IEEE TRANSACTIONS ON
EMERGING TOPICS
IN COMPUTING

Received 20 March 2013; revised 6 July 2013; accepted 8 July 2013. Date of publication 18 July 2013;
date of current version 21 January 2014.

Digital Object Identifier 10.1109/TETC.2013.2274042

Scheduling Co-Design for Reliability and Energy in Cyber-Physical Systems

MAN LIN (Member, IEEE)[1], YONGWEN PAN[1], LAURENCE T. YANG (Member, IEEE)[1], MINYI GUO (Member, IEEE)[2], AND NENGGAN ZHENG[3]

[1]St. Francis Xavier University, Antigonish, NS B2G 2W5, Canada
[2]Shanghai Jiaotong University, Shanghai 200030, China
[3]Zhejiang University, Zhejiang 321000, China

CORRESPONDING AUTHOR: M. LIN (mlin@stfx.ca)

This work was supported by the National Science Engineering Research Council, Canada. The work of N. Zheng was supported by the National Natural Science Foundation of China under Grant 61003150.

ABSTRACT Energy aware scheduling and reliability are both very critical for real-time cyber-physical system design. However, it has been shown that the transient faults of a system will increase when the processor runs at reduced speed to save energy consumption. In this paper, we study total energy and reliability scheduling co-design problem for real-time cyber-physical systems. Total energy refers the sum of static and dynamic energy. Our goal is to minimize total energy while guaranteeing reliability constraints. We approach the problem from two directions based on the two different ways of guaranteeing the reliability of the tasks. The first approach aims at guaranteeing reliability at least as high as that of without speed scaling by reserving recovery job for each scaled down task. Heuristics have been used to guide the speed scaling and shutdown techniques that are used to lower total energy consumption while guaranteeing the reliability. The second way to guarantee the reliability of the tasks is to satisfy a known minimum reliability constraint for the tasks. The minimum reliable speed guarantees the reliability level of tasks, and is used as a constraint in the energy minimization problem. Both static and dynamic co-design methods are explored. Experimental results show that our methods are effective.

INDEX TERMS Real-time systems, dynamic energy, static energy, leakage control, reliability.

I. INTRODUCTION

With the advance of engineering and networking technology, many devices are now connected into a network or with the Internet. Embedded systems are emerging into Cyber-Physical Systems (CPS) [7], [19], [28]. A cyber-physical system is strongly coupled with physical systems, functioning as sensor based systems, on-line monitoring systems, or on-line controlling systems. A cyber-physical system can be found in diverse areas such as automotive, aerospace, health care, transportation, building and process control, entertainment etc.

There are challenges related to various aspects on designing Cyber-Physical Systems [11], [14], [18], [31]. In this paper, we study scheduling methods in CPS systems that take both energy consumption and system reliability into account. Unlike desktop systems, many Cyber-Physical Systems operate with limited energy supply. Therefore, energy efficient computing and communication, and energy aware resource management are of great importance to CPS [5], [23], [29], [34], [42]. Reliability is also of extreme importance in many of these CPS such as aerospace systems, transportation systems and medical monitoring systems [4], [41].

The energy consumption of the CPS in this paper refers to the sum of static and dynamic energy of the CMOS based processing unit and/or communication links. Dynamic power consumption is caused by switching activities of transistors and has been previously considered as the dominant part of system energy consumption. Static energy, also called leakage energy, is consumed whenever the processor is on. Static (Leakage) energy consumption is comparable to the dynamic energy consumption in modern chips, and thus cannot be ignored anymore. According to [30], with processor size shrinking, supply voltages and threshold voltages of CMOS circuits shrink too, which results in exponential increases in

sub-threshold leakage current. At and under 70 *nm* processor technology, the static power consumption occupies more than 50% of total energy. In order to obtain minimal total energy consumption, static power consumption must be considered together with dynamic power consumption. Dynamic Voltage Scaling (DVS) method can reduce dynamic energy consumption by scaling down the voltage level. Leakage control method can reduce static energy consumption during idle period. DVS and Leakage control method can be combined to reduce total energy consumption.

Reliability of a job is defined as the probability of the job being correctly executed before its deadline [44]. Many CPS used in critical areas such as defence, aerospace, and nuclear power generation require high-level reliability. For a CPS, fault occurs unpredictably during run time due to various reasons, such as hardware failures, electromagnetic interferences as well as the effects of cosmic ray radiations [44]. If the faults are not dealt with in time, it may lead to a disaster. Faults usually can be classified as transient faults and permanent faults, and the transient faults occur much more frequently than permanent faults [2]. We only concentrate on transient faults in this work. It has been reported that when CPU runs with a reduced speed, the transient faults of the system will increase and thus the reliability of the system is reduced [45].

In this paper, we will focus on the total energy and reliability co-design problem considering a set of periodic tasks scheduled with Earliest Deadline First (EDF) policy. The tasks here can be sensing, computing or communicating tasks. As we consider highly reliable CPS, the worst execution times of the tasks are assumed known in a priori. Two approaches are used to manage reliability and total energy scheduling co-design.

1) Low total energy consumption with guaranteed reliability at least as high as that without voltage scaling:
   The first approach aims at guaranteeing reliability at least as high as that without voltage scaling while at the same time reducing total energy. It is achieved by heuristically applying leakage control methods to previous reliability-aware DVS methods [39], [45] such that total energy can be reduced.
2) Minimum reliability level as a constraint to the energy minimization problem:
   The second approach aims at minimizing energy consumption with reliability as a constraint. The minimum reliable speed guarantees the reliability level of a task, and can be derived by looking up the reliability curve. The minimum reliable speed constraint will be used as a constraint in the energy minimization problem, solved with convex optimization techniques.

The organization of the rest of the paper is as follows. Section 2 presents related works. Section 3 introduces the models we use in this paper. Sections 4 and 5 present detailed algorithms about the two aspects. Then Sections 6 and 7 discuss our simulation results and conclusions.

II. RELATED WORKS

Energy management of CPS has gained a lot of attention. There are two commonly used techniques to lower the energy consumption in CPS: slowdown and shutdown. Slowdown technique is used in Dynamic Voltage Scaling (DVS) to reduce dynamic energy consumption in processing units or communication links. Shutdown technique can put some computing nodes, sensing nodes, communication links or other devices to sleep to save static energy. Next, we describe the related works on dynamic and static energy management. We will also describe previous works on reliability aware DVS scheduling. Then we will discuss the framework of our works on reliability aware scheduling that integrates both slowdown and shutdown technique in order to save energy and maintains reliability of the system.

A. DYNAMIC VOLTAGE SCALING

The Dynamic Voltage Scheduling (DVS) strategy is an effective strategy to reduce dynamic power consumption. Most processors in current computer systems are composed of CMOS circuits. DVS can be used to reduce the energy consumption of processors or interconnection networks [27], [32]. When the workloads is light, a processor or communication link does not need to execute with maximum speed. Instead, its execution frequency can be slowdown. The time to finish the job is extended accordingly. Since the energy dissipated per cycle with CMOS circuit scales quadratically to the supply voltage, DVS can provide a large energy saving by frequency and voltage scaling. A lot of DVS based real-time scheduling algorithms without considering static power have been proposed to reduce energy [16], [24], [36], [37].

B. STATIC POWER MANAGEMENT

With the development of processor technology, the energy saved by DVS has been significantly limited with the dramatic increase of the static power consumption caused by leakage current. According to [30], with processor size shrinking, supply voltages and threshold voltages of CMOS circuits shrink too, which results in exponential increases in sub-threshold leakage current. At 180 *nm*, the leakage current is negligible so the static power consumption can be ignored. At 130 *nm*, static power consumption occupies 10% to 30% of total energy. At 70 *nm*, the static power consumption occupies more than 50% of total energy. So at and under 70 *nm* process technologies, static power consumption would be as important as dynamic power consumption. While DVS effectively reduces the dynamic power consumption caused by switching or transistors, the execution time to finish jobs would be extended which leads to more static power consumption. Note that when the processor is idle, the static power is still consumed. So the total energy is not as efficiently saved as that would be when we only consider dynamic power consumption with DVS. In order to obtain the minimum of total energy consumption, we must take static power consumption into consideration.

Leakage control method takes static power consumption into consideration. The main idea is to shutdown the processor when it is idle and then wake it up when the next job comes. The longer the idle interval, the more static power consumption can be saved with power shutdown. Shutting down and waking up a processor cost energy. *Procrastination* method is used to help avoid frequent shutdown. The idea is to delay the execution of an incoming job, so that small intervals can be merged for power shutdown to save more energy (less shutdown overhead).

