Schedule Swapping: A Technique for Temperature Management of Distributed Embedded Systems

Farzad Samie Ghahfarokhi
Department of Computer Engineering
Sharif University of Technology
Tehran, Iran
Email: samie@ce.sharif.edu

Alireza Ejlali
Department of Computer Engineering
Sharif University of Technology
Tehran, Iran
Email: ejlali@sharif.edu

***Abstract*—A distributed embedded system consists of different processing elements (PEs) communicating via communication links. PEs have various power characteristics and in turn, have different thermal profiles. With new technologies, processor power density is dramatically increased which results in high temperature. This alarming trend underscores the importance of temperature management methods in system design. The majority of proposed techniques to address thermal issues, impose severe penalties on performance and reliability. We present Schedule Swapping, a technique for reducing peak temperature in distributed embedded systems while satisfying real-time constraints. Contrary to many other approaches, our proposed technique does not use slack time for reducing power dissipation but leaves it to be used by recovery mechanisms (rollback re-execution). The more slack time, the more the number of possible recoveries and the more reliability. We also introduce a simple yet effective scheme to ensure that all the deadlines will be met if our technique is used. This scheme also determines the order in which tasks should transmit their data in Schedule Swapping. Our experimental results show up to $18.1°C$ reduction in peak temperature. On average, Schedule Swapping achieves the peak temperature reduction of $11.13°C$.**

***Keywords*-thermal management; embedded system; distributed embedded system; task migration;**

I. Introduction

Advances in semi-conductor technology have led to an increase in the level of on-chip integration and production of chips with smaller feature sizes. With this trend of increasing power density of chip, the thermal issues in systems are becoming acute [1]. Elevated temperature not only results in increasing the circuit delay but also degrades the system reliability [2]. Furthermore, increased temperature can decrease the mean time to failure and cause physical damages to the chip [3]. In general, thermal issues are becoming significant determinant of performance, reliability and cost of processors [4]. Thermal problem is more prominent in embedded systems where exploiting cooling mechanisms such as fan and modern heat sinks is not a common practice due to mobility, power, reliability and size constraints [5].

Researchers have recently investigated many techniques for thermal management both at design time through static scheduling [6], static voltage scaling [6], packaging and task allocation [7], and at run time through various dynamic scheduling ([5], [8] and [9]) and dynamic thermal management (DTM) methods ([10]). Since the program workload may vary when running under different inputs and conditions, it is difficult to predict such variations prior to the time when the task is executed [11]. Making decisions on thermal management at design time (static solutions) when sufficient information about the workload and communication requirements are not available, are not preferable solutions [8]. In contrast, run time solutions can dynamically monitor the workload and operating conditions, and trigger the response to thermal emergencies.

As in many embedded systems reliability is a major concern [12], rollback recovery schemes use slack time to increase system reliability. Slack time also is exploited by voltage and frequency scaling schemes to lowering power consumption and in turn, reducing on-chip temperature [13]. So some of low power and DTM techniques which use slack time, have a significant negative impact on system reliability [2]. We describe these thermal management schemes in Section II. Since reduced supply voltage has negative impacts on reliability [13], we should look for thermal management techniques which do not scale operating voltage/frequency.

In this paper, we propose *Schedule Swapping*, a technique to manage temperature in distributed embedded systems while maintaining real-time constraints. This technique leverages a full voltage swing, thus will not cause reliability degradation. When the temperature of a processor exceeds a predetermined threshold, it will swap its assigned tasks with the coldest processor. We present a design time scheme that is used to check whether the real-time constraints will be satisfied after Schedule Swapping or not. This scheme also determines the order in which the tasks' data are transmitted to meet all deadlines. To the best of our knowledge this is the first thermal management technique presented for distributed real-time embedded systems.

The rest of this paper is organized as follows: Section II provides an overview on DTM techniques and limitations of them. Section III presents our used task, system and

thermal models for distributed embedded system. Section IV describes the basic idea and details of our proposed scheme. The experimental results are presented in Section V. Finally, Section VI concludes the paper.

