Formal Verification of Distributed Dynamic Thermal Management

Muhammad Ismail*, Osman Hasan†, Thomas Ebi†, Muhammad Shafique‡, Jörg Henkel‡

*School of Electrical Engineering and Computer Science, National University of Sciences and Technology, Islamabad, Pakistan
†Chair for Embedded Systems (CES), Karlsruhe Institute of Technology (KIT), Germany
‡Email: {10mseemismail, osman.hasan}@seecs.nust.edu.pk; {thomas.ebi, muhammad.shafique, henkel}@kit.edu

Abstract—Simulation is the state-of-the-art analysis technique for distributed thermal management schemes. Due to the numerous parameters involved and the distributed nature of these schemes, such non-exhaustive verification may fail to catch functional bugs in the algorithm or may report misleading performance characteristics. To overcome these limitations, we propose a methodology to perform formal verification of distributed dynamic thermal management for many-core systems. The proposed methodology is based on the SPIN model checker and the Lamport timestamps algorithm. Our methodology allows specification and verification of both functional and timing properties in a distributed many-core system. In order to illustrate the applicability and benefits of our methodology, we perform a case study on a state-of-the-art agent-based distributed thermal management scheme.

I. INTRODUCTION AND RELATED WORK

As the semiconductor industry moves towards smaller technology nodes, elevated temperatures resulting from the increased power densities are becoming a growing concern [11]. High temperatures aggravate reliability threats like aging [11], [12] and soft errors [3], [9], [21]. In fact, Dynamic Thermal management (DTM) of distributed nature has been identified as one the key reliability challenges in the ITRS roadmap [14]. At the same time, the growing integration density is paving the way for future many-core systems consisting of hundreds and even thousands of cores on a single chip [14], [24]. From a thermal management perspective, these systems bring both new opportunities as well as new challenges. For one, whereas DTM in single-core systems is largely limited to Dynamic Voltage and Frequency Scaling (DVFS), many-core systems present the possibility for spreading power consumption in order to balance temperature over a larger area through the mechanism of task migration. However, the increased problem space related to the large number of cores makes the complexity of DTM grow considerably.

Traditionally, DTM decisions have been made using centralized approaches with global knowledge. These, however, quickly become infeasible due to lack of scalability when entering the many-core era [7], [8], [22], [24]. As a result, distributed thermal management schemes have emerged [5], [7], [8], [10] which tackle the complexity and scalability issues of many-core DTM by transforming the problem space from a global one to many smaller regional ones which can exploit locality when making DTM decisions. For a distributed DTM scheme to achieve the same quality as is possible from one using global knowledge, however, it becomes necessary for there to be an exchange of state information across regions in order to negotiate a near-optimal system state configuration [7].

The choice of tuning parameters for this negotiation has been identified as a critical issue in ensuring a stable system [15]. Up until now these distributed DTM schemes have been exclusively analyzed using either simulations or running on real systems. However, due to the non-exhaustive nature of simulation, such analysis alone is not enough to account for and guarantee stability in all possible system configurations. Especially when considering many-core systems, the number of different configurations, e.g., task-to-core mappings, grows exponentially with the number of cores. Even if some corner cases can be specifically targeted, there is no proof that these represent a worst-case scenario, and it is never possible to consider or even foresee all corner cases. Moreover, using simulation we may show that for a given set of tasks and cores, a small number of mappings result in localized minima. However, in distributed DTM approaches this actually means that a local region of cores may be successfully applying DTM from their point of view although from the global view temperatures are really maximal. Thus, simulation based analysis cannot be considered complete and often results in missing critical bugs, which in turn may lead to delays in deployment of DTM schemes as happened in the case of Foxton DTM that was designed for the Montecito chip [6].

The above mentioned limitations can be overcome by using model checking [4] for the analysis of distributed DTM. The main principle behind model checking is to construct a computer based mathematical model of the given system in the form of an automata or state-space and automatically verify, within a computer, that this model meets rigorous specifications of intended behavior. Due to its mathematical nature, 100% completeness and soundness of the analysis can be guaranteed [2]. Moreover, the ability to provide counter examples in case of failures and the automatic nature of model checking makes it a more preferable choice for industrial usage as compared to the other mainstream formal verification approaches like theorem proving.

