Temperature Aware Scheduling for Embedded Processors

Ramkumar Jayaseelan Tulika Mitra
Department of Computer Science
National University of Singapore
{ramkumar,tulika}@comp.nus.edu.sg

Abstract

Power density has been increasing at an alarming rate in recent processor generations resulting in high on-chip temperature. Higher temperature results in poor reliability and increased leakage current. In this paper, we propose a temperature aware scheduling technique in the context of embedded multi-tasking systems. We observe that there is a high variability in the thermal properties of different embedded applications. We design temperature-aware scheduling (TAS) scheme that exploits this variability to maintain the system temperature below a desired level while satisfying a number of requirements such as throughput, fairness and real time constraints. Moreover, TAS enables exploration of the tradeoffs between throughput and fairness in temperature-constrained systems. Compared against standard schedulers with reactive hardware-level thermal management, TAS provides better throughput with negligible impact on fairness.

1 Introduction

Decreasing feature sizes and increased complexity have resulted in very high power density in modern processors. The power dissipated is converted into heat and the processors are pushing the limits of packaging and cooling solutions [2]. The problem is prominent in the embedded domain where mobility and size constraints do not warrant for elaborate cooling mechanisms such as fan and heat sink. Increased operating temperature affects reliability. Moreover, leakage power increases exponentially with operating temperature. Increasing leakage power can further raise the temperature resulting in a thermal runaway [14]. Hence, there is a need to control temperature at all levels of system design.

Designing the thermal package for the worst-case power dissipation has become prohibitively expensive. Instead, packages are designed for worst typical behavior and rely on Dynamic Thermal Management (DTM) techniques to control the temperature. Many hardware and software-based DTM techniques have been proposed recently [3, 6, 7, 12, 13, 15]. Most of them, with the exception of [15], are reactive in nature. They invoke appropriate mechanism to cool down the system when the temperature reaches a threshold. Cooling down typically involves techniques such as global clock gating or dynamic voltage scaling that degrade the system throughput. As such, reactive techniques do not have control over the system behavior that leads to the threshold temperature. In contrast, predictive techniques [15] anticipate the future thermal behavior and take appropriate measures to avoid thermal emergencies.

Recently there has been significant interest in thermal management in embedded systems. Majority of the thermal management techniques proposed in the context of embedded systems employ static or design-time approaches [18, 8]. Recently, there have been a huge proliferation of multimedia-dominated personal embedded devices, such as PDAs, cell phones, portable audio/video players etc. These high-performance devices provide multiple functionalities, include operating system, and support concurrency through multi-tasking. For example, it is quite common for a user to run a audio clip decoder on his/her PDA, while at the same time browse the web or perform word processing.

In this scenario, where the embedded device runs a mix of soft real-time applications (e.g., audio decider) and best effort tasks (e.g., word processing) chosen at runtime by the user, the static or design-time thermal management techniques are no longer effective. Hence we propose a dynamic predictive temperature-aware scheduling (TAS) strategy for embedded multi-tasking systems consisting of a mix of soft real-time applications and best-effort applications. Our scheme exploits the variation between the temperature profiles [14, 15] of different applications to maintain the temperature below the threshold while maintaining high throughput. The basic idea of the scheduler is to differentiate among the tasks based on their predicted thermal profiles and penalize the hot tasks if necessary to maintain high system throughput. Moreover, we show that a simple parameter can control the tradeoffs between throughput and fairness of the TAS scheme. We evaluate our scheduling scheme on a ARM cortex A8 like embedded processor model. We observe that TAS can achieve higher throughput while maintaining QoS guarantees for soft real-time tasks with marginal loss in fairness among the best-effort tasks

compared to traditional schedulers employed in conjunction with DTM techniques.

2 Related Work

Dynamic Thermal Management (DTM) mechanisms can be hardware or software-based. In both hardware and software based schemes, the temperature sensors on the processor are continuously monitored and when this temperature exceeds a predefined threshold, appropriate mechanisms are invoked to reduce the temperature. Different thermal management schemes differ in the mechanisms that they employ to maintain the temperature below the threshold. Hardware-based DTM mechanisms include chip wide mechanisms such as dynamic voltage scaling [14], global clock gating [6] and ILP-based techniques [3, 15, 7].

