last we provide a simple example to demonstrate the idea of this algorithm.

1) Algorithm Description: Algorithm 2 implements an approximation algorithm for the NVPTS problem by applying Algorithm 1 to the rounded instance. Algorithm 2 prunes the available energy (Line 1), the scheduling period (Line 2), the energy consumption of task (Line 3), execution time of tasks (Line 4), ready time of tasks (Line 5), switch overhead of sleep modes (Line 6) and the harvested energy (Line 7). Finally, the result is obtained by applying Algorithm 1 to the pruned variables. It's obvious that the execution time of applying to pruned variables to Algorithm 1 is effectively reduced. Note that in order to apply Algorithm 1 to rounded input, some modifications of the algorithm, such as handling boundary values of variables, have to be made, and a few constraints have to be slightly relaxed according to R in a practical implementation. Next, we discuss the time complexity and the error bound of the approximate solution while the size of a round is R. Before entering the discussion, we define some terms. Let M be the maximal value of the task where the required energy and time of the task are not greater than the D and E_t respectively. In addition, let C be the minimal required energy or time to execute a task. Note that both M and C can be found in polynomial time by searching all variables sequentially.

Algorithm 2

```
Input: $I = (R, T, F, e(), v(), t(), r(), Z, o(), p(), h(), E_0, D)$
Output: The value, energy and time of an optimal schedule $\lambda$
1: $E'_0 \leftarrow \lceil E_0/R \rceil$
2: $D' \leftarrow \lceil D/R \rceil$
3: $e'() \leftarrow \lceil e()/R \rceil$
4: $t'() \leftarrow \lceil t()/R \rceil$
5: $r'() \leftarrow \lfloor r()/R \rfloor$
6: $o'() \leftarrow \lceil o()/R \rceil$
7: $h'(i) \leftarrow \left\lceil (\sum_{k \le R*i} h(k) - \sum_{k<i} h'(k-1) * R)/R \right\rceil$
8: $I' = (T, F, e'(), v(), t'(), r'(), Z, o'(), p(), h'(), E'_0, D')$
9: return $Algorithm\ 1(I')$
```

2) Properties:

Lemma 2. *The time complexity of Algorithm 2 is $O([DE_t/R^2]^2[NM + K])$.*

Proof: The time complexity of Algorithm 2 can be represented as $O(R_I + A_I)$ where R_I is the time complexity of rounding all variables; A_R is the time complexity of applying Algorithm 1 to the rounded variables. According to Lemma 1, the time complexity of A_R is $O([D'E'_t]^2[NM + K])$ where $E'_t \leftarrow \lceil E_t/R \rceil$; $D' \leftarrow \lceil D/R \rceil$. Thus, the time complexity of A_R can be rewritten as $O([DE_t/R^2]^2[NM + K])$. We now derive the time complexity of $O(R_I)$. The time complexity can be calculated as the sum of the following parts. First, the time complexity of rounding E_0 and D is $O(1)$. Second, it takes $O(NM)$ time to round e and t since there are N tasks and M available frequency levels. Third, rounding the switch overhead takes $O(K)$ time due to existing K sleep modes. Fourth, it costs $O(E_t)$ time to prune all harvested energy. At last, the time of rounding r for all tasks is $O(N)$. Therefore, $O(R_I)$ can be denoted by $O(1 + NM + K + N + E_t) = O(NM + K + E_t)$. By considering all of previous statements, $O(R_I + A_I) = O([DE_t/R^2]^2[NM+K]) + O(NM + K+E_t)$. Thus, we can conclude that the time complexity of Algorithm 2 is $O([DE_t/R^2]^2[NM + K])$. ■

