Thermal Analysis of Multiprocessor SoC Applications by Simulation and Verification

DIPANKAR DAS, P. P. CHAKRABARTI, and RAJEEV KUMAR
Indian Institute of Technology Kharagpur

Overheating of computer chips leads to degradation of performance and reliability. Therefore, preventing chips from overheating in spite of increased performance requirements has emerged as a major challenge. Since the cost of cooling has been rising steadily, various architecture and application design techniques are used to prevent chip overheating. Temperature-aware task scheduling has emerged as an important application design methodology for addressing this problem in multiprocessor SoC systems.

In this work we present the formulation and implementation of a method for analyzing the thermal (chip heating) behavior of a MPSoC task schedule, during the early stages of the design. We highlight the challenges in developing such a framework and propose solutions for tackling them. Due to nondeterminism in task execution times and decision branches, multiprocessor applications cannot be evaluated accurately by the current state-of-the-art *thermal simulation* and *steady-state* analysis methods. Hence an analysis covering nondeterministic execution behaviors is required for thermal analysis of MPSoC task schedules. To address this issue we propose a model checking-based approach for solving the thermal analysis problem and formulate it as a hybrid automata reachability verification problem. We present an algorithm for constructing this hybrid automata given the task schedule, a set of power profiles of tasks, and the Compact Thermal Model (CTM) of the chip. Information about task power consumption is inferred from Markov chains which are learned from power profiles of tasks, obtained from simulation or emulation runs. A numerical analysis-based algorithm which uses CounterExample-Guided Abstraction Refinement (CEGAR) is developed for reachability analysis of this hybrid automata. We propose a directed simulation methodology which uses results of a time-bounded analysis of the hybrid automata modeling thermal behavior of the application, to simulate the expected worst-case execution runs of the same. The algorithms presented in this work have been implemented in a prototype tool called *HeatCheck*. We present experimental results and analysis of thermal behavior of a set of task schedules executing on a MPSoC system.

Categories and Subject Descriptors: D.2.4 [**Software Engineering**]: Software/Program Verification—*Formal methods*; C.4 [**Computer System Organization**]: Performance of Systems—*Modeling techniques*

General Terms: Verification, Algorithms, Performance, Measurement

This work was partially funded by a Microsoft Research India graduate fellowship to D. Das.
Authors' address: D. Das, P. P. Chakrabarti, R. Kumar, Department of Computer Science and Engineering, Indian Institute of Technology Kharagpur, India; email: {ddas, ppchak, rkumar}@cse.iitkgp.ernet.in.

DOI 10.1145/1698759.1698765 http://doi.acm.org/10.1145/1698759.1698765

Additional Key Words and Phrases: Thermal analysis, chip temperature, hybrid automata, multiprocessor system-on-chip, Markov chain

ACM Reference Format:

Das, D., Chakrabarti, P. P., and Kumar, R. 2010. Thermal analysis of multiprocessor SoC applications by simulation and verification. ACM Trans. Des. Autom. Electron. Syst. 15, 2, Article 15 (February 2010), 52 pages.
DOI = 10.1145/1698759.1698765 http://doi.acm.org/10.1145/1698759.1698765

1. INTRODUCTION

In recent times, there has been a rapid proliferation of high-performance Multiprocessor System-on-Chip (MPSoC) embedded systems, like gaming consoles, which are based on platforms like the Cell BE [Moore 2006], Xenon[1], and NVidia CUDA.[2] The emerging structure of applications executing on such systems consists of tasks (code blocks) statically mapped and scheduled to cores, and communicating and synchronizing amongst themselves through shared memory or on-chip communication. During the early design stages, operating on task-level models of applications, the design-flow for such MPSoC applications proceeds in two phases, namely a *synthesis* phase followed by an *analysis* phase. The most important synthesis step in the early design stage of MPSoC applications is static *task scheduling* [Pop et al. 2006], wherein the code-blocks (tasks) of the application are mapped to different processors, and a partial precedence order between tasks is enforced. The objective of task scheduling is to obtain a static schedule which is functionally correct [Das et al. 2007] and conforms to certain design constraints and objectives. These design constraints and objectives are typically time of execution [Dasdan et al. 1998; Das et al. 1998], energy [Hua et al. 2006], battery-aware power consumption [Luo and Jha 2001], and the recently emerging temperature-aware design objectives [Hung et al. 2005]. Various manual and automatic methods are employed for the task-scheduling step. A multiprocessor application is obtained directly from the task schedule by replacing tasks with code-blocks and adding appropriate communication operations. The analysis phase starts once the scheduled and mapped task-level model of the multiprocessor application has been synthesized. Here the synthesized multiprocessor application is checked for functional correctness, and conformance to various design constraints and objectives. If the design objectives and constraints are not satisfied by the synthesized application, then the synthesis step is reiterated and a new design is obtained. Sometimes during design-space exploration, analysis is performed hand-in-hand with the synthesis step.

Although execution time and power awareness have traditionally been the primary design constraints; in recent times temperature-aware system design has attained significance. Overheated chips have increased degradation in lifetime and reliability [Shin et al. 2007], gradually leading to loss of functionality. Moreover, in modern chips leakage power is significant, and is exponentially dependent on chip temperature [Su et al. 2003]. This, in conjunction with high

[1]http://en.wikipedia.org/wiki/Xenon_(processor)
[2]http://en.wikipedia.org/wiki/CUDA

costs of cooling, has resulted in temperature-aware system design gaining importance in recent times. Although several hardware techniques like thermal-aware floorplanning [Huang et al. 2006], and circuit-level power reduction techniques [Venkatachalam and Franz 2005], as well as Dynamic Thermal Management (DTM) strategies [Yeo et al. 2008; Jayaseelan and Mitra 2008; Rao and Vrudhula 2008] have been widely employed, the domain of MPSoC systems presents some unique challenges and opportunities for containing chip temperature.

The application design process for MPSoC systems allows us to take advantage of task scheduling and mapping to develop applications which avoid executing tasks on overheating cores and allow them to cool [Zhang and Chatha 2007; Choi et al. 2007]. Various static voltage-frequency scaling strategies can also be explored. This ability to manipulate thermal behavior by modifying the task schedule and mapping has lead to the emergence of temperature awareness as a design objective for MPSoC task scheduling [Rao et al. 2007]. At the same time, MPSoC systems usually have several other requirements, like real-time constraints [Dasdan et al. 1998] and Quality-of-Service (QoS) requirements [Bowman et al. 1998], on execution of applications. Hence, there is a challenge to avoid employment of Dynamic Voltage-Frequency Scaling (DVFS) [Isci et al. 2006]-based temperature management, since these methods may slow cores in an unpredictable manner, which in turn could lead to breach of other timing/QoS constraints. Additionally the assignment of tasks to processors is static, with the aim of guaranteeing constraints on execution time [Dasdan et al. 1998], or volume of communicated data [Satish 2009], thereby mitigating migration-based [Jayaseelan and Mitra 2008] thermal management strategies. Such being the case, it is of interest to devise *postsynthesis thermal analysis* methods which evaluate schedules during the early design stages, and provide an estimate of peak temperature, as well as execution paths leading to the same.

In cases where execution time is an optimization parameter and not a constraint on the design, it is desirable to select a schedule which causes the least chip temperature rise, and thereby yields performance benefits by not getting slowed down by DVFS methods, as well as saving leakage power [Su et al. 2003]. In these cases, a thermal analysis method can be used for comparing schedules, during the early design stages, and identifying which schedule causes lower temperature rise (does not breach a temperature limit within a time bound) than others (breaches the temperature limit within the time bound). Additionally, a thermal analysis tool can identify worst-case (in terms of temperature rise) execution behaviors of concurrent multicore applications, and hence provide important feedback to the designer. Moreover, thermal analysis is often a precursor step in the development of Dynamic Thermal Management (DTM) methods [Skadron et al. 2004], which require a detailed study of chip-thermal behavior of applications. Indeed thermal analysis is complementary to DTM methods like Dynamic Voltage-Frequency Scaling (DVFS). In this work we focus on the development of a thermal analysis method for checking synthesized MPSoC task schedules, in the early deign stage, for conformance to bounds on chip temperature. Therefore the proposed method contributes to the postsynthesis analysis phase of the MPSoC application design-flow.

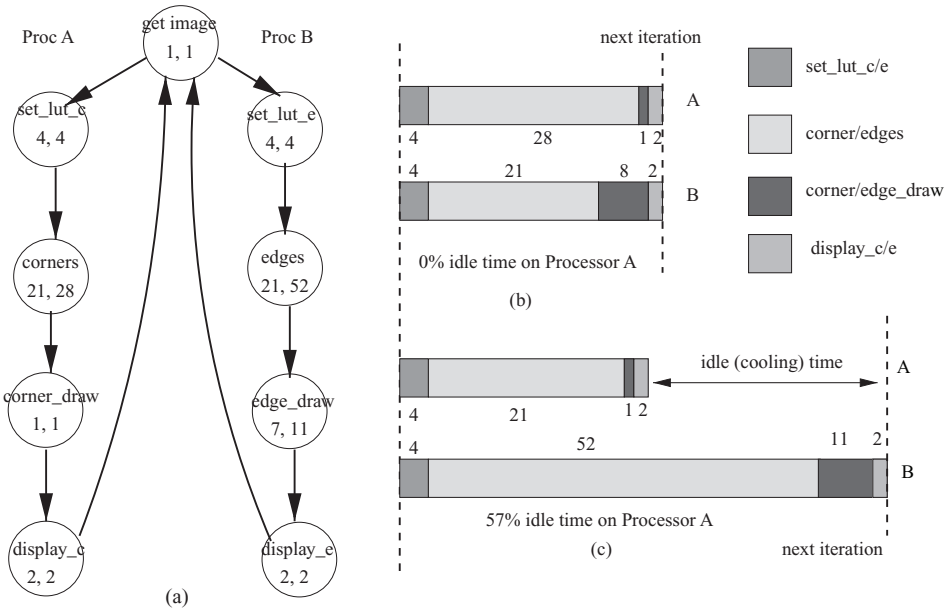

Fig. 1. A nondeterministic task graph: (a) of the *susan* testcase from the MiBench suite and time taken by tasks in x20,000 cycles. Two, (b) and (c), execution runs for one iteration. The execution run in (b) dissipates power at a higher rate than that in (c) since it has less *idle* time.

The state-of-the-art thermal analysis methods are thermal simulation [Skadron et al. 2004] and steady-state analysis [Zhan and Sapatnekar 2005], on block-level [Skadron et al. 2004] and grid-based [Li et al. 2006] models. Thermal simulation evaluates the transient thermal behavior of an application when the power dissipation in the chip varies with time, depending on the code segment being executed. On the other hand, steady-state analysis methods evaluate steady-state temperature values for a given power dissipation and ignore dynamic variations. Typically the average or worst-case power dissipation is chosen for steady-state analysis. Although these methods give a reasonable estimate of temperature variations for single-core processors, such analysis is often inaccurate for multicore systems, as explained next.

Often in case of multicore systems a thread may be *idle*, waiting for the completion of some task executing on another core. In this case the core executing the idle thread typically throttles the clock frequency or shuts down [Jejurikar and Gupta 2004] in order to save power. This results in a significant difference in power dissipation between times when the thread is active, and when it is idle. When code-blocks (tasks) execute nondeterministically, due to different inputs or cache behavior, the ratio of active and idle durations varies significantly, leading to nondeterministic variation in thermal behavior of MPSoC systems. In this case simulation-based methods suffer from issues of incomplete coverage, since they can evaluate only one nondeterministic run at a time.

As an example consider the execution of the task graph for the *susan* testcase (Figure 1(a)) from the MiBench [Guthaus et al. 2001] suite. This task graph performs two operations, namely corner and edge detection, on one input image

which is fetched by the *get_image* task. The corner and edge detection are performed on two cores (A and B) simultaneously and continue ad infinitum, synchronized at the start of the *get_image* task. The corner detection is performed by the *set_lut_c*, *corners*, *corner_draw*, and *display_c* tasks while edge detection is performed by *set_lut_e*, *edges*, *edge_draw*, and *display_e*. The range of time taken for the execution of different tasks executing on an Alpha EV6 core is presented in Figure 1(a).

Two, among several other, nondeterministic executions of the task graph are shown in Figures 1(b) and 1(c), respectively. In the execution run in Figure 1(b), each iteration (processing one image) completes in 0.7 million cycles ($35 \times 20{,}000$ cycles) and none of the threads is ever idle. On the other hand, one iteration of the execution run in Figure 1(c) takes 1.38 million cycles ($69 \times 20{,}000$ cycles) and core A is idle for 57% of the time, allowing it and the silicon in the vicinity to cool down. In a thermal simulation, either one of the two (and several other) nondeterministic execution runs could have been analyzed, resulting in inaccurate analysis.

On the other hand, in order to perform steady-state analysis the worst-case or average power consumption values must be obtained. In the case of multicore systems, this depends not only on the power dissipation profile of individual code blocks, but also on the knowledge of which code blocks may execute concurrently (for worst-case analysis) and the total duration of execution (for average-case analysis). This requires the construction of a reachability graph of the application model. Moreover, the worst-case power dissipation configuration may occur for a small time duration and may dissipate significantly higher power than other configurations, leading to an overapproximate analysis. Hence worst-case steady-state analysis is often not performed for thermal analysis of multiprocessor systems. On the other hand, the average power dissipation may mask runs which dissipate higher power and hence underestimate the peak chip temperature. For the example presented in Figure 1, the average-case power dissipation will approximately correspond to a configuration with 27% idle time in processor A.

Clearly, there is a need for a thermal analysis methodology which covers all nondeterministic execution paths arising due to concurrent execution of tasks on multiprocessor platforms. However, the large number of execution paths of the task-graph model (often of the order of 2^{300}) poses a challenge for conventional simulation-based analysis. In order to tackle this problem, we propose a model checking-based approach for this analysis, which has the advantage of employing *backtracking* and *state-matching*. Backtracking allows the analysis to build on stored results of partial simulation runs, thereby avoiding re-execution from the start for each path analyzed. State-matching avoids the simulation of many redundant execution runs, when the current configuration (consists of executing tasks and an appropriate abstraction of chip temperatures) matches with a previously analyzed configuration. However, we would like to reiterate here that even though a model checking-based problem formulation and solution technique has been proposed in this work, the thermal analysis problem addressed is not a formal verification problem. Instead, the aim of this analysis is to provide feedback to the designer regarding worst-case

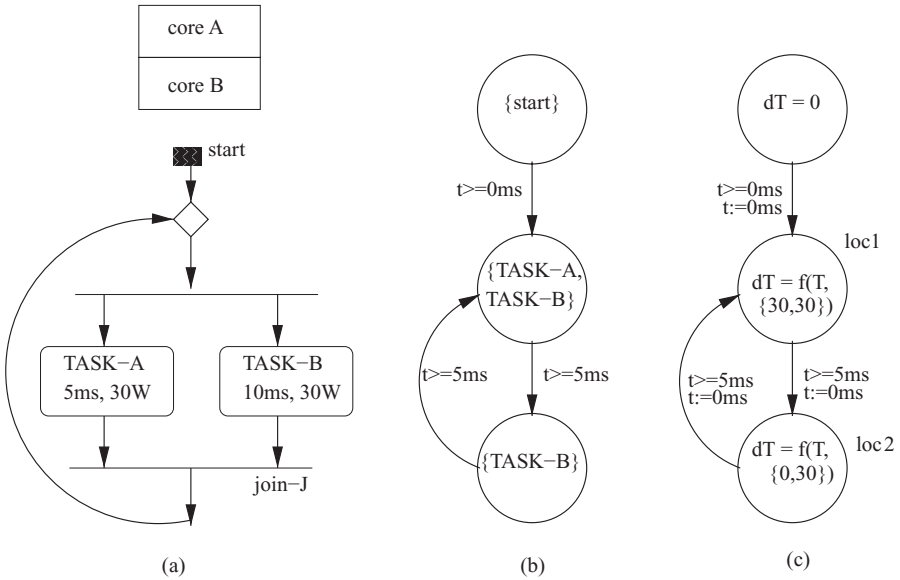

Fig. 2. Modeling thermal behavior of an application as hybrid automata. (a) Task-level model of an application and a simple chip floorplan; (b) reachability graph (automata) of application; (c) hybrid automata with continuous dynamics parametrized by power dissipation in cores obtained from task mappings. The invariant $t \leq 5\,ms$ is associated with both $loc1$ and $loc2$.

execution paths (in terms of chip heating), and expected chip temperatures, in the design-flow of MPSoC applications. We also use results of a time-bounded exhaustive analysis to identify potential worst-case execution (in terms of chip temperature rise) paths, and then perform directed thermal simulations.

Central to an exhaustive and systematic analysis of thermal behavior of a MPSoC application is a model of thermal behavior of such an application. In this work the thermal behavior of an application is modeled as a hybrid automata, and the thermal analysis problem is formulated as a hybrid automata [Alur et al. 2006] reachability verification problem. We note that, at an architectural-level abstraction, chip temperature is governed by differential equations modeling heat-flow and heat storage in the chip [Skadron et al. 2004]. These equations are parametrized by the power dissipation in different die-segments of the chip and may be considered to be of the form $dT = f\,(T,\, P)$. Here, T is the vector of temperatures of different die-segments of the chip, while P is the power dissipation in these die-segments. The power dissipation in various die-segments is dependent on code-segments (tasks) executing concurrently on different cores. The set of concurrently executing tasks changes in discrete steps with the execution of the multiprocessor application, introducing discrete variations in the manner in which chip temperature changes. Hence, chip-temperature varies in discrete steps, due to changes in concurrently executing tasks, as well as continuous dynamics guided by analytical equations.

As an example consider the task-level application model in Figure 2(a), which consists of two tasks *TASK-A* and TASK-B executing concurrently on cores *A and B* ad infinitum, with each core corresponding to a die segment of the chip.

In this model *TASK-A* executes for 5 ms while TASK-B executes for 10 ms, while both tasks dissipate 30 W each in the cores they are mapped to. These tasks synchronize during each loop (at node *join-J*) such that the execution in each iteration proceeds with *TASK-A* and *TASK-B* executing for 5 ms and then only *TASK-B* executing for another 5 ms (Figure 2(b)). Hence, there are two configurations of executing tasks. When both *TASK-A* and *TASK-B* execute concurrently for 5 ms, 30 W is dissipated in each die-segment (corresponding to cores A and B). However, when only *TASK-B* executes, then 30 W is dissipated in core *B* while core *A* may be switched off and dissipate no power (0 W). This difference in power dissipation in die-segments alters the manner in which chip temperature changes (since $dT = f(T, P)$). Hence, when there is a discrete change in configuration of executing tasks from {*TASK-A, TASK-B*} to {*TASK-B*}, a discrete step occurs in the chip temperature variation. These discrete variations, at an appropriate level of granularity, can be represented by a reachability graph (automata) of the task-level model as shown in Figure 2(b). A direct mapping from execution states (concurrently executing tasks) to power dissipation, in conjunction with analytical equations for a given power dissipation, lets reachability graph states be mapped to hybrid automata [Asarin et al. 2006] locations (Figure 2(c)). In this case checking bounds on chip temperature translates to a reachability verification problem on this hybrid automata.

There are several challenges in formulating and solving the thermal analysis problem as a hybrid automata reachability problem. We note that the hybrid automata analysis-based method requires as input the power dissipation profile (power dissipation over time) of each task, in each die segment, for all execution runs of the task. It is difficult to obtain this set of power profiles for each execution run of each task in a practical manner. This is because current architectural power estimation tools [Brooks et al. 2000; Loghi et al. 2007] and hardware emulation setups [Ignowski et al. 2008] can only profile a subset of tasks and task execution runs, during each simulation/emulation run of the application. In this setting, obtaining power dissipation profiles for each run of each task may require an unfeasibly large number of simulation/emulation runs. Moreover, obtaining testcases which trigger the appropriate execution runs is also a nontrivial problem. We propose to overcome this issue by employing a learning-based technique with broadly similar objectives as methods which have been used for predicting application behavior in the context of dynamic power management [Jung and Pedram 2008]. In this work we learn a discrete-time Markov chain from available simulation profiles, and subsequently querying the learned Markov chain for estimating the power profiles for task execution runs which are not available from any simulation/emulation run.

