Improving Processor Lifespan and Energy Consumption Using DVFS Based on ILP Monitoring

Shikang Xu
Department of Electrical and Computer Engineering
University of Massachusetts Amherst

Israel Koren
Department of Electrical and Computer Engineering
University of Massachusetts Amherst

C. M. Krishna
Department of Electrical and Computer Engineering
University of Massachusetts Amherst

***Abstract*—Cyber-physical systems have a major role to play in improving efficiency and safety in transportation networks, healthcare, energy generation and distribution, among other fields. The thermal management of such systems is important since high operating temperature dramatically reduces chip lifetime; such reduced chip lifetime has obvious cost, reliability and environmental implications. This paper presents a mechanism for thermal management. We use dynamic voltage scaling preferentially applied to highly parallel segments of the workload while continuing to guarantee all hard deadlines. Experimental results show that when the workload contains time-varying levels of instruction-level parallelism (as is quite common), this lightweight algorithm improves chip lifetime over previously proposed methods. This approach is easy to implement, requires no hardware modifications and imposes very little overhead.**

***Keywords*—DVFS, Real-time systems, Cyber-physical systems, Lifetime extension**

I. Introduction

The trend in cyber-physical systems (CPS) is towards running tasks on an integrated computing platform, thereby pooling hardware resources. Processors in such a platform are often subject to high thermal stresses due to their substantial computational workloads and often high ambient temperatures. Such stresses increase device failure rates and truncate lifetimes (both highly nonlinearly as operating temperatures rise). This is of especial concern in critical applications where reliability is vital.

Dynamic voltage and frequency scaling (DVFS) has long been used for energy saving and thermal management. The typical approach to DVFS in real-time systems has been to apply it as uniformly as possible to the entire computational task [1], subject to the need to guarantee meeting task deadlines. The implicit assumption in such algorithms is that processor power consumption is roughly constant over time. However, that is not necessarily true as workloads often exhibit significant time-varying levels of instruction-level parallelism (ILP) [2]. This results in considerable variability over time in processor power consumption. The algorithm in this paper is based on the observation that the gains from voltage and frequency scaling can be enhanced when taking such variability into account. This is done by preferentially slowing down the (high-ILP) higher-power region of the schedule, thereby gaining greater energy/temperature reduction (and reduced thermal damage) for the same extent of slowdown. The ILP level can easily be monitored using performance counters (measuring the number of instructions retired per clock cycle) which are a part of most modern processors, thereby making this a practical approach. We call this workload-aware DVFS (WA-DVFS); we show that such an approach enhances the mean chip lifetime.

We make standard assumptions regarding the computational workload. The workload is assumed to be periodic, with the relative task deadlines equal to their respective periods. (Integrating sporadic tasks into a periodic framework is fairly simple.) The worst-case execution time (WCET) estimate of each task is known; its actual execution time is never greater than the WCET and is a random variable whose statistics can be measured.

We further assume that there are two voltage (and corresponding frequency) levels; the frequency levels are denoted f_{high} and f_{low}, respectively. Extending the work to more than two levels is quite simple; however, circuits' supply voltages keep dropping with advances in technology and so the scope for a large number of levels is shrinking.

The rest of this paper is organized as follows. Section II compares our approach to related prior work. Section III presents the algorithm we use and Section IV some simulation results. The paper concludes with a brief discussion in Section V.

II. Related Work

Since its introduction by Weiser *et al.* in [3], DVFS has been widely studied. Pillai and Shin developed perhaps the most widely cited DVFS algorithms for real-time systems; this is used as a baseline in studying the performance of our algorithm [1]. Aydin *et al.* developed a DVFS technique to use average workload information to predict the early completion of future executions and use this extra slack to further slow down the processor while keeping the timing constraint in view [4]. In [5], the authors designed and implemented a DVFS algorithm aiming at reducing the power of the whole system, not only that of the processor. This was integrated with a dynamic slack-based scheduler for real-time system [6], to select a feasible frequency. In [7], Bini *et al.* developed a framework to analyze and design real-time systems minimizing the energy consumption with timing constraints. This framework takes into account discrete processor speed

This work was partially supported by the National Science Foundation under grant CNS 1329831.