Model checking has been successfully used for analyzing some unicore DTM schemes (e.g., [19], [23]). Similarly, probabilistic model checking of a DTM for multicore architectures is presented in [18]. This work conducted a probabilistic analysis of frequency effects through DVFS, time and power spent over budget along with an estimate of required verification efforts. In order to raise the level of formally verifying complex DTM schemes, statistical model checking of power gating schemes has been recently reported [16]. However, to the best of our knowledge, so far no formal verification method, including model checking, has been used for the verification of
a\textit{ distributed} DTM for many-core systems. This paper intends to fill this gap and proposes a methodology for the functional and timing verification of distributed DTM schemes.

\textbf{A. Our Novel Contributions and Concept Overview}

We present a novel methodology for formal verification of distributed DTM schemes in many-core systems. The key idea is to leverage the SPIN model checker [13] (an open source tool for the formal verification of distributed software systems) and Lamport timestamps [17] (a technique that enables to determine the order of events in a distributed system execution). Our methodology allows the designer to formally model or specify the behavior of distributed DTM schemes in the PROcess MEnt LaNgua (PROMELA). These models are then verified to exhibit the desired functional properties using the SPIN model checker as it directly accepts PROMELA models. Our methodology introduces the Lamport timestamps algorithm in the PROMELA model of the given distributed DTM scheme to facilitate the verification of timing properties via the SPIN model checker.

In order to illustrate the effectiveness of our methodology, we perform a case study on the formal verification of a state-of-the-art agent-based distributed DTM scheme, namely Thermal-aware Agent-based Power Economy (TAPE) [7]. The main reason behind the choice of TAPE for this case study is that it is highly scalable while still achieving comparable results to centralized DTM approaches which operate on global system knowledge like PDTM [25] and HRTM [20].

\textbf{Paper Organization:} The rest of the paper is organized as follows: Section II provides an overview of model checking and SPIN to aid the understanding of the rest of the paper. The proposed methodology is presented in Section III. This is followed by the formal modeling and verification of the TAPE algorithm in Section IV. Formal verification results are discussed in V. Finally, Section VI concludes the paper.

\section{Preliminaries}

\textbf{A. Model Checking}

Model checking [4] is primarily used as the verification technique for reactive systems, i.e., systems whose behavior is dependent on time and their environment, like controllers of digital circuits and communication protocols. The inputs to a model checker include the finite-state model of the system that needs to be analyzed along with the intended system properties, which are expressed in temporal logic. The model checker automatically and exhaustively verifies if the properties hold for the given system while providing an error trace in case of a failing property. The state-space of a system can be very large, or sometimes even infinite. Thus, it becomes computationally impossible to explore the entire state-space with limited resources of time and memory. This problem, termed as state-space explosion, is usually resolved by developing abstract, less complex, models of the system. Moreover, many efficient techniques, like symbolic and bounded model checking, have been proposed to alleviate the memory and computation requirements of model checking.

\textbf{B. SPIN Model Checker}

SPIN model checker [13], developed by Bell Labs, is a widely used formal verification tool for analyzing distributed and concurrent software systems. The system that needs to be verified is expressed in a high-level language PROMELA, which is based on Dijkstra’s guarded command language and has a syntax that is quite similar to the C programming language. The behavior of the given distributed system is expressed using asynchronous processes. Every process can have multiple instantiations to model cases where multiple distributed modules with similar behavior exist. The processes can communicate with one another via synchronous (rendezvous) or asynchronous (buffered) message channels. Both global and local variables of boolean, byte, short, int, unsigned and single dimensional arrays can be declared. Defining new data types is also supported.

Once the system model is formed in PROMELA then it is automatically translated to a automaton or state-space graph by SPIN. This step is basically done by translating each process to its corresponding automaton first and then forming an asynchronous interleaving product of these automata to obtain the global behavior [13].

The properties to be verified can be specified in SPIN using Linear Temporal Logic (LTL) or assertions. LTL allows us to formally specify time-dependant properties using both \textbf{logical} (conjunction ($\&\&$), disjunction ($||$), negation ($!$), implication ($\rightarrow$) and equality ($\leftrightarrow$)) and \textbf{temporal operators}, i.e., always ($\mathcal{A} \Box$), eventually ($\mathcal{E} \Box$), eventually ($\mathcal{E} <\rightarrow$)), next ($\mathcal{X}$) and until ($\mathcal{U}$). For verification, the given property is first automatically converted to a Büchi automaton and then its synchronous product with the automaton representing the global behavior is formed by the SPIN model checker. Next, an exhaustive verification algorithm, like the Depth First Search (DFS), is used to automatically check if the property holds for the given model or not. Besides verifying the logical consistency of a property, SPIN can also be used to check for the presence of deadlocks, race conditions, unspecified receptions and incompleteness. Moreover, for debugging purposes, SPIN also supports random, interactive and guided simulation.

