Thermal Aware Task Scheduling for Enhanced Cyber-Physical Systems Sustainability

Shikang Xu, Israel Koren and C. M. Krishna

Abstract—Cyber-Physical Systems (CPS) are increasingly used in a variety of transportation, healthcare, electricity grid, and other applications. Thermal stress is often a major concern for processors embedded in such systems. High operating temperatures can dramatically shorten processor life. This in turn can require provisioning of significant amounts of additional computational hardware to withstand more frequent failures, with obvious implications for sustainability. This paper describes a novel approach to reduce thermally-induced damage in CPS processors by targeting Dynamic Voltage and Frequency Scaling (DVFS) to high-activity task phases. That is, by preferentially slowing down high-activity task phases, significant additional savings in energy and thermal stress can be attained for a given amount of computational slowdown; this approach is shown to be superior to conventional methods that use DVFS without regard to activity levels. Also, task reassignment across cores is driven by estimates of current core reliability, which is superior to the usual approach of simply using either current temperature or temperature history. Our approach leads to a significant reliability improvement (around 20%) over baseline DVFS techniques.

Index Terms—Sustainability, DVFS, Cyber-physical Systems, Real-time Systems, Embedded Systems, Reliability improvement, Lifetime Extension

1 Introduction

Semiconductor failure rates rise rapidly with device temperature. This is of special concern in Cyber-Physical Systems (CPS) operating in harsh environments and requiring high reliability. Reliability requirements coupled with thermally-accelerated hardware failures can significantly increase the amount of hardware that has to be provisioned, with obvious implications for sustainability. This has motivated efforts to reduce thermal stress while still meeting the often-stringent deadline requirements of critical, real-time CPS workloads.

There have been many studies on Dynamic Thermal Management (DTM). Current DTM techniques include (*a*) Dynamic Voltage and Frequency Scaling (DVFS) [1], (*b*) migrating tasks from hotter to cooler cores [2], and, (*c*) core throttling for a certain cooling-off period when a temperature threshold is exceeded [3].

In this paper, we propose two DTM techniques for CPSs containing multi-core processors and workloads with hard deadline timing constraints. The two techniques are designed to improve the reliability of processors by evenly distributing thermal stress temporally (on each core) and spatially (among cores). The distinguishing feature of this paper is a recognition that power consumption varies considerably not only in the execution of different tasks but also over the execution of the same task. This is usually driven by varying levels of Instruction Level Parallelism (ILP) [4]. We exploit such variability to improve reliability and reduce the energy consumption of individual cores. We also reassign tasks between cores where appropriate, based on an estimate of their effective aging rates in order to balance the thermal related wearout among cores.

Our simulation results indicate that the two proposed techniques (intra- and inter-core) can achieve more than 20% reliability improvement compared to using a previously proposed DVFS algorithm for a target system reliability of 1 - 1e-6 (“six nines”).

The rest of the paper is organized as follows. Section 2 discusses some previous work in thermal-aware computing. Section 3 provides basic background in VLSI circuit reliability and CPS. Section 4 presents a theoretical model and details of the proposed thermal management techniques. Section 5 provides numerical results. The paper concludes with a brief discussion in Section 6.

2 Related Previous Work

Two techniques have been commonly used in DTM and Dynamic Reliability Management (DRM) of multi-core systems, namely, DVFS and thermal-aware task allocation/task reassignment.

Various DVFS-based approaches have been proposed to keep temperature below a prescribed limit [3], [5], [6], [7], [8]. Solid-state failure mechanisms which are accelerated by heating are explicitly taken into account in a few studies, e.g., electromigration in [9], [10] and oxide breakdown in [11]. Static task assignment to cores in a multicore system is evaluated in [12] while offline (static) mapping of tasks in a task graph is studied in [13].

Thermal-aware task allocation and task reassignment have been used to better distribute thermal stress among cores. This technique has been used for DTM of datacenters and computing clusters [14], [15], [16] and has also been studied for multi-core processors. For instance, [2], [17] migrate tasks to reduce inter-core temperature differences. In [18], various approaches to assigning tasks to processors are compared in order to minimize the instantaneous temperature of a multi-core processor. Task reassignment based on inputs from wearout (degradation) sensors is studied in [19]; however, such sensors are not yet widely available on contemporary processors. The issue of whether some cores should initially be kept unused (for later use) rather than

having all cores active is considered in [20], [21]. Evolving the task migration policy (in a non-real-time application) using reinforcement learning is reported in [22]. It was not until recently that using reliability as a criterion for task mapping in many-core system began to be studied [23], [24].

Some DTM work has focused on systems with workloads that have hard deadlines [1], [25], [26], [27] [28]; the goal is usually either to satisfy a given temperature constraint or to reduce energy consumption.

The underlying assumption in contemporary work is that the power consumption is fairly steady over the lifetime of a task. By contrast, in the algorithms proposed here, we exploit the frequent considerable variation in power consumption during the execution of a single task. Such a variation is caused by dynamically varying levels of Instruction Level Parallelism (ILP) within the executed code. The central idea of the paper is that since high-ILP segments consume more energy than low-ILP segments, using a given amount of slack to preferentially slow down high-ILP segments provides greater energy savings.

3 Background

3.1 VLSI Circuit Reliability

The reliability of VLSI circuits is affected by multiple failure mechanisms. Modeling these has been an active research topic for decades. Oxide breakdown and electromigration are reported to be dominant permanent failure mechanisms of VLSI circuits as CMOS technology scales [29].

Oxide (or dielectric) breakdown is caused by the formation of a low resistance path in an oxide insulating area and is a major contributor to circuit failure. The Mean-Time-To-Failure (MTTF) due to oxide breakdown is given by [30]:

$$MTTF_{bd} = A_{bd} \times V^{-(a-bT)} \times e^{\frac{X+(Y/T)+ZT}{kT}} \quad (1)$$

where V is the voltage applied to the gate oxide, T is the absolute temperature in Kelvin (K), k is Boltzmann's constant and A_{bd} is a scale factor. Typical values cited for the other parameters are [30]: $a = 78$, $b = -0.0081$, $X = 0.759$ eV, $Y = -66.8$ $eV \times K$ and $Z = -8.37E-4eV/K$.

Electromigration (EM) is another cause of circuit failure [31]; Black's model is widely used [32]:

$$MTTF_{em} = A_{em} \times J^{-n} e^{\frac{E_a}{kT}} \quad (2)$$

where A_{em} is a scale factor, J is the current density, E_a is activation energy and n is a material based constant. For copper, these values are J=1e6 A/cm [33], $E_a = 0.9$ eV and $n = 1.1$ [34].

The failure of a system is a random process and the reliability of a system at time t is the probability that the system is functional throughout the time interval $[0, t]$. The probability of a device failure occurrence during $[0, t]$ is often modeled by the Weibull distribution:

$$F(t) = 1 - R(t) = 1 - e^{-(t/\eta)^{\beta}} \quad (3)$$

where $F(t)$ is the failure occurrence probability, $R(t)$ is the reliability function, β is the Weibull slope parameter (a typical value is β=2 [35]), and η is a scale parameter satisfying $\eta = MTTF/\Gamma(1 + 1/\beta)$ [11].

The above reliability expressions model the reliability of circuits under constant temperature. In practice, however, the working environment of a processor is varying and so is its temperature. In this paper, the approach of [11] is adopted to calculate the reliability in a dynamic thermal environment. Time is divided into k time frames, [0,Δ),[Δ, 2Δ),..., [(k-1)Δ, $k\Delta$] and in each time frame, the temperature and voltage are assumed to be constant. The resulting reliability of a functional block, over k frames, denoted by $R_{blk}(t)$ is given by

$$R_{blk}(t) = R_{blk}(k\Delta) = \prod_{i=1}^{i=k} [1 - (R_{blk}((i-1)\Delta) - R_{blk}(i\Delta))] \quad (4)$$

where $R_{blk}(i\Delta)$ is based on the temperature of the block at $i\Delta$; with multiple failure mechanisms taken into account, $R_{blk}(i\Delta)$ is equal to the product of the reliability derived from each failure mechanism using the temperature at the ith time interval.

The reliability of a core at time t is the product of the reliabilities of all the functional blocks of the core at time t. The reliability of the system is the product of the reliabilities of all cores in the system.

3.2 System Model

Workload: The workload consists largely of periodic tasks with deadlines. Added to the task mix may be aperiodic tasks: there exist standard techniques for scheduling them within a periodic framework [36]. The basic goal of scheduling real-time CPS tasks is to meet the deadline (i.e., finishing the task execution before the deadline). A task i is characterized by its Worst Case Execution Time (WCET) w_i when executing at a given reference frequency and its period p_i; each period p_i, an iteration (or job) of that task is released for execution. Real-time CPS workloads are associated with task deadlines; typically, the relative deadline (the interval from the arrival of an iteration of task i to the time it should be finished) of a periodic task is set equal to its period, p_i; the worst-case utilization of task i is thus $u_i = w_i/p_i$.

The actual execution time of a task iteration is a random variable: its value is unknown until its execution is complete. However, its statistics (e.g., Cumulative Distribution Function (CDF) and the Expected Execution time (EET)) can be determined in advance by profiling.