Recently there has been widespread interest in software and system level thermal management schemes. Rohou et al. [12] propose a software-based technique where hot tasks are not allocated processing time when the system reaches a threshold. Kumar et al. [9] propose a similar software-based technique that examines the interplay between hardware and software thermal management. Reactive thermal management for hard real time systems has been discussed in [16]. While the above mentioned schemes are reactive in nature, predictive schemes for thermal management have also been proposed. Predictive techniques anticipate the future thermal behavior and take appropriate measures to avoid thermal emergencies. A predictive scheme for multimedia applications has been proposed in [15]. It exploits the frame-based nature of media applications to predict the temperature of the next frame based on the offline profiles of execution of similar frames. It is not immediately clear how this technique can be extended to predict the temperature of arbitrary applications.

In this paper we present a temperature aware scheduling strategy in the context of high performance embedded multi-tasking systems. Unlike previous reactive and predictive thermal management schemes, our strategy can optimize the system performance while maintaining other requirements such as real time constraints and fairness. In the next section we present our temperature aware scheduling framework and the temperature model.

3 Temperature Aware Scheduling Framework and Thermal Model

The goal of our thermal management framework is to maintain the temperature below the threshold while satisfying a variety of system level scheduling requirements such as throughput, fairness and real time constraints. Power consumption and hence the the thermal profiles vary between different tasks. Given a set of tasks with varying thermal profiles, the temperature can be controlled by varying the relative amount of time for which hot tasks and cold tasks execute. Our thermal aware scheduling framework exploits this observation in conjunction with voltage scaling to maintain the processor temperature below the maximum specified temperature.

The framework consists of a predictive thermal model and the temperature aware scheduler. The predictive thermal model is used to characterize the thermal properties of each task and also predict the change in temperature when a task executes starting from an initial temperature. The temperature aware scheduler uses the thermal properties of the tasks from the model and the task execution time requirements to determine the time for which each task executes. Our framework consists of both real time and best effort tasks. The scheduler ensures that real time tasks meet the deadline and for best effort tasks, the goal is to maximize the throughput while maintaining a user supplied level of fairness. For maintaining fairness as well as to maintain real time constraints, our scheduler exploits dynamic voltage and frequency scaling. Next we present our thermal model and Section 4 presents our temperature aware scheduler.

3.1 Thermal Model

In this section, we present a thermal model to predict the processor temperature at any point during the execution of a specific application. We use a predictive thermal model which models the temperature profile of a given application as an exponential function of the form [18]

$$T(t) = T_s - (T_s - T_{init}) \times e^{-Kt} \quad (1)$$

where T_s is the steady state temperature of the application which is defined as the temperature the processor would reach if the application executes indefinitely, $T(t)$ is the temperature of the processor after the application executes for t time units, T_{init} is the initial temperature, and K is a processor specific and application independent constant.

The value of the application independent processor specific constant K can be determined by fitting the observed temperature profiles for different applications into the exponential function. This process is done offline and the computed value of K is used in the predictive thermal model. For our processor model[1] we compute the value of $K = 0.00472$.

The steady state temperature of an application can be determined online by observing the temperature change over a period of time when the application executes and rearranging Equation 1.

$$T_s = \frac{T_c - T_{init} \times e^{-Kc}}{(1 - e^{-Kc})} \quad (2)$$

where T_s is the steady state temperature of the application, T_c is the temperature after the application executes for c

[1]The details of the processor model is presented in the experimental section

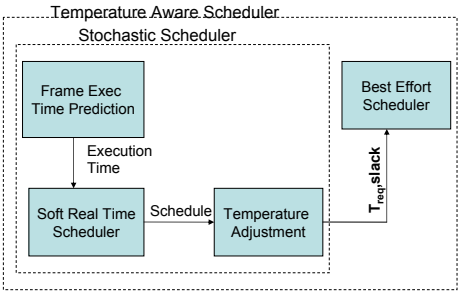

Figure 1. Temperature Aware Scheduling Policy

time units, T_{init} is the initial temperature before the application starts execution, and K is the application independent processor specific constant that is computed offline.

Once the steady state temperature of the application is known, Equation 1 can be used to predict the change in temperature when this application executes starting from any initial temperature.