978–1–5090–0172–9/15/$31.00

levels, scaling overhead, and the impact of I/O. Zhuo *et al.* described a system level energy efficient task scheduling to reduce both CPU power and external device power [8], [9]. Lu *et al.* introduced a procrastination-based DVFS to achieve energy savings for systems with discrete frequencies [10].

Cho *et al.* consider computational workloads with parallel regions, running on a multi-processor system. Using Amdahl's Law, they derive frequency allocations that improve both energy consumption and performance [11].

DVFS has also been applied for thermal management of processors. Murali *et al.* propose an approach to keep the temperature below a threshold by using an offline computed table and online frequency selection according to the table [12]. Zhuo *et al.* propose a workload prediction scheme based on which a PID DVFS controller is used to maximize the performance without violating the thermal related reliability constraint [8]. Lee *et al.* develop a temperature-aware DVFS where the temperature is predicted using performance counters [13].

These works either slow down the processor regardless of the workload characteristics [1], [4], [5], [7] or use task profiling to predict workload behavior [8], [10]. In none of these is the parallelism of the task estimated and then used for thermal management, as is done in this paper.

III. Workload-Aware Voltage Scaling Algorithm

Like all DVFS algorithms for real-time systems, we exploit the slack that is generated when the workload utilization is less than 1.

Slack can be classified as *static* or *dynamic*. Static slack is known ahead of time; it is the extra time left over, even if all tasks run to their (known) worst-case execution times (WCETs). Dynamic slack is additional slack that is generated whenever a task finishes early, i.e., without consuming its WCET. Dynamic slack is known only upon task completion and expires at the deadline of the generating task. We assume a periodic workload; techniques exist to handle aperiodic tasks within the framework of periodic workloads [14].

The WA-DVFS presented in this paper is designed to make a DVFS decision every pre-defined time step (ΔT a configurable parameter). The voltage and frequency scaling decision is based on the available slack and the parallelism of the current workload. There will be no deadline miss using the proposed WA-DVFS algorithm since it uses the same static and dynamic slack policy as in [1]. Based on offline profiling, the workload is divided into the following four parts and the slack needed to run each individual part under lower voltage and frequency is calculated: high-IPC portion within the average execution time (AET), low-IPC portion within AET, high-IPC portion in WCET beyond AET and low-IPC portion in WCET beyond AET. At the beginning of every time step, the current ILP level is classified as either high or low, using the observed Instruction Per Cycle (IPC) in the previous time step. After that, the available slack is updated (by adding the newly generated dynamic slack and removing expired and used slack). If the slack reserved for current parallelism (high or low) is enough to execute the workload under lower voltage and frequency for the coming time step without missing the deadline, voltage and frequency scaling is applied.

The pseudo code of our algorithm is presented in Figures 1 to 5. An initialization step, to record the available static slack, is carried out every LCM (Least Common Multiple) of the task periods, as shown in Figure 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 AET (denoted by $s_{high_{AET}}$ and $s_{low_{AET}}$) are then calculated (lines 2 and 3). Then, the slack needed to slow down the high-IPC part of the WCET above the high-IPC portion of the AET (denoted by $s_{high_{WCET-AET}}$) is calculated in line 4. The rest of the code consists of allocating slack to the four parts of the workload mentioned above. Allocation stops once the slack has been exhausted. For example, if the slack is only sufficient to slow down the high-IPC portion of the AET, none will be left for the other portions. Slack reserved for the high IPC portion (s_{high}) will be increased by the amount of slack reserved for the high-IPC portion of the AET. (Dynamic slack, to be discussed below, is also assigned in the same order.) These are initial and static allocations. Slack reserved for the high/low IPC portion ($s_{high/low}$) will be updated dynamically as new slack is released or the current slack expires.

