A Fast High-Level Event-Driven Thermal Estimator for Dynamic Thermal Aware Scheduling

Jin Cui, *Student Member, IEEE,* and Douglas L. Maskell, *Member, IEEE*

***Abstract*—Thermal aware scheduling (TAS) is an important system level optimization for many-core systems. A fast event driven thermal estimation method, which includes both the dynamic and leakage power models, for monitoring temperature and guiding dynamic TAS (DTAS) is proposed in this paper. The fast event driven thermal estimation is based upon a thermal map, with occasional thermal sensor-based calibration, which is updated only when a high level event occurs. To minimize the overhead, while maintaining the estimation accuracy, prebuilt look-up-tables and predefined leakage calibration parameters are used to speed up the thermal solution. Experimental results show our method is accurate, producing thermal estimations of similar quality to an existing open-source thermal simulator, while having a considerably reduced computational complexity. Based on this predictive approach, we take full advantage of a projected future thermal map to develop several heuristic policies for DTAS. We show that our proposed predictive policies are significantly better, in terms of minimizing average/peak temperature, reducing the dynamic thermal management overhead and improving other real-time features, than existing DTAS schedulers, making them highly suitable for heuristically guiding thermal aware task allocation and scheduling.**

***Index Terms*—Dynamic thermal aware scheduling (DTAS), event-driven, multiprocessor, predictive policy, thermal map.**

I. Introduction

THE COMPLEXITIES of modern processors are increasing year on year, as is the number of cores on a single chip and the transistor density. As such, power and thermal effects will remain a major issue in microprocessor design for the foreseeable future. Moreover, the decreasing gate lengths, oxide thicknesses, and threshold voltages in current and future manufacturing processes are likely to further aggravate the power-thermal side effects, as reducing the feature size means that leakage power becomes a more significant component in the total power consumption. Minimizing the power-thermal side effects is a major issue which is attracting significant interest from both industry and academia. In fact, the International Technology Roadmap for Semiconductors (ITRS) predicts that system-level design techniques [2], particularly at the behavioral level, will play an increasingly more important role in minimizing power and thermal effects [3].

Power-thermal optimization and design methodologies, and especially dynamic thermal management (DTM) techniques, have become commonplace in current generation processors. For example, dynamic voltage and frequency scaling (DVFS) and power/clock gating technologies are used in state-of-the-art processors [e.g., Turbo Boost and Speed Step in Intel's Core i7 and power management in their single-chip cloud computing (SCC) platform [4]]. DTM typically uses on-chip digital thermal sensors (DTS) and additional hardware to trigger an interrupt handler in the operating system (OS) to limit device overheating. Current DTM techniques are reactive and have a relatively long response time (several hundreds of milliseconds) [5]–[10]. Predictive DTM is being investigated [5], [6], [11]. Predictive methods, such as ARMA in [5] and predictive dynamic thermal management (PDTM) in [6], were proposed to achieve a better thermal distribution for DTM by using Kalman filters and regressive training to estimate the temperature change in the next time interval, based on the readings from a set of thermal sensors. An agent-based power-distribution method, TAPE, was proposed in [11]. TAPE uses a local agent, with power/energy budget information, to manage DVFS and if necessary reallocate/migrate tasks. TAPE is an efficient DTM mechanism which takes advantage of a distributed control style, but lacks thermal behavior accuracy. TAPE, like the core-based model used in PDTM, oversimplifies the power-temperature relationship, and does not use the real thermal information or a multicore aware model-based thermal estimation. Yeo and Kim [10] proposed a DTM that can dynamically classify tasks into clusters based on the thermal profile from DTS. Tasks are allocated based on the characteristics of a cluster. However, this technique has an inaccurate thermal model and a high clustering overhead for large task sets. In fact, many researchers [6], [10], [12], [13] assume the temperature transient on multicore is a simple exponential relationship determined by a thermal time constant. Such a simplification is suitable for a single core, but there is no simple analytical solution for calculating the temperature transient on multicore. The current trend is that the number of DTSs per core is decreasing as the number of cores increases. For example, Intel's Core 2 Duo and AMD's Quad-Core Opteron processor have multiple DTSs on each core, while Intel's 48 core SCC only has one DTS for each core. However, in many-core systems, these sensor based approaches are likely to be even less practical. First, the

Manuscript received April 19, 2011; revised August 26, 2011 and November 11, 2011; accepted December 19, 2011. Date of current version May 18, 2012. A preliminary version of this paper was presented at the Asia and South Pacific Design Automation Conference in 2010 [1]. This paper was recommended by Associate Editor D. Atienza.

The authors are with the School of Computer Engineering, Nanyang Technological University, 639798, Singapore (e-mail: cuij0003@ntu.edu.sg; asdouglas@ntu.edu.sg).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCAD.2012.2183371

DTSs have accuracy problems, including being effected by manufacturing parameter variance, A/D errors, the inability to place the sensor at the location of the thermal hot-spot, and sensor noise [8], [14], [15]. In addition, the relatively long thermal latency of these systems makes them ineffective in predictive thermal management and even affects their usefulness for reactive thermal management. These problems are further compounded by the engineering impracticality of multiple sensors in many-core systems. Providing access to 1000s of sensors and providing distributed DVFS where each core has its own PLL and private clock network, and its own separate power island present a number of floor-planning and physical design challenges [7].

One possible way of overcoming some of these problems in many-core systems could be to use predictive thermal-aware task allocation and task migration [e.g., thermal-aware scheduling (TAS)] for core level thermal management in conjunction with reactive DTM globally. That is, use global DVFS as a safety net but perform thermal management using task scheduling and if necessary, migration.

Both TAS and task scheduling using energy/power objectives (or constraints) [9], [12], [16]–[19] are active areas of research. The energy/power budget, which is relatively easy to calculate, does not consider the chip's thermal constraints, and cannot be guaranteed to keep the system within the normal operating temperature range. As such, the objectives of energy/power-aware scheduling deviate significantly from TAS. In addition, many of the proposed TAS implementations [13], [20]–[28] only consider static scheduling and search for an optimal solution to a static schedule under predefined power-thermal constraints using exhaustive searching, linear programming algorithms, synthesis methods, heuristic approximation, and so on.

Generally, uniprocessor TAS [18], [28] is much easier than that for multiprocessor systems since the power-thermal model for a single core can be relatively simple. Also, the temperature variance on a spatial scale and migration among cores need not be considered. In static TAS, the quality of the solution is the most important goal, and little attention is paid to the solution overhead because they run offline at design time. Many of these techniques only consider the dynamic power and ignore the contribution of the leakage (static) power. Leakage power is considered in [9] and [18]. Reference [18] is an offline algorithm which is only applicable to a single core, as there is no simple analytical solution for the complex differential equation set and the temperature-leakage relationship in the multicore case. While [9] considers leakage power, it assumes a constant leakage power and ignores the temperature leakage relationship.

The more complex problem of dynamic TAS (DTAS) has also been considered by [29]–[31]. Heuristic algorithms, such as the coolest-first and neighbor-aware [31] policies, provide a simple solution; however, the task allocation decision of these algorithms only depends on the current temperature of each core (i.e., look-current). An empirical model [29] was proposed that calculates the probability of accepting the newly arrived task, for each core, based on the historical temperature of each core (i.e., look-backward).

Very little has been published in the area of predictive (i.e., look-ahead) DTAS-based thermal management [1], [32], [33]. Lung *et al.* [32] used a predicted steady-stage temperature analysis, but ignored the task's temperature transient characteristic. Additionally, sorting tasks in descending power order may not be practical in a real system. The thermal model in [33] is computationally efficient, but is not very accurate due to several simplifications, particularly in ignoring core-to-core heat transfer and the temperature transient.

The main reason for the lack of good predictive thermal management techniques is their reliance on thermal estimation. To be effective, a thermal estimator must have a low computation overhead so that it can be integrated within the runtime scheduler. Current thermal estimation methods are used mostly for simulation and design automation purposes, and focus on accuracy rather than computational efficiency. Thermal simulators (i.e., TEMPEST [34], HotSpot [34], [35], TSIC [31], and those based on finite element methods) are mainly used in thermal-aware layout exploration and micro-architecture optimization. HotSpot is an open-source thermal model widely used in academic circles, which can also be used as a simulator for high level TAS [29]; however, its large computational overhead makes it unsuitable for use in real-time DTAS. An efficient software based (sensor free) temperature monitor has been proposed [36] that could replace HotSpot, but it lacks predictive estimation for scheduling purposes. Frequency-domain transforms and other optimizations have been used to accelerate the HotSpot simulation [37], [38], but the computational overhead is still very large.

In summary, there are currently a number of impediments to the use of TAS for thermal management. These include: 1) the lack of an effective fast thermal estimator for DTAS (or for fast high level simulation); 2) most current TAS policies are based on a non-saleable sensor approach, using look-current or look-backwards techniques; 3) current look-ahead techniques use long latency sensor data for coarse grained DTM and are not suited for the much finer reaction interval needed for TAS; and 4) the few TAS techniques that do make use of a thermal model do not consider the temperature-leakage power relationship.

In this paper, we present an event driven thermal estimation method suitable for DTAS, based on atomic power events and system-level events (i.e., task allocation, deallocation, context switch, migration, and so on). A fast look-up table (LUT) based method, presented in [1], is extended to include a leakage power model. The LUT based method reduces the computational overhead of thermal estimation, while still providing good accuracy. We also propose a technique for using occasional sensor based calibration to eliminate the effects of long term temperature drift. The new contributions are as follows.

1) A fast and accurate event-driven thermal estimator is proposed with an accuracy comparable to that of HotSpot, but with a computational overhead three orders of magnitude less, making it suitable for DTAS or fast high level thermal simulation.
2) A temperature-leakage power relationship is added to the LUT-based model.
3) Long term temperature calibration based on existing high latency DTS is added to eliminate thermal drift.

4) A technique for improving the scalability of the thermal estimator for large many-core systems is proposed.
5) Four new look-ahead DTAS polices are developed based on a predicted thermal map derived using our fast event-driven thermal estimation. These new policies are compared to existing DTAS policies from the literature.