Accuracy of the Prediction Model In order to check the accuracy of the predictive thermal model, we run a set of embedded benchmarks on a ARM Cortex A8 [1] like embedded processor model and observe the temperature profiles of the processor. We compare the observed temperature from these runs with the temperatures predicted from the model. For each application, we obtained the temperature curve from HotSpot with a sampling frequency of 1 millisecond. We also applied our model to predict the temperature variations and compared the temperature curves from the model and from HotSpot. For each benchmark, we measured the peak error that this the temperature difference at the point at which the predicted and the observed curves diverge the most. The maximum peak error is 0.6°C and the average peak error is 0.14°C across all the benchmarks. Hence our model provides sufficient accuracy for software based thermal management. In the next section we present our temperature aware scheduler.

4 Temperature Aware Scheduling

An overview of our thermal aware scheduler is shown in Figure 1. Our system consists of soft real-time (multimedia) and best-effort tasks. Our soft real time tasks comprise of periodic multimedia tasks that release a job per period, e.g., decoding a video frame every 30 ms. We employ a hierarchical scheduling structure typically used in multi-media systems [5, 10, 11].

The thermal aware scheduler consists of two subschedulers to handle soft real time tasks and best effort tasks. The execution requirements for the next frame is predicted using a frame execution time predictor. We employ the histogram based method for execution time rediction proposed in [17] for its accuracy and ease of implementation. The predicted frame execution times are given as input to the soft real time scheduler which schedules the soft real time tasks. We employ a simple static priority soft real time scheduler in our scheme where the audio decoding task has a higher priority than the video decoding task.

Our thermal aware scheduler has an additional thermal adjustment phase. This phase takes the soft real time schedule and the predicted frame execution time requirements as input and has two main parts (i) Ensure that the soft real time task remains below the threshold frequency/voltage scaling the soft real time tasks if necessary (ii) Compute the starting temperature (T_{req}) for the next period so that the temperature of the soft real time tasks remain below the threshold. The slack and the required temperature (T_{req}) is provided as input to the best effort task scheduler. We employ a modified version of a round robin scheduler as our best effort scheduler. Our best effort scheduler classifies tasks into hot and cold tasks and controls temperature by changing the execution time provided to the hot and cold tasks. Next we present the thermal adjustment phase employed in conjunction with the soft real time scheduler.

4.1 Thermal Adjustment Phase

The thermal adjustment phase takes the frame execution time prediction and soft real time schedule as input and performs the following tasks (i) Compute T_{req} the starting temperature for the next set of soft real time task such that temperature during the next invocation remains below the threshold (ii) Ensure that current set of soft real time tasks maintain the temperature below the threshold , voltage scaling/ dropping frames if necessary. We explain a case with two real time tasks R1 and R2 in the remainder of this discussion but the scheme can be extended to multiple soft real time tasks

4.1.1 Computing T_{req}

T_{req} is defined as the maximum initial temperature such that the execution of the next real-time task(s) is guaranteed not to exceed T_{max}. As the execution time and period of real-time tasks are known, it is easy to compute T_{req}. For example, suppose the system will execute two soft real-time tasks for $t1$ and $t2$ time units in the near future with steady state temperatures T_{s1} and T_{s2}. Then T_{req} can be determined by using Equation 1 as

$$T_{req} = T_{s1} - (T_{s1} - T_{s2})e^{Kt1} + (T_{s2} - T_{max})e^{K(t1+t2)} \quad (3)$$

This can be easily extended to multiple soft real-time tasks.

4.1.2 Voltage Scaling Soft Real Time Tasks

This phase also checks if the execution of the soft real time tasks maintains the temperature below the threshold using the model. For instance if the present temperature of the system is T_{init} and there are two soft real-time tasks for $t1$