Another challenge faced is that the hybrid automata constructed for thermal analysis often contains 60–120 continuous variables corresponding to differential equations governing chip temperature. Hybrid systems of such size cannot be analyzed by hybrid automata analysis tools available in the public domain [Frehse 2005; Henzinger et al. 1997].To address this problem we develop a novel numerical analysis-based CEGAR (CounterExample Guided Abstraction

Refinement) [Clarke et al. 2003] method in conjunction with a gamut of heuristics, to solve the thermal analysis problem effectively. We also propose a setup wherein results of exhaustive thermal analysis for relatively small time-bounds are utilized for performing directed simulations of thermal behavior. This approach is applied when the proposed CEGAR-based hybrid automata reachability analysis approach does not scale.

A prototype tool called *HeatCheck* is developed for solving the thermal analysis problem and experimental results for analysis of various static schedules on a quad-code multiprocessor SoC are presented. We would like to emphasize that this work does not present an off-the-shelf thermal analysis tool, but instead presents a methodology for formalizing and solving the thermal analysis problem using available power and thermal models.[3] Hence the accuracy of results depend on the power and thermal models used. At the same time, more accurate power [Eisley et al. 2006] and thermal models (for example HotSpot 4.0), as and when they are developed by researchers, can be fit into the proposed framework for obtaining better results.

We summarize the contributions of this work as follows.

(1) a formal hybrid automata model of chip-thermal behavior of statically scheduled MPSoC applications, described at a task-level abstraction (multiple independent applications are modeled by sets of tasks, with no interaction between tasks from different sets).
(2) formulation of the thermal analysis problem for statically scheduled MPSoC applications as a hybrid automata reachability analysis problem.
(3) a model-checking-based method for solving the thermal analysis problem. We propose a counterexample-guided abstraction refinement-based method for addressing a special class of hybrid automata reachability problem, pertinent to performing thermal analysis.
(4) a directed simulation method which identifies execution paths which are expected to lead to the highest temperature rise, and then simulates these paths for long time intervals to reason about the worst-case behavior.
(5) a Discrete-Time Markov Chain (DTMC) learning-based method for estimating the power dissipation profile of tasks. This method has broadly similar objectives as learning-based prediction schemes like Jung and Pedram [2008].

The rest of the article is organized as follows. In Section 2 we present the model of task schedules and the chip-thermal model used in this work. Thereafter we formulate the thermal analysis problem as a hybrid automata reachability verification problem. We present the algorithms for learning power dissipation profiles, and solving the thermal analysis problem in Section 3. We present experimental results and analysis of HeatCheck on a set of MPSoC application schedules in Section 4. Finally we conclude this work in Section 5.

[3]Simplescalar PAnalyzer and HotSpot 3.1 [Huang et al. 2006] were used in this work.

2. MODELING AND PROBLEM FORMULATION

In this section we present a formulation of the thermal analysis problem for task-level multiprocessor SoC applications. First, we present the model of multiprocessor applications used in this work and then briefly describe the thermal model of chips. We argue that knowledge about power consumption and duration of execution of actions (tasks) is sufficient to analyze the overall thermal behavior of a multiprocessor SoC system. Subsequently we formulate the thermal analysis problem as a hybrid automata reachability verification problem. Finally, we identify and elucidate the challenges in formulating and solving the thermal analysis problem.

2.1 The Model for Concurrent Applications

The thermal behavior of a system is dependent on both the chip-thermal model and the task schedule. In this subsection we present the model for representing a multiprocessor SoC task schedule.

Concurrent multiprocessor SoC applications consist of threads of tasks executing on different processors. Each task represents either a code-block or a data communication operation, or an operation involving both code execution and data transfer. The threads synchronize among themselves to model data-flow and messaging between processors. Multiple applications are modeled by treating them as sets of independent threads, with no communication/synchronization between the sets of threads. Moreover typical applications execute continuously with different inputs, leading to the ad infinitum execution of threads unless the system is halted. In this section we present a task-level model for multiprocessor SoC applications, which is similar to proposed task-level modeling languages for multimedia [Drake 2006], embedded [Pop et al. 2006], and server [Isard et al. 2007] applications.

The model for concurrent applications used in this work is a directed graph G(V, E) which we call an *application graph*. Here each node $v \in V$ corresponds to a task (an *Action* node) in the model or to a decision/merge node as part of a decision construct (Figure 3(b)). Each edge $\langle u, v \rangle \in E$ corresponds to a precedence order between nodes u and v. A decision/merge node has exactly one/two incoming edges and two/one outgoing edges from and to nodes mapped to the same processor. The time of start of execution of an action node i is denoted by t_i, while the time taken for completing execution is denoted by δ_i. Time is considered discretized into nonoverlapping intervals of equal size (Δ) called *time-slots* and all events occur in time which are discrete multiples of Δ. The time duration δ_i is therefore described in terms of the number of time-slots (of size Δ) elapsed. This is bound by an integer interval $\langle d_i, D_i \rangle$ which is marked as a label on the action node. The delay of an action may be any discrete integer value between d_i and D_i ($d_i \leq \delta_i \leq D_i$). This time range for execution delay of an action node may be obtained either from static Worst-Case Execution Time (WCET) analysis [Wilhelm et al. 2008] or from a set of performance simulation runs [Sun et al. 2005]. The decision and merge nodes are control nodes which are assumed to take infinitesimally small time to execute.

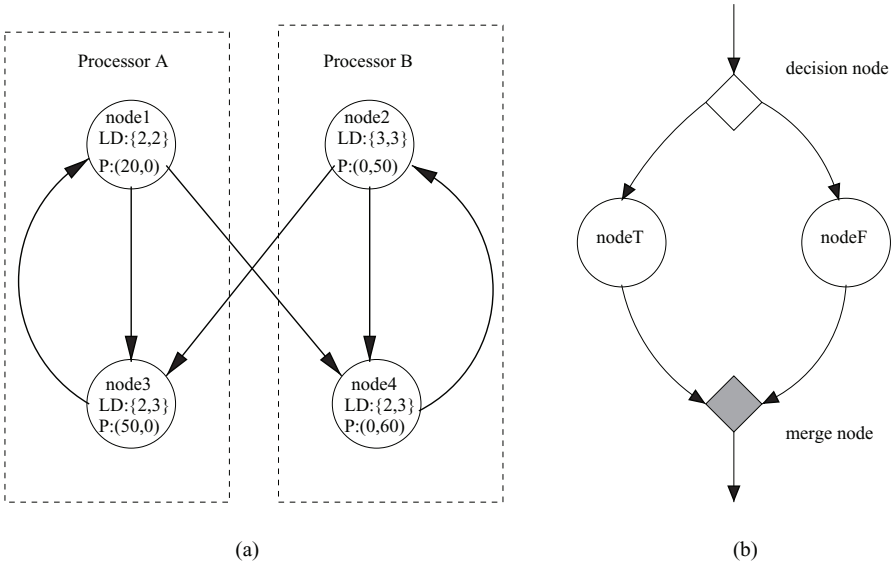

Fig. 3. An application graph model of an interacting multiprocessor application (a) and a decision construct (b) consisting of one node in each branch.

Action and decision nodes in the model are "and" nodes, thereby a node i starts executing only when all predecessors j of i have completed execution ($t_i \leftarrow \max(\forall_{j \in predecessor(i)} t_j + \delta_j)$). Merge nodes are "xor" nodes which execute as soon as one predecessor completes execution. Each node is mapped to a processor and nodes mapped to a single processor are completely ordered. This ordering is explicitly represented by precedence edges between successive nodes in a processor. Each processor has exactly one node denoted as the *start* node for the processor and one node denoted as the *finish* node. The execution of actions on each processor continues ad infinitum. This is modeled by an edge between the *finish* node and the *start* node of each processor. The *start* nodes of all the processors start executing at time $t = 0$. For example, consider the model in Figure 3(a). Here the set of actions is {node1, node2, node3, node4}, where node1 and node3 are mapped to processor A and node2 and node4 are mapped to processor B. Each node is labeled with the time range of execution. For example, consider *node3*, which has label L_D:{2, 3}. This indicates that node3 takes a minimum of 2 time-slots and a maximum of 3 time-slots to execute. Also in the example presented, node1 and node2 are the *start* nodes while node3 and node4 are the *finish* nodes.

Action nodes correspond to computation or communication components of the application and hence dissipate power in die segments, containing various architectural units, of the chip. This power dissipation varies over the duration of execution of the action node (task). Additionally, the power dissipation profile over the duration of execution varies with different times of completion of execution. For example, Figure 4 shows the power dissipation profile of two runs of an action node (*edges* from example in Figure 1(a)) in the die segment

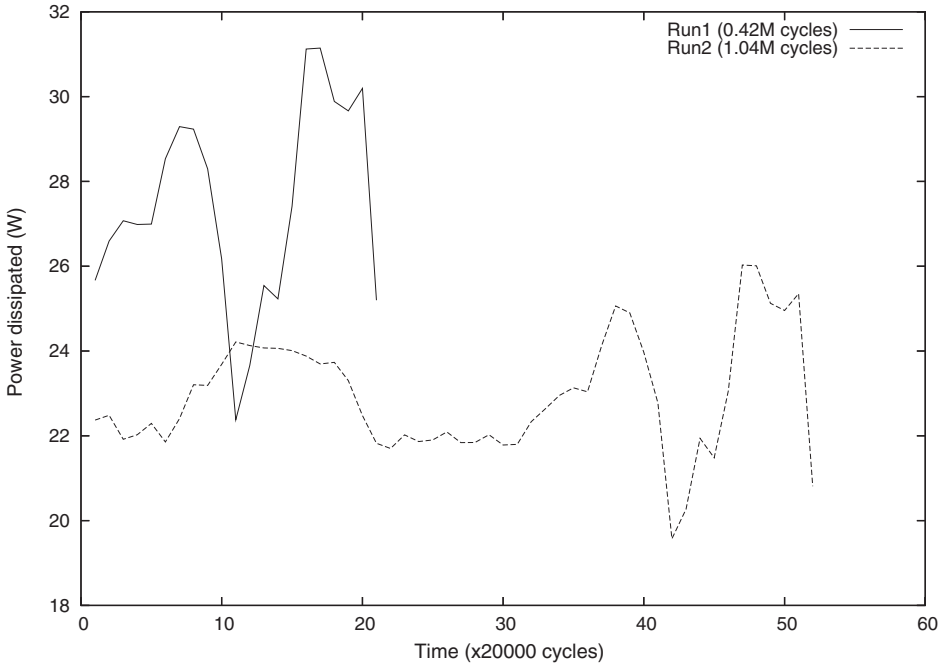

Fig. 4. Power dissipation profiles of two runs of the *edges* task from the task-graph of *susan* in Figure 1 in the "Int" die segment of the chip floorplan in Figure 6. This die segment comprises of the integer arithmetic unit and integer registers.

"Int" (Figure 6), consisting of the integer arithmetic unit and integer registers. The average power dissipation in time-slots of 20,000 cycles is illustrated for the two runs lasting 0.42 and 1.04 million cycles respectively. Therefore, each action node is associated with a set of trajectories, where each trajectory denotes the power dissipation over time for an execution run of the task, in a die segment of the chip. A mapping, L_{power} is used to represent this set of trajectories. It may be noted here that the power dissipation in various die segments of a multicore chip is guided by the task mapping (L_P).

The power dissipation of action nodes in Figure 3 in the two die segments ($Block1$ and $Block2$) in the silicon layer of the floorplan in Figure 5 are shown in Figure 3(a). For example, power dissipation of $node3$ is $P : \{50, 0\}$, indicating that $50W$ is dissipated in $Block1$ and $0W$ in $Block2$. For the sake of simplicity and understanding, they are constant over time and independent of execution completion times of action nodes.

Formally, the *application graph* is a tuple G: $\langle V, E, L_P, L_D, L_T, L_{power}, S, F \rangle$. Here C is a set of processors to which the application set is mapped and D is the set of die segments of the multicore chip.

(1) V is the set of application graph nodes.
(2) E is the set of edges indicating precedence such that $\langle u, v \rangle$ indicates that $v \prec u$.

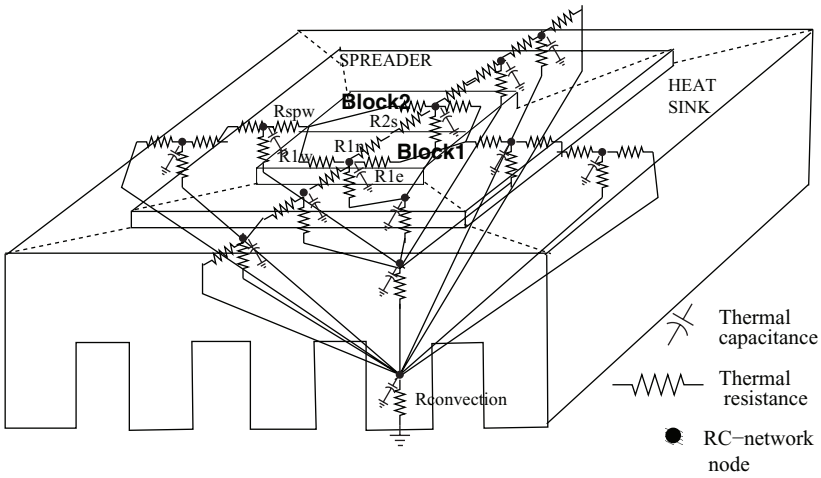

Fig. 5. The RC model for the complete chip floorplan consisting of three layers, namely the silicon die, the heat spreader, and the heat sink. There are 13 RC-network nodes.

(3) L_P:$V \rightarrow C$, is a mapping of nodes to processors.
(4) L_D:$V \rightarrow Z^+ \times Z^+$ is a mapping of a node to the lower and upper bounds on delay.
(5) L_T:$V \rightarrow$ {*Decision, Merge, Action*} is a mapping of a node to the type of node.
(6) L_{power}:$V \times D \times Z^+ \times Z^+ \rightarrow Re^+$ is a mapping denoting the average power dissipated in a die segment ($d \in D$) by an action node ($v \in V$) in a time-slot ($t \in Z^+$) of size Δ, given that the time of completion of the action node is $t_{complete}$ ($t_{complete} \in Z^+$).
(7) $S \subseteq V$ is the set of start nodes such that $\bigcup_{\forall s \in S}\{L_P(s)\} = C$ and $\forall\ s_1, s_2 \in S$ $L_P(s_1) \neq L_P(s_2)$.
(8) $F \subseteq V$ is the set of finish nodes such that $\bigcup_{\forall f \in F}\{L_P(f)\} = C$ and $\forall\ f_1, f_2 \in F$ $L_P(f_1) \neq L_P(f_2)$.

2.2 Thermal Model of a Chip

The duality between heat transfer and electrical phenomenon is the basis for most architectural-level heat transfer models of chips proposed in the literature [Skadron et al. 2004; Shang et al. 2004; Li et al. 2006]. Heat flow is considered analogous to "current" and the temperature difference arising out of heat flow through a thermal resistance is considered analogous to "voltage." The heat holding capacity of a segment of the chip is modeled as a "capacitor" while the thermal resistance is equivalent to a "resistor." Therefore, the dynamic thermal behavior of a chip is modeled as an RC circuit as shown in the example thermal model of a 3-layered chip in Figure 5. The silicon die is considered partitioned into two die segments (*Block*1 and *Block*2 in Figure 5), denoted by the set of die partitions D. The heat dissipation due to convection is modeled as a resistance $R_{convection}$ connected to "ground". Each segment of the chip, which

may be part of the die, spreader, or heat sink, is associated with a node ($n \in N$) in the RC network. Each node is connected to a heat "capacitor" and lateral and vertical resistances corresponding to the segment of the chip it represents. The methodology for calculating the thermal resistances and capacitances is presented in Skadron et al. [2004] and Huang et al. [2006]. RC models for interconnects and vias [Huang et al. 2006] and on-chip network fabric [Shang et al. 2004] have also been developed. The heat dissipation in each die segment is modeled as a "current source" connected to a network node corresponding to the die segment.

Given a chip-thermal model, the trajectory of the node temperatures is described by the differential equation

$$A * dT + B * T = P(t). \tag{1}$$

Here A is the matrix of thermal capacitances, T is the matrix of temperatures of different nodes, B is the matrix of thermal conductance, and $P(t)$ is the matrix of power dissipated at each *RC-network* node (with values which vary with time).

2.3 The Thermal Analysis Problem

The thermal analysis problem seeks to determine the occurrence of hot-spots on a die during the continuous execution of a multiprocessor SoC application. It checks if a segment of the die heats up to a temperature (T_{max}) in violation of temperature bounds imposed on the design. The inputs to the problem are the Chip-Thermal Model (CTM) of the processor and the application graph representing the schedule. The power dissipation of each task in the application graph, in each die segment of the chip, is also required. For this, a mapping (f_{AD}) from power dissipation in architectural units (Ar) to power dissipation in die partitions (D) is necessary. This information is usually derived from chip floorplans (see Figure 6). The power dissipation in a die-segment is the sum of power dissipation in different architectural units comprising the die segment. The power dissipation of each architectural unit is in turn available from power-performance simulation [Brooks et al. 2000] or from temperature measurements during hardware emulation [Ignowski et al. 2008]. Additionally, for architectural units like the clock tree, in this work we distribute the power dissipation (in the clock tree) among all die segments in the proportion of their areas. For example, the power dissipation in die segment "*Rest*" (Figure 6) is the sum of power dissipation in the I-Cache, D-Cache, DTLB, and the branch predictor, and a proportionate portion of the power dissipation in the clock tree.

Problem Statement. The inputs to the thermal analysis problem are:

(1) a die whose silicon layer is partitioned into *die segments* (D).
(2) an application graph G representing the task schedule, including the mapping representing the trajectories of power consumption of action nodes in different *die segments* (L_{power}).

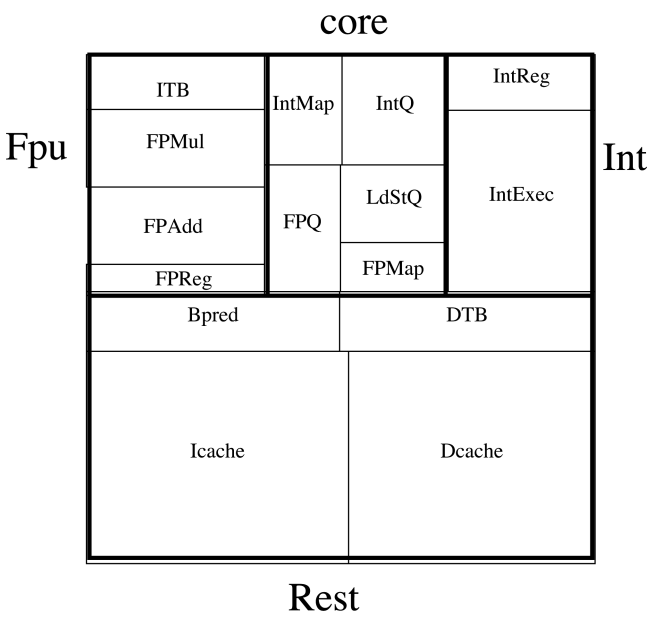

Fig. 6. The floorplan for the ALPHA EV6 core with architectural units marked out. Four die segments are shown in bold.

(3) the chip-thermal model RC-network (A and B denoting the thermal capacitance and conductance, respectively).
(4) the maximum temperature bound (T_{max})

Given the preceding, the thermal analysis problem seeks to determine:

(1) Is there a RC-network node $n \in N$ whose temperature is greater than T_{max}?

It is notable that although the thermal analysis problem is stated as a constraint satisfaction problem, and later formulated as a hybrid automata reachability verification problem (Section 2.4), it is actually a performance analysis problem and not a verification problem. Neither is the satisfaction of the temperature bound (T_{max}) essential, in many cases, for correct functioning of the system, nor can such temperature bounds be guaranteed at deployment due to several reasons like process variations. However, this analysis provides an early insight into the thermal behavior of the MPSoC application, a method for comparing task schedules, and identifying worst-case (thermal) execution paths.