This paper is organized as follows. The concept of an "event" is defined in Section II. Section III introduces the basic power/thermal model for high level thermal estimation and leakage power evaluation. Our event driven thermal estimator and the corresponding calibration of the leakage power model is detailed in Section IV. Heuristic look-ahead DTAS policies based upon a future thermal map are described in Section V. Section VI presents and discusses the experimental results. Section VII presents our conclusions.

II. Power Event Model

A many-core [multicore, chip multiprocessor (CMP) or multiprocessor system-on-chip (MPSoC)] device can be abstracted as a set of rectangular regions for high level thermal simulation or optimization [21], [24], [25], [31]. These regions could represent a complete processor core (or individual modules) depending on the desired granularity. Each region is then considered to have a single power input and a single temperature characteristic. This core-level abstraction is sufficient for system level optimization [24], [25], because:

1) large short-time power variations rarely happen within an application/task, and usually the power input only experiences a significant change after the occurrence of a high level event (e.g., task allocation and deallocation, a context switch on a core, preemption, exception handling or voltage/frequency scaling, and so on) [24], [25], [31];
2) rapid power variance in the short term (at micro-second intervals) is not able to introduce a large change in core temperature due to the thermal mass of the core [31].

As a result, a continuous power profile can be simplified to a sequence of constant power events. The definition of a power event used in this paper is as follows.

Definition 1 (Power Event): A power event is associated with the (relatively instantaneous) increase or decrease in power generated by a core. The power profile is described by, and converted to, a sequence of these power events.

Task allocation and deallocation can be seen as power events, with an instantaneous power change at the beginning and end of the task. Preemption can also be treated as a decrease in power followed by a power increase, while DVFS can be regarded as a power increase when the clock frequency or voltage rises and a power decrease when they are scaled down. Thus, instead of using a time driven methodology which updates the core temperature at fixed time intervals, like HotSpot, we use an event driven thermal model to update the core temperature only when an event occurs. These events are managed and captured dynamically by the scheduler.

Power events can be determined by profiling the power consumption of an application (or task). Due to the thermal mass of the system, a very fine-grained power profile is not necessary for TAS. Instead, we use the average power consumption over a larger time interval to describe the power behavior for high level optimization. The power events are then the critical points where large power variations can induce significant temperature changes in the core. We define a threshold value, P_{AE}, to determine power events, with a smaller P_{AE} resulting in a finer-grained power profile. A more coarse grained power profile is suitable for DTAS, while a finer granularity could be used for thermal simulation. Fig. 1 shows two different power profiles decomposed into atomic events for different P_{AE} values. The solid lines represent the original power profile and the dash-dot lines show the power profile approximated as power events.

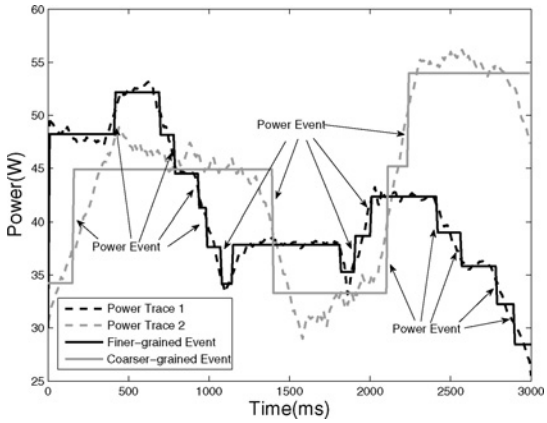

Fig. 1. Power event profiling.

Power profiling can be carried out statically (i.e., offline) or dynamically (while the application is running). Offline profiling is the most common approach for simulation and scheduling. Accurate power profiles are obtained using an architecture-specific simulator (e.g., SimpleScalar [39] and Wattch [40]). Many of the previous TAS algorithms described in the literature assume that the power consumption of each task is profiled by a simulator before run-time and then stored for later use [22], [25]. Dynamic, or online, profiling using performance counters [26], [35], [41] has also been shown to be feasible for high level optimization. As profiling is not the focus of this paper, we only consider static profiling and do not explore dynamic profiling further.

III. Power-Thermal Model

The abstracted set of rectangular regions representing the individual cores in a many-core system can be represented as a thermal RC network, described by a set of linear ordinary differential equations (LODE). This thermal RC model is a good choice for high level estimation due to the accuracy and efficiency of thermal estimation [34]. HotSpot solves the same thermal RC model at each simulation point, resulting in a large computational overhead. For a many-core system, consisting of cores on a silicon layer with a heat spreader and heatsink, the relationship between power (current), temperature (voltage), the thermal resistances, and capacitances for each node can be expressed as follows:

$$C_i \frac{dT_i}{dt} = p_i + \sum_{j \in Path} \frac{(T_i - T_j)}{R_j} \quad i \in \textit{power source}$$
$$C_i \frac{dT_i}{dt} = \sum_{j \in Path} \frac{(T_i - T_j)}{R_j} \quad i \notin \textit{power source} \qquad (1)$$

where C_i is the thermal capacitance of node i, T_i is the node temperature as a function of the time t, p_i denotes the instantaneous power input at node i, *Path* is the set of all thermal conduction paths that connect with node i, R_j denotes the resistance of each thermal conduction path in *Path*, and T_j is the temperature of the adjacent node in each thermal conduction path. For a processor, the power input is only applied to the silicon layer.

We have previously described a LUT-based event driven methodology which eliminates the need for solving the LODEs at a fine time interval [1]. We assume that the thermal resistances and capacitances are constant.[1] The methodology in [1] also assumed that the power input was constant (i.e., a constant leakage power). Unfortunately, it is not possible to consider leakage power to be a constant and thus ignore the interaction between leakage power and temperature.

Leakage power has two main components [43], [44]: the subthreshold leakage and the gate leakage. There are several important observations relating to leakage power indicating that leakage power should not be ignored in any sensor-less, or reduced sensor, thermal estimation.

1) The subthreshold leakage current can be approximated by $A \cdot exp(-B/T)$, where A and B are constants, and T is the temperature [43]. This exponential function can be replaced by a piece-wise linear (PWL) function [44] allowing the fast estimation of leakage power.
2) The gate leakage is relatively smaller than the subthreshold leakage, and can usually be ignored for high level optimization [43].
3) A transistor consumes leakage power even when idle (i.e., not switching). This leakage component is referred to as inactive leakage, P_{inactive} [44]. This component can be estimated using the PWL or exponential functions described earlier.
4) Dynamic power, induced by switching activity, contributes to the transistor gate temperature. This affects the leakage power. Leakage induced by dynamic power is referred to as active leakage, P_{active}. The ratio of leakage power to dynamic power does not vary significantly with activity [43]. Skadron *et al.* [42] gave a ratio between static and dynamic power in the range from 30% to 38%, based upon benchmarking on different architectures.

Thus, we have developed a basic leakage power model, with low computational overhead, which can be added to our existing LUT-based thermal estimator. The total leakage power consists of the inactive and active leakage components, as follows:

$$P_{\text{static}} = P_{\text{active}} + P_{\text{inactive}}. \tag{2}$$

A simple and efficient n-PWL function was proposed in [44] for estimating the inactive leakage power-temperature relationship. We have found that the simple 1-PWL implementation gives sufficient accuracy for high level thermal modeling, described by

$$P_{\text{inactive}}(T) = F_{\text{tech}} S_{\text{core}} (\alpha T + \beta) \cdot V \tag{3}$$

[1]HotSpot uses the same assumption [42] when the chip temperature varies over a small range, i.e., a range of less than 50 °C.

where α, β can be predetermined by HSPICE simulation and experimentation; F_{tech} is the leakage current per unit area, and depends on the manufacturing technology, design style and supply voltage V; and S_{core} denotes the area of the core.

For active leakage, [42] gives an empirical equation that shows the ratio between the dynamic power and the static leakage induced by signal activities, as follows:

$$P_{\text{active}} = P_{\text{dynamic}} \cdot \left(\frac{R_0}{V_0 T_0^2} e^{\frac{B_{\text{tech}}}{T_0}} \cdot V T^2 \cdot e^{\frac{-B_{\text{tech}}}{T}}\right) \tag{4}$$

where T_0 is the ambient temperature, R_0 is the ratio between dynamic power and static leakage at T_0 and nominal voltage V_0, and B_{tech} is a process technology constant that depends on the ratio between the threshold voltage and the subthreshold slope, computed using the leakage current and saturation drive current numbers from ITRS 2001. The total static power is then the sum of the inactive and active leakage components, both of which are temperature dependent.

IV. Event Driven Estimation

In this section, we present the details of our updated event driven thermal estimator. This approach is used to update the system thermal map without using thermal sensors and shows how to predict the temperature of each core at the next time interval for guiding a thermal aware scheduler.

A. Look-Up Table Based Approach

The equivalent RC network for the many-core (multicore) system is determined by analyzing the relationships between adjacent core regions, with the thermal resistances and capacitances being obtained from research experiments or technical documents [34], [45]. These physical parameters are independent of our methodology. We then build several LUTs which reflect the temperature transient in each core. These LUTs are built off-line by solving (1), under the assumption that a unit-step power (1 W) is injected into one of the cores. The computation of the LUTs, including solving the RC network, is carried out off-line, using tools such as MATLAB or Maple, and as such, the efficiency of building the LUTs is unimportant. At this stage, the number of distinct cores determines the number of LUTs that need to be generated. We will, in a later section, present a technique for reducing the number of LUTs using thermal symmetry.

Table I gives an example of the LUT structure for a 2×2 CMP [1 W of power is injected at Core(0, 0)]. In each time interval, the relative temperature increment of each core is recorded, with the last row of the table indicating the steady temperature of each core. It should be noted that even though we have only shown an example of a 2×2 CMP (due to space limitations), this method can be easily extended to many-core systems. In a many-core scenario, the overhead associated with solving the thermal RC network increases exponentially with the number of nodes. Our LUT-based approach eliminates the need to solve the RC network at run-time, at the expense of increased memory needed to store the pre-built LUTs. Table I also shows the use of a non-uniform time interval to reduce the storage requirements for the prebuilt LUTs. A finer time

TABLE I
LOOK-UP TABLE FOR 2 × 2 CMP

TDL A	Core(0, 0)	Core(0, 1)	Core(1, 0)	Core(1, 1)
0 ms	0	0	0	0
10 ms	0.1553	0.0007	0.0007	0.0000
20 ms	0.1788	0.0012	0.0012	0.0000
30 ms	0.1844	0.0016	0.0016	0.0001
40 ms	0.1880	0.0019	0.0019	0.0001
50 ms	0.1912	0.0021	0.0021	0.0001
...	...	...	...	...
500 ms	0.2013	0.0648	0.0648	0.0446
520 ms	0.2014	0.0649	0.0649	0.0447
540 ms	0.2015	0.0650	0.0650	0.0448
...	...	...	...	...
1000 ms	0.3233	0.0854	0.0854	0.0639
1050 ms	0.3235	0.0856	0.0856	0.0640
...	...	...	...	...
2000 ms	0.3504	0.1116	0.1116	0.0891
2100 ms	0.3506	0.1118	0.1118	0.0893
...	...	...	...	...
Steady	0.3819	0.1428	0.1428	0.1200

step is used initially (we use 10 ms),[2] which is increased as the temperature gets closer to the steady-state value. This is because the temperature profile of the heating/cooling stage shows a more rapid initial change, with the rate of change in temperature slowing with time.