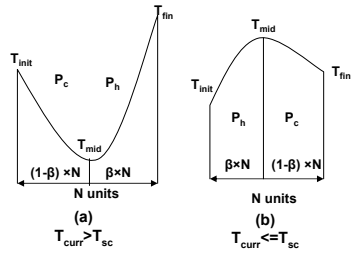

Figure 2. CPU Share between Hot and Cold Tasks

and $t2$ time units in the near future with steady state temperatures T_{s1} and T_{s2}. The temperature at the end of Task 1 and Task 2 are given by

$$T_1 = T_{s1} - (T_{s1} - T_{init})e^{-Kt1} \quad (4)$$

$$T_2 = T_{s2} - (T_{s2} - T_1)e^{-Kt2} \quad (5)$$

If $T_1 < T_{max}$ and $T_2 < T_{max}$ then the phase computes the slack and presents the slack and T_{req} to the best effort scheduler. If either one of tasks exceed the threshold then the then the corresponding tasks's frequency is lowered to the next lower frequency level. After lowering the frequency the temperature and deadline constraints are verified.If either constraints are not met then the frame is dropped. At the end of the temperature adjustment phase, a feasible soft real time schedule with frequency levels for each task as well as the corresponding slack and T_{req} values are computed. The slack and T_{req} values are sent to the best effort scheduler which uses it for scheduling the best effort tasks. Next we present our best effort scheduler.

4.2 Best Effort Scheduler

We first categorize the best-effort tasks into **hot tasks** and **cold tasks**. P_c is a cold task if its steady state temperature is below T_{req} as it would cool down the system. Similarly, P_h is a hot task if its steady state temperature is above T_{req} as it may heat up the system beyond T_{req}.

We observe that a schedule alternating between hot and cold tasks provides a good solution. The scheduler considers a pair of tasks (one hot and one cold) at a time. The problem now boils down to dividing up CPU share between these two tasks so as to keep the temperature below T_{max} and the temperature at the end of the schedule is below T_{req}. If the set of best effort tasks consists of only hot tasks then our best effort scheduler uses a voltage scaled version of the hot task as the cold task.

4.3 CPU Share between a Hot and Cold Task

Given (1) current temperature T_{curr}, (2) a hot task (P_h) with steady state temperature T_{sh}, and (3) a cold task (P_c) with steady state temperature T_{sc}, the goal of the scheduler is to allocate N time units between P_h and P_c so as to maintain the system temperature below T_{max} and the temperature at the end of the schedule is T_{req}. In particular, we determine the maximum share $0 \leq \beta \leq 1$ that can be allocated to the hot task (i.e., it executes for βN) while maintaining the system temperature below T_{max} and temperature at the end of the schedule is less than T_{req}.

Case 1: $T_{curr} \geq T_{sc}$ In this case, the cold task should be scheduled first to cool down the system and maximize the share for the hot task. Figure 2(a) shows the temperature curve over N time units. T_{mid} is the temperature after executing the cold task and T_{fin} is the final temperature.

$$T_{fin} = T_{sh} - (T_{sh} - T_{mid}) \times e^{-K\beta N} \quad (6)$$

$$T_{mid} = T_{sc} - (T_{sc} - T_{curr}) \times e^{-K(1-\beta)N} \quad (7)$$

Clearly, the temperature constraints are satisfied if $T_{fin} < T_{req}$. Hence, the maximum value of β can be obtained by substituting $T_{fin} = T_{req}$ and solving for β

$$\beta = \frac{\ln(\frac{C2}{C1+C3\times e^{-KN}})}{K \times N} \quad (8)$$

where $C1 = T_{sh} - T_{req}$; $C2 = T_{sh} - T_{sc}$; $C3 = T_{curr} - T_{sc}$

Case 2: $T_{curr} < T_{sc}$ In this case the maximum share for the hot task is obtained when it is scheduled first. This scenario is shown in Figure 2(b). Here the temperature is guaranteed to be below T_{max} if $T_{mid} <= T_{max}$ and the final temperature constraint is satisfied if $T_{fin} <= T_{req}$. So the value of β can be obtained from

$$T_{mid} = T_{sh} - (T_{sh} - T_{curr}) \times e^{-K\beta N} \quad (9)$$

$$T_{fin} = T_{sc} - (T_{sc} - T_{mid}) \times e^{-K(1-\beta)N} \quad (10)$$

Substituting $T_{mid} = T_{max}$ and solving for β

$$\beta_1 = \frac{\ln(\frac{T_{sh}-T_{curr}}{T_{sh}-T_{max}})}{kN} \quad (11)$$

Using $T_{fin} = T_{req}$ we get

$$\beta_2 = \frac{\ln(\frac{(T_{req}-T_{sc})+(T_{sh}-T_{curr})}{T_{sh}-T_{sc}})}{kN} \quad (12)$$