II. A Brief Review on DTM Techniques

We can classify DTM techniques into several categories: those that reduce power consumption ([14], [10] and [15]) and those that try to balance the load and power ([16], [17]). The first category contains the techniques, whose goal is to reduce the power dissipation, such as:

- Clock Gating [14] stalls the clock to zero the dynamic power consuming and let the processor to cool in response to the thermal overload. However it has a negative impact on performance and can cause missed deadlines in real-time systems [18].
- Fetch Gating [10] stalls fetching instructions to lessen the activity and power density in the pipeline when the chip reaches a temperature threshold.
- Dynamic Voltage Scaling (DVS) [10] adjusts CPU voltage/frequency to reduce power dissipation in response to the thermal situation. Despite this technique is effective at reducing power dissipation, and hence decreasing temperature, it has some potential drawbacks. First, lowering the supply voltage and frequency increase the execution time of the tasks which may potentially violate the timing constraints of system. Secondly, lowering the supply voltage exacerbates noise immunity and noise margin [13]. In the systems with a high reliability requirement, DVS may not be an appropriate choice to use.
- Architecture-level mechanisms [15] adapt micro-architecture parameters, such as instruction window size, fetch width and issue width, and get reduction in on-chip temperature at the cost of performance degradation.

The second class of DTM techniques contains those which attempt to distribute workload and power. This approach performs the load balancing at two different levels of granularity:

- *Activity Migration* [16] is a fine-grained example which moves computation between replicated units on the die. In this scheme, the units with high activity (such as register file or issue window) replicated to avoid hotspots. When one unit heats up, the computation will transferred to the second unit allowing the first to turn cold.
- *Task Migration (TM)* or *Process Migration* [17] is a coarse-grained power balancing technique that was proposed for multi-core platforms and MPSoCs. This technique transfers the threads from the overheated core to the cold cores periodically due to temperature balancing of the cores. TM is not applicable to the systems which execute dependent tasks (e.g. task graphs). This can be explained by these facts that *(i)* the target processor may have no enough idle time to allot to the migrated task, *(ii)* migrating a task can lead to increase in schedule length and violation of deadline.

III. Models

A. Task Model

A task set T of t dependent periodic tasks is denoted by $T = \{T_1, T_2, \cdots, T_t\}$. Data dependencies between tasks can be captured by a directed acyclic graph named task graph. In the task graph, nodes represent tasks and directed edges represent data dependencies among tasks. If T_i has data dependency on T_j, an edge will connect T_i to T_j in task graph. We can make a general assumption that all the tasks have identical periods and deadlines [9].

B. System Model

We consider a distributed embedded system consists of n processing elements (PEs) like processors, microprocessors, DSPs, FPGAs and ASICs. PEs are connected by communication units (CUs). We denote the PEs by a set $P = \{P_1, P_2, \cdots, P_n\}$. After scheduling task graph and allocation of each task to a particular computational note, each PE P_i will be associated with a set of assigned tasks.

C. Thermal Model

Based on the known duality between heat transfer and electrical phenomena, the temperature profile of a given task is an exponential function of the form [1]:

$$T(t) = T_s \times (1 - e^{-t/R_{th}C_{th}}) + T_{init} \times e^{-t/R_{th}Cth} \quad (1)$$

where $T(t)$ is the temperature of the processor after the task executes for t units of time, T_s is the steady state temperature of task which processor might reach if the task executes for a long time, T_{init} is the initial temperature, and R_{th} and C_{th} are thermal resistance and thermal conductance respectively, which are processor specific constant parameters. Thermal RC constant time characterizes the exponential rise and fall in temperature.

The temperature behavior of a processor which executes a sequence of tasks periodically, includes short-term temperature reactions and long-term ones. The former occurs within each task graph period. As the task sequence repeats itself, the temperature of processor at the end of period, elevates as compared to the beginning of the period. It approaches to its steady state after many periods repetition. In the steady state, the thermal profile experiences a recurring pattern [6]. Figure 1 shows this long-term behavior.

IV. Schedule Swapping Scheme

This section presents our proposed scheme to manage the temperature in distributed embedded systems.