An error in the thermal estimate will occur if the time instant, associated with the occurrence of an atomic power event, lies between consecutive rows. While it is possible to use linear interpolation or some other interpolation to get the temperature between time points, we have found that the results obtained from just rounding to the nearest temperature value are accurate enough. If memory is a concern, and a coarse-grained time interval is used, the accuracy will degrade and interpolation should be considered.

B. Thermally Different Locations

We make use of thermal symmetry to reduce the computational overhead when determining the thermal profile of each core. A thermally different location (TDL) is defined [1] for sets of cores which are symmetrical in their relative location, and thus have similar thermal characteristics. The important implication of TDL is that if a task is allocated to a core in a TDL, it will produce the same thermal effects and distribution as it would when allocated to another core with the same TDL. The TDLs for a number of abstracted multicore systems are given in Fig. 2. Here, the characters in the core regions denote the TDLs. Cores with the same TDL have identical architectures and are symmetrical in layout.

We implement two functions, *tdl*() and *trans*(,), to simplify the thermal calculation in a multicore scenario by using the core (module) symmetry for a specific architecture layout. The function *tdl*(location) is used to determine the LUT for the TDL corresponding to the core location, while the function *trans*(tempInc,location) transforms the temperature increment values based on the core location. A transformation of the temperature increment values is needed to cater for the difference between the core location where the event occurs and the core location where the power was injected when the LUT was built. This means that one LUT can be used for symmetric cores (modules), thus significantly reducing the LUT storage requirement. Coordinate transformation is relatively simple with eight possible operations, being 0: no change, 1: mid-x mirroring, 2: mid-y mirroring, 3: principal diagonal mirroring, 4: secondary diagonal mirroring, 5: center point mirroring, 6: clockwise rotation, and 7: counter-clockwise rotation. A more complete discussion of the *tdl*() and *trans*(,) functions, along with an explanatory example, can be found in [1].

[2]A reasonable time interval is the time interval between two consecutive scheduling rounds of the OS kernel, i.e., a OS timer tick.

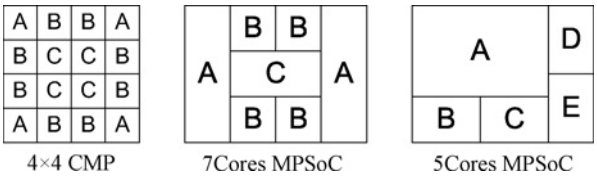

Fig. 2. Some possible CMP, MPSoC layouts and their TDLs.

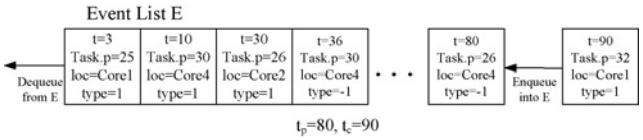

Fig. 3. Data structure related to events.

C. Updating and Estimating Online

Atomic power events *ae* are globally recorded for all cores on chip, according to their sequence, by an event list $\mathbb{E}$ in the OS kernel. $\mathbb{E}$ is a queue, as shown in Fig. 3, where new events are enqueued onto the tail of $\mathbb{E}$ and older ones are dequeued from the head of $\mathbb{E}$. Each element tuple $ae(t, P, loc)$ in $\mathbb{E}$ denotes one atomic power event with several properties: t is the time instant of the event occurrence, P indicates the power (a positive value indicates an increase in power), *loc* indicates the core location where the power event takes place. In addition, two global variables t_c and t_p represent the current time instant and the time instant of the previous event occurrence (i.e., the last thermal map update).

The thermal map, which records each core's temperature, is updated each time an atomic power event occurs. The current thermal map T_{t_c} at time t_c can be calculated based on the previous thermal map T_{t_p} at t_p. That is, the current thermal map is obtained by adding the temperature increment of each core in the interval Δt to the previous thermal map, as follows:

$$T_{t_c} = T_{t_p} + \Delta T_{\Delta t = t_c - t_p} \tag{5}$$

where T_{t_c}, T_{t_p}, and $\Delta T_{\Delta t = t_c - t_p}$ are there N-element vectors which denote the current thermal map, the previous thermal map, and the temperature increment map of the N cores, respectively. The calculation of the temperature increment in the interval from t_p to t_c is given as follows:

$$\Delta T_{\Delta t = t_c - t_p} = \sum_{ae \in \mathbb{E}} ae.P \times trans(\Re_{t_c - ae.t}^{tdl(ae.loc)} - \Re_{t_p - ae.t}^{tdl(ae.loc)}, ae.loc) \tag{6}$$

where $\Re^{tdl(ae.loc)}_{t_c-ae.t}$ represents the row corresponding to the time instant $t_c - ae.t$ in the LUT pointed to by the core location $ae.loc$. Similarly, $\Re^{tdl(ae.loc)}_{t_p-ae.t}$ denotes the row corresponding to the time instant $t_p - ae.t$ in the same LUT. The functions $tdl()$ and $trans(,)$ used to simplify the calculation due to core symmetry were described earlier. In this case, the function $trans(,)$ transforms the temperature increment values $\Re^{tdl(ae.loc)}_{t_c-ae.t} - \Re^{tdl(ae.loc)}_{t_p-ae.t}$ associated with the first argument, based on the core location passed as the second argument. Thus, the temperature increment, described by (6), can easily be obtained by indexing two rows of the LUT, subtracting them, and then performing a simple transformation.

This observation is based upon the characteristics of linear time-invariant (LTI) systems where the overall response can be determined by accumulating the individual responses, and thus (6) can be expressed by the following theorem, the proof of which has been presented in [1].

Theorem 1: The temperature increment at one core can be treated as the accumulation of every individual temperature increment at this core induced by each atomic power event. The thermal distribution induced by atomic power events adheres to the superposition principle when leakage power is constant (i.e., not affected by temperature).

When an event occurs, in addition to updating the thermal map, the event list also needs to be updated. Older events, which no longer induce a temperature change, should be dequeued from $\mathbb{E}$. The criterion $t_c - ae.t > Steady$ decides which events should be dequeued, where *Steady* is the time constant from initial state to steady state and its value is listed in the last row of the LUT. Deleting older events, which have reached steady state, shortens the event list and improves the performance of event driven estimation.

D. Fast Estimation With Leakage Power Calibration

In practice, as mentioned in Section III, leakage power is a function of temperature and cannot be ignored. If the leakage power varies with temperature, the previous simplified LTI model converts into a non-linear model whose solution depends on an iterative procedure [34], [43] which is not computationally efficient. To get an accurate transient temperature profile, we have added the leakage power model described by (2)–(4) to HotSpot.[3] We then use an iterative procedure, where the leakage power is updated according to the current temperature and added to the dynamic power as a new power input to HotSpot for the next iteration. This iterative process, where the leakage power varies with both dynamic power and core temperature, needs to be performed at a reasonably fine time interval; otherwise the temperature errors become significant. This increases the computational overhead of HotSpot significantly.

To eliminate this iterative overhead, we propose an empirical calibration factor r, a function of t, P and T, that adjusts the thermal response (the LUT values in our case) to account for the temperature leakage power dependence. This converts the non-linear problem into an approximately linear one, with a small error. For a given architecture, and for various P, T pairs, we can plot the temperature profiles over t for both the non-temperature dependent leakage and the temperature dependent leakage profiles. The ratio between the two profiles is r. After examining the temperature difference profiles for a number of processor architectures, such as Alpha 21264, ARM7TDMI, PowerPC 405, and so on, we found r to be similar to the simplified expression for annealing, $T(t) = \frac{T_0}{1+kt}$. We make the observation that for any power event the ratio between the non-temperature dependent leakage-temperature profile and the temperature dependent profile will initially be one (both temperature responses start at the same initial temperature) and at steady state will exhibit a steady state offset S. This offset will be a function of temperature T and power P. Thus, we determine the calibration factor r as follows:

$$r(t, P, T) = S - \frac{S-1}{k(t)+1} \tag{7}$$

where k is the annealing rate and has a hyperbolic characteristic

$$k = A + Bt^{-\gamma}. \tag{8}$$

The calibration factor at the current time instant is based on the thermal characteristics at the previous instant. Thus we introduce a variable change to modify (7) as

$$r(t, P, T) = S - \frac{S-1}{(A + Bt^{-\gamma})(t-1)+1} \tag{9}$$

where S, A, and B are functions of both the temperature T and the power P, and are related to the leakage parameters in Section III. The calibration factor r thus depends on the power input $ae.P$, the core temperature $T^{ae.loc}_{t_p}$ at time t_p, and the time interval $\Delta t = t_c - t_p$ between sequential power events. The core location $ae.loc$ only introduces a very small error and can be ignored. γ is related to the leakage parameters, and is in the range $\gamma = 0.5$–1.[4] To minimize the overhead without significantly sacrificing accuracy, we let $\gamma = 1$. The simplified calibration factor is then given as follows:

$$r(t, P, T) = S - \frac{S-1}{At - \frac{B}{t} - A + B + 1}. \tag{10}$$

Plotting S, A, B versus temperature T and power P (Fig. 4) shows that the surfaces can be approximated as planes (with minimal error). Thus, for a given architecture, it is a simple matter to determine spot values for S, A, B at three different P and T points and thus define the individual planes. This is done offline. For example, for the 4×4 CMP layout, using $F_{\text{tech}} = 0.048$ A/mm^2, $S_{\text{core}} = 16\,\text{mm}^2, \alpha = 0.04, \beta = 0.2, B_{\text{tech}} = 1.3\,\text{V}, V = 1.3$ V [42], [43] in the leakage model of Section III, we can determine a 3×3 leakage parameter matrix L offline, where $L \cdot [T\ P\ 1]^T = [S\ A\ B]^T$, as follows:

$$L = \begin{bmatrix} 0.0013 & 1.7426 \times 10^{-4} & 1.0061 \\ -7.03 \times 10^{-7} & -6.0070 \times 10^{-6} & 0.0033 \\ -0.0035 & -2.8987 \times 10^{-4} & 1.7552 \end{bmatrix}.$$

[3] A leakage model has been added to HotSpot V5.0 [46], but is not used as transient analysis and comments in the code suggest accuracy issues.