In this paper, our algorithms assume that the Earliest Deadline First (EDF) algorithm is used for task scheduling. EDF was selected due to its widespread use by the CPS community. However, we should stress that the algorithms proposed in this paper can be used with any other real-time scheduling algorithm.

Computational Platform: We assume a conventional computational platform, consisting of multiple processing cores sharing main memory and lower-level caches, backed up by flash memory. Memory density has expanded greatly in recent years; there is enough memory so that each core has rapid access to the text segment of any task that it may be assigned to execute. Task inputs (deriving from other task outputs or from sensors) are placed in predesignated locations in memory.

All cores are assumed to have two frequency and voltage levels; a Dynamic Voltage and Frequency Scaling (DVFS) approach is used to select the appropriate level. Extending the work to more than two levels is quite simple; however,

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSUSC.2019.2958298, IEEE Transactions on Sustainable Computing

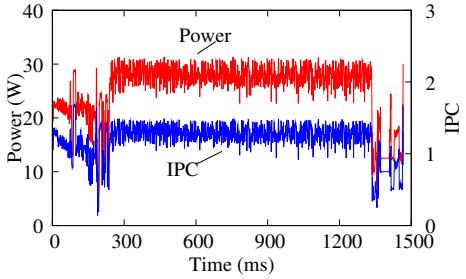


Fig. 1. Power and IPC variation for the benchmark Typeset from Mibench [37]

given that the maximum supply voltage to a chip keeps dropping with advances in technology, the scope for a larger number of voltage levels is shrinking.

Workload Assignment: There is no migration of task iterations: once an iteration starts executing on a given core, it stays there until its end. However, there is nothing preventing the $n+1'$st iteration of a task from being activated on a different core to that of the n'th iteration. Due to the shared memory elements, this entails no meaningful overhead. Selection of the appropriate node on which to activate a task iteration is a key function of our heuristics.

4 Thermal Management Techniques

4.1 Objective

The objective of our algorithm is to maximize the reliability of CPS platforms, under the constraint that the hard deadlines of the computational workload continue to be met. This is done by reducing the rate at which thermally accelerated processor aging takes place. To do so, 1) we focus voltage/scaling on those segments of the execution that exhibit the greatest instruction-level parallelism, and are thus the most power-hungry; and 2) we balance the thermal stress on different cores by adjusting the tasks activated on each core (i.e., task reassignment).

4.2 Intra-core: Workload-Aware Voltage and Frequency Scaling Algorithm (WA-DVFS)

Like all DVFS algorithms for real-time systems, we exploit the slack that is generated when the workload utilization is less than 1. *Static slack* is the slack that exists even if all iterations run to their WCETs; it can be calculated before runtime. *Dynamic slack* is generated when (as is usually the case) iterations complete before their WCETs. Dynamic slack can only be determined at runtime, upon the completion of a task iteration. Slack generated by a task iteration expires at the deadline of that iteration.

Key to our approach is the observation that CPS workloads often exhibit significant time-varying levels of instruction-level parallelism (ILP) *within individual task iterations*. ILP correlates positively with power consumption. An example of the power and IPC (Instructions retired Per Cycle) variation that exists in an actual CPS workload is shown in Fig. 1.

The main idea behind our WA-DVFS algorithm is to use the available slack to preferentially slow that part of the workload with higher power consumption: this allows greater energy savings and thermal benefits per unit of consumed slack. The ILP can be easily monitored using performance counters (measuring the number of instruction retired per cycle, i.e., IPC) which are part of most modern processors.

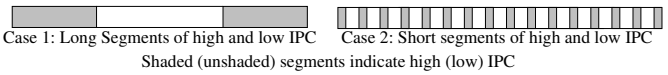


Fig. 2. Illustrating Extreme Cases 1 and 2

A user-defined threshold divides the IPC region into high and low levels. This threshold may be determined by profiling the workload in advance (note that the workload of a CPS is known prior to its deployment). Another method to determine the threshold is to allow the system to learn an appropriate value based on operational data.

4.2.1 An Approximate Model

To obtain an idea of how much improvement one might expect from the proposed approach, we use a simple and approximate model of a single task. For purposes of thermal modeling, we treat the entire processor as a single node. Heat flows are modeled by means of thermal equivalent circuits [38]. Thermal resistance expresses the amount of heat that flows across an interface given the temperature differential across that interface; thermal capacitance represents the amount of heat required to raise the temperature of the node by one degree. The thermal inertia of the chip is expressed by means of the thermal time-constant, which is the product of the chip thermal resistance and capacitance.

Exact thermal models are too complicated to analyze and require simulation. However, two extreme cases are tractable and can shed light on how much lifetime improvement we can expect with targeted voltage/frequency scaling. In Case 1 (Fig. 2), the high- and low-parallelism segments are each of duration much longer than the thermal time constant of the chip. In Case 2, we assume that these segments are of duration much shorter than the thermal time constant. In all cases, the segments are assumed to be much longer than the time required to carry out voltage and frequency scaling; since the scaling time is small [39], this is not a limiting assumption in practice. Fig. 2 illustrates both cases. Due to the length of these segments, we can use the steady-state temperature; in Case 1, each segment is long enough so that steady state temperature is reached for most of its duration; in Case 2, due to the fine-grained interleaving of low- and high-parallelism segments, the temperature holds relatively steady through the lifetime of the task. The detailed analysis for both cases are presented in the Appendix.

Denote by $A_{eff}(t)$ the effective age of the node at time t. The effective age takes into account the accelerated aging caused by elevated temperatures [38]; it is equal to the chronological age when the node is at room temperature (300K) and rises exponentially with an increase in temperature. In Fig. 3, we plot the improvement in system reliability, where reliability at time t is calculated as $R(t) = exp(-(A_{eff}(t)/\eta)^{\beta})$. The reliability improvement over a baseline algorithm is defined as

$$\text{Improvement} = 1 - \frac{1 - R(t_{ref})_{WA-DVFS}}{1 - R(t_{ref})_{Baseline}} \quad (5)$$

where t_{ref} is the time when $R(t_{ref})_{Baseline}$ reaches the lower bound of the system reliability requirement (e.g., 1 - 1e-6). We use as baseline the widely-cited DVFS algorithm proposed in [40].

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSUSC.2019.2958298, IEEE Transactions on Sustainable Computing

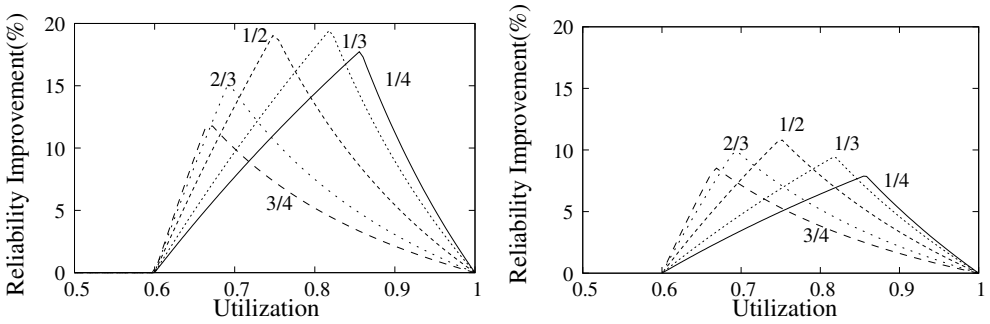


Fig. 3. Processor Reliability Improvement with WA-DVFS

Fig. 3 indicates the range of improvement possible; finely interleaved segments yield the lowest, while very long segments provide the greatest improvement in processor reliability. Note that benefits from this approach accrue when the utilization is moderate. For very low utilizations, the entire workload can be slowed down; for utilizations close to one, there is not much scope to slow anything down.

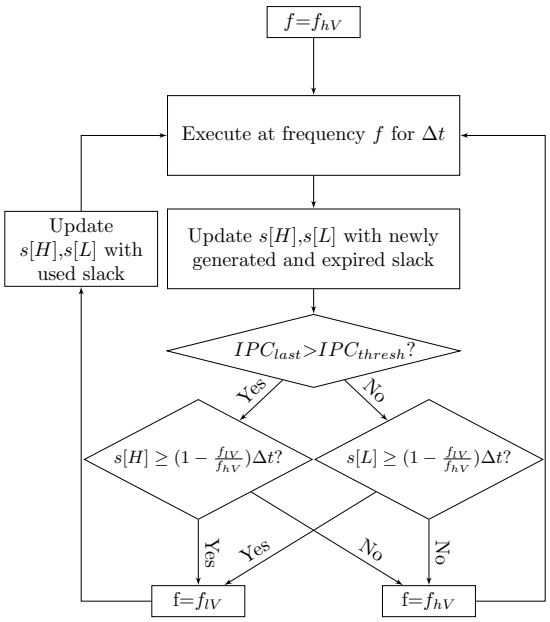


Fig. 4. Flow Chart of WA-DVFS

4.2.2 WA-DVFS Algorithm Description

DVFS algorithms operate by exploiting static and dynamic slack to run at lower voltage/frequency levels. Other DVFS algorithms use up available slack at the first available opportunity. However, this can result in slack being wasted in slowing down low-ILP segments when it can more profitably be used to slow down later high-ILP segments. WA-DVFS characterizes task segments by the two-tuple (x, y), where $x = PE$ denotes the segment of the task prior to its Expected Execution Time and $x = BE$ that beyond; and $y = L, H$ denote Low- and High-execution ILP. There are four segments specified by all combinations of (x, y). Given a certain amount of available slack, we assign it to each such segment to prevent one segment encroaching on the quota of another (details are provided below).