\section{Our Methodology for Formal Verification of Distributed DTM}

The most critical functional aspect of any distributed DTM scheme is its ability to reach near-optimal system state configuration from all possible scenarios. Moreover, the time required to reach such a stable state and the effect of various parameters on this time is the most interesting timing related behavior. The proposed methodology, depicted in Figure 1), utilizes the SPIN model checker to verify properties related to both of these aspects for any given distributed DTM scheme. Our methodology exhibits 7 key steps (discussed in detail in the subsequent sub-sections).

1. \textbf{Modeling System in PROMELA:} a model of the distributed DTM scheme and on-chip many-core system is constructed in the PROMELA language; Lamport Timesteps are added.

2. \textbf{Simulation:} the model is simulated using SPIN to identify modeling bugs at an early stage, i.e., prior to the rigorous formal verification of the model.

3. \textbf{Check for Deadlocks:} the presence of deadlocks in the model are checked using the efficient and rigorous deadlock check feature of the SPIN model checker.

4. \textbf{Specification of LTL Properties:} the desired functional properties for the given DTM are represented in LTL.
5) **Functional Verification**: the LTL properties are formally checked using the verification algorithms of SPIN.

6) **Debug or Model Simplification/ Optimization**: in case of a property failure, the error trace is generated for debugging purposes, whereas in case of state-space explosion, the model is simplified using variable abstractions.

7) **Timing Verification**: The timing properties are verified based on the Lamport Timestamps algorithm.

### A. Constructing the Model of Distributed DTM

The PROMELA model of the given distributed DTM system can be constructed by individually describing each autonomous node of the system using one or more processes. Each process description will also include message channels for representing their interactions with other processes and in this way they can share the information of thermal effects and logical time. Moreover, an initialization process should also be used to assign initial values to the variables used to represent the physical starting conditions of the given DTM system along with the range of each variable. The coding can be done in a quite straightforward manner due to the C like syntax of PROMELA. However, choosing the most appropriate data type for each variable of the given scheme needs careful attention.

**Discretization**: Due to the extensive interaction of DTM schemes with their continuous surroundings, some of the variables used in the models of such schemes are continuous in nature. Temperature is a foremost example in this regard. However, due to the automata based verification approach of model checking, variables with infinite precision cannot be handled. Choosing data-types with large set of possible values also results in state-space-explosion problem because of the large number of their possible combinations. Therefore, we have to discretize all the real or floating-point variables, which usually have either infinite or a large set of values, of the given DTM scheme. The lesser the number of possible values, the faster is the verification. On the other hand, lowering the number of possible values may compromise the exhaustiveness of the analysis. However, it is important to note that the discretization of the variables is not a major concern in functional verification since our focus is on the coverage of all the possible scenarios and not on the computation of exact values.

### B. Modeling Timing Behavior using Lamport Timestamps

Just like any verification exercise of engineering systems, the verification of timing properties of distributed DTM schemes is a very important aspect. For example, we may be interested in the time required to reach a stable state after $n$ tasks are equally mapped to different tiles in a distributed DTM scheme. However, due to the distributed nature of these schemes, formal specification and verification of timing properties is not a straightforward task as we may be unable to distinguish between which one of the two occurring events occurred first. Lamport timestamps algorithm [17] provides a very efficient solution to this problem. The main idea behind this algorithm is to associate a counter with every node of the distributed system such that it increments once for every event that takes place in that node. The total ordering of all the events of the distributed system is achieved by ensuring that every node shares its counter value with any other node that it communicates with and it updates its counter value whenever it receives a value greater than its own counter value.

In this paper, we propose to utilize Lamport timestamps algorithm to determine the total number of events in the PROMELA model of a given distributed DTM scheme. The main advantage of this choice is that we can utilize the SPIN model checker, which specializes in the verification of distributed systems, to specify and verify both functional and timing properties of the given DTM scheme.

