Performance-Aware Task Scheduling for Energy Harvesting Nonvolatile Processors Considering Power Switching Overhead

Hehe Li[1], Yongpan Liu[1], Chenchen Fu[2], Chun Jason Xue[2], Donglai Xiang[1], Jinshan Yue[1], Jinyang Li[1], Daming Zhang[1], Jingtong Hu[3] and Huazhong Yang[1]
[1] Tsinghua National Laboratory for Information Science and Technology, Department of Electronic Engineering, Tsinghua University, Beijing, China
[2]Department of Computer Science, City University of Hong Kong, Hong Kong SAR
[3]School of Electrical and Computer Engineering, Oklahoma State University, Stillwater, OK, USA
ypliu@tsinghua.edu.cn, jasonxue@cityu.edu.hk, jthu@okstate.edu

ABSTRACT

Nonvolatile processors have manifested strong vitality in battery-less energy harvesting sensor nodes due to their characteristics of zero standby power, resilience to power failures and fast read/write operations. However, I/O and sensing operations cannot store their system states after power off, hence they are sensitive to power failures and high power switching overhead is induced during power oscillation, which significantly degrades the system performance. In this paper, we propose a novel performance-aware task scheduling technique considering power switching overhead for energy harvesting nonvolatile processors. We first give the analysis of the power switching overhead on energy harvesting sensor nodes. Then, the scheduling problem is formulated by MILP (Mixed Integer Linear Programming). Furthermore, a task splitting strategy is adopted to improve the performance and an heuristic scheduling algorithm is proposed to reduce the problem complexity. Experimental results show that the proposed scheduling approach can improve the performance by 14% on average compared to the state-of-the-art scheduling strategy. With the employment of the task splitting approach, the execution time can be further reduced by 10.6%.

CCS Concepts

•**Computer systems organization** → *Embedded software; Real-time system specification;*

Keywords

Power switching overhead; nonvolatile processors; scheduling; energy harvesting

1. INTRODUCTION

A cyber-physical system (CPS) is an integration of computation, communication and physical processes, where its physical resources are monitored and controlled by a computation and communication core. The CPS has the potential of being adopted in a wide range of scenarios. However, it is limited by the need of frequent battery maintenance. Energy harvesting technology, which utilizes ambient energy sources (e.g. radio-frequency radiation, solar energy, thermal gradient and vibration energy. [1]) has been proposed as a promising alternative for powering CPS due to its free-of-maintenance property. Nevertheless, the unstable nature of these power sources brings a great challenge to the performance of energy harvesting systems. Power variance incurs considerable state loss and power switching overhead when the tasks are executed, which leads to significant performance degradation. Therefore, an energy-driven scheduling policy is in desperate need to match the tasks with the transient power.

Previous scheduling methods for self-powered sensor nodes can be divided into two categories based on whether a single task can be separated: inter-task scheduling and intra-task scheduling. Inter-task scheduling aims to schedule tasks in a coarse-grained task-level manner. Lots of inter-task scheduling methods, such as adaptive duty cycle [2], task migration [3] and dynamic voltage and frequency scaling [4], are devoted to improving performance. Nonvolatile processors (NVP) can back up system states more efficiently and resume execution after power failure [5], which promising candidates to support intra-task scheduling. Recently, Zhang *et.al.* proposed intra-task scheduling with energy migration [6, 7] for efficient operations on NVP based sensor nodes [8].

However, previous intra-task scheduling omits the different power switching overhead among tasks on nonvolatile sensor nodes. Although computing tasks can be suspended efficiently by NVP and switching overhead is quite low, tasks involving sensing, memory access and transmitting operations can not be backed up during power failures. Rather large switching overhead is observed in such tasks. Failing to distinguish those task features will cause inefficient task scheduling problems. Previous works such as Ref. [9] only take the preemption overhead caused by saving memory spaces or other internal factors into consideration, while our strategy focuses on the performance overhead induced by the power switching process.

This paper proposes a power switching overhead aware scheduling method for NVP based energy harvesting sensor nodes. The major challenges lie in allocating different types of tasks on proper time slots. The main contributions of this work are as follows:

- Analyzing power switching overhead for the operations on energy harvesting sensor nodes (Section 2).
- Formulating the performance-aware task scheduling problem

DAC '16, June 05-09, 2016, Austin, TX, USA
© 2016 ACM. ISBN 978-1-4503-4236-0/16/06…$15.00
DOI: http://dx.doi.org/10.1145/2897937.2898059

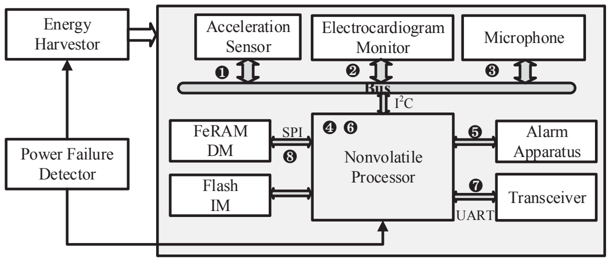

Figure 1: A nonvolatile energy harvesting sensor node consisting of an energy harvesting module, a nonvolatile processor, nonvolatile storage and peripheral sensors. The numbers represent tasks executed on the platform.

considering power switching overhead to a mixed integer linear programming (MILP) problem. Both timing and power constraints are considered. The objective is to minimize the makespan of all tasks (Section 3).