TABLE 1
Notation I

f	Processor frequency setting
t_{sys}	System time, initialized at 0
f_{hV}	Processor high frequency level
f_{lV}	Processor low frequency level
σ	f_{hV}/f_{lV}
h	Worst case execution time of high-IPC phases of all tasks at frequency f_{hV}
l	Worst case execution time of low-IPC phases of all tasks measured at frequency f_{hV}
p_i	Period of task i
w_i	WCET of task i
a_i	Actual execution time of task i
u_i	Utilization of task i ($u_i = \frac{w_i}{p_i}$)
n	Total number of tasks
LCM	Least common multiple of all p_is
s_{static}	Static slack
$s_{dynamic_{new}}$	Newly-obtained dynamic slack
$s_{req}[(x,y)]$,$s_{rsv}[(x,y)]$	Slack needed and reserved to slow down IPC phase $(x, y) \in$[(PE,H), (PE,L), (BE,H), (BE,L)]
s[H]	Currently available slack for high IPC phase
s[L]	Currently available slack for low IPC phase
$s_t[i]$	Available slack amount ($s_t[i].slack$) and expiry time ($s_t[i].expire$) of dynamic slack from task i
Δt	Time step
Δ	Reliability update interval
IPC_{thresh}	IPC threshold used to separate high- and low-IPC
IPC_{last}	IPC in the previous time step
M_{hI} (M_{lI})	Statistical mean execution time of high(low)-IPC as a fraction of the high(low)-IPC part in the worst case
s_{expire}	Total expired slack
U_i^{dyn}	Effective utilization of task i
$\gamma_{m,n}$	Accumulated reliability difference between core m and n
T_{ss_i}	Steady-state temperature of task i

WA-DVFS makes the DVFS decision every pre-defined time step (Δt, a configurable parameter which does not have to be the same as the reliability update interval Δ mentioned in Section 3.1). WA-DVFS does not miss deadlines since it uses the same static and dynamic slack usage policy as the DVFS algorithm proposed in [40].

A high-level view of WA-DVFS is shown in Fig. 4. The pseudo-code of the main algorithm is presented in Algorithms 1 to 7. The notations used in the pseudo-codes and the flow chart can be found in Table 1.

At the beginning of every Δt, the available slack will be updated with the newly generated and expired slack and suitably allocated to the various task segments. Then the workload is checked to see which IPC phase it is in,

based on the average IPC in the previous time step. If the corresponding allocated slack is enough to slow down the workload execution, the execution in this step will be at low frequency, otherwise at high frequency. Note that frequency changes only happen at the start of each time-step.

An initialization step, to record the available static slack, is carried out every LCM (Least Common Multiple) of the task periods, as shown in Algorithm 2. The total available static slack at the beginning is calculated based on the WCETs (denoted by s_{static}). The slack values needed to slow down the high- and low-IPC portions of the PE (denoted by $s_{req}[(PE, H)]$ and $s_{req}[(PE, L)]$) are then calculated (lines 3 and 4 of Algorithm 2).

Algorithm 1 Workload-Aware Dynamic Voltage/Frequency Scaling (WA-DVFS)

WA-DVFS:
At every time step
1 IF t_{sys} mod LCM =0
2 **Initialize( )** *//see Algorithm 2*
ENDIF
3 IF $t_{sys} = 0$
4 $f = f_{hV}$
5 ELSE
6 set IPC_{last} equal to the average IPC in the previous time step
7 **HandleNewSlack(t_{sys})** *//see Algorithm 4*
0 **HandleExpiredSlack(t_{sys})** *//see Algorithm 5*
9 IF $IPC_{last} \geq IPC_{thresh}$
10 IF $s[H] \geq (\sigma - 1) \cdot \Delta t$
11 $f = f_{lV}$
12 $s[H] = s[H] - (\sigma - 1) \cdot \Delta t$
13 **UseSlack(**$(\sigma - 1) \cdot \Delta T$**)** *//see Algorithm 3*
14 ELSE
15 $f = f_{hV}$
ENDIF
16 ELSE
17 IF $s[L] \geq (\sigma - 1) \cdot \Delta t$
18 $f = f_{lV}$
19 $s[L] = s[L] - (\sigma - 1) \cdot \Delta t$
20 **UseSlack(**$(\sigma - 1) \cdot \Delta t$**)**
21 ELSE
22 $f = f_{hV}$
ENDIF
ENDIF
ENDIF
23 $t_{sys} = t_{sys} + \Delta t$

Then, the slack needed to slow down the high-IPC portion of the WCET beyond the high-IPC portion of the EET (denoted by $s_{req}[(BE, H)]$) is calculated in line 5 of Algorithm 2. The rest of the code consists of allocating slack to the four parts of the workload mentioned above (line 6, details in Algorithm 6) and reset the dynamic slack associated with each task (line 7-10). In Algorithm 6, the available slack (static or dynamic) will be reserved for phases in the following sequence: $(PE, H), (PE, L), (BE, H), (BE, L)$. The variable $s_{rsv}[(x, y)]$, where $(x, y) \in [(PE, H), (PE, L), (BE, H), (BE, L)]$, is used to record the slack reserved for each phase. The slack will be reserved for (PE, H) first and added to the amount of slack in $s_{rsv}[(PE, H)]$. If the value $s_{rsv}[(PE, H)]$ reaches $s_{req}[(PE, H)]$ and there is still slack available, the slack will be reserved for (PE, L) and then to phases (BE, H) and (BE, L) as long as there is slack available. Slack reservation stops once the slack has been exhausted. Slack reserved for the high or low IPC portion ($s[H]$ or $s[L]$) will be updated during the reservation for each phase (lines 5 and 9 in Algorithm 6, y is the second term in the tuple, i.e. H or L). The value of $s_{rsv}[(x, y)]$ will increase when new slack is available and reserved and decrease (Algorithm 7) when slack expires. The value of $s[H]$ and $s[L]$ will increase when new slack is reserved and decrease when the processor is running in low frequency and slack expires.

Algorithm 2 Initialization (WA-DVFS)

Initialize( )
1 $s[H]$=0, $s[L]$=0
2 $s_{static} = (\frac{1}{\sum_{i=1}^{n} w_i/p_i} - 1) \sum_{i=1}^{n} w_i$
3 $s_{req}[PE, H] = h \cdot M_{hI} \cdot (\sigma - 1)$
4 $s_{req}[PE, L] = l \cdot M_{lI} \cdot (\sigma - 1)$
5 $s_{req}[BE, L] = h \cdot (1 - M_{hI}) \cdot (\sigma - 1)$
6 $ReserveSlack(s_{static})$
7 FOR i in 1 to n
8 $s_t[i].slack = 0$
9 $s_t[i].expire = +\infty$
ENDFOR
10 $s_{dynamic_{new}} = 0$
11 $IPC_{last} = +\infty$

Algorithm 3 Slack Consumption

UseSlack(s)
1 FOR ALL $s_t[i]$, in $s_t[i].expire$ in ascending order
2 IF $s_t[i].slack \geq s$
3 $s_t[i].slack- = s$
4 **break**
5 ELSE
6 $s- = s_t[i].slack$
7 $s_t[i].slack = 0$
ENDIF
ENDFORALL

At the beginning of every time step, after the update of the IPC value, the available slacks are updated using functions `HandleNewSlack(t)` and `HandleExpiredSlack(t)`.

The function `HandleNewSlack(t)` assigns newly available dynamic slack released from tasks which finish prior to their WCET in the previous time step. The amount of slack and its expiry time for task i will be saved in $s_t[i].slack$ and $s_t[i].expire$, respectively; the amount is the difference between the worst-case and actual execution time and the expiry time is the deadline of that task iteration. The function `UseSlack(s)` handles the use of the available slack. Note that dynamic slack is always associated with a deadline; slack is used in an earliest-deadline-first order. The total amount of newly available dynamic slack is accumulated in $s_{dynamic_{new}}$. Assignment of the newly available dynamic slack is carried out using the same approach as used in the static slack assignment.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSUSC.2019.2958298, IEEE Transactions on Sustainable Computing

Algorithm 4 Injecting New Slack

HandleNewSlack(t)
1 FOR ALL task i finishing during $t - \Delta t$
2 $s_t[i].slack$=$w_i - a_i$
3 $s_{dynamic_{new}} += s_t[i].slack$
4 $s_t[i].expire$=deadline of the current iteration of task i
ENDFORALL
5 $ReserveSlack(s_{dynamic_{new}})$

When a deadline is reached, the slack associated with that task iteration expires. To deal with the expired slack, the function `HandleExpiredSlack(t)` removes the slack in the reverse order in which it was assigned (Algorithm 7); slack assigned to the low-IPC portion of the WCET will be removed first.