We propose to use a global array `now` such that its size is equal to the number of distributed nodes in the given distributed DTM system. Thus, each node will have a unique index in this array and all the processes that are used to model the behavior of this particular node will use the same index. Whenever an event takes place inside a process the value of the corresponding indexed variable in the array `now` is incremented. Whenever two nodes communicate, they can share the values of their corresponding variables in the array `now` and can update them based on the Lamport Timestamps algorithm.

### C. Functional Verification in SPIN

Once the model is developed, we propose to check it via the random and interactive simulation methods of SPIN. The randomized test vectors often reveal some critical flaws and the bugs, which can be fixed by updating the PROMELA model. The main motivation of performing this simulation is to be able to catch PROMELA modeling flaws at an earlier stage.

**Deadlocks**: Distributed systems are quite prone to enter deadlocks, i.e., the situation under which two nodes are waiting for results of one another and thus the whole activity is stalled. It is almost impossible to guarantee that there is no deadlock in a given distributed DTM system using simulation. Model checking, on the other hand, is very well-suited for detecting deadlocks. The deadlock check can be expressed in LTL as $\neg \ltl{X(true)}$, which ensures that at any point in the execution(always), a valid next state must exist and thus there is no point in the whole execution from where the progress halts. If a deadlock is found, then the corresponding error trace is executed on the PROMELA model using simulation to identify
A. An Overview of TAPE

The Thermal-aware Agent-based Power Economy (TAPE) [7] is an advanced distributed DTM approach for many-core systems organized in a grid structure. It employs the concept of a fully distributed agent-based system in order to deal with the complexity of thermal management in many-core systems. Each core is assigned its own agent which is able to negotiate with its immediate neighbors (i.e. adjacent cores, as depicted in Figure 2). Thermal management itself is performed by distributing power budgets which dictate task execution among the cores. Thus the agent negotiation consists of distributing this power budget based on the concept of supply-and-demand, taking the currently measured temperatures into account. Since each agent is only able to trade with its neighbors (east, west, north and south), multiple agent negotiations are required to propagate power budget across the chip. At start-up, the available tasks are randomly mapped on the cores in the grid. Every core $n$ keeps track of its free$_n$ and used$_n$ power units and new task assignment to a core results in increasing and decreasing its used$_n$ and free$_n$ power units, respectively, by a number that is determined by the requirements of the newly assigned task. Re-mapping of tasks is automatically invoked when either there are no free power units available in the node or the difference of temperatures in the neighboring nodes goes beyond certain threshold. The tasks are re-mapped to the nodes having the highest sell$_T_n$ - buy$_T_n$ values and thus the sell$_T_n$ and buy$_T_n$ values of a core govern its agent negotiations.

D. Timing Verification in SPIN

The final step in our methodology is to do the timing verification. For this purpose, we propose to use a very rarely used but useful feature of SPIN, i.e., the ability to compute the ranges of model variables during execution [1]. The values in the range of 0 to 255 are trackable only but various counting variables can be utilized in conjunction to increase this range if required. Based on this feature, we keep track of the values of the array now and thus can verify timing properties of DTM schemes in terms of event executions, such as the time units required to attain the evenly distributed temperature condition under a given set of parameters.

IV. CASE STUDY ON THE FORMAL VERIFICATION OF TAPE DTM

In order to illustrate the applicability and effectiveness of our methodology, we perform a case study on an advanced state-of-the-art Distributed DTM scheme, namely the Thermal-aware Agent-based Power Economy (TAPE) [7].

B. Modeling TAPE in PROMELA

As discussed in Section III, the first step in our methodology (see Figure 1) is to construct the system model of TAPE DTM in PROMELA following the subsequent steps.

Modeling Task Mapping and Re-Mapping: Task mapping and re-mapping is an essential component of TAPE. We developed structured macros, given in Algorithm 1, for these functionalities in PROMELA so that they can be called from the processes running in every core. The PROMELA keyword inline allows us to declare new functions, just like C. Both of these functions are called with two parameters, i.e., $n$, which identifies the node for task execution and the $TTim$, which represents the task time. The higher the value of $TTim$, the more power units it will consume and it is assumed that only 1 power unit is consumed for 1 time unit. As such the free power units are converted to used ones and the temperature is incremented by $4^\circ C$ for each power unit (obtained from specific heat capacity of silica). In case of remapping, we have to find max(sell$_T_n$ - buy$_T_n$) according to [7] to find a suitable tile for mapping the task. The variable now is incremented in every execution of Algorithm 1 to keep track of the time.