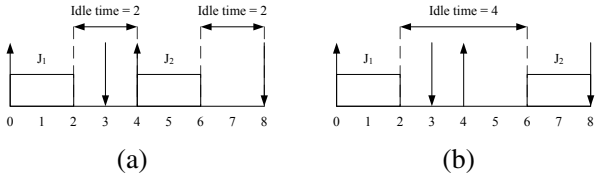


FIGURE 1. Leakage control method. (a) Power shutdown. (b) Procrastination.

Fig. 1 shows a simple example [25] about processor shutdown and task procrastination strategy. Suppose there are two jobs J_1 ($a_1 = 0, d_1 = 3, e_1 = 2$) and J_2 ($a_2 = 4, d_2 = 8, e_2 = 2$), where a_i, d_i and e_i are the arrival time, deadline and worst-case execution time (WCET) of J_i respectively. In Fig. 1(a), the processor is shut down twice during the idle intervals. But if J_2 is not executed until time point 6 (procrastination), the processor only needs to be shutdown once and can get the same length of idle interval while meeting the deadlines, as shown in Fig. 1(b).

Lee *et al.* are the first ones who used leakage control [12] LC-EDF extends the idle interval by procrastinating the next coming job as late as possible when the processor is idle. But this algorithm only uses power shutdown to save static power consumption. It does not consider DVS technology to save dynamic power consumption, which means all the tasks are executed at maximum speed. Thus this algorithm does not optimize the total power consumption. Jejurikar *et al.* [8], [10] proposed a static task-level algorithm called CS-DVSP, which computes the maximum procrastination length for each task based on the utilization factor while meeting all time constraints. This algorithm combines the DVS, power shutdown and procrastination method together successfully and gets a much better power performance for periodic tasks. But in their algorithm the procrastination length of every job generated by the same task has the same length. Meanwhile, CS-DVSP can only be applied when the relative deadline is equal to the period of each task. Further, in [9], Jejurikar *et al.* also proposed a dynamic algorithm to reduce total energy based on their previous theory. Chen *et al.* made some improvements to CS-DVSP, and developed a new algorithm called P-Procrastination [3]. The idea is to control the ratio for the residual interval and the procrastination interval so that the procrastination is less greedy. It is proved that greedy procrastination might sacrifice the possibility to turn off the processor in the near future, and hence, might consume more energy. Niu and Quan [24] proposed a static job-level algorithm called DVSLK which also adopts leakage control method and DVS strategy to manage power. Since this is a job-level static algorithm, the procrastination length is determined for every job much more precisely. Meanwhile, it can also be applied when the deadline is not equal to the period for each task.

These existing algorithms [3], [8]–[10], [12] which combine DVS and leakage control method for power management did not consider reliability.

C. PREVIOUS WORK ON RELIABILITY AWARE DVS ALGORITHM

Some researches have been studied to combine the DVS and fault tolerance together, aiming at reducing power consumption while processing all the faults successfully and meeting all the time constraints [33], [38]. Most of these previous researches either focused on tolerating fixed number of faults or assumed constant fault rate when applying DVS for energy savings. Zhu *et al.* are the first ones who considered changed fault rate when exploring the trade-off between reliability and energy consumption in real-time embedded systems. Studies show that reducing the supply voltage for lower frequency during the power management would result in exponentially increased fault rates [45]. It means the excess of implementing DVS during the power management would lead to lower reliability of the real-time system.

Previous work focuses on either power management with DVS and leakage control method, or reliability guaranteed algorithms with DVS technique only. Zhu *et al.* proposed static task-level reliability aware algorithms SUF/LUF and corresponding dynamic algorithms for periodic tasks [44]. Although the leakage power has been considered in the algorithm, the processor are not put into sleep during scheduling. Our method SUF-L [26] extends SUF with leakage control method. To the best of knowledge, this is the first research effort that tries to derive better energy performance with the combination of shutdown and slowdown techniques while the reliability of the system is guaranteed.

SHR proposed by Zhao *et. al.* [39] adopts single fault assumption and uses shared-recovery technique where a shared recovery block/slack that can be used by any faulty task at run-time. SHR is more efficient than SUF, and gets much better energy performance. However, SHR only works for a periodic task set with tasks of the same deadline/period. Our second method SUF-S [26] is to extend the shared-recovery technique to the case where the periods of the tasks in the task set may be different. Zhao *et al.* also proposed a technique called Generalized Shared Recovery (GSHR) [40] where multiple shared recovery block can be used by any faulty task. Given a reliability goal, GSHR determines the optimal number of recoveries and the task-level processing frequencies to minimize the energy consumption while achieving the reliability goal and meeting the timing constraints.

D. OUR WORKS

Considering the two approaches mentioned in Section 1, our works can be illustrated from the two aspects (see Fig. 2). First, under the requirement of guaranteed reliability as without DVS, SUF/LUF [44] algorithms are adopted to reserve enough slacks for potential recovery job. Then, we use leakage control method and shared-recovery technique to reduce energy consumption [26]. In this paper, we would further improve the energy performance with slack reclaim technique. Second, when reliability level is given as a constraint, the problem can be formulated and solved with convex optimization technique. Further, we use leakage control method to improve the energy performance as well.

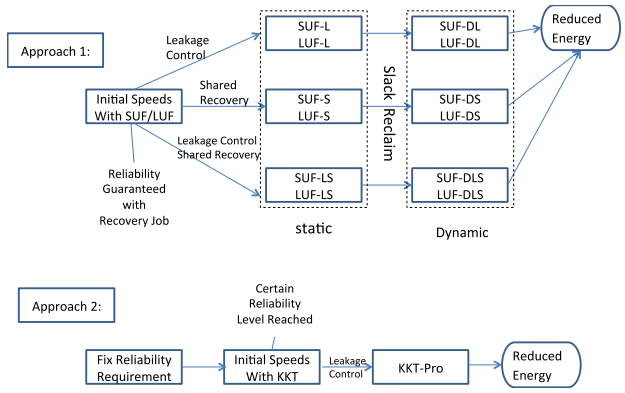


FIGURE 2. Framework for energy and reliability co-design.

III. MODELS

Next, we show the models, which are also adopted in [26].

A. TASK MODEL

The task model under consideration is a periodic task set. Each task T_i is modelled as (a_i, e_i, d_i, p_i), $1 \leq i \leq n$ where n is the number of tasks; a_i is the arrival time; e_i is the worst case execution time (WCET) when the processor runs with maximum speed (s_{max} will be normalized to 1); d_i is the deadline; and p_i is the period of the task. There have been works on estimating the WCET of tasks (either computational tasks or communication tasks) of various types of programming language, under various architecture [1], [13], [15], [17], [35].

The tasks are scheduled with preemptive EDF policy. Schedulability test checks whether all the tasks can be feasibly scheduled without missing any deadline. For periodic tasks scheduled by a preemptive EDF scheduler on a single processor, the schedulability test condition is that all the periodic task set's total utilization is equal to or less than 100% [20], that is, $U = \sum_{i=1}^{n} (\frac{e_i}{p_i}) <= 1, 1 \leq i \leq n$.

B. POWER MODEL

We adopt the power model in [10]. The power consumption of the tasks can be divided into two parts: *dynamic power consumption* (P_{dyn}) and *static power consumption* (P_{stat}). Dynamic power consumption consists of the switching power for charging and discharging the load capacitance, and is defined as: $P_{dyn(f)} = C_{eff} V_{dd}^2 f$, where C_{eff} is the effective switching capacity, V_{dd} is the supply voltage and f is the operating frequency/speed. Static power consumption consists of the power consumed by the subthreshold leakage and the reverse bias junction current [22]. The static power per CMOS circuit is defined as: $P_{stat(f)} = V_{dd} \times I_{subn} + |V_{bs}| \times I_j$, where V_{dd} is the supply voltage, I_{subn} is the subthreshold current, V_{bs} is the body bias voltage and I_j is the reverse bias junction. Thus, the total energy consumed during the active mode for time period t is

$$E_{(f)} = P_{dyn(f)} \times t + L_g \times P_{stat(f)} \times t, \tag{1}$$

where L_g is the number of logic gates in the processor circuit.

C. CRITICAL SPEED

s_{cri}, the *critical speed* (or *threshold speed*), is defined as the speed to execute a cycle with the minimum total energy consumption [3], [10], [24], [44]. When processor is active (not idle), either increasing or decreasing the processor speed would consume more total energy than executing with s_{cri}. However, not all the jobs can be executed with s_{cri}, as it will cause some jobs missing deadlines. Therefore, the jobs of a system would be executed with the speeds between s_{cri} and s_{max}.

