Theorem Proving Based Formal Verification of Distributed Dynamic Thermal Management Schemes

Muhammad Usama Sardar[a,*], Osman Hasan[a], Muhammad Shafique[b], Jörg Henkel[b]

[a]*School of Electrical Engineering and Computer Science (SEECS)*
National University of Sciences and Technology (NUST), Islamabad, Pakistan
[b]*Chair for Embedded Systems (CES)*
Karlsruhe Institute of Technology (KIT), Karlsruhe, Germany

Abstract

Distributed Dynamic Thermal Management (DDTM) schemes are widely being used nowadays to cater for the elevated chip temperatures for many-core systems. Traditionally, DDTM schemes are analyzed using simulation or emulation but the non-exhaustive and incomplete nature of these analysis techniques may compromise on the reliability of the chip. Recently, model checking has been proposed for formally verifying simple DDTM schemes but, despite several abstractions, the analysis is limited to less than 100 cores due to the state-space explosion problem. As a more scalable approach for next-generation many-core systems, we propose a methodology based on theorem proving to perform formal verification of DDTM schemes. The proposed approach allows specification and verification of both functional and timing properties for any number of cores and for all times. For this purpose, the paper provides a higher-order-logic formalization of a generic DDTM scheme. The proposed generic model can be specialized to formally specify most of the existing DDTM schemes and thus formally verify their thermal properties, like temperature bounds and balancing and time to reach thermal stability, as higher-order-logic theorems. As an illustrative example, the paper presents a formal model and analysis of a Distributed Task Migration based DDTM

*Muhammad Usama Sardar

Email addresses: usama.sardar@seecs.nust.edu.pk (Muhammad Usama Sardar), osman.hasan@seecs.nust.edu.pk (Osman Hasan), muhammad.shafique@kit.edu (Muhammad Shafique), henkel@kit.edu (Jörg Henkel)

scheme for many-core systems.

Keywords: Higher-order Logic, Theorem Proving, Dynamic Thermal Management, Task Migration, Many-core Systems

---

1. Introduction

The ever-increasing transistor integration density, governed by the Moore's law, and the ubiquitousness of many-core systems have led to an enormous increase in the on-chip power consumption as well as temperature [1]. These higher temperatures have a detrimental impact on the reliability [2], performance [3] and lifetime [2] of the chip. Traditionally, packaging and mechanical cooling solutions (e.g. heat sinks, fans) have been used as a thermal mitigation technique. However, they are prohibitively expensive for worst-case temperature [4] and therefore not too scalable for many-core systems [5]. As an alternative, Dynamic Thermal Management (DTM) [6] has been identified as a useful solution for mitigating the thermal emergencies with minimum degradation in performance.

1.1. Classification of DTM Schemes

DTM techniques in many-core systems can be hardware-based, like Dynamic Voltage and Frequency Scaling (DVFS) [7] and Stop-go [8, 6], software-based, like task migration [1, 9] and task scheduling [6], and hybrid [10, 8, 6]. Hardware-based DTM techniques like DVFS are generally known to have more effectiveness to manage temperatures but at the cost of a greater performance degradation [10]. Based on whether the scheme is applied locally to a core or to the entire processor, DTM techniques can be classified into two categories: *centralized* and *distributed* [11]. In centralized DTM (CDTM), the central controller takes a global decision based on knowledge of the parameters (e.g. temperature, power and workload distribution) of all the cores in the system. However, CDTM suffers from a single point of failure [9], higher monitoring and communication traffics [1] (e.g. data from sensor of each core [12] and issuance of DTM commands to each core [1]) and computation overhead [12] for managing DTM at the central controller. Therefore, CDTM may suffer from scalability problems [1, 9] for future many-core systems with hundreds or thousands of cores on a single chip [13, 14]. Distributed DTM (DDTM) schemes have been proposed to mitigate these scalability problems. In DDTM, local decisions are taken. If agents are associated with cores then

typically the communication is limited to neighborhood only and if the agents are associated with applications then an agent may negotiate with all or a selected subset of agents concurrently [15].

1.2. Simulation and Emulation Based Verification of DDTM Schemes

Conventionally, DDTM schemes have been analyzed using simulation and emulation. However, for DDTM schemes, both of these methods lack exhaustiveness [16] in terms of coverage of all the possible combinations of values of the variables involved. *This is primarily due to the involvement of a number of continuous variables.* In the context of DDTM, temperature is undeniably the most important continuous parameter. Additionally, many DDTM schemes (e.g. [17]) involve continuous weights or tuning parameters in the cost function. Hence, an exhaustive simulation of all the values of the continuous variables in DDTM requires exponential number of tests, which in turn demands high computational power, a significant amount of time and high memory resources. But due to limited resources of time and memory, only a subset of all DDTM scheme inputs may practically be tested by simulation-based techniques. This leads to another major shortcoming of simulation-based verification of DDTM schemes, i.e., selection of test vectors. The distributed nature of DDTM further complicates this problem. A random selection of test vectors cannot offer a guarantee of correctness of DDTM scheme since it might miss the meaningful portion of the design space. Even though an intelligent designer might carefully craft some corner cases, these may not represent the worst-case scenario for complex DDTM schemes. Moreover, it may not be possible to consider or even foresee all corner cases. Consequently, simulation-based verification of DDTM schemes is incomplete with respect to error detection, i.e., all errors in a system cannot be detected and thus may result in the deployment of a faulty or inefficient DDTM scheme, which in turn can result in unwanted outcomes.

1.3. Model Checking Based Formal Verification of DDTM Schemes

Recently model checking [18, 19, 20, 21] has been explored for the evaluation of DDTM schemes. Model checking [22] is a widely used formal method. The main idea is to make a finite-state model (FSM) of the system and express the intended system properties in temporal logic. The model checker automatically verifies if the properties hold for the given system while providing an error trace in case of a failing property. However, due to inherent limitations, model checking fails to capture continuous variables, which are

frequently encountered (e.g., temperature, time, weights) in the analysis of DDTM schemes. So these variables have been discretized in all the existing model checking based analysis of DDTM schemes. The performance of DDTM techniques, being highly dependent on the values of temperature and weights, is seriously affected by discretization. Even after discretization, only a few values of each continuous variable have been used in the existing DDTM models in their formal analysis [18, 19, 20, 21] due to the state-space explosion problem, i.e., the inability of a model checker to explore the state space of large models due to the associated large computational power, time and memory requirements. To avoid this problem, only simplified models of DDTM schemes have been verified using model checkers which compromises on the accuracy of the analysis [23]. Moreover, it is possible, in fact a common occurrence, that many key assumptions required for the model checking based proofs are in the mind of the verification engineer and are not documented. Lack of scalability is another major limitation of model checking in the context of verifying DDTM schemes because the properties verified by a model checker only assure that they hold for the selected grid size and initial conditions, e.g., number of tasks, temperatures, weights. Hence, if the properties are verified for 10 tasks, for example, nothing can be claimed about higher number of tasks until and unless it is explicitly proved. Similarly, there is no proof that the properties verified for a fixed grid size, for instance 3x3 grid, will remain valid for larger grid sizes. Therefore, model checking cannot ascertain absolute correctness for DDTM schemes in next-generation thousand-core systems [13, 14].

The increasing usage of many-core systems in safety-critical domains, such as medical, transportation and military, calls for high standards of safety, security and reliability. The above-mentioned limitations often result in missing critical bugs, which may cause a minor injury, irreversible severe injury, amputation or even death. Moreover, they can also result in delays in deployment of DTM schemes as happened in the case of Foxton DTM that was designed for the Montecito chip [24], which in turn caused wastage of time, money and effort.

1.4. Novel Contributions

In this paper, we present a methodology based on higher-order-logic theorem proving [25] for the analysis of DDTM schemes in many-core systems. The key contribution of the paper is a set of predicates, given in Section 4.1, that can be customized to formally model most of the existing DDTM

schemes, irrespective of the number of cores and variable ranges for many-core systems. Such a model can in turn be used to formally verify functional and performance related properties of DDTM schemes as higher-order-logic theorems within the sound core of a theorem prover. To the best of our knowledge, this is the first theorem proving based verification approach for DDTM schemes. The paper also presents the selection and higher-order-logic formalizations of important properties of a generic DDTM scheme, which can be customized to analyze most of the DDTM schemes. Some of the key benefits of the proposed methodology are scalability, flexible modeling and explicit presence of assumptions with the verified properties. Our generic model also covers most of the features, such as DVFS, task migration, heat dissipation and transmission, neighborhood and scalability, exhibited by DDTM schemes. The main challenges associated with verification are intensive user interaction, tedious effort for the development of proofs, distributed nature of the system and notion of time. We have chosen the HOL4 [26] theorem prover for our work, mainly because of its mature libraries of reasoning support in arithmetic and probabilistic analysis. To illustrate the effectiveness of our methodology and the proposed generic formal model of DDTM schemes, we use them to formally model and analyze a distributed task migration scheme [27]. The continuous and detailed nature of the model and the formally verified properties with universally quantified variables for number of cores, temperature and time are some of the distinguishing features of our results compared to the model checking based verification of the same scheme [19]. In contrast to earlier attempts [18, 19, 20, 21] for the analysis of DDTM schemes, our methodology precisely mentions the assumptions under which the properties are verified.

1.5. Paper Organization

The rest of the paper is organized as follows: A critical analysis of the related work is presented in Section 2. Section 3 briefly introduces HOL4 theorem prover along with the features and important symbols. Section 4 presents the proposed methodology along with the formalization of a generic DDTM scheme and the identified important properties for thermal stability of any DDTM scheme. The illustrative example of Distributed Task Migration scheme is presented in Section 5. Finally, we conclude the paper in Section 6 by summarizing our research work and highlighting some potential directions for future work.

2. Related Work

Distributed control systems have many applications in safety-critical domains and thus give us a very relevant analysis problem to the one considered in this paper. Formal methods have been extensively used to analyze a wide variety of safety-critical distributed control systems and software. Many initial works [28, 29, 30] in this domain have utilized model checking tools, such as SPIN, SESA and Symbolic Model Verifier (SMV) for the specification and verification of the main algorithm used for safe distributed control. Due to the inherent limitations of model checking tools for verifying complex systems, theorem provers, such as PVS and HOL, were later applied, for instance in [31], for thorough and detailed verification. With the same motivation, we employ theorem proving for our distributed system.