Variable Initialization Process: An initialization process, given in Algorithm 2, is used to initialize the data types for all the variables used in our model and perform the initial task mapping. The TAPE algorithm utilizes two normalizing factors, $a_s$ and $a_b$, to reflect temperature effects on the values of buy$_T_n$ and sell$_T_n$ and four weights $\omega_{u,b}, \omega_{u,s}, \omega_{f,b}$ and $\omega_{f,s}$ to reflect the effects of the variables used$_n$ and free$_n$ on the values of buy$_{base}$ and sell$_{base}$. All these variables are represented as real numbers in the TAPE algorithm of [7]. However, SPIN does not support real numbers and hence these variables must be discretized as explained in the previous section. The ranges
of these variables can be found from the TAPE description [7]. For example, the values of normalizing factors, $a_s$ and $a_b$, must be bounded in the interval $[0, 1]$. Due to the inability to express real numbers, we use integers in the interval $[1, 9]$ for assigning values to variables $a_s$ and $a_b$. In order to nullify the effect of this discretization on the overall behavior of the model, we have to divide the equations containing variables $a_s$ and $a_b$ by 10 whenever they are used in the PROMELA model. Similarly, based on the behavior of the TAPE model, integers in the interval $[0, 7]$ are used for the weight variables $\omega_{u,s}, \omega_{f,b}$ and $\omega_{f,s}$ and the interval $[7, 14]$ for the weight variable $\omega_{u,b}$. Therefore, in order to retain the behavior of the TAPE model, we divide these variables by 7 whenever they are used (See e.g., Lines 11 and 12 of Algorithm 2).

### Discretization of the Temperature Variable

Finally, we also have to discretize the allowable increments and decrements for the temperature variable $T_m$, which is assigned the value of the initial temperature $T_0 = 30^\circ C$ at start-up. For this purpose, we assume that the power units consumed for the execution time of a particular task amounts to 1mJ of energy. Thus, the worst-case temperature change that happens in one power unit consumption for the task can now be calculated to be approximately equal to $4\,^\circ K$ using the relationship $1mJ/(CV)$, where $C$ represents the heat capacity equal to $1.547J\cdot cm^{-3}\cdot K^{-1}$ for Silica and $V$ represents the volume of a core, which can be reasonably assumed to be equal to 1mm x 1mm x 150um. This discretization does not affect the verification objectives since we are only interested in the stability condition irrespective of the transient effects.

### Lampport Timestamps for Timing

We increment the value of $now_n$ whenever the node $n$ gives a free power unit to one of its neighbors as a result of agent negotiation or whenever the values $sellT_n$ and $buyT_n$ are updated or whenever mapping, re-mapping, sending or receiving takes place.

### Iterative Model Construction and Verification Issues

It is worth mentioning that the above mentioned PROMELA model of TAPE was finalized after numerous runs through the proposed methodology, i.e., it had to go through deadlock checks and several stages of simplifications and optimizations.

#### Issue-1: Deadlocks

We identified a deadlock in our first PROMELA model of TAPE, which occurred because a single channel was used to model both receiving and sending, which in turn lead to the possibility of missing the status update of a missing neighbor. To avoid this situation, we have used two different processes per node to model sending and receiving channels. Interestingly, this kind of a critical aspect, which prevents the system to achieve stability, was not mentioned or caught by the simulation-based analysis of TAPE that is reported in [7]. This point clearly indicates the usefulness of the proposed approach and using formal methods for the verification of distributed DTM schemes. Likewise, the above mentioned...
variable ranges had to be finalized after many simplification and optimization stages so that the model can be verified by the SPIN model checker without encountering the state-space explosion problem. Moreover, we abstracted the DVFS considerations from the TAPE model since its presence has nothing to do with the functional or timing verification and its removal results in the simplification of the PROMELA model, which in turn leads to a reduced state-space.

Issue-2: Run-Time Scenario Due to the mathematical nature of the PROMELA models, merely the formal specification of TAPE in PROMELA allowed us to catch many system issues. For example the exemplary runtime scenario of Figure 3 in [7] was found to be incorrect as we were not able to recreate it for our PROMELA model for the same input values.

These issues clearly indicate the shortcomings of simulation and are quite convincing to motivate the usage of formal methods for the verification of distributed DTM systems.