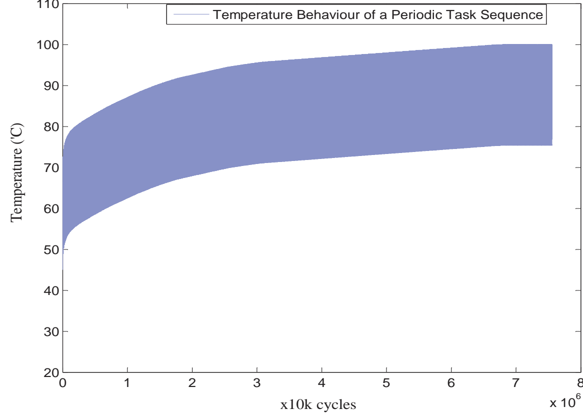

Figure 1: long-term temperature changes

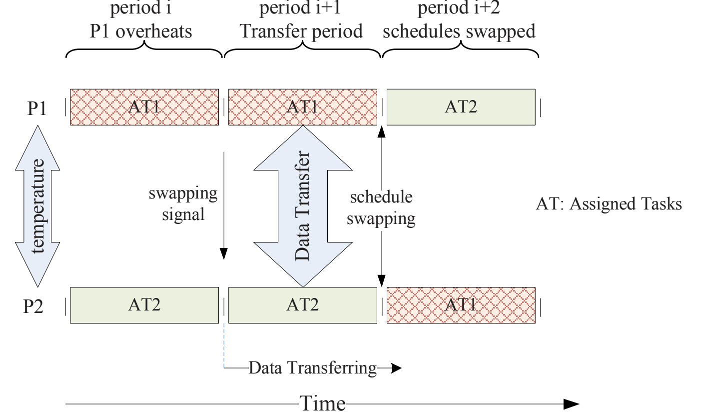

Figure 2: Different phases of Schedule Swapping between PE P1 and PE p2.

A. Main Idea

According to the system model presented in Section III, each of the PEs execute its assigned task periodically. Since the circuit activity may vary across the tasks, which in turn causes significant variations in their power dissipation characteristics, a substantial difference between the thermal behavior of processors has been experienced.

The phases required in our scheme for Schedule Swapping between PEs are illustrated in Figure 2 and are as follows:

- Each processor reads its temperature sensor and broadcast the value to other PEs over their communication units (e.g. common bus) periodically.
- Then each PE sort out all temperature values being sent by other PEs. While the temperature values are serialized over the communication link, they can be sorted by insertion sort in $O(n)$. When the temperature value of a new PE is received, it will be inserted into its correct position. Since insertion sort is an online algorithm and can process the inputs in a serial fashion [19], it is the best choice to be used in our technique.
- When the temperature of a processor exceeds a certain threshold, it will negotiate with the coldest PE by sending a swapping signal.
- In the next period, in addition to normal execution, data transfer will be done. We named this period, *Transfer Period*. After finishing the execution of each task $T_i \in AT_j$, (where j is one of the involved PEs) in Transfer Period, the data required by other PE to execute T_i in the next period, will be sent to it.
- If the second hottest processor exceeds the temperature threshold, its schedule will be swapped by the second coldest processor and etc.

After completion of swapping the schedules, the overheated PE begins to cool down and the cool PE starts to heat up.

Example: Schedule Swapping technique is explained with the help of an example illustrated in Figure 2. In the period i, P_1 overheats and its scheduling will be swapped with the scheduling of a cool PE, P_2. P_1 sends the swapping signal to P_2. In the period $i+1$, tasks' data are transferred between P_1 and P_2. Finally, in the period $i+2$ schedules are swapped; P_1 executes the tasks were assigned to P_2 and vice versa.

Needing large amounts of data, some PEs may miss the deadline if they use Schedule Swapping technique. We present ESTF, a design time scheme to guarantee meeting timing constraints in the Schedule Swapping technique. The PEs that can assist in Schedule Swapping will be determined at design time by the ESTF scheme. It will be explained in the next subsection.