$$\beta = min(\beta_1, \beta_2) \quad (13)$$

Best Effort Scheduling Policy The run queue consisting of the ready tasks is split into two queues corresponding to the hot and cold tasks, respectively. The scheduler also keeps track of the CPU share given to each task so far. Whenever the scheduler is invoked, it selects the task with least share in the hot queue (P_h) and the task with the least share in the cold queue (P_c). Let N be the scheduling unit. The maximum share, β, that can be allocated to P_h in the next $2N$ time units is determined using Eqn 8 or Eqn 13. Our best effort scheduler examines the CPU share allocated to both the tasks and tries to maintain fairness while ensuring that the hot task gets no more than $\beta \times 2N$ time units.

Enforcing Fairness The scheduling scheme discussed earlier cannot ensure fairness as it gives higher preference to cold tasks in trying to keep the system temperature below T_{max}. To obtain a tradeoff between throughput and fairness, we employ selective voltage scaling for the hot tasks in conjunction with our thermal-aware scheduler. We assume that the processor supports two voltage levels V_{min} and V_{max} with corresponding frequencies f_{min} and f_{max}. To ensure fairness, we define **minimum share** s_{min} for any task. If the current share of a hot task is below s_{min}, then its voltage scaled version is transferred to the cold queue. The parameter s_{min} represents the tradeoff between fairness and throughput. Given an aggressive value of s_{min}, the system spends most of its time in voltage scaled mode thus reducing throughput. A smaller value of s_{min}, in contrast, may lead to unfairness towards the hot tasks.

5 Experimental Evaluation

Setup We use SimpleScalar 3.0 architectural simulator with configurations similar to a ARM Cortex A8 Embedded Processor [1]. The processor model is a in-order dual issue processor with 32 KB instruction/data caches, 512 entry branch miss-prediction buffer and a 13 entry branch miss-prediction pipeline. The temperature values are obtained using HotSpot-3.0 [14], an architecture-level thermal simulator working in conjunction with Wattch [4], an architecture-level power simulator. We use Wattch's linear scaling to obtain the power consumption at 1.2 V and 1.5 Ghz [1] and for voltage scaling we use a lower operating frequency of 800 Mhz which we find sufficient to remove all temperature violations [13]. We use a thermal resistance of $1.83^oC/W$ and a thermal capacitance of $112.4mJ/^oC$ and an ambient temperature of 40^oC. We assume that the temperature should not exceed 80°C based on the cooling solution [9]. The benchmarks selected from MiBench, MediaBench and EEMBC benchmark suites have steady state temperatures in the range $63.65^oC - 88.5^oC$. We have tasks with low (`patricia`, `gs`, `apcm`), medium (`jpeg`, `mpeg`, `mp3`, `blowfish`,`crc`,) and high (`rijndael`, `sha`, `susan`) thermal profile. We create eight task sets using different combinations of these benchmarks as shown in Table 1. Each task set contains applications with varying thermal characteristics. Four of these task sets have soft real-time applications (`mpeg` and `mp3`) while the other four have only best-effort applications. We assume a frame rate of 30 ms for `mpeg` and 26 ms for `mp3`. Tasks sets containing real-time applications are simulated till 450 video or audio frames complete decoding. Tasks sets containing only best effort applications are simulated for a total of 500 time slices (each time slice = 20ms). Of these task sets, S8 consists of only hot applications and hence in this case our scheduler performs scheduling by using a voltage scaled version of available hot tasks as cold tasks.

	Soft Real-Time	Best Effort
S1	`mpeg, mp3`	`sha, jpeg, adpcm, crc`
S2	`mpeg, mp3`	`rijndael, susan, patricia, gs`
S3	`mpeg, mp3`	`susan, jpeg, blowfish, gs`
S4	`mpeg, mp3`	`rijndael, sha, adpcm, patricia`
S5		`jpeg, susan, crc, gs`
S6		`sha, rijndael, crc, gs`
S7		`sha, susan, jpeg, patricia`
S8		`susan, rijndael, jpeg, blowfish`