Algorithm 5 Slack Expiry

HandleExpiredSlack(t)
1 $s_{expire} = 0$
2 FOR ALL $s_t[i].expire \leq t$
3 $s_{expire} += S[i].slack$
4 $s_t[i].slack = 0$
5 $s_t[i].expire = +\infty$
ENDFORALL
4 RemoveSlack(s_{expire})

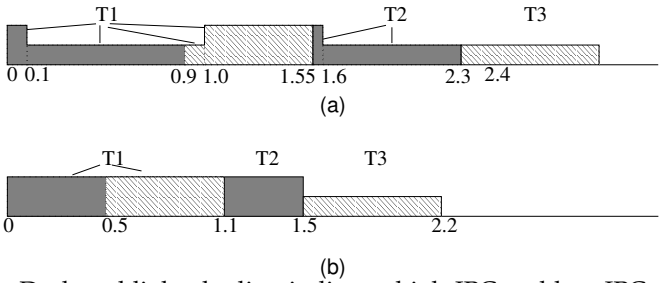
Algorithm 6 Reserve Slack

ReserveSlack(s)
1 FOR (x, y) in [(PE,H), (PE,L), (BE,H), (BE,L)]
2 IF $s_{rsv}[(x, y)] < s_{req}[(x, y)]$
3 If $s > s_{req}[(x, y)] - s_{rsv}[(x, y)]$
4 $s- = s_{req}[(x, y)] - s_{rsv}[(x, y)]$
5 $s[y]+ = s_{req}[(x, y)] - s_{rsv}[(x, y)]$
6 $s_{rsv}[(x, y)] = s_{req}[(x, y)]$
7 ELSE
8 $s_{rsv}[(x, y)]+ = s$
9 $s[y]+ = s$
10 break
ENDIF
ENDIF
ENDFOR

Algorithm 7 Remove Slack

RemoveSlack(s)
1 FOR (x, y) in [(BE,L),(BE,H),(PE,L),(PE,H)]
2 IF $s_{rsv}[(x, y)] < s$
3 $s- = s_{rsv}[(x, y)]$
4 $s[y]- = s_{rsv}[(x, y)]$
5 $s_{rsv}[(x, y)] = 0$
6 ELSE
7 $s_{rsv}[(x, y)]- = s$
8 $s[y]- = s$
9 break
ENDIF
ENDFOR

Example: We compare the result of WA-DVFS with the widely-cited cycle-conserving DVFS algorithm of Pillai and Shin in [40] (denoted by P-DVFS), shown in Algorithm 8. There are three tasks with parameters $(w_1 = 2, p_1 = 4)$, $(w_2 = 1, p_2 = 5)$, and $(w_3 = 0.5, p_3 = 5)$, respectively. The available frequencies are $f_{hV} = 1, f_{\ell V} = 0.5$, and $\Delta t = 0.1$.



Dark and light shading indicates high-IPC and low-IPC segments, respectively.
Tall (Short) boxes indicate execution at frequency $f_{hV}(f_{lV})$.

Fig. 5. Task schedule for the example according to (a) WA-DVFS (b) P-DVFS

From offline profiling, it is known that T1 contains a high IPC phase and a low IPC phase, T2 only contains a high IPC phase and T3 only a low IPC phase. Consider the first few moments of that execution. Assume that at f_{hV}, the actual run times of the first iteration (i.e., job) of task T_1 are 0.4 for the high-IPC and 0.6 for the low-IPC segments, respectively. The actual run times of the first iteration of T_2, T_3 at f_{hV} are $0.4, 0.35$, respectively.

Algorithm 8 P-DVFS

select_frequency():
1 IF $\sum_{i=1}^{n} U_i^{dyn} \leq \frac{1}{\sigma}$
2 set frequency f_{lV}
3 ELSE
4 set frequency f_{hV}
ENDIF
upon_task_release(Task i):
5 set $U_i^{dyn} = w_i/p_i$
6 select_frequency()
upon_task_completion(Task i):
7 set $U_i^{dyn} = a_i/p_i$
8 select_frequency()

Fig. 5 provides a comparison between the baseline P-DVFS algorithm and WA-DVFS. The dark shaded boxes are high IPC phases while the light shaded boxes are low IPC phases; taller boxes indicate a frequency of f_{hV} while shorter boxes indicate $f_{\ell V}$.

WA-DVFS preferentially allocates slack so that more of the high-IPC segments can be run at low frequency. By contrast, P-DVFS takes no account at all of the task IPC level and keeps running at f_{hV} until early task completions drop the total utilization temporarily below $1/\sigma$.

4.3 Inter-core: Reliability-Aware Online Task reassignment

As shown in [28], reliability is maximized when the cores are thermally balanced. Thus, a reliability aware scheduler should dynamically attempt to equalize the reliability of the cores by reassigning tasks as needed.

The Largest Task First (LTF) scheme was shown in [18] to reduce the temperature difference among cores more efficiently than other offline partitioning algorithms. Our initial task assignment, therefore, follows the LTF algorithm.

As a workload executes, the estimated reliability of each core will be updated periodically with a period of Δ. Using Equation 4 to compute the current reliability, we only

need temperature data for the last interval, i.e., the one following the previous reliability update. The workloads of the cores will not be adjusted when there is a small difference in reliability, because such a small difference may well be reversed by the time the next job arrives. Instead, the reliability decreasing rate of each core m (the reliability difference between two consecutive updates), $\delta_m(k\Delta) = R_m(k\Delta) - R_m((k-1)\Delta)$, is monitored. The proposed algorithm updates the accumulated difference in the reliability decreasing rate between each pair (m, n) of cores until it exceeds a given threshold and only then does a workload adjustment happen on these two cores. The accumulated difference is denoted by γ_{mn} and is defined as:

$$\gamma_{mn}(t) = \sum_{k=0}^{\lceil \frac{t-t_l(m,n,t)}{\Delta} \rceil} (\delta_m(t_l(m,n,t)+k\Delta) - \delta_n(t_l(m,n,t)+k\Delta)) \quad (6)$$

where $t_l(m, n, t)$ is the latest time smaller than t when the workload on either core m or core n was adjusted.

The pseudo-code of the online load adjustment is shown in Algorithm 9. The initial value of each γ_{mn} is 0. If γ_{mn} is positive, the reliability of core m decreases faster than that of core n and vice versa. If so, the workload adjustment algorithm will be invoked (line 5 in Algorithm 9). Workload adjustment includes reassigning/swapping tasks between the cores for which γ_{mn} is being checked. If the load adjustment (task reassignment and swapping that will be introduced below) fails due to high load (large utilization) on the cores, the pair of cores with the second largest absolute value of γ_{mn} is chosen. Then, a check is made to see if this quantity exceeds the threshold. If so, task reassignment or swapping is performed. If not, the execution continues with the current task assignment. If the task reassignment or swapping also fails for the pair of cores with the second largest absolute value of γ_{mn}, the pair with third, or fourth largest value and so on is considered. This process continues until we run out of core pairs, or load adjustment (task reassignment or swapping) is successfully performed on one pair of cores or the pair of cores chosen has an $abs(\gamma_{mn})$ smaller than the threshold. Every time the load adjustment is invoked, only the load on one pair of cores will be adjusted. Then, all the γ_{mn}s that are related to the pair of load adjusted cores are reset to 0 (for example, when workload on core j and core k is adjusted, the γ_{mn}s that involve core j and core k are set to 0, as is shown in line 7 of Algorithm 9).

The first load adjustment that is attempted is reassigning a task from the less reliable core to the more reliable core (line 6 in Algorithm 9). A successful reassignment indicates that the next iteration of the reassigned task will be executed on the more reliable core. In order to guarantee that all tasks meet their deadlines, the utilization of the target core must be smaller than or equal to 1 after reassignment.

When the utilization of each core is close to 1, it is possible that no task can be reassigned from one core to another. When this is encountered, a task swapping process will be attempted (line 10 in Algorithm 9). In the task swapping process, each core has a task list in which tasks are sorted according to their $u_i \times T_{ss_i}$ value (T_{ss_i} is the steady-state temperature of task i). This product is chosen because T_{ss_i} indicates the thermal stress caused by executing task i and u_i is the fraction of time the task executes on the core. The task swap process is described below with an example. Assume the tasks on core 1 and core 2 need to be swapped and core 1 is the less reliable core. Tasks on core 1 are τ_{11}, τ_{12},..., τ_{1n} in $u_i \times T_{ss_i}$ descending order. Similarly tasks on core 2 are τ_{21}, τ_{22},..., τ_{2n}.