[4] The γ value is determined offline by simulation, for a specific processor layout. For the Alpha, $\gamma = 0.752$; for the 4×4 CMP, $\gamma = 0.823$.

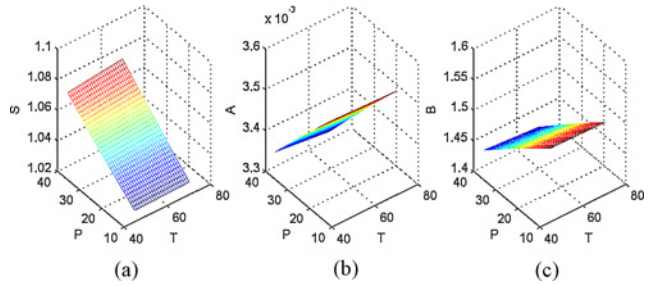

Fig. 4. Plots of S, A, B versus power and temperature for a 4 × 4 CMP. (a) S plot. (b) A plot. (c) B plot.

The heating and cooling stages are no longer symmetrical. We apply an annealing factor ($\sigma = 0.99$–1) to account for the difference between heating and cooling. We use: if $ae.P < 0$, then $\sigma = 0.995$, else $\sigma = 1$.

Using this method to calibrate the LUT requires a change to the temperature increment calculation (6), as follows:

$$\Delta T_{\Delta t = t_c - t_p} = \sum_{ae \in E} \sigma \times ae.P \times trans(\Re^{tdl(ae.loc)}_{t_c - ae.t} \cdot r(t_c - ae.t, ae.P, T^{ae.loc}_{t_p}) - \Re^{tdl(ae.loc)}_{t_p - ae.t} \cdot r(t_p - ae.t, ae.P, T^{ae.loc}_{t_p}), ae.loc). \quad (11)$$

The two calibration factors are determined at the time instances $t_c - ae.t$ and $t_p - ae.t$, corresponding to the two fetched rows of the LUT. The S, A, B parameters are calculated once only, based on the core temperature $T^{ae.loc}_{t_p}$ and the power input $ae.P$ at the previous event instant t_p.

While this approach simplifies the calculation of the temperature leakage power dependency, it does have the potential to introduce errors into the temperature calculation, such as in a thermal runaway scenario (very high temperature and power combination) where the S, A, B plots can no longer be approximated by plane surfaces. However, we would expect a task's DTAS prediction to indicate that the temperature threshold for the core would be exceeded and it would not be allocated. DTM would still trigger if there was an inappropriate DTAS allocation.

E. Algorithms for Monitoring and Prediction

At each power event, Algorithm 1 is invoked by the OS scheduler to update the thermal map and the event list. Algorithm 1 takes the previous thermal map T_{t_p}, the event list $\mathbb{E}$, the current time instant t_c, the previous time instant t_p, and the newly arrived tasks ae_{new} as inputs and returns the current thermal map T_{t_c}. First, the algorithm traverses the event list $\mathbb{E}$ and for each power event ae calculates the two calibration factors r_{t_c} and r_{t_p} and then the temperature increment induced by the event [as in (11)]. The temperature increments induced by each event in $\mathbb{E}$ are accumulated in ΔT. The current thermal map, T_{t_c}, is then obtained by adding ΔT to the last thermal map T_{t_p}. After updating the thermal map, older events are deleted from $\mathbb{E}$ based on the criterion: $t_c - ae.t > Steady$. Finally, newly arrived events ae_{new} are enqueued and the last time interval t_p is updated to t_c.

Equations (5) and (6) can also be used as an estimator for a future thermal map at the next time instant t_n. Algorithm 2 is similar to Algorithm 1, except that we use current thermal map

Algorithm 1 Event Driven Thermal Map Monitoring

Input: T_{t_p}, $\mathbb{E}$, t_c, t_p, ae_{new}
Output: T_{t_c}

```
begin
    ΔT = 0;
    for each ae in 𝔼 do
        index two rows ℜ^{tdl(ae.loc)}_{t_p−ae.t} and ℜ^{tdl(ae.loc)}_{t_c−ae.t} from LUT;
        [S A B]′ = L · [T^{ae.loc}_{t_p} ae.P 1]′;
        r_{t_c} = r(t_c − ae.t, S, A, B), r_{t_p} = r(t_p − ae.t, S, A, B);
        inc = trans(ℜ^{tdl(ae.loc)}_{t_c−ae.t} · r_{t_c} − ℜ^{tdl(ae.loc)}_{t_p−ae.t} · r_{t_p}, ae.loc);
        ΔT = ΔT + σ × ae.P × inc;
    end
    T_{t_c} = T_{t_p} + ΔT;
    for each ae in 𝔼 do
        if (t_c − ae.t > Steady) then
            dequeue ae from 𝔼;
        end
    end
    enqueue ae_new into 𝔼;
    t_p = t_c;
    return T_{t_c};
end
```

T_{t_c}, $\mathbb{E}$, t_n and t_c as the four inputs to produce an estimate of the thermal map T_{t_n}, at the next (future) time instant. The current thermal map can come from Algorithm 1 or the readings from thermal sensors. In our simulated system, where task arrival is unknown (but can only occur relative to an OS timer tick), the event horizon is set to the next OS timer tick. However, the time interval between tn and tc could be any reasonable value decided by users. If the incoming task set was well known/defined, this interval could be determined based on the anticipated task arrival times, and could be several OS timer ticks. Predicting the future thermal map is useful for guiding a scheduler, and allows various heuristic DTAS algorithms to be developed. That way, we could avoid allocating a task which would cause a core to overheat (or does not allow the core to cool sufficiently) by favoring a core with a smaller temperature increment (or a larger temperature decrement).

F. Thermal Calibration

Our proposed sensor free thermal estimator acts like an open-loop system as each event point update does not use any temperature feedback from the real chip itself, and is just based on the last thermal map. Additionally, it does not account for variations in individual core characteristics[5] due to process variations, and so on. While this could result in accumulated errors in the long term, our experiments have shown that the method is suitable for DTAS and is accurate enough for medium term thermal estimation. The reason for this minimal error is as follows.

1) Each power event introduces only a small error in the leakage calibration stage [as (11) provides a good approximation for the effect of leakage power].

[5]It should be noted that DTS also suffers from similar problems.

Algorithm 2 Event Driven Thermal Prediction

Input: T_{t_c}, $\mathbb{E}$, t_n, t_c
Output: T_{t_n}

begin
  $\Delta T = 0$;
  for *each ae in* $\mathbb{E}$ **do**
    index two rows $\Re^{tdl(ae.loc)}_{t_c - ae.t}$ and $\Re^{tdl(ae.loc)}_{t_n - ae.t}$ from LUT;
    $[S\ A\ B]' = L \cdot [T^{ae.loc}_{t_c}\ ae.P\ 1]'$;
    $r_{t_n} = r(t_n - ae.t, S, A, B)$, $r_{t_c} = r(t_c - ae.t, S, A, B)$;
    $inc = trans(\Re^{tdl(ae.loc)}_{t_n - ae.t} \cdot r_{t_n} - \Re^{tdl(ae.loc)}_{t_c - ae.t} \cdot r_{t_c},\ ae.loc)$;
    $\Delta T = \Delta T + \sigma \times ae.P \times inc$;
  end
  $T_{t_n} = T_{t_c} + \Delta T$;
  return T_{t_n};
end

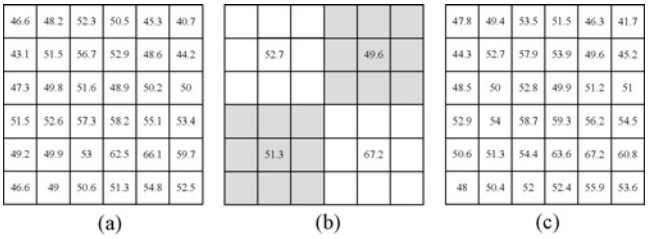

Fig. 5. Example of calibration using real temperature readings. (a) Estimated thermal map before calibration. (b) DTS readings. (c) Thermal map after calibration.

2) The errors introduced by an increase in power tend to counteract those introduced by a decrease in power.

To eliminate any long-term temperature drift, we propose using the on-chip DTS for coarse grained temperature calibration. These sensors are not suitable for fine grained thermal estimation due to their relatively slow access times. For multicore systems with individual sensors on each core, it is relatively easy to use the sensor information to directly update the thermal map. However in many-core systems, we would suggest that a single DTS, representative of a core in a cluster, be used to provide long term temperature stability. The reading from this sensor would be used to adjust the individual core temperature entries in the thermal map based on the difference between the current thermal map temperature and the measured temperature, as shown in Fig. 5. Other techniques, such as [15], could be used if additional accuracy of the thermal map calibration is required.

G. Scalability of Our Approach

The event-driven thermal estimation method, running as part of the scheduler in a centralized OS, is only suitable for current CMP or small many-core systems. If the number of cores becomes large (e.g., several hundreds or even thousands as predicted), then a fully centralized algorithm will not be effective, as the event list will grow with an increase in the number of cores. To overcome this problem, we propose that the event-driven algorithm be distributed to a single core in a subset of cores. Fig. 6 shows a basic framework for a distributed version of our event-driven method, for a subset cluster size of one. For each core subset, we build an event list to replace the global event list. Similarly, the global LUT is no longer useful, and instead each core subset stores a LUT relevant to its location. In Fig. 6, this is equivalent to a single transformed table, specific to the core's TDL, and as such the transformations of Section IV-B and in (6) and (11) are not needed. Therefore, each core subset only needs to calculate the temperature increments, calibrated by r, induced by its own power events. The final step uses a single designated core (globally) to accumulate the results from all the cores to complete the thermal estimation. The only significant overhead induced by this distributed version is the communication necessary for accumulating the individual results from each core. It should be noted that the scheduler will still need to control task allocation globally after obtaining the overall thermal map.