Ismail et al. [18] were the pioneering ones to propose the usage of formal methods in the context of DDTM scheme verification. In particular, the authors proposed a formal methodology, based on the SPIN model checker [32] and Lamport Timestamps, for the verification of DDTM schemes. The methodology allowed to detect deadlocks in the given DDTM schemes and identified two other key properties of DDTM schemes to be verified: i) The possibility of eventually reaching a stable state, i.e., a state when the distributed nodes of the given DDTM scheme attain an even distribution of temperatures or power, and thus their mutual transactions cease to exist. ii) The time to reach this stable state. The methodology was illustrated by verifying the TAPE DDTM scheme [17]. Given the complexity of the TAPE algorithm, due to the presence of many continuous weight parameters, and the general model checking limitations in the context of DDTM scheme verification, described in Section 1, the verification could be done only for a 3x3 grid, even after many abstractions in the TAPE algorithm [17] while considering at most 10 distinct values for the continuous variables. These abstractions limit the usefulness of applying the proposed methodology for analyzing DDTM schemes as the exhaustiveness of the analysis is compromised.

Bukhari et al. [19] presented the formal analysis of a distributed task migration algorithm [27] using the nuXmv model checker. The main motivation of this work was to be able to build upon the available data type of rational numbers in nuXmv and leverage upon the scalable SMT-based bounded model checking (BMC) capabilities of nuXmv to verify DDTM schemes for larger grids. Moreover, the built-in support for verifying the time for the

verification of a property in nuXmv allowed to skip the usage of Lamport Timestamps and thus reduce the size of the model. The authors were able to verify that the temperature of all the cores is eventually less than or equal to the estimated average temperature of the chip for a set of finite tasks for the considered distributed task migration algorithm [27] over a 9x9 grid but there were still numerous abstractions in the model and analysis. For instance, the average temperature estimation equation for each core originally involves an integral but it has been discretized by an approximate expression that involves a summation. A major shortcoming of the work is that the analysis is done for a temperature range of a single core, i.e., 41°C to 56°C whereas the temperatures for multi-core chips can go as high as 124°C [33]. Moreover, the consideration of the second-level neighbors in a distributed system added unnecessary complexity to the verification problem. Although BMC is sound but it is incomplete, since the diameter of the transition system for a complete procedure is usually unknown. Moreover, capacity issues arise for larger depths.

Iqtedar et al. [20] proposed to use probabilistic model checking in the context of DDTM verification. The main motivation of this work is to provide quantitative information regarding the steady-state probability of success of a functional property instead of satisfaction or dissatisfaction of the property. The approach has been illustrated by verifying the TAPE algorithm for a 4x4 grid using the PRISM model checker. However, some algorithms implemented in PRISM, for instance the Power method for the computation of steady-state probabilities, are based on numerical methods [23] which are inherently incomplete. Moreover, probabilistic model checking tools generally make use of unverified algorithms and optimization techniques [23]. Finally, the properties proved using probabilistic model checking include values, instead of generic mathematical expressions, that introduce approximations in results [23]. Therefore, the computations done by the PRISM model checker cannot be considered to be completely accurate and thus compromise on the accuracy of the analysis. Iqtedar et al. also extended their work by using approximate (statistical) model checking to verify TAPE for a 9x9 grid [21]. The advantage of using statistical model checking in the context of DDTM verification is to analyze the DTM schemes on larger number of cores. However, a key limitation of both works [20, 21] is the use of global variables in the modeling, which does not truly depict the distributed paradigm of DDTM schemes. Moreover, there were a lot of abstractions in both models. Finally, the approximate model checking is inherently incomplete, besides suffering

from all the above-mentioned issues. Hence, DDTM scheme verified by this methodology may lead to the burning of the chip in worst case scenarios.

In order to overcome the above-mentioned limitations, this paper provides a higher-order-logic theorem proving based approach for the formal verification of DDTM schemes. Given the high expressiveness of higher-order logic, we develop generic models of DDTM schemes while representing the continuous variables as true *real* numbers. Moreover, the powerful induction techniques can be used in interactive theorem proving to verify generic properties that are valid for all possible values of grid sizes, weights and time.

3. Preliminaries

In this section, we give a brief introduction to the higher-order-logic theorem proving and the HOL4 theorem prover [34], which is the higher-order-logic theorem prover that we have used in the proposed methodology, to facilitate the understanding of the rest of the paper.

3.1. Theorem Proving

Theorem proving [35] allows verifying that the model of the system (implementation) satisfies its corresponding requirements (specification) by mathematical reasoning. The high expressiveness of higher-order logic [36, 37] allows the formalization of most of DDTM schemes, including their continuous and randomized components, and the soundness and completeness characteristics of theorem proving provide the kind of assurance that is needed in analyzing safety-critical systems, such as DDTM schemes. The main idea behind the theorem proving based analysis of a system is to model the behavior and the desired properties of the given system in an appropriate logic and verify their relationship as a theorem in a theorem prover. A theorem prover is a computer software that allows to verify theorems through the application of predefined inference rules and axioms or already existing theorems and thus preserves soundness.

3.2. HOL4 Theorem Prover

HOL4 is an interactive theorem prover and supports propositional, predicate and higher-order logic. It has support for both forward and backward proofs. Its logical core consists of only 5 fundamental axioms and 8 primitive rules of inference [38]. Soundness is assured as the only way to derive new theorems is to apply rules of inference to these basic axioms or any other

already proved theorem. It has been successfully applied to prove correctness for a wide variety of systems, including software, hardware and physical systems.

The main motivation for selecting HOL4 for our work is the high expressiveness of higher-order logic that can be leveraged to verify correctness of most of the existing DDTM schemes including their random and unpredictable components using appropriate random variables. The availability of an extensive probability theory in HOL4 [23] enables the verification of probabilistic properties of any DDTM algorithm in addition to the deterministic properties. We utilize the real theory of HOL4 to model the continuous variables as true *real* numbers. Moreover, the proposed formal model of DDTM theories is built upon the HOL4 theories on *arithmetic*, *lists* and *pairs*. Some of the commonly used functions from these theories, along with their mathematical interpretations, are given in Table 1.

Table 1: HOL4 Symbols and Functions

HOL4 Symbol	**Meaning**
$\wedge$	Logical *and*
$\vee$	Logical *or*
$\neg$	Logical *negation*
`SUC n`	Successor of a non-negative integer (*num*) (n+1)
`h :: L`	Adds a new element h to list L
`L1 ++ L2`	Appends two lists $L1$ and $L2$ together
`HD L`	Head element of list L
`TL L`	Tail of list L
`EL n L`	n^{th} element of list L
`MEM a L`	True if a is a member of list L
`LENGTH L`	Length of list L
`LUPDATE e n L`	Replaces the n^{th} element of list L by value e
`SNOC b L`	Appends element b at the end of list L
`FST (a, b)`	First component of a pair
`SND (a, b)`	Second component of a pair
`flr x`	Floor of a real number x
`sqrt y`	Square root of a real number y

4. Proposed Methodology

The proposed approach, depicted in Fig. 1, utilizes the sound core of the HOL4 theorem prover to carry out rigorous analysis of DDTM schemes. The key advantages of using higher-order-logic theorem proving for our work include generalization, scalability and ability to deal with continuous variables and thus they allow us to overcome the limitations of simulation and model checking in the context of verifying DDTM schemes.

The first step in the proposed methodology, depicted in Fig. 1, is to capture the behavior of the given DDTM scheme using higher-order logic. The next step is to use this formal model along with the precisely expressed desirable system properties to form a proof goal. The proof goal is then input to HOL4 where it is interactively verified to demonstrate that the system satisfies the key requirements. We now describe the two main steps of the proposed methodology, i.e., developing a formal model of the given DDTM scheme and the formalization of its desired properties in detail.

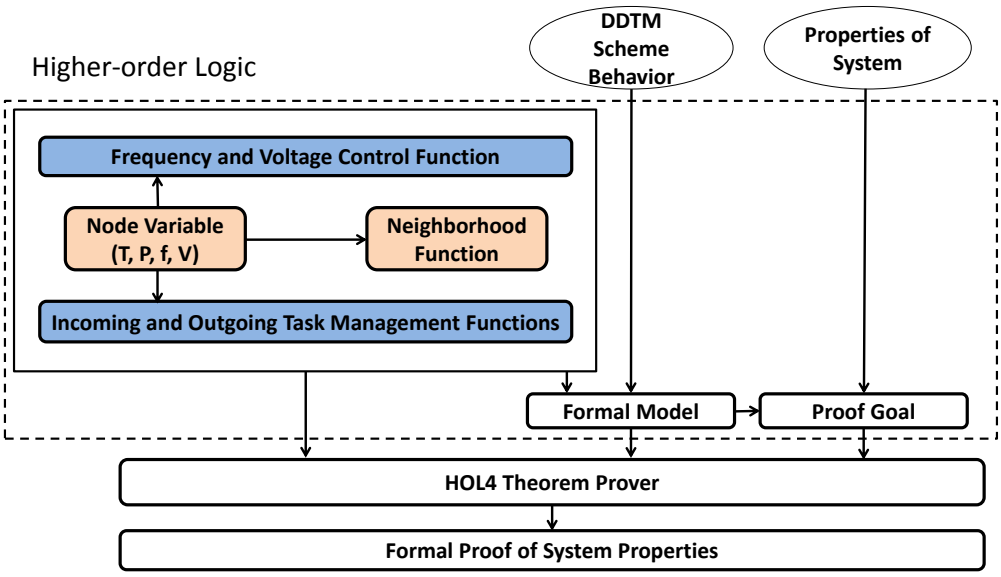


Figure 1: Proposed Methodology

4.1. Formalization of a Generic DDTM Scheme

In this section, we outline the formalization of a generic DDTM scheme. This formal model can be adapted to formalize most of the existing DDTM

schemes and for illustration purposes we would demonstrate its specialization to distributed task migration algorithm [27] in Section 5.

4.1.1. Assumptions

We essentially use the same set of assumptions as used in the corresponding model checking based analysis [19] for the purpose of a fair comparison. The assumptions used in devising the proposed methodology (Table 2) are as follows:

1. In our formalization, the heat dissipation due to the assigned task load is abstracted in terms of the temperature of the core, as was the case for the corresponding model-checking based verification [19]. Since the task is migrated to neighbors only, the heat transmission is considered in terms of the change in temperatures of the neighboring cores involved in task migration. Multiple agent negotiations are required to propagate the heat across the chip.
2. We abstract the process of task migration and assume that a task is migrated completely in a single time unit, as considered in [19] as well. The main intent behind using this assumption is to reduce the proof clutter and thus to simplify the understandability of the proofs presented in the paper. In the future work, this assumption can be relaxed and a more refined relation can also be utilized for the duration of the transfer of the *task context*, i.e., program code, stacks, registers, heap, for distributed memory architectures, as presented in [39].
3. In our methodology, we assume a symmetric 2D grid of cores, i.e., the cores can be arranged in 2D such that the number of rows and columns is the same. This is because of Def. 7 where we replace the number of columns by the square root of the number of cores. A wide variety of DDTM schemes, e.g., [17, 9], for many-core systems satisfy this assumption.
4. For simplicity, the most commonly used 4 primary parameters are selected to characterize a core using a quadruple (Temperature, task load, frequency, voltage) in our formal model. Based on our literature survey, these four parameters characterize almost all the behaviors depicted by a core, e.g., [9, 27]. In case of additional primary parameters used in some DDTM scheme, the methodology requires trivial modifications to adapt it to n-tuple for a core, where n is the desired number of primary parameters in DDTM scheme.