Table 1. Composition of Task Sets

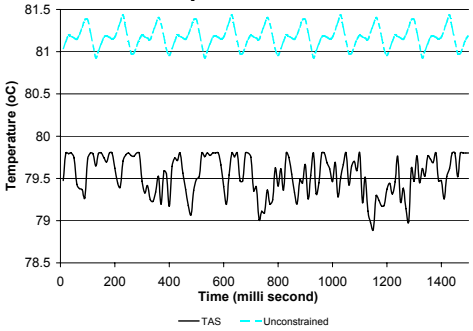

Figure 3. Temperature Profile for TAS

Traditional Thermal Management: Our temperature-aware scheduler (**TAS**) maintains the temperature below the threshold by appropriately scheduling the best-effort tasks. We compare it against a standard round robin (**RR**) scheduler for the best-effort tasks with a time slice of 20ms. The RR scheduler cannot guarantee that the temperature will not exceed the threshold and hence dynamic thermal management (DTM) techniques need to be engaged. We employ two popular DTM techniques in conjunction with RR scheduler: dynamic voltage scaling (**DVS**) and global clock gating (**CG**). *DVS* lowers the voltage/frequency of the processor whenever the system hits the threshold temperature. In case of *CG*, the processor global clock is gated (i.e., the processor remains idle). Once a DTM mechanism is engaged, the system begins to cool down. The normal operating voltage and frequency are resumed once the system temperature goes sufficiently below the maximum temperature. We implement a binary DVS scheme [13] that is shown to be performing as well as multi-level DVS schemes.

Maintaining Temperature : Figure 3 shows the temperature profiles for our thermal aware scheduling scheme as well as the temperature profile if no DTM scheme is present. We observe that *TAS* is able to keep the temperature below T_{max} for all the task sets by changing either the share of processing time given to hot and cold tasks or by appropriately scaling the operating frequency.

Throughput: Let t_{min} and t_{max} be the time the system spends in lower (800 MHz) and higher frequency (1.5GHz) for *DVS* and *TAS*. Note that *TAS* does not use low voltage as a consequence of hitting the threshold temperature. Instead, it selectively lowers the voltage of hot tasks to ensure fairness. For both cases, the throughput is defined as

$$Throughput = \frac{t_{max} + 0.53 \times t_{min}}{t_{max} + t_{min}} \tag{14}$$

Task Set	Throughput				Fairness			
	TAS(0)	TAS(0.2)	DVS	CG	TAS(0)	TAS(0.2)	DVS	CG
S1	1.0	0.96	0.91	0.82	0.94	0.97	0.98	0.94
S2	1.0	0.92	0.89	0.84	0.89	0.96	0.98	0.92
S3	1.0	0.89	0.86	0.82	0.84	0.94	0.98	0.94
S4	1.0	0.98	0.94	0.88	0.91	0.96	0.97	0.90
S5	1.0	1.0	0.95	0.87	0.97	0.97	0.98	0.91
S6	1.0	0.93	0.92	0.89	0.88	0.96	0.98	0.96
S7	1.0	0.96	0.90	0.81	0.89	0.92	0.98	0.90
S8	0.82	0.82	0.84	0.73	0.99	0.99	0.99	0.96

Table 2. Throughput and Fairness of Thermal-aware Scheduler (TAS) with $s_{min} = 0$, $s_{min} = 0.2$ and DTM Schemes

Let t_{idle} be the idle time under global clock gating. Then

$$Throughput_{CG} = \frac{t_{max}}{t_{max} + t_{idle}} \quad (15)$$

Table 2 shows the throughput for *TAS*, *DVS* and *CG*. We use two versions of *TAS* scheme with different values of s_{min} that controls fairness (see Section 4.3). The first version ($s_{min} = 0$) maximizes the throughput while ignoring fairness. The second version ($s_{min} = 0.2$) introduces a voltage-scaled version of a hot task when its share drops below 0.2.

The results shows that *TAS* performs better than clock gating in all cases. It performs better than *DVS* in cases where there is at least one cold task (S1−S7). In the case where only hot tasks are available (S8), *TAS* gives almost the same throughput as *DVS*. As expected, the throughput of *TAS* drops as the minimum expected share (s_{min}) is increased from 0 to 0.2.