Theorem 3. *The approximation ratio of Algorithm 2 is $\frac{C}{(2C+NR)}$.*

Proof: The concept behind this proof is that when Algorithm 1 is applied to the rounded input, scheduling a task will cost at most R extra energy and time due to the overestimated execution time and ready time. In the following proof, we assume that the approximate solution is X, and the value of the solution is V_X. Without loss of generality, we also assume that there is an optimal solution S' where the the set of scheduled tasks is $S' = \{s_1, s_2, ..., s_k\}$, and the value of the solution is V_{opt}. Now, we can find a task $t \in S'$ such that Equation 7 holds. In order to simplify the context, we define that $A = \{s_1, s_2, ..., s_{t-1}\}$; $B = \{s_{t+1}, s_{t+2}, ..., s_k\}$; $V_A = \sum_{i \in A} v_i$; $V_B = \sum_{i \in B} v_i$. To complete the proof, we now show how Equation (8) is derived. We know that $V_X \ge V_A$ because V_A is a feasible sequence for the Algorithm 2. Also, $kR \times M/C \ge V_B$ because the highest value density in terms of required energy or time is M/C, and thus the value of the set is less than or equal to $kR \times M/C$. Since the M and k respectively denote the maximal task value and number of scheduled tasks, it is true that $M < V_X$ because placing the task with value M is a feasible sequence; $k < N$ because the number selected tasks will not exceed the number of tasks. By considering all equations above, Equation (8) can be derived, and thus the error is proved to be smaller than $\frac{C}{(2C+NR)}$.

$$\begin{aligned} &\sum_{i<t} e_i \le E_t - kR \wedge \sum_{i<t} t_i \le D - kR \wedge \\ &(\sum_{i>t, i \in S'} e_i \le kR \vee \sum_{i>t, i \in S'} t_i \le kR) \end{aligned} \tag{7}$$

$$\begin{aligned} &V_X \ge V_A \\ &V_X + kR * M/C \ge V_A + V_B \\ &V_X + kR * M/C + M \ge V_A + V_B + v_{s_t} \\ &(2 + (kR/C))V_X \ge V_{opt} \\ &V_X \ge \frac{C}{(2C + NR)} V_{opt} \end{aligned} \tag{8}$$

■

Remarks: Obviously, the execution time of Algorithm 2 can be much better than Algorithm 1 by rounding some input variables. However, the error ratio of the approximate solution increases when R increases. According to Theorem 3, the error ratio of Algorithm 2 is close to 0 if R outnumbers C. In contrast, the error ratio is close to $\frac{1}{2}$ if C outnumbers R. In a practical implementation of a offline scheduling, in order to increase the precision of the result, the granularities of time and energy are usually very small (i.e., μsec, μJ). Therefore, C is usually a very large number in order to represent the required energy or time of a task, and thus this implies that there is a great flexibility to choose proper R according to the demands of simulation preciseness and system performance.

3) A Simple Example: To better understand the idea of Algorithm 2, we provide a simple example and demonstrate how the algorithm works when R is 2. We take the instance shown in figure 1 as the input of this example. Before invoking

Algorithm 1, some input variables are rounded, so D, E_0, $o(2)$ are trimmed to 25, 100, 1 respectively; also, $t(1,1)$, $t(2,1)$, $t(3,1)$, $r(2)$, $r(3)$ are rounded to 6, 12, 11, 5, 9 respectively; moreover, 50 energy is harvested at time 10. The table in Fig. 3 shows the table V derived by Algorithm 2. Similar to the procedure in section III-B3, the optimal sequence is derived by executing task 1 at time 0, entering sleep mode 2 from time 6 to 10, and executing task 3, and thus the optimal sequence required $42(21 \times R)$ time and $260(130 \times R)$ energy. However, the required time is higher than the result in Fig. 2, because the execution time of task 1 and task 3 are overestimated after we round the variables. Note that as R grows, the errors on time and energy increases, and thus the total value of approximate result may decrease correspondingly.