2.4 Problem Formulation as Hybrid Automata Reachability Verification

In this work we propose to model the thermal behavior of a multiprocessor system as a hybrid system [Alur et al. 2006], which we call a *thermal automata*. A hybrid system consists of a *discrete component* which describes the transition between locations and a *continuous component* which describes trajectories of the continuous variables. In the context of the thermal analysis problem, the discrete component describes the changes in configurations of

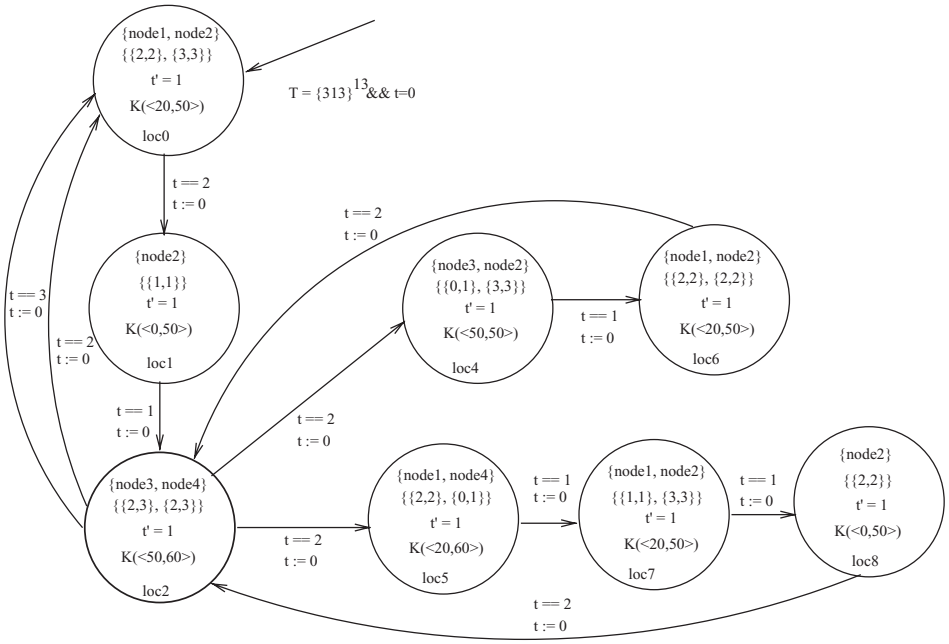

Fig. 7. The thermal automata (a) of the application model presented in Figure 3 for the die in Figure 5. The highlighted location (loc2) is described in the text. We use the notation $K(P(t))$ to denote the equation $dT = -A^{-1}BT + A^{-1}P(t)$.

concurrently executing tasks, while the continuous component describes temperature variations guided by the chip-thermal model. The thermal automata of the *application graph* in Figure 3(a) executing on the die in Figure 5 is shown in Figure 7.

2.4.1 *The Discrete Component.* The discrete component of the thermal automata is described by locations and guarded transitions between them. Each location in the thermal automata is characterized by the actions executing concurrently in the application graph (indicated by the *action node label*). Each location also contains information about the minimum and the maximum time for which an action node can continue to execute in a processor. For example, node3 and node4 can execute for {2, 3} time-slots each in loc2 (Figure 7). A transition from a location is triggered when a clock attains a certain value (*val*), indicating the duration for which the preceding configuration of executing actions may exist. This is modeled by a guard on the transitions of the form t==*val*. Each location is also associated with an invariant ($t \leq y$) restricting the duration of execution in that location to a maximum possible time. Besides this, a reset event t:=0 is associated with each transition, which indicates that the variable "*time*" (t) is set to 0 when the transition is triggered. This underlying discrete structure of the thermal automata is obtained from the application model by a reachability graph construction, as described later in Section 3.2.

Due to *decision* constructs and variation in the execution time of actions, the discrete transitions are nondeterministic. For example, consider the transitions $\{loc2, loc4\}$, $\{loc2, loc0\}$ (with the guard t==3), $\{loc2, loc0\}$ (with the guard t==2) and $\{loc2, loc5\}$. Once the execution starts at location $loc2$, the execution may nondeterministically continue for two or three time-slots. If the execution continues for three time-slots, then $\{loc2, loc0\}$ (with the guard t==3) is triggered. If, on the other hand, the execution counties for two time-slots, then one of $\{loc2, loc4\}$, $\{loc2, loc0\}$ (with the guard t==2) or $\{loc2, loc5\}$ is triggered nondeterministically.

2.4.2 *The Continuous Component.* The continuous component of the thermal automata consists of linear differential equations on the continuous variables of the thermal automata. The set of temperatures (T) of each thermal RC-network node $n \in N$ along with the variable "*time*" (t) comprise the set of continuous variables of the hybrid system. A set of linear differential equations $dT = -A^{-1}BT + A^{-1}P(t)$, on the node temperatures with respect to time describes the thermal model of the chip. The relation $t' = 1$ indicates that the clock progresses at a constant rate. Here, $P(t)$ is the matrix of power dissipated in die segments when the set of action nodes, indicated by the *action node label* of a location, are executing. $P(t)$ is computed in this work as $P(t) \leftarrow \sum_{\forall i \in Labels(l)} P_i(t)$, thereby summing power dissipation of concurrently executing tasks in die segments which are shared by more than one of these tasks (for example, the $L2$-cache).

It may be recalled that power dissipated by a task in an architectural unit varies with each time-slot, hence $P(t)$ is a piecewise constant function. However, we note that the variation in power dissipation between different runs of a task (characterized by the execution completion time) is not represented. The underlying assumption is that when an action node has significantly varying power dissipation profiles for different execution runs, then it is replaced by a set of distinct action nodes, each having power dissipation profiles with little variation with execution runs. Often, the input application graph has to be modified to ensure the same. This is done by splitting an action node into several action nodes which are nondeterministically executed by decision constructs as shown in Figure 8(a) and 8(b). The criteria that split nodes must satisfy is that the power dissipation of an action node, in a given time-slot and die segment, must be uniquely determined by the duration for which the action node has already executed. It should be oblivious of the execution run of an action node, determined by the completion time of the node. Hence $\forall_{d_{node\_id} \le t\_end1 \le D_{node\_id}} \ \forall_{d_{node\_id} \le t\_end2 \le t\_end1} \ \forall_{d \in D} \ \forall_{0 \le t \le t\_end2} L_{power}(d, node\_id, t, t\_end)1 = L_{power}(d, node\_id, t, t\_end)2$). Clearly, one way to attain this objective is splitting the nodes such that they have a unique time of completion ($d_{node} = D_{node}$). However, this often results in a large number of decision branches in the model, which in turn increases the size of the discrete component of the thermal automata (Section 2.4.1). This problem can be addressed in practice by allowing a bounded variation in the power dissipation, in a die segment, for a given time-slot, for different runs of a task. In this case, overapproximation is ensured by assigning the power dissipation of the node

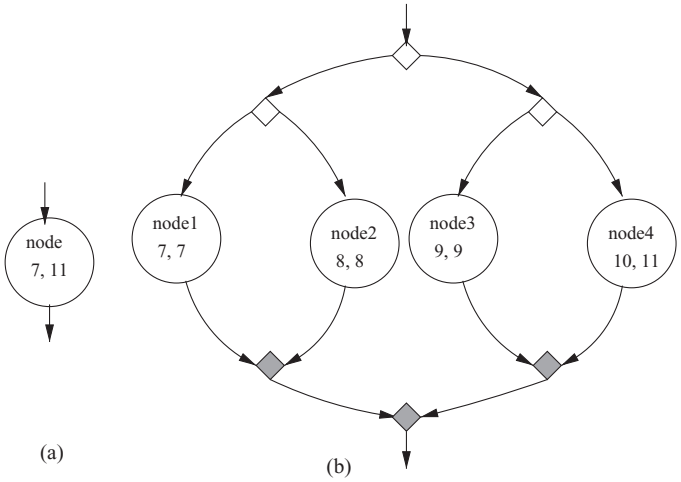

Fig. 8. Splitting the action node *draw_edges* (a) into four action nodes (b) with node1, node2, and node3 having unique times of completion and node4 being associated with a time range.

in a time-slot to the maximum power dissipation (in each time-slot) among different executions.

The node splitting process described earlier ensures that a unique power dissipation pattern for each die segment (using the mapping f_{AD}, Section 2.3) is associated with an action node. In this case the power dissipation profile (in die segments) for each thread $P_i(t)$ can be uniquely determined from the action node currently executing in the thread (at time t), and the duration for which it has already executed. In particular $P_i(t) = \bigcup_{d \in D} L_{power}(d, node_i, D_i - t_{max} + t, \phi)$, where D_i is the upper bound on the time of execution of $node_i$, t_{max} is the maximum time for which $node_i$ can continue to execute in the given location, and t is the clock variable. Note that the value of L_{power} is now oblivious of the execution run characterized by the time of completion of execution of node i (last field of L_{power} is ϕ).

The value of the continuous variables at any instant of time are determined by solving the linear differential equations associated with the appropriate locations. We use the notation $\Phi_l(T_{start}, \tau, t)$ to indicate the solution to the system of equations at time t in a location l, when the initial node temperature at time τ is denoted by T_{start}. The start location is marked by an incoming arrow, and the initial configuration $\langle \{313\}^{13}, 0 \rangle$ (Figure 7) indicates that the temperature of all the 13 nodes is 40 °C (313 K) and the value of the clock is 0 s. The differential equation associated with *loc*2 is written as $K(< 50, 60 >)$ (Figure 7) which represents the equation $dT = -A^{-1}BT + A^{-1}P(t)$, where for all values of t, the matrix element $P(t)[Block1] = 50$, $P(t)[Block2] = 60$ (recall Figures 5 and 3(a)) and $\forall_{i \notin \{Block1, Bloc2\}} P(t)[i] = 0$.

A *state* of the thermal automata denotes the current node temperatures, time, and location of the control. It is defined as a $|T| + 2$ tuple of the form $(loc, \langle T_0, T_1, \ldots, T_{|T|-1}, t \rangle)$, where *loc* is a thermal automata location, t is the value of the timer associated with the location, and $T_0..T_{|T|-1}$ are node temperatures.

There are two types of transitions between thermal automata *states*, namely a *discrete transition* and a *discretized-continuous* transition. A discrete transition relation describes the discrete jump from one location to another, while the discretized-continuous transition models the variations in node temperatures across a time-slot (of size Δ), at a given location. A formal description of the thermal automata is presented in Appendix A.

Once a thermal automata for a multiprocessor SoC application is constructed, the thermal analysis problem directly maps to the *hybrid automata reachability verification* problem. The reachability verification problem checks whether or not any trajectory enters a forbidden region. In the case of thermal analysis, this region is $\bigvee_{temp \in T} temp - T_{max} > 0$.

2.5 Challenges in Problem Specification and Solution

Although the hybrid automata formulation (Section 2.4) of the thermal analysis problem yields a formal framework for the problem, there are some practical challenges in specifying and solving the problem in this manner. Primary among these are incomplete information about power dissipation of tasks and the large number of continuous variables in the nondeterministic *thermal automata*. In this subsection we briefly describe these two issues and state the methods employed to address them.

2.5.1 *Incomplete Information about Power Profiles of Tasks.* An important input to the task-level thermal analysis method is the set of power dissipation profiles of each task, in each die segment, for all runs of the task. A simulation/emulation-based power estimation technique can practically compute power dissipation profiles for only a subset of runs of each task, thereby leaving gaps in the input to the thermal analysis problem.

The state-of-the-art for measuring power dissipation of tasks in various architectural units (and hence die segments), are power-performance simulators like Wattch [Brooks et al. 2000], Ptolemy2 [Buck et al. 1994], Turandot-Powertimer [Moudgill et al. 1999], and Simplescalar PAnalyzer (http://www.eecs.umich.edu/~panalyzer/). These simulators analyze a trace of execution of an application to calculate the power dissipation in different architectural units (and hence die segments). Power dissipation profiles of individual tasks can be obtained from simple modifications to the simulator in conjunction with some instrumentation of the application code. Additionally, power dissipation profiles of tasks, in die segments, may be obtained from calculations based on chip temperature measurements [Wang et al. 2007].

The issue with using these simulation/emulation methods is that they can only analyze one execution run of the application at a time. In order to obtain power dissipation profiles for all possible runs of tasks, characterized by time of execution, at least $max_{v \in V}(D_v - d_v + 1)$ and at most $\Pi_{v \in V}(D_v - d_v + 1)$ simulation runs must be conducted. Additionally, discovering the set of inputs which sensitize all possible execution runs of tasks may require much effort. Hence only a partial set of power profiles of tasks can be obtained practically by power-performance simulation. This leaves a gap in the input to the thermal analysis problem, this gap being the incomplete definition of the mapping L_{power}.

In this work we propose a novel methodology of learning a model of power dissipation of tasks (separately for each die-segment) from available power profiles. Thereafter the remaining power profiles, corresponding to execution runs which were not invoked during simulation, are estimated by querying the learned model. We use a Markov chain learning algorithm [Ron et al. 1996] to learn the power dissipation behavior of a task, and then query it to infer power profiles, thereby filling the input gap.

2.5.2 *Large Number of Continuous Variables and Nondeterminism.* The state-of-the-art for hybrid automata analysis are approximation-based methods involving polyhedral approximation [Asarin et al. 2000; Frehse 2005] and predicate abstraction [Alur et al. 2006]. Various hybrid automata checkers like CHARON [Alur et al. 2000], d/dt [Asarin et al. 2000], and PHAVer (Polyhedral Hybrid Automata Verifier) [Frehse 2005] provide a framework for approximate analysis. A survey of the state-of-the-art for hybrid reachability analysis is found in Asarin et al. [2006]. Although methods using *zonotope* representation [Girard 2005] of the overapproximated reachable region are known to scale well (upto 100 continuous variables) for differential equations of the form: $dT = A * T$, hybridization [Frehse 2005]-based methods used to analyze hybrid systems with differential equations of the form: $A * dT/dt + B * T = P(t)$ do not scale well (beyond 10 continuous variables). It may be recalled that the hybrid system modeling thermal behavior requires differential equations of the form: $A * dT/dt + B * T = P(t)$ (Section 2.4.2). Since typical thermal analysis problems involve 60–120 continuous variables, approximation-based approaches cannot be directly used for the same.

Numerical methods [Barton and Lee 2002; Esposito and Kumar 2007; Park and Barton 1996] for hybrid system analysis, on the other hand, are simulation-based methods which scale well with the number of variables. Most such methods identify discrete transitions and then perform "*discontinuity locking*" assisted backtracking [Barton and Lee 2002] to simulate the system dynamics across a discrete transition. However, most state event simulation methods assume a deterministic hybrid automata model where the first enabled discrete transition is taken [Barton and Lee 2002]. This feature is in contradiction with the requirements for analysis of the thermal automata, where nondeterminism is inherent and essential.

Clearly, state event simulators show an inability to handle nondeterminism while approximation-based methods show an inability to handle a large number of continuous variables, both of which are needed for solving the thermal analysis problem. However, unlike many other hybrid systems, the guards on discrete transitions of the thermal automata are predicates depending only on the value of the timer. This allows us to analyze and predict discontinuity in the behavior of the system with ease. We take advantage of this property of the thermal automata and develop a numerical analysis-based hybrid state-space exploration algorithm. Also, the issue of large amount of nondeterminism in the thermal automata is tackled by a CounterExample-Guided Abstraction Refinement (CEGAR) [Clarke et al. 2003] method in conjunction with several heuristic-based pruning methods.

3. ALGORITHMS

In this section we present the algorithms and proposed methods to formulate and solve the thermal analysis problem. First, we present a machine learning-based approach for obtaining the power profiles for all execution runs of tasks. Thereafter, we present the algorithm for constructing the discrete component of the thermal automata, and finally present the CEGAR-based algorithm for solving the thermal analysis problem. We also propose several engineering improvements to the CEGAR-based algorithm, and a method to perform directed simulations.

3.1 Obtaining Power Dissipation Profiles of Tasks

The set of power dissipation profiles (in each die-segment) for every execution run of each task is an important ingredient for thermal analysis. Unfortunately, there are no known empirical methods or tools which can be used to obtain this data in a practical manner. Hence, the power dissipation profiles (in each die segment) for several execution runs must be estimated from available power dissipation profiles of other runs, obtained from power-performance simulation [Moudgill et al. 1999; Brooks et al. 2000] or hardware emulation [Ignowski et al. 2008]. To address this issue, we propose a machine learning approach based on Markov chains to estimate the power dissipation profiles for every execution run of a task. In the proposed method one Discrete-Time Markov Chain (DTMC) is learned for modeling power dissipation of all runs of each task in each die segment of the chip. Hence for a task-graph with $|V|$ action nodes (tasks) executing on a chip with $|D|$ die-segments, at most $|V \times D|$ DTMCs will be learned (each task may not dissipate power in all die segments). Thereafter, each learned DTMC is queried to obtain power dissipation profiles of a task, in a specific die-segment, for all runs of the task.

The learning-based estimation method consists of three steps, namely obtaining training examples, learning the DTMCs, and analyzing the learned DTMCs for estimating power dissipation profiles (for each die-segment) of tasks, for all execution runs. In this subsection, we present the intuition behind using DTMCs as the model for task-level power dissipation. Appendix B.1 presents the method employed for obtaining training examples for the machine learning step. We use the machine learning algorithm proposed by Ron et al. [1996] for learning a DTMC from a set of training examples presented as strings. Appendix B.2 presents the method employed to analyze a learned DTMC to estimate the power dissipation profile (in a die segment) for each run of the task.

The use of Markov chain as the underlying analytical model of power dissipation is guided by similar uses for system-level performance/power analysis [Zamora et al. 2007] and analysis of computation and communication for MPSoC design [Marculescu et al. 2006]. Intuitively, the behavior of a task can be envisioned as a graph where nodes represent subtasks executing in a particular processor state and edges represent transitions between these subtask executions. Each subtask executing in a different processor state dissipates a certain amount of power in a die-segment d. A finite sequence of nodes corresponds to

the power dissipation profile for one run of the task, in the die-segment d. The transition between nodes may be memoryless or may depend on the previous L nodes visited. For example, the next state of a cache implementing a LRU replacement policy is dependent not only on the current state but also on the sequence in which data items were added to the cache. For most practical scenarios this sequence of cache states needed to determine the next state will be bound by a finite constant.

It may be recalled that in this work we discretize time into intervals and the power dissipation in each interval is considered constant (see Section 2.1). Hence the power dissipation profile for one run of a task, in one die-segment d, may be viewed as a string where each symbol denotes the average power dissipated by the task in one time slot in that die-segment (d). This is also consistent with cycle-wise or time-slot (consisting of several cycles)-based power traces of the application obtained from simulation runs on a power/performance simulator. Hence, in this work we use a Discrete-Time Markov Chain (DTMC) formalism since it naturally fits into the intuitive performance model of a task presented earlier and is consistent with the notion of discretized time and power dissipation used in this work. Additionally, in order to reflect the dependence of performance on a finitely bound history of execution of length L, *Markov chains of order L* [Ron et al. 1996] are used in this work. A Markov chain of order L can also be represented by a *L-Probabilistic Suffix Automata* (*L-PSA*) model, which in turn is a subclass of *Probabilistic Finite Automata* (*PFA*) [Ron et al. 1996].[4]

Given a set of training instances as strings over Σ and a bound L, an algorithm for learning L-PSA (and hence a Markov chain of order L) is presented in Ron et al. [1996]. We use this algorithm for learning a set of PFAs, each of which models the power dissipation of one action node $v \in V$ in a particular die segment ($d \in D$). It may be noted that, due to task mapping, most tasks dissipate power in a subset ($D_{power} \subset D$) of all the die segments of the chip, and not in all die segments. Hence at most $|V \times D|$ PFAs are learned.

An example PFA for the power dissipation of action node "edge_draw" in the die segment "Fpu" of the floorplan shown earlier in Figure 6 is presented in Figure 9. This die segment consists of the floating point adder, multiplier, and registers, and the Instruction Translation Lookaside Buffer (ITLB). The power dissipation of a die segment is computed as the sum of power dissipation of architectural units mapped to the die segment in question. Once these PFAs are learned, they are analyzed to obtain power profiles for all possible execution runs of tasks (details presented in Appendix B.2).