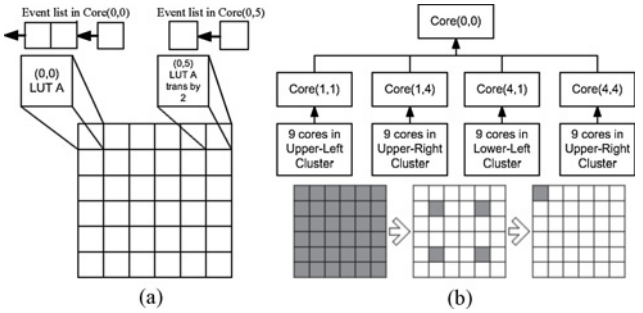

Fig. 6. Distributed version of our algorithms. (a) Basic framework for distributed version. (b) Optimization of hierarchy collection.

V. Heuristic Task Allocation

If the OS kernel can estimate the temperature, better heuristic task allocation methods could be proposed to guide thermally aware DTAS. We propose several predictive task allocation policies based on our event driven thermal estimator.

A. Future Coolest First

Algorithm 2 allows the prediction of the future thermal map at the next time interval. Future Coolest First simply finds the coolest idle core in the future thermal map and allocates the new task to that core. The rationale is that the coolest core is the most likely to be able to accommodate the power consumption of the new task and thus avoid overheating. This heuristic policy is very simple and efficient, being similar to coolest first [31] except that it uses the predicted temperature.

B. Future Neighbor Aware

Future neighbor aware, similar to neighbor aware [31], considers the future temperature of cores in the neighborhood of the idle candidate. A thermal weight T_w is calculated as follows:

$$T_w = a_1 T + a_2 T_{an} + a_3 T_{dn} + \frac{a_4}{N_{fe}} + \frac{a_5}{N_{ic}} \tag{12}$$

where T is the temperature of the candidate core, T_{an} is the average temperature of immediately adjacent cores, T_{dn} is the average temperature of the diagonally adjacent neighbors, N_{fe} is the number of free edges of the candidate core and N_{ic} is the number of idle cores in the neighborhood, a_1, a_2, a_3, a_4 and a_5 are predefined constants to adjust the importance of each factor. Then, the idle core with the smallest weight indicates the best candidate for task allocation.

C. Future Task Aware

This policy considers both the temporal and spatial scales for thermal distribution. For each power task, we define a metric R_{TT} relating temperature and the remaining task runtime, as follows:

$$R_{TT} = T(\frac{e}{t_n - t_s} - 1) \tag{13}$$

where T is the is the temperature of the candidate core, e is the execution time, t_s is the task start time and t_n is the prediction time instant, and where $R_{TT} = 0$ for an idle core. For each idle candidate, we then calculate the weight as follows:

$$T_w = \frac{a_1}{N_{an}} \sum_{core \in an} R_{TT_{an}} + \frac{a_2}{N_{dn}} \sum_{core \in dn} R_{TT_{dn}} \tag{14}$$

where *an* and *dn* denote the immediately adjacent core set and the diagonally adjacent core set, respectively, N_{an} and N_{dn} are the number of cores in the two sets. a_1 and a_2 are predefined constants. Intuitively, we allocate the task to the core with the smallest weight (i.e., the smallest average R_{TT} around the neighborhood). The rationale is that the temperature change at the start of a power task is rapid, and slows down over time. Also, the smaller the remaining execution time, the shorter the interval till the core becomes idle.

D. Future Temperature Trend

For each new task, we classify idle cores into two sets based on the difference in the current and predicted temperature, as a temperature-increasing set ($Core_+$) and a temperature-decreasing set ($Core_-$). The weight for each set is as follows:

$$\begin{aligned} T_{w+} &= T \times a_+ \quad if\ core \in Core_+ \\ T_{w-} &= \frac{T}{a_-} \quad if\ core \in Core_- \end{aligned} \tag{15}$$

where a_+ is the temperature increment and a_- is the temperature decrement. For each set, we choose the core with the smallest weight, giving two possible candidates for task allocation: one is from $Core_+$ and the other is from $Core_-$. We randomly allocate the task to one of these cores. The rational here is that a smaller temperature increment (or a larger temperature decrement) in the next time interval should be better. This policy takes advantage of both the current and the future thermal map.

VI. Experiment

In this section, we present the results of performance comparisons between our LUT-based event driven thermal estimator (both for thermal simulation and for DTAS), the HotSpot thermal simulator, and a number of fast thermal estimators [37], [38] and DTAS algorithms [6], [31], [32], [39] from the research literature. We use HotSpot in two ways: first, we use an iterative approach which accounts for the temperature leakage power dependence (referred to as variant-P) and second, the standard HotSpot approach which does not account for the temperature leakage power dependence (referred to as invariant-P). For invariant-P, the power task sets used by HotSpot are generated by Wattch at the same resolution as the HotSpot time increment interval. For variant-P the power input to HotSpot is the sum of the power input for invariant-P and the static power calculated from (1)–(3) at the previous time increment. All experiments are conducted on a PC with a 2 GHz Intel core 2 duo, and 2 GB of memory.

A. Validating the Event-Driven Estimator

To validate the accuracy and efficiency of our fast LUT-based thermal estimator, we examine its capabilities for uniprocessor core-level and micro-architectural level simulation. Core-level thermal estimation, where a single temperature reading is used to represent the core temperature, is suitable for high level thermal optimization, such as DTM or DTAS on a per core basis. At the micro-architectural level, thermal estimation or simulation must give a detailed thermal distribution on a per module basis, to aid thermal-aware floorplanning.

First, we examine the accuracy of our estimator by examining the effect, on the uniprocessor core-level temperature, of categorizing a continuous power trace as a sequence of atomic events. Fig. 7 shows the predicted core temperature, determined from an Alpha 21264 uniprocessor simulator using the power event profile from Fig. 1. The lower pair of traces represent a synthetic continuous power (solid line) and the atomic power using a $P_{\mathrm{AE}} = 3$ W (dashed line), while the upper trace represents the *variant-P* core temperature determined by HotSpot (iteratively called to include the leakage power/temperature relationship). The discrete points (dots) overlaying the HotSpot temperature profile represent the core temperature determined by our event driven thermal estimator at an atomic event in the lower (dashed) power trace. Fig. 7 shows that there is good agreement between the temperatures determined by the two techniques (with a maximum temperature error of 0.1%).

Next we examine a more realistic application, an MPEG2 decoder, running on the Alpha simulator over a much longer time period (47 s). Fig. 8 shows that our estimated temperature (including the leakage power/temperature relationship), based on atomic events, accurately matches the results from HotSpot, but at a much reduced computational overhead. The execution time for the HotSpot simulation is 11 372 ms (using a 10 ms interval), while the total execution time for our LUT-based thermal estimator is just 4.36 ms (547 atomic events with a P_{AE} of 1.2 W). This represents a runtime improvement of three orders of magnitude.

Our estimator has a significantly reduced computational overhead compared to HotSpot as we do not need to solve a large equation set in real-time. The algorithm complexity of our method is $\Theta(e \cdot N) = \Theta(r_{\mathrm{ae}} \cdot Steady \cdot N)$, where e is the average number of atomic power events in the event list, r_{ae} is the average arrival rate of power events, N is the number

TABLE II
ERROR AND OVERHEAD FOR SPEC CPU 2000

Scenario	Benchmark	gcc	gzip	bzip	lucas	mesa	parser	swim	vortex	mpegdec
	Execution runtime (s)	9.5	8	18.6	15.5	6.7	10.2	3.7	25.5	47
Scenario 1	Our temp. error (°C)	0.14	0.16	0.06	0.13	0.2	0.16	0.25	0.03	0.07
Core level	Our overhead (μs)	10	8	9	8	7	6	10	11	8
P_{AE} = 3 W	Our runtime (ms)	0.49	0.496	0.803	0.464	0.364	0.312	0.15	1.034	3.36
Variant-P	HotSpot runtime (ms)	2175	1923	4378	3879	1769	2472	955	6550	11 372
Scenario 2	Our temp. error (°C)	0.31	0.22	0.11	0.17	0.34	0.22	0.29	0.12	0.09
MA level	Our overhead (μs)	285	234	245	221	246	204	292	278	201
P_{AE} = 1.2 W	Our runtime (ms)	27.93	25.74	56.6	43.54	21.4	24.15	15.48	70.9	124.02
Variant-P	HotSpot runtime (ms)	77 185	62 709	145 672	121 375	52 955	80 143	29 171	197 447	386 190
	Our temp. error (°C)	0.145	0.164	0.082	0.132	0.159	0.178	0.251	0.034	0.069
Scenario 3	FATA temp. error (°C)	0.097	0.112	0.038	0.135	0.181	0.136	0.182	0.056	0.067
MA level	TMM temp. error (°C)	0.08	0.105	0.05	0.062	0.172	0.128	0.091	0.051	0.059
P_{AE} = 1.2 W	HotSpot runtime (ms)	51 869	43 679	101 357	84 893	36 488	55 952	20 119	135 238	255 755
Variant-P	Our runtime (ms)	18.94	18.12	37.92	30.15	16.79	20.17	9.32	47.35	87.29
	FATA runtime (ms)	519.6	487.7	1029.8	763.8	312.6	750.2	193	1274.4	2158.3
	TMM runtime (ms)	678	651.9	1535.7	1081.4	499.8	799.3	283	1788.9	3148.1

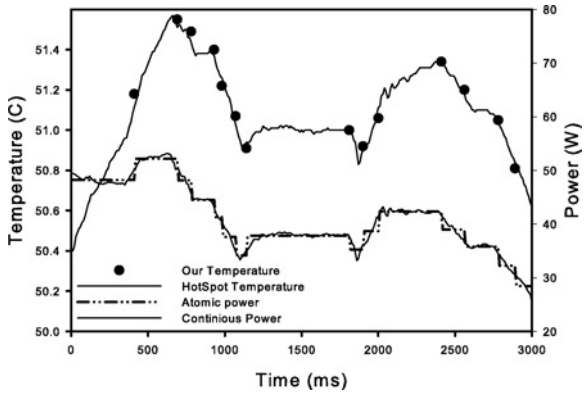

Fig. 7. Variant-P thermal simulation of a single core processor using a synthetic power input.