To save communication bandwidth and decrease the time needed for swapping the schedules, the executable code of tasks will reside in all PEs. A more effective solution is to determine the potential candidates for Schedule Swapping at design time. Many of embedded systems execute a set of applications known at design time [20]. This provides the opportunity to determine the thermal profile of tasks and uncover the PEs which may be involved in Schedule Swapping. Then we can only copy the executable code of tasks which assigned to the potential candidate PEs, at design time.

While this technique does not reduce operating voltage/frequency, it leaves the available slack time to be used by rollback recovery and re-execution mechanisms. Dedicating the slack time to re-execution, preserves fault tolerance and system reliability. On the other hand, not reducing voltage/frequency avoids the increase in the rate of transient faults [13].

Not affecting the tasks execution time, our technique guarantees all the deadlines. The only potential risk of missing a deadline, that may be associated with this technique, is in data transfer phase. In the next subsection we discuss a scheme to guarantee real-time constraints.

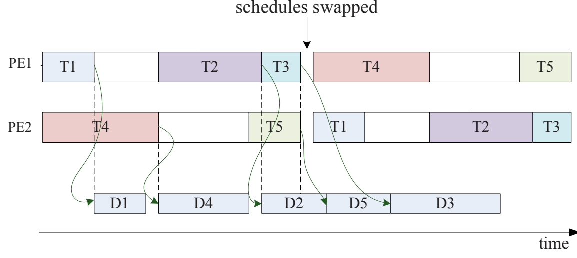

Figure 3: Scheduling data transfer for two PEs involved in Schedule Swapping bt ESTF scheme.

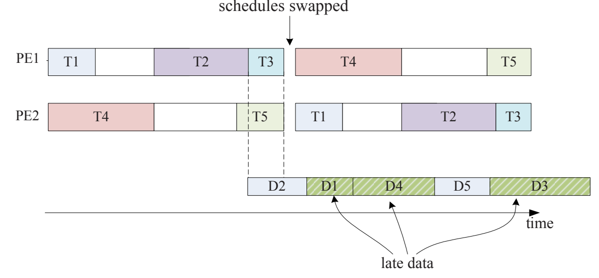

Figure 4: An example of scheduling data transfer which did not use ESTF scheme and caused deadline miss.

B. Satisfying Timing Constraints

As discussed in the previous subsection, data transferring is the only phase that can potentially lead to deadline miss if we do not consider timing constraints. In this subsection we present a solution to guarantee meeting all deadlines in Schedule Swapping scheme.

Let $LDTT_i$ be the longest time required by task T_i to transmit its data through communication link. If the data transfer between PEs can be scheduled at design time, the deadline will be met and the Schedule Swapping can be exploited at run time. We present a simple and effective scheduling scheme for data transferring between two PEs named *Earliest Start Time First (ESTF)*. It is noteworthy that we do not aim at introducing a new task scheduling; our scheme will be performed after the tasks mapped to PEs and task scheduling is done. Given the mapped and scheduled task graph, ESTF scheme should be used for every two PEs. It is as follows:

1) Two PEs, their assigned tasks, $LDTT$, start time and finish time of each task are given as input.
2) Select the task which has the earliest start time in scheduling (e.g. task T_k). If two tasks have an identical start time, randomly select one of them.
3) After the finish time of task T_k in the scheduling, if the communication link is idle, allot the link to T_k for $LDTT_k$ units of time.
4) If any task remains, return to step 2.

If the data of each task is received before its start time in next period, the whole tasks can be start to execute at their specified time and the deadline will be met. Existing a task which violates this condition, Schedule Swapping can not be used for these two PEs. Figure 3 illustrates an example of ESTF scheme. Tasks T_1 and T_4 have the earliest start time; T_1 is selected randomly to transmit its data first. Then T_4 and T_2 block the link, respectively. After finish time of T_5, the link is occupied, so we should wait until the link becomes idle and then allot the link to T_5. Finally, T_3 which has the latest start time will occupy the link. As depicted in this figure, the whole tasks will receive their required data before their next execution time, and the Schedule Swapping can be used for these two PEs.