**Algorithm 4 Agent Process**

```
 n : identification of node
 proctype agent(n,east,west,north,south)
1: sellTn = sellbasea + (an · (Tmn − Ta))/10;
2: buyTn = buybasea − (an · (Tmn − Ta))/10;
3: if
4: ::(sellTn − buyTn) − (sellTn[i] − buyTn[i]) > τ →
5: if
6: ::free[n] > 0 →
nown = nown + 1; free[n] = free[n] − 1;
7: ::else →
8: nown = nown + 1; usedn = usedn − 1; free[n][i] =
9: ::if
10: ::taskvector > deadline →
remapping(a,b,taskvector); Tmn = Tmn − 4;
11: ::else→ skip;
12: fi
13: fi
14: ::else → skip;
15: fi
*/trading results in change of base buy/sell value*/
16: if
17: ::(buyTn = lastbuy[n]) || (sellTn = lastsell[n]) →
18: nown = nown + 1; lastbuy[n] = buyTn; lastsell[n] = sellTn;
19: eastbuyTn, sellTn, nown;
20: westbuyTn, sellTn, nown;
21: northbuyTn, sellTn, nown;
22: southbuyTn, sellTn, nown;
23: :: skip;
24: fi;
```

**V. Formal Verification Results**

**A. Experimental Setup**

We use the version 6.1.0 of the SPIN model checker and version 1.0.3 of ispin along with the WINDOWS 7 Professional OS running on i7-2600 processor, 3.4 GHz(8 CPUs) with 16 GB memory. The verification is done for a 3x3 grid of nodes (cores) with all of them running the processes and channels described in the previous section. The complete model contains 18 processes and 350 lines of code. The SPIN utility BITSTATE is used for verification purposes since it uses 2808 MB of space while allowing to work with up to 4.8 · 10^9 states.

**B. Verification Results**

**Functional Verification Results:** The most interesting functional characteristic to be verified in the context of TAPE is to ensure that the agent trading is always able to achieve a stable power budget distribution. For instance, it needs to be shown that no circular trading loops emerge where power budget is continuously traded preventing the system from stabilizing. Another possibility is that localized minima form which act as a barrier that prevents power budget from propagating. As a result, cores on one side of the barrier would no longer be able to obtain power budget even if it were available globally, and new tasks would be mapped to the other side of the barrier where power budget has accumulated. If such a scenario is possible, it would result in high temperatures and frequent re-mapping inside the region with the power budget not allowing the system to stabilize even though a global stable configuration would be possible. The non-occurrence of such instabilities can be ensured by verifying that “Eventually the sell-buy value between any two adjacent tiles would become very small”. We have to verify 12 such properties so that all possible node pairs of a 3x3 grid are covered. For example, this property (p001) can be expressed for nodes 00 and 01 as:

```
[]<>((sell_T[0].vector[0]−buy_T[0].vector[0])
−(sell_T[0].vector[1]−buy_T[0].vector[1])<3))
```

In a similar way, we expressed and verified the rest of the 11 properties as well. All the 12 properties hold for TAPE, which guarantees its functional correctness, and the verification statistics of these properties is given in Table I. No unreachable code was detected during the analysis, which ensures that the verification was exhaustive and complete.

**Timing Verification Results:** We formally executed the PROMELA model of the previous section and observed the effect of number of tasks and the values of variables \(a_s\) and \(a_b\), on the number of events and temperature. The values of the global list \(now\) are observed for acquiring the information about the number of events. The overall results of the timing verification are summarized in Graphs 1-2 and Fig. 3-4. These results have been observed for Total Power units=128, 10 Tasks, \(\omega_{u,s} = 2/7, \omega_f,b = 2/7, \omega_f,s = 2/7\) and \(\omega_u,b = 8/7\).