Algorithm 9 Pseudo Code for Online Load Adjustment

Online_load_adjustment()

```
1  Update all γ_mn;
2  Define P as the set containing all core pairs
3  (j,k)=the core pair in P with maximum abs(γ_jk);
4  WHILE(P is not empty)
5    IF(abs(γ_{j,k})≥ Threshold)
     // assume core j is less reliable
6      IF(reassign_success(j,k))
7        γ_{j,m} = γ_{k,m} = γ_{m,j} = γ_{m,k} = 0 for all m;
8        break;
9      ELSE
10       IF(swap_success(j,k))
11         γ_{j,m} = γ_{k,m} = γ_{m,j} = γ_{m,k} = 0 for all m;
12         break;
13       ELSE
14         remove (j,k) from P;
15         (j,k)=the core pair in P with maximum
   abs(γ_mn);
         ENDIF
       ENDIF
16   ELSE
17     continue execution with current assignment;
18     break;
     ENDIF
   ENDWHILE
```

Firstly, the swapping algorithm will try to swap τ_{11} and τ_{2n}. If the utilization on either core is greater than 1 after the swap, with the initial assumption that tasks with larger product tend to have larger utilization, the algorithm will try to swap τ_{11} with two tasks, τ_{2n} and τ_{2n-1}. If the swap still fails due to utilization larger than 1 on one of the cores, the swapping between τ_{11} and three tasks,τ_{2n},τ_{2n-1} and τ_{2n-2}, will be tried. If the swapping continues to fail, tasks on core 2 will continue to be added to swapping until the newly added task on core 2 has a larger $u_i \times T_{ss_i}$ than τ_{11}. If swapping τ_{11} fails, then τ_{12} will be tried to swap with tasks on core 2. Tasks on core 1 will be chosen in order to swap with tasks on core 2 if previous swap fails until the chosen core 1 task has a smaller $u_i \times T_{ss_i}$ than τ_{2n} . It is also possible that some tasks have large utilization, low steady-state temperature and a small $u_i \times T_{ss_i}$ value. If such a task resides on the more reliable core, it may be chosen as the first task to be swapped according to the previous process and lead to failure due to worst-case utilization being greater than 1. In order to avoid this situation, if the above swapping process fails, the proposed algorithm will try to swap one task on the more reliable core with one or multiple task(s) on the less reliable core, starting from τ_{2n}, in a similar way to the swapping process above.

5 Experimental Results

5.1 Simulation Configuration

The simulated system has two homogeneous cores sharing a cache. The cores can run at two frequency levels, a high frequency of 2.0 GHz and a low frequency of 1.2 GHz. The power files of the workloads were obtained using Gem5 [41] and McPAT [42]. The power files using different DRM algorithms are sent to TILTS [43] to calculate the temperature trace. Temperature is then used to calculate the reliability of the processor during the execution of the workloads. Benchmarks from SPEC06 are used (Even though SPEC06 is not a CPS benchmark, programs in the suite utilize different function blocks of the processor and give different levels of IPC phase within the same task. This is similar to tasks on today's CPS with multi-core processors where tasks have very different characteristics). Each benchmark has a given period and acts as an independent CPS task. The execution time assigned to each iteration of a task is picked randomly over a given interval (according to a normal distribution with mean equal to its average execution time). The aim is to choose execution time and period for each task so that a given worst-case and average processor utilization that are needed to test the algorithms are maintained. The reliability improvement shown in the figures in this section are calculated using Equation 5, i.e., we compare the reliabilities using different thermal aware techniques at the time instance at which the reliability of the system using the baseline algorithm is 1-1e-6.

5.2 Performance of WA-DVFS

We compare our WA-DVFS to P-DVFS [40] in terms of chip reliability, using both synthetic and standard benchmark (SPEC06) workloads. The synthetic workload is based on power traces generated using the observed approximate linear dependence of power consumption on IPC. This function is obtained using linear regression over SPEC06 benchmarks. In the low (high) IPC phase(s) of the synthetic power traces, the IPC value has a normal distribution around a specified low (high) mean IPC value. The synthetic workload is used to explore the impact on WA-DVFS's performance of the characteristics of the workload (e.g., the IPC difference between high and low IPC phases, the length of each IPC phase and the accuracy of workload estimation).

The reliability improvement using the proposed algorithm running synthetic workloads is shown in Fig. 6. Fig. 6a-6e, show the reliability improvement of WA-DVFS when the workload execution time is accurately estimated (EET=WCET).

Fig. 6a-6c show the reliability improvement of WA-DVFS when there is only a single low IPC phase and a single high IPC phase in the workload. Each curve in these figures shows the impact of a different length of the high IPC phase (denoted by "H=") as a fraction of the WCET.

Fig. 6a shows the reliability improvement of WA-DVFS and P-DVFS over the case where DVFS is not applied. Fig. 6b and 6c show the benefit of WA-DVFS over P-DVFS when the IPC difference between the low and high IPC phases is large (2.0) or small (1.0).

In these figures, when the worst-case utilization is close to 1, little or no slack is available and the two algorithms behave similarly as there is limited opportunity for voltage scaling. At the other extreme, for low utilizations (below 0.6 in our example), there is enough slack to run the entire workload at the low voltage and frequency and all scaling algorithms behave similarly. Between these two extremes, WA-DVFS outperforms P-DVFS by more than 15% when the workload contains intervals of sufficient IPC disparity. Since WA-DVFS relies on such a disparity for its functioning, as the disparity drops, so does the benefit of this algorithm over P-DVFS (see Fig. 6c).

Fig. 6d shows the benefit of WA-DVFS over P-DVFS for four values of the phase length (expressed as a multiple of the processor's thermal time-constant $\tau_{thermal}$=25ms, which is obtained via simulation). Here, we assume that the high- and low-IPC segments are of equal length and that the task consumes its WCET. As the segment size drops below about a quarter of the thermal time-constant, the benefit of WA-DVFS drops as well since the processor has an opportunity to cool down during the low-IPC segments.

Fig. 6e shows the impact of Δt. Larger Δt values impose a coarser granularity on the actions of the algorithm. In Fig. 6e, the workload has a single low IPC phase and a single high IPC phase. Up to a step size of 50 ms, there is little degradation in performance; beyond that, the algorithm's performance deteriorates markedly. With a large step, IPC will be monitored less accurately. Also, the slack needed to execute in low frequency for each step is large. Thus, if there is only a small amount of slack left, it cannot be utilized. On the other hand, a very small step size will introduce more overhead. In this study, the step size used is 50 ms.

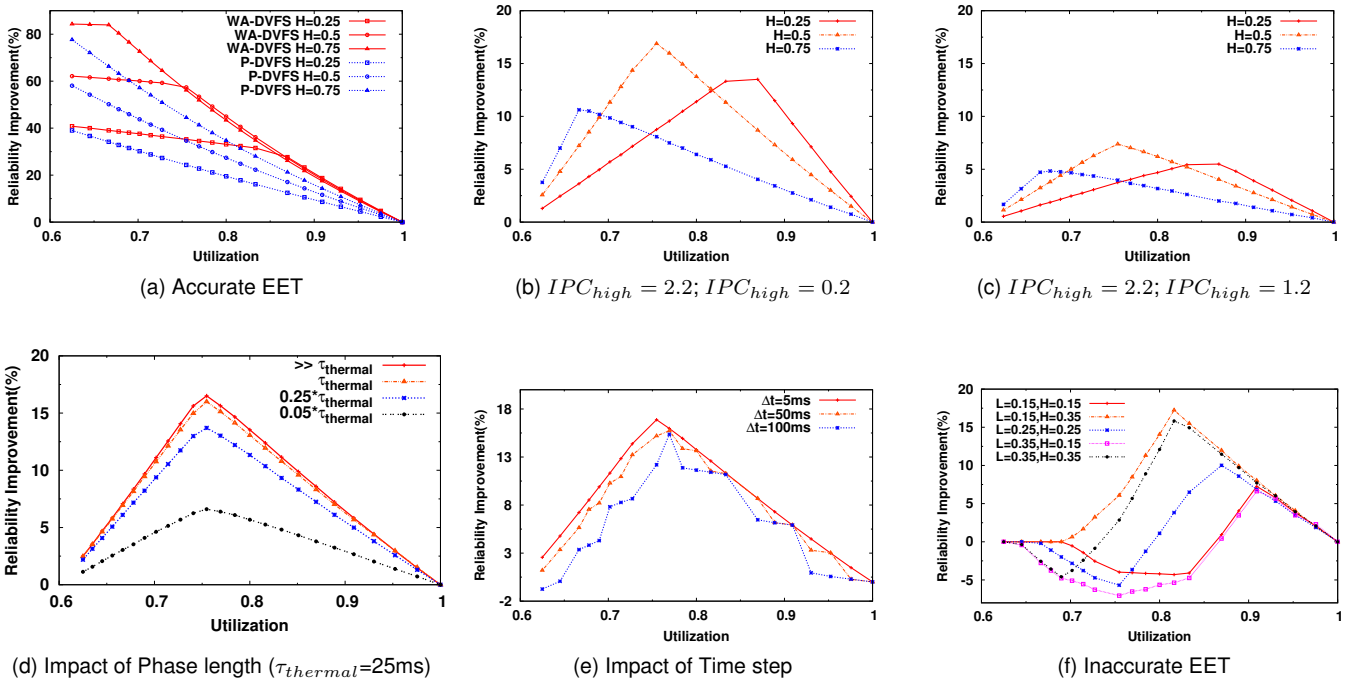
The situation where execution time is not accurately predicted is studied in Fig. 6f. In this figure, the length of the low IPC phase is assumed to be 0.5 of the total expected execution time. "L"("H") is the ratio of the actual length of the low (high) IPC phase to the WCET. As before, the comparison is against P-DVFS.