The ESTF scheme gives highest priority to the tasks with the earliest start time to transmit their data. The rationale behind the ESTF is straightforward: the task which will execute earlier, requires its input data earlier. We do not claim that ESTF is the best or the only scheme, but it is simple and effective. Suppose that we do not use ESTF and another task is selected for transmitting data instead of T_1 and T_4. For instance, Figure 4 shows a scenario in which task T_2 is selected to transmit its data first, contrary to the ESTF scheme. As shown in this figure, exploiting a scheme different from ESTF causes the required data of T_1, T_3 and T_4 to arrive late and in turn deadline miss.

V. Experimental Results

A. Simulation Setup

To setup a distributed system, compute the temperature values and power dissipation, we use the following established tools.

- For modeling a distributed embedded system, we use TrueTime [21], a MATLAB/Simulink-based tool. TrueTime provides an infrastructure to simulate the behavior of multitask real-time processors connected by a communication network.
- We use HotSpot (version 5) [22] as our thermal simulator. HotSpot is based on the well-known duality between heat transfer and electrical phenomena. This duality can be exhibited by the fact that we can describe both these phenomena by exactly the same differential equations. HotSpot takes as input the power trace values over time, the chip floorplan and the configuration details of package.
- Power trace values must be supplied to the HotSpot, so we need a power simulator. PTScalar [23] is a cycle-accurate microarchitecture level power simulator that takes both dynamic and leakage power consumption into account. It is an extension of Simple-Scalar toolset which models power dissipation at the 65nm node, using a model like Wattch [24]. Furthermore, it models the temperature and voltage dependency of leakage power.

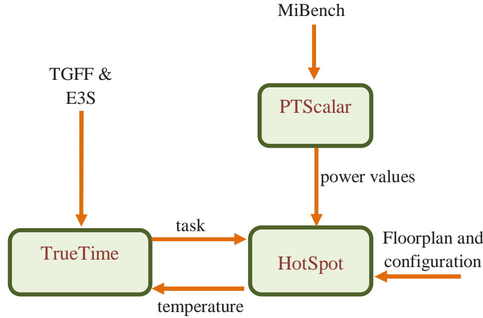

Figure 5: General tool chain for simulation setup.

- We use the Embedded System Synthesis Benchmarks Suite (E3S) as the task graphs. E3S has five benchmarks in total which represents different applications. A hard deadline is associated to each sink task. We also use TGFF [25], which provides pseudo-random task graphs.
- As workloads, we use a set of benchmarks from the MiBench [26], an embedded benchmark suite. These benchmarks are selected from automative category (qsort, bitcount, basicmath, susan), consumer category (lame, jpeg), security category (sha), network category (patricia, dijkstra) and telecomm category (fft, adpcm, gsm).

To setup the experimental framework, we first configured a distributed system in TrueTime, consisting of some computational nodes (PEs) and a communication infrastructure to support communications among tasks and then implemented the scheduled task graphs (of E3s and TGFF) on the nodes. Afterwards, we assigned the applications of MiBench to the tasks of task graphs randomly, and calculated the power dissipation of them by PTscalar. We also coupled the TrueTime with HotSpot thermal simulator to perform the temperature simulation in parallel with task execution. The task execution modeling and swapping of schedules is done by TrueTime. Figure 5 shows the simulation setup.

Most of embedded processors do not posses modern packages, heat sinks and heat spreaders [5]. To simulate this lack, we set the parameters of HotSpot to the values summarized in Table I [27].

Table I: HotSpot parameters for embedded processors.

Parameter	Value
Die Thickness	$0.15mm$
Convection Capacitance	$140J/K$
Convection Resistance	$1.5K/W$ - $4K/W$
Heat Sink Thickness	$1mm$
Spreader Thickness	$0.1mm$

B. Results

Using the simulation tool chain, we first run each task graph with randomly assigned workloads until it reaches to steady state temperature without using any thermal management technique. Afterwards, we repeated the simulations and this time used Schedule Swapping technique. We only report the results for 11 benchmarks consist of Office (3 benchmarks), Networking (4 benchmarks) and TGFF (4 benchmarks).