Table 2: Assumptions of Methodology

No.	Assumption	Justification	Source
1.	Heat dissipation and transmission	Abstraction	[19]
2.	Time for task migration	Abstraction	[19]
3.	Symmetric 2D grid of cores	Valid for a wide variety of DDTM schemes	[17, 9]
4.	Primary parameters of a core	Valid for a wide variety of DDTM schemes and can easily be updated to n-tuple	[9, 27]
5.	Neighborhood of a core	Valid for a wide variety of DDTM schemes and can easily be updated to 8-tuple	[17, 27]

5. For simplicity, 4 neighbors are considered in the methodology presented and formalized as a quadruple. This is motivated by the observation that most of DDTM schemes, e.g., [17, 27], utilize 4 neighbors. However, for DDTM schemes with 8 neighbors, the methodology can easily be adapted to a 8-tuple for neighborhood.

4.1.2. Formalization of a Single Core (Node)

We identified four primary parameters to characterize a single core, i.e., *temperature*, *sequence of task loads assigned*, *operating frequency* and *supply voltage*. We propose to utilize the *pair* theory of HOL4 to formalize these parameters as a quadruple, i.e., *(Temperature, Task load list, Frequency, Voltage)* with data type $real\#((real\ list)\#(real\#real))$. Thanks to the high expressiveness of higher-order logic, all these continuous variables can be modeled as true *real* numbers, as opposed to all existing formal DDTM models [18, 19, 20, 21].

The sum of task load assigned to a single core is provided by the function `task_load`, formalized below:

Definition 1. *Task Load of a Single Core*

```
⊢ ∀ node.  task_load node = sum_real (FST (SND node))
```

The function `sum_real` [40] provides the sum of the list of real numbers provided in its argument. The variable `node` represents the quadruple of parameters of a single core and the expression `FST(SND node)` accesses the second component, i.e., task load list, from the quadruple. Similarly, we define the functions `temp, task_load_list, freq` and `vol` that return the first, second, third and fourth components of the quadruple, i.e., temperature, task load list, frequency and voltage of the corresponding core, respectively.

This formalization enables us to model distributed DVFS by allowing the frequency and voltage control function, shown in Fig. 1.

4.1.3. Formalization of the Grid of Nodes

The grid of nodes at a specified instant is formalized as a list `nodes_t`, with data type $(real\#((real\ list)\#(real\#real)))\ list$, where each element represents a core. Therefore, the length of the list equals the number of cores in a many-core system. A specific core in the list can be identified by its unique index number, which ranges from 0 to $N-1$, where N is the total number of cores in a many-core system.

This formalization helps us achieve scalability since the properties are proved for a list of cores, irrespective of its length. This is one of key strengths of our proposed methodology for verifying DDTM schemes in next-generation thousand-core systems [13, 14] compared to all prevailing formal DDTM models [18, 19, 20, 21], which provide support for a specific number of cores, e.g., the largest models have 9x9 cores only [19, 21], even after various abstractions in the model and analysis.

4.1.4. Formalization of Neighborhood in Nodes

Different DDTM schemes have different interpretation of neighborhood in the 2D grid of cores. Most of the distributed schemes, e.g. [27, 17], consider four immediate neighbors (left, right, top and bottom). However, some schemes, e.g. [9], take into account the diagonal cores and thus count 8 neighbors. We propose a neighborhood function, depicted in Fig. 1, that accepts the index of a core, whose neighbors are to be found, and the number of columns in the grid and returns a quadruple or 8-tuple of non-negative integers, representing the indices of 4 or 8 neighbors, respectively. In general, a symmetric 2D grid of cores is considered, so the number of columns is simply the square root of the number of cores. The proposed formalization returns the indices of the neighboring cores while considering the grid position of the core, whose neighbors are required. Some of the grid cores do not have all the neighbors, like the left and top neighbors of a top-left corner core do not exist. In case a neighbor does not exist, the number N, which represents the total number of the given grid of cores, is returned in its corresponding position of the returned quadruple or 8-tuple. It is important to note that N cannot be the index of any core since it is the total number of cores and the indices of cores range between 0 and $N-1$.

We propose to formalize the neighborhood behavior for 4 adjacent neighbors in HOL4 as follows:

Definition 2. *Neighborhood*

```
⊢ ∀ n col.  neighbors4 n col =
     (if n MOD col ≠ 0 then n - 1 else col * col,
     if n MOD col ≠ col - 1 then n + 1 else col * col,
     if col - 1 < n then n - col else col * col,
     if n ≤ (col - 1) * col - 1 then n + col else col * col)
```

The variable `n` represents the index of the core whose neighborhood is to be found and `col` represents the number of columns in the 2D grid of cores. The function `neighbors4` returns the quadruple of indices of neighbors of the form (left, right, top, bottom). The *if* part of the formalization contains the conditions for the existence of the neighbor.In case of existence, index of the respective neighboring core is returned. In case of missing neighbor, we return `col*col`, which is basically the total number of cores in the grid. The individual neighbor can be accessed by the HOL functions `left`, `right`, `top` and `bottom` which accept a quadruple of neighborhood. The `neighbors4` function can easily be tweaked to model the 8 neighbors scenario.

4.1.5. Incoming and Outgoing Task Management

We propose to recursively update all the elements of the list of cores at every new time instant. The proposed formalization essentially involves a condition for activation of task migration policy, e.g., when temperature of a core goes beyond estimated average temperature in [27]. Once the task migration policy is activated for a core, the migration criteria depending on task migration policy under investigation, e.g., 2 conditions in [9], is evaluated, as defined below:

Definition 3. *Selection of Core for Task Migration*

```
⊢ ∀ n_cur n_des col migration_criteria.
selected_core n_cur n_des col migration_criteria =
    if migration_criteria ∧ n_des ≠ col * col
    then n_des else n_cur
```

The variables `n_cur` and `n_des` represent the current and probable destined cores, respectively. The function `migration_criteria`, depending on the scheme under consideration, evaluates whether the conditions of task migration are fulfilled for a specific neighbor. The second condition simply makes sure that `n_des` is a valid core based on the indexing of the cores since `col*col` represents a missing neighbor. Therefore, the function `selected_core` returns `n_des` if it is suitable for task migration, otherwise it returns `n_cur`.

We then defined a function, called `destined_core` (The formal definition is not included in the paper due to space constraints but can be found in our HOL4 script [40]), which uses a nested application of Def. 3 to sequentially check the left, right, top and bottom neighbor for the migration criteria and finally selects the best suited neighboring core as the destined core for task migration. In case the migration conditions are not satisfied for any of the 4 neighbors, the current core itself is returned.

Next, we formally define the function `task_in` that deals with the task migrated to a core:

Definition 4. *Incoming Task*

```
⊢ ∀ n_cur nodes_t cond_in valid_in.
task_in n_cur nodes_t cond_in valid_in =
    if cond_in
    then LUPDATE valid_in n_cur nodes_t else nodes_t
```

The variable `cond_in` represents the conditions for a core to accept the task and the variable `valid_in` represents the quadruple with updated values of the temperature and task load list of the current core `n_cur`. If the conditions are satisfied, we use the `LUPDATE` function, available in the list theory of HOL4, to update the corresponding element of the list of cores `nodes_t`, otherwise there is no change in `nodes_t`.

The function `task_in_update`, given in our HOL4 script [40], applies Def. 4 to all the four neighbors of the current core. Hence, the list of cores `nodes_t` is sequentially updated by all the incoming tasks from its four neighbors.

Next, we formalize the task migrated from a core as follows:

Definition 5. *Outgoing Task*

```
⊢ ∀ n_cur nodes_t cond_out valid_out invalid_out.
task_out_update n_cur nodes_t cond_out valid_out invalid_out =
    if cond_out
    then LUPDATE valid_out n_cur nodes_t
    else LUPDATE invalid_out n_cur nodes_t
```

The variable `cond_out` represents the conditions for the migration of a task from the current core `n_cur` to one of its neighbor. The variables `valid_out` and `invalid_out`, used in Def. 5, represent the quadruple with updated values of the temperature and task load list of the current core `n_cur` in case of satisfaction or violation of conditions, respectively. The values are updated using the `LUPDATE` function, as in Def. 4, accordingly.

It can be observed for the above-mentioned formal task migration functions that whenever the migration criteria is fulfilled, a suitable task load is selected to be migrated. Generally, the maximum loaded task is selected for migration which, in our model, translates to maximum value of the second component in the quadruple of the current core at the given time. We find the index of this maximum value in the task load list by using the function `max_index` [40] and replace this value by zero using the `LUPDATE` function to show that the task has been migrated. Finally, the migrated task is assigned to the destined core, which is identified using the function `destined_core` [40] in our formalization. We append the migrated task load at the end of the task load list of the destined core using the `SNOC` function, which appends the given element at the end of the list.

4.1.6. Notion of Time

The behavior of the time-dependent variables is formalized by HOL predicates on non-negative integers, as done in [41, 42]. Whereas, these non-negative integers represent the ticks of a clock counting physical time in any appropriate units, e.g., nanoseconds. The granularity of the clocks tick is believed to be chosen in such a way that it is sufficiently fine to detect properties of interest [42]. For instance, the list of cores `nodes_t` to model the 2D grid at a fixed instance has the type $(real\#((real\ list)\#(real\#real)))\ list$, so the time-dependent list of cores `nodes` in general would have the type $num \rightarrow (real\#((real\ list)\#(real\#real)))\ list$. The methodology allows the update of the time-dependent variables by relating the value at the next time instant *t+1* to the value at the current time instant *t*. Moreover, modeling time by a non-negative integer, i.e., type *num* in HOL4, allows us to apply induction.

The time variable `t` is incorporated in the overall DDTM formal model as follows:

Definition 6. *DTM Algorithm*

```
⊢ ∀ n_cur col t temp_ref nodes.
    dtm n_cur col t temp_ref nodes =
    (nodes (t + 1) =
     task_out_update n_cur
       (task_in_update n_cur col (temp_ref t) (nodes t))
     (cond_out_dtm n_cur (temp_ref t)
       (task_in_update n_cur col (temp_ref t) (nodes t)))
     (valid_out_dtm n_cur
```

```
    (task_in_update n_cur col (temp_ref t) (nodes t)))
  (invalid_out_dtm n_cur
    (task_in_update n_cur col (temp_ref t) (nodes t))))
```

The function `dtm` provides the updated list of cores at time *t+1* after incorporating the alterations in the parameters (temperature, task load, frequency and voltage) due to task migrations to and from the current core. The variable `temp_ref` with the type *num* → *real list* represents the reference temperature at which the task migration policy is activated. The functions `task_out_update` (Def. 5) and `task_in_update` [40] update the list of cores at the same time instant t and the updated value is assigned to the `nodes` list at time *t+1*. Hence, this definition represents how the `nodes` list is updated in time after doing the necessary task migrations for a single core.