Fairness: The share of a best effort task P_i is given by

$$s(i) = \frac{t_{max}(i) + 0.53 \times t_{min}(i)}{\Sigma_{i=1}^{Q} t_{max}(i) + 0.53 \times \Sigma_{i=1}^{Q} t_{min}(i)} \quad (16)$$

$$s_{CG}(i) = \frac{t_{max}(i)}{\Sigma_{i=1}^{Q} t_{max}(i)} \quad (17)$$

where $t_{max}(i)$ and $t_{min}(i)$ are the amount of time task P_i spends at maximum and minimum voltage and Q is the number of best-effort tasks. Based on the share given to each best effort application, our metric for fairness is

$$F = 1 - \frac{\Sigma_{i=1}^{Q} |s_{fair} - s(i)|}{Q} \quad (18)$$

where s_{fair} is the expected fair share.

Table 2 shows the fairness metric for *TAS*, *CG*, and *DVS*. *TAS* with $s_{min} = 0$ maximizes the throughput with lower fairness than DVS. However, if we use $s_{min} = 0.2$, then the fairness improves and becomes comparable to *DVS* with higher throughput than DVS . As the scheduler tries to be more fair (higher s_{min}), the throughput drops. Thus our predictive scheme can provide a tradeoff between system throughput and fairness when operating under thermal constraint. A reactive DVS scheme, on the other hand, only operates at one of these points.

6 Conclusion

In this paper, we have proposed a simple thermal model to predict the temperature when an arbitrary application starts execution from an initial temperature. Based on our model, we have designed and experimentally evaluated a thermal-aware scheduling strategy for multi-tasking embedded systems. Our thermal-aware scheduler can provide better throughput in comparison to DTM employed in conjunction with conventional schedulers.

7 Acknowledgements

This work is partially supported by NUS research project R-252- 000-292-112.

References

[1] ARM Cortex A8 Processor. http://www.arm.com/products/CPUs/ARM_Cortex-A8.html.

[2] S. Borkar. Design Challenges of Technology Scaling. *IEEE Micro*, 19(4), 1999.

[3] D. Brooks and M. Martonosi. Dynamic Thermal Management for High-Performance Microprocessors. In *HPCA*, 2001.

[4] D. Brooks, V. Tiwari, and M. Martonosi. Wattch: A Framework for Architectural-level Power Analysis and Optimizations. In *ISCA*, 2000.

[5] P. Goyal, X. Guo, and H. M. Vin. A Hierarchical CPU Scheduler for Multimedia Operating Systems. In *OSDI*, 1996.

[6] S. Gunther et al. Managing the Impact of Increasing Microprocessor Power Consumption. *Intel Technology Journal*, 2001.

[7] S. Heo, K. Barr, and K. Asanovic. Reducing Power Density through Activity Migration. In *ISLPED*, 2003.

[8] W-L. Hung et al. Thermal-aware task allocation and scheduling for embedded systems. In *DATE*, 2005.

[9] A. Kumar et al. HybDTM: A Coordinated Hardware-Software Approach for Dynamic Thermal Management. In *DAC*, 2006.

[10] J. Nieh and M. S. Lam. A smart scheduler for multimedia applications. *ACM Trans. Comput. Syst.*, 2003.

[11] J. Regehr and J. A. Stankovic. Hls: A framework for composing soft real-time schedulers. In *RTSS*, 2001.

[12] E. Rohou and M. Smith. Dynamically Managing Processor Temperature and Power. In *Workshop on Feedback-Directed Optimization*, 1999.

[13] K. Skadron. Hybrid Architectural Dynamic Thermal Management. In *DATE*, 2004.

[14] K. Skadron et al. Temperature-aware Microarchitecture: Modeling and Implementation. *ACM TACO*, 1(1), 2004.

[15] J. Srinivasan and S. V. Adve. Predictive Dynamic Thermal Management for Multimedia Applications. In *ICS*, 2003.

[16] S. Wang and R. Bettati. Reactive Speed Control in Temperature-Constrained Real-Time Systems. In *ECRTS*, 2006.

[17] W. Yuan and K. Nahrstedt. Energy-Efficient Soft Real-Time CPU Scheduling for Mobile Multimedia Systems. In *SOSP*, 2003.

[18] S. Zhang and K. S. Chatha. Approximation algorithm for the temperature-aware scheduling problem. In *ICCAD*, 2007.