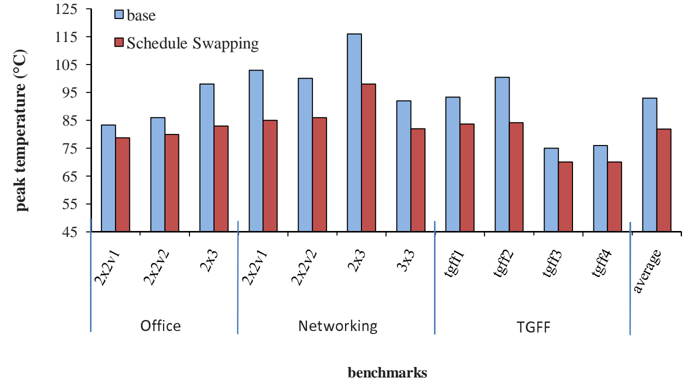

Figure 6: Peak temperature reduction by our technique for different benchmarks.

Table II: The maximum and the average reduction in peak temperature.

Peak Temperature	Base	Schedule Swapping	Reduction
Average	92.99°C	81.85°C	11.13°C
Maximum	116.53°C	98.43°C	18.1°C

The simulation results are depicted in Figure 6. This figure shows the peak temperature of different benchmarks for two cases: when no thermal management is used (base), and when Schedule Swapping is used. Our technique achieves a peak temperature reduction up to 18.1°C and 11.13°C on average. Note that by having many cold PEs and a few hot PEs in a system, Schedule Swapping will demonstrate more efficacy to reduce peak temperature. The peak temperature reduction, the maximum peak temperature and the average of peak temperature attained by Schedule Swapping is reported in Table II.

VI. Conclusion

Reducing peak temperature of processor and maintaining its high reliability are two major challenges in embedded systems design. In this paper we have presented Schedule Swapping, a technique to reduce peak temperature in distributed embedded systems while satisfying real-time constraints and preserving system reliability. In this technique, when a processor overheats, its assigned tasks and schedule will be swapped with the schedule of the coldest processor. Also we have presented ESTF, an algorithm to understand the feasibility of meeting all deadlines if Schedule Swapping is used. This is a design time algorithm that should be used for each two pairs of PEs to check the real-time constraints in data transfer phase. ESTF determines the order in which

tasks' data should be transferred to arrive at their destination before the start of task execution. While our technique does not reduce supply voltage and leaves slack time for fault tolerance, it does not cause the system reliability to degrade. The proposed technique has achieved a peak temperature reduction as much as $18.1°\mathrm{C}$ and $11.13°\mathrm{C}$ on average.

REFERENCES

[1] K. Skadron, M. Stan, W. Huang, S. Velusamy, K. Sankaranarayanan, and D. Tarjan, “Temperature-aware computer systems: Opportunities and challenges,” *IEEE Micro*, vol. 23, no. 6, pp. 52–61, 2003.

[2] R. Rajaraman, K. Ramakrishnan, N. Vijaykrishnan, Y. Xie, and M. Irwin, “Temperature and voltage scaling effects on electrical masking,” 2008.

[3] J. Council, “Failure mechanisms and models for semiconductor devices,” *JEDEC Publication JEP122-A*, 2002.

[4] K. Banerjee, M. Pedram, and A. Ajami, “Analysis and optimization of thermal issues in high-performance vlsi,” in *Proceedings of the 2001 international symposium on Physical design (ISPD01)*, 2001, pp. 230–237.

[5] R. Jayaseelan and T. Mitra, “Temperature aware scheduling for embedded processors,” in *22nd International Conference on VLSI Design*, 2009, pp. 541–546.

[6] ——, “Temperature aware task sequencing and voltage scaling,” in *International Conference on Computer-Aided Design, ICCAD*, 2008, pp. 618–623.

[7] W.-L. Hung, Y. Xie, N. Vijaykrishnan, M. Kandemir, and M. J. Irwin, “Thermal-aware task allocation and scheduling for embedded systems,” *Design, Automation and Test in Europe Conference and Exhibition, DATE'05*, vol. 2, pp. 898–899, 2005.