j \ i	0	1		6		10		21	
0	0	-∞	…	-∞		-∞		-∞	
1	-∞	-∞		-∞	…	-∞		-∞	
⋮			⋱	…		…		…	
60	-∞	-∞	…	5		-∞	…	-∞	…
			⋮		⋱	…		…	
65	-∞	-∞	…	-∞	…	5		-∞	
				⋮			⋱	…	
130	-∞	-∞	…	-∞	…	-∞	…	13	
					⋮				⋱

Fig. 3: Table V constructed by Algorithm 2

IV. Performance Evaluation

A. Experimental Setup

We performed a series of simulations and experiments to demonstrate the usability and performance of the proposed algorithms. Our experiments are composed of 2 parts. First, we evaluated the output sequences of the proposed algorithms by executing the sequences on a device produced by TI, MSP430FR5969 [4], and show the difference between simulation results and execution traces of the device. Second, to investigate the trade-off for both presented algorithms, we applied randomized input instances to the proposed algorithms, and compared the output value and running time of the algorithms.

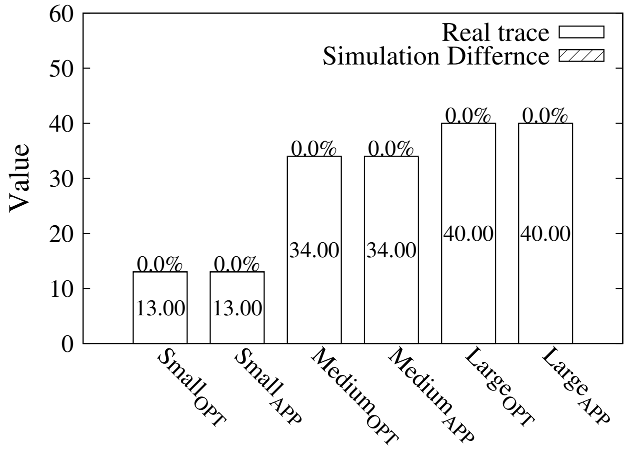
The device features 64 KB non-volatile memory, FRAM, so that the progresses of tasks, such as counter, value and variables do not loss when a power failure occurs. Also, it is equipped with 7 sleep modes and 8 frequency levels. To measure the power consumption and switch overhead of each sleep mode, we used an energy analysis tool, EnergyTrace technology [2], produced by Texas Instruments to observe the energy consumption of MSP430FR5969. Specifically, for each sleep mode, we toggle the power mode of the device 15000 times in a short time via setting the bits of the Status Register, and observed the change in the energy consumption of the device.

We compared the optimal and approximation algorithms, respectively denoted as OPT and APP. For each input instance, both algorithms derive a feasible schedule, in which the start time, speed level, sleeping start time, sleep time and sleep mode of each task are determined. We validated the proposed algorithms by applying the output sequences to the device, and measured the differences between the simulation results and real traces. Furthermore, to study the impact of R on task scheduling, we compare the results derived by the approximation algorithm with various R. Both algorithms trade off the total system value against the required energy; thus, the performance metrics were total value and energy consumption. The total value is defined as the sum of all completed tasks in a sequence. The energy consumption is defined as the required energy of the sequence. That is, the consumed energy of all sleep and task execution in a sequence.