3.2 The Algorithm for Constructing the Execution Automata

The discrete component of the thermal automata is called an *execution automata*, which is similar to a reachability graph of the application. It consists

[4]A PFA M is a 5-tuple $(Q, \Sigma, \tau, \gamma, \pi)$, where Q is a finite set of states, Σ is a finite alphabet, $\tau : Q \times \Sigma \rightarrow Q$ is the transition function, $\gamma : Q \times \Sigma \rightarrow [0, 1]$ is the next symbol probability function, and $\pi : Q \rightarrow [0, 1]$ is the initial probability distribution over the starting states. The constraints $\forall_{q \in Q} \Sigma_{\sigma \in \Sigma} \gamma(q, \sigma) = 1$ and $\Sigma_{q \in Q} \pi(q) = 1$ must be satisfied by any PFA.

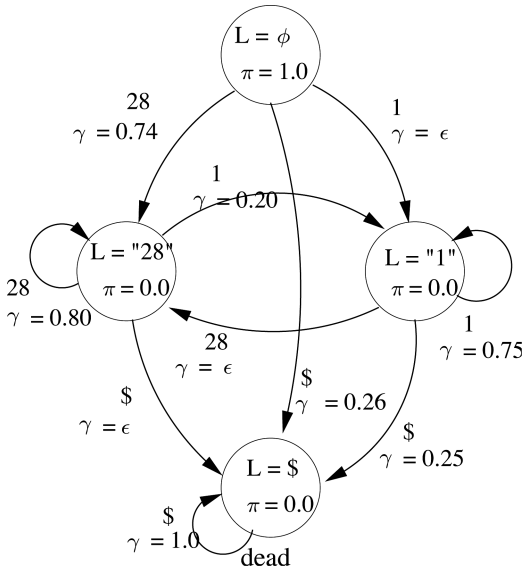

Fig. 9. A probabilistic finite automata modeling the power dissipation of "edge_draw" action node (Figure 1) in die segment "Fpu" (Figure 6), consisting of the FP adder, FP multiplier, FP registers, and Instruction Translation Lookaside Buffer (ITLB). It consisting of two symbols "1" and "28" corresponding to a power dissipation of 0.10 and 2.89 W, respectively. $ is the terminating symbol.

of *execution states* and *transitions* between them. Each execution state is associated with a set of action nodes executing, the duration of occurrence of the execution state, and the minimum and maximum additional time for which each action node can execute. The *transitions* reflect the the functional behavior of the system describing legal changes in the configuration of execution.

Formally the *execution automata* is defined as a tuple $EA : \langle S_{EA}, T_{EA}, S_0, L_{EA}, G_{EA} \rangle$.

(1) S_{EA} is a set of execution states.
(2) T_{EA}:$S_{EA} \times S_{EA} \times Z^+$, is a set of transitions between execution states which is labeled by the value of the timer at which the transition can occur.
(3) $s_0 \in S_{EA}$ is the start execution state of the transition system.
(4) L_{EA} is a labeling, mapping execution states to the set of concurrently executing actions nodes in each processor ($\emptyset$ if no action is executing in a processor) and the minimum and maximum time for which an action may execute after the attainment of the execution state. For example, consider the execution state *loc*2 in Figure 10(a), which indicates concurrently executing nodes *node*3 and *node*4. Here, $L_{EA}(loc2)$={$\langle node3, 2, 3\rangle$, $\langle node4, 2, 3\rangle$} indicates that both nodes *node*3 and *node*4 may execute for a minimum of 2 time-slots and a maximum of 3 time-slots. Formally $L_{EA} : S_{EA} \rightarrow (\{V \cup \emptyset\} \times Z^+ \times Z^+)^{|C|}$, where C is the set of processors.

The construction of the execution automata (*CreateEA*(), Figure 11) starts with creating the start execution state s_0 which is labeled with the start set of

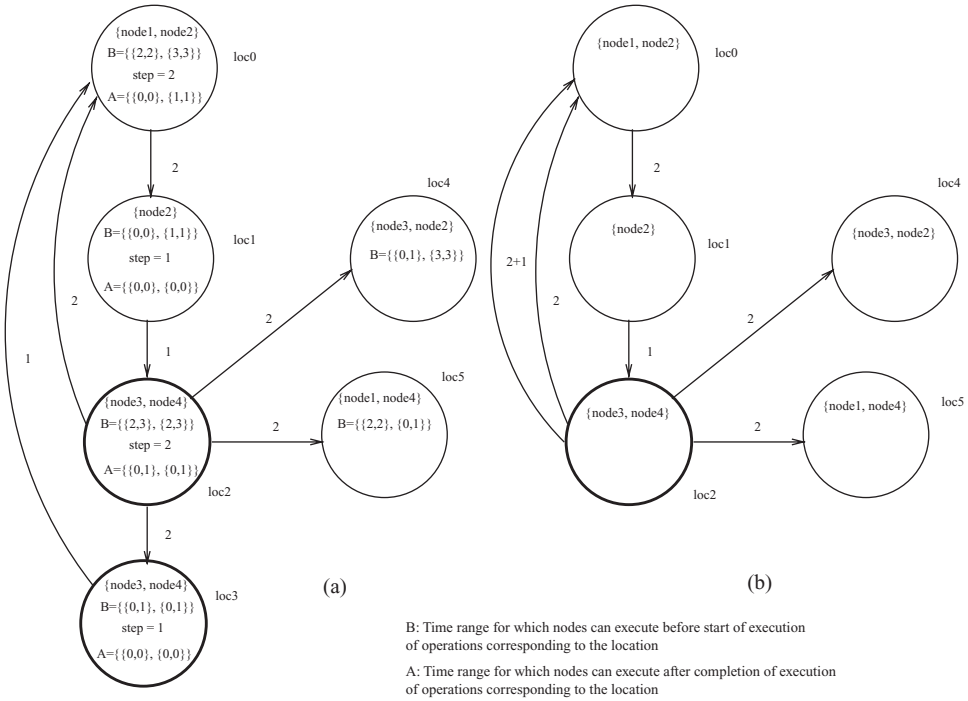

Fig. 10. Two intermediate snapshots of the execution automata of the application graph in Figure 3(a) illustrating the merging of states *loc*2 and *loc*3. The highlighted locations are described in the text.

the application model (S) and the minimum and maximum time of execution of these actions. This execution state is added to a priority queue of execution states (S_{EA}) and marked as *unexplored* (line 2, *CreateEA*(), Figure 11). This priority queue is sorted by the minimum time elapsed to reach an execution state from the start execution-state (s_0). Thereafter, each *unexplored* execution state $s \in S_{EA}$ from the head of the priority queue is *explored* (lines 3–5, *CreateEA()*, Figure 11) to discover new execution states and transitions between s and these newly discovered or revisited execution states.

The exploration of an execution state consists of four steps, presented in Figure 11 and illustrated through the exploration of execution state *loc*2 (Figure 10). The illustration presented in Figure 10 consists of two snapshots during the construction of the execution automata of the application graph introduced in Section 2.1 in Figure 3(a). The steps for exploration of an execution state are presented next:

(1) Finding the minimum time "*step*" for which the execution state is guaranteed to exist (line 2, *ExploreNode*(), Figure 11). For example, the execution state *loc*2 in Figure 10(a) is guaranteed to exist for 2 time-slots, since both node3 and node4 execute for {2, 3} time-slots.
(2) Find which processors will execute a different action (a *decision* to "*Change*" node of execution in the processor), continue to execute the same action

```
/* (s.nodes)_p is the action node executing in processor p in execution-state s */
/* (s.min)_p/(s.max)_p is the min/max time for which action in processor p can execute */
/* s.min_time is the minimum time elapsed to reach s from the start node s_0 */
/* queue is a priority queue of unexplored execution-states sorted by s.min_time */

ExploreNode()
1:   input execution-state s, application-graph G output execution-automata EA
2:   for all p, step ← Max(Min(step, (s.min)_p), 1)/*smallest (s.min)_p or 1 if all (s.min)_p=0*/
3:   for all p, (s.min)_p ← Max((s.min)_p - step, 0), (s.max)_p ← Max((s.max)_p - step, 0)
     /* New value of (s.min)_p and (s.max)_p after execution for "step" time-slots */
4:   for all p, decisions_p ← "Change", "Undecided" or "Keep" depending on
5:   if (s.max)_p = 0, {(s.min)_p = 0 ∧ (s.max)_p > 0} or {(s.min)_p > 0}
6:   for all combinations π of processors with decisions_p = "Undecided"
7:     for all combinations σ of decisions at decision nodes (if any)
8:       for all p, (new.nodes)_p ← Succ_πσ((s.nodes)_p)
       /* (new.nodes)_p is assigned to the next action in processor p,
       when actions in the combination π and those with decision_p = "Change" complete
       and the decisions in the combination σ are taken */
9:       for all p, ⟨(new.min)_p, (new.max)_p⟩ ← L_D((new.nodes)_p)
10:      if new ∉ S_EA
11:        S_EA ← S_EA ∪ {new}, T_EA ← T_EA ∪ ⟨s, new, step⟩
12:        queue.add(new), new.min_time = s.min_time + step
13:      else T_EA ← T_EA ∪ ⟨s, new, step⟩
           new.min_time = min(s.min_time + step, new.min_time)

CreateEA()
1:   application-graph G output execution-automata EA
2:   queue.add(s_0)
3:   while ¬queue.empty()
4:     s = queue.remove()
5:     ExploreNode(s) /* Explore un-explored node with minimum value of min_time*/
6:     if (s.nodes = parent(s).nodes ∧ s.num_inedges = 1)
7:           for all {s, v, time} ∈ T_EA, delete node s and edge ⟨s, v, time_s,v⟩ and
           add an edge ⟨parent(s), v, time_s,v + time_parent(s),s⟩
```

Fig. 11. Algorithm for constructing the execution automata of the model of the multiprocessor application.

(*decision* is "*Keep*"), or may either continue to execute the current action or execute a different action (*decision* is "*Undecided*") in the next execution state (lines 4–5, *ExploreNode*(), Figure 11). In execution state *loc*2 the nodes *node*3 and *node*4 may either execute for 2 or 3 time-slots. When any of these nodes executes for 2 time-slots, the processor containing that node must have a change in the action executing on the processor. On the other hand, when a node executes for 3 time-slots, the processor containing that node does not see a change in the action being executed (*step* is 2 time-slots). Hence, the status of the processor is "*Undecided*", as it depends on the nondeterministic choice of execution time of the node executing in it. This is the case with both processors A and B in *loc*2.

(3) Explore execution states by making all combinations of changes for nondeterministic choices arising out of decision branches and variable execution delays (lines 6–13, *ExploreNode*(), Figure 11). For example, at the execution state *loc*2 the algorithm explores all the combinations to search execu-

tion states $loc0$ (Processors A and B discontinue executing current nodes), $loc3$ (both continue), $loc4$ (Processor A continues) and $loc5$ (Processor B continues).

(4) Once an execution state has been explored and has exactly one incoming transition, merge execution states which have the same actions executing (lines 6–7, *CreateEA*(), Figure 11). For example, execution states $loc2$ and $loc3$ have the same nodes executing and hence they are merged as shown in Figure 10(b).

Once the execution automata has been constructed, the thermal automata is easily constructed by associating the appropriate differential equations $dT = -A^{-1}BT + A^{-1}P(t)$ and $t' = 1$ with the locations (execution states).

Thereafter, in order to solve the thermal analysis problem, the reachability analysis problem must be solved. However, the size of the problem (in the number of continuous variables and nondeterministic branching in the execution automata) makes it difficult to directly apply known hybrid automata analysis methods. In the next section we propose a CEGAR-based engineering solution to tackle these issues.

3.3 Solving the Thermal Analysis Problem

In this section we present a CEGAR (CounterExample Guided Abstraction Refinement)-based hybrid automata reachability verification method for solving the thermal analysis problem (Figure 12). The proposed algorithm performs an *execution-time* bounded depth-first traversal of all hybrid-automata states by systematic invocation of all combinations of discretized-continuous and discrete transitions from discovered states (lines 17–23, Figure 12).

Discretized-continuous transitions are explored by applying the adaptive Runge Kutta [Press et al. 1992] method for the duration of a time slot (Δ) associated with a discretized-continuous transition. Since the state-space exploration is done in a depth-first manner, the stack of the DFS search algorithm contains a partial or complete execution trace. Although new states are discovered on invoking both the discrete and discretized-continuous transitions, the proposed algorithm only stores those states discovered after invoking a discrete transition. However, all states discovered by invoking discretized-continuous transitions are checked for the reachability property.

Since a state consists of several (50–150) continuous variables, the probability of two states being exactly the same is very low. Hence, the nondeterminism in the thermal automata translates to an exponential (in the *execution time* bound) number of states being explored. Typically, for the experiments performed in this work, the number of states is expected to be of the order of 2^{300}. In order to tackle this problem we propose the use of an *abstraction-refinement* paradigm. In this paradigm, the node temperatures (between ambient temperature and the maximum temperature bound T_{max}) are partitioned into a finite number of intervals (I_T). Each node temperature associated with a state is mapped to an interval ($int_i \in I_T$) by a location-dependent *abstraction operation* $\psi(loc)$:$\Re^{+|T|} \rightarrow I_T^{+|T|}$ ($loc \in L$, is a thermal automata location). This creates an *abstract-state* (line 22, Figure 12) corresponding to each state. Each *abstract-state* is defined as a $|T| + 2$ tuple of the form

HeatCheck()
input thermal automata H, stack (the DFS stack)
1: **if** $violates\_temperature\_bounds(X^{abs}_{stack[stack\_top].location})$
2: **/*counterexample-guided refinement needed*/**
3: **for** i = 0 to $stack\_top$-1
4: loc ← stack[i+1].location
5: $\chi_{loc} \leftarrow$ **simulate_rk4**($\chi_{stack[i].location}$, stack[i+1].time - stack[i].time)
6: $int \leftarrow \psi_{loc}(\chi_{loc}, loc)$ /* int is the set of intervals containing χ_{loc}*/
7: **if** $violates(\chi_{loc})$ **/* A concrete counterexample is obtained*/**
8: **dump_stack**(i), **return** FALSE
9: **/*refine in accordance with the counterexample*/**
10: $\forall_{t \in T}$ add $[int[t].lower, \chi_{loc}[t] - W_{lower})$, $[\chi_{loc}[t] - W_{lower}, \chi_{loc}[t] + W_{upper})$ and
11: $[\chi_{loc}[t] + W_{upper}, int[t].upper)$ to set of intervals I_T to reduce overapproximation
12: $\forall_{t \in T}$ add $[\chi_{loc}[t] - \epsilon, \chi_{loc}[t] + \epsilon)$ to I_T to discharge counterexample
13: $X^{abs}_{loc} \leftarrow \{\psi_{loc}(\chi_{loc}), loc\}$
14: **else if** $stack[stack\_top].time < time\_limit$
15: **/*state space exploration in abstract domain*/**
16: $current\_loc = stack[stack\_top].location$
17: **for** all discrete transitions $\langle current\_loc, new\_loc\rangle$
18: /*Let $X_{current\_loc} = \langle \bigcup_{k \in T} int[k], stack[stack\_top].time\rangle$*/
/*obtain overapproximated state from abstract-state*/
19: $X_v \leftarrow \{\bigcup_{t \in T} int[t].upper, current\_loc\}$
/*take discretized-continuous step*/
20: $X_v \leftarrow$ **simulate_rk4**(X_v, $\langle current\_loc, new\_loc\rangle.time\_elapsed$)
21: $stack\_top \leftarrow stack\_top + 1$, $stack[stack\_top].location \leftarrow new\_loc$
/*obtain new abstract-state*/
22: $X^{abs}_{stack[stack\_top].location} \leftarrow \langle \psi_{new\_loc}(\chi_v), new\_loc\rangle$
23: **HeatCheck**(H, $stack$)

Fig. 12. Algorithm for analyzing the thermal automata for violation of temperature constraints. χ denotes the vector of temperatures.

$(loc, \langle int_0, int_1, \ldots, int_{|T|-1}, t\rangle)$, where loc is a thermal automata location, t is the value of the timer associated with the location, and $int_0..int_{|T|-1}$ are node temperature intervals.

An abstract-state X^{abs}_1, is said to *dominate* another abstract-state, X^{abs}_2, if X^{abs}_1 corresponds to higher node temperatures at an earlier time instant at the same location, that is, $X^{abs}_1.loc = X^{abs}_2.loc$, and $X^{abs}_1.time \leq X^{abs}_2.time$, and $\forall_{i \in N}(X^{abs}_1.int_i.end \geq X^{abs}_2.int_i.end)$ (here $int_i.end$ denotes the end, or the largest value, of the interval int_i). Each *abstract-state* corresponds to an *abstract domain* which is described by a two tuple $(\langle I_T, \psi()\rangle)$ consisting of a set of intervals (I_T) and the location-dependent abstraction operation $(\psi())$.

Discrete transitions are applied on the abstract-states in the same manner in which it is applied on unabstracted states. The discretized-continuous transitions are applied on an abstract-state in three steps. The first step creates a temporary *state* consisting of the upper bound of the intervals in the abstract-state, thereby overapproximating the current temperature (line 19, Figure 12). The second step consists of invoking the continuous dynamics for a specified amount of time (line 20, Figure 12). It may be noted that while computing the iteration coefficients of the Runge Kutta method, the proposed algorithm uses the lower bounds of the intervals associated with the abstract-state (overapproximates rate of heating for the monotonic differential equations).

The combination of the previous steps ensures an overapproximation of the continuous dynamics in the abstract paradigm. Finally, the new abstract-state is obtained by applying the abstraction operation $\psi(loc)$ (line 22, Figure 12). It may be noted that in order to reduce the overapproximation in the proposed algorithm, the abstract-states are not obtained after every discretized-continuous transition is invoked. Instead, the abstract-states are evaluated when a discrete transition is triggered after applying a transitive closure of discretized-continuous transitions. Besides this, when a newly discovered abstract-state X_{new}^{abs} is *dominated* by an explored abstract-state X_{old}^{abs}, it is not explored any further.

If there are no violations of the thermal properties in the overapproximated abstract domain, then there is a guarantee that the actual system is safe. However, the converse is not true and hence any counterexample *run* (stored in the DFS stack) which produces a thermal property violation must be evaluated without any abstraction (lines 5–7, Figure 12). If the counterexample is spurious and it does not violate any temperature bounds, then the algorithm performs a counterexample-guided refinement. Otherwise, the counterexample is dumped into a log and a violation is notified (line 8, Figure 12).

The counterexample-guided refinement consists of adding more intervals, to create a new abstract domain, so that the spurious counterexample does not violate the thermal property in the new abstract domain. Once the new intervals are added, the spurious counterexample is reevaluated in the new abstract domain. During the simulation of a spurious counterexample, for each node temperature (*node_temp*) obtained at the time of execution of a discrete transition, a new interval $[node\_temp - \epsilon, node\_temp + \epsilon)$ ($\epsilon < 10^{-8}$) is added (line 12, Figure 12) to the set of intervals (I_T). This interval (for example, *int C* in Figure 13) ensures that in the new abstract domain constructed with modified intervals, the spurious counterexample behaves very closely to the way it would have had there been no abstraction. Moreover, intervals containing *node_temp* are split into smaller intervals in order to reduce the amount of overapproximation for subsequent state-space exploration (lines 10–11, Figure 12). For this, three intervals $[int.lower, node\_temp - W_{lower})$, $[node\_temp - W_{lower}, node\_temp + W_{upper})$, and $[node\_temp + W_{upper}, int.upper)$ are added, for an appropriately chosen W_{lower} and W_{upper} (lines 10–11, Figure 12). Here W_{lower} and W_{upper} are selected in such a way that the new intervals are always contained inside the original interval (*int*). These intervals (for example, *int B* in Figure 13) tighten the bound on the amount of abstraction allowed and help in avoiding spurious counterexamples in the future. It may be noted that intervals on the temperature scale are ordered ($\preceq$) in terms of one interval being completely included in another. Formally $int_u \preceq int_v$ *iff* $int_u.lower \geq int_v.lower$ and $int_u.upper \leq int_v.upper$. Besides this, the refinement operation involving the addition of new intervals preserves this ordering between newly added intervals and those which are already present. Figure 13 shows the ordering between different intervals added as a result of refinement. Here we would like to state that while constructing the abstract-state, the abstraction operation ($\psi(loc)$) always selects the shortest interval (minimum value of *int.upper-int.lower*) corresponding to a node temperature in a state.