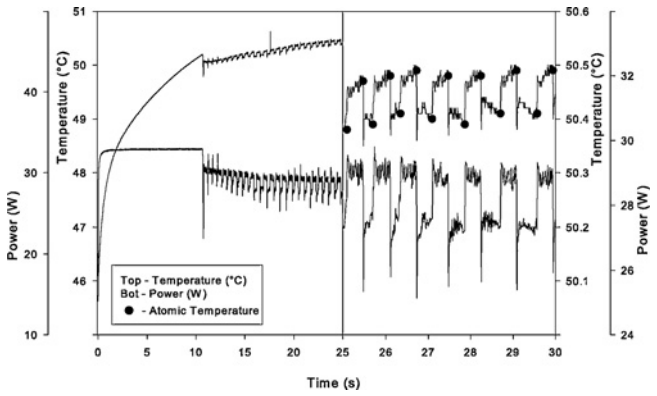

Fig. 8. Variant-P thermal simulation for a MPEG2 decoder at the core level.

of cores (modules) in the processor. Since N and *Steady* are a constant for a given processor, the algorithm complexity only depends on e and r_{ae}.

Last, we use the Alpha 21264 simulator to examine the suitability of our estimator for uniprocessor core-level and micro-architectural level simulation. This experiment consists of three separate scenarios.

In the first scenario, we perform a *variant-P* core level simulation (including the temperature/leakage power relationship) with a P_{ae} = 3 W. In this scenario, the HotSpot iteration time is set at 10 ms. The average temperature error (as absolute error relative to the HotSpot temperature), the average computational overhead for a single thermal map update, and the actual simulation runtime for each benchmark are presented in Table II. Table II shows that there is good temperature agreement with HotSpot, but with a three orders of magnitude reduction in the simulation runtime. It should be noted that the total simulation execution time for the benchmarks in Table II is approximately the product of the overhead for a thermal map update by the number of atomic events in the queue, over the whole simulation.

The second scenario is similar to the above scenario, except that we perform a 30-module micro-architectural level simulation with a P_{ae} = 1.2 W. Again, there is close agreement between our event driven method and the results from HotSpot with a three orders of magnitude reduction in simulation time. The thermal map overhead has increased relative to scenario 1 (above) because of the need to accumulate all the events on the 30 modules in the micro-architectural simulation.

In the third scenario, we examine the performance of our LUT based estimator by comparing it with some of the fast thermal estimators [37], [38] from the literature. FATA [37] implements an improved fourth-order Runge–Kutta solver with an adaptive step size. TMM [38] takes advantage of moment matching in the frequency domain, where temperature can be calculated as the convolution of the power input's response. Since FATA and TMM (and the original HotSpot) do not model the temperature/leakage power dependency for transient temperature estimation, we only consider the invariant-P case so as to provide a fair comparison. That is, we have removed the temperature/leakage power relationship described by (7)–(10) from our estimator. We have set the window size of TMM to 10 000. The average temperature error (as absolute error relative to the HotSpot temperature) and the actual simulation runtime for each benchmark are presented in Table II. Our thermal estimation has a 2000–3000× speedup compared to HotSpot, and is 20–40× faster than FATA and TMM while maintaining similar temperature errors.

These experiments show that it is feasible to estimate core/module temperature based on a power trace decomposed into atomic events. This demonstrates that only a few power events are sufficient and accurate enough to describe the temperature transient, thus suggesting that event driven estimation

is likely to be suitable for high level DTAS, DTM and fast thermal simulation in a multi/many core environment.

B. Event Driven Thermal Estimation for CMP Systems

We have developed a 4×4 CMP simulator which can update the thermal map and track atomic power events using our event driven approach. The thermal RC model and parameters used for each processor are the same as those used in the HotSpot simulations, and thus the LODE used to build the LUTs is identical to that used in HotSpot. The leakage parameter matrix, L, for the 4×4 CMP is the same as the one pre-calculated in Section IV-D. Both the LUTs and the leakage parameter matrices are calculated offline.

To test the performance of our proposed thermal estimator we generate a number of artificial power task sets based on power profiling of selected applications in SPEC CPU 2000. These power task sets are derived using SimpleScalar Alpha and Wattch, which are then converted into consecutive atomic power events. For example, the GCC benchmark can be converted into several tasks with an average power consumption in the range [15 W:30 W],[6] determined using a power threshold $P_{\text{ae}} = 5$ W, and with execution times in the range [10 ms:500 ms]. Our event driven approach is unrelated to the features of a task set, and is applicable to any power profile. We assume that the arrival time of the tasks is randomly distributed in the range [0 s:30 s]. Each task set contains around 200 power tasks (parts of the benchmarks) that are randomly distributed among the cores.

We use these task sets to examine the performance of our estimator, relative to HotSpot, on a 4×4 CMP. Here, we examine the contribution of the leakage power to the CMP core temperature by examining both the invariant-P (no temperature/leakage power dependence) and the variant-P (which accounts for the temperature/leakage power dependence) scenarios. In this experiment, the initial chip temperature is 45 °C, the HotSpot time increment is 100 μs, and DTM (e.g., DVFS, migration, clock gating, and so on) is not triggered. We simply observe whether our estimation of the thermal behavior is similar to that of HotSpot.

Fig. 9 shows the temperature estimation results for a 5000 ms task set subsection for selected cores in the CMP. The bottom solid line (grey) shows the *invariant-P* HotSpot temperature transient for cores (0, 0), (1, 1), (1, 3), and (3, 3) of the 4×4 CMP, while the upper solid line (black) shows the temperature profile for the HotSpot *variant-P* scenario (i.e., the complete leakage power model). The asterisks (*) represent our event driven estimation of the core temperature, at each update point where an atomic event occurs. Fig. 9 shows that there is a significant difference between the two temperature profiles determined by HotSpot. These results are not unexpected, and show that the temperature leakage power dependence cannot be ignored, thus validating the inclusion of a full leakage model into our event driven estimator. As such, we will not consider the HotSpot *invariant-P* model further. Fig. 9 also shows the accuracy of our proposed fast event driven estimation [the asterisks coincide with the HotSpot iterative leakage power (*variant-P*) model line], validating our calibration factor based approach described in Section IV-D.

[6]L2 cache power is not included.

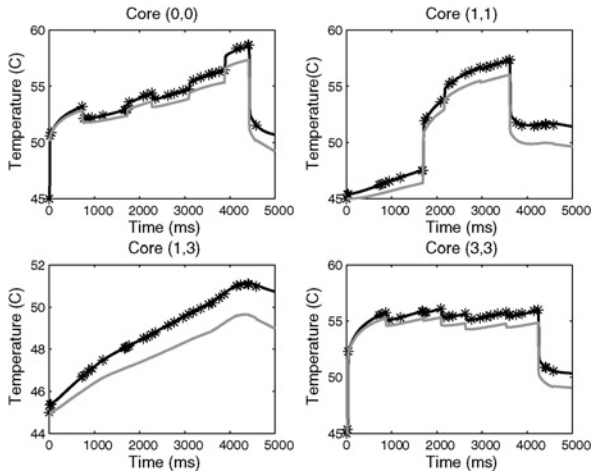

Fig. 9. Thermal estimation validation.

TABLE III
PERFORMANCE, AVERAGE ERROR, OVERHEAD COMPARISON

	1 ms		10 ms		100 ms	
	Ours	HotSpot	Ours	HotSpot	Ours	HotSpot
WC error (°C)	0.82	0.67	1.44	4.12	3.16	6.17
Ave error (°C)	0.65	0.43	0.89	1.58	1.78	2.66
Runtime (ms)	20.3	42 415	24.4	39 736	31.1	37 221
Memory (MB)	1.22	–	0.136	–	0.02	–

To determine the speedup of our event driven estimator, we compare our algorithm to HotSpot, using different time intervals when building the LUTs. To make the comparison fairer, we also vary the time increment interval for HotSpot. Table III compares the absolute worst case temperature error, the average temperature error, and the simulation run-time for the 30 s task sets described earlier. The worst case error is relative to the HotSpot temperature profile determined using a time increment interval of 100 μs. Table III shows that the error increases with the granularity of the time interval used to build the LUTs. The HotSpot error decreases with granularity, but as the HotSpot interval approaches the task execution time, the error increases significantly. Note that a finer granularity in the LUT also increases the memory requirements for our event driven estimator. In most cases, a 10 ms time interval is a good choice as this gives good accuracy, while keeping the memory usage relatively low. The runtime increases with granularity for the LUT-based estimator because additional interpolation is required. The overhead for HotSpot decreases with increasing increment interval. Table III also shows that our event driven estimator has a significantly reduced runtime compared to HotSpot (three orders of magnitude for a 10 ms update).

To determine the long term accuracy of our temperature estimator, we run the simulator for a longer time period (e.g., 2 min, 5 min, 10 min, and 20 min). In this experiment we do not use calibration from external sensor readings. The power task set, from the previous experiment, is regenerated to contain more power events over a longer time interval. The new task set contains 20 000 tasks whose arrival instants are randomly

TABLE IV
ALGORITHM PERFORMANCE FOR VARYING RUNTIME AND INPUT POWER GRANULARITY

Runtime length	2 min	5 min	10 min	20 min
Average temp. error	1.10%	1.95%	2.97%	2.35%
Worst temp. error	2.65%	4.21%	4.82%	4.63%
$\mathbf{P}_{AE}$	0.05 W	0.2 W	1 W	5 W
Average temp. error	0.004%	0.29%	0.46%	1.05%
Average overhead (μs)	683	379	198	126

distributed in the range [0 s:20 min], and whose execution times are extended to the range [10 ms:2 min]. A 10-ms-interval LUT is used for this scenario. Table IV shows the average error and worst case error for the different simulation periods. The long-term simulation of our open-loop thermal estimator does not result in significant errors, compared with HotSpot. The main reasons for this accuracy are explained in Section IV-F. The worst case error usually appears at the end of power tasks with a long execution time (e.g., $\geq$ 20 s). Thus, we can conclude that (10) is suitable for cases where the power task execution time is below 20 s, but underestimates the leakage power error closer to the steady state.