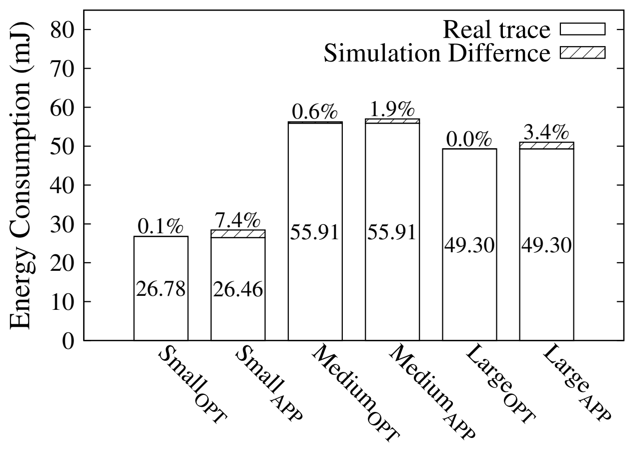
In order to consider different aspects of applications and workloads, we implemented three tasks for the experiment based on existing benchmarks and applications [3, 5]. The tasks are matrix multiplication, sensing temperature and calculating PI which represent memory-intensive, I/O intensive and computation-intensive applications respectively. In the simulations, to model these tasks accurately, the granularity for time and energy is set as 20 ms and 0.1 mJ respectively. Note that the granularity depends on applications' requirements and system characteristics. Considering different scenarios, we targeted three task sets (i.e., small, medium and large), all of which comprised of the implemented tasks, and each task set are randomly generated corresponding to a range of utilization factor. Note that a low utilization system has a higher chance to sleep longer compared to a high utilization. We choose D = 1000 and E = 600 for medium and large task sets; however, for small task sets, we choose D = 400 and E = 300 since the system utilization is rather small. After both algorithms determined schedule sequences for each task task, we ran the sequences on the device and observed the difference between our simulations and real traces. Also, to test various aspects of workloads, 100 task sets were randomly generated and as input instances for each scenario. We then applied the dynamic-programming algorithm and approximation algorithm with various rounding parameters to each task set. The values for R are in the range 10 to 100. By comparing the performance metrics and execution time, we observed the trade-off between performance metrics and R.

B. Experimental Results

1) Real Platform Validation: Figure 4(a) and 4(b) respectively show the total value and total energy consumption under the pseudopolynomial-time optimal algorithm (OPT) and the approximation algorithm (APP), where the rounding parameter, R, is 10; also, the differences between simulation results and the real traces are shown in these figures. In figure 4(a), the total values under both algorithms are identical in each scenario. The average energy differences are 0.02 mJ, 0.35 mJ and 0 mJ for small, medium and large scenarios respectively. The differences in energy consumption grow when approximate results are applied to the device. As shown in Figure 4(b), the differences are respectively 1.96 mJ, 1.09 mJ and 1.7 mJ for small, medium and large scenarios. Our results show that the simulation differences of the optimal results and approximate

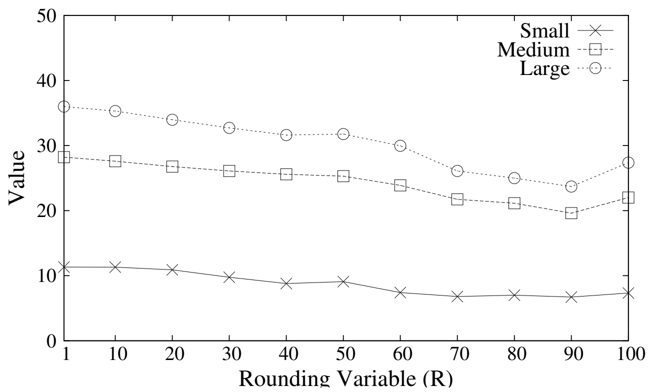


(a) Total value

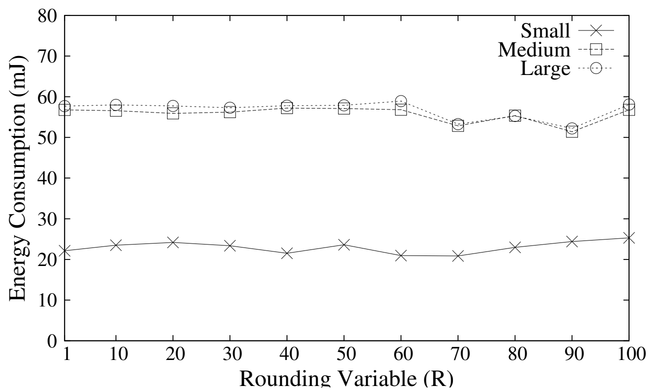


(b) Energy consumption

Fig. 4: Results based on a real device



(a) Total value



(b) Energy consumption

Fig. 5: The impact of different rounding variables on the value and the energy

results are respectively less than 1% and 8% for each case, this validate the usability of the proposed algorithms for the real devices.