The number of columns in Def. 6 is specified as follows:

Definition 7. *Number of Columns*

```
⊢ ∀ n_cur t temp_ref nodes.
    dtm_helper n_cur t temp_ref nodes =
    dtm n_cur (flr (sqrt (&LENGTH (nodes t)))) t temp_ref nodes
```

We use the square root of the length of the `nodes` list to model the number of columns in a symmetric 2D grid of cores. Since `sqrt` function returns a real number, we apply the `flr` function to be consistent with the types.

Based on the above definitions, we are now in the position of formally expressing the complete algorithm operation as follows:

Definition 8. *Function to apply DDTM to all cores simultaneously*

```
⊢ (∀ t temp_ref nodes.
    dtm_rec 0 t temp_ref nodes = dtm_helper 0 t temp_ref nodes) ∧
  (∀ n t temp_ref nodes.  dtm_rec (SUC n) t temp_ref nodes =
    dtm_helper (SUC n) t temp_ref nodes ∧
    dtm_rec n t temp_ref nodes)
```

The first predicate represents the base case with a single core. In this situation, the function applies Def. 7 to a single core. In case of a many-core system, the function applies recursion on the number of cores to apply DDTM scheme under consideration to all the cores at the same time instant. The first argument to the `dtm_rec` function is one less than the number of cores in the system. For instance, for quad-core system, this argument will be three.

4.1.7. New Task Mappings

We propose to use a sequence of tasks to model incoming new tasks. Whenever a new task is initiated, a core is selected for mapping this task on the basis of the criteria of the DDTM scheme under consideration. Most of the DDTM schemes use either maximization or minimization criteria for task mappings. For example, the TAPE algorithm [17] selects a core with the maximum $sell - buy$ value for this purpose. The maximization criteria is formalized as follows:

Definition 9. *Maximum value of a list with respect to an initial value*

```
⊢ (∀ init.  max_val init [ ] = init) ∧
  (∀ init h t.  max_val init (h::t) = max (max_val init t) h)
```

The function `max_val` returns the maximum value of the list in its argument with respect to an initial value `init` which should be smaller than the maximum value of the list. In case of a *num* list or a list of non-negative real numbers, it is 0. The variables `h` and `t` represent the head element and tail of the list, respectively. The minimization criteria can be formalized similarly.

Finally, the assignment of the task to the selected core is realized in our model by appending the load of the new task at the end of the task load list of the selected core by utilizing the `SNOC` function, which appends a new element at the end of the list.

4.1.8. Behavior of Temperature, Task Load and Execution Time

The proposed approach leverages upon the high expressiveness of higher-order logic to model temperature, task load and execution time relationships, which have been abstracted in all of the state-based formal DDTM models [18, 19, 20, 21]. For instance, change in temperature can be expressed as a function of task load, which defines the temperature change while migrating a unit loaded task. Similarly, task load can be formalized as a function of time, i.e., a function describing the behavior of task load changes with the execution of a task in unit time. The already existing formal DDTM models [18, 19, 20, 21] use the assumption of unity relationship of temperature, task load and execution time in order to avoid the pragmatic issue of state-space explosion. The incorporation of the precise relationships certainly increases the completeness of the models, i.e., their closeness to the actual scenario.

In the light of the above-mentioned formalizations, we recapitulate the effectiveness of the proposed framework to model any DDTM scheme comprehensively and accurately. For DVFS, the frequency and voltage control

function enables the variation of the frequency and voltage of each core to constrain the temperature. For task migration, the incoming and outgoing task management function ascertains the migration of suitable task from a core satisfying the migration criteria to the best suited neighboring core.

4.2. Generic Properties of DDTM Schemes

The proposed approach allows generic mathematical proofs of functional and timing properties of DDTM schemes in order to guarantee functionally correct thermal behavior for any number of cores and schemes generally rather than for a particular finite instance. In our proposed methodology, the desired properties are expressed formally to depict the main objective of DDTM scheme. The properties can then be verified by putting the formal model of the given DDTM scheme in the assumption and the property as the conclusion.

In order to maintain the soundness of the analysis, we thoroughly verified a number of properties of all our definitions and functions, explained in Section 4.1. For instance, the main properties of `max_val` function (Def. 9), which finds the maximum of the list and is the most frequently used function in the migration criteria of DDTM schemes, are given in Table 3. The properties for other functions can be accessed from our proof script [40].

It is of utmost importance to define the bounds of the temperatures of all the cores under all the scenarios of any DDTM scheme because higher temperature has severe detrimental impact on reliability (e.g., electromigration, stress migration, time-dependent dielectric breakdown, thermal cycling, and negative bias temperature instability) [2, 43, 9, 44] and performance (i.e., increased propagation delay and signal integrity issues) [9, 44, 3]. Moreover, a small increase in temperature causes an exponential boost in leakage power dissipation at normal operating conditions [45, 8, 12]. So, in order to maintain thermal stability, the key functional property of all DDTM schemes is the upper bound of the temperature of a core in a many-core system. Furthermore, the upper bounds of the temperatures is the only shared characteristic between all DDTM schemes as the notion of stability varies from one scheme to the other. For instance, some DDTM schemes, e.g. [27, 9, 17], target balancing the temperature profile for improving lifetime and stress; and some schemes, e.g. [46], focus on avoiding critical temperature to avoid chip damage and burnout, and some on both, e.g. [46].

In addition to the properties of the functions and definitions, we propose some generic properties for any DDTM scheme. We propose to verify a

Table 3: Formally Verified Properties of Def. 9

Property	**Formalization**
If all elements of list L are greater than or equal to some value n then max_val n L is going to be a member of L.	$\vdash \forall$ n L. ($\sim$NULL L $\wedge$ ($\forall$ x. MEM x L $\Rightarrow$ (n $\leq$ x))) $\Rightarrow$ (MEM (max_val n L) L)
If all elements of list L are greater than or equal to some value n then max_val n L is going to be greater than or equal to all members of L.	$\vdash \forall$ n L. ($\sim$NULL L $\wedge$ ($\forall$ x. MEM x L $\Rightarrow$ (n $\leq$ x))) $\Rightarrow$ ($\forall$ y. MEM y L $\Rightarrow$ (y $\leq$ max_val n L))
If an element x of list L is greater than or equal to some value n then max_val n L is going to be greater than or equal to all members of L.	$\vdash \forall$ n L. ($\sim$NULL L $\wedge$ ($\exists$ x. MEM x L $\Rightarrow$ (n $\leq$ x))) $\Rightarrow$ ($\forall$ y. MEM y L $\Rightarrow$ (y $\leq$ max_val n L))
The maximum value of the appended list L1 ++ L2 is the higher value of the maxima of the lists L1 and L2.	$\vdash \forall$ L1 L2. max_val 0 (L1 ++ L2) = max (max_val 0 L1) (max_val 0 L2)
If the maximum value of two lists L1 and L2 is the same then the maximum value of the appended list L1 ++ L2 is equal to the maximum of the list L1 or L2.	$\vdash \forall$ L1 L2. (max_val 0 L1 = max_val 0 L2) $\Rightarrow$ (max_val 0 (L1 ++ L2) = max_val 0 L2)
For any list L, max_val 0 L is greater than or equal to 0.	$\vdash \forall$ L. 0 $\leq$ max_val 0 L
The maximum values of the two lists L1 and L2 cannot be both less than the maximum value of the appended list L1 ++ L2.	$\vdash \forall$ L1 L2. $\neg$((max_val 0 L1 $<$ max_val 0 (L1 ++ L2)) $\wedge$ (max_val 0 L2 $<$ max_val 0 (L1 ++ L2)))
The maximum value of the maximum value of the list L is the same value.	$\vdash \forall$ L. max_val 0 [max_val 0 L] = max_val 0 L

thermal stability property that gives the bounds of the temperature at the next time instant, formalized as follows:

Property 1. *Bounds of Temperature Profile*

$\forall$ `n.` `lower_bound` $\leq$ `temp (EL n (nodes (t + 1)))` $\wedge$
`temp (EL n (nodes (t + 1)))` $\leq$ `upper_bound`

The variable `n` represents the index of cores and `t` represents time. The variable `nodes (t+1)` represents the list of cores at next time instant *t+1*. The expression `EL n nodes (t+1)` represents the parameters of *nth* core of the 2D grid at the next time instant *t+1*. The variables `lower_bound` and `upper_bound` represent the lower and upper bounds of the temperature of any core at the next time instant. Hence, the property verifies the lower and upper bounds of the temperature of any core at the next time instant, depending on the scheme under consideration. This is one of the key properties of any DDTM scheme.

Next, for a given sequence of task loads, there exists a time at which the temperature of all the cores does not exceed the threshold temperature, i.e.,

Property 2. *Avoiding Threshold Temperature*

$\exists$ `t.` $\forall$ `n.` `temp(EL n (nodes t))` $\leq$ `temp_ref`

where the variable `t` represents time, `n` represents the index of cores and `temp_ref` represents the threshold temperature. The description of the variable `temp_ref` varies from one scheme to the other. For DDTM schemes that focus on avoiding the critical temperature, `temp_ref` in Property 2 can be replaced with the critical temperature. For DDTM schemes that target the balancing of temperature, this property makes sure that the temperature of each core is within a certain bound of the average estimated in a distributed manner, i.e.,

Property 3. *Temperature Balancing*

$\exists$ `t.` $\forall$ `n.` `temp(EL n (nodes t))` $\leq$ `EL n (temp_avg t)` + Δ

where `temp_avg` represents the estimated average temperature and Δ represents a small number for tolerance. This is a more realistic property as compared to the property considered in [19] which requires temperature of *all* the cores to be less than or equal to the estimated average temperature of the chip, which is possible only in case of an overestimation of actual average.

The timing properties deal with a very pertinent and legitimate question, i.e., how fast these stability criteria are achieved. For instance, the upper bound for the time to stability is formalized as:

Property 4. *Time to Stability*
$\exists$ `t.` $\forall$ `n.` `t < ts` $\Rightarrow$ `temp(EL n (nodes t))` $\leq$ `temp_ref`

where `ts` represents the upper bound of the time to stability and all the other variables have the same context as in the above-mentioned functional properties.