After initial static slack assignment at the beginning of each LCM of task periods, the available slack for high-/low- IPC phase is set equal to the pre-calculated static slack (line 15). Dynamic slack will be set to 0 since the dynamic slack expires at the beginning of each LCM (Figure 2, lines 16 to 22). At the beginning of every time step, the IPC value is updated with counter values from the previous time step (except for the very first time step since it has no previous step). The available slacks (s_{low} and s_{high}) will also be updated. To update, the `HandleNewSlack(t)` and `HandleExpiredSlack(t)` functions are called.

The function `HandleNewSlack(t)` (shown in Figure 4) assigns newly available dynamic slack from tasks which finish early in the previous time step. The amount of slack and its expiry time from task i will be saved in $S[i].slack$ and $S[i].expire$, respectively; the amount is the difference between the worst-case and actual execution time. When a deadline is reached, the slack associated with that task iteration expires. The total amount of newly available dynamic slack is accumulated in sd_{new} (Figure 4, lines 1 to 5). Assignment of the newly available dynamic slack is carried out using the same order as used in the static slack assignment (lines 6 to 15).

To deal with the expired slack, the function `HandleExpiredSlack(t)` removes the slack in the reverse order in which it was assigned; slack assigned to the low-IPC portion of the WCET will be removed first. The function `UseSlack(s)` (Figure 3) handles the use of the available slack. Note that a dynamic slack is always associated with a deadline; slack are used in an earliest-deadline first order.

IV. Experimental Results

A. System Configuration

We compare our algorithm to the previously developed DVFS technique by Pillai and Shin (denoted by P-DVFS) [1] in terms of chip lifetime. We use both synthetic and standard

Notation

f_{high}: Processor high frequency level, normalized to 1
f_{low}: Processor low frequency
$p_i, we_i/ae_i$: Period, WCET/actual execution time of task i
LCM: Least common multiple of all p_is
h: Execution time of high-IPC phase at f_{high}
l: Execution time of low-IPC phase at f_{high}
s_{static}: Static slack
$sd_{new}/sd_{expired}$: Newly-obtained/expired dynamic slack
$s_{high_{AET}/low_{AET}/high_{WCET}/low_{WCET}}$: Slack needed to slow down the 4 high/low IPC phases
s_{high}/s_{low}: Current available slack for high/low IPC phase
$S[i].slack, S[i].expire$: Amount and expiry time of slack for task i
ΔT: time step
$IPC_{threshold}$: IPC threshold used to separate high- and low-IPC
IPC_{last}: IPC in the previous time step
M_h/M_l: Statistical mean execution time of high-/low-IPC as a fraction of the low-IPC part in the worst case
f: Frequency the processor is running at
t_{sim}: Simulation time, initialized at 0

WA-DVFS:

At every time step
IF t_{sim} mod LCM =0
  Initialize( )
IF $t_{sim} = 0$
  $f = f_{high}$
ELSE
  set IPC_{last} equal to the average IPC in the past time step
  HandleNewSlack()
  HandleExpiredSlack(t_{sim})
  IF $IPC_{last} \geq IPC_{threshold}$
    IF $s_{high} \geq (\frac{1}{f_{low}} - 1) * \Delta T$
      $f = f_{low}$
      $s_{high} = s_{high} - (\frac{1}{f_{low}} - 1) * \Delta T$
      UseSlack($(\frac{1}{f_{low}} - 1) * \Delta T$)
    ELSE
      $f = f_{high}$
  ELSE
    IF $s_{low} \geq (\frac{1}{f_{low}} - 1) * \Delta T$
      $f = f_{low}$
      $s_{low} = s_{low} - (\frac{1}{f_{low}} - 1) * \Delta T$
      UseSlack($(\frac{1}{f_{low}} - 1) * \Delta T$)
    ELSE
      $f = f_{high}$
$t_{sim} = t_{sim} + \Delta T$