2) Performance Comparision: We considered each input scenario and observed the trade-off between output value and running time. We randomly generated 100 task sets for each input scenario according to the pattern of the three targeting tasks. For each task set, we then compared the output sequences derived by Algorithm 1 and 2 with various R. Figure 5 shows the output value of the approximation in respect of different rounding variables and scenarios. As shown in figure 5(a), in general, when the rounding variable is small, e.g. 10, the output values are about 99%, 98% and 98% respectively comparing to each optimal case. However, as rounding variable grows, the value declines as well. If we choose a larger R, e.g. 100, the output will then be about 66%, 79% and 76% respectively. Figure 5(b) shows the energy consumption of the results with different rounding variables R. The rounding variable does not affect the total energy consumption much, but from the declination of tasks executed indicated in figure 5(a), we can know that the overhead of rounding has affected the output. That is, rounding variables degrade the preciseness of measuring energy consumption.

3) Overhead Measurements: Figure 6 shows the speed-up of different rounding variables. With Algorithm 2 applied, the speed-up of running time is significant. For example, when R is 20, the average speed-up is about 20,000, 63,000 and 74,600 times in respect of different scenarios. To be more specifically, compared to the average running times for Algorithm 1 which respectively are 53 minutes, 37 hours and 71 hours, the average running times for Algorithm 2 are 0.165, 2.1 and 3.5 seconds respectively. That is, the trade-off between running time and output value will be the main consideration when choosing rounding variable R.

V. Concluding Remarks

This paper presents a pseudopolynomial-time optimal algorithm maximizing QoS represented by value for a DVFS-enabled nonvolatile processor under resource constraints. The algorithm constructs an optimal operation sequence which is a configuration of task execution and system sleeping. To reduce

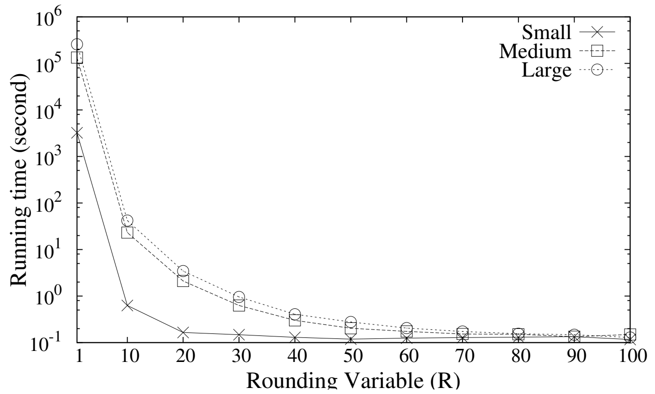


Fig. 6: The impact of different rounding variables on the running time

running time of the pseudopolynomial-time algorithm, we present an approximation algorithm which applies the rounded input instance to the algorithm, and derive the approximation ratio of the approximation algorithm. We show the usability of the proposed algorithms by executing the output sequences on a device equipped with FRAM produced by Texas Instruments, and the results show that the differences between simulations and real traces are respectively 0.1%, 0.6% and 0.0% for small, medium and large tasks sets, which represent the scenarios under different workloads. Compared to the pseudopolynomial-time optimal algorithm, the experiment results show that the approximation algorithm reduced the running time from 2 days to 3 seconds and bring less than 3% value degradation.

REFERENCES

[1] Facts and Statistics on Wearable Technology. http://www.statista.com/topics/1556/wearable-technology/.

[2] MSP EnergyTrace Technology. http://www.ti.com/tool/energytrace.

[3] MSP430 Competitive Benchmarking. http://www.ti.com/lit/an/slaa205c/slaa205c.pdf.