Considering the above-mentioned functional and timing properties, we recapitulate the usefulness of the proposed framework to verify any DDTM scheme thoroughly and precisely. By verifying the bounds of the temperature, we guarantee the functionality of DDTM schemes that they would maintain a safe operating temperature. Moreover, timing properties ascertain that the stability criteria are achieved within a reasonable time. Hence, our methodology is quite generic and flexible and can be used for the verification of most of the existing DDTM schemes. However, the main limitation of our technique is lack of automation and extensive user-interaction for verifying the properties of the system. Moreover, the user-interaction generally requires extensive training in higher-order-logic theorem proving. However, we have tried to facilitate the user intervention process by providing a generic set of definitions and theorems that can be customized or specialized to model and verify many commonly used aspects of DDTM schemes almost automatically.

5. Distributed Task Migration

Liu et al. [27] proposed a fully distributed task migration policy for balanced on-chip temperature distribution. Since in a DDTM scheme, each core is only aware of its own temperature and the temperature of its immediate neighboring cores, the proposed policy utilizes the distributed average signal tracking algorithm [47] to dynamically estimate the average temperature of the chip. According to this algorithm, the estimated average temperature converges to the actual average temperature of the chip.

We selected this scheme because it takes into account the load influence from the neighboring adjacent cores and considers estimated average temperature instead of arbitrarily chosen fixed temperature threshold to achieve thermal balance. Simulation results using a 6x6 grid demonstrate that the proposed scheme reduces the thermal hot spots by 30% and also reduces

on-chip temperature variance as compared to existing distributed thermal management methods [27].

5.1. Algorithm

In order to perform the minimum number of task migrations, the policy is activated for a core only when its temperature exceeds the estimated average temperature, in contrast to fixed temperature threshold which has several problems. Defining the current core as the one where the task migration policy is applied to, and the destined core as the core that could potentially accept migrated task from current core, the migration criteria is fulfilled when the following three conditions are met:

1. $T_{des} < T_{cur}$, where T_{des} is the temperature of the destined core and T_{cur} is that of the current core.
2. $P_{des} < P_{cur}$, where P_{des} is the task load of the destined core in the new execution cycle, and P_{cur} is the counterpart of the current core.
3. $TNP_{des} < TNP_{cur}$, where TNP_{des} is the total task load including the immediate neighboring cores surrounding the destined cores in the new execution cycle, and TNP_{cur} is the counterpart of the current cores.

The algorithm flow of the scheme is given in Algorithm 1 [27] and Fig. 2. In case the temperature T of a current core exceeds the estimated global average temperature T_{avg}, the above-mentioned task migration conditions are evaluated for each of its neighboring core one by one. Whenever all the three conditions are met, the neighboring core is selected as a potential candidate for task migration and the thermal and load parameters of the current core are updated by the respective parameters of the selected core. After repeating the procedure for all the 4 neighbors, the algorithm selects the best choice as the destined core for task migration. In the example flow of Fig. 2 (b), node 0 represents the current core and nodes 1-4 represent the neighboring cores. The checked cores are depicted by filled grey nodes while the destined core is denoted by filled black node. The algorithm checks the migration conditions on all the 4 neighbors and finally selects node 3 as the destined core to migrate the task to.

5.2. Formal Specification

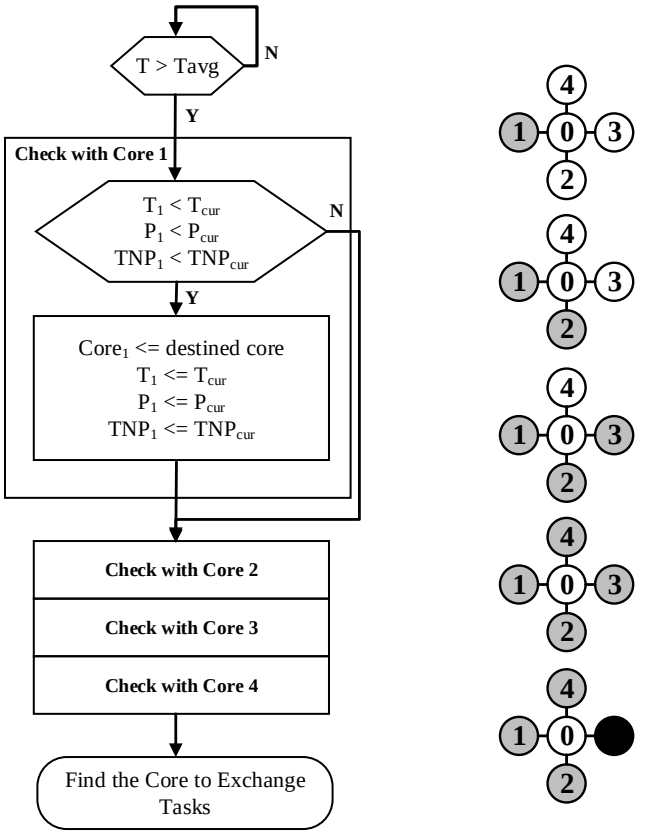
In this section, we utilize the proposed methodology, described in Section 4.1, to formally express the behavior of the distributed task migration algorithm [27], summarized in Section 5.1. The assumptions specifically related to the selected task migration scheme (Table 4) are listed below:

Algorithm 1 Distributed thermal management algorithm for avoiding hot spots [27]

Require: Task loads, many-core system configuration
Ensure: Optimized temperature distribution
Start simulation at room temperature
for each execution cycle **do**
1. Simulate power traces under different task loads
2. Obtain temperature responses of many-core system, and estimate average temperature using distributed state tracking algorithm
if migration criteria are met **then**
Perform distributed task migration using the proposed scheme in Fig. 2 core by core.
end if
end for

1. For a fair comparison with Bukhari et al. [19], we espouse their assumption of an equivalent change in temperature as the task load. Hence, during task migration, the temperature of the current core from which the task is migrated decreases by the same value as the load of the migrated task and the temperature of the destined core increases by the same value. In fact, the rate of change of temperature during increasing and decreasing process can be different. Also, the absolute temperature of two cores can be different depending upon other neighbors. However, the temperature resulting from a given workload may be same and other temperature components (like ambient temperature and temperature increase from other neighbors) act as an offset. If required, this assumption can be removed and a more generalized relation can also be utilized using the methodology presented in Section 4.1.8.
2. We espouse the assumption in [19] that each core executes a single unit of task load in one time unit. The granularity of clocks tick is thus assumed to be chosen to support this assumption [42].

The total task load of a core and its neighboring cores can be formalized for the given task migration algorithm by making use of Defs. 1 and 2 of Section 4.1 to sum the task loads of the current core and its neighbors, as follows:



(a) Task migration method flow (Applicable to any arbitrary core)

(b) A graphic example (The right core is chosen)

Figure 2: Distributed Task Migration algorithm [27].

Definition 10. *Total Task Load*

```
⊢ ∀ n col nodes_t.  total_task_load n col nodes_t =
    task_load (EL n nodes_t) +
    if left (neighbors4 n col) < col * col
     then task_load (EL (left (neighbors4 n col)) nodes_t)
     else 0 +
    if right (neighbors4 n col) < col * col
     then task_load (EL (right (neighbors4 n col)) nodes_t)
     else 0 +
    if top (neighbors4 n col) < col * col
```

Table 4: Assumptions of Case Study

No.	Assumption	Justification	Source
1.	Linear task load and temperature relation	Abstraction	[19]
2.	Granularity of time	Abstraction	[19, 42]

```
    then task_load (EL (top (neighbors4 n col)) nodes_t)
    else 0 +
  if bottom (neighbors4 n col) < col * col
    then task_load (EL (bottom (neighbors4 n col)) nodes_t)
    else 0
```

The variable `n` represents the index of the core for which the total task load is required and the variable `nodes_t` represents the list of cores at a given time. `EL n nodes_t` represents the *nth* core of the 2D grid. The *if* conditions in Def. 10 check the existence of the neighboring core, and if it exists, its task load is added.

Next, the migration criteria, which is required in Def. 3 of Section 4.1, can be defined by formalizing the three migration conditions mentioned in Section 5.1, as follows:

Definition 11. *Migration Criteria*

```
⊢ ∀ n_cur n_des col nodes_t.
migration_criteria_dtm n_cur n_des col nodes_t =
    temp (EL n_des nodes_t) < temp (EL n_cur nodes_t) ∧
    task_load (EL n_des nodes_t) +
     max_val 0 (task_load_list (EL n_cur nodes_t)) <
     task_load (EL n_cur nodes_t) ∧
    total_task_load n_des col nodes_t <
     total_task_load n_cur col nodes_t
```

The function `temp` used in this definition returns the first component of the quadruple, i.e., temperature, of the corresponding core. Hence, the first predicate ensures that the temperature of the destined core `n_des` is lesser than that of the current core `n_cur`. The function `max_val` finds the maxima of the real list in its argument with respect to a given reference value (0 in this case). Since task loads are non-negative, `max_val 0 (task_load_list (EL n_cur nodes_t))` represents the maximum value of the task load assigned to the current core. This task load is added to the task load of the probable

destined core to model the next execution cycle condition assuming migration has taken place. Therefore, the second predicate guarantees that the task load of the destined core in the next execution cycle is lesser than that of the current core. The final predicate utilizes Def. 10 to certify that the total task load of the destined core is lesser than that of the current core. The conjunction of all the three predicates represents that the return value will be *True* only if all the three conditions are satisfied, as required by the algorithm.

The conditions for incoming task, which are required in Def. 4 of Section 4.1, can be formalized for the given task migration scheme as follows:

Definition 12. *Incoming Task Conditions*

```
⊢ ∀ n_cur neighbor col temp_avg_t nodes_t.
cond_in_dtm n_cur neighbor col temp_avg_t nodes_t =
    EL neighbor temp_avg_t < temp (EL neighbor nodes_t) ∧
    (destined_core neighbor col nodes_t = n_cur)
```

The variable `temp_avg_t` with data type *real list* represents the estimated average temperature of each core at a given time and the variable `neighbor` represents one of the four neighboring cores. The first condition checks whether the temperature of the neighboring core has exceeded its estimated average temperature, i.e., task migration policy is activated on the neighboring core. The second condition ensures that the current core `n_cur` is selected as the destined core for `neighbor` to migrate the task.