In Fig. 6f, the slacks are assigned according to the expected length of each phase. WA-DVFS behaves slightly worse than P-DVFS for a certain utilization range, especially when the actual high IPC phase is short. This is because the proposed algorithm assigns slack according to its prior (inaccurate) information and assigns more slack to the high IPC phase than needed. Since the actual high IPC phase is shorter than expected, the high-IPC phase does not use all its assigned slack which is then wasted.

Next, we study the performance of WA-DVFS for standard benchmarks. The workload consists of 4 or 5 randomly selected benchmarks from SPEC06. Each of the selected benchmarks acts as an independent task. The utilization is randomly assigned and normalized to the desired total system utilization. The ratio between actual execution time and WCET (the time to run the benchmark using provided input in SPEC06) of each task is randomly generated during simulation and is in the range (0,1]. To capture all characteristics of the workload, when the actual execution time is smaller than WCET, the power profile is shrunk proportionally instead of being cut off.

Table 2 shows the average reliability improvement of 100 workloads consisting of 4 tasks and a further 100 workloads consisting of 5 tasks. When the total utilization of the core is low (e.g., around 0.65), all workloads can be run at low frequency and there is little to be gained in using WA-DVFS. When the utilization is high (e.g., 0.85 or higher), WA-

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSUSC.2019.2958298, IEEE Transactions on Sustainable Computing



(a) Accurate EET (b) $IPC_{high} = 2.2; IPC_{high} = 0.2$ (c) $IPC_{high} = 2.2; IPC_{high} = 1.2$

(d) Impact of Phase length ($\tau_{thermal}$=25ms) (e) Impact of Time step (f) Inaccurate EET

Fig. 6. Reliability Improvement for Synthetic Workload

TABLE 2
Improvement (in %) over No DVFS using Actual Workload

	P-DVFS	WA-DVFS		
IPC_{thresh}	NA	30%	60%	90%
0.65	16.0	15.7	16.5	15.6
0.7	15.4	16.4	17.4	16.3
0.75	13.6	17.3	18.5	17.1
0.8	11.0	18.2	19.4	17.8
0.85	9.0	18.7	19.6	18.4
0.9	8.1	19.4	20.1	19.0

DVFS provides substantial gains benefit as it preferentially slows down thermally-intense phases. The impact of the IPC threshold is shown in the rightmost three columns of Table 2. Here, the IPC threshold is dynamically obtained based on the actual IPC observed over a given window. For example, if the threshold is set at 60%, we identify the IPC value which would be greater than for 60% of what was seen over the window. When the threshold is high (90%), most of the workload is treated as low-IPC phase. When the threshold is low (30%), most of the workload is treated as high-IPC phase. In either of these two cases, the WA-DVFS will behave like P-DVFS and thus has lower improvement.

Table 3 shows the temperature variation when using WA-DVFS and P-DVFS for a given utilization. This variation is expressed as the standard deviation of the temperature of individual functional blocks (e.g., branch prediction unit, integer register file, load and store queue and floating point register file) within the processor. The standard deviation of temperature is over the execution of all task sets mentioned above. WA-DVFS has a lower temperature variation than P-DVFS, especially for the blocks that are hottest during execution, such as the integer register file. As an example, Fig. 7 shows the temperature variation over time of the integer register file and the load/store queue.

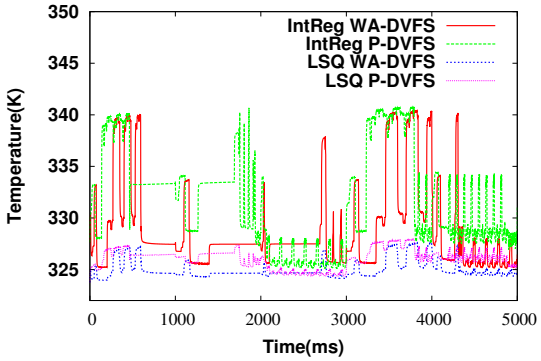
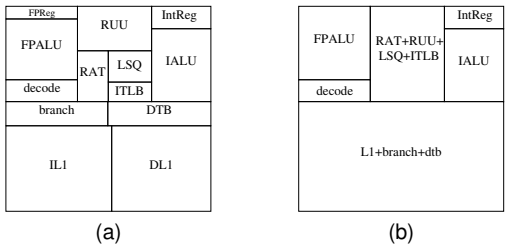


Fig. 7. Temporal Temperature Variation for the IntReg and LSQ Blocks

TABLE 3
Standard Deviation of Temperature

	P-DVFS			WA-DVFS		
Utilization	0.9	0.75	0.65	0.9	0.75	0.65
Dcache	2.092	1.739	1.098	1.818	0.784	0.943
Bpred	1.031	0.862	0.525	0.867	0.271	0.462
IntReg	10.574	8.920	5.637	9.250	3.734	4.507
LSQ	2.626	2.233	1.382	2.254	0.768	1.116
FPReg	0.800	0.664	0.407	0.667	0.212	0.368



(a) (b)

Fig. 8. (a) Original Alpha Floorplan [44] (b) Simplified Floorplan

5.3 Evaluation of Online Task Reassignment

The target hardware platform for our experiments is a system with multiple Alpha 21264 cores. Even though the

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSUSC.2019.2958298, IEEE Transactions on Sustainable Computing

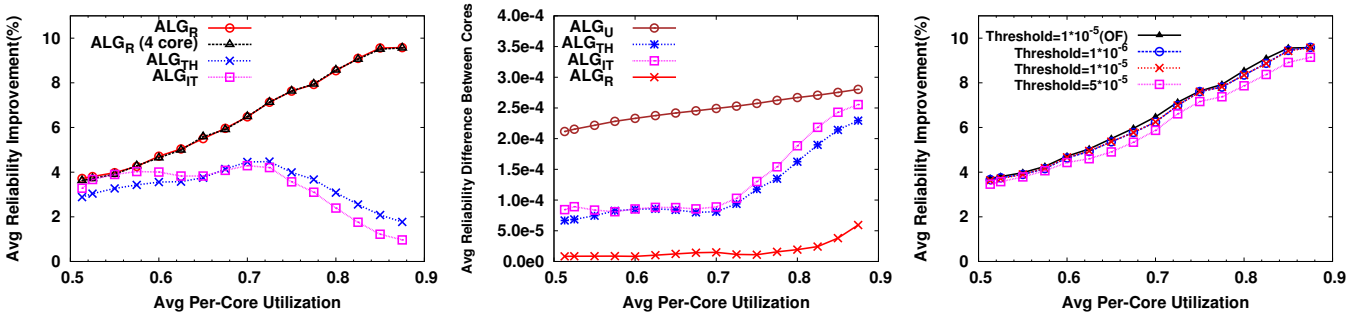


Fig. 9. Reliability Comparisons: (a) Reliability Improvement Over the Utilization Based Algorithm (b) Reliability Difference Between Cores (c) Reliability Improvement for Different Threshold Values and When Using the Original Floorplan (OF)

Alpha architecture has been discontinued for several years, its structure is often used in research work since its floorplan and other specifications are available to academic researchers; furthermore, it is reasonably close to cores today. We used Wattch [45] to calculate the power profile and TILTS [43] to calculate the temperature. In our experiments, we used the original Alpha floorplan to estimate the temperature of each block and calculate the reliability of the system. This reasonably accurate reliability value is used to assess the quality of our algorithm that relies on only approximate temperature estimations that are based on a simplified floorplan (see Fig. 8). In the simplified floorplan, some blocks are merged, reducing the total number of blocks from 13 to 6 : Integer Register File, Integer ALU, Floating Point, Decode, Branch Prediction and Load-Store Queue. This reflects the reality that exact temperature information for each block in the original floorplan is never available in practice to the algorithm (since the temperature is often estimated using performance counters [46]). We show that the lack of such exact temperature information has very little impact on the effectiveness of our algorithm.

The workload is generated following the same approach as was used for generating the actual workload to test WA-DVFS. The only difference is that there are 8 tasks in each workload. The algorithm proposed in this paper (denoted by ALG_R) is compared against the following alternatives: (*a*) Utilization balancing scheduling (denoted by ALG_U), which assigns tasks to each core before the start of execution to (approximately) balance core utilization [18], (*b*) Instantaneous temperature-based scheduling (denoted by ALG_{IT}) which triggers task reassignment when the average temperature difference between the cores is greater than a given threshold, and (*c*) Temperature history-based scheduling (denoted by ALG_{TH}) which records the history of the temperature difference between cores. When the accumulated difference exceeds a threshold, tasks will be moved from the historically hotter to the colder core [2].