The next experiment examines how the power threshold value P_{ae} affects the error and overhead. As mentioned in Section II, the P_{ae} threshold value is used to define atomic events and thus affects the time complexity of our event driven thermal method. While a coarse grained power profile is suitable for DTAS, a smaller P_{ae} can be used to observe more detail in the thermal behavior (such as for thermal simulation at the architectural level). However, very small values of P_{ae} result in a fine-grained power profile, generating a large number of power events which need to be queued to the event list, resulting in a very significant computational overhead. For this experiment, a subset of the task set from the first example is regenerated using the different P_{ae} threshold values shown in Table IV. As expected, the temperature error increases with P_{ae} while the overhead decreases. We would suggest a P_{ae} value of 2 W to 5 W for DTAS and a P_{ae} value of 0.5 W to 1 W for thermal simulation.

C. Heuristic Predictive Task Allocation

In this section, we examine the performance of our fast event driven thermal estimator, in a DTAS scenario, when combined with our proposed heuristic scheduling policies. We compare our future coolest first (FC), future neighbor aware (FN), future task aware (FTA), and future temperature trend (FTT) policies, all based on the future thermal map, with previous dynamic scheduling approaches [e.g., coolest first (C) [31], neighbor aware (N) [31], and historical window for possibility of allocation (HWP) [29]], which are based on the current thermal map or historical thermal information in terms of the peak temperature, the average temperature and the spatial diversity. In FN we use the same parameters as in [31] ($a_1 = 0.45$, $a_2 = 0.25$, $a_3 = 0.15$, $a_4 = 5.1$, and $a_5 = 2.2$) and in FTA we use $a_1 = 0.7$ and $a_2 = 0.3$. Some existing DTM policies can also be modified to a DTAS scenario and thus can also be used for comparison purposes. We have modified the incremental task allocation (ITA) algorithm from [32] (and converted from a 3-D model to a 2-D model). Here, we assume that the speed for each core is a constant, and that tasks arriving simultaneously are sorted in descending power order. We also compare our DTAS algorithms with the PDTM algorithm proposed by [6]. Here, we use the temperature values from our simulator (rather than from DTSs) to generate the historical temperature profile used for the recursive application-based thermal prediction. We implement the predictive thermal model of [6], let $\Delta t_m = t_n - t_c = 10$ ms, and assume priority adjustment is disabled. For this paper, we generate 10 000 task sets with similar characteristics to those in Section VI-B.

In the first scenario, DTM is not used for comparing the peak/average temperature and spatial diversity and we assume that the cores can always work normally without any safety threshold. The purpose of this assumption is to exclude the effect of DTM (e.g., the use of migration or DVFS when the temperature threshold is exceeded) and just observe if the dynamic scheduling policies are efficient in terms of minimizing the peak/average temperature and the spatial/temporal diversity on chip. Here, spatial/temporal diversity is used to measure the degree of thermal balancing among the cores. A lower diversity indicates that the policy has a better ability to balance the thermal side effects on chip, thus reducing ageing due to rapid heating/cooling and dramatic temperature differences. These two metrics are defined in [31].

Table V (Scenario 1) shows that the results for the dynamic scheduling polices based on our predictive future thermal map have much better effect on the peak temperature (Peak T) and the average temperature (Ave T) minimization. FC and FN are superior to the simple C and N schedulers, showing that the look-ahead approach is an effective technique for avoiding unpredictable hot cores on chip. In terms of minimizing temperature, FN and HWP[7] have a similar effect, while the two more complex predictive polices FTA and FTT perform better than HWP and PDTM. ITA, with the smallest overhead, is only comparable to the simple C and N schedulers. It is notable that HWP and PDTM both achieve better average spatial diversity (Ave SD) and average temporal diversity (Ave TD) results than FTA and FTT. The reason is that we do not take advantage of the future thermal map to carry out task migration, as in [5] and [6], and the future time slot is too short compared to historical windows (longer historical windows are more useful to balance the thermal distribution than shorter prediction if we do not use migration or other DTM mechanisms). In other words, our policies only consider how to heuristically optimize the temperature by using the instantaneous temperature (e.g., the future thermal map or current thermal map), rather than using the temperature transient over a long period. Our proposed algorithms have a higher computational overhead (Overhead) than C, N, and ITA; however, this overhead is still acceptable particularly considering the improvement in thermal performance. Compared to HWP, our FTT technique has similar performance (better temperature minimization but slightly worse temperature diversity) with only a slight increase in complexity. However, it is uncertain if HWP and

[7]The temperature history used in our implementation of this algorithm is derived directly from HotSpot, rather than simulated thermal sensors.

TABLE V
PERFORMANCE COMPARISON FOR TASK ALLOCATION

Scenario	Metrics	C	N	HWP	PDTM	ITA	FC	FN	FTA	FTT
Scenario 1	Peak T (°C)	119.65	104.5	98.57	99.76	110.31	106.12	98.63	95.72	94.35
	Ave T (°C)	112.13	102.64	96.15	99.45	102.19	102.56	95.04	95.29	91.81
	Ave SD (°C)	12.37	4.83	3.24	3.35	3.61	8.76	4.32	3.85	4.12
	Ave TD (°C)	28.75	24.65	14.53	15.75	14.84	28.32	25.51	16.78	18.45
	Overhead (μs)	37	98	145	235	25	146	157	172	152
Scenario 2	DTM TT (times)	1873	1625	1478	1357	1646	1573	1512	1377	1407
	Complete T (s)	312.17	281.99	268.32	264.76	278.12	271.9	270.93	260.62	262.33
Scenario 3	Resp time (ms)	1267	965	872	905	507	987	893	642	728
	Rejection ratio	45.7	36.3	32.2	37.8	38.8	39.5	31.7	23.7	25.5

PDTM, with their reliance on sensor data, are able to scale to large many-core systems.

In the second scenario, we use the same task set as above, but set a thermal safety threshold (we use 80 °C) for each core to trigger DTM. Our simulator implements task migration in DTM by putting tasks on the hot core back into the ready queue and simply shutting the core down for a period of 50 ms (during this time it consumes no power). The migrated tasks are inserted at the head of the ready queue for immediate scheduling to some cooler core. Other DTM mechanisms (e.g., DVFS) are not used in this example. We simply count the number of DTM trigger times (DTM TT) and observe the overall completion time (Complete T) of all 10 000 power tasks. Intuitively, a reduced number of DTM trigger times indicates a smaller overhead induced by DTM, meaning that the whole task set would finish earlier.

Table V (Scenario 2) shows that policies based on predictive allocation alleviate the DTM loading and its overhead. Allocating a task to the right core (before it starts running) is better than frequent migration at run-time in terms of the completion time, as proactive allocation can avoid the repeated allocation/de-allocation on the same core and reduce the system overhead dramatically. FTA and PDTM performed best in terms of the number of DTM trigger events, with FTA having a reduced completion time due to the algorithm's reduced overhead. In fact, when considering all 10 000 power tasks, we could save as much as 8–16% off the execution time when using our predictive policies, comparing to the non-predictive policies. We conclude that scheduling, guided by the future thermal profile, will improve the overall system performance under strict DTM thermal constraints.

The last scenario shows the efficiency of our heuristic policies for use in a soft real-time system which allows tasks to be discarded if a deadline is missed. We generate a similar power task set to scenario 1, with an additional attribute: the deadline for each task. The deadline (relative to the start time of the task) is assumed as its execution time multiplied by a factor which is randomly distributed in the range [1.3:4]. DTM for migration is still enabled. We investigate the average response time (Resp Time) and the rejection ratio (Rejection Ratio) for the task set. The average response time is the latency between the arrival time and the start time of a task, while the rejection ratio is the number of tasks that are discarded. In evaluating how thermal-aware scheduling can affect the key measurements in real-time system performance, a lower response time and a lower rejection ratio are better.

Table V (Scenario 3) shows similar trends as in Scenarios 1 and 2, indicating that policies that reduce DTM events are also better for time constrained scenarios. Temperature minimization and thermal balancing means that more tasks can be scheduled to be executed simultaneously making it possible to schedule a task earlier. FTA outperforms the other methods in this scenario since it is the only heuristic policy to use task attributes in the scheduling decision. While ITA has a reduced response time due to its simplicity, it has an unacceptably high rejection ratio ($\sim$40% of tasks were rejected).

These results show that our FTA and FTT policies are generally superior to existing DTAS techniques in terms of core temperature, DTM trigger times, task response times and task rejection ratios, and are comparable to HWP and PDTM in terms of spatial and temporal diversity. We have not compared our algorithms to algorithms for course-grained DTM, such as [10] and [47]. However, our fine-grained DTAS policies could be combined with these DTM policies, to decide which core a task is migrated to after a DTM event.

VII. CONCLUSION

We proposed a fast LUT-based event-driven thermal estimation method based on atomic power events. First, prebuilt LUTs, representing the temperature increment due to one unit power input to a core, are used to determine the overall thermal map by accumulating all the thermal increments induced by events in the event list. We then developed an iteration free calibration which accounts for leakage power. Our proposed thermal estimator is validated by experimentation and is shown to be suitable for DTAS. We then developed a predictive future thermal map which we use to propose several online heuristic scheduling policies. These policies are shown to produce better overall thermal and real-time effects than previous DTAS solutions.

REFERENCES

[1] J. Cui and D. Maskell, "High level event driven thermal estimation for thermal aware task allocation and scheduling," in *Proc. ASP-DAC*, Jan. 2010, pp. 793–798.

[2] D. Atienza, P. G. D. Valle, G. Paci, F. Poletti, L. Benini, G. D. Micheli, J. M. Mendias, and R. Hermida, "HW-SW emulation framework for temperature-aware design in MPSoCs," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 12, no. 3, article 26, Aug. 2007.

[3] *International Technology Roadmap for Semiconductors 2009 Edition*. (2009) [Online]. Available: http://www.itrs.net/links/2009ITRS/Home2009.htm

[4] *Intel Single-Chip Cloud Computer*. (2011, Aug.) [Online]. Available: http://techresearch.intel.com/ProjectDetails.aspx?Id=1

[5] A. K. Coskun, T. S. Rosing, and K. C. Gross, "Proactive temperature balancing for low cost thermal management in MPSoCs," in *Proc. ICCAD*, Nov. 2008, pp. 250–257.