If both the above-mentioned conditions are satisfied, the task with the maximum load is going to be migrated from neighboring core to the current core. The quadruple with updated parameters, which is needed in Def. 4 of Section 4.1, is formalized for the case of the given task migration scheme as follows:

Definition 13. *Incoming Task Valid Operation*

```
⊢ ∀ n_cur neighbor nodes_t.
valid_in_dtm n_cur neighbor nodes_t =
    (temp (EL n_cur nodes_t) +
     max_val 0 (task_load_list (EL neighbor nodes_t)),
    SNOC (max_val 0 (task_load_list (EL neighbor nodes_t)))
     (task_load_list (EL n_cur nodes_t)),
    freq (EL n_cur nodes_t),
    vol (EL n_cur nodes_t))
```

The expression `max_val 0 (task_load_list (EL neighbor nodes_t))` represents the maximum value of the task load assigned to the neighboring core `neighbor`. The function `SNOC` is utilized to append this maximum loaded task of the neighboring core to the end of the task load list of the current core `n_cur` as required by our methodology (Section 4.1). Hence, in accordance with assumption 1, the temperature of the current core increases by the same value. The frequency and the voltage of the core remain unchanged since the selected scheme involves only task migration.

Next, we specialize generic Def. 5 of Section 4.1 for the selected task migration scheme. We formalize the conditions for task migration from a core as follows:

Definition 14. *Outgoing Task Conditions*

```
⊢ ∀ n_cur temp_avg_t nodes_t.
cond_out_dtm n_cur temp_avg_t nodes_t =
    EL n_cur temp_avg_t < temp (EL n_cur nodes_t) ∨
    max_val 0 (task_load_list (EL n_cur (nodes_t))) < 1
```

The function `cond_out_dtm` checks if the temperature of the current core `n_cur` is greater than its estimated average temperature, i.e., if the task migration policy is activated on the current core. Moreover, since the same action is to be taken in case the maximum task load of the current core is less than unity, we include this condition in Def. 14.

When the above condition for task migration is valid, the updated quadruple of parameters for Def. 5 of Section 4.1 is provided by the following definition:

Definition 15. *Outgoing Task Valid Operation*

```
⊢ ∀ n_cur nodes_t.  valid_out_dtm n_cur nodes_t =
    (temp (EL n_cur nodes_t) -
     max_val 0 (task_load_list (EL n_cur nodes_t)),
    LUPDATE real_0
     (max_index (task_load_list (EL n_cur nodes_t))
      (max_val 0 (task_load_list (EL n_cur nodes_t))))
       (task_load_list (EL n_cur nodes_t)),
    freq (EL n_cur nodes_t),
    vol (EL n_cur nodes_t))
```

The function `max_index` accepts a list and a value and returns the index of the first matching value in the list. The expression `(max_index (task_load_list (EL n_cur nodes_t)) (max_val 0 (task_load_list (EL n_cur nodes_t)) ))` provides the index of the maximum loaded task of the current core. The `LUPDATE` function overwrites this maximum loaded task by 0 to depict that the current core has migrated the task. The temperature is decreased by the value of the maximum loaded task of the current core.

When both of the outgoing task conditions are not true, the updated quadruple of parameters for Def. 5 of Section 4.1 is provided by the following definition:

Definition 16. *Outgoing Task Invalid Operation*

```
⊢ ∀ n_cur nodes_t.  invalid_out_dtm n_cur nodes_t =
    (temp (EL n_cur nodes_t) - 1,
    LUPDATE
      (max_val 0 (task_load_list (EL n_cur nodes_t)) - 1)
       (max_index (task_load_list (EL n_cur nodes_t))
        (max_val 0 (task_load_list (EL n_cur nodes_t))))
         (task_load_list (EL n_cur nodes_t)),
    freq (EL n_cur nodes_t),
    vol (EL n_cur nodes_t))
```

The function `invalid_out_dtm` represents the situation in which the core continues to execute its tasks. In accordance with assumption 2, the `LUPDATE` function is utilized to decrement the value of maximum loaded task of the current core by unity. The temperature also drops by unity according to assumption 1.

Def. 8, given in Section 4.1, can now be used to completely represent the distributed task migration algorithm [27] based on Defs. 10 - 16. The formal specification of the algorithm, given in this subsection, models the continuous variables as true *real* numbers, in contrast to all existing formal DDTM models [18, 19, 20, 21]. Moreover, we model DDTM scheme for any number of cores, whereas the existing formal DDTM models [18, 19, 20, 21] use a limited number of cores. Thus, our approach is scalable for next-generation thousand-core systems [13, 14]. Finally, complex relations can be modelled accurately due to the high expressiveness of our proposed approach. Thus, our formal model achieves higher degree of accuracy and reliability.

5.3. Formal Verification

In this section, we present the theorems proved in HOL4 to validate the functionality of distributed task migration algorithm [27], formalized above. Since the original algorithm comes without any paper proof, the proofs were developed and then verified in HOL4. The algorithm uses the estimated average temperature to activate the task migration policy, hence for this algorithm, `temp_ref` denotes the average temperature estimated in a distributed way.

Considering the detrimental effects of high temperature on the reliability, performance and lifetime of the chip, we verify a property that if the condition of activation of the migration policy is not satisfied for all the cores, then their temperature decreases by `t3` temperature units in $t3$ time instants.

Theorem 1. *When Task Migration Policy is not Activated*

```
⊢ ∀ t n_cur t3 temp_ref nodes.
     ∼(NULL (nodes t)) ∧
     (∀ t1 t2.  LENGTH (nodes t1) = LENGTH (nodes t2)) ∧
     (∀ t.  dtm_rec (PRE (LENGTH (nodes t))) t temp_ref nodes) ∧
     n_cur < LENGTH (nodes t) ∧
     (∀ n_cur t.  temp(EL n_cur (nodes t)) ≤ EL n_cur (temp_ref t))
     ⇒ (temp (EL n_cur (nodes (t+t3))) =
     temp (EL n_cur (nodes t)) - &t3)
```

The first assumption of the theorem states that the list of `nodes` at time `t` must not be an empty list, meaning that we should have at least one core in the grid. According to the second assumption, the length of `nodes` list remains the same at all times, which is true as the number of cores never change in a given 2D grid. The third assumption ensures that the DTM scheme is recursively applied on all cores. According to the fourth assumption, the index of the current core, i.e., `n_cur`, must be less than the length of the `nodes` list. This assumption ensures that the current core is a part of the given 2D grid of cores. The fifth assumption formally describes that for all cores `n_cur` and for all times `t`, the temperature of all the cores in the grid is less than or equal to their estimated average temperature. This condition is added to represent the situation when the task migration policy is not activated on any core. If the given conditions hold for $t3$ time instants, the conclusion of the theorem ensures that the temperature of any core after $t3$ time instants will be decremented by `t3`. This should be true

as in our model, in case a task migration policy is not activated, a core continues to execute its assigned task loads and the temperature of each core is decremented by unity in unit time.

Next, we verify the upper bound of the temperature under the scenario when the migration conditions are not satisfied.

Theorem 2. *No Migration Scenario*

```
⊢ ∀ t n_cur t4 temp_ref nodes.
     ∼(NULL (nodes t)) ∧
     (∀ t1 t2.  LENGTH (nodes t1) = LENGTH (nodes t2)) ∧
     (∀ t.  dtm_rec (PRE (LENGTH (nodes t))) t temp_ref nodes) ∧
     n_cur < LENGTH (nodes t) ∧
     (flr(sqrt(&LENGTH (nodes t))) ** 2 = LENGTH(nodes t)) ∧
     (∀ n_cur t.destined_core n_cur (flr(sqrt(&LENGTH (nodes t))))
       (nodes t) = n_cur) ∧
     (∀ t.  1 ≤ max_val 0 (task_load_list (EL n_cur (nodes t)))) 
     ⇒ temp (EL n_cur (nodes (t+t4))) ≤
     temp (EL n_cur (nodes t)) - &t4
```

The first four assumptions are the same as the ones used in Theorem 1. The HOL operator ** represents the exponentiation and hence, the fifth assumption makes sure that the square of the number of columns is equal to the length of the `nodes` list, which holds for our model because we consider symmetric 2D grid of cores. The next assumption states that for all cores and for all times, the `destined_core` function for current core `n_cur` returns itself as the best suited destination, meaning that the current core has no suitable neighboring core for task migration. The final assumption is that for all times, the maximum value of the task load of the current core exceeds unity. Given that the above conditions hold for $t4$ time instants, the upper bound for temperature after $t4$ time instants will be the current temperature minus $t4$.

Finally, we verify a property that gives the bounds of the temperature at the next time instant without any constraints on the core temperatures or destination cores. Hence, it considers all the possible scenarios which DDTM scheme can go into.

Theorem 3. *Generic Temperature Bounds*

```
⊢ ∀ t n_cur temp_ref nodes.
     ∼(NULL (nodes t)) ∧
```

```
(∀ t1 t2.  LENGTH (nodes t1) = LENGTH (nodes t2)) ∧
(∀ t.  dtm_rec (PRE (LENGTH (nodes t))) t temp_ref nodes) ∧
n_cur < LENGTH (nodes t) ∧
(flr (sqrt (&LENGTH (nodes t))) ** 2 = LENGTH (nodes t)) ∧
(∀ n_cur.0 <= max_val 0 (task_load_list (EL n_cur (nodes t))))∧
(left (neighbors n_cur (flr (sqrt (LENGTH (Nodes t))))) <
 LENGTH (Nodes t)) ∧
(right (neighbors n_cur (flr (sqrt (LENGTH (Nodes t))))) <
 LENGTH (Nodes t)) ∧
(top (neighbors n_cur (flr (sqrt (LENGTH (Nodes t))))) <
 LENGTH (Nodes t)) ∧
(bottom (neighbors n_cur (flr (sqrt (LENGTH (Nodes t))))) <
 LENGTH (Nodes t))
⇒ temp (EL n_cur (nodes t)) -
max_val 0 (list_of_max_val (nodes t)) - 1 ≤
temp (EL n_cur (nodes (t + 1))) ∧
temp (EL n_cur (nodes (t + 1))) ≤
temp (EL n_cur (nodes t)) +
4 * max_val 0 (list_of_max_val (nodes t))
```

The first five assumptions are the same as the ones used in Theorem 2. The next assumption states that the maximum value of the task load has to be non-negative, which holds since task loads in real world are non-negative. The last four assumptions state that the index of the `left`, `right`, `top` and `bottom` neighbors must be less than the length of the nodes list. This assumption ensures that the neighbors are a part of the given 2D grid of cores. On the basis of these generic assumptions, the theorem gives the lower and upper bounds of the temperature of any core, i.e., $max\_val\ 0\ (list\_of\_max\_val\ (nodes\ t)) - 1$ and $4 * max\_val\ 0\ (list\_of\_max\_val\ (nodes\ t))$, respectively. The function `list_of_max_val` provides the list of maximum values of the task loads assigned to all the cores. Therefore, the lower bound is one less than the current temperature minus the maximum value of the task load of all the cores and the upper bound is the current temperature plus 4 times the maximum value of the task load of all the cores.