**Graph 1: Effect of \(a_s\)**

**Graph 2: Effect of \(a_b\)**

The events to stability in the graphs refers to the time required to reach a stable state, i.e., a state where power is evenly distributed and no redundant trading of power units take place – only due to elevated temperatures. The value of \(a_b\) is kept constant at 0.2 and the value of \(a_s\) is varied from 0.1 to 0.9 in Graph 1, whereas, \(a_s\) is kept constant at 0.2 in Graph 2 and \(a_b\) is varied from 0.1 to 0.9. From both the graphs, it can be clearly observed that the TAPE algorithm reaches a
TABLE I: Statistics of all the 12 properties verified where p0001 is related to node (0,0)–(0,1) and so on

<table>
<thead>
<tr>
<th>Property</th>
<th>Transitions</th>
<th>States stored</th>
<th>Memory Usage(MB)</th>
<th>Verification Time(sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Property 1 (p0001)</td>
<td>25244478</td>
<td>4786929</td>
<td>2808.8</td>
<td>80.9</td>
</tr>
<tr>
<td>Property 2 (p0102)</td>
<td>25295692</td>
<td>4787223</td>
<td>2808.808</td>
<td>81.3</td>
</tr>
<tr>
<td>Property 3 (p1011)</td>
<td>24991059</td>
<td>4755543</td>
<td>2803.658</td>
<td>83.5</td>
</tr>
<tr>
<td>Property 4 (p1112)</td>
<td>25193755</td>
<td>4727341</td>
<td>2816.247</td>
<td>82.7</td>
</tr>
<tr>
<td>Property 5 (p2021)</td>
<td>25244421</td>
<td>4769296</td>
<td>2808.808</td>
<td>84.1</td>
</tr>
<tr>
<td>Property 6 (p2122)</td>
<td>25244421</td>
<td>4769296</td>
<td>2808.808</td>
<td>83.9</td>
</tr>
<tr>
<td>Property 7 (p0010)</td>
<td>24672155</td>
<td>4721204</td>
<td>2564.477</td>
<td>76</td>
</tr>
<tr>
<td>Property 8 (p1020)</td>
<td>24947360</td>
<td>4751100</td>
<td>2564.477</td>
<td>76.9</td>
</tr>
<tr>
<td>Property 9 (p0111)</td>
<td>25055780</td>
<td>4760200</td>
<td>2564.477</td>
<td>76.9</td>
</tr>
<tr>
<td>Property 10 (p1121)</td>
<td>25097601</td>
<td>4764553</td>
<td>2564.477</td>
<td>77.3</td>
</tr>
<tr>
<td>Property 11 (p0212)</td>
<td>24982388</td>
<td>4749616</td>
<td>2564.477</td>
<td>76.8</td>
</tr>
<tr>
<td>Property 12 (p2122)</td>
<td>25114849</td>
<td>4759582</td>
<td>2564.477</td>
<td>77.3</td>
</tr>
</tbody>
</table>

Fig. 3: Effect of Tasks on Events to Stability.

Fig. 4: Effect of Tasks on Maximum Temperature.

stable state only when \( a_s + a_b < 1 \). The same behavior was observed for other values of \( a_s \) and \( a_b \) as well. This is a key design parameter and was not reported in the simulation based analysis of TAPE in [7].

Fig. 3 shows that the number of events to reach stability increase almost proportionally with an increase in the number of tasks until the first 30 tasks. As the tasks are increased beyond this threshold, the event executions appear to be happening in parallel which in turn reduces the number of events required to reach stability. Moreover, it is also observed from Fig. 3 that the task remapping happens only after the number of tasks are beyond a certain threshold, i.e., when the number of free power units of a tile differs considerably with its neighbor or almost all of its free power units are consumed. This highlights the fact that there is some room for improvement in the remapping mechanism of TAPE. Fig. 4 shows that the maximum measured temperature, \( T_{m(max)} \), also increases with the number of tasks.

It is important to note that the number of events required to reach stability has no relationship with the physical clock of the system and it just provides a relative indication for the amount of time required to reach stability. The distinguishing characteristic of the analysis presented in this section is its exhaustive nature, which cannot be attained by the traditional simulation due to the large number of possibilities. Moreover, the verification process is completely automatic and the human interaction is only required for debugging purposes.

C. Discussion

Generalization: The above methodology is general enough to be used to formally verify both functional and timing properties of any distributed DTM system since these DTM schemes can be described by concurrent communicating processes and thus their behaviors can be captured by the PROMELA language. The main challenge in the modeling phase exists in assigning appropriate data-types to the variables involved and the proposed methodology provides a step-wise approach to address this issue. Moreover, we are always interested in verifying deadlock-free behaviors, functional and timing properties and the proposed methodology caters for all these three verification aspects using the SPIN model checker and Lamport timestamps algorithm.