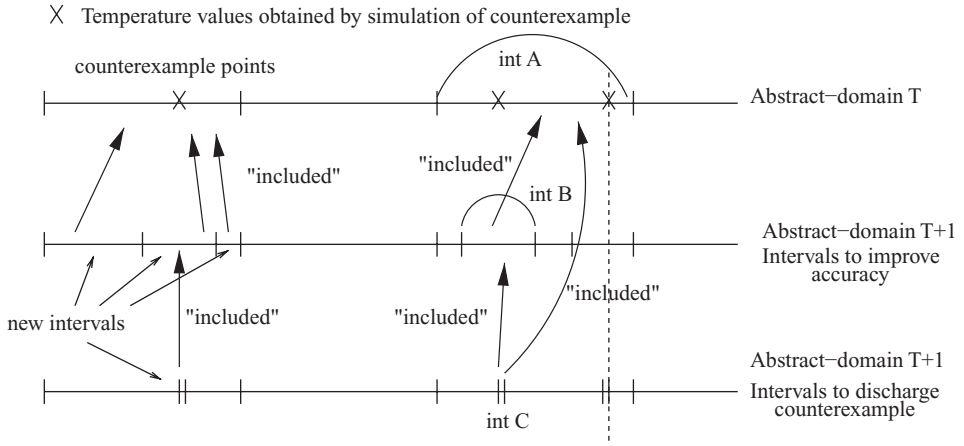

Fig. 13. Counterexample-guided refinement, showing new intervals added on the basis of the temperature values attained after simulation of the counterexample. It also shows the "included" ($\preceq$) relation between intervals ($u \preceq v$ if there is an arrow pointing from u to v). For any interval int, $int \preceq int$.

We observe that the states which have already been explored and do not violate the thermal properties must not be rechecked in a new abstract domain. This causes different portions of the abstract-state space to be in different abstract domains. In order to compare abstract-states from different domains for matching states, we introduce the notion of the *last refined state on the stack* (*LRSOS* state). Each abstract-state (X^{abs}) is associated with the identifier of the latest state which was refined, and is on the DFS stack, at the time when X^{abs} is explored (the LRSOS state). A new abstract-state X^{abs}_{new} is compared with a discovered abstract-state X^{abs}_{old} if either $X^{abs}_{old}.lrsos$ is the initial abstract-state X^{abs}_0 (corresponding to ambient temperatures and the start location, at time $t = 0$), or $X^{abs}_{new}.lrsos = X^{abs}_{old}.lrsos$. Thereafter, if X^{abs}_{old} *dominates* X^{abs}_{new}, X^{abs}_{new} is not explored any further. This is because, if $X^{abs}_{old}.lrsos = X^{abs}_0$, then X^{abs}_{old} has the largest possible intervals, which had allowed greater (or equal) overapproximation than possible from any abstract-state, including X^{abs}_{new}. Additionally, the temperature bounds were not violated by any run involving X^{abs}_{old}, since the analysis halts after the first counterexample is encountered. Hence, if an abstract-state X^{abs}_{new} is dominated by an abstract-state X^{abs}_{old}, then no run from X^{abs}_{new} can violate the temperature bound, since no run from X^{abs}_{old} did. On the other hand if $X^{abs}_{new}.lrsos = X^{abs}_{old}.lrsos$, then X^{abs}_{old} and X^{abs}_{new} are in the same *abstract domain*, and are associated with the same set of intervals (I_T). Moreover, the sequence of thermal automata locations which are analyzed without any approximation (up to the abstract-state $X^{abs}_{old}.lrsos$), is the same for runs leading up to both X^{abs}_{new} and X^{abs}_{old}. Additionally, subsequent analysis from the abstract-state $X^{abs}_{old}.lrsos$ (and also $X^{abs}_{new}.lrsos$) to both X^{abs}_{old} and X^{abs}_{new}, occurred with the same sets of intervals (I_T), and therefore the same amount of overapproximation (the set of intervals are updated only in case of a refinement). In

this case, if X_{old}^{abs} dominates X_{new}^{abs}, then each run from X_{new}^{abs} is guaranteed not to violate the temperature bounds.

3.4 Engineering Improvements to the Thermal Analysis Method

In this subsection we propose several engineering enhancements to the CEGAR-based thermal analysis method presented earlier. These methods are a heuristic-based state-space pruning, and state-matching, analyzing a subset of discrete transitions at each thermal automata location, and storing only those abstract-states which are associated with a small subset of thermal automata locations.

3.4.1 *Heuristic-Based Pruning.* Since a time-bounded analysis is performed, several heuristic methods can be applied to prune the state-space search when there is a guarantee that the thermal property cannot be violated within the time bound. This is done by checking if the temperature limit T_{max} is greater than the sum of the current node temperature ($T_{current}$) and an overapproximation of the amount by which the node temperatures can increase. Hence we check if $T_{max} > T_{current} + (t_{bound} - t_{current})R_{max}$, where R_{max} is the overapproximated estimate of temperature increase, t_{bound} is the time bound, and $t_{current}/T_{current}$ is the current time/node temperatures. R_{max} is precomputed as the maximum rate of temperature increase, at ambient temperature, among all locations.

3.4.2 *Heuristic-Based State-Matching.* It may be recalled that while checking if an abstract-state X_1^{abs} *dominates* X_2^{abs}, we checked whether $X_1^{abs}.time \leq X_2^{abs}.time$ and $\forall_{i \in N}(X_1^{abs}.int_i.end \geq X_2^{abs}.int_i.end)$. When $X_1^{abs}.time > X_2^{abs}.time$, the check for dominance can be relaxed to checking $\forall_{i \in N}(X_1^{abs}.int_i.end \geq X_2^{abs}.int_i.end + R_{max}^i(X_1^{abs}.time - X_2^{abs}.time))$. Here R_{max}^i is the precomputed maximum rate at which temperature of node i can increase at any location (at ambient temperature).

3.4.3 *Analysis of a Subset of Transitions.* Several discrete transitions from a location l model the effect of the execution state (configuration of executing tasks) executing for successively increasing time duration. Hence, these transitions are of the form $\langle l, l'_0, k\Delta\rangle$, $\langle l, l'_1, (k+1)\Delta\rangle, \ldots, \langle l, l'_i, (k+i)\Delta\rangle$. Due to this, the branching at each location is sometimes high (> 100), thereby limiting the scope of an exhaustive analysis. In order to reduce the effects of high branching at each location, we propose a method wherein only one out of k discrete transitions are examined during thermal analysis. This mimics the effect of lower branching achieved when the size of time-steps is k-times larger (or $k\Delta$). However, unlike altering the size of time-steps, this method does not change the underlying discrete behavior of the application (execution automata is not affected), but at the same time it is unable to leverage on any gains in terms of fewer calls to the differential equation solver, which might result from increasing the time-step.

3.4.4 *Storing States at a Subset of Locations.* Every time the abstraction operation is invoked to obtain an abstract-state from a given thermal state, a certain amount of overapproximation occurs. This overapproximation may become significant as compared to the rise in temperature, if there are discrete transitions at short time intervals (within which there is a small temperature rise/fall). This overapproximation usually leads to spurious counterexamples being analyzed, and hence increases the time taken for thermal analysis. This problem is compounded if the peak temperature is close to the temperature bound being analyzed.

In this work we propose an engineering enhancement, wherein states are stored at only a subset of locations of the thermal automata ($L_s \subseteq L$). The smallest set of such locations must contain at least one location in each cycle of the thermal automata H. In particular, we perform a depth-first traversal of the thermal automata, and on encountering a back-edge we check whether a location in L_s is present in the cycle thus formed. If there is one such location, then the set L_s is left unaltered. Otherwise, we add the location with the incoming back-edge to the set L_s. We call the locations in L_s, "loop-back"-locations.

3.5 Directed Simulation for Thermal Analysis

In order to study the thermal profile of an application, it is often necessary to perform thermal analysis for long simulation-time durations (> 10 s). However, for several models it is not possible to perform exhaustive analysis for long simulation-time durations given a reasonable time bound (say < 3 hrs), and computing resources. In this subsection we propose a directed simulation-based approach for performing analysis for long simulation-time durations.

The directed simulation-based approach consists of first performing a time-bounded (T_{bound} time-units) exhaustive analysis of the thermal automata, and identifying execution cycles which are expected to cause the highest temperature rise in different cores ("hot"-execution cycles). Thereafter, the selected execution cycles are simulated for a long time duration of T_{long} time-units. At this point, the set of "hot"-cycles may be revised by performing the exhaustive analysis from a different chip temperature point, and the simulation may be continued.

3.5.1 *Identifying "Hot"-Execution Cycles.* An execution path (π_H) is defined as a finite ordered sequence of thermal automata discrete transitions, $\pi_H : \langle l_0, l_1, y_0\rangle, \langle l_1, l_2, y_1\rangle, \ldots, \langle l_k, l_{k+1}, y_k\rangle$. It may be recalled that the model of the application consists of loops connecting the *finish* and *start* nodes in each processor/core. Due to the presence of these loops, the execution automata, and hence the thermal automata, has cycles comprising of discrete transitions; $\sigma_H : \langle l_c, l_{c+1}, y_c\rangle, \langle l_{c+1}, l_{c+2}, y_{c+1}\rangle, \ldots, \langle l_{c+m}, l_c, y_{c+m}\rangle$. Such being the case, an execution path containing a sufficiently large number of discrete transitions is expected to contain one or more iterations of one or more thermal automata cycles. These cycles can be easily identified by examining the given execution path π_H.

Additionally, we argue that if an execution path π_H contains several cycles $\sigma_H^0, \sigma_H^1, \ldots, \sigma_H^j$, then there is another execution path π'_H containing repeated invocations of only one cycle, say σ_H^i $(0 < i < j)$, which heats a given thermal RC-network node $n \in N$ more than or close to π_H. Here σ_H^i heats up the given thermal RC-network node more than the cycles $\sigma_H^0, \sigma_H^1, \ldots, \sigma_H^{i-1}, \sigma_H^{i+1}, \ldots, \sigma_H^j$. Therefore path π'_H is of the form $\pi'_H : \pi_{pre}(\sigma_H^i)^p \pi_{post}$ (with $\pi_{post} \subset \sigma_H^i$)). An exhaustive analysis, for a sufficient time-bound T_{bound}, is guaranteed to explore this path π'_H. Once this path π'_H is analyzed, the cycle σ_H^i can be identified, and the execution path $\pi_{pre}(\sigma_H^i)^q$ can be simulated for a user-defined time limit. This process of identifying "hot"-execution cycles, and simulating them for long time durations, is repeated for each thermal RC-network node $n \in N$ of interest to the designer.

It may be noted that execution paths, which are candidates for containing the "hot"-execution cycle, are those which correspond to a sequence of *non-dominated* abstract-states during exhaustive thermal analysis. Therefore, only those execution paths which breach the time-bound T_{bound} (no abstract-states were dominated) are candidates for containing the "hot"-execution cycle. Additionally, if at a later stage during exhaustive thermal analysis, some execution path corresponds to an abstract-state X_{new}^{abs} at a location l, such that X_{new}^{abs} dominates the abstract-state X_{old}^{abs} corresponding to a candidate execution path π_H^i, then this path π_H^i is discarded from the set of candidate execution paths (Π_H). The candidate execution path, $\pi_H^{hi} \in \Pi_H$, which causes the highest rise in the temperature of the thermal RC-network node n is analyzed for identifying the "hot"-execution cycle. We note that for this thermal analysis, we reduce the initial size of intervals significantly (upto 100X) to distinguish between execution paths having small temperature differences. It is expected that these small differences will magnify when the simulation is performed for a long time duration.

3.6 Accounting for Leakage Power

Due to technology scaling, leakage power [Su et al. 2003] has emerged as an important source of power dissipation which must be accounted for in any thermal analysis. Leakage power is *static* as it is independent of the executing application, and is dependent only on circuit components which are powered "on". However, leakage power is exponentially dependent on temperature [Su et al. 2003]. This creates a feedback loop during thermal analysis, wherein power dissipation is dependent on chip temperature, and chip temperature in turn is dependent on power dissipation.

In order to account for leakage power in the proposed method, we modify the equation describing the trajectory of node temperatures as

$$A * dT + B * T = P(t) + L(T). \tag{2}$$

Here $P(t)$ denotes the vector of dynamic power dissipated in each thermal RC-network node, while $L(T)$ denotes the vector of leakage power dissipated at each thermal RC-network node. As before, A is the matrix of thermal capacitances, T is the matrix of temperatures of different nodes in the thermal RC-network,

and B is the matrix of thermal conductance. While $P(t)$ is time dependent (guided by the power profile), $L(T)$ is dependent on the temperatures of the RC-network nodes. It may be noted that $P(t)$ and $L(T)$ have nonzero values only for those RC-network nodes which correspond to die-segments of the chip (with the exception of the node modeling ambient surroundings, for which $P(t)$ has a nonzero value). Nodes which do not correspond to chip die-segments, for example, the nodes corresponding to the spreader, do not dissipate any power.

During thermal analysis, the temperature-dependent leakage power dissipation at different thermal RC-network nodes ($L(T)$) are updated before invoking any discretized-continuous step. Hence $L(T)$ values are updated in every time-slot of size Δ (20,000 cycles for the experiments presented in this work). Leakage power dissipation in each chip die segment (corresponding to a thermal RC-network node) is dependent on the temperature of the die segment. Each element in the vector $L(T)$ denotes the leakage power dissipated in a given die segment d at a certain die-segment temperature $temp_d \in T$. These leakage power values are obtained by querying a set of look-up tables, one for each die segment (corresponding to a thermal RC-network node dissipating leakage power), which store values of leakage power dissipated in that die segment at certain predesignated temperatures. These stored leakage power values are obtained by analyzing the circuits corresponding to the chip die segments in question. This analysis is performed apriori either by circuit synthesis tools[5], or by power-performance analysis setups like PAnalyzer (http://www.eecs.umich.edu/~panalyzer/). Each element in the vector $L(T)$ is obtained by linear extrapolation between stored look-up table values [Liu et al. 2007]. For example, Figure 14 illustrates leakage power dissipated in the "Rest" die-segment (Figure 6) at temperatures 313 K to 383 K in steps of 10 K. Linear extrapolation is performed between the stored values to obtain leakage power at any given temperature of the die segment.

It is pertinent to note node temperatures continue to be overapproximated in an *abstract domain* (Section 3.3) when extending the analysis to account for leakage power in the manner presented in this subsection. This is because leakage power monotonically increases with node temperature, and node temperatures monotonically increase with power dissipated. Hence, when a node temperature is overapproximated while exploring an *abstract-state*, the power dissipation of any run from this state is also overapproximated due to overapproximation of leakage power. This in turn further overapproximates the node temperature, thereby ensuring that the abstraction always leads to overapproximation of node temperatures.

4. EXPERIMENTS AND RESULTS

In this section we present experimental results for the prototype HeatCheck tool and use the tool for thermal analysis of different static schedules of three applications from the MiBench suite [Guthaus et al. 2001]. We first provide

[5]http://synopsys.com/Tools/Implementation/PhysicalImplementation/Pages/ICCompiler.aspx

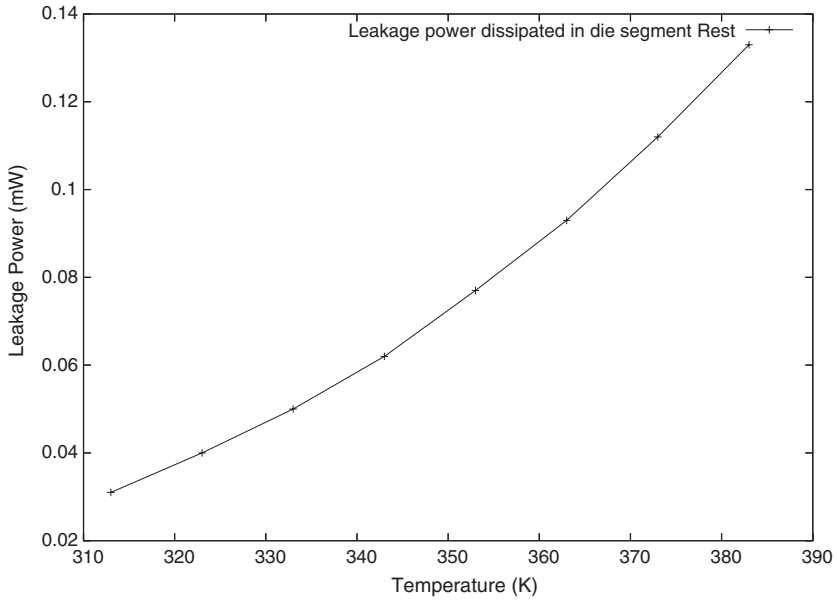

Fig. 14. Variation of leakage power in any one of the "Rest" die segment (Figure 6) with change in temperature. Leakage power at temperatures 313 K to 383 K in steps of 10 K are shown. Linear extrapolation is performed between these points. This plot denotes leakage power for 180 nm technology, obtained using the Simplescalar PAnalyzer tool (http://www.eecs.umich.edu/~panalyzer/).

details of the experimental setup used in this work and then provide experimental results and analysis.

4.1 Experimental Setup

The experimental setup used in this work consists of two components. The first component is a multicore architecture on which multithreaded applications are executed, while the second component is a suite of code-block-level abstractions of the multithreaded applications themselves. All experiments were performed on a 2 GHz Pentium machine with 1GB RAM and virtual memory limited to 1GB.

4.1.1 *Multicore Platform Used for Experiments.* The multicore platform used in this work consists of 4 Alpha EV6 cores with a common 4 MB L2 cache. The floorplan for the chip used in this work is presented in Figure 15. The floorplan of each Alpha EV6 core is obtained after rearranging architectural units in the floorplan used in Skadron et al. [2004]. The manufacturing technology of the chip is 180nm and accordingly the size of each core is 8.63×8.63 mm. For a thermal analysis the ambient temperature of the surroundings is assumed to be 40°C (313 K) [Skadron et al. 2004]. The chip used in the experiments is 3-layered consisting of the silicon die, a heat spreader, and a heat sink [Skadron et al. 2004]. The parameters (thermal conductance and thermal capacitance matrices) of the Compact Thermal Model (CTM) are obtained by

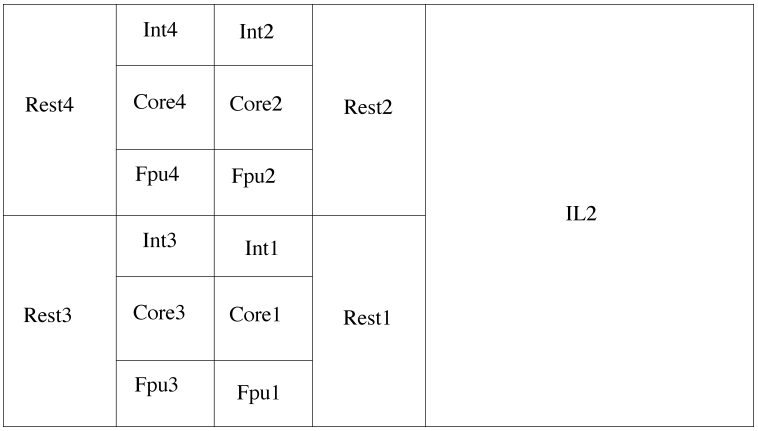

Fig. 15. The die floorplan used in the experiments. It consists of 4 Alpha EV6 cores and a common L2 cache. Each core is partitioned into 4 die segments, where each die segment corresponds to a set of architectural units.

using HotSpot 3.1 [Skadron et al. 2004]. The CTM for the floorplan has 61 nodes corresponding to 62 continuous variables (61 node temperatures and time) in the thermal automata.

4.1.2 *Code-Block-Level Application Model.* The applications used in the experiments are selected from the MiBench benchmark suite [Guthaus et al. 2001]. Code blocks which take more than 20,000 cycles are marked manually and the data and control-flow dependencies between the code blocks are ascertained. The code blocks are then mapped and statically scheduled on different processors of the multicore architecture.

The power dissipation of each code block is measured by simulation on the Simplescalar PAnalyzer (http://www.eecs.umich.edu/~panalyzer/) for several execution runs in order to get sample traces for learning power dissipation. The upper and lower bounds on time of execution of code-blocks are set to the greatest and least time delay (respectively) observed during these power-performance simulation runs. We perform experiments for a 180nm circuit operating at 1.5 V with a clock frequency of 3 GHz, an clock skew of 16 ps, an L1 cache miss latency of 6 cycles, an L2 cache miss latency of 12 cycles, 64 KB two-way LRU DL1 and IL1 caches and 4 MB eight-way LRU L2 cache.