Our work emphasizes the importance of determining the bounds of the temperature after applying DDTM scheme which can assist the designers in reducing the cost of the design. Instead of focusing on the worst-case scenario, they can design cooling solutions for temperature close to average and rely on verified DDTM schemes to operate when this temperature is

Table 5: A fair comparison of the strategies

No.	Criteria of comparison	Ismail et al. [18]	Bukhari et al. [19]	Iqtedar et al. [20]	Our model
1.	Verification tool	SPIN model checker	nuXmv model checker	PRISM model checker	HOL4 theorem prover
2.	Modeling continuous variables	Integers	Rational numbers	Integers	Real numbers
3.	Maximum number of cores	9	81	16	Any number of cores
4.	Temperature range for analysis	30°-86°C	41°-56°C	30-62 K	Any desired continuous range
5.	Number of tasks in a core	1	1	1	Any desired range
6.	Number of power units per task	1	1-4	6	Any desired range
7.	Maximum number of tasks in the system	9	324	96	Any desired value
8.	Human effort (Approximate man hours)	100	80	50	600

exceeded. Moreover, our proof goals are generalized, meaning that they can cater any number of cores, thus ensuring scalability for future many-core systems [13, 14]. Furthermore, the continuous variables are modeled as real numbers. The most useful benefit of the proposed approach is its accuracy as the theorems are being verified in a formal way using a sound theorem prover. Thus, there is no risk of human error.

The high accuracy of the proved scheme in HOL4 comes at the cost of intensive user interaction. It is important to note that formalizing and then verifying its properties was a very tedious effort since it is very tiresome to develop proofs of complex real-time systems, as the original scheme was proposed without any analytical proof. The distributed nature of the scheme and the notion of time further complicate the proofs as there are two variables for induction, i.e. the list of nodes and time. Theorem 3 was the most challenging one. The complete formalization took around 4000 lines of HOL code and approximately 600 man hours. Our HOL4 proof script is available for download [40], and thus can be benefitted by researchers and verification engineers for further developments and analysis of different DDTM schemes.

5.4. Comparison with Model Checking Based Analysis

All the existing formal models [18, 19, 20, 21] employ model checkers to perform the analysis of DDTM schemes, whereas our proposed methodology utilizes the theorem prover to carry out rigorous analysis of DDTM schemes. In this section, we present a fair and well-structured comparison (Table 5) between the model checking and theorem proving verification strategies specifically for DDTM schemes. Each point is explained as follows:

1. The verification methodology of Ismail et al. [18] was primarily based on using the SPIN model checker and Lamport Timestamps. Bukhari et al. [19] utilized the scalable SMT-based BMC capabilities of nuXmv model checker and Iqtedar et al. [20, 21] used a probabilistic model checker PRISM while we exploit the sound core of HOL4 theorem prover for the formal analysis of DDTM schemes.
2. Ismail et al. and Iqtedar et al. modelled the continuous variables with the available data type of integers whereas Bukhari et al. incorporated rational numbers for modeling continuous variables. We use the true *real* numbers to precisely model the continuous variables which increases the completeness of the analysis and helps in tackling some corner cases which would otherwise be missed.
3. A critical limitation of model checking based analysis of DDTM schemes is that the analysis could not be carried out for larger than 9x9 grids, thus limiting its application for many-core systems, which typically involve hundreds of cores [48]. Leveraging upon the induction based verification, our approach allows any number of cores since the number of cores in our formalization and the properties are universally quantified. Scalability is one of the major benefits of our work.
4. A major shortcoming of model checking based analysis is that it is done for limited temperature ranges (86°C at most) whereas the temperatures for multi-core chips can go as high as 124°C [33]. Our strategy allows the analysis of a wide range of many-core systems over any desired range of temperature.
5. The analysis of model checking considers only one task assigned to a core, whereas our model allows any number of tasks assigned to a core, i.e., any desired queue size.
6. Number of power units assigned to a task in the model proposed by Bukhari et al. was in the range 1-4 whereas our model allows any number of power units assigned to a task. This allows us to consider heavy loaded tasks as well.
7. Up to 324 tasks have been analyzed by using model checkers while many-core systems have to deal with large number of tasks. Out strategy facilitates the analysis for any number of tasks.
8. The advantages of our methodology come at the cost of extensive user-interaction for the verification of properties while model checking benefits from the automated proofs of the properties. Thus, model checking

Table 6: Analysis in nuXmv model checker

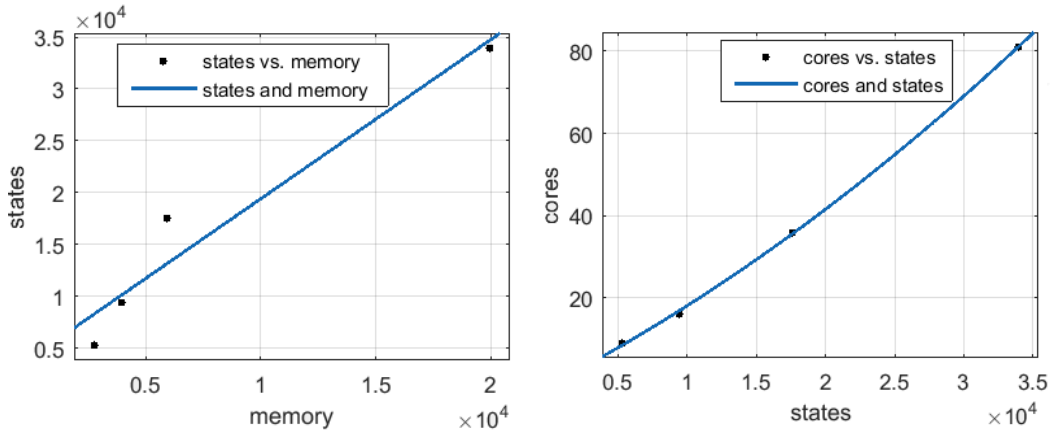
S. No.	Number of cores	State transitions	RAM (MB)
1	9	5286	1103.6
2	16	9452	1402.91
3	36	17582	2102.83
4	81	33945	2875.6

based approaches in the related work take up to 100 man hours for the analysis, whereas our approach required 600 man hours.

In order to clarify the advantage of our strategy, we carried out the model checking based analysis of number of state transitions for the same case study (Distributed task migration by Liu et al.) that we considered. This case study was considered by Bukhari et al. using the nuXmv model checker. Table 6 shows the results of the analysis carried out on a high end machine CentOS 5.0 OS running on a Intel Xeon processor E5-2407 v2 (2.40 GHz, 4 CPUs) with 32 GB of RAM. We have considered only 4 test values because significant modifications are required to vary the number of cores in the nuXmv model and then significant amount of time is required for verification to complete. For estimation purposes, we used a polynomial curve fitting tool to derive relationships of number of states, cores and memory (RAM). Fig. 3a depicts the estimated linear relationship of the number of state transitions to verify the property and memory (RAM in MB) required for verification. Similarly, the relationship of number of cores in the system and number of state transitions to verify the property is estimated with the second degree polynomial, as depicted in Fig. 3b. Using this estimation, we found that a maximum of 12x12 grid can be analyzed on the high end machine with the mentioned resources. It is worth mentioning that this is valid only for the abstracted model of distributed task migration presented in [19]. For a fine-grained model, the number of cores that can be analyzed will decrease alarmingly. Therefore, this limits the model checking based analysis of DDTM schemes for many-core systems with hundreds and thousands of cores [48]. In contrast, our strategy provides support for any number of cores, and is thus suitable for many-core systems.

Besides the above-mentioned limitations, the following properties cannot be handled by model checkers:

1. Since model checkers only support state-based models so the verifica-



(a) States vs. Memory (b) Cores vs. States

Figure 3: Relation of states, cores and memory

tion of properties with for all quantifiers on the variables of temperature and number of cores is not possible. Such universally quantified properties are important for validating the functionality of a DDTM scheme under consideration, since otherwise the scheme is verified only for specific number of cores and may result in unwanted scenarios for greater number of cores. On the other hand, the theorem proving approach can be used to verify expression for any number of cores, as depicted by the universal quantification on the number of cores in the properties verified in Section 5.3.

2. Both SPIN and nuXmv do not support the verification of probabilistic properties. The PRISM model checker supports the verification of probabilistic properties for Markovian models. On the other hand, the HOL4 theorem prover contains the theories of measure and probability [23] and Markov chains [49], and thus supports reasoning about a wider range of probabilistic properties of the system under verification.

Our proposed method is not meant for replacing all other existing analysis methods. Rather it has to work together and complement other existing methods in the following ways:

1. Simulation and emulation are incomplete with respect to error detection, as explained in Section 1.2. So our proposed method of theorem proving can be used for early diagnosis of errors and thus to ensure high

standards of accuracy. However, the benefit of simulation and emulation is that they work on the real system whereas theorem proving works on a formal model of the system.

2. Model checking can usually handle only the discretized and abstracted models of DDTM schemes, due to its inherent state-space explosion problem. Moreover, it lacks scalability for higher number of cores, as explained in Section 1.3. Compared to model checking, our proposed method of theorem proving can provide generic results for all cores. Moreover, fine-grained thermal modeling is possible because of high expressiveness of higher-order logic. On the other hand, model checking can provide automatic verification and also quantitative information in the case of probabilistic model checking. Moreover, theorem proving based verification for a specific value of a variable is usually very cumbersome as the induction can only work for generic values. Similarly, certain properties, like deadlock, are more difficult to check in theorem proving. Furthermore, theorem proving demands significant manual involvement. Thus, both these formal verification techniques have to play together to leverage upon their strengths, as proposed in [50], to obtain the desired accuracy levels while minimizing user interaction.

6. Conclusions

The accuracy of the analysis results for DDTM schemes is a dire need given their extensive usage in SoCs, which in turn have many safety and financial-critical application domains, like medicine, transportation and stock exchange markets. In this regard, thermal issues are among the foremost ones. Simulation, emulation and model checking have limitations in certifying 100% completeness. To ensure high standards of reliability and soundness, we propose a higher-order-logic theorem proving based methodology to verify any DDTM scheme for many-core systems. The approach is scalable and has the ability to deal with continuous variables. The practical effectiveness of the proposed methodology is illustrated by presenting a formal model and verification of a distributed task migration algorithm.

The proposed work opens the doors to many new directions of research. An interesting future direction is to leverage upon the high expressiveness of higher-order-logic and utilize calculus and differential theoretic reasoning to extend distributed task migration algorithm by incorporating estimated

average temperature using the signal tracking algorithm [47]. Another important direction for future work is to verify some probabilistic properties by utilizing the higher-order-logic formalization of probability theory [23]. Moreover, we are also targeting model and verification of some larger case studies, such as the TAPE algorithm [17] which involves a number of weight parameters that make the verification problem quite challenging.

References

[1] Y. Ge, Q. Qiu, Q. Wu, A multi-agent framework for thermal aware task migration in many-core systems, IEEE Transactions on Very Large Scale Integration Systems 20 (10) (2012) 1758–1771. doi:10.1109/TVLSI.2011.2162348.