D. FAULT MODEL

The transient fault of the system will increase when the system runs at a reduced speed [45]. Assuming the transient faults follow Poisson Distribution [38], the average transient fault rate for a job running at frequency f and voltage V is [45]:

$$\lambda(f, V) = \lambda_0 \times g(f, V), \tag{2}$$

where λ_0 is the average fault rate corresponding to V_{max} and f_{max}. At the lowest frequency f_{min} and supply voltage V_{min}, the average fault rate is assumed to be [45]:

$$\lambda_{max} = \lambda_0 \times 10^d, \tag{3}$$

$d(> 0)$ is a constant. So for the job running at f and V where $V = f \times V_{max} = f$ (V_{max} is normalized to be 1), the average transient fault rate can be expressed as [45]:

$$\lambda(f, V) = \lambda(f) = \lambda_0 \times 10^{\frac{d(1-f)}{1-f_{min}}}, \tag{4}$$

from Formula (4), we can easily derive that $g(f, V) = 10^{\frac{d(1-f)}{1-f_{min}}} > 1$ for $f < f_{max}$ and reducing the supply voltage for lower frequency results in exponentially increased fault rates.

For a given average error occurrence rate of λ, according to the Poisson model, the probability of r occurrences in a unit interval is given by $P(X = r) = e^{-\lambda}\lambda^r / r!$. When $r = 0$, we can derive the probability of no fault during an interval of interest. The probability of no-fault (P_{nf}) during the running

interval of a job with execution time e_i at a given speed s can thus be expressed [45] as

$$P_{nf}(s) = e^{-\lambda(f,V)\times\frac{e_i}{s}}. \quad (5)$$

Equation (5) not only considers the duration of the job (which is determined by the execution time of the job and the speed s used to run the job), but also the average fault rate determined by the frequency and voltage change (equation 4).

Fig. 3 shows the probability of no fault under various speeds for job J (with execution time $e_i = 10$, constant $d = 2$ and $f_{min} = 0.41$). If we define the reliability level as the no-fault probability. One can observe that the reliability loss depends on how much the speed is scaled down. If one requires that the task has only up to 0.01% reliability loss, then the no-fault probability due to task slow down has to be bigger than 99.99% and the corresponding minimum speed of the task will be set to 0.75.

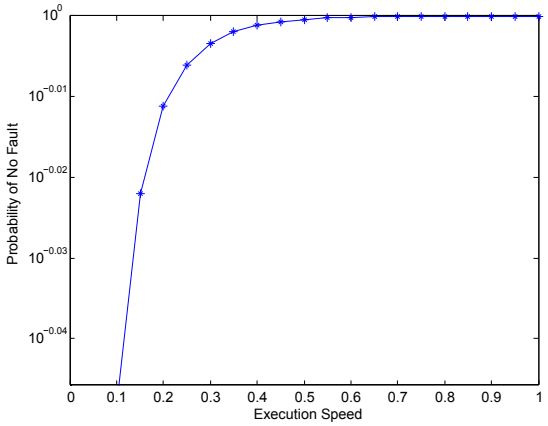


FIGURE 3. Probability of no fault.

IV. GUARANTEEING RELIABILITY AS TASKS NOT SCALED

The fist approach guarantees reliability at least as high as that without voltage scaling by reserving recovery job for each scaled down task. This approach is based on the previous reliability aware DVS algorithms [45]. In order to guarantee that the reliability of every job after implementing DVS be no less than its original reliability (which means the probability of correctly completing the job at maximum speed), a recovery job is arranged after each scaled down primary job [45]. Although the leakage power has been considered in these reliability aware DVS algorithms [39], [45], the processor is always active during the scheduling. Our approach is to extend these previous methods with leakage control method.

A. STATIC ALGORITHMS

Our previous method SUF-L [26] is the first method that extends a reliability aware DVS algorithm SUF with leakage control method.

Essentially, SUF-L determines the scaled speed of each task and when to shutdown the processor. To guarantee reliability, each scaled down task will be arranged with a recovery job to calculate the initial scaled down speed of the tasks. The initial speed is calculated statically by SUF [45] which chooses the first m smallest utilization tasks to scale down and reserves time slots for each scaled down task. SUF-L then precomputes procrastination length for each task. The execution speed and procrastination length of every job generated by the same task would be the same. The procrastination length calculation is based on an existing procrastination algorithm CS-DVSP [10]. The difference is that in our calculation, in order to guarantee the reliability of the system, we must reserve some slacks for potential recovery job tolerance.

At run time, the scheduler schedules the tasks using EDF policy and sets the speed of the tasks as the precalculated initial speed. If any fault occurs during the execution, then a recovery job with a maximum speed will be executed. When the processor is idle, the scheduler will determine whether shutting down the processor or not based on the precalculated critical interval. The detailed algorithm for SUF-L can be found in Algorithm 1 [26].

Algorithm 1 SUF-L: The Leakage Control Algorithm

```
Calculate initial speed for each task by SUF;
Calculate procrastination length L_i for each task i;
During the scheduling:
if (Processor is not idle) then
    Schedule jobs according to EDF policy;
else
    Calculate next coming job J_i's arrival time a_i;
    Judge if there is any job J_j's arrival time a_j is between
    a_i and (a_i + L_i);
    Set the wake up time to be min(a_i + L_i, a_j + L_j);
    if (sleep time interval is longer than critical interval)
    then
        shutdown the processor until wake up time;
    else
        Keep processor in idle state until next job arrives;
    end if
end if
```

Another method SUF-S is also described in [26] to lower the total energy consumption by shared-recovery job method for tasks with various deadlines. This is based on the single fault occurrence assumption during a job execution [39]. By finding the shared recovery job, the recovery slack can be used to slow down the tasks further. The algorithm SUF-LS [26] combines leakage control and shared-recovery method to achieve better total energy saving while guaranteeing reliability with single fault occurrence assumption.

The SUF-L, SUF-S and SUF-LS algorithms are all static methods where we assume all the tasks run with their worst case execution time (WCET). The actual execution time (AET) of a task is normally less than its WCET. Therefore, the static algorithms are pessimistic in the energy saving. Next,

we describe the dynamic algorithms for SUF-L, SUF-S and SUF-LS algorithms which are named SUF-DL, SUF-DS and SUF-DLS algorithms, respectively.

B. DYNAMIC ALGORITHMS

The static algorithms assume that every job executes with WCET. But many jobs finish earlier than their WCETs and thus generate slacks. For example a program may jump out of a loop earlier than the worst case once some condition is met. After taking this fact into consideration, each job would be finished with an actual workload between best case workload and worst case workload, and an extra slack time may be generated. The actual workload of every job can not be predicted until the job releases.

We denote the slack generated during the scheduling as $\bar{S}$ ($\bar{a}$, $\bar{l}$, $\bar{e}$, $\bar{d}$) where $\bar{a}$ stands for slack's arrival time, $\bar{l}$ stands for the slack's length, $\bar{e}$ stands for slack's end time, and $\bar{d}$ stands for slack's deadline. While constructing slack during scheduling, the following rules would be followed: $\bar{a}$ is equal to the source job's finish time, $\bar{l}$ is unused workload divided by job's execution speed and $\bar{e} = \bar{a} + \bar{l}$. Note that, in order to guarantee the schedulability, a new slack inherits the deadline of the early-completed source job which generates the slack. For performance of the dynamic algorithm, we use a conservative slack reclaiming method. Any future job that wants to reclaim a slack must satisfy that it arrives before slack's end time and the job's deadline is no earlier than the slack's deadline [44].

1) SUF-DL: THE LEAKAGE CONTROL DYNAMIC ALGORITHM

Next, we describe SUF-DL (shown in Algorithm 2) which uses the slack-reclaim technique with SUF-L to exploit the run-time variations in task execution for further total energy saving.

In the case when AET is less than WCET, a slack will be generated for a job. With these extra idle intervals, we can apply leakage control more often than SUF-L.

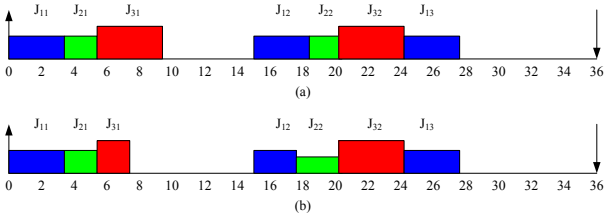


FIGURE 4. SUF-L and SUF-DL.

We adopt the following periodic task set illustrate SUF-DL: $\{T_1(p_1 = 12, e_1 = 2), T_2(p_2 = 17, e_2 = 1), T_3(p_3 = 18, e_3 = 4)\}$. Fig. 4 shows the scheduling results of applying SUF-L and SUF-DL algorithms to the above task set. After applying SUF algorithm, T_1 and T_2's initial speed is 0.58, T_3's initial speed is 1 and the new utilization is 0.61. Then we can calculate the procrastination length (L_i) of every task and result in $L_1 = L_2 = L_3 = 3$.