- Proposing an heuristic scheduling strategy with polynomial complexity to obtain an efficient solution. A task splitting strategy (**TS**) is also presented to further improve the performance and mitigate the impact from the fluctuation of the power input (Section 4).

2. MOTIVATION

This section first describes the power switching overhead in different tasks on NVP based energy harvesting sensor nodes, followed by a motivation example for task scheduling considering power switching overhead.

2.1 Power switching overhead challenge

Fig. 1 shows a storage-less energy harvesting platform [10, 11] powered by ambient energy sources. Due to variations of energy sources and lacking of energy buffers, the collected energy is just-in-time for usage and the node oscillates between ON and OFF states. Frequent power failures happen and power switching overhead becomes relevant. Power switching overhead is defined as extra operating time/energy induced by rollbacks, backups and restorations caused by power interruptions. According to whether a task can be resumed after a power interruption, tasks are classified into two categories: tasks which can (cannot) be resumed are with low (high) power switching overhead, denoted as Ls (Hs) tasks.

For example, computing tasks running on NVP, which can be suspended and resumed within less than $10\mu s$ when power failures happen, are classified into the Ls type. While tasks, involving data transmission among sensors, off-chip storage, wireless transceivers and processors, belong to Hs tasks, because the relevant peripheral devices can not store states during power interruptions and lead to expensive rollbacks and initializations. Fig. 2 shows the typical power switching overhead in a wild animal monitoring application [12], which shows 6-7 orders of magnitude of differences in switching overhead.

2.2 Motivation example

The following example demonstrates the necessity of taking power switching overhead into consideration. The objective of scheduling on energy harvesting platform is to minimize the operating length (makespan) of multiple tasks, which are modeled as a directed acyclic graph (DAG) $G = (V, E)$. Each vertex $v \in V$ corresponds to a task, characterized by execution time t and the power consumption p. Each edge $e \in E$ represents a precedence constraint, where a task cannot be executed until all preceding tasks are finished.

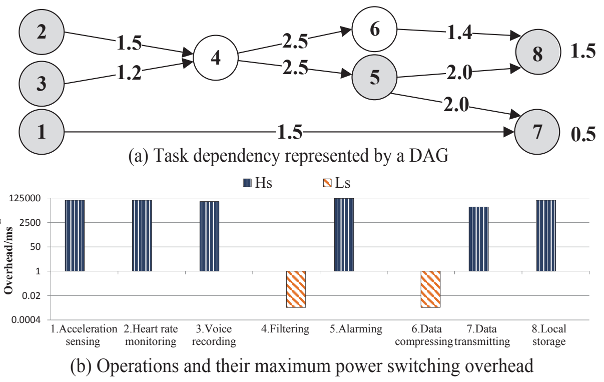

Figure 2: (a) Task graph of a wild animal monitoring application. The weights of edges represent the execution time (min). (b) Operation names and their task types (Ls/Hs) together with their maximum power switching overhead.

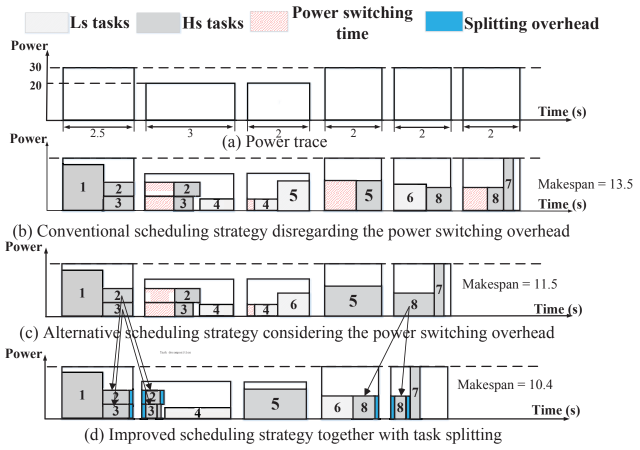

Figure 3: Power pulses (a) and the comparison between the conventional scheduling strategy (b) and the alternative scheduling strategy (c) considering power switching overhead. (d) Scheduling results improved by task splitting.