[2] J. Srinivasan, S. V. Adve, P. Bose, J. A. Rivers, Lifetime reliability: Toward an architectural solution, IEEE Micro 25 (3) (2005) 70–80. doi:10.1109/MM.2005.54.

[3] M. Pedram, S. Nazarian, Thermal modeling, analysis, and management in VLSI circuits: Principles and methods, IEEE 94 (8) (2006) 1487–1501. doi:10.1109/JPROC.2006.879797.

[4] K. Skadron, M. R. Stan, K. Sankaranarayanan, W. Huang, S. Velusamy, D. Tarjan, Temperature-aware microarchitecture: Modeling and implementation, ACM Transactions on Architecture and Code Optimization 1 (1) (2004) 94–125. doi:10.1145/980152.980157.

[5] V. Hanumaiah, S. Vrudhula, Temperature-aware DVFS for hard real-time applications on multicore processors, IEEE Transactions on Computers 61 (10) (2012) 1484–1494. doi:10.1109/TC.2011.156.

[6] J. Donald, M. Martonosi, Techniques for multicore thermal management: Classification and new exploration, ACM SIGARCH Computer Architecture News 34 (2) (2006) 78–88. doi:10.1145/1150019.1136493.

[7] D. N. Truong, W. H. Cheng, T. Mohsenin, Z. Yu, A. T. Jacobson, G. Landge, M. J. Meeuwsen, C. Watnik, A. T. Tran, Z. Xiao, et al., A 167-processor computational platform in 65 nm CMOS, IEEE Journal of Solid-State Circuits 44 (4) (2009) 1130–1144. doi:10.1109/JSSC.2009.2013772.

[8] P. Chaparro, J. Gonzalez, G. Magklis, C. Qiong, A. Gonzalez, Understanding the thermal implications of multi-core architectures, IEEE Transactions on Parallel and Distributed Systems 18 (8) (2007) 1055–1065. doi:10.1109/TPDS.2007.1092.

[9] Z. Liu, S. X.-D. Tan, X. Huang, H. Wang, Task migrations for distributed thermal management considering transient effects, IEEE Transactions on Very Large Scale Integration Systems 23 (2) (2015) 397–401. doi:10.1109/TVLSI.2014.2309331.

[10] A. Kumar, L. Shang, L.-S. Peh, N. K. Jha, System-level dynamic thermal management for high-performance microprocessors, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 27 (1) (2008) 96–108. doi:10.1109/TCAD.2007.907062.

[11] M. Kadin, S. Reda, A. Uht, Central vs. distributed dynamic thermal management for multi-core processors: Which one is better?, in: Great Lakes Symposium on Very Large Scale Integration, ACM, 2009, pp. 137–140. doi:10.1145/1531542.1531577.

[12] M. Al Faruque, J. Jahn, T. Ebi, J. Henkel, Runtime thermal management using software agents for multi-and many-core architectures, IEEE Design and Test of Computers 27 (6) (2010) 58–68. doi:10.1109/MDT.2010.94.

[13] ITRS, International Technology Roadmap for Semiconductors (2013). URL http://www.itrs.net

[14] S. Borkar, Thousand core chips: A technology perspective, in: Design Automation Conference, ACM, 2007, pp. 746–749. doi:10.1145/1278480.1278667.

[15] M. Shafique, J. Henkel, Agent-based distributed power management for kilo-core processors, in: Computer-Aided Design, IEEE, 2013, pp. 153–160. doi:10.1109/ICCAD.2013.6691112.

[16] A. Fedeli, F. Fummi, G. Pravadelli, Properties incompleteness evaluation by functional verification, IEEE Transactions on Computers 56 (4) (2007) 528–544. doi:10.1109/TC.2007.1012.

[17] T. Ebi, M. A. Al Faruque, J. Henkel, TAPE: Thermal-aware agent-based power economy multi/many-core architectures, in: Computer-Aided Design, IEEE, 2009, pp. 302–309. doi:10.1145/1687399.1687457.

[18] M. Ismail, O. Hasan, T. Ebi, M. Shafique, J. Henkel, Formal verification of distributed dynamic thermal management, in: Computer-Aided Design, IEEE, 2013, pp. 248–255. doi:10.1109/ICCAD.2013.6691126.

[19] S. A. A. Bukhari, F. K. Lodhi, O. Hasan, M. Shafique, J. Henkel, Formal verification of distributed task migration for thermal management in on-chip multi-core systems using nuXmv, in: Formal Techniques for Safety-Critical Systems, Vol. 476, Springer, 2014, pp. 32–46. doi:10.1007/978-3-319-17581-2_3.

[20] S. Iqtedar, O. Hasan, M. Shafique, J. Henkel, Formal probabilistic analysis of distributed dynamic thermal management, in: Design, Automation and Test in Europe, IEEE, 2015, pp. 1221–1224.

[21] S. Iqtedar, O. Hasan, M. Shafique, J. Henkel, Probabilistic formal verification methodology for decentralized thermal management in on-chip systems, in: Enabling Technologies: Infrastructures for Collaborative Enterprises, IEEE, 2015, pp. 210–215. doi:10.1109/WETICE.2015.39.

[22] C. Baier, J.-P. Katoen, Principles of model checking, MIT Press, 2008.

[23] O. Hasan, S. Tahar, Formalized probability theory and applications using theorem proving, IGI Global, 2015.

[24] D. Dunn, Intel delays Montecito in roadmap shakeup, EE Times, Manufacturing/Packaging.

[25] J. Harrison, Handbook of practical logic and automated reasoning, Cambridge University Press, 2009.

[26] K. Slind, M. Norrish, A brief overview of HOL4, in: Theorem Proving in Higher Order Logics, Vol. 5170 of Lecture Notes in Computer Science, Springer, 2008, pp. 28–32. doi:10.1007/978-3-540-71067-7_6.

[27] Z. Liu, X. Huang, S.-D. Tan, H. Wang, H. Tang, Distributed task migration for thermal hot spot reduction in many-core microprocessors, in: ASIC, IEEE, 2013, pp. 1–4. doi:10.1109/ASICON.2013.6811821.

[28] R. Alur, T. A. Henzinger, P.-H. Ho, Automatic symbolic verification of embedded systems, IEEE Transactions on Software Engineering 22 (3) (1996) 181–201. doi:10.1109/32.489079.

[29] S. K. Shukla, D. J. Rosenkrantz, S. S. Ravi, Simulation and validation tool for self-stabilizing protocols, in: SPIN Workshop, Volume 32 of DIMACS Series in Discrete Mathematics and Theoretical Computer Science, 1997.

[30] T. Tsuchiya, S. Nagano, R. B. Paidi, T. Kikuno, Symbolic model checking for self-stabilizing algorithms, IEEE Transactions on Parallel and Distributed Systems 12 (1) (2001) 81–95. doi:10.1109/71.899941.

[31] J. Eberhard, A. Tripathi, Semantics-based object caching in distributed systems, IEEE Transactions on Parallel and Distributed Systems 21 (12) (2010) 1750–1764. doi:10.1109/TPDS.2010.48.

[32] G. J. Holzmann, The model checker SPIN, IEEE Transactions on Software Engineering 23 (5) (1997) 279–295. doi:10.1109/32.588521.

[33] E. Glocker, D. Schmitt-Landsiedel, Modeling of temperature scenarios in a multicore processor system, Advances in Radio Science 11 (2013) 219–225. doi:10.5194/ars-11-219-2013.

[34] M. J. C. Gordon, T. F. Melham, Introduction to HOL: A theorem proving environment for higher order logic, Cambridge University Press, 1993.

[35] M. J. C. Gordon, Mechanizing programming logics in higher order logic, in: Current Trends in Hardware Verification and Automated Theorem Proving, Springer, 1989, pp. 387–439. doi:10.1007/978-1-4612-3658-0_10.

[36] C. E. Brown, Automated Reasoning in Higher-order Logic: Set Comprehension and Extensionality in Church's Type Theory, College Publications, 2007.

[37] T. F. Melham, Higher order logic and hardware verification, Vol. 31, Cambridge University Press, 2009.

[38] HOL4, http://hol-theorem-prover.org/ (2015).

[39] J. Jahn, M. A. A. Faruque, J. Henkel, Carat: Context-aware runtime adaptive task migration for multi core architectures, in: Design, Automation and Test in Europe, IEEE, 2011, pp. 1–6. doi:10.1109/DATE.2011.5763093.

[40] M. U. Sardar, Formal verification of distributed dynamic thermal management schemes using HOL4 - HOL proof script (2015).
URL http://save.seecs.nust.edu.pk/projects/fDTM.html

[41] R. Cardell-Oliver, The formal verification of hard real-time systems, Ph.D. thesis, Citeseer (1992).

[42] O. Hasan, S. Tahar, Performance analysis and functional verification of the stop-and-wait protocol in HOL, Journal of Automated Reasoning 42 (1) (2009) 1–33. doi:10.1007/s10817-008-9105-6.

[43] F. J. Mesa-Martinez, E. K. Ardestani, J. Renau, Characterizing processor thermal behavior, ACM SIGPLAN Notices 45 (3) (2010) 193–204. doi:10.1145/1735971.1736043.

[44] K. Kang, J. Kim, S. Yoo, C.-M. Kyung, Runtime power management of 3-D multi-core architectures under peak power and temperature constraints, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 30 (6) (2011) 905–918. doi:10.1109/TCAD.2010.2101371.

[45] E. Kursun, C.-Y. Cher, Temperature variation characterization and thermal management of multicore architectures, IEEE Micro 29 (1) (2009) 116–126. doi:10.1109/MM.2009.18.

[46] J. Kong, S. W. Chung, K. Skadron, Recent thermal management techniques for microprocessors, ACM Computing Surveys 44 (3) (2012) 13:1–13:42. doi:10.1145/2187671.2187675.

[47] F. Chen, Y. Cao, W. Ren, Distributed average tracking of multiple time-varying reference signals with bounded derivatives, IEEE Transactions on Automatic Control 57 (12) (2012) 3169–3174. doi:10.1109/TAC.2012.2199176.

[48] A. Pathania, V. Venkataramani, M. Shafique, T. Mitra, J. Henkel, Distributed fair scheduling for many-cores, in: Design, Automation and Test in Europe Conference, 2016.

[49] L. Liu, O. Hasan, S. Tahar, Formal reasoning about finite-state discrete-time markov chains in HOL, Journal of Computer Science and Technology 28 (2) (2013) 217–231. doi:10.1007/s11390-013-1324-6.

[50] S. Ray, R. Sumners, Combining theorem proving with model checking through predicate abstraction, IEEE Design Test of Computers 24 (2) (2007) 132–139. doi:10.1109/MDT.2007.38.