4.2 Experimental Results

In this subsection we present experimental results for thermal analysis of a set of multicore schedules of three applications from the MiBench suite, namely *susan, basicmath, and FFT*. We present a short description of the applications and the schedules analyzed prior to describing experimental results for thermal analysis by HeatCheck. We present experimental results for the thermal automata construction step, a time-bounded analysis of the thermal automata, analysis of the thermal automata for long time durations, and a study of increasing the size of time-step on simulation speed and accuracy.

For the thermal automata construction step, experimental results comprise of the size of the model after splitting action nodes, the size of the thermal automata constructed, and time taken for constructing the same. Analysis of the multicore schedules with temperature bounds set to 70, 80, 90, and $100^{\circ}C$, and time bound set to 300 million cycles (0.1 s real time) are presented and discussed. We present two sets of experimental results for this time-bounded thermal analysis: one with power dissipation values learned using the method in Section 3.1, and the other with power dissipation set to the average of different power profiles from power-simulation runs. This helps us in examining the efficacy of the machine learning approach proposed in this work. Additionally, experimental results using a hybrid automata analysis tool called PHAVer [Frehse 2005] are presented in order to highlight the challenges in analyzing hybrid systems with a large number of continuous variables. These experiments with the PHAVer tool do not incorporate the effects of leakage power dissipation as formulated in Section 3.6.

We then present experimental results for the method which identifies "hot"-execution cycles and simulates it for a long time durations of 1 s real time, or 3 billion cycles (Section 3.5). We compare the peak chip temperatures obtained by simulating a random execution path, and the selected "hot"-execution cycle, over the simulation window of 3 billion cycles. We also study the effects of modifying the size of the time-step on the speed and accuracy of thermal simulation, and the size of the thermal automata. For this we modify the size of the time-step from 20,000 cycles, to 40,000, 60,000, 80,000, and 100,000 cycles, and present results for the size of the thermal automata, the time taken for a thermal simulation for 1 s (3 billion cycles) by this simulation. Finally we summarize the experimental results and discuss the best possible ways for performing thermal analysis for different types of multicore applications.

4.2.1 *Susan.* Susan is a corner and edge detection (in images) software from the MiBench automotive suite. The edge detection operation consists of first initializing the look-up tables (*set_lut_e*) and memory (*init_mem*), then finding edges using a 3×3 (*edges*3*x*3) or 5×7 (*edges*5*x*7) mask depending on the input. Once the edges are found, they are drawn on the image (*edge_draw*) and written (*display_e*) to a file. The corner detection operation consists of first initializing the look-up tables (*set_lut_c*) and memory *init_mem*, then finding corners using a *quick* corner detection (*q_corners*) algorithm. Once the corners are found, they are drawn on the image (*corner_draw*) which is written (*display_c*) to a file.

The lower and upper bounds on latency of the tasks executing on an Alpha EV6 core (obtained by simulation on the Simplescalar simulator [Austin et al. 2002]) are shown in Figure 16 along with the different action nodes. A total of 18 simulation runs were conducted in order to obtain the bounds on time of execution and sample power dissipation profiles. The machine learning and power estimation step took 41 minutes 41.22s to complete, with the largest PFA consisting of 2164 nodes and 99449 transitions.

The schedule for the susan application analyzed in this experiment is shown in Figure 16. It consists of the susan corners (mapped to *Proc*3 and *Proc*4)

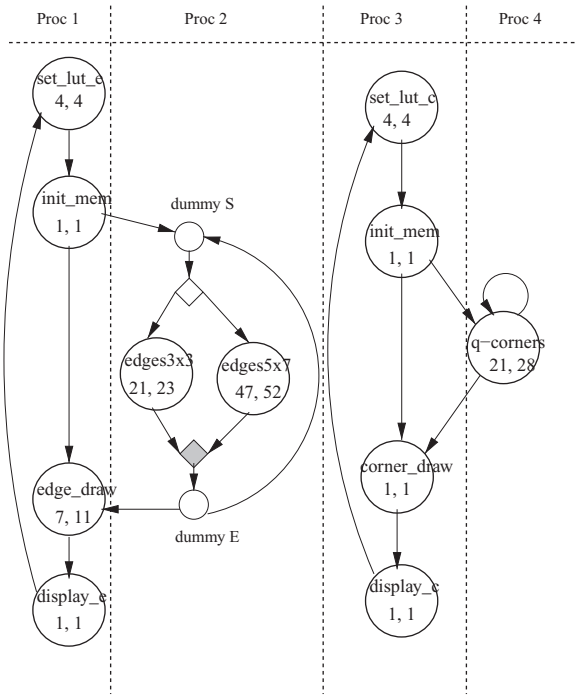

Fig. 16. The susan corners and susan edges operations mapped to the different processors. The time duration of execution of code-blocks is given in $\times 20000$ cycles.

and susan edges (mapped to *Proc*1 and *Proc*2) operations executing concurrently and continuously. The continuity of control flow is indicated by an edge from the *final* to the *start* node in each processor. *Proc2* nondeterministically executes one of *edges*3*x*3 or *edges*5*x*7, after *init_mem* completes execution in *Proc*1. This nondeterministic execution is modeled by a *decision-merge* construct in *Proc*2. Since *decision* and *merge* nodes cannot be start or finish nodes, additional dummy nodes *dummyS* and *dummyE* are added as start and finish nodes, respectively. Moreover the task *edge_draw* in *Proc*1 can execute only when one of *edges*3*x*3 or *edges*5*x*7 completes execution.

Experimental results for thermal analysis are presented in Table I. Thermal automata construction takes a small amount of time (17.56 s), as does the analysis where there is a violation of the temperature limit (upto 131 s). This is the case since the temperature violations occur in the first execution run analyzed. However, when all paths must be analyzed to prove the absence of a temperature limit violation (for $100^{\circ}C$), then the analysis times out. This is due to two reasons, namely the large size of the thermal automata (with 14164 nodes), and the large number of outgoing discrete transitions from locations (up to 168 transitions). It is notable that all discrete transitions (when the time-step is 20,000 cycles) are analyzed for the results presented in Table I. Additionally PHAVer runs out of memory ($> 1\,GB$) for each experiment without being able to perform even one iteration of state-space exploration.

Table I. Experimental Results for Analysis of the Susan Testcase

Schedule	Nodes Model	locs TA	Time Build EA	Analysis Tool	Results (Time, Outcome, States Searched) 70°C	80°C	90°C	100°C
susan learned	132	14164	17.56 s	*HeatCheck*	50.98 *s* 97.76 M 3215	80.88 *s* 154.20 M 5073	130.48 *s* 224.92 M 7399	*t.o.* *N.V.* 80000
				PHAVer	*mem*	*mem*	*mem*	*mem*
susan average	132	2043	5.72 s	*HeatCheck*	15.97 *s* 64.66 *M* 1775	26.12 *s* 102.36 *M* 2811	41.16 *s* 145.64 *M* 4004	66.93 *s* 194.18 *M* 5331
				PHAVer	*mem*	*mem*	*mem*	*mem*

(t.o.: Time taken > 1*hrs*, mem: Memory bound of 1 GB exceeded, EA: Execution Automata, TA: Thermal Automata). Results indicate the time taken for analysis (*time*), the outcome (*outcome*) of the analysis; which is either N.V. (No Violation) or the number of cycles (in millions) taken for the temperature bound to be violated; and the number of states searched (*states searched*).

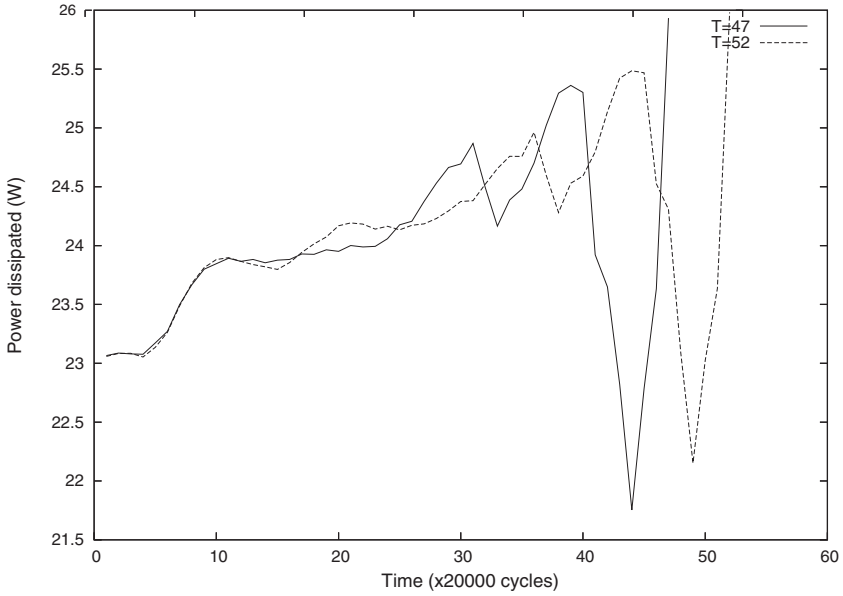

Fig. 17. Two learned power dissipation profiles (completing in 47 and 52 time units, respectively) of the *edges*5*x*7 code-block in die segment *Int*.

Another notable observation about this schedule is the significant difference (up to 40%) in the analysis performed with learned power profiles and average power dissipations. This is due to the significant variation in power dissipation of code-blocks of the *susan* testcase (Figure 17) which cannot be captured by an average power dissipation model.

We study the efficacy of selecting one out of every 1, 5, 10, and 15 transitions, for an exhaustive analysis when the temperature limit is not breached. Experimental results for the same are presented in Table II. The time bound for these experiments is 5 million cycles, while the temperature limit is set to 320 K (less than one Kelvin above the peak temperature). Table II also presents results for analysis when states are stored and compared only at thermal

Table II. Experimental Results for Analysis of the *Susan* Testcase When 1-of-k Discrete Transitions are Analyzed at Each Location

State Storage Strategy		1 of 1	1 of 5	1 of 10	1 of 15
all locations	*Time*	*t.o.*	*t.o.*	253.74 s	38.39 s
	States	114500	257500	17782	2754
	Counterexamples	12374	2226	575	38
loop-back locations	*Time*	*t.o.*	*t.o.*	152.86 s	43.51 s
	States	95500	54500	8615	2454
	Counterexamples	0	0	0	0

(t.o.: Time taken $>$ 1hrs), and when states are stored at locations with an incoming back-edge transition. Results are presented for the time taken for exhaustive analysis of the (for time-bound of 5 million cycles, and temperature limit of 320 K) thermal automata, the number of states explored, and the number of spurious counterexamples discharged.

automata locations which have incoming back-edge transitions. There are 111 such "loop-back"-locations in the thermal automata. We present the time taken for analysis (and the number of thermal abstract-states explored), and the number of counterexamples discharged during analysis.

Clearly, selecting *1-of-k* transitions helps in the exhaustive analysis of the thermal automata at an increased analysis granularity. On the other hand, storing states at "loop-back"-locations reduces the number of states stored, and often reduces time taken for analysis. Additionally, due to less overapproximation on account of fewer locations where states are stored, the number of spurious counterexamples discharged is reduced considerably.

Experimental results presented until now pertain to time-bounded analysis of the thermal automata. This analysis can be performed from ambient temperatures (as in the results presented), or may be performed from temperatures obtained after the thermal automata has been simulated for a long time duration (say 3 billion cycles). Additionally, directed simulations may be performed along execution cycles which are identified by an exhaustive thermal analysis for a short time duration. Figure 18 shows the difference in peak chip-temperature (in die-segment *Int2*, Figure 15) with time, for a random simulation run and a directed simulation. The results show a significant (up to 20°C) difference in peak chip temperature between random and directed simulations. This difference is attributed to the large variation in the active to idle ratio in *Proc*2 (from 45% idle time to 20% idle time), between the best- and worst-case execution runs. A random simulation is expected to correspond to a run of about 33% idle time. The "hot"-execution cycle is detected by performing 7 exhaustive thermal analysis steps of 2.0 million cycles each, by analyzing *1-of-5* transitions. Each of the aforesaid thermal analysis steps started from the thermal state which had the highest node temperatures in the previous analysis step. The complete analysis took 683 s. It is notable that after the elapse of 300 million cycles, the peak temperature along the "hot"-execution cycle is 392.79 K.

In this work we study the effect of varying the size of the time-step on the time taken for simulation, and the changes in the underlying discrete structure of the thermal automata. We vary the size of the time-step from 20,000 cycles to 100,000 cycles, in steps of 20,000 cycles. We present experimental results for the size of the thermal automata, the peak chip temperature, and time taken

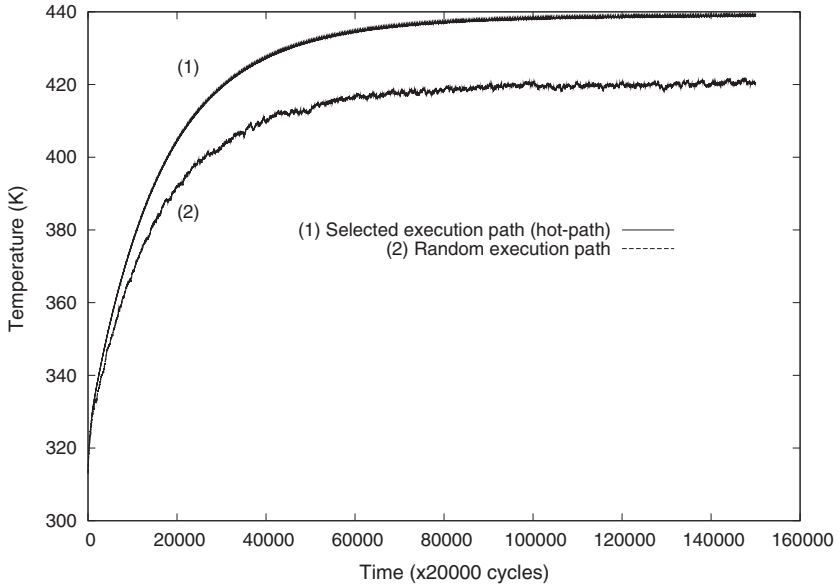

Fig. 18. The variation of peak chip temperature of the *susan* testcase (in die segment *Int2* in Figure 15) with time, for a random simulation and directed simulation of an identified "hot"-cycle, for long runs of 1 s (3 billion cycles).

Table III. Experimental Results for Analysis of the Susan Testcase with Variation in Size of the Time-Step from 20,000 cycles to 100,000 cycles

	20,000	40,000	60,000	80,000	100,000
Locations of TA	14164	3025	1012	631	331
Peak Chip-temperature	404.38 K	399.90 K	393.61 K	387.80 K	378.17 K
Simulation time	163.12 s	111.23 s	524.89 s	817.46 s	682.11 s

(TA: Thermal Automata). Results for size of the thermal automata, the peak chiptemperature after simulation for 3 billion cycles, and time taken for simulation of 3 billion cycles are presented.

for simulation of the thermal automata. This simulation is performed for a time bound of 3 billion cycles, and the first transition (from the list of transitions) at each location is always selected. Clearly, with increase in size of the time-step, the number of possible distinguishable behaviors of a task decreases, thereby reducing the amount of nondeterminism in the model. This is reflected in the decreasing size of the thermal automata, with an increase in the size of the time-step (Table III). At the same time, it may be noted that changing the size of the time-step causes significant variations in the execution behavior of the task-level model. For example, if the size of the time-step is 100,000 cycles, then a task executing for 120,000 cycles (6 time-units of 20,000 cycles each) is now considered to be executing for 200,000 cycles (2 time-units of 100,000 cycles each). This causes significant deviation in the chip-thermal behavior of the application in question, leading to loss of accuracy. For example, the variation in peak chip temperature is upto 26$^{\circ}C$ when the time-step is varied from 20,000 cycles to 100,000 cycles (Table III). However, the experimental results indicate that there is no guarantee of continuous performance gains, in terms of higher

Table IV. Experimental Results for Analysis of the Basicmath Testcase

Schedule	Nodes Model	locs TA	Time Build EA	Analysis Tool	Results (Time, Outcome, States Explored) 70°C	80°C	90°C	100°C
basicmath learned	20	18	9*ms*	*HeatCheck*	5.27 *s* 61.86 *M* 41	8.24 *s* 99.48 *M* 53	11.83 *s* 140.62 *M* 78	15.13 *s* 186.58 *M* 89
				PHAVer	*mem*	*mem*	*mem*	*mem*
basicmath average	20	18	9*ms*	*HeatCheck*	5.42 *s* 62.40 *M* 41	8.40 *s* 100.08 *M* 53	12.04 *s* 141.40 *M* 78	15.36 *s* 187.48 *M* 89
				PHAVer	*mem*	*mem*	*mem*	*mem*

(mem: Memory bound of 1 GB exceeded, EA: Execution Automata, TA: Thermal Automata). Results indicate the time taken for analysis (*time*), the outcome (*outcome*) of the analysis; which is the number of cycles (in millions) taken for the temperature bound to be violated; and the number of states searched (*states searched*).

simulation speed, when the time-step is increased. The simulation speed is the most when the time-step is 40,000 cycles. This unexpected behavior is attributed to the fact that an adaptive solver is utilized in our experimental setup. This solver takes several substeps within each time-step, which are guided by the shape of the temperature and power-dissipation trajectories. The time taken for simulation is decided by the number of such solver substeps taken, which in turn is largely dependent on the shape of the trajectory and not only the size of each time-step.

4.2.2 *Basicmath.* Basicmath is an application from the MiBench automotive suite which performs several mathematical operations pertinent to automotive controllers. In the code partitioning we perform, the application consists of four code-blocks. These are *cubic_eqn* for solving some cubic equations, *random_eqn* for solving a set of random equations, *integer_sqrt* for finding some square roots, and *angle_conv* to perform some radians to degrees angle conversion. The time taken by these code-blocks (x20,000 cycles) to execute are provided in Figure 19. The profiling step for generating sample power dissipation traces and obtaining execution time bounds consisted of 4 runs of the application. The learning and estimation step completed in 23 minutes 44.26*s*, with the largest PFA learned consisting of 818 states and 13073 transitions.

We analyze a schedule of the *basicmath* application where tasks execute sequentially (Figure 19). Sequential execution is enforced by edges between tasks, for example, the edge {*random_eqn*, *integer_sqrt*} ensures that *integer_sqrt* starts executing only after *random_eqn* completes execution in one iteration. Moreover the edge {*angle_conv*, *cubic_eqn*} ensures that *cubic_eqn* can start the next iteration run only after *angle_conv* completes execution in the previous iteration.

Similar to the experimental results for *susan*, the execution automata construction and thermal analysis takes a small amount of time for the *basicmath* testcase. As before, PHAVer [Frehse 2005] runs out of memory without performing a single iteration. The notable differences with *susan* are the smaller size of the execution automata, due to the sequential nature of the schedule, the

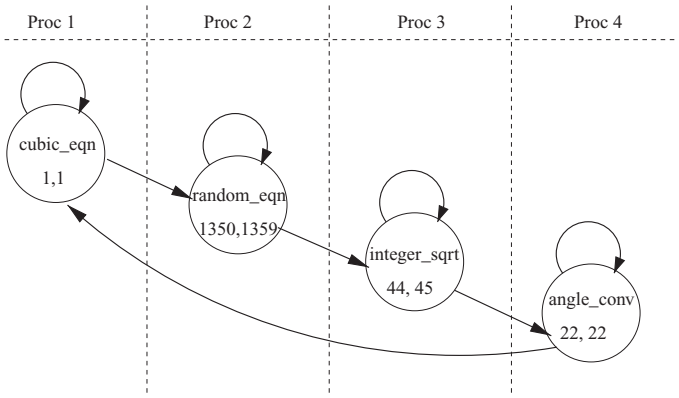

Fig. 19. The basicmath application with all four operations executing sequentially in. The time duration of execution of action nodes is in x20000 cycles.

smaller state space explored, and the reduced difference between the analysis using average power dissipation and learned power values. The reduced number of thermal states explored is due to the fewer discrete transitions which must be triggered to reach an error state. This in turn is due to the large time of execution of the *random_eqn* action node (27.00 to 27.18 million cycles) and the sequential nature of the schedule which allows upto 27.18 million cycles to elapse before the occurrence of a discrete transition. Similarity between analysis with average and learned power profiles is due to the small variation in time of execution of tasks ($< 2\%$ in most cases) and power dissipation profiles of individual task runs. Figure 20 shows one learned power dissipation profile of the *rand_eqn* code-block in die segment *Int* (lasting 27.0M cycles), which is almost constant for most time duration.