Assuming power switching overhead can be omitted on NVP based sensor nodes, previous schedulers select tasks with longer remaining time and lead to Fig. 3 (b). However, they introduce rather large power switching overhead when both Hs and Ls tasks are executed. The alternative scheduling policy takes power switching overhead into consideration and minimizes switching overhead by allocating appropriate time slots to tasks, as shown in Fig. 3 (c). The overhead is shown by yellow rectangles. The total makespan of Fig. 3 (c) is smaller than that of Fig. 3 (b) due to the reduction of the power switching overhead.

Since rather large power switching overhead comes from inappropriate peripheral hardware support, we can split Hs tasks into multiple pieces by modifying the workflow of Hs tasks, hence each of which are recovered from the nearest splitting point instead of the very beginning. Though task splitting will introduce some performance overhead, it provides better opportunities to match task execution with power profiles. Fig. 3 (d) shows a better scheduling result which can be achieved by task splitting, where blue regions denote checkpoint overhead. The above example suggests that the challenges of scheduling lie in how to match tasks with power profiles considering the power switching overhead constraint. The scheduling policy should balance task dependencies, degrees of parallelism, splitting granularities and task types.

3. PROBLEM FORMULATION

This section first defines variables and system parameters of architecture and DAG instance proposed in Section 2. After that, the scheduling objective and optimization constraints of a mixed-integer linear programming formulation (MILP) are presented.

3.1 Variables

The system parameters and variables are defined in Table 1. $x_{i,k}$ indicates the proportion of execution time that task i is spent in power pulse k. The harvested power is modeled by intermittent power pulses, where a power failure happens when the voltage of power supply drops below a certain threshold.

Table 1: Parameters and variables of the system model

	Symbol	Description
Task	$\mathcal{V}$	Task set $\mathcal{V} = \{\tau_1, \tau_2, ..., \tau_N\}$
	$d_{i,j}$	Binary parameter indicating if task j depends on the result of task i
	t_i	Execution time of task τ_i
	p_i	Power consumption of task τ_i
	r_i	Binary variable indicating whether task τ_i is a task with high power switching overhead or not
	R	Recovery time of task τ_i if τ_i is a task with low power switching overhead
Power	$\mathcal{P}$	Power pulse set, $\mathcal{P} = \{\rho_1, \rho_2, ...\rho_m, ...\}$
	T_k	Time duration of power pulse ρ_k
	P_k	Amplitude of power pulse ρ_k
	Φ	Subset of the power pulse set $\mathcal{P}$
Variable	X_i	$X_i = [x_{i,1}, x_{i,2}, ...x_{i,k}, ...x_{i,m}]$. $x_{i,m}$ is a fractional variable between [0,1] which indicates the proportion of execution time that task i is on power pulse m and M is the number of used power pulses.

3.2 Scheduling objective

The main objective is to minimize the total execution time of all tasks, by mapping each task into one or more power pulses. Therefore, it is equal to find a minimum number of power pulses to execute all tasks in the DAG. The scheduling objective is as follows.

$$\begin{aligned} &obj : \min M = |\Phi|, \Phi \subset \mathcal{P} \\ &s.t : \mathcal{V} \text{ can be executed within } \Phi, \end{aligned} \tag{1}$$

where M is the number of used power pulses and $|\Phi|$ represents the number of elements in Φ. A power pulse ρ_i can only be added to Φ in order due to the temporal continuity, which means that $\rho_{k-1} \in \Phi$ if $\rho_k \in \Phi$.

We further convert the optimization problem into a MILP problem. Assuming that actual execution of different tasks can overlap as long as the constraints are satisfied, a function $g_e(X)$ is defined to get the end time of a task.

***Definition 1*: End function**: End function $g_e(X_i)$ takes the value of a vector with positive elements and returns the index of the last non-zero element, as shown in Equation 2.

$$g_e(X_i) = \max\{k|\sum_{j=k}^{M} x_{i,j} > 0, k \in [1, M], k \in Z\}, \forall i \in [1, N] \tag{2}$$

The end function $g_e(X_i)$ is used to construct the optimization target and necessary constraints. However, $g_e(X_i)$ is defined non-linearly related to variable $x_{i,j}$. In the following analysis, the complexity is reduced to be linear by introducing intermediate variables. Two accumulation vectors A and A' are introduced and defined as follows.

$$A = [\alpha_{i,k}], \alpha_{i,k} = \sum_{j=1}^{k} x_{i,j}, \; A' = [\alpha'_{i,k}], \alpha'_{i,k} = \sum_{j=k}^{M} x_{i,j} \tag{3}$$

Based on A and A', it is easy to locate the first and the last non-zero element in X_i. A simple example gives a clear perspective. Given $X_i = [0, 0, 0.3, 0.7, 0, 0.2, 0]$, we have $A_i = [0, 0, 0.3, 1, 1, 1.2, 1.2]$ and $A'_i = [1.2, 1.2, 1.2, 0.9, 0.2, 0.2, 0]$.