[4] MSP430FR5969 LaunchPad. http://www.ti.com/tool/msp-exp430fr5969.

[5] MSP430FR5969 LaunchPad Out-of-box Demo. http://software-dl.ti.com/msp430/msp430_public_sw/mcu/msp430/MSP-EXP430FR5969/latest/index_FDS.html.

[6] A. Ravinagarajan, D. Dondi, and T. S. Rosing. DVFS Based Task Scheduling in a Harvesting WSN for Structural Health Monitoring. In *Proc. of IEEE DATE*, pages 1518–1523, 2010.

[7] C. Moser, J.-J. Chen, and L. Thiele. Reward Maximization for Embedded Systems with Renewable Energies. In *Proc. of ACM/IEEE RTCSA*, pages 247–256, 2008.

[8] C. Moser, J.-J. Chen, and L. Thiele. Power Management in Energy Harvesting Embedded Systems with Discrete Service Levels. In *Proc. of ACM/IEEE ISLPED*, pages 413–418, 2009.

[9] C. Moser, L. Thiele, D. Brunelli, and L. Benini. Adaptive Power Management for Environmentally Powered Systems. *IEEE Trans. on Comput.*, 59(4):478–491, 2010.

[10] C. Rusu, R. Melhem, and D. Mosse. Maximizing the System Value While Satisfying Time and Energy Constraints. In *Proc. of IEEE RTSS*, pages 246–255, 2002.

[11] D.-M. Zhang, S.-C. Li, A. Li, Y.-P. Liu, X. S. Hu, and H.-Z. Yang. Intra-task Scheduling for Storage-less and Converter-less Solar-powered Nonvolatile Sensor Nodes. In *Proc. of IEEE ICCD*, pages 348–354, 2014.

[12] D.-M. Zhang, Y.-P. Liu, X. Sheng, J.-Y. Li, T.-D. Wu, C. J. Xue, and H.-Z. Yang. Deadline-aware Task Scheduling for Solar-powered Nonvolatile Sensor Nodes with Global Energy Migration. In *Proc. of ACM/IEEE DAC*, pages 126:1–126:6, 2015.

[13] H.-H. Li, Y.-P. Liu, C.-C. Fu, C. J. Xue, D.L. Xiang, J.-S. Yue, J.-Y. Li, D.-M. Zhang, J.-T. Hu, and H-Z. Yang. Performance-Aware Task Scheduling for Energy Harvesting Nonvolatile Processors Considering Power Switching Overheads. In *Proc. of ACM/IEEE DAC*, 2016.

[14] H.-H. Li, Y.-P. Liu, Y.-Q. Wang, R. Luo, and H.-Z. Yang. Using Nonvolatile Processors to Reduce Leakage in Power Management Approaches. In *Proc. of IEEE EDSSC*, pages 1–2, 2014.

[15] J. Chen, T.-Q. Wei, and J.-L. Liang. State-Aware Dynamic Frequency Selection Scheme for Energy-Harvesting Real-Time Systems. *IEEE Tran. on VLSI*, 22(8):1679–1692, 2014.

[16] J. K. Dey, J. Kurose, and D. Towsley. On-Line Scheduling Policies for a Class of IRIS (Increasing Reward with Increasing Service) Real-Time Tasks. *IEEE Trans. Comput.*, 45(7):802–813, 1996.

[17] J. Lu, S.-B. Liu, Q. Wu, and Q.-R. Qiu. Accurate Modeling and Prediction of Energy Availability in Energy Harvesting Real-time Embedded Systems. In *Proc. of IEEE IGCC*, pages 469–476, 2010.

[18] J. W. S. Liu, K. J. Lin, W. K. Shih, A. C. -s. Yu, J. Y. Chung, and W. Zhao. Algorithms for Scheduling Imprecise Computations. *Computer*, pages 305–310, 1991.