The experimental results for a dual-core system are shown in Fig. 9. The improvement in a dual-core system reliability, over the utilization balancing algorithm, for our algorithm and the temperature-history and instantaneous temperature algorithms, are plotted in Fig. 9a. When the processor utilization is low, the cores are cool and not much improvement can be achieved by any algorithm. As the utilization increases, the thermal stress increases and the reliability improvement of our algorithm increases steadily. We simulated total utilizations up to an average core utilization of 0.875; higher utilizations are unlikely to occur in practice in a CPS. A quad-core system was also simulated (Fig. 9a curve ALG_R(4 core)). The improvement achieved by the proposed reliability aware algorithm over the static utilization balancing algorithm is similar to that in the dual-core case. The reliability improvement is due to the effectiveness with which our algorithm balances the reliabilities of the cores. Fig. 9b shows the reliability difference between the two cores (in a dual-core system) for all four algorithms. The reliability difference between cores is much smaller when our proposed algorithm is used.

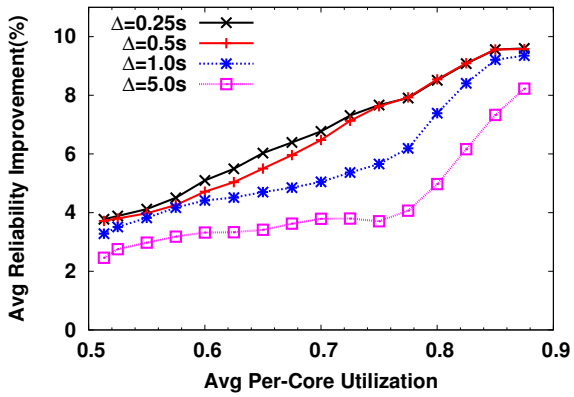


Fig. 10. The impact of the reliability update interval

The impact of using only an approximate core temperature, based on a simplified floorplan (Fig. 8b), is shown in Fig. 9(c) to be negligible. The curve in Fig. 9c marked by Threshold=1×10^{-5} (OF) shows the resulting reliability improvement when a more precise thermal information is used based on the Original Floorplan (OF). Using the temperature from the original floor plan for the reassignment algorithm can be seen as the case where accurate temperature information is available.

Finally, Fig. 10 shows the reliability improvements for five different values of the reliability update interval for a dual-core system. More frequent updates result in improved performance.

5.4 Combining intra-core and inter-core techniques

The online task reassignment algorithm determines the task assignment and the DVFS technique determines the frequency of a core when executing the tasks assigned

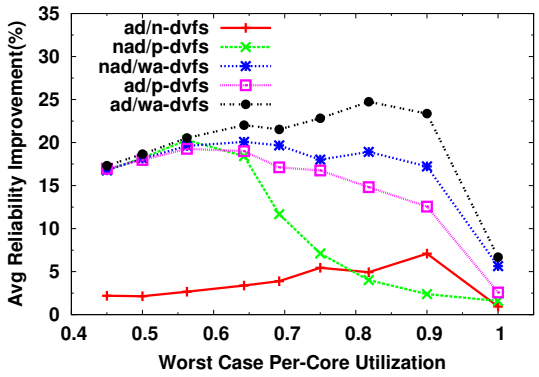


Fig. 11. Reliability Improvement using different thermal management techniques

to it. We compare five combinations of techniques: online adjustment and no DVFS on each core (ad/n-dvfs), no online adjustment and using P-DVFS on each core (nad/p-dvfs) , no online adjustment and using WA-DVFS on each core (nad/wa-dvfs), online adjustment and using P-DVFS on each core (ad/p-dvfs) and online adjustment and using WA-DVFS on each core (ad/wa-dvfs). The situation where no online adjustment and no DVFS on each core is used as baseline (all improvements are w.r.t. this configuration). Fig. 11 shows the reliability improvement of a system with two cores. As is shown, when the utilization is high or low, DVFS algorithms have similar reliability improvements. When the utilization is low, both DVFS algorithm can allow the cores to execute under low thermal stress, thus there will not be a considerable reliability difference among cores. When the utilization is high, the reassignment can be difficult or even impossible. Overall, using the proposed intra-core and inter-core thermal management performs better than prior techniques.

6 Conclusion

Improving processor reliability contributes to sustainability by reducing hardware provisioning requirements. Our heuristics enhance reliability by focusing DVFS on high-IPC stretches of the executing code and by using estimates of the thermally-accelerated age of each core in making task-to-core assignment.

Several extensions to this work are currently being pursued. These include (a) advance task profiling to obtain their IPC characteristics and execution time cumulative distribution function, and exploiting that information in initial task assignment, (b) studying the impact of adding graphical processing units (GPUs) to the computational platform, and (c) selecting task dispatch rates to trade off quality of control (of the CPS application) against the aging of the computational platform.

Acknowledgments

The authors would like to thank the reviewers for their careful reading of the draft manuscript and their helpful comments.

This work was supported in part by the National Science Foundation under grants CNS-1329831 and CNS-1717262.

An initial version of part of this paper was presented at [47]. That version introduced the idea behind the intra-core DVFS algorithm but did not include any analytical modeling; furthermore, several experimental results have been added here.

References

[1] V. Hanumaiah and S. Vrudhula, "Temperature-aware dvfs for hard real-time applications on multicore processors," *IEEE Transactions on Computers*, vol. 61, no. 10, pp. 1484–1494, 2012.

[2] D. Cuesta, J. Ayala, J. Hidalgo, D. Atienza, A. Acquaviva, and E. Macii, "Adaptive task migration policies for thermal control in mpsocs," in *2010 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, 2010, pp. 110–115.

[3] V. Hanumaiah, S. Vrudhula, and K. Chatha, "Performance optimal speed control of multi-core processors under thermal constraints," in *Design, Automation Test in Europe Conference Exhibition, 2009. DATE '09.*, 2009, pp. 1548–1551.

[4] J. L. Hennessy and D. A. Patterson, *Computer Architecture, Fifth Edition: A Quantitative Approach*, 5th ed. Morgan Kaufmann Publishers Inc., 2011.

[5] V. Hanumaiah, R. Rao, S. Vrudhula, and K. Chatha, "Throughput optimal task allocation under thermal constraints for multi-core processors," in *Design Automation Conference, 2009. DAC '09. 46th ACM/IEEE*, 2009, pp. 776–781.

[6] V. Hanumaiah, D. Desai, B. Gaudette, C.-J. Wu, and S. Vrudhula, "Steam: A smart temperature and energy aware multicore controller," *ACM Trans. Embed. Comput. Syst.*, vol. 13, no. 5s, pp. 151:1–151:25, Oct. 2014.

[7] S. Murali, A. Mutapcic, D. Atienza, R. Gupta, S. Boyd, L. Benini, and G. De Micheli, "Temperature control of high-performance multi-core platforms using convex optimization," in *Proceedings of the Conference on Design, Automation and Test in Europe*, 2008, pp. 110–115.

[8] Y. Lee, H. Chwa, K. G. Shin, and S. Wang, "Thermal-aware resource management for embedded real-time systems," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. PP, pp. 1–1, 10 2018.

[9] Z. Lu, W. Huang, J. Lach, M. Stan, and K. Skadron, "Interconnect lifetime prediction under dynamic stress for reliability-aware design," in *IEEE/ACM International Conference on Computer Aided Design, 2004. ICCAD-2004.*, 2004, pp. 327–334.

[10] Z. Lu, W. Huang, M. Stan, K. Skadron, and J. Lach, "Interconnect lifetime prediction for reliability-aware systems," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 15, no. 2, pp. 159–172, 2007.

[11] C. Zhuo, D. Sylvester, and D. Blaauw, "Process variation and temperature-aware reliability management," in *Design, Automation Test in Europe Conference Exhibition (DATE), 2010*, 2010, pp. 580–585.

[12] Y. Ma, T. Chantem, X. S. Hu, and R. P. Dick, "Improving lifetime of multicore soft real-time systems through global utilization control," in *Proceedings of the 25th Edition on Great Lakes Symposium on VLSI*, ser. GLSVLSI '15. New York, NY, USA: ACM, 2015, pp. 79–82.

[13] A. Das, A. Kumar, and B. Veeravalli, "Temperature aware energy-reliability trade-offs for mapping of throughput-constrained applications on multimedia mpsocs," in *Proceedings of the Conference on Design, Automation & Test in Europe*, ser. DATE '14, 2014, pp. 102:1–102:6.

[14] M. T. Chaudhry, T. C. Ling, A. Manzoor, S. A. Hussain, and J. Kim, "Thermal-aware scheduling in green data centers," *ACM Comput. Surv.*, vol. 47, no. 3, pp. 39:1–39:48, Feb. 2015.

[15] H. Sun, P. Stolf, and J.-M. Pierson, "Spatio-temporal thermal-aware scheduling for homogeneous high-performance computing datacenters," *Future Generation Computer Systems*, vol. 71, pp. 157 – 170, 2017.

[16] Q. Tang, S. K. S. Gupta, and G. Varsamopoulos, "Energy-efficient thermal-aware task scheduling for homogeneous high-performance computing data centers: A cyber-physical approach," *IEEE Transactions on Parallel and Distributed Systems*, vol. 19, no. 11, pp. 1458–1472, Nov 2008.