In order to represent $g_e(X_i)$ and necessary constraints, we introduce three binary variables $\beta_{i,k}$, $\beta'_{i,k}$ and $\gamma_{i,k}$ to denote whether $\alpha_{i,k}$, $\alpha'_{i,k}$ and $x_{i,k}$ is positive or zero, respectively ($\beta_{i,k} = 1$ if $\alpha_{i,k} > 0$ and $\beta_{i,k} = 0$ otherwise. $\beta'_{i,k}$ and $\gamma_{i,k}$ can be constructed in the similar way). The following constraints are developed.

$$N_1\beta_{i,k} \geq \alpha_{i,k}, \; N_2\beta'_{i,k} \geq \alpha'_{i,k}, \; N_3\gamma_{i,k} \geq x_{i,k} \tag{4}$$

where $N_1, \; N_2, \; N_3$ are sufficiently large numbers. Hence we have

$$\beta'_{i,k} = \begin{cases} 1, & \alpha'_{i,k} > 0 \\ 0 \text{ or } 1, & \alpha'_{i,k} = 0 \end{cases} \quad \gamma_{i,k} = \begin{cases} 1, & x_{i,k} > 0 \\ 0 \text{ or } 1, & x_{i,k} = 0 \end{cases} \tag{5}$$

where $\beta_{i,k}$ follows the same expression.

Recall that $\beta_{i,k}$, $\beta'_{i,k}$ and $\gamma_{i,k}$ should be used to denote whether $\alpha_{i,k}$, $\alpha'_{i,k}$ and $x_{i,k}$ is positive or zero. Therefore, based on Equation 5, $\beta_{i,k}$, $\beta'_{i,k}$ and $\gamma_{i,k}$ should be as small as possible to guarantee that they are equal to zero when $\alpha_{i,k} = 0$, $\alpha'_{i,k} = 0$ and $x_{i,k} = 0$. To do so, in the following formulation, several constraints are presented to provide the minimization limitation to $\beta_{i,k}$, $\beta'_{i,k}$ and $\gamma_{i,k}$. We will explain later when introducing those constraints. In this way, the end function can be represented linearly as follows.

$$g_e(X_i) = \sum_{k=1}^{M} \beta'_{i,k}, \; \forall i \in [1, N] \tag{6}$$

Equation 1 shows that the objective of the proposed MILP is to minimize the number of used power pulses. In other words, the objective is to minimize the total completion time of all tasks.

$$\text{obj. } \min\{\max\{g_e(X_i)|i \in [1, N]\}\} \tag{7}$$

To reduce the objective function to a linear form, we introduce a variable Y.

$$\text{Let } Y \geq g_e(X_i), \forall i \in [1, N] \tag{8}$$

Given the linear form of $g_e(X_i)$ as shown in Equation 6, the objective can be re-written as:

$$\text{obj. } \min Y \tag{9}$$

Based on Equation 6 and 7, $\beta'_{i,k}$ is guaranteed to be as small as possible. The scheduling objective has to meet some constraints caused by the nature of tasks and hardware, which will be illustrated in the following subsection.

3.3 Constraints

Power switching, timing, energy and power peak constraints should be considered to guarantee the successful backup and correct execution sequence depending on the task dependency.

Power switching constraint: A task with high power switching overhead will be executed from the very beginning if it met power interruption before. Hence, *Hs* tasks should be executed without power interruption, which means that they should be mapped to only one power pulse, that is

$$r_i \sum_{k=1}^{M} \gamma_{i,k} \leq 1, \; \forall i \in [1, N] \tag{10}$$

This constraint only works for tasks with high power switching overhead, because Equation 10 is absolutely true for Ls tasks because $r_i = 0$. $\gamma_{i,k}$ is constrained to be as small as possible. Hence, $\gamma_{i,k}$ is constrained to be 0 when $x_{i,k} = 0$.

Timing constraint: The task dependency is constrained in Equation 11. $M - \sum_{k=1}^{M} \beta_{j,k}$ represents the start executing time of task τ_j. When the binary parameter $d_{ij} = 1$, the start time of τ_j cannot be less than $g_e(X_i)$, meaning that task τ_j can only be executed after task τ_i is completed. This equation provides $\beta_{j,k}$ a large potential to be as small as possible (be 0 when $\alpha_{i,k} = 0$) so as to satisfy the constraint.

$$d_{i,j} \cdot (M - \sum_{k=1}^{M} \beta_{j,k}) \geqslant g_e(X_i),\ \forall i, j \in [1, N] \tag{11}$$

$$\sum_{k=1}^{M} x_{i,k} T_k - (1 - r_i) R(\sum_{k=1}^{M} \gamma_{i,k} - 1) = t_i,\ \forall i \in [1, N] \tag{12}$$

Equation 12 means that the total execution time of task i is equal to the sum of the execution time of each power pulse that is spent on task i. If τ_i is a task with low power switching overhead, R times the number of interruptions should be added to the total execution time.