Table V presents experimental results for analysis of the *basicmath* testcase, when one out of every 1, 2, 3, and 5 transitions are selected, and when thermal states are stored only at "loop-back"-locations. The temperature limit for the analysis is 424 K (for which there is no violation of the temperature bound), and the time bound is 1 billion cycles. Like in the case of the *susan* testcase, the number of spurious counterexamples and time taken for analysis reduces, when states are stored only at "loop-back"-locations (3 in number). Moreover, the time taken for exhaustive analysis improves when a subset of discrete transitions (1-of-k) are analyzed. It is notable that no location of the thermal automata of the *basicmath* testcase has more than 3 outgoing discrete transitions. Hence the time taken for analysis, and number of states explored when 1-of-3 and 1-of-5 transitions are selected, are similar.

We present experimental results comparing directed and random simulations for a long time duration (1 s real time, 3 billion cycles), in Figure 21. There is no noticeable difference between the random simulation run and the directed simulation run. This is due to the low degree of nondeterminism in the *basicmath* testcase model (Figure 19), due to low variation in execution delay of tasks ($<1\%$ of sum of lower bounds of execution delays).

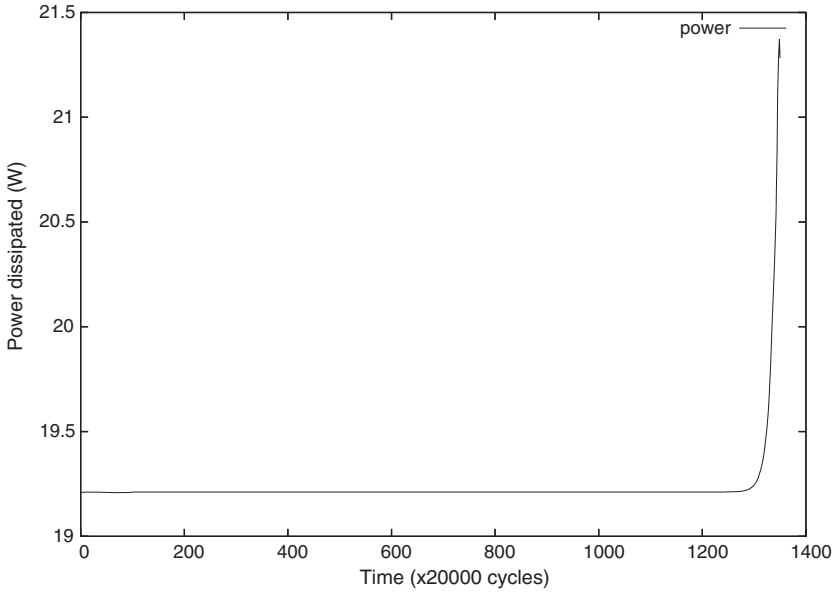

Fig. 20. A learned power dissipation profile of the *random_eqn* code-block in die segment *Int*.

Table V. Experimental Results for Analysis of the *Basicmath* Testcase when 1-of-*k* Discrete Transitions are Analyzed at Each Location

State Storage Strategy		1 *of* 1	1 *of* 2	1 of 3	1 *of* 5
all locations	*Time*	*t.o.*	143.47 *s*	79.18 *s*	78.17 *s*
	States	24000	762	456	456
	Counterexamples	1707	5	4	4
loop-back locations	*Time*	*t.o.*	109.68 *s*	71.00 *s*	71.12 *s*
	States	11000	668	455	455
	Counterexamples	310	4	3	3

(t.o.: Time taken $> 1hrs$), and when states are stored at locations with an incoming back-edge transition. Results are presented for the time taken for exhaustive analysis of the (for time bound of 1 billion cycles, temperature limit of 424 K) thermal state space, the number of states explored, and the number of spurious counterexamples discharged.

A comparison of performance and accuracy of analysis when the time-step is modified is presented in Table VI. Unlike the *susan* testcase analyzed earlier (Table III), the *basicmath* testcase shows a general trend of improved simulation time, with a negligible loss in accuracy. This is firstly due to the task graph with lower granularity (large time-step) being similar to the one with higher granularity (smaller time-step). We also believe that due to the almost constant power dissipation of tasks (notably the *random_eqn* task, Figure 20), node temperatures rise steadily and monotonically, without many inflections. In this case, larger time-steps can be taken without causing much loss in accuracy, and restricting the increase in the number of substeps the solver takes in each time-step, so as to attain an overall gain in performance.

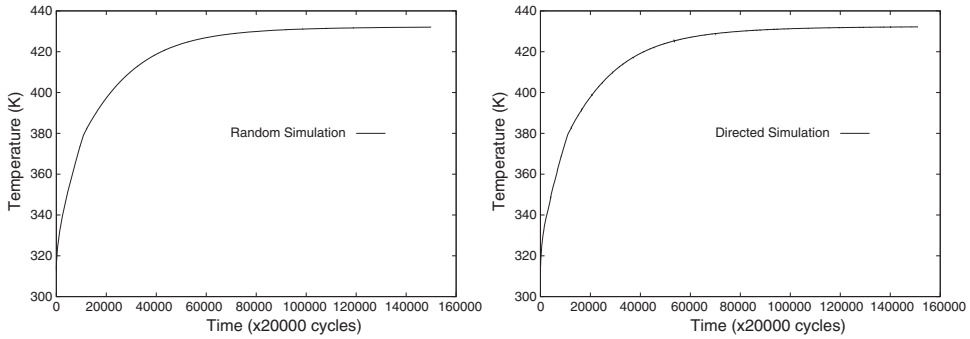

Fig. 21. The variation of peak chip temperature of the *basicmath* testcase (in die segment *Int2* in Figure 15) with time, for a random simulation (left) and directed simulation (right), for long runs of 1 s (3 billion cycles).

Table VI. Experimental Results for Analysis of the Basicmath Testcase with Variation in Size of the Time-Step from 20,000 Cycles to 100,000 Cycles

	20,000	40,000	60,000	80,000	100,000
Locations of TA	18	11	10	10	9
Peak chip-temperature	432.04 K	432.04 K	432.04 K	432.02 K	432.04 K
Simulation time	92.93 s	93.28 s	80.48 s	72.97 s	59.35 s

(TA: Thermal Automata). Results for size of the thermal automata, the peak chip temperature after simulation for 3 billion cycles, and time taken for simulation of 3 billion cycles are presented.

4.2.3 *FFT*. The FFT (Fast Fourier Transform) application, from the MiBench Telecommunications suite, consists of FFT and inverse FFT operations. The FFT operation consists of four tasks mapped to action nodes *init*, *fix_test*, *fft*, and *display* while the inverse FFT operation consists of the tasks *i_init*, *i_fix_test*, *i_fft*, and *i_display*. Here *init*, and *i_init* allocate memory for the operation, *fix_test*, and *i_fix_test* generate a random testcase of user-defined size, *fft*, and *i_fft* perform the FFT and inverse FFT operations, respectively, while *display* and *i_display* print the results on the standard output. The time of execution of these tasks are shown in Figure 22.

In order to obtain the power dissipation profile for different tasks, 4 random testcases (waves) of lengths 4096 and 8192 time units were analyzed for FFT and inverse FFT operations, respectively. The largest PFA learned was that modeling the power dissipation of *i_display* in the "Int" die segment. It consisted of 321 states and 4800 transitions. The process of learning the PFAs and estimating the power dissipation of tasks was completed in 2 minutes 37.77 seconds.

We perform three experiments with the same schedule but different mappings of the FFT application in order to study the effects of task mapping on thermal behavior. For this, we consider three schedules where tasks, corresponding to the FFT and the inverse FFT operations, are mapped to adjacent processors *Proc*3 and *Proc*1 (*fft*_3_1), *Proc*4 and *Proc*2 (*fft*_4_2), and diagonally opposite processors *Proc*4 and *Proc*1 (*fft_dis*).

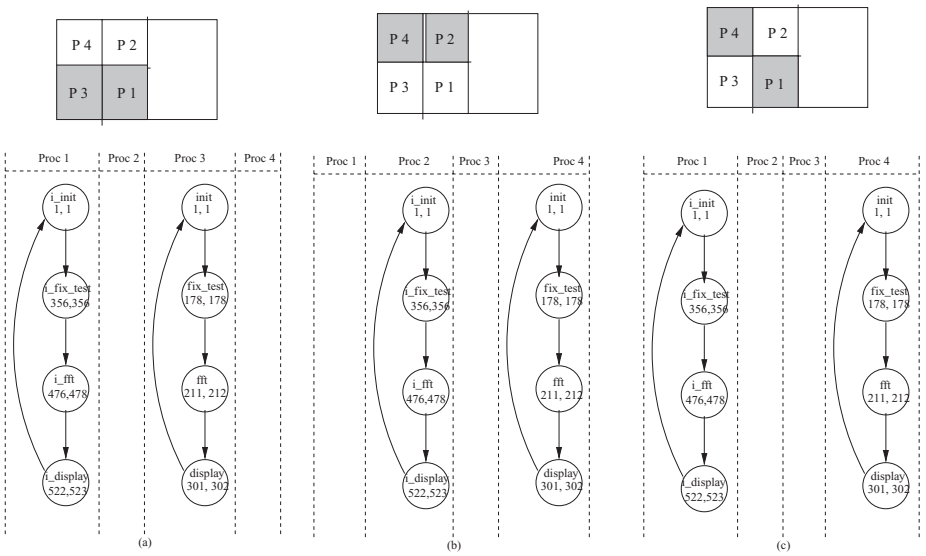

Fig. 22. The FFT (Fast Fourier Transform) operation and the inverse FFT operation executing in adjacent processors Proc3 and Proc1 (a), adjacent processors Proc4 and Proc2 (b), and diagonally opposite processors Proc4 and Proc1 (c). (time of execution in x20000 cycles).

Experimental results (Table VII) for the three schedules of *FFT* follow the trend of *susan* and *basicmath*. While execution automata construction, and analysis in the presence of temperature-bound violation takes a small amount of time, the analysis times out when the temperature limit is not breached ($100^{\circ}C$). Moreover, the difference between analysis with average power dissipation and learned power dissipation is low ($<1\%$) due to little variation in power dissipation of code-blocks over time of execution.

The three mappings of the FFT application analyzed in this experiment show significant variation in thermal behavior. In agreement with conventional wisdom about heating effects due to spacial locality, the mapping *fft*_4_2 (Figure 22(b)) with tasks mapped to adjacent cores heats faster than *fft_dis* which has tasks mapped to diagonally opposite cores (Figure 22(c)). However, *fft*_3_1 which also has mapping to adjacent cores does not heat faster than *fft_dis* in accordance with the spacial locality argument. For understanding this anomaly we first note that power dissipation in the *Int* (Figure 6) die segment of the core is 10–20 times more than any other core die segment. Hence the location of the *Int* die segments of cores dissipating power has greater significance in dictating peak temperature, than the task mapping to cores. In the mapping *fft*_3_1, the Int die segments ($Int1$ and $Int3$) have relatively "hot" silicon of active cores to the bottom (Figure 23(a)), "cool" silicon of the unutilized cores to the top (Figure 23(a)) and the heat spreader further up (Figure 23(a)). Clearly the "cool" silicon die region facilitates in bringing down the temperature of the *Int* die segments, while the relatively "hot" active region does not contribute much to this cause. Hence the amount of "cool" silicon, whether die or spreader, in close vicinity of the Int die segments has a profound effect on the thermal

Table VII. Experimental Results for Analysis of the (adjacent mapping) fft_3_1, fft_4_2 and (diagonal mapping) fft_dis

Schedule	Nodes Model	locs TA	Time build EA	Analysis Tool	Results (Time, Outcome, States Explored) 70°C	80°C	90°C	100°C
fft_3_1 learned	12	8888	1607 *ms*	*HeatCheck*	26.93 *s* 85.68 *M* 77	32.65 *s* 141.56 *M* 125	40.26 *s* 216.36 *M* 190	*t.o.* *N.V.* 24000
				PHAVer	*mem*	*mem*	*mem*	*mem*
fft_3_1 average	12	8888	1607 *ms*	*HeatCheck*	24.68 *s* 85.74 *M* 77	30.59 *s* 140.84 *M* 126	38.92 *s* 216.48 *M* 190	*t.o.* *N.V.* 27000
				PHAVer	*mem*	*mem*	*mem*	*mem*
fft_4_2 learned	12	8888	1607 *ms*	*HeatCheck*	26.44 *s* 79.06 *M* 69	30.79 *s* 123.50 *M* 107	36.58 *s* 178.36 *M* 155	43.61 *s* 246.36 *M* 218
				PHAVer	*mem*	*mem*	*mem*	*mem*
fft_4_2 average	12	8888	1607 *ms*	*HeatCheck*	24.13 *s* 79.16 *M* 69	29.10 *s* 123.50 *M* 107	33.86 *s* 178.26 *M* 155	41.87 *s* 246.32 *M* 218
				PHAVer	*mem*	*mem*	*mem*	*mem*
fft_dis learned	12	8888	1607 *ms*	*HeatCheck*	26.21 *s* 81.90 *M* 73	31.47 *s* 136.04 *M* 121	38.08 *s* 192.50 *M* 169	45.91 *s* 274.54 *M* 242
				PHAVer	*mem*	*mem*	*mem*	*mem*
fft_dis average	12	8888	1607 *ms*	*HeatCheck*	25.23 *s* 81.86 *M* 73	29.69 *s* 135.88 *M* 121	35.34 *s* 192.50 *M* 169	43.86 *s* 274.52 *M* 241
				PHAVer	*mem*	*mem*	*mem*	*mem*

(t.o.: Time taken $> 1\,hrs$, mem: Memory bound of 1 GB exceeded, EA: Execution Automata, TA: Thermal Automata). Results indicate the time taken for analysis (*time*), the outcome (*outcome*) of the analysis; which is either N.V. (No Violation) or the number of cycles (in millions) taken for the temperature bound to be violated; and the number of states searched (*states searched*).

behavior of the system. In the mapping *fft*_4_2, only the heat spreader is available and there is no "cool" silicon die region available in the vicinity of the *Int* die segments (Figure 23(b)). Hence *fft*_4_2 heats more quickly than *fft*_3_1. The third mapping *fft_dis* has one *Int* die segment in the vicinity of "cool" silicon die region (Figure 23(c)), while the other is adjacent to the heat spreader. This mapping heats slowly compared to *fft*_4_2, but by virtue of power dissipation in die segment *Int4* (with "cool" silicon die region only on one side) heats faster than *fft*_3_1.

Results of experiments when a subset of transitions are analyzed, and thermal states stored at a subset of locations (which have incoming back-edge transitions), are presented in Table VIII. The general trend of reduced analysis times when a small subset of transitions are analyzed is imminent in the results. Moreover, experimental results with *1-of-3* and *1-of-5* transitions selected are similar, since the maximum number of transitions from any location, amongst all locations visited during both these time-bounded analyses, is 3 (even though there are locations with up to 5 outgoing transitions). The time taken for analysis when states are stored at "loop-back"-locations (7 in number) is usually smaller than when states are stored at each location, with the exception of the

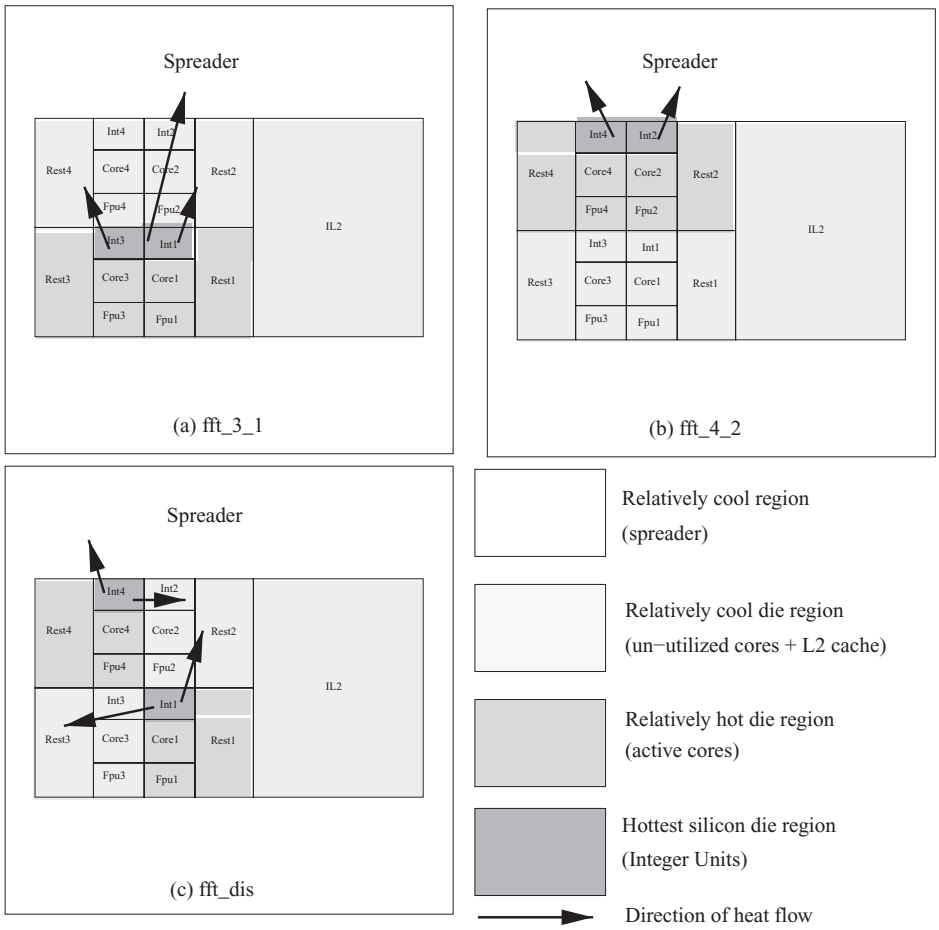

Fig. 23. Active ("hot") and unutilized ("cool") silicon die regions for mappings: (a) fft_3_1; (b) fft_4_2; and (c) fft_dis. Arrows roughly indicate most likely flow of heat in different configurations.

case when all transitions are analyzed (*1-of-1*). We note that there is an advantage in storing states at all locations, since this allows us to match more states and hence reduce the number of analyzed execution sequences. For the *fft_3_1* testcase, some execution sequences which are parts of long cycles are not explored when states are stored and matched at each location. In case of state storage (and matching) at "loop-back"-locations, the exploration of these execution sequences halts only after exploring all locations in the long cycle (and its branches).

Due to absence of decision branches and less variation in execution delays of tasks, as well as contiguous execution of tasks in individual cores, we do not observe significant variation among different execution behaviors of the *fft_3_1* testcase. Hence, like the *basicmath* testcase, the difference between a directed simulation of the thermal automata and a random simulation is low, as shown in Figure 24.

Table VIII. Experimental Results for Analysis of the *fft_3_1* Testcase When 1-of-k Discrete Transitions are Analyzed at Each Location

State Storage Strategy		1 of 1	1 of 2	1 of 3	1 of 5
all locations	*Time*	265.01 *s*	27.01 *s*	23.26 *s*	23.17 *s*
	States	3284	77	42	42
	Counterexamples	100	2	1	1
loop-back locations	*Time*	*t.o.*	23.37 s	20.90 *s*	20.88 *s*
	States	20500	48	24	24
	Counterexamples	0	0	0	0

(t.o.: Time taken $> 1hrs$), and when states are stored at locations with an incoming back-edge transition. Results are presented for the time taken for exhaustive analysis of the (for time bound of 50 million cycles, temperature limit of 335 K) thermal state space, the number of states explored, and the number of spurious counterexamples discharged.