[8] A. Coskun, T. Rosing, K. Whisnant, and K. Gross, “Static and dynamic temperature-aware scheduling for multiprocessor socs,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 16, no. 9, pp. 1127–1140, 2008.

[9] V. Swaminathan and K. Chakrabarty, “Real-time task scheduling for energy-aware embedded systems,” *Journal of the Franklin Institute*, vol. 338, no. 6, pp. 729–750, 2001.

[10] D. Brooks and M. Martonosi, “Dynamic thermal management for high-performance microprocessors,” in *International Symposium on High-Performance Computer Architecture, ISCA'01*, 2001, pp. 171–182.

[11] A. Coskun, “Efficient thermal management for multiprocessor systems,” Ph.D. dissertation, University Of California, San Diego, 2009.

[12] V. Narayanan and Y. Xie, “Reliability concerns in embedded system designs,” *Computer*, vol. 39, no. 1, pp. 118–120, 2006.

[13] D. Zhu, R. Melhem, and D. Mossé, “The effects of energy management on reliability in real-time embedded systems,” in *International Conference on Computer-Aided Design, ICCAD*, 2004, pp. 35–40.

[14] V. Tiwari, D. Singh, S. Rajgopal, G. Mehta, R. Patel, and F. Baez, “Reducing power in high-performance microprocessors,” in *Proceedings of the 35th Annual Design Automation Conference, DAC*, 1998, pp. 737–742.

[15] R. Jayaseelan and T. Mitra, “Dynamic thermal management via architectural adaptation,” in *Proceedings of the 46th Annual Design Automation Conference, DAC*, 2009, pp. 484–489.

[16] S. Heo, K. Barr, and K. Asanović, “Reducing power density through activity migration,” in *Proceedings of the 2003 international symposium on Low power electronics and design, ISLPED*, 2003, pp. 217–222.

[17] J. Donald and M. Martonosi, “Techniques for multicore thermal management: Classification and new exploratio,” in *33rd International Symposium on Computer Architecture, ISCA'06*, 2006, pp. 78–88.

[18] A. Ferreira, D. Mosse, and J. Oh, “Thermal faults modeling using a rc model with an application to web farms,” in *19th Euromicro Conference on Real-Time Systems, ECRTS'07*, 2007, pp. 113–124.

[19] A. Borodin and R. El-Yaniv, *Online computation and competitive analysis*. Cambridge University Press Cambridge, 1998.

[20] J. Henkel and S. Parameswaran, *Designing embedded processors: a low power perspective*. Springer Publishing Company, Incorporated, 2007.

[21] D. Henriksson, A. Cervin, and K. Årzén, “Truetime: Realtime control system simulation with matlab/simulink,” in *Proceedings of the Nordic MATLAB Conference, Copenhagen, Denmark*, 2003.

[22] K. Skadron, M. Stan, K. Sankaranarayanan, W. Huang, S. Velusamy, and D. Tarjan, “Temperature-aware microarchitecture: Modeling and implementation,” *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 1, no. 1, p. 125, 2004.

[23] W. Liao, L. He, and K. Lepak, “Temperature and supply voltage aware performance and power modeling at microarchitecture level,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, TCAD*, vol. 24, no. 7, 2005.

[24] D. Brooks, V. Tiwari, and M. Martonosi, “Wattch: a framework for architectural-level power analysis and optimizations,” *ACM SIGARCH Computer Architecture News*, vol. 28, no. 2, p. 94, 2000.

[25] R. Dick, D. Rhodes, and W. Wolf, “Tgff: task graphs for free,” in *Proceedings of the 6th international workshop on Hardware/software codesign*, 1998, pp. 97–101.

[26] M. Guthaus, J. Ringenberg, D. Ernst, T. Austin, T. Mudge, and R. Brown, “Mibench: A free, commercially representative embedded benchmark suite,” in *2001 IEEE International Workshop on Workload Characterization*, 2001, pp. 3–14.

[27] S. Sharifi, A. Coskun, and T. Rosing, “Hybrid dynamic energy and thermal management in heterogeneous embedded multiprocessor socs,” in *Proceedings of Asia and South Pacific Design Automation Conference, ASPDAC*, 2010.