Energy constraint: This constraint is used to ensure that the total amount of energy consumed by all executing tasks should be lower than supplied energy of each power pulse, as shown by Equation 13.

$$P_k T_k \geq \sum_{i=1}^{N} x_{i,k} \cdot T_k \cdot p_i,\ \forall i \in [1, N], k \in [1, M] \tag{13}$$

Peak power constraint: This constraint is used to guarantee that each task can be feasibly executed under the assigned power pulse.

$$P_k \geq p_i \cdot \gamma_{i,k}, \forall\ i \in [1, N], k \in [1, M] \tag{14}$$

3.4 Matrix form formulation

Furthermore, in order to implement the MILP in Lingo, we formulate the overall scheduling objective in the matrix form. The overall design variables can be represented by the following matrix.

$$\boldsymbol{X}_{(M,N)} = [X_1^T, X_2^T, ..., X_N^T] \tag{15}$$

The time duration and power amplitude of the power pulse set are denoted by the following vectors.

$$\boldsymbol{T} = [T_1, T_2, ..., T_M]^T,\ \boldsymbol{P} = [P_1, P_2, ..., P_M]^T \tag{16}$$

The execution time, power consumption and the task types of the task set $\mathcal{V}$ can be denoted by the following vectors.

$$\begin{gathered}\boldsymbol{t} = [t_1, t_2, ..., t_N]^T,\ \boldsymbol{p} = [p_1, p_2, ..., p_N]^T \\ \boldsymbol{r} = [r_1, r_2, ..., r_N]^T,\ \boldsymbol{g_e} = [g_e(X_1), g_e(X_2), ..., g_e(X_N)]^T\end{gathered} \tag{17}$$

Let $B_j = \sum_{k=1}^{M} \beta_{j,k}$, $j \in [1, N]$. Define the diagonal matrix

$$\boldsymbol{B}_{(N,N)} = diag(M - B_j) \tag{18}$$

The matrix of $\{\gamma_{i,k}\}$ is expressed as,

$$\boldsymbol{\gamma}_{(N,M)} = \begin{bmatrix} \gamma_{11} & \dots & \gamma_{1,M} \\ \vdots & \ddots & \vdots \\ \gamma_{N1} & \dots & \gamma_{N,M} \end{bmatrix} \tag{19}$$

The dependency matrix $\boldsymbol{d}_{(N,N)}$ is defined as follows.

$$\boldsymbol{d}_{(N,N)} = \begin{bmatrix} d_{11} & \dots & d_{1,N} \\ \vdots & \ddots & \vdots \\ d_{N1} & \dots & d_{N,N} \end{bmatrix} \tag{20}$$

Let $\boldsymbol{G_e} = [\underbrace{\boldsymbol{g_e}, \boldsymbol{g_e}, ..., \boldsymbol{g_e}}_{N}]$, the scheduling problem is described by the following MILP formulation. All constraints defined from Equation 10 to Equation 14 are represented by the following expressions in matrix form.

$$\begin{aligned} &\text{obj.}: \ \min\ Y \\ &s.t.\ \boldsymbol{r}.*(\boldsymbol{\gamma}\mathbf{1}_{(M,1)}) \leq \mathbf{1}_{(N,1)},\ \boldsymbol{dB} \geq \boldsymbol{G_e} \\ &\quad \boldsymbol{X}^T\boldsymbol{T} - R(\mathbf{1}_{(N,1)} - \boldsymbol{r}).*(\boldsymbol{\gamma}\mathbf{1}_{(M,1)} - \mathbf{1}_{(N,1)}) = \boldsymbol{t} \\ &\quad \boldsymbol{Xp} \leq \boldsymbol{P},\ \boldsymbol{\gamma}.*(\boldsymbol{p}\mathbf{1}_{(1,M)}) \leq \mathbf{1}_{(N,1)}\boldsymbol{P}^T,\ \boldsymbol{g_e} \leq Y\mathbf{1}_{(1,N)}, \end{aligned} \tag{21}$$

where $.*$ represents to multiply the corresponding elements of two matrixes.

4. HEURISTIC SCHEDULING WITH TASK SPLITTING

The MILP formulation in Section 2 is known to be computationally intensive and not scalable. Since task splitting can further reduce the makespan by decomposing a Hs task into small tasks and make lots of new task combination, there will be a much larger design space to explore, which is unacceptable for a MILP solver. Therefore, we propose an heuristic algorithm to solve the scheduling problem.