In SUF-DL, during the scheduling, suppose the actual workload of J_{31} is 2, so J_{31} finishes at time 7.17, earlier than time 9.17 in SUF-L. A slack $\bar{S}_{31}(\bar{a}_{31} = 7.17, \bar{l}_{31} = 2, \bar{e}_{31} = 9.17, \bar{d}_{31} = 12)$ is constructed which can be used to extend the shut down length. In this case, the processor can be shut down for 7.83 time units, which is longer than that in SUF-L. At time 15, J_{12} arrives with the actual workload 1.5. A slack of $\bar{S}_{12}(\bar{a}_{12} = 17.58, \bar{l}_{12} = 0.46, \bar{e}_{12} = 18.44, \bar{d}_{12} = 24)$ is generated after J_{12} is finished, which can be used to further reduce J_{22}'s execution speed from 0.58 to 0.53. Scaled down speed can efficiently reduce dynamic power consumption and with the longer idle interval, the processor can be shut down for a longer time to save more static power consumption. So, with the reclaimed slack technique, the total energy consumption is further reduced. The time complexity of SUF-DL is $O(N^2)$ (N is the number of tasks in a task set).

Algorithm 2 SUF-DL: The Leakage Control Dynamic Algorithm

```
1: Calculate initial speeds for each task by SUF;
2: Calculate procrastination length for each task;
3: At each time tick
4: Update the slack list;
5: if (Processor is not idle) then
6:     On execution of a new job J_i:
7:     if (s_i is between s_cri and s_max and slack list is not
       empty) then
8:         Use slack if there is any, for scaling down the current
           executing job ;
9:     end if
10:    Schedule jobs according to EDF policy;
11:    if (J_i is finished with unused workload) then
12:        Construct the slack and put it into slack list;
13:    end if
14: else
15:    Calculate next coming job J_i's arrival time a_i;
16:    Judge if there is any job J_j's arrival time a_j is between
       a_i and (a_i + L_i);
17:    Set the wake up time to be min(a_i + L_i, a_j + L_j);
18:    if (sleep time interval is longer than critical interval)
       then
19:        shutdown the processor until wake up time;
20:    else
21:        Keep processor in idle state until next job arrives;
22:    end if
23: end if
```

During scheduling, all slacks are stored in a slack list according to the deadlines. Earlier generated slacks would be placed forward. A slack $\bar{S}_i(\bar{a}_i, \bar{l}_i, \bar{e}_i, \bar{d}_i)$ can be reclaimed by $J_j(a_j, e_j, d_j)$ in the two following situations:

- $\bar{a}_i \geq a_j$ & $\bar{e}_i \leq d_j$ & $\bar{d}_i \leq d_j$, the whole slack can be reclaimed.
- $\bar{a}_i \leq E_j$ & $\bar{e}_i \leq d_j$ & $\bar{e}_i \geq a_j$ & $\bar{d}_i \leq d_j$, only the slack from a_j to $\bar{e}_i$ can be reclaimed.

Meanwhile, in order to guarantee the reliability, for jobs with initial speeds less than maximum speed, slacks can be reclaimed once slacks satisfy the above conditions. For jobs with initial speeds equal to maximum speed, the slacks can be reclaimed only if it is long enough to tolerate the primary job. This is because in the case when the execution speed is s_{max}, no recovery job was reserved. If the slack allows a scale down, a recovery job has to be reserved to guarantee reliability. The slack has to be at least as long as the primary job in order for the scale to occur without compromising the reliability.

Algorithm 3 SUF-DS: The Shared-Recovery Dynamic Algorithm

1: Calculate initial speed for each task by SUF;
2: At each time tick
3: Update the slack list;
4: **On execution of a new job** J_i:
5: **if** (s_i is between s_{cri} and s_{max} && slack list is not empty) **then**
6: Use the slack, if there is any for the current executing job;
7: **end if**
8: **if** (s_i is between s_{cri} and s_{max} && R_i has never been shared) **then**
9: Construct a new job list with all jobs arriving between a_i and d_i;
10: Schedule the new job list and judge any job J_j can share recovery job with J_i according to the *SRJ rule* [26];
11: **if** (J_j can share recovery with J_i) **then**
12: Update J_i's execution speed;
13: Mark R_i and R_j have been shared;
14: **end if**
15: Schedule J_i with new execution speed under EDF policy;
16: **else**
17: Schedule J_i with initial speed under EDF policy;
18: **end if**
19: **if** (J_i is finished and there exists unused workload) **then**
20: Construct the slack and put it into slack list;
21: **end if**

2) SUF-DS: THE SHARED-RECOVERY DYNAMIC ALGORITHM

SUF-DS is a dynamic algorithm combining the slack-reclaim technique and SUF-S. Both slack-reclaim and shared-recovery can slow down a task further either by making use of slack or the shared-recovery job. We choose to use slack-reclaim technique first, and shared-recovery technique is applied only if the job's speed is still over critical speed after slack is reclaimed. There are two reasons for this. One is that slack has end time and it can not be reclaimed any more once the end time is missed. And the second reason is that the complexity of pre-scheduling is higher than that of slack-reclaim. Note that WCETs are used during pre-schedule for shared-recovery technique [26]. Algorithm 3 is the detailed description of SUF-DS algorithm.

3) SUF-DLS: THE COMBINED DYNAMIC ALGORITHM

We further improve SUF-LS with the slack-reclaim technique. Since the leakage control method, shared-recovery technique and slack-reclaim technique are all used in SUF-DLS, we need carefully arrange the structure of algorithm. Generally, the leakage control method is adopted when processor is idle to extend the power down interval and reduce static power consumption; the shared-recovery technique can further scale down execution speed when processor is not idle to reduce dynamic power consumption. The slack-reclaim technique is able to reduce both dynamic and static power consumption. In SUF-DLS, we choose to use slack-reclaim technique first, leakage control method during processor's idle time and shared-recovery technique only if the job's speed is still over critical speed after slack reclaimed. The same as SUF-LS, we must use the leakage control method in pre-scheduling as well when we try to find the sharing-recovery job in pre-shceduling. The time complexity of SUF-DLS is $O(N^2)$ (where N is the number of tasks in a task set). The details of SUF-DLS are described in Algorithm 4.

V. GUARANTEEING MINIMUM RELIABILITY CONSTRAINT

For a given application, if the maximum tolerable reliability loss is known in advance, then we can formulate the energy co-design problem as an optimization problem with the objective function as the energy consumption subject to a speed limit constraint. The minimum speed is set using the relation between the reliability and speed.

Assume the minimum reliability speed for a periodic task set is s_{rel}. As the critical speed s_{cri} is the speed that consumes the least energy, we set s_{rel} to s_{cri} when $s_{rel} < s_{cri}$. For a task set including n periodic tasks, the energy minimization problem with minimum reliability constraint can be expressed as follows. (e_i, p_i, s_i) represents the execution time, period and speed of task i.

$$\textit{minimize} \qquad E_{(s)} = \sum_{i=1}^{n} E_{i(s_i)} \tag{6}$$

$$\textit{subject to} \qquad \sum_{i=1}^{n} U_{i(s_i)} = \sum_{i=1}^{n} \frac{e_i}{s_i \times p_i} \leq 1 \tag{7}$$

$$s_{rel} \leq s_i \leq s_{max} \quad i = 1, \ldots, n. \tag{8}$$

Formula (7) guarantees the schedulability of the system and the constraint in Formula (8) is to guarantee the reliability level of the system. The formula can be easily extended to accommodate the case where each task has an individual reliability constraint. According to [6], $E_i(s_i)$ strictly increases with s_i within the interval $[s_{rel}, s_{max}]$ for each task. Thus, there are two cases for the above problem:

- case 1: If $\sum_{i=1}^{n} U_{i(s_{rel})} \leq 1.0$, then $s_i = s_{rel}$ is the optimal solution.

Algorithm 4 SUF-DLS: The Combined Dynamic Algorithm

```
1: Calculate initial speed for each task by SUF;
2: Calculate procrastination length for each task;
3: At each time tick
4: Update the slack list;
5: if (Processor is not idle) then
6:    On execution of a new job J_i:
7:    Judge if there is any slack can be used for the current
      executing job;
8:    if (s_i == s_max) then
9:       Schedule J_i under EDF policy.
10:   else
11:      if (s_i < s_max and R_i has not been shared) then
12:         Construct a job list with the primary job and the
            recovery job arriving between a_i and d_i:
13:         Pre-Schedule each primary job with recovery job
            under EDF policy
14:         if (There is idle interval in the pre-schedule) then
15:            Judge whether to shutdown processor or not
16:         else
17:            if (a job can share recovery job with J_i accord-
               ing to the SRJ rule) then
18:               Update s_i
19:            end if
20:            Schedule J_i with speed s_i under EDF policy
21:         end if
22:      end if
23:   end if
24: else
25:   Calculate next coming job J_i's arrival time a_i;
26:   Judge if there is any job J_j's arrival time a_j is between
      a_i and (a_i + L_i);
27:   Set the wake up time to be min(a_i + L_i, a_j + L_j);
28:   if (sleep time interval is longer than critical interval)
      then
29:      shutdown the processor until wake up time;
30:   else
31:      Keep processor in idle state until next job arrives;
32:   end if
33: end if
```

- case 2: If $\sum_{i=1}^{n} U_{i(s_{rel})} > 1.0$, then in the optimal solution, $\sum_{i=1}^{n} U_{i(s_i)} = 100\%$.