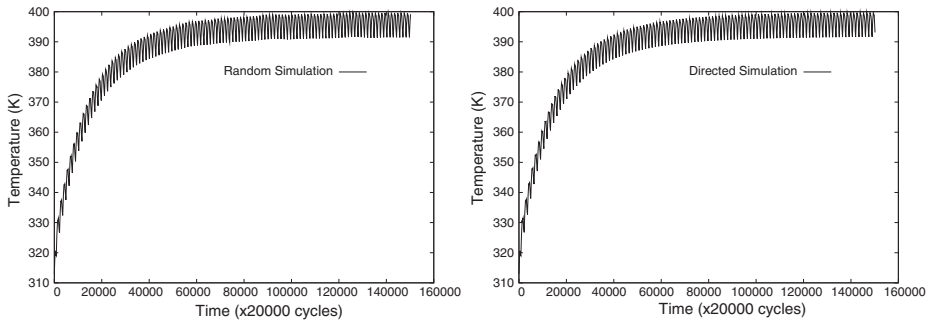

Fig. 24. The variation of peak chip temperature of the *fft_3_1* testcase (in die segment *Int3* in Figure 15) with time, for a random simulation (left) and directed (right) simulation of an identified "hot"-cycle, for long runs of 1 s (3 billion cycles).

Experimental results for the study of efficacy of increasing the size of time-step from 20,000 cycles to 100,000 cycles are shown in Table IX. We observe an almost 30% reduction in simulation time when the size of the time-step is increased form 20,000 to 40,000 cycles. However, the simulation time increases with subsequent increase of the time-step size. The accuracy of analysis is not significantly deteriorated (about 4 K), since large delay values of individual tasks and low variations in execution delays causes the task graph with lower granularity (larger time-step) to have few variations from the task-graphs with higher granularity.

4.3 Discussion on Experimental Results

In this subsection we discuss the trends observed in the experimental results presented earlier in Section 4.2. In particular we evaluate the different engineering enhancements proposed in this work, and establish some correlation between the cost of thermal analysis, the underlying discrete structure of the application task graph, and power dissipation profiles of tasks. We expect exhaustive thermal analysis to be performed only for short time durations ($<$ 50 million cycles), and the results to be used either to identify "hot"-cycles, or compare two or more schedules.

Table IX. Experimental Results for Analysis of the *fft_3_1* Testcase with Variation in Size of the Time-Step from 20,000 Cycles to 100,000 Cycles

	20,000	40,000	60,000	80,000	100,000
Locations of TA	8888	4096	2711	2031	1623
Peak chip-temperature	401.13 K	396.88 K	397.02 K	401.49 K	397.16 K
Simulation time	144.65 s	93.24 s	150.01 s	455.65 s	503.83 s

(TA: Thermal Automata). Results for size of the thermal automata, the peak chip temperature after simulation for 3 billion cycles, and time taken for simulation for 3 billion cycles are presented.

The thermal automata of benchmark testcases analyzed in this work are of three distinct types. The *susan* testcase has a thermal automata with a large number of locations (14164), as well as a large number of outgoing discrete transitions from locations (upto 168). The *basicmath* testcase has a small number of locations (18), and a small number of outgoing discrete transitions from locations (up to 3). The *fft_3_1* testcase (and other testcases based on the *FFT* application) has small branching (up to 5 outgoing discrete transitions), but a large number of locations (8888). This thermal automata also has some long cycles. The amount of branching in the thermal automata is guided by the decision branches in the task-graph model, and the variation in execution delay of tasks.

Increasing size of time-step. Experimental results indicate that an increase in the size of the time-step does not always guarantee a reduction in the cost of analysis. We observe improvements in time taken for analysis if there are small variations in power dissipated in different processors. For example, in case of the *basicmath* testcase, the power dissipation is almost constant for more than 95% of the execution time. However, when there are significant variations in power dissipation, like in case of the *susan* testcase, these gains are not observed in all cases. Moreover, the accuracy of analysis is significantly hampered. We believe that for generic problems which have significant variations in power dissipation, exhaustive analysis with a subset (*1-of-k*) of transitions (which mimics lower branching achieved when the size of time-step is increased) is a viable alternative to increasing the size of the time-step.

Identifying "hot"-cycles. Directed simulation presents a way to reason about the expected worst-case thermal behavior of the application, over long execution time windows. For an effective directed simulation it is necessary that most (preferably all) cycles in the thermal automata be short, in terms of execution time duration, so that they can be covered by a short time-bounded analysis. Even in this case, a subset of (*1-of-k*) transitions may have to be analyzed for performing exhaustive analysis in many cases (*1-of-5* transitions are analyzed for the *susan* testcase). If there are long cycles in the thermal automata, then the time bound for analysis to identify "hot"-cycles will have to be large. In case of models with high branching (like the *susan* testcase), it will be infeasible to perform this analysis with reasonable resources. However, if an application model has low branching (but long cycles), then it implies that the variation between the expected worst case (along the "hot"-cycle) and random simulation will be small. In this case a *1-of-k* exhaustive analysis, or even a random simulation may provide a good estimate of thermal behavior. It is also notable that

long cycles occur in the thermal automata when there are few synchronizations between threads (no synchronization in the *FFT* testcases), and the execution delays are such that their LCM is large. The execution of such models, which have less synchronization, is such that either power dissipation continuously occurs, or negligible power is dissipated in processors. In this case the time-step can be increased to reduce branching further, as well as improve speed of simulation in certain cases.

We believe that the designer must select an appropriate strategy, and employ engineering enhancements proposed in this work, in conjunction with the nature of the thermal automata and the power dissipation profile of tasks.

5. CONCLUSIONS

In this work we address the problem of thermal analysis of multiprocessor SoC (MPSoC) applications and propose a model checking-based approach for solving this problem. This problem is currently addressed by simulation-based methods which cannot guarantee sufficient coverage. Hence the formal method-based technique presented in this work improves upon the current state-of-the-art. In this work we propose a hybrid automata-based model for describing the thermal behavior of an MPSoC application, which to the best of our knowledge is the first application of hybrid automata for modeling such behavior. We also map the thermal analysis problem to a hybrid automata reachability verification problem.

We identify that although there are no known tools for power dissipation analysis of all execution runs of a task, it is needed for thermal analysis. This problem is addressed by a machine learning approach using Markov chains to model power dissipation for execution runs which were not covered by power-performance simulation.

We argue that due to the large number of continuous variables, available hybrid automata analysis tools cannot be directly used for solving the preceding reachability analysis problem. To this end we develop a method, inspired by two well-known principles of model checking, namely counterexample-guided abstraction refinement [Clarke et al. 2003] and bounded model checking [Clarke et al. 2001], for solving this problem. We develop a proof-of-concept tool called HeatCheck and use it to analyze several static schedules on a quad code multiprocessor system. The performance of HeatCheck is significantly better than the polyhedral hybrid automata verification tool PHAVer [Frehse 2005]. We also propose a directed simulation setup, for studying the thermal behavior of the application for long execution time durations.

REFERENCES

ALUR, R., DANG, T., AND IVANIVIC, F. 2006. Predicate abstraction for reachability analysis of hybrid systems. *Trans. Embed. Comput. Syst. 5*, 1, 152–199.

ALUR, R., GROSU, R., HUR, Y., KUMAR, V., AND LEE, I. 2000. Modular specification of hybrid systems in charon. In *Proceedings of the 3rd International Workshop on Hybrid Systems: Computation and Control (HSCC'00).* Springer, 6–19.

ASARIN, E., DANG, T., FREHSE, G., GIRARD, A., LE GUERNIC, C., AND MALER, O. 2006. Recent progress in continuoushybrid reachability analysis. In *Proceedings of the IEEE International Symposium on Computer-Aided Control Systems Design*. 1582–1587.

ASARIN, E., DANG, T., MALER, O., AND BOURNEZ, O. 2000. Approximate reachability analysis of piecewise-linear dynamical systems. In *Proceedings of the 3rd International Workshop on Hybrid Systems: Computation and Control (HSCC'00)*. Springer, 20–31.

AUSTIN, T. M., LARSON, E., AND ERNST, D. 2002. Simplescalar: An infrastructure for computer system modeling. *IEEE Comput. 35*, 2, 59–67.

BARTON, P. I. AND LEE, C. K. 2002. Modeling, simulation, sensitivity analysis, and optimization of hybrid systems. *ACM Trans. Model. Comput. Simul. 12*, 4, 256–289.

BOWMAN, H., FACONTI, G. P., AND MASSINK, M. 1998. Specification and verification of media constraints using UPPAAL. In *Design, Specification and Verification of Interactive Systems,* P. Markopoulos and P. Johnson, Eds. Springer, 261–277.

BROOKS, D., TIWARI, V., AND MARTONOSI, M. 2000. Wattch: A framework for architectural-level power analysis and optimizations. In *Proceedings of the 27th Annual International Symposium on Computer Architecture (ISCA'00)*. ACM Press, New York, 83–94.

BUCK, J., HA, S., LEE, E. A., AND MESSERSCHMITT, D. G. 1994. Ptolemy: A framework for simulating and prototyping heterogenous systems. *Int. J. Comput. Simul. 4*, 2, 155–182.

CHOI, J., CHER, C.-Y., FRANKE, H., HAMANN, H., WEGER, A., AND BOSE, P. 2007. Thermal-Aware task scheduling at the system software level. In *Proceedings of the International Symposium on Low Power Electronics and Design (ISLPED'07)*. ACM, New York, 213–218.

CLARKE, E., BIERE, A., RAIMI, R., AND ZHU, Y. 2001. Bounded model checking using satisfiability solving. *Formal Methods Syst. Des. 19*, 1, 7–34.

CLARKE, E., GRUMBERG, O., JHA, S., LU, Y., AND VEITH, H. 2003. Counterexample-Guided abstraction refinement for symbolic model checking. *J. ACM 50*, 5, 752–794.

DAS, D., CHAKRABARTI, P. P., AND KUMAR, R. 2007. Functional verification of task partitioning for multiprocessor embedded systems. *ACM Trans. Des. Autom. Electron. Syst. 12*, 4, 44.

DAS, D., CHAKRABARTI, P. P., AND KUMAR, R. 2009. Scenario-Based timing verification of multiprocessor embedded applications. *ACM Trans. Des. Autom. Electron. Syst. 14*, 3, 37.

DASDAN, A., RAMANATHAN, D., AND GUPTA, R. K. 1998. A timing-driven design and validation methodology for embedded real time systems. *ACM Trans. Des. Autom. Electron. Syst. 3*, 4.

DRAKE, M. AND HOFFMANN, H. R. R. A. S. 2006. Mpeg-2 decoding in a stream pro-gramming language. In *Proceedings of the 20th International Symposium on Parallel and Distributed Processing (IPDPS'06)*.

EISLEY, N., SOTERIOU, V., AND PEH, L.-S. 2006. High-Level power analysis for multi-core chips. In *Proceedings of the International Conference on Compilers, Architecture and Synthesis for Embedded Systems (CASES'06)*. ACM, New York, 389–400.

ESPOSITO, J. M. AND KUMAR, V. 2007. A state event detection algorithm for numerically simulating hybrid systems with model singularities. *ACM Trans. Model. Comput. Simul. 17,* 1, 1.

FREHSE, G. 2005. Phaver: Algorithmic verification of hybrid systems past hytech. In *Proceedings of the 8th IEEE International Workshop on Hybrid Systems: Computation and Control (HSCC'05)*. 258–273.

GIRARD, A. 2005. Reachability of uncertain linear systems using zonotopes. In *Proceedings of the 8th IEEE International Workshop on Hybrid Systems: Computation and Control (HSCC'05)*. 291–305.

GUTHAUS, M. R., RINGENBERG, J. S., ERNST, D., AUSTIN, T. M., MUDGE, T., AND BROWN, R. B. 2001. MiBench: A free, commercially representative embedded benchmark suite. In *Proceedings of the IEEE 4th Annual Workshop on Workload Characterization*.

HENZINGER, T. A., HO, P.-H., AND WONG-TOI, H. 1997. HYTECH: A model checker for hybrid systems. *Int. J. Softw. Tools Technol. Transfer 1*, 1-2, 110–122.

HUA, S., QU, G., AND BHATTACHARYYA, S. S. 2006. Energy-Efficient embedded software implementation on multiprocessor system-on-chip with multiple voltages. *ACM Trans. Embed. Comput. Syst. 5*, 2, 321–341.

HUANG, W., GHOSH, S., VELUSAMY, S., SANKARANARAYANAN, K., SKADRON, K., AND STAN, M. R. 2006. Hotspot: A compact thermal modeling methodology for early-stage vlsi design. *IEEE Trans. VLSI Syst. 14*, 5, 501–513.

HUNG, W.-L., XIE, Y., VIJAYKRISHNAN, N., KANDEMIR, M., AND IRWIN, M. J. 2005. Thermal-Aware task allocation and scheduling for embedded systems. In *Proceedings of the Conference on Design, Automation and Test in Europe (DATE '05)*. IEEE Computer Society, 898–899.

IGNOWSKI, J. S., BOSTAK, C., AND PARKS, W. H. 2008. Power estimation for a semiconductor device. US Patent application US20080234953.

ISARD, M., BUDIU, M., YU, Y., BIRRELL, A., AND FETTERLY, D. 2007. Dryad: Distributed data-parallel programs from sequential building blocks. *SIGOPS Oper. Syst. Rev. 41*, 3, 59–72.

ISCI, C., BUYUKTOSUNOGLU, A., CHER, C.-Y., BOSE, P., AND MARTONOSI, M. 2006. An analysis of efficient multi-core global power management policies: Maximizing performance for a given power budget. In *Proceedings of the Annual ACM/IEEE International Symposium on Microarchitecture (MICRO'06)*. IEEE Computer Society, 347–358.

JAYASEELAN, R. AND MITRA, T. 2008. Temperature aware task sequencing and voltage scaling. In *Proceedings of the IEEE International Conference on Computer-Aided Design (ICCAD'08),* S. R. Nassif and J. S. Roychowdhury, Eds. IEEE, 618–623.

JEJURIKAR, R. AND GUPTA, R. 2004. Procrastination scheduling in fixed priority real-time systems. *SIGPLAN Not. 39,* 7, 57–66.

JUNG, H. AND PEDRAM, M. 2008. Improving the efficiency of power management techniques by using bayesian classification. In *Proceedings of the 9th International Symposium on Quality Electronic Design (ISQED'08)*. IEEE Computer Society, 178–183.

LI, P., PILEGGI, L. T., ASHEGHI, M., AND CHANDRA, R. 2006. LC thermal simulation and modeling via efficient multigrid-based approaches. *IEEE Trans. Comput. Aided Des. Integr. Circ. Syst. 25*, 9, 1763–1776.

LIU, Y., DICK, R. P., SHANG, L., AND YANG, H. 2007. Accurate temperature-dependent integrated circuit leakage power estimation is easy. In *Proceedings of the Conference on Design, Automation and Test in Europe (DATE'07)*. 1526–1531.

LOGHI, M., BENINI, L., AND PONCINO, M. 2007. Power macromodeling of mpsoc message passing primitives. *Trans. Embed. Comput. Syst. 6*, 4, 31.

LUO, J. AND JHA, N. K. 2001. Battery-Aware static scheduling for distributed real-time embedded systems. In *Proceedings of the 38th Conference on Design Automation (DAC'01)*. ACM Press, New York, 444–449.

MARCULESCU, R., OGRAS, U. Y., AND ZAMORA, N. H. 2006. Computation and communication refinement for multiprocessor soc design: A system-level perspective. *ACM Trans. Des. Autom. Electron. Syst. 11,* 3, 564–592.

MITCHELL, T. M. 1997. *Machine Learning*. McGraw-Hill Higher Education.

MOORE, S. K. 2006. Winner: Multimedia monster. *IEEE Spectrum. 43*, 1, 20–23.

MORARI, M. AND THIELE, L., EDS. 2005. *Proceedings of the 8th International Workshop on Hybrid Systems: Computation and Control (HSCC'05)*. Lecture Notes in Computer Science, vol. 3414. Springer.

MOUDGILL, M., WELLMAN, J.-D., AND MORENO, J. H. 1999. Environment for powerpc microarchitecture exploration. *IEEE Micro 19*, 3, 15–25.

PARK, T. AND BARTON, P. I. 1996. State event location in differential-algebraic models. *ACM Trans. Model. Comput. Simul. 6*, 2, 137–165.

POP, P., ELES, P., PENG, Z., AND POP, T. 2006. Analysis and optimization of distributed real-time embedded systems. *ACM Trans. Des. Autom. Electron. Syst. 11*, 3, 593–625.

PRESS, W. H., TEUKOLSKY, S. A., VETTERLING, W. T., AND FLANNERY, B. P. 1992. *Numerical Recipes in C: The Art of Scientific Computing.* Cambridge University Press, UK.

RAO, R. AND VRUDHULA, S. 2008. Efficient online computation of core speeds to maximize the throughput of thermally constrained multi-core processors. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design (ICCAD '08)*. IEEE Press, 537–542.

RAO, R., VRUDHULA, S., AND CHAKRABARTI, C. 2007. Throughput of multi-core processors under thermal constraints. In *Proceedings of the International Symposium on Low Power Electronics and Design (ISLPED'07)*. ACM, New York, 201–206.

RON, D., SINGER, Y., AND TISHBY, N. 1996. The power of amnesia: Learning probabilistic automata with variable memory length. *Mach. Learn. 25*, 2-3, 117–149.

SATISH, N. R. 2009. Compile time task and resource allocation of concurrent applications to multiprocessor systems. Tech. rep. UCB/EECS-2009-19, University of California Berkeley.

SHANG, L., PEH, L.-S., KUMAR, A., AND JHA, N. K. 2004. Thermal modeling, characterization and management of on-chip networks. In *Proceedings of the 37th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'37)*. IEEE Computer Society, 67–78.

SHIN, J., ZYUBAN, V. V., HU, Z., RIVERS, J. A., AND BOSE, P. 2007. A framework for architecture-level lifetime reliability modeling. In *Proceedings of the Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'07)*. IEEE Computer Society, 534–543.

SKADRON, K., STAN, M. R., SANKARANARAYANAN, K., HUANG, W., VELUSAMY, S., AND TARJAN, D. 2004. Temperature-Aware microarchitecture: Modeling and implementation. *ACM Trans. Archit. Code Optim. 1*, 1, 94–125.

SU, H., LIU, F., DEVGAN, A., ACAR, E., AND NASSIF, S. 2003. Full chip leakage estimation considering power supply and temperature variations. In *Proceedings of the International Symposium on Low Power Electronics and Design (ISLPED'03)*. ACM, New York, 78–83.

SUN, F., RAVI, S., RAGHUNATHAN, A., AND JHA, N. K. 2005. Synthesis of application-specific heterogeneous multiprocessor architectures using extensible processors. In *VLSI Design*. IEEE Computer Society, 551–556.

VENKATACHALAM, V. AND FRANZ, M. 2005. Power reduction techniques for microprocessor systems. *ACM Comput. Surv. 37*, 3, 195–237.

WANG, X., SHAKOURI, A., FARSIU, S., AND MILANFAR, P. 2007. Extraction of power dissipation profile in an ic chip from temperature map. In *Proceedings of the 23rd Annual IEEE Symposium on Semiconductor Thermal Measurement and Management (SemiTherm'07)*. 51–56.

WILHELM, R., ENGBLOM, J., ERMEDAHL, A., HOLSTI, N., THESING, S., WHALLEY, D., BERNAT, G., FERDINAND, C., HECKMANN, R., MITRA, T., MUELLER, F., PUAUT, I., PUSCHNER, P., STASCHULAT, J., AND STENSTRÖM, P. 2008. The worst-case execution-time problem—Overview of methods and survey of tools. *Trans. Embed. Comput. Syst. 7*, 3, 1–53.

YEO, I., LIU, C. C., AND KIM, E. J. 2008. Predictive dynamic thermal management for multicore systems. In *Proceedings of the 45th Annual Design Automation Conference (DAC'08)*. ACM, New York, 734–739.

ZAMORA, N. H., HU, X., AND MARCULESCU, R. 2007. System-Level performance/power analysis for platform-based design of multimedia applications. *ACM Trans. Des. Autom. Electron. Syst. 12*, 1, 2.

ZHAN, Y. AND SAPATNEKAR, S. S. 2005. A high efficiency full-chip thermal simulation algorithm. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design (ICCAD'05)*. IEEE Computer Society, 635–638.

ZHANG, S. AND CHATHA, K. S. 2007. Approximation algorithm for the temperature-aware scheduling problem. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design (ICCAD'07)*. IEEE Press, 281–288.

Received March 2008; revised October 2009; accepted November 2009