4.1 Task splitting strategy

In order to further reduce the makespan, the task splitting algorithm decomposes a critical task with time length t_i into several small pieces. Although we focus on the uniform splitting hereinafter, it is easy to extend the splitting strategy to nonuniform partitions. However, task splitting will induce some time/energy overhead caused by restarting programs. We denote the overhead added to each sub task after a single split as S, which is obtained by referring to the actual initialization time of the Hs tasks. For instance, $S = 1691\mu s$ for a temperature sensing task because it requires $1691\mu s$ to set up the data sensing operation. Therefore, the time length of each sub task is $t_i' = \frac{t_i}{k} + S$ after k times of uniform splits. Intuitively, larger splitting overhead is induced if k is too large. Meanwhile, it cannot provide enough performance improvement if k is too short. Therefore, there will be an optimal splitting granularity for each task. Three factors should be taken into consideration for the selection of the splitting granularity: the task length, the average power pulse length and the power fluctuation ratio, which will be discussed in the experiments.

4.2 Algorithm framework

After task splitting, we need a more efficient heuristic algorithm to explore different task combinations. The major idea is to transform the MILP formulation into several smaller problems with less variables. We propose a greedy approach to fill each power pulse considering the power switching overhead. The definition of the power utilization in each power pulse is given as follows.

Definition 2: **Power utilization**: Given a power pulse ρ_k with length T_k, height P_k and candidate tasks set $\mathcal{V}_{can} = \{\tau_{i_1}, \tau_{i_2}, ..., \tau_{i_n}\}$, $\mathcal{V}_{can} \in \mathcal{V}$, the power utilization, denoted as U, is defined as the ratio of the energy consumed by the efficient executed tasks to the amount of total energy provided by the power pulse set. It can be expressed as Equation 22.

$$U = \frac{\sum_n x_{i_n,k} T_k p_k}{P_k T_k},\ i_n \in [1, N], k \in [1, M] \tag{22}$$

Since tasks with high power switching overhead lead to more performance degradation if power failures happen, we assign high priorities to them if the present power pulse is long enough to execute Hs tasks. Based on this observation, the heuristic algorithm is shown in algorithm 1.

The flow of our proposed heuristic scheduling policy is depicted in Algorithm 1. Given task combinations generated by task splitting (Line 4), the procedure in the loop selects proper tasks for each power pulse set Φ until all tasks are done. Line 8 selects candidate tasks through Algorithm 2. The principle of the task selection algorithm is that a Ls (Hs) task can only be filled into the current power pulse if the earliest *start* (*end*) time is shorter than the length of the current power pulse. It's due to the fact that Hs tasks have to be finished in the current power pulse, while Ls tasks can be interrupted. Line 9 to line 13 assign high priorities to tasks with high power switching overhead if they can be finished in the current power pulse. Line 14 selects tasks by solving the local MILP algorithm. Due to the reduction of the number of optimization variables, the computation complexity O' is exponentially decreased compared to that of the original MILP. The total computation complexity mO' is smaller than MILP's computation complexity O, where m is the amount of the task combinations.

Algorithm 1 Heuristic task scheduling algorithm with the task splitting algorithm $K_{min} = F(\mathcal{V}, \Phi)$

```
Input:
 1: A task set $\mathcal{V}$ presented by a DAG $G(V,E)$;
 2: A power pulse set $\Phi = \{\rho_1, \rho_2, ..., \rho_m ...\}$.
Output: The minimum number of the power pulses $K$ to
    complete all tasks in the task set $\mathcal{V}$.
 3: Initialize $K_{min} \leftarrow 0$, $m \leftarrow 0$. Executed task set $\mathcal{E} \leftarrow \varnothing$
 4: for all Task splitting strategies do
 5:     $m \leftarrow 0$;
 6:     while $\mathcal{V} \neq \varnothing$ do
 7:         $m = m + 1$
 8:         Select $\mathcal{V}_{can} = Cand(\mathcal{V}, \rho_m)$
 9:         for all $Hs$ tasks $\tau_j \in \mathcal{V}_{can}$ do
10:             if $\rho_m \rightarrow T \geq t_j$ and $\rho_{m+1} \rightarrow T \leq t_j$ then
11:                 Add $\tau_j$ to $\mathcal{E}$;
12:             end if
13:         end for
14:         Select $\mathcal{E} \in \mathcal{V}_{can}$ to maximize $U$ by solving local
    MILP;
15:         Update $G$ and $\mathcal{V}$ by removing tasks in $\mathcal{E}$;
16:     end while
17:     $K = m$;
18:     if $K < K_{min}$ then
19:         $K_{min} = K$;
20:     end if
21: end for
22: Return $K_{min}$.
```

Algorithm 2 Candidate tasks selection $\mathcal{V}_{can} = Cand(\mathcal{V}, \rho_m)$

```
Input:
 1: Current task set $\mathcal{V}$ represented by a directed acyclic graph
    $G(V,E)$.
 2: Current power pulse $\rho$ with length $T$.
Output: Candidate tasks $\mathcal{V}_{can}$ from the task set $\mathcal{V}$.
 3: Calculate the earliest finish time $ef$ for all $Hs$ tasks and the
    earliest start time $es$ for all $Ls$ tasks;
 4: for all Tasks $\tau \in \mathcal{V}$ do
 5:     if ($\tau \rightarrow Hs$ and $\tau \rightarrow ef < \rho_m \rightarrow T$) or ($\tau \rightarrow Ls$ and
    $\tau \rightarrow es < \rho_m \rightarrow T$) then
 6:         Add $\tau$ to the candidate set $\mathcal{V}_{can}$;
 7:     end if
 8: end for
 9: Return $\mathcal{V}_{can}$.
```

5. EXPERIMENT EVALUATION

In this section, we provide comprehensive evaluations to demonstrate the effectiveness of the proposed scheduling algorithm and analyze some influence factors.