Fig. 1. Workload-Aware Dynamic Voltage/Frequency Scaling

benchmark (SPEC06) workloads to study the performance of our algorithm. The synthetic workload is based on power traces generated using the observed roughly linear dependence of power consumption on IPC. This linear relation is obtained using linear regression over SPEC06 benchmarks. All workloads contain only low and high IPC phase(s). 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

Initialize( )

s_{high}=0, s_{low}=0
$s_{static} = \sum_{i=1}^{n} we_i(\frac{1}{\sum_{i=1}^{n} e_i/p_i} - 1)$
$s_{high_{AET}} = h * M_h * (\frac{1}{f_{low}} - 1)$
$s_{low_{AET}} = l * M_l * (\frac{1}{f_{low}} - 1)$
$s_{high_{WCET-AET}} = h * (1 - M_h) * (\frac{1}{f_{low}} - 1)$
For s in $(s_{high_{AET}}, s_{low_{AET}}, s_{high_{WCET}}, s_{low_{WCET}})$
  IF $s_{static} > 0$
    IF $s \leq s_{static}$
      reserve s amount slack for corresponding portion
      s_{static}-=s
    ELSE
      reserve s_{static} amount slack for corresponding portion
      $s_{static} = 0$
  ELSE break
Update s_{high} and s_{low} according to above reservation
FOR i in 1 to n
  $S[i].slack = 0$
  $S[i].expire = +\infty$
$s_{high} = s_{static_{high}}$
$s_{low} = s_{static_{low}}$
$s_{dynamic_{high}} = 0$
$s_{dynamic_{low}} = 0$

Fig. 2. Initialization

UseSlack(s)

FOR ALL tasks i in S, in $S[i].expire$ ascending order
  IF $S[i].slack \geq s$
    $S[i].slack- = s$
    break
  ELSE
    $s- = S[i].slack$
    $S[i].slack = 0$

Fig. 3. Slack Consumption

value.

The simulated system has two frequency levels, a high frequency of 2.0 GHz and a low frequency of 1.2 GHz. (As mentioned previously, it is fairly straightforward to extend this work to more than two frequency levels.) The power files of the workloads were obtained using Gem5 [15] and McPAT [16]. The power files using different DVFS algorithms are sent to HotSpot [17] to get the temperature trace using different DVFS algorithms. Temperature is then used to calculate the aging of processor after running the workloads.

The Mean Time To Failure (MTTF) is a widely used metric to express processor lifetime. The MTTF due to common permanent failure mechanisms (e.g., electro-migration and oxide-breakdown) under a stable temperature was studied in [18]. To model the lifetime of a processor with time-varying temperature, we use the *thermal aging factor*, γ [19]: a processor with thermal aging factor γ_1 executing for time τ will experience the same thermal impact as the same processor with thermal aging factor γ_2 executing for time $\frac{\gamma_1}{\gamma_2}\tau$. When run at a constant temperature of T, $\gamma = 1/MTTF(T)$, which is also known as the failure rate.

HandleNewSlack(t)

1 $sd_{new} = 0$
2 FOR ALL task i finishes during $t - \Delta t$
3 S[i].slack=$we_i - ae_i$
4 $sd_{new} += S[i].slack$
5 S[i].expire=deadline of task i
6 **For** s in $(s_{high_{AET}}, s_{low_{AET}}, s_{high_{WCET}}, s_{low_{WCET}})$
7 **IF** $sd_{new} > 0$
8 **IF** $s \leq sd_{new}$
9 reserve s amount slack for corresponding portion
10 $sd_{new} -= s$
11 **ELSE**
12 reserve s_{new} amount slack for corresponding portion
13 $sd_{new} = 0$
14 **ELSE** break
15 Update s_{high} and s_{low} according to above reservation

Fig. 4. Injecting New Slack

HandleExpiredSlack(t)