[19] K.-S. Ma, X.-Q. Li, Y.P. Liu, J. Sampson, Y. Xie, and V. Narayanan. Dynamic Machine Learning Based Matching of Nonvolatile Processor Microarchitecture to Harvested Energy Profile. In *Proc. of IEEE/ACM ICCAD*, pages 526–537, 2015.

[20] K.-S. Ma, Y. Zheng, S.-C. Li, K. Swaminathan, X.-Q. Li, Y.-P. Liu, J. Sampson, Y. Xie, and V. Narayanan. Architecture Exploration for Ambient Energy Harvesting Nonvolatile Processors. In *Proc. of IEEE HPCA*, pages 526–537, 2015.

[21] M.-M. Xie, M.-Y. Zhao, C. Pan, J.-T. Hu, Y.-P. Liu, and C. J. Xue. Fixing the Broken Time Machine: Consistency-aware Checkpointing for Energy Harvesting Powered Non-volatile Processor. In *Proc. of ACM/IEEE DAC*, pages 184:1–184:6, 2015.

[22] N. Guan, M.-Y. Zhao, C. J. Xue, Y.-P. Liu and W. Yi. Modular Performance Analysis of Energy-Harvesting Real-Time Networked Systems. In *Proc. of IEEE RTSS*, pages 65–74, 2015.

[23] N. Roseveare, and B. Natarajan. An Alternative Perspective on Utility Maximization in Energy-Harvesting Wireless Sensor Networks. *IEEE Trans.on Vehicular Technology*, pages 344–356, 2014.

[24] Q. Ju and Y. Zhang. Charge Redistribution-Aware Power Management for Supercapacitor-Operated Wireless Sensor Networks. *IEEE Sensors Journal*, 16(7):2046–2054, 2016.

[25] S.-B. Liu, J. Lu, Q. Wu, and Q.-R. Qiu. Harvesting-Aware Power Management for Real-Time Systems With Renewable Energy. *IEEE Trans. on VLSI*, 20(8):1473–1486, 2012.

[26] S.-B. Liu, Q.-R. Qiu, and Q. Wu. Energy Aware Dynamic Voltage and Frequency Selection for Real-Time Systems with Energy Harvesting. In *Proc. of IEEE DATE*, pages 236–241, 2008.

[27] S. Sudevalayam and P. Kulkarni. Energy Harvesting Sensor Nodes: Survey and Implications. *IEEE Communications Surveys Tutorials*, 13(3):443–461, 2011.

[28] V. Raghunathan, A. Kansal, J. Hsu, J. Friedman, and M. Srivastava. Design Considerations for Solar Energy Harvesting Wireless Embedded Systems. In *Proc. of IEEE IPSN*, pages 457–462, 2005.

[29] X. Lin, Y.-Z. Wang, S.-Y. Yue, N. Chang, and M.-S. Pedram. A Framework of Concurrent Task Scheduling and Dynamic Voltage and Frequency Scaling in Real-time Embedded Systems with Energy Harvesting. In *Proc. of IEEE ISLPED*, pages 70–75, 2013.

[30] X. Pan and R. Teodorescu. NVSleep: Using Non-volatile Memory to Enable Fast Sleep/Wakeup of Idle Cores. In *Proc. of IEEE ICCD*, pages 400–407, 2014.

[31] Y.-K. Zhang, Y. Ge, and Q.-R. Qiu. Improving Charging Efficiency with Workload Scheduling in Energy Harvesting Embedded Systems. In *Proc. of ACM/IEEE DAC*, pages 57:1–57:8, 2013.

[32] Y.-Q. Wang, Y.-P. Liu, S.-C. Li, D.-M. Zhang, B. Zhao, M.-F. Chiang, Y.-X. Yan, B.-K. Sai, and H.-Z Yang. A 3us Wake-up Time Nonvolatile Processor Based on Ferroelectric Flip-flops. In *Proc. of IEEE EDSSC*, pages 149–152, 2012.