5.1 Experimental setup

Four real applications are used as the testbenches: Structure health monitoring (SHM) [13], ECG monitoring (ECG) [14], Wild animal monitoring (WAM) [12] and multi-media application (Media) [15]. The task splitting overhead is calculated by measuring on real chips. Moreover, TGFF are used for the pseudo-random task graph generation. We develop a simulator based on Lingo to evaluate the scheduling algorithm.

5.2 Heuristic scheduling algorithm validation

The following three scheduling strategies are compared with each other: Scheduling results of the intra-task strategy [7]; Optimal MILP results solved by Lingo; Proposed heuristic scheduling algorithm with task splitting (Heuristic + TS). The experiment results are listed in Table 2. HM is the harmonic mean of the results, T_e and T_s are the execution time (makespan) and the power switching time.

Table 2: Comparison among the conventional scheduling, MILP and the heuristic algorithm.

Benchmark	Intra [7]		MILP		Heuristic + TS	
	$T_e(s)$	$T_s(s)$	$T_e(s)$	$T_s(\mu s)$	$T_e(s)$	$T_s(\mu s)$
SHM [13]	7.1	0.1	7	3	7	3
ECG [14]	20.3	12.4	9.8	6	8.1	9
WAM [12]	21.7	10.1	14.6	3	12.1	6
Media [15]	22.6	11.0	15.5	3	12.0	9
bus	11.9	1.8	12	9	10.9	12
creds1	10.6	0.9	8.7	15	8.7	15
packed	11.9	3.0	10.1	9	8.5	12
packets	15.8	2.9	13.1	3	13.1	3
simple	8.7	0.6	9.5	0	8.1	9
HM	12.1	0.6	10.4	4.6	9.3	6.4

Table 2 indicates that the MILP strategy can reduce the execution time by 14% compared to the work in Ref. [7] mainly because the power switching overhead is reduced dramatically (about 5 orders of magnitude). Moreover, the proposed heuristic scheduling strategy can estimate the ideal result with an error rate of 9.7%. By adopting the task splitting strategy, the execution time is further reduced by 10.6%. Furthermore, the MILP requires several minutes to get results or even fails in large scale problems while the heuristic algorithm can be completed within several seconds, which validates its time efficiency.

5.3 Task types analysis

The power switching overhead is the most important factor. Fig. 4 validates that the amount of Hs tasks and their locations have influence on the system performance. The results are derived by using random tasks. R_{Hs} is the ratio of high recovery overhead tasks. With the increasing of R_{Hs}, the average execution time tends to increase due to the increasing of power switching overhead. We further notice that the execution time of root intense (RI) and uniform distribution (UD) are short. It is because that the proposed scheduling strategy gives higher priority to Hs tasks. If Hs task nodes gather near the neighbourhood of the root node, they can be adequately considered by the strategy and the successor nodes are unlocked easily. The uniform location is also performance friendly because it gives more optionality to the scheduler. However, the

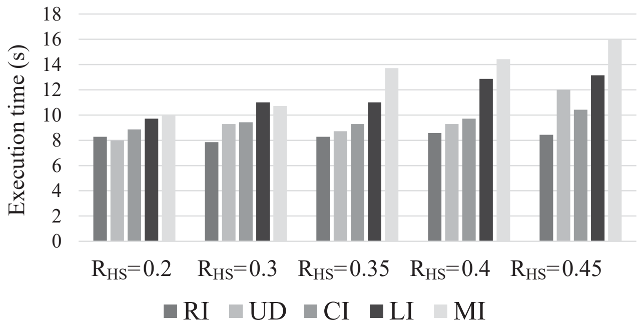

Figure 4: Relationship between the execution time and different task location patterns for $R_{Hs} = 0.2, 0.3, 0.35, 0.4, 0.45$. RI (Root node intense); UD (Uniform distribution); CI (Critical node intense); LI (Leaf node intense); MI (Middle nodes intense). R_{Hs} is the ratio of high power switching overhead tasks.