For case 2, the problem can be transformed to be the following optimization problem.

$$\begin{aligned} \textit{minimize} \quad & E_{(s)} = \textstyle\sum_{i=1}^{n} E_{i(s_i)} && (9)\\ \textit{subject to} \quad & \textstyle\sum_{i=1}^{n} \frac{e_i}{s_i \times p_i} = 1 && (10)\\ & s_i \geq s_{rel} \quad i = 1, \ldots, n && (11)\\ & s_i \leq s_{max} \quad i = 1, \ldots, n. && (12) \end{aligned}$$

The above problem has the same form as the ENERGY-LU problem described in [6], which is a separable convex optimization problem with n unknowns, $2n$ inequality constraints and 1 equality constraint. Previous researches [6], [21], [43] have proved that such convex optimization problem can be solved by using the Karush-Kuhn-Tucker(KKT) optimality conditions [39].

As described in [21], for nonlinear problems of the following form where $f_{(x)}$ is convex:

$$\begin{aligned} \textit{minimize} \quad & f_{(\vec{x})} = f_{(x_1,\ldots,x_n)}\\ \textit{subject to} \quad & g_{1(\vec{x})} \leq b_1\\ & \vdots\\ & g_{n(\vec{x})} \leq b_n. \end{aligned}$$

We can find necessary and sufficient KKT conditions subject to regularity conditions on the constraints $g_{i(x)}$ that are always fulfilled for linear constraints. At the optimum point $\vec{x_e}$ we must have:

$$\begin{aligned} \forall j: \quad & \tfrac{\partial f_{(\vec{x})}}{\partial x_j} + \textstyle\sum \lambda_i \times \tfrac{\partial g_{i(\vec{x})}}{\partial x_j} = 0\\ \forall i: \quad & \lambda_i \times (b_i - g_{i(\vec{x})}) = 0\\ \forall i: \quad & \lambda_i \geq 0. \end{aligned}$$

The nonlinear optimization theory states that the solution of our problem should satisfy KKT conditions as follows.

$$E_i'(s_i) - \lambda \frac{U_i}{s_i^2} + \bar{\mu}_i - \mu_i = 0, \qquad i = 1, 2, \ldots, n \quad (13)$$

$$\bar{\mu}_i(s_{max} - s_i) = 0, \qquad i = 1, 2, \ldots, n \quad (14)$$

$$\mu_i(s_i - s_{rel}) = 0, \qquad i = 1, 2, \ldots, n \quad (15)$$

$$\bar{\mu}_i \geq 0, \qquad i = 1, 2, \ldots, n \quad (16)$$

$$\mu_i \geq 0, \qquad i = 1, 2, \ldots, n \quad (17)$$

where λ, $\bar{\mu}_i$, μ_i are Lagrange multipliers and $U_i = \frac{e_i}{p_i}$. Following the steps in [6], we can find the optimal set of speeds for every task. Details of theory proofs are given in [6] already, we only briefly introduce the procedure as follows.

In order to solve the original problem, we can solve the problem without considering upper speed bound first and the KKT conditions for the new problem is transformed to:

$$E_{i(s_i)}' - \lambda \frac{U_i}{s_i^2} - \mu_i = 0, \qquad i = 1, 2, \ldots, n \quad (18)$$

$$\mu_i(s_i - s_{rel}) = 0, \qquad i = 1, 2, \ldots, n \quad (19)$$

$$\mu_i \geq 0, \qquad i = 1, 2, \ldots, n \quad (20)$$

where λ, μ_i are Lagrange multipliers. In order to solve the problem, we need to find a suitable Lagrange multiplier λ satisfying $\lambda = \frac{s_i^2}{U_i^2} E_{i(s_i)}' = \frac{s_j^2}{U_j^2} E_{j(s_j)}' \quad \forall i, j$. After finding a certain λ, we can derive a set of speeds that satisfies (17) and (18) at the same time. If all these derived speeds are less than upper speed bound, then it is the solution for the original problem. Otherwise, we set the execution speed of the task with the smallest period to be maximum speed and calculate the remaining tasks' speeds again.

The next question is how to find the reasonable Lagrange multiplier λ. If the functions of task utilization $U_{(s)}$ and Lagrange multiplier λ are monotone, then we can use the

dichotomy methods to find the optimal λ. With variable speed s ($s_{rel} \leq s \leq s_{max}$), $U_{(s)} = \frac{e}{s \times p}$ is monotone decreasing obviously for any periodic task. The function monotonicity of λ depends on the power model of the system. One can easily prove that the Lagrange multiplier λ is monotone increasing with speeds between s_{rel} and s_{max} for the adopted power model in [10].

Proof 1: For a periodic task T_i with execution speed s, $\lambda_{(s)} = \frac{s^2 \times p_i}{c_i} \times E'_{(s)} = \frac{s^2 \times p_i}{e_i} \times (P_{(s)} \times \frac{e_i}{s})'$. Using the power model E in Section III-B with detailed power model parameters in [10], we derive:

$$\begin{aligned}\lambda_{(s)} = p_i \times [&0.6665s^3 + 0.6665s^2 + (0.052414s^2 \\ &+ 0.052414s - 0.0572831) \\ &\times e^{1.83(\frac{s}{2}+0.5)} - 13.44 \times 10^{-4}].\end{aligned} \tag{21}$$

The necessary and sufficient condition of the proof is that the derivative of the function is larger than 0 for variable s between s_{rel} and s_{max}.

$$\begin{aligned}\lambda'_{(s)} = p_i \times [&1.9995s^2 + 1.333s + (0.104828s + 0.052414) \\ &\times e^{1.83(\frac{s}{2}+0.5)} + (0.052414s^2 + 0.052414s \\ &- 0.0572831) \times e^{1.83(\frac{s}{2}+0.5)} \times 0.915].\end{aligned} \tag{22}$$

After simple transformation, we derive:

$$\begin{aligned}\lambda'_{(s)} = p_i \times [&1.9995s^2 + 1.333s + (0.04795881s^2 \\ &+ 0.15278681s - 0.0000000365) \\ &\times e^{1.83(\frac{s}{2}+0.5)}].\end{aligned} \tag{23}$$

All $1.9995s^2$, $1.333s$, $(0.04795881s^2 + 0.15278681s - 0.0000000365)$ and $e^{1.83(\frac{s}{2}+0.5)}$ obviously are larger than 0 for variable s between s_{rel} and s_{max}. Thus, $\lambda'_{(s)}$ is a positive number as well and λ is monotone increasing for speeds between s_{rel} and s_{max}.

After proving the function monotonicity, we can use Algorithm 5 to find the optimal speed. Here, we define s_{low} and s_{up} in the algorithm to be the current lower and upper bound speed in each loop, s_{low} initialized to be s_{rel} and s_{up} initialized to be s_{max} at the beginning.

In step 5, in order to derive other tasks' speeds with the same λ, we can use dichotomy method since λ is monotone increasing with the speed as we have proved. In step 7 to 12, with the current speed set, if $U_{(s)} > 1$, it means $U_{(s)}$ should be smaller. Since $U_{(s)}$ is monotone decreasing with speed, the speeds should be higher and the λ would be higher as well. So we keep the s_{up} and reset the $s_{low} = \frac{s_{low}+s_{up}}{2}$ to recalculate again.

The time complexity of the algorithm is $O(N^2)$, where N is the number of tasks in the task set.

With the above energy optimization problem with reliability constraints solved, each task's initial speed is derived. Since these initial speeds are between s_{rel} and s_{max}, there would still leave some slacks during the scheduling especially when $\sum_{i=1}^{n} U_{i(s_{rel})} < 1.0$. Regardless of these slacks, it would lead to a waste of energy if the static power is considered. Thus we adopt leakage control method during idle time and develop a new algorithm called KKT-Pro. In KKT-Pro, the procrastination length for every task is calculated the same as that in CS-DVSP [10] in advance. During the scheduling, each task would execute with initial speed under EDF policy. While the processor is idle and new job arrives, we find the longest procrastination length so that the processor can be shut down for a longer time. The simulation results and analysis details will be given in Section VI.