[6] I. Yeo, C. C. Liu, and E. J. Kim, "Predictive dynamic thermal management for multicore systems," in *Proc. DAC*, 2008, pp. 734–739.
[7] O. Khan and S. Kundu, "Hw/SW co-design architecture for thermal management of chip multiprocessor," in *Proc. DATE*, Apr. 2009, pp. 952–957.
[8] Y. Zhang and A. Srivastava, "Accurate temperature estimation using noisy thermal sensors," in *Proc. DAC*, Jul. 2009, pp. 472–477.
[9] M. Bao, A. Andrei, P. Eles, and Z. Peng, "On-line thermal aware dynamic voltage scaling for energy optimization with frequency/temperature dependency consideration," in *Proc. DAC*, Jul. 2009, pp. 490–495.
[10] I. Yeo and E. J. Kim, "Temperature-aware scheduler based on thermal behavior grouping in multicore systems," in *Proc. DATE*, Apr. 2009, pp. 946–951.
[11] T. Ebi, M. A. A. Faruque, and J. Henkel, "TAPE: Thermal-aware agent-based power economy for multi/many-core architectures," in *Proc. ICCAD*, Nov. 2009, pp. 302–309.
[12] C.-Y. Yang, J.-J. Chen, L. Thiele, and T.-W. Kuo, "Energy-efficient real-time task scheduling with temperature-dependent leakage," in *Proc. DATE*, Mar. 2010, pp. 9–14.
[13] V. Hanumaiah and S. Vrudhula, "Reliability-aware thermal management for hard real-time applications on multicore processors," in *Proc. DATE*, Mar. 2011, pp. 1–6.
[14] E. Rotem, J. Hermerding, C. Aviad, and C. Harel, "Temperature measurement in the Intel core duo," in *Proc. DAC*, 2006.
[15] S. Sharifi and T. S. Rosing, "Accurate direct and indirect on-chip temperature sensing for efficient dynamic thermal management," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 29, no. 10, pp. 1586–1599, Oct. 2010.
[16] F. Paterna, A. Acquaviva, A. Caprara, F. Papariello, G. Desoli, and L. Benini, "An efficient on-line task allocation algorithm for qos and energy efficiency in multicore multimedia platforms," in *Proc. DATE*, Mar. 2011, pp. 1–6.
[17] J. Cong and K. Gururaj, "Energy efficient multiprocessor task scheduling under input-dependent variation," in *Proc. DATE*, Apr. 2009, pp. 411–416.
[18] H. Y. Yang, B. Veeravalli, and Y. Ha, "Leakage-aware dynamic scheduling for real-time adaptive applications on multiprocessor systems," in *Proc. DAC*, Jun. 2010, pp. 493–498.
[19] L. Niu, "System-level energy-efficient scheduling for hard real-time embedded systems," in *Proc. DATE*, Mar. 2011, pp. 1–4.
[20] M. Chrobak, C. Durr, M. Hurand, and J. Robert, "Algorithms for temperature-aware task scheduling in microprocessor systems," in *Proc. AAIM*, 2008, pp. 120–130.
[21] S. Zhang and K. S. Chatha, "Approximation algorithm for the temperature-aware scheduling problem," in *Proc. ICCAD*, Nov. 2007, pp. 281–288.
[22] S. Irani and K. R. Pruhs, "Algorithmic problems in power management," *ACM SIGACT*, vol. 36, no. 2, pp. 63–76, 2005.
[23] S. Murali, A. Mutapcic, D. Atienza, R. Gupta, and G. D. Micheli, "Temperature-aware processor frequency assignment for MPSoCs using convex optimization," in *Proc. CODES+ISSS*, 2007, pp. 111–116.
[24] T. Chantem, R. P. Dick, and X. S. Hu, "Temperature-aware scheduling and assignment for hard real-time applications on MPSoCs," in *Proc. DATE*, Mar. 2008, pp. 288–293.
[25] Y. Xie and W. L. Hung, "Temperature-aware task allocation and scheduling for embedded multiprocessor systems-on-chip (MPSoC) design," *J. VLSI Signal Process.*, vol. 45, no. 3, pp. 177–189, Dec. 2006.
[26] C.-Y. Yang, J.-J. Chen, and T.-W. Kuo, "Energy-efficiency for multi-frame real-time tasks on a dynamic voltage scaling processor," in *Proc. CODES+ISSS*, 2009, p. 10.
[27] T. Chantem, R. P. Dick, and X. S. Hu, "Temperature-aware scheduling and assignment for hard real-time applications on MPSoCs," in *Proc. DATE*, Mar. 2008, pp. 288–293.
[28] S. Zhang and K. S. Chatha, "Thermal aware task sequencing on embedded processors," in *Proc. DAC*, Jun. 2010, pp. 585–590.
[29] A. K. Coskun, T. S. Rosing, and K. Whisnant, "Temperature aware task scheduling in MPSoCs," in *Proc. DATE*, Apr. 2007, pp. 1–6.
[30] A. K. Coskun, T. S. Rosing, K. A. Whisnant, and K. C. Gross, "Static and dynamic temperature-aware scheduling for multiprocessor SoCs," *IEEE Trans. Very Large Scale Integr. Syst.*, vol. 16, no. 9, pp. 1127–1140, Sep. 2008.
[31] K. Stavrou and P. Trancoso, "Thermal-aware scheduling for future chip multiprocessors," *EURASIP J. Embedded Syst.*, vol. 2007, no. 48926, p. 15, 2007.
[32] C.-L. Lung, Y.-L. Ho, D.-M. Kwai, and S.-C. Chang, "Thermal-aware on-line task allocation for 3-D multicore processor throughput optimization," in *Proc. DATE*, Mar. 2011, pp. 1–6.
[33] V. Hanumaiah, R. Rao, S. Vrudhula, and K. S. Chatha, "Throughput optimal task allocation under thermal constraints for multi-core processors," in *Proc. DAC*, Jul. 2009, pp. 776–781.
[34] K. Skadron, M. R. Stan, W. Huang, S. Velusamy, K. Sankaranarayanan, and D. Tarjan, "Temperature-aware microarchitecture," in *Proc. ISCA*, Jun. 2003, pp. 2–13.
[35] R. Joseph and M. Martonosi, "Run-time power estimation in high-performance microprocessors," in *Proc. ISLPED*, 2001, pp. 135–140.
[36] W. Wu, L. Jin, J. Yang, P. Liu, and S. X. Tan, "Efficient power modeling and software thermal sensing for runtime temperature monitoring," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 12, no. 3, article 25, Aug. 2007.
[37] X. Chen, R. P. Dick, and L. Shang, "Properties of and improvements to time-domain dynamic thermal analysis algorithms," in *Proc. DATE*, Mar. 2010, pp. 1165–1170.
[38] P. Liu, H. Li, L. Jin, W. Wu, S. X.-D. Tan, and J. Yang, "Fast thermal simulation for runtime temperature tracking and management," *IEEE Trans. Comput.Aided Des. Integr. Circuits Syst.*, vol. 25, no. 12, pp. 2882–2893, Dec. 2006.
[39] D. Burger, T. M. Austin, and S. Bennett, "Evaluating future microprocessors: The simple scalar tool set," Comput. Sci. Dept., Univ. Wisconsin, Madison, WI, Tech. Rep. 1308, 1996.
[40] D. Brooks, V. Tiwari, and M. Martonosi, "Wattch: A framework for architecture-level power analysis and optimizations," in *Proc. ISCA*, Jun. 2000, pp. 83–94.
[41] K.-J. Lee and K. Skadron, "Using performance counters for runtime temperature sensing," in *Proc. IPDPS*, Apr. 2005, p. 8.
[42] K. Skadron, M. R. Stan, W. Huang, S. Velusamy, K. Sankaranarayanan, and D. Tarjan, "Temperature-aware microarchitecture: Extended discussion and results," Dept. Comput. Sci., Univ. Virginia, Charlottesville, VA, Tech. Rep. CS 2003-08, 2003.
[43] J. H. Choi, A. Bansal, M. Meterelliyoz, J. Murthy, and K. Roy, "Leakage power dependent temperature estimation to predict thermal runaway in FinFET circuits," in *Proc. ICCAD*, Nov. 2006, pp. 583–586.
[44] Y. Liu, R. P. Dickt, L. Shang, and H. Yang, "Accurate temperature-dependent integrated circuit leakage power estimation is easy," in *Proc. DATE*, Apr. 2007, pp. 1–6.
[45] G. Paci, P. Marchal, F. Poletti, and L. Benini, "Exploring temperature-aware design in low-power MPSOCs," in *Proc. DATE*, Mar. 2006, pp. 1–6.
[46] *Hotspot v5.0*. (2011, Aug.) [Online]. Available: http://lava.cs.virginia.edu/HotSpot/index.htm
[47] F. Mulas, M. Pittau, L. Benini, A. Acquaviva, and D. Atienza, "Thermal balancing policy for streaming computing on multiprocessor architectures," in *Proc. DATE*, Mar. 2008, pp. 734–739.
[48] J. Donald and M. Martonosi, "Techniques for multicore thermal management: Classification and new exploration," in *Proc. ISCA*, 2006, pp. 78–88.

Jin Cui (SM'07) received the Bachelors degree in mechanical engineering and automation and the Masters degree in computer science and engineering from Northeastern University, Shengyang, China, in 2004 and 2007, respectively. He is currently pursuing the Ph.D. degree with the School of Computer Engineering, Nanyang Technological University, Singapore.

His current research interests include reconfigurable computing, power-thermal-aware optimization for multiprocessor systems, and real-time embedded systems.

Douglas L. Maskell (M'85) received the B.E. (Hons.), M.Eng.Sc., and Ph.D. degrees in electronic and computer engineering from James Cook University, Townsville, Australia, in 1980, 1984, and 1996, respectively.

He is currently an Associate Professor with the School of Computer Engineering, Nanyang Technological University, Singapore. His current research interests include embedded systems, reconfigurable computing, and algorithm acceleration for hybrid high performance computing systems.

Dr. Maskell is an Active Member of the Reconfigurable Computing Group, Center for High Performance Embedded Systems.