[17] F. Mulas, M. Pittau, M. Buttu, S. Carta, A. Acquaviva, L. Benini, D. Atienza, and G. De Micheli, "Thermal balancing policy for streaming computing on multiprocessor architectures," in *Design, Automation and Test in Europe, 2008. DATE '08*, 2008, pp. 734–739.

[18] J.-J. Chen, C.-M. Hung, and T.-W. Kuo, "On the minimization of the instantaneous temperature for periodic real-time tasks," in *Real Time and Embedded Technology and Applications Symposium, 2007. RTAS '07. 13th IEEE*, 2007, pp. 236–248.

[19] A. S. Hartman and D. E. Thomas, "Lifetime improvement through runtime wear-based task mapping," in *Proceedings of the Eighth IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS '12)*, 2012, pp. 13–22.

[20] C. Bolchini, L. Cassano, and A. Miele, "Lifetime-aware load distribution policies in multi-core systems: An in-depth analysis," in *Proceedings of the 2016 Conference on Design, Automation & Test in Europe (DATE '16)*, 2016, pp. 804–809.

[21] C. Bolchini, M. Carminati, M. Gribaudo, and A. Miele, "A lightweight and open-source framework for the lifetime estimation of multicore systems," in *2014 IEEE 32nd International Conference on Computer Design (ICCD)*, Oct 2014, pp. 166–172.

[22] A. Das, R. A. Shafik, G. V. Merrett, B. M. Al-Hashimi, A. Kumar, and B. Veeravalli, "Reinforcement learning-based inter- and intra-application thermal optimization for lifetime improvement of multicore systems," in *Proceedings of the 51st Annual Design Automation Conference(DAC '14)*, 2014, pp. 170:1–170:6.

[23] M. Haghbayan, A. Miele, A. M. Rahmani, P. Liljeberg, and H. Tenhunen, "A lifetime-aware runtime mapping approach for many-core systems in the dark silicon era," in *2016 Design, Automation Test in Europe Conference (DATE '16)*, March 2016, pp. 854–857.

[24] ——, "Performance/reliability-aware resource management for many-cores in dark silicon era," *IEEE Transactions on Computers*, vol. 66, no. 9, pp. 1599–1612, Sept 2017.

[25] C.-Y. Yang, J.-J. Chen, T.-W. Kuo, and L. Thiele, "An approximation scheme for energy-efficient scheduling of real-time tasks in heterogeneous multiprocessor systems," in *Design, Automation Test in Europe Conference Exhibition, 2009. DATE '09.*, 2009, pp. 694–699.

[26] E. Seo, J. Jeong, S. Park, and J. Lee, "Energy efficient scheduling of real-time tasks on multicore processors," *IEEE Transactions on Parallel and Distributed Systems*, vol. 19, no. 11, pp. 1540–1552, 2008.

[27] N. Fisher, J.-J. Chen, S. Wang, and L. Thiele, "Thermal-aware global real-time scheduling on multicore systems," in *Real-Time and Embedded Technology and Applications Symposium, 2009. RTAS 2009. 15th IEEE*, 2009, pp. 131–140.

[28] T. Chantem, Y. Xiang, X. S. Hu, and R. P. Dick, "Enhancing multicore reliability through wear compensation in online assignment and scheduling," in *Design, Automation Test in Europe Conference Exhibition (DATE), 2013*, March 2013, pp. 1373–1378.

[29] J. Srinivasan, S. Adve, P. Bose, and J. Rivers, "The impact of technology scaling on lifetime reliability," in *2004 International Conference on Dependable Systems and Networks*, June 2004, pp. 177–186.

[30] J. Srinivasan, S. V. Adve, P. Bose, and J. A. Rivers, "The case for lifetime reliability-aware microprocessors," in *Proceedings of the 31st Annual International Symposium on Computer Architecture(ISCA '04)*, 2004, pp. 276–.

[31] E. Karl, D. Blaauw, D. Sylvester, and T. Mudge, "Multi-mechanism reliability modeling and management in dynamic systems," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 16, no. 4, pp. 476–487, 2008.

[32] J. R. Black, "Mass transport of aluminum by momentum exchange with conducting electrons," in *6th Annual Reliability Physics Proceedings*, 1968.

[33] Z. Lu, W. Huang, M. Stan, K. Skadron, and J. Lach, "Interconnect lifetime prediction with temporal and spatial temperature gradients for reliability-aware design and run 134 time management: Modeling and applications," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 2006.

[34] J. Srinivasan, S. Adve, P. Bose, and J. Rivers, "Lifetime reliability: toward an architectural solution," *Micro, IEEE*, vol. 25, no. 3, pp. 70–80, 2005.

[35] E. Wu, J. Su, W. Lai, E. Nowak, J. McKenna, A. Vayshenker, and D. Harmon, "Interplay of voltage and temperature acceleration of oxide breakdown for ultra-thin gate oxides," *Solid-State Electronics*, vol. 46, no. 11, pp. 1787 – 1798, 2002.

[36] J. W. S. W. Liu, *Real-Time Systems*, 1st ed. Prentice Hall PTR, 2000.

[37] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, and R. B. Brown, "Mibench: A free, commercially representative embedded benchmark suite," in *Proceedings of the Fourth Annual IEEE International Workshop on Workload Characterization. WWC-4 (Cat. No.01EX538)*, Dec 2001, pp. 3–14.

[38] C. Krishna, "Ameliorating thermally accelerated aging with state-based application of fault-tolerance in cyber-physical computers," *IEEE Transactions on Reliability*, vol. 64, no. 1, pp. 4–14, March 2015.

[39] A. Mazouz, A. Laurent, B. Pradelle, and W. Jalby, "Evaluation of cpu frequency transition latency," *Comput. Sci.*, vol. 29, no. 3-4, pp. 187–195, Aug. 2014.

[40] P. Pillai and K. G. Shin, "Real-time dynamic voltage scaling for low-power embedded operating systems," *SIGOPS Oper. Syst. Rev.*, vol. 35, no. 5, pp. 89–102, Oct. 2001.

[41] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M. D. Hill, and D. A. Wood, "The gem5 simulator," *SIGARCH Comput. Archit. News*, vol. 39, no. 2, pp. 1–7, Aug. 2011.

[42] S. Li, J. H. Ahn, R. Strong, J. Brockman, D. Tullsen, and N. Jouppi, "Mcpat: An integrated power, area, and timing modeling framework for multicore and manycore architectures," in *42nd Annual IEEE/ACM International Symposium on Microarchitecture, 2009. MICRO-42.*, Dec 2009, pp. 469–480.

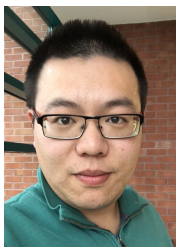
[43] Y. Han, I. Koren, and C. M. Krishna, "Tilts: A fast architectural-level transient thermal simulation method." *J. Low Power Electronics*, vol. 3, no. 1, pp. 13–21, 2007.

[44] Ptscalar. [Online]. Available: http://eda.ee.ucla.edu/PTscalar/ Dec. 2003

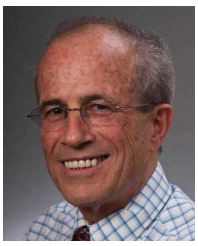
[45] D. Brooks, V. Tiwari, and M. Martonosi, "Wattch: a framework for architectural-level power analysis and optimizations," in *Proceedings of the 27th International Symposium on Computer Architecture, 2000.*, 2000, pp. 83–94.

[46] R. Rodrigues, A. Annamalai, I. Koren, and S. Kundu, "A study on the use of performance counters to estimate power in microprocessors," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 60, no. 12, pp. 882–886, Dec 2013.

[47] S. Xu, I. Koren, and C. M. Krishna, "Improving processor lifespan and energy consumption using DVFS based on ILP monitoring," in *2015 Sixth International Green and Sustainable Computing Conference (IGSC)*, Dec 2015, pp. 1–6.



Shikang Xu received his B.S. degree from Sichuan University, Chengdu, China and his M.S. degree from University of Massachusetts Amherst, Amherst, USA. He is now a PhD candidate in Computer Engineering of University of Massachusetts Amherst. His research interests include Cyber-physical systems, real-time systems, performance/reliability evaluation and fault-tolerance.



Israel Koren (M'76 - SM'87 - F'91) is currently a Professor Emeritus of Electrical and Computer Engineering at the University of Massachusetts, Amherst and a Fellow of the IEEE. He has been a consultant to numerous companies including IBM, Analog Devices, Intel, AMD and National Semiconductors. His research interests include Fault-Tolerant systems, Computer Architecture, VLSI yield and reliability, Secure Cryptographic systems, and Computer Arithmetic.

C. M. Krishna received his PhD from the University of Michigan at Ann Arbor. He is currently on the faculty of the Department of Electrical and Computer Engineering at the University of Massachusetts. His research interests include Cyber-physical systems, real-time systems, performance/reliability evaluation and distributed computing.