The method can be easily extended to the case where s_{rel} is different for different tasks.

Algorithm 5 Find Optimal Speed

1: **while** All speeds in the speed set is no less than s_{rel} **do**
2: **while** Utilization of the task set is not equal to 1 and $s_{low} < s_{up}$ **do**
3: Initialize the first task's speed to be $\frac{s_{low}+s_{up}}{2}$;
4: Calculate the current λ with the speed;
5: Calculate every other task' speed with the same λ and derives s_i individually;
6: Calculate current task set's utilization $U_{(s)}$;
7: **if** $(U_{(s)} > 1)$ **then**
8: Reset the $s_{low} = \frac{s_{low}+s_{up}}{2}$;
9: **end if**
10: **if** $(U_{(s)} < 1)$ **then**
11: Reset the $s_{up} = \frac{s_{low}+s_{up}}{2}$;
12: **end if**
13: **end while**
14: **if** Any task's speed is less than s_{rel} **then**
15: Fix current the largest period task's execution speed to be minimum speed;
16: **end if**
17: **end while**

VI. SIMULATION RESULTS AND ANALYSIS

In this section, the following algorithms are simulated in a discrete single processor simulator to evaluate energy and reliability performance.

The power model and technology parameters of the processor used in the simulation are from [22] (The s_{cri} for this model is around 0.4 [10]). The overhead of processor power down/up is the same as that used in [10], where $P_{idle} = 240mW$, $E_0 = 483\mu J$, $t_0 = 2ms$, $L_g = 4\,000\,000$. In our simulation, the power settings are the same as those in [24]. To evaluate energy consumptions, we set up system utilizations from 0.1 to 0.9 for each algorithm. For each point in the chart, we generate 20 task sets and each task set is executed with five million time units. Every task set includes 20 periodic tasks. The periods and the WCETs in each task set are randomly chosen in the range of [10, 200] and [1, 10], respectively. Transient faults are assumed to follow the Poisson Distribution with an average fault rate of $\lambda_0 = 10^{-6}$ at f_{max} per megabit [45]. The same as [44], in order to take the effects of DVS into consideration, the exponent in the fault model $d = 2$, which means the average fault rate with

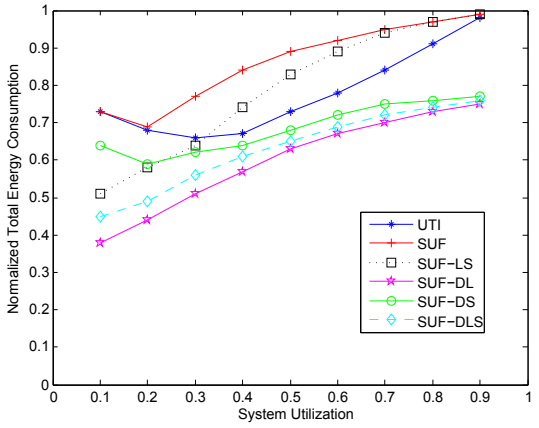
s_{min} is assumed to be 100 times higher than that of s_{max}. The energy consumptions by every algorithm are normalized to that consumed by NPM. And the probability of failure is calculated through dividing the number of faulty jobs by the total number of executed jobs.

A. ALGORITHMS GUARANTEEING RELIABILITY AS TASK NOT SCALED

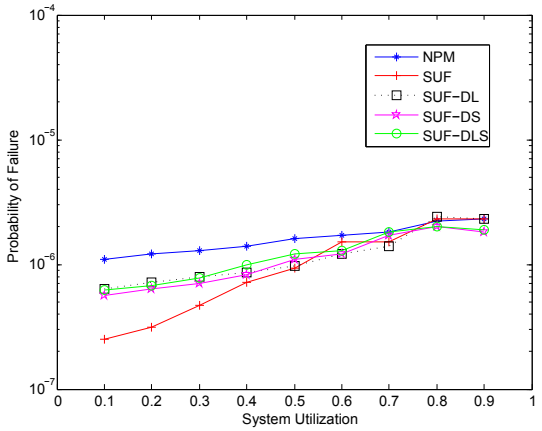
In [26], we have shown the simulation results for the static SUF-L(S)(LS)/LUF-L(S)(LS) based algorithms.

Here, we show the dynamic algorithms of the corresponding algorithms. All jobs' actual workloads are chosen from $\frac{WCET}{2}$ to $WCET$ randomly.

- SUF-DL/LUF-DL: the dynamic leakage control method combined with SUF/LUF task-level algorithm.
- SUF-DS/LUF-DS: the dynamic shared-recovery technique combined with SUF/LUF job-level algorithm.
- SUF-DLS/LUF-DLS: the combined dynamic leakage and shared-recovery job-level algorithm.



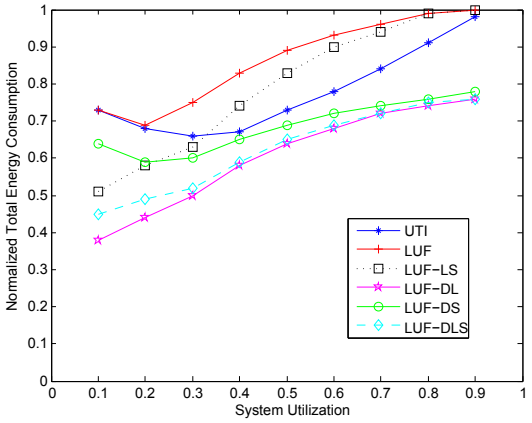
(a)



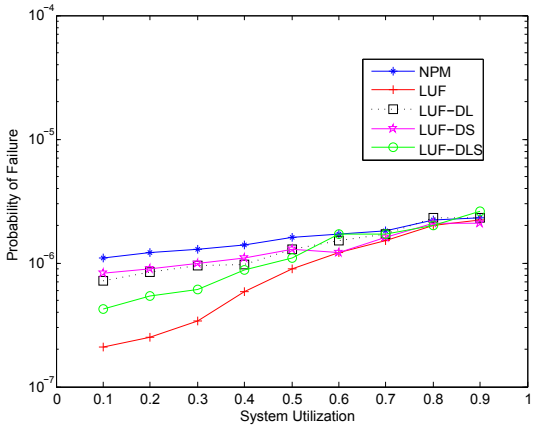
(b)

FIGURE 5. Simulation results of the SUF-based dynamic algorithms. (a) Energy consumption. (b) Probability of failure.

Figs. 5 and 6 show the simulation results of these algorithms. Figs. 5(a) and (b) are about the performance of the



(a)



(b)

FIGURE 6. Simulation results of the LUF-based dynamic algorithms. (a) Energy consumption. (b) Probability of failure.

SUF based dynamic algorithms while Figs. 6(a) and (b) are about the LUF based algorithms. The performance of SUF based algorithm is similar to LUF based algorithms.

As shown in Fig. 5(a), compared with static algorithm SUF-LS, nearly all dynamic algorithms can achieve at least 10% energy saving except for SUF-DS. The exceptional case (SUF-DS consumes more energy than SUF-LS) occurs when the system utilization is very low (less than 0.2). When system utilization is high, we can observe that SUF-DL, SUF-DS and SUF-DLS save more energy than SUF-LS.

We also observe that SUF-DLS consumes no less energy compared with that of SUF-DL. The reason is that in dynamic algorithms, workload is often much less and more slacks can be merged to shut down the processor in SUF-DL. But after combining the shared-recovery technique in SUF-DLS, it may lead to less chances to shut down the processors and thus the total power consumption is higher. Generally, SUF-DS can save at most 25% energy, and SUF-DL and SUF-DLS can reach more than 30% energy saving on average compared with SUF.

Figs. 5(b) and 6(b) show the probability of failure of these algorithms. With the recovery jobs arranged in advance, our

proposed schemes in this section all have lower probability of failure than NPM, which shows that the reliability of the system is guaranteed. With the slack-reclaim technique in our schemes, many jobs' execution speeds are further decreased. Therefore, the probability of failure is higher than that in SUF.

B. ALGORITHMS GUARANTEEING MINIMUM RELIABILITY CONSTRAINT

- NPM: all jobs are executed with the maximum speed, and processor is always on.
- UTI: all jobs are executed with $max(U, s_{cri})$, and the processor is always on. This is supposed to be optimal if only dynamic power is considered.
- CS-DVSP: a task-level oriented leakage control method [10].
- KKT: static KKT based task-level algorithm with $s_{rel} = s_{cri}$.
- KKT-Pro: static leakage and reliability considered KKT based task-level algorithm.