1 $sd_{expire} = 0$
2 FOR ALL $S[i].expire \leq t$
3 $sd_{expire} += S[i].slack$
4 $S[i].slack = 0$
5 $S[i].expire = +\infty$
6 **For** s in $(sd_{low_{WCET}}, sd_{high_{WCET}}, sd_{low_{AET}}, sd_{high_{AET}})$
7 **IF** $sd_{expire} > 0$
8 **IF** $s \leq sd_{new}$
9 $sd_{expire} -= s$
10 s=0;
11 **ELSE**
12 s-=sd_{expire}
13 $sd_{expire} = 0$
14 **ELSE** break
15 Update s_{high} and s_{low} according to above reduction

Fig. 5. Slack Expiry

This model can be extended fairly simply to time-varying temperatures as follows. Due to thermal inertia, it is reasonable to assume that the temperature of a processor remains stable over a short interval dt. Since the system workload is periodic, the benefit of the proposed WA-DVFS over P- DVFS is $\frac{Liftime_{WA-DVFS}}{Lifetime_{P-DVFS}} - 1 = \frac{\int_{T_0}^{T_0+P} \gamma_{P-DVFS}(t)dt}{\int_{T_0}^{T_0+P} \gamma_{WA-DVFS}(t)dt} - 1$, where P is the LCM of the periods of all tasks.

B. Numerical Results

The performance of the proposed algorithm running synthetic workloads is shown in Figures 6 and 7. Figures 6 and 7(a, b), show the lifetime benefits of WA-DVFS when the workload execution time is accurately estimated (AET=WCET). Figure 6 shows the lifetime benefit of WA-DVFS when there is only a single low IPC phase and a single high IPC phase in the workload. Each curve in Figure 6 shows the impact of a different length of the high IPC phase (denoted by "H=") as a fraction of the WCET.

Figure 6(a) shows the lifetime benefits of WA-DVFS and P-DVFS over the situation where DVFS is not applied. Compared with no DVFS, using DVFS gains a significant lifetime improvement. The proposed WA-DVFS performs better than P-DVFS. Figures 6 (b) and (c) show the benefit of WA-DVFS over P-DVFS when the IPC difference between the low and high IPC phases is large or small. In these two figures, when the 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 the same. 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 Figure 6(c)).

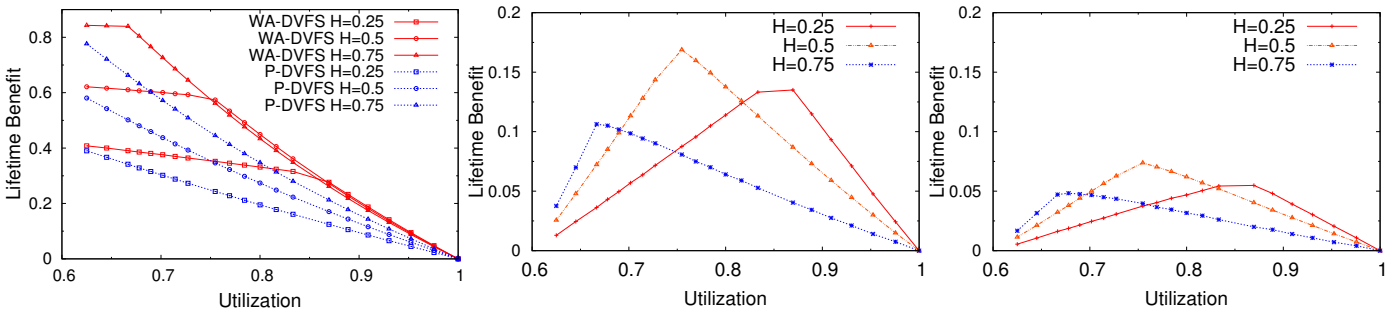
We next consider the impact of the phase length and the algorithm time-step, ΔT. In Figure 7(a), the benefit of WA-DVFS over P-DVFS for four values of the phase length (expressed as a multiple of the processor's thermal time-constant $\tau_{thermal}$=25ms), is shown. Here, we assume that the high- and 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.