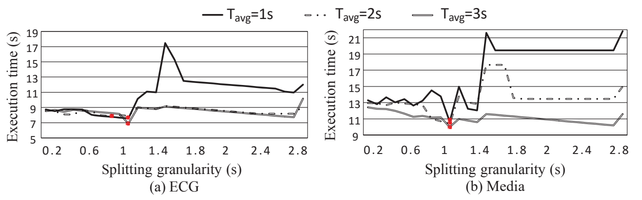

Figure 5: Execution time (s) vs. Splitting granularity for different average power length (T_{avg}).

performance is bad if the task location pattern is leaf node intensive (LI) or middle nodes intensive (MI) because these two patterns are more likely to block the execution progress. The above results inspire the system developers that the Hs and Ls tasks should be carefully arranged for better performance.

5.4 Power effect analysis

Fig. 5 shows how the task splitting granularity and the power variance influence the overall performance. The splitting granularity means the length of each sub task after splitting. The execution time gradually drops with the decreasing of splitting granularity as the splitted tasks are inserted into small slots where tasks are not able to be pushed in before. However, as the oscillating curve shows, more splitting operations do not ensure better performance, for the changed structure may lead to longer critical paths and higher splitting overhead. We can see that there exists an optimal task splitting solution that can obviously improve the overall performance, as marked by red points. The execution time with shorter power length is usually higher, because shorter power length causes more power switching overhead. Moreover, trends of different curves are similar. It indicates that our strategy has the anti-disturbance ability, which means that the proposed scheduling policy can still achieve pleasurable results if the actual power input slightly deviates from the original one.

6. CONCLUSION

In this paper, we propose a performance-aware task scheduling strategy for energy harvesting nonvolatile processors considering the power switching overhead. The scheduling problem is first formulated into a MILP problem. Furthermore, an heuristic scheduling strategy with task splitting is proposed. The experimental results show that the proposed scheduling strategy can improve the performance by 14% compared with the state-of-the-art scheduling strategy, and the task splitting strategy can further improve the performance by 10.6%. In addition, the results manifest the conclusion that power conditions and task types have great influence on the system performance. With the development of energy harvesting systems, the increasing amount of data to process and growing demand for communication will further aggravate the influence of power switching overhead on the performance of systems and the proposed strategy will be quite promising.

7. ACKNOWLEDGMENTS

This work was supported in part by NSF CNS-1464429, NSFC Grant #61271269 and the Research Grants Council of the Hong Kong SAR, China (Project No. CityU 11214115). Corresponding author: Yongpan Liu, `ypliu@tsinghua.edu.cn`.

8. REFERENCES

[1] Kaisheng Ma and et al. Architecture exploration for ambient energy harvesting nonvolatile processors. In *HPCA'15*, pages 526–537. IEEE, 2015.

[2] Yu Gu and et al. Spatiotemporal delay control for low-duty-cycle sensor networks. In *RTSS'09*, pages 127–137. IEEE, 2009.

[3] Yang Ge and et al. A multi-agent framework for thermal aware task migration in many-core systems. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 20(10):1758–1771, 2012.

[4] Shaobo Liu and et al. Harvesting-aware power management for real-time systems with renewable energy. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 20(8):1473–1486, 2012.

[5] Y Liu and et al. 4.7 a 65nm reram-enabled nonvolatile processor with 6x reduction in restore time and 4x higher clock frequency using adaptive data retention and self-write-termination nonvolatile logic. In *ISSCC'16*, pages 84–86. IEEE, 2016.

[6] Daming Zhang and et al. Intra-task scheduling for storage-less and converter-less solar-powered nonvolatile sensor nodes. In *ICCD'14*, pages 348–354. IEEE, 2014.

[7] Daming Zhang and et al. Deadline-aware task scheduling for solar-powered nonvolatile sensor nodes with global energy migration. In *DAC'15*. IEEE, 2015.

[8] Cong Wang and et al. Storage-less and converter-less maximum power point tracking of photovoltaic cells for a nonvolatile microprocessor. In *ASP-DAC'14*, pages 379–384. IEEE, 2014.

[9] Gang Yao and et al. Bounding the maximum length of non-preemptive regions under fixed priority scheduling. In *RTCSA'09*, pages 351–360. IEEE, 2009.

[10] Yongpan Liu and et al. Ambient energy harvesting nonvolatile processors: from circuit to system. In *DAC'15*, page 150. ACM, 2015.

[11] Mengying Zhao and et al. Software assisted non-volatile register reduction for energy harvesting based cyber-physical system. In *DATE'15*, pages 567–572. EDA Consortium, 2015.

[12] Wei Liu and et al. Design and implementation of a hybrid sensor network for milu deer monitoring. In *ICACT'12*, pages 52–56. IEEE, 2012.

[13] Wei Liu and et al. Application specific sensor node architecture optimizationąłexperiences from field deployments. In *ASP-DAC'12*, pages 389–394. IEEE, 2012.

[14] 48 half-hour excerpts of two-channel ambulatory ecg recordings. http://physionet.org/physiobank/database/mitdb/, 2013.

[15] Yujin Zhang. Image engineering–image processing and analyzing, 1999.