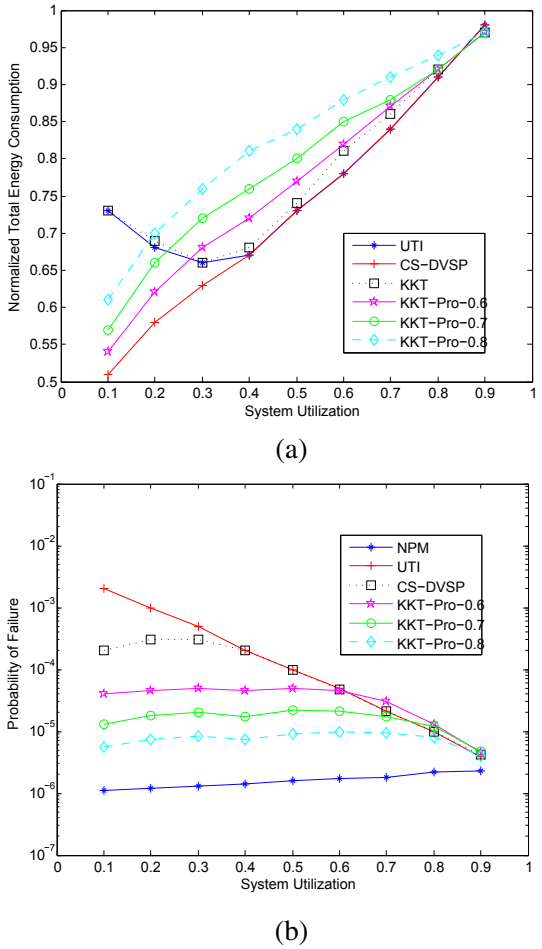


FIGURE 7. Simulation results of the KKT-based static algorithms. (a) Energy consumption. (b) Probability of failure.

Fig. 7 shows the simulation results of different algorithms. In KKT algorithm, without considering reliability, all tasks can execute from s_{cri} to s_{max}. In KKT-Pro, we choose several $s_{rel} = \{0.6, 0.7, 0.8\}$ to show the different impacts of s_{rel} on energy and reliability aspects.

Fig. 7(a) shows the total energy consumption performance with different algorithms. According to the results, CS-DVSP consumes the minimum energy (Note that CS-DVSP does not take reliability into account.). As the processor is always on in UTI algorithm, UTI algorithm is no longer the optimum scheme for energy performance with static energy is considered. Without considering probability of failure, original KKT algorithm consumes almost the same energy compared with UTI. After taking reliability into consideration, KKT-Pro-0.6/0.7/0.8 consumes more energy. With higher reliability speed, the lower bound of the speed is higher and more dynamic power is consumed. Compared with CS-DVSP, KKT-Pro-0.6 consumes at most 4% more energy on average, while KKT-Pro-0.7 with 7% more and KKT-Pro-0.8 with 10% more.

Fig. 7(b) shows the probability of failure with different algorithms. Without considering any power management method, NPM has the lowest probability of failure. CS-DVSP and UTI have the highest probability of failure, almost 10/50/70 times higher than KKT-Pro-0.6/0.7/0.8, which means if 1 000 000 jobs are executed, more than 500 faults would occur with CS-DVSP and UTI while only less than 50/10/7 faults would occur with KKT-Pro-0.6/0.7/0.8. Although the reliability of the system can not be guaranteed (as high as that without DVS) by KKT-Pro, with 4%/7%/10% more energy consumption, KKT-Pro-0.6/0.7/0.8 have some reliability improvements compared with KKT algorithm.

VII. CONCLUSIONS

After taking both static power management and system reliability aspects into consideration, energy saving algorithm design using both slowdown and shutdown techniques in cyber-physical systems becomes more complicated. In this work, we propose algorithms to approach total energy and reliability co-design problem from two different aspects. The first is to maintain reliability level via reserving recovery task and further use the leakage control method and the shared-recovery technique to lower total energy. The second one minimizes total energy consumption with reliability expressed as a constraint. Both static and dynamic algorithms have been described. The proposed algorithms are evaluated through simulation. The results show the effectiveness of our algorithms using both slowdown and shutdown techniques for scheduling co-design for reliability and total energy in real-time CPS.

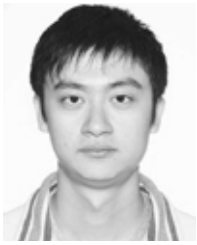
ACKNOWLEDGMENT

The authors would also thank D. Zhu and B. Zhao for their kind help during the work.