For obvious reasons, the step size, ΔT, of WA-DVFS can affect its performance; this is explored in Figure 7(b). Here 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. This is because a large ΔT is a measure of the granularity of the segment over which scaling is carried out. With a large step, parallelism will be monitored less accurately. Slack smaller than the amount needed to slow down a step will not be wasted.

The case where execution time is not accurately estimated is studied in Figure 7(c). In this figure, the length of the low/high 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.

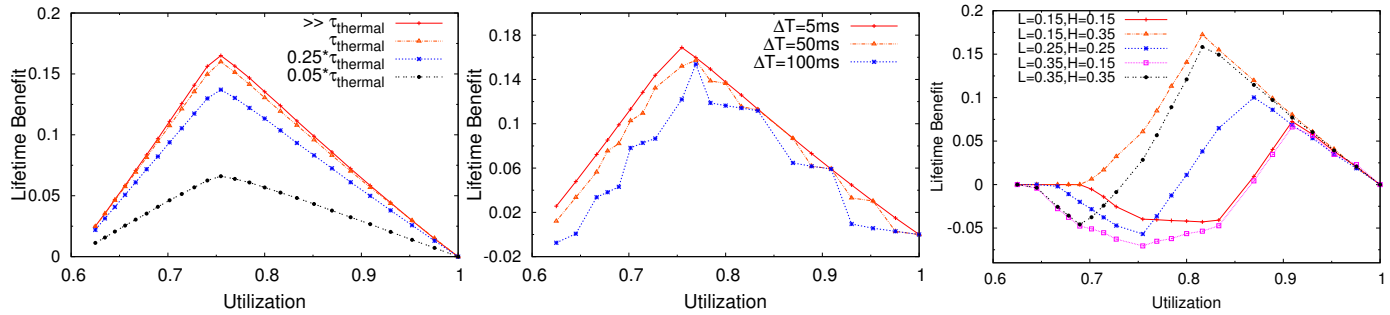
In Figure 7(c), the slacks are assigned according to the expected length of each phase. WA-DVFS behaves slightly worse than P-DVFS for a certain utilization range, especially when the actual high IPC phase is short. This is because the proposed algorithm assigns slack according to its prior (inaccurate) 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 on real (not synthetic) benchmarks. The high and low IPC phases are provided by the `hmmer` and `mcf` SPEC06 benchmarks, respectively. Figure 8 (a) compares the performance of WA-DVFS and P-DVFS to no DVFS . The workload used contains a single low IPC phase and a single high IPC phase and the workload execution time is accurately estimated. The results in



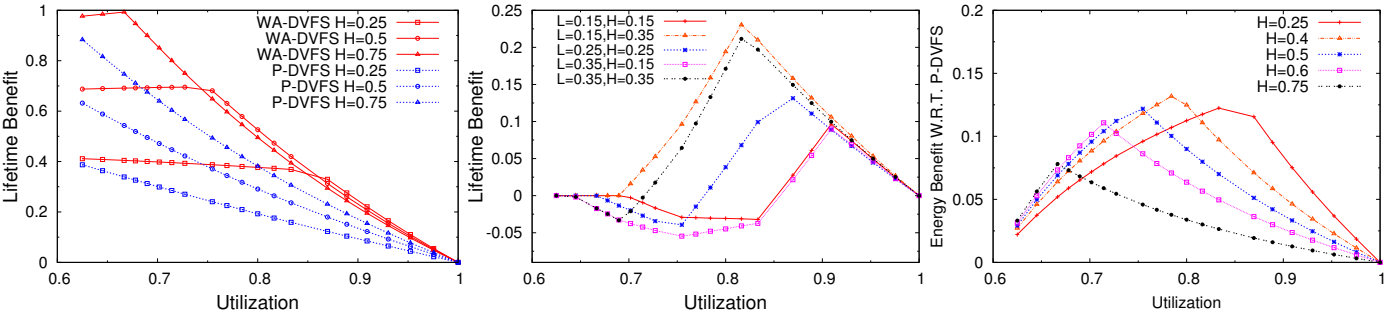
(a) (b) (c)