Limitations: The main limitation of using model checking for the formal verification of DTM schemes is the state-space explosion problem. We proposed to encounter this problem by shrinking the size of the system model by lowering the number of possible values of variables, or in other words, discretizing them. The main compromise being made here is on the exhaustiveness of the analysis in terms of possible input values to the DTM schemes. However, as far as the functional verification of the distributed DTM schemes is concerned, the variable discretization does not impact our main objective, which is to observe the given schemes under all combinations.

VI. CONCLUSIONS

The paper presents a formal verification methodology for distributed DTM systems. The proposed method mainly utilizes the SPIN model checker and Lamport timestamps algorithm to verify both functional and timing properties. To the best of our knowledge, this is the first formal verification approach for distributed DTM systems. For illustration purposes, the paper presents the successful formal verification of TAPE, which is a recently proposed agent-based DTM. This case study clearly indicates the applicability of the proposed methods to verify other prominent distributed DTM approaches.
ACKNOWLEDGMENTS
This work was supported in parts by the German Research Foundation (DFG) as part of the priority program “Dependable Embedded Systems” (SPP 1500 - spp1500.itec.kit.edu). The authors would also like to thank Gerard Holzmann, who is the developer of SPIN model checker, for the useful email discussions and constructive feedback about this work.

REFERENCES

APPENDIX
Algorithm 5 Thermal-aware Agent-Based Power Economy [7]
sellbase\[n\]: Base sell value of a tile \[n\] at time \[t\] buy\[base\][n\]: Base buy value of a tile \[n\] at time \[t\] \[Tn\]: Temperature of a tile \[n\] at time \[t\] sell\[Tn\]: Sell value of a tile \[n\] at a temperature \[Tn\] buy\[Tn\]: Buy value of a tile \[n\] at a temperature \[Tn\] \[N\]: Set of all the neighboring tiles of tile \[n\] lastbuy, lastsell: Last buy/sell values of a tile \[n\] sent to all \[i\] \[N\] buy\[N\][n\], sell\[N\][n\]: List of buy/sell values of neighboring tiles stored in \[n\] \[free\][n\]: Free power units of tile \[n\] \[used\][n\]: Power units used for running tasks on tile \[n\] \[τ\]: Tasks running on tile \[n\] at time \[t\] \[tn\]: sell threshold of tile \[n\] 1: loop 2: for all tiles \[n\] in parallel do 3: at every time interval \[Δt\] do // Calculate base sell value 4: sell\[base\][n\] ← \((u_{n,a} \cdot \text{used}_n + w_{f,a} \cdot \text{free}_n)\) 5: buy\[base\][n\] ← \((w_{h,b} \cdot \text{used}_n = w_{f,b} \cdot \text{free}_n)\) // The temperature increase may happen due to change in PE activity. Modify buy/sell value 6: sell\[Tn\] ← sell\[base\][n\] + a\[b\](\[Tn\] - \[Tbase\]) + 7: buy\[Tn\] ← buy\[base\][n\] - a\[b\](\[Tn\] - \[Tbase\]) 8: if \[\text{sell}_{Tn} - \text{buy}_{Tn} > \text{τ}_n\] then 9: if any free power units are left then 10: decrement \[\text{free}_n\] 11: else 12: apply DVFS on \[n\] to get more free power units 13: decrement \[\text{used}_n\] 14: if the task does not meet the given deadline as DVFS is used then 15: (re-)mapping needs to be invoked 16: else 17: graceful performance degradation if allowed 18: end if 19: end if 20: increment \[\text{free}_n\] 21: end if 22: if \[\text{buy}_{Tn} \neq \text{lastbuy} \text{ or sell}_{Tn} \neq \text{lastsell}\] then 23: send \[\text{buy}_{Tn}\] to all \[i\] \[N\] 24: send \[\text{sell}_{Tn}\] to all \[i\] \[N\] 25: lastbuy ← \[\text{buy}_{Tn}\] 26: lastsell ← \[\text{sell}_{Tn}\] 27: end if // This procedure will propagate until a stable state is reached. 28: end at 29: if received updated buy/sell values from any \[i\] \[N\] then 30: update \[\text{buy}_i\], \[\text{sell}_i\] 31: end if 32: if new task mapped to \[n\] requiring \& power units then 33: \[\text{free}_n \leftarrow \text{free}_n - k\] 34: apply DVFS to PE on tile \[n\] 35: \[\text{used}_n \leftarrow \text{used}_n + k\] 36: end if 37: end for 38: end loop