REFERENCES

[1] S. Byhlin, A. Ermedahl, J. Gustafsson, and B. Lisper, "Applying static WCET analysis to automotive communication software," in *Proc. 17th ECRTS*, 2005, pp. 249–258.
[2] X. Castillo, S. R. McConnel, and D. P. Siewiorek, "Derivation and caliberation of a transient error reliability model," *IEEE Trans. Comput.*, vol. 31, no. 7, pp. 658–671, Jul. 1982.
[3] J. J. Chen and T. W. Kuo, "Procrastination determination for periodic real-time tasks in leakage-aware dynamic voltage scaling systems," in *Proc. IEEE/ACM ICCAD*, Piscataway, NJ, USA, Nov. 2007, pp. 289–294.
[4] A. Gokhale, M. McDonald, S. Drager, and W. McKeever, "A cyber physical systems perspective on the real-time and reliable dissemination of information in intelligent transportation systems," DTIC, Fort Belvoir, VA, USA, Tech. Rep., 2010.
[5] S. K. S. Gupta, T. Mukherjee, G. Varsamopoulos, and A. Banerjee, "Research directions in energy-sustainable cyber-physical systems," *Sustain. Comput., Informat. Syst.*, vol. 1, no. 1, pp. 7–57, 2011.
[6] A. Hakan, V. Devadas, and D. Zhu, "System-level energy management for periodic real-time tasks," in *Proc. 27th IEEE Int. RTSS*, Dec. 2006, pp. 313–322.
[7] I. Lee and O. Sokolsky, "Medical cyber physical systems," in *Proc. 47th ACM/IEEE Design Autom. Conf.*, Jun. 2010, pp. 743–748.
[8] R. Jejurikar and R. Gupta, "Dynamic voltage scaling for systemwide energy minimization in real-time embedded systems," in *Proc. Int. ISLPED*, New York, NY, USA, 2004, pp. 78–81.
[9] R. Jejurikar and R. Gupta, "Dynamic slack reclamation with procrastination scheduling in real-time embedded systems," in *Proc. 42nd Annu. Conf. Design Autom.*, New York, NY, USA, 2005, pp. 111–116.
[10] R. Jejurikar, C. Pereira, and R. K. Gupta, "Leakage aware dynamic voltage scaling for real-time embedded systems," in *Proc. Design Autom. Conf.*, 2004, pp. 275–280.
[11] E. A. Lee, "Cyber physical systems: Design challenges," in *Proc. 11th IEEE ISORC*, May 2008, pp. 363–369.
[12] Y. H. Lee, K. P. Reddy, and C. M. Krishna, "Scheduling techniques for reducing leakage power in hard real-time systems," in *Proc. 15th Euromicro Conf. Real-Time Syst.*, 2003, pp. 105–112.
[13] Y. T. Li and S. Malik, "Performance analysis of embedded software using implicit path enumeration," *ACM SIGPLAN Notices*, vol. 30, no. 11, pp. 88–98, 1995.
[14] M. Lin, "Synthesis of control software in a layered architecture from hybrid automata," in *Hybrid Systems: Computation and Control*. New York, NY, USA: Springer-Verlag, 1999, pp. 152–164.
[15] M. Lin, "Timing analysis of PL programs," *Control Eng. Pract.*, vol. 8, no. 6, pp. 697–703, 2000.
[16] M. Lin and C. Ding, "Parallel genetic algorithms for DVS scheduling of distributed embedded systems," in *Proc. HPCC*, 2007, pp. 180–191.
[17] M. Lin and J. Malec, "Timing analysis of reactive rule-based programs," *Control Eng. Pract.*, vol. 6, no. 3, pp. 403–408, 1998.
[18] M. Lin, L. Xu, L. T. Yang, X. Qin, N. G. Zheng, Z. H. Wu, and M. K. Qiu, "Static security optimization for real-time systems," *IEEE Trans. Ind. Informat.*, vol. 5, no. 1, pp. 22–37, Feb. 2009.
[19] M. Lindberg and K-E Arzén, "Feedback control of cyber-physical systems with multi resource dependencies and model uncertainties," in *Proc. IEEE 31st RTSS*, Nov./Dec. 2010, pp. 85–94.
[20] J. W. S. Liu, *Real-Time Systems*. Englewood Cliffs, NJ, USA: Prentice-Hall, 2000.
[21] D. G. Luenberger, *Linear and Nonlinear Programming*. New York, NY, USA: Springer-Verlag, 1984.
[22] S. M. Martin, K. Flautner, T. Mudge, and D. Blaauw, "Combined dynamic voltage scaling and adaptive body biasing for lower power microprocessors under dynamic workloads," in *Proc. IEEE/ACM ICCAD*, New York, NY, USA, 2002, pp. 721–725.
[23] S. Nabar, J. Walling, and R. Poovendran, "Minimizing energy consumption in body sensor networks via convex optimization," in *Proc. Int. Conf. BSN*, 2010, pp. 62–67.
[24] L. Niu and G. Quan, "Reducing both dynamic and leakage energy consumption for hard real-time systems," in *Proc. Int. Conf. CASES*, New York, NY, USA, 2004, pp. 140–148.
[25] Y. Pan and M. Lin, "Dynamic leakage aware power management with procrastination method," in *Proc. CCECE*, 2009, pp. 247–251.
[26] Y. W. Pan, M. Lin, and L. T. Yang, "Reducing total energy for reliability-aware DVS algorithms," in *Proc. 8th Int. Conf. UIC*, vol. LNCS 6905. 2011, pp. 576–589.
[27] P. Pillai and K. G. Shin, "Real-time dynamic voltage scaling for low-power embedded operating systems," in *Proc. ACM SIGOPS Operat. Syst. Rev.*, 2001, pp. 89–102.
[28] R. Poovendran, "Cyber–physical systems: Close encounters between two parallel worlds," *Proc. IEEE*, vol. 98, no. 8, pp. 1363–1366, Aug. 2010.
[29] M. K. Qiu, E. Sha, M. Liu, M. Lin, S. X. Hua, and L. T. Yang, "Energy minimization with loop fusion and multi-functional-unit scheduling for multidimensional dsp," *J. Parallel Distrib. Comput.*, vol. 68, no. 4, pp. 443–455, 2008.
[30] D. Reed. (2003). *Keeping Leakage Current Under Control* [Online]. Available: http://www.eetimes.com/design/communications-design/4137563/Keeping-leakage-current-under-control
[31] L. Sha, S. Gopalakrishnan, X. Liu, and Q. Wang, "Cyber-physical systems: A new frontier," in *Machine Learning in Cyber Trust*. New York, NY, USA: Springer-Verlag, 2009, pp. 3–13.
[32] L. Shang, L. S. Peh, and N. K. Jha, "Dynamic voltage scaling with links for power optimization of interconnection networks," in *Proc. 9th Int. Symp. HPCA*, 2003, pp. 91–102.
[33] O. S. Unsal, I. Koren, and C. M. Krishna, "Towards energy-aware software-based fault tolerance in real-time systems," in *Proc. ISLPED*, New York, NY, USA, 2002, pp. 124–129.
[34] C. J. Xue, G. Xing, Z. Yuan, Z. Shao, and E. Sha, "Joint sleep scheduling and mode assignment in wireless cyber-physical systems," in *Proc. 29th IEEE ICDCS Workshops*, Jun. 2009, pp. 1–6.
[35] J. Yan and W. Zhang, "WCET analysis for multi-core processors with shared L2 instruction caches," in *Proc. IEEE Real-Time Embedded Technol. Appl. Symp.*, Apr. 2008, pp. 80–89.
[36] L. Yang, M. Lin, and L. T. Yang, "Integrating preemption threshold to fixed priority DVS scheduling algorithms," in *Proc. 15th IEEE Int. Conf. Embedded RTCSA*, Aug. 2009, pp. 165–171.
[37] L. Yang, M. Lin, and L. T. Yang, "Multi-core fixed priority DVS scheduling," *Algorithms and Architectures for Parallel Processing*. New York, NY, USA: Springer-Verlag, 2012, pp. 517–530.
[38] Y. Zhang and K. Chakrabarty, "Energy-aware adaptive checkpointing in embedded real-time systems," in *Proc. Conf. DATE*, Washington, DC, USA, 2003, p. 10918.
[39] B. Zhao, H. Aydin, and D. Zhu, "Reliability-aware dynamic voltage scaling for energy-constrained real-time embedded systems," in *Proc. IEEE ICCD*, Oct. 2008, pp. 633–639.
[40] B. Zhao, H. Aydin, and D. Zhu, "Generalized reliability-oriented energy management for real-time embedded applications," in *Proc. 48th DAC*, San Diego, CA, USA, Jun. 2011, pp. 381–386.
[41] N. G. Zheng, Z. H. Wu, M. Lin, L. T. Yang, and G. Pan, "Infrastructure and reliability analysis of electric networks for E-textiles," *IEEE Trans. Syst., Man, Cybern. C, Appl. Rev.*, vol. 40, no. 1, pp. 36–51, Jan. 2010.
[42] N. G. Zheng, Z. H. Wu, M. Lin, and L. T. Yang, "Enhancing battery efficiency for pervasive health-monitoring systems based on electronic textiles," *IEEE Trans. Inf. Technol. Biomed.*, vol. 14, no. 2, pp. 350–359, Mar. 2010.
[43] X. Zhong and C. Z. Xu, "Energy-aware modeling and scheduling for dynamic voltage scaling with statistical real-time guarantee," *IEEE Trans. Comput.*, vol. 56, no. 3, pp. 358–372, Mar. 2007.
[44] D. Zhu and H. Aydin, "Reliability-aware energy management for periodic real-time tasks," in *Proc. 13th IEEE RTAS*, Washington, DC, USA, Apr. 2007, pp. 225–235.
[45] D. Zhu, R. Melhem, and D. Mosse, "The effects of energy management on reliability in real-time embedded systems," in *Proc. IEEE/ACM ICCAD*, Washington, DC, USA, Nov. 2004, pp. 35–40.

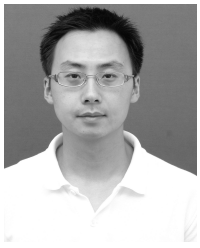
MAN LIN received the B.E. degree in computer science and technology from Tsinghua University, Beijing, China, in 1994, and the Lic. and Ph.D. degrees from the Department of Computer Science and Information, Linkopings University, Linköping, Sweden, in 1997 and 2000, respectively. She is currently an Associate Professor of computer science with St. Francis Xavier University, Canada. Her current research interests include system design and analysis, power aware scheduling, and optimization algorithms. Her research is supported by the National Sciences and Engineering Research Council, Canada, and Canada Foundation for Innovation.



YONGWEN PAN received the B.E. degree from Southeast University, Nanjing, China, 2007, and the master's degree from St. Francis Xavier University, Canada, in 2011. He is currently with QA Consultant, Toronto, ON, Canada. His research was supported by the National Sciences and Engineering Research Council, Canada. His current research interests include real-time system scheduling.

LAURENCE T. YANG is with the Department of Computer Science, St. Francis Xavier University, Canada. He received the B.E. degree in computer science and technology from Tsinghua University, Beijing, China, and the Ph.D. degree in computer science from the University of Victoria, BC, Canada. His current research interests include parallel and distributed computing, embedded, and ubiquitous/pervasive computing. His research is supported by the National Sciences and Engineering Research Council, Canada and Canada Foundation for Innovation.

MINYI GUO received the B.Sc. and M.E. degrees in computer science from Nanjing University, Nanjing, China, and the Ph.D. degree in computer science from the University of Tsukuba, Tsukuba, Japan. He is currently the Zhiyuan Chair Professor and Chair of the Department of Computer Science and Engineering, Shanghai Jiao Tong University (SJTU), Shanghai, China. Before joined SJTU, he was a Professor with the School of Computer Science and Engineering, University of Aizu, Aizu, Japan. He received the National Science Fund for Distinguished Young Scholars from NSFC in 2007, and was supported by 1000 recruitment program of China in 2010. His current research interests include parallel/distributed computing, compiler optimizations, embedded systems, pervasive computing, and cloud computing. He has published more than 250 publications in major journals and international conferences, including the IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS, the IEEE TRANSACTIONS ON COMPUTERS, the *ACM Transactions on Autonomous and Adaptive Systems*, *INFOCOM, IPDPS, ICS, ISCA, HPCA, SC, WWW, PODC*. He received five best paper awards from international conferences. He is on the editorial board of the IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS and the IEEE TRANSACTIONS ON COMPUTERS. He is a member of ACM, IEICE IPSJ, and CCF.



NENGGAN ZHENG received the B.Sc. degree in biomedical engineering and the Ph.D. degree in computer science in 2002 and 2009, respectively, both from Zhejiang University, Hangzhou, China. Currently, he is an Associate Professor of computer science with the Qiushi Academy for Advanced Studies, Zhejiang University. His current research interests include brain machine interface and real time systems.