Fig. 6. Lifetime benefit of WA-DVFS over P-DVFS assuming an accurate execution time estimation: (a)Benefit of WA-DVFS and P-DVFS over no DVFS assuming an accurate execution time estimation (b) Large IPC difference (0.2 vs 2.2), (c) Small IPC difference (1.2 vs 2.2)



(a) (b) (c)

Fig. 7. (a)Benefit of WA-DVFS over P-DVFS for different phase lengths (b) Benefit of WA-DVFS over P-DVFS for different time steps (c) Benefit of WA-DVFS over P-DVFS for an inaccurate execution time estimation



(a) (b) (c)

Fig. 8. (a) Benefit of WA-DVFS and P-DVFS over no DVFS using an actual benchmark workload assuming an accurate execution time estimation (b) Benefit of WA-DVFS over P-DVFS using an actual benchmark workload assuming an inaccurate execution time estimation (c) Energy saving of WA-DVFS over P-DVFS using an actual benchmark workload and assuming an accurate execution time estimation

these two figures are consistent with the results shown earlier in Figure 6(a).

Figure 8(b) compares the performance of WA-DVFS and P-DVFS when the actual execution is lower than the WCET. The result is similar to that in Figure 7(c). When the actual length of the high IPC phase is smaller than the length of the low IPC phase, WA-DVFS performs marginally worse than P-DVFS for some utilizations. This result indicates that the profiling of workload can be used to decide when to use WA-DVFS and when to use P-DVFS. When the actual ratio $length\_of\_low\_IPC/length\_of\_high\_IPC$ is larger than estimated and the utilization is small, P-DVFS is the better choice. Furthermore, as seen in Figure 8(c), WA-DVFS offers some energy savings over P-DVFS (calculated as $1 - \frac{Energy_{WA-DVFS}}{Energy_{P-DVFS}}$).

Figure 9 shows the temperature variation when using WA-DVFS and P-DVFS for a given utilization. This variation is expressed as the standard deviation of temperature of individual functional blocks within the processor. As expected, WA-DVFS has a lower temperature variation than P-DVFS.

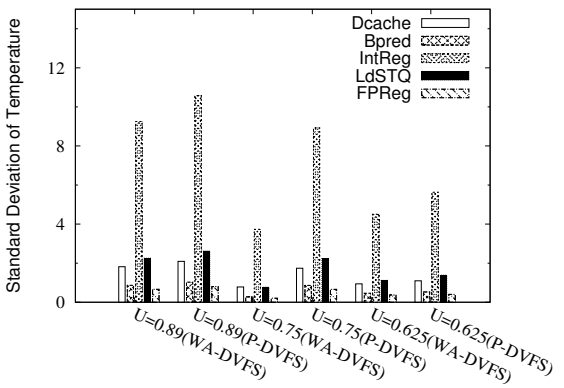


Fig. 9. Temperature variations when Using WA-DVFS and P-DVFS

V. Conclusion

In this paper, a workload-aware DVFS algorithm is proposed to extend the lifetime of processors in CPS and real-time systems where deadline constraints must be met. This algorithm is motivated by the fact that the actual parallelism of computational workloads can vary widely with time. The realized parallelism in the execution is limited by both the inherent logical parallelism (or sequentiality) of the code and the ability of the hardware platform to support superscalar execution. Modern superscalar processors which are capable of supporting a high degree of instruction-level parallelism are increasingly being used in high-end CPS applications. The natural variability that exists within the logical parallelism of the code gives us an opportunity to preferentially apply the available slack to slow down the more energy-dense highly-parallel part of the execution, thereby achieving higher energy savings and reducing thermal stress over a traditional real-time DVFS approach.

This algorithm is lightweight; modern processors provide the facility to easily estimate current levels of instruction-level parallelism. Such an approach is most useful in conditions where the utilization is moderate, when the variation in the execution time of individual tasks is not large, and when the contiguous phases of high and low parallelism are longer than the thermal time constant of the processor. This algorithm can be applied instead of the standard DVFS algorithm when such conditions are satisfied. In such cases, our results indicate that this algorithm achieves better than 15% extension in the mean lifetime. We have also studied the case where the estimated execution time is different from actual one. The proposed algorithm still gains more than a 10% benefit at some utilizations.

Extensions to this algorithm, currently under development, include task allocation to multicore platforms to maximize the benefit from WA-DVFS, and using learning-based methods to augment *a priori* information about the workload and thereby enhancing the performance of this algorithm.

References

[1] 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.

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

[3] M. Weiser, B. Welch, A. Demers, and S. Shenker, "Scheduling for reduced cpu energy," in *Proceedings of the 1st USENIX Conference on Operating Systems Design and Implementation*, ser. OSDI '94. Berkeley, CA, USA: USENIX Association, 1994. [Online]. Available: http://dl.acm.org/citation.cfm?id=1267638.1267640

[4] H. Aydin, R. Melhem, D. Mossé, and P. Mejía-Alvarez, "Power-aware scheduling for periodic real-time tasks," *IEEE Trans. Comput.*, vol. 53, no. 5, pp. 584–600, May 2004.

[5] M. Lawitzky, D. Snowdon, and P. Stefan, "Integrating real time and power management in a real system," in *Proceeding of the 4th Workshop on Operating System Platforms for Embedded Real-Time Application, Jul 2008*.

[6] S. Brandt, S. Banachowski, C. Lin, and T. Bisson, "Dynamic integrated scheduling of hard real-time, soft real-time, and non-real-time processes," in *Real-Time Systems Symposium, 2003. RTSS 2003. 24th IEEE*, Dec 2003, pp. 396–407.

[7] E. Bini, G. Buttazzo, and G. Lipari, "Minimizing cpu energy in real-time systems with discrete speed management," *ACM Trans. Embed. Comput. Syst.*, vol. 8, no. 4, pp. 31:1–31:23, Jul. 2009.

[8] J. Zhuo and C. Chakrabarti, "System-level energy-efficient dynamic task scheduling," in *Design Automation Conference, 2005. Proceedings. 42nd*, June 2005, pp. 628–631.

[9] ——, "Energy-efficient dynamic task scheduling algorithms for dvs systems," *ACM Trans. Embed. Comput. Syst.*, vol. 7, no. 2, pp. 17:1–17:25, Jan. 2008.

[10] Z. Lu, Y. Zhang, M. Stan, J. Lach, and K. Skadron, "Procrastinating voltage scheduling with discrete frequency sets," in *Design, Automation and Test in Europe, 2006. DATE '06. Proceedings*, vol. 1, March 2006, p. 6.

[11] S. Cho and R. Melhem, "Corollaries to amdahl's law for energy," *Computer Architecture Letters*, vol. 7, no. 1, pp. 25–28, Jan 2008.

[12] 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*, ser. DATE '08. New York, NY, USA: ACM, 2008, pp. 110–115. [Online]. Available: http://doi.acm.org/10.1145/1403375.1403405

[13] J. S. Lee, K. Skadron, and S. W. Chung, "Predictive temperature-aware dvfs," *Computers, IEEE Transactions on*, vol. 59, no. 1, pp. 127–133, Jan 2010.

[14] J. W. S. W. Liu, *Real-Time Systems*, 1st ed. Upper Saddle River, NJ, USA: Prentice Hall PTR, 2000.

[15] 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. [Online]. Available: http://doi.acm.org/10.1145/2024716.2024718

[16] 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 *Microarchitecture, 2009. MICRO-42. 42nd Annual IEEE/ACM International Symposium on*, Dec 2009, pp. 469–480.

[17] W. Huang, S. Ghosh, S. Velusamy, K. Sankaranarayanan, K. Skadron, and M. Stan, "Hotspot: a compact thermal modeling methodology for early-stage vlsi design," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 14, no. 5, pp. 501–513, May 2006.

[18] 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.

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