# Towards Model Checking-Driven Fair Comparison of Dynamic Thermal Management Techniques under Multi-Threaded Workloads

Syed Ali Asadullah Bukhari, Faiq Khalid, *Student Member, IEEE,* Osman Hasan, *Senior Member, IEEE,* Muhammad Shafique, *Senior Member, IEEE,* and Jörg Henkel, *Fellow, IEEE.*

***Abstract*—Dynamic thermal management (DTM) techniques are being widely used for attenuation of thermal hot spots in many-core systems. Conventionally, DTM techniques are analyzed using simulation and emulation methods, which are in-exhaustive due to their inherent limitations and cannot provide for a comprehensive comparison between DTM techniques owing to the wide range of corresponding design parameters. In order to handle the above discrepancies, we propose to use model checking, a state-space based formal method, to model, evaluate and compare DTM techniques across various functional and performance parameters. The suggested framework includes a modeling flow and a set of generic modules that realistically model many-core and DTM parameters like temperature, power, application, inter-core communication and task migration etc. For analysis purpose, the framework provides a common ground for comparing DTM techniques by formalizing DTM principles and performance parameters as a set of logical properties. These properties are verified for different task load configurations, e.g., multi-threaded, malleable, and the applications which do not support migration. We analyze state-of-the-art central (c-) and distributed (d-) DTM techniques to demonstrate the generality and efficacy of our approach. Our formal analysis shows that the state-of-the-art cDTM technique performs better than dDTM in terms of achieving thermal stability, task migration and communication overhead. We believe that conventional analysis methods do not facilitate such an exhaustive comparison among the DTM techniques.**

***Index Terms*—Model Checking, Formal Analysis, centralized, distributed, Dynamic Thermal Management, Task Migration, Many-Core, nuXmv, Formal Verification.**

## I. Introduction

DUE to higher transistor density and the resulting up-scaling of multi-cores to many-cores, thermal emergencies appear as bottle-neck while achieving the desired processing performance [1], [2]. An improper engagement of these issues may lead to reliability concerns, malfunctioning and even physically damaging the chips. DTM techniques have been suggested, both at hardware and software level to mitigate this problem [3]–[5]. The hardware DTM methods, like Dynamic Voltage and Frequency Scaling (DVFS), mitigate the heating problem by throttling the operating frequency and voltage [6], [7]. Although DVFS results in cooling of a heated chip, yet, this solution compromises the performance [4], [7], since the processor is run at a lower frequency to consume lesser power and consequently generate less heat. An alternate method is to employ strategies to manage processing load across the chip to prevent the over-heating. In contrast to DVFS, this dynamic redistribution of tasks among the cores results in a performance efficient solution, as the physical parameters remain unchanged [8]–[10].

A DTM strategy based on task migration is executed by DTM controller(s) or agent(s) running on core(s) in a many-core chip [11]. A DTM technique is considered as global or centralized (c-), if a global DTM controller is implemented on one of the cores in the system [10], [12]. This centralized controller communicates with all the cores in the system and collects parameters like temperature, core utilization etc. to take a task migration decision. However, with a large number of cores, the communication and computation overhead of the centralized controller becomes the bottleneck as it becomes quite difficult to manage the temperature of the centralized controller itself. Moreover, a centralized controller is a single point of failure. In a distributed (d-) configuration [13], [8], [14], each core in the system has a DTM controller running, that communicates and share parameters with the other neighboring DTM controllers only. Since the control is no longer centralized in the dDTM approach, therefore, the issues of scalability and excessive communication overhead with the central controller are resolved.

In each of the above configurations, DTM agents moderate the chip temperature by invoking a *proactive* [12], [15], or *reactive* [8], [16] task migration algorithm. In former, the agents develop a model of the overall thermal behavior of the chip and use it along with various prediction techniques to make management decisions before a temperature threshold is violated. Whereas, in reactive DTMs, the agents initiate the task migration once the threshold is reached for a certain core.

Based on the above-mentioned configurations, a variety of DTM schemes have been proposed. Apart from the central or distributed nature of the DTM controllers, these techniques make use of various parameters, like effective chip temperatures, transient effects, task loads and thermal estimation methods, to make their DTM decisions. *This variety of parameters complicates the task for comparing different DTM techniques and identifying the most effective one in terms of performance for a given scenario becomes very challenging.* Mostly, DTM techniques are analyzed using simulation and emulation methods [17], [18]. However, due to the sampling based nature of these methods and inherent inability to test all the possible scenarios and due to the extensive computational cost, these methods do not guarantee a complete and exhaustive analysis. In the recent years, the use of formal methods has been strongly advocated for analyzing DTM techniques to overcome the drawbacks in the traditional analysis methods [19]–[22]. A detailed review of DTM analysis using formal methods is provided in Section II.

S. A. A. Bukhari, F. Khalid and O. Hasan are with School of Electrical Engineering and Computer Science, National University of Sciences and Technology, Islamabad, Pakistan. (e-mail: ali.asadullah, faiq.khalid, osman.hasan@seecs.nust.edu.pk).

M. Shafique is with Institute of Computer Engineering, Vienna University of Technology, Vienna, Austria. (e-mail: muhammad.shafique@tuwien.ac.at).

J. Henkel is with Chair of Embedded Systems, Karlsruhe Institute of Technology, Karlsruhe, Germany. (e-mail: henkel@kit.edu).

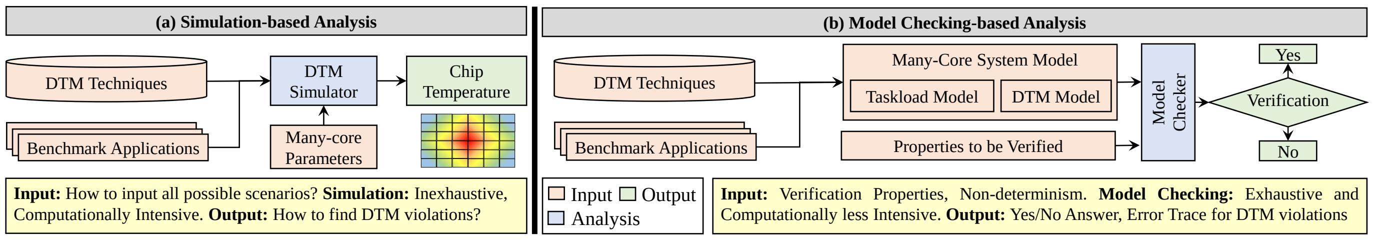
(a) Simulation-based Analysis
DTM Techniques
Benchmark Applications
DTM Simulator
Many-core Parameters
Chip Temperature
Input: How to input all possible scenarios? Simulation: Inexhaustive, Computationally Intensive. Output: How to find DTM violations?
(b) Model Checking-based Analysis
DTM Techniques
Benchmark Applications
Many-Core System Model
Taskload Model
DTM Model
Properties to be Verified
Model Checker
Verification
Yes
No
Input
Output
Analysis
Input: Verification Properties, Non-determinism. Model Checking: Exhaustive and Computationally less Intensive. Output: Yes/No Answer, Error Trace for DTM violations

*Figure 1:* A comparison between simulation and model checking -based formal analysis of DTM schemes.

In this paper, we present the analysis of DTM techniques for multi-threaded workloads using model checking, which is a state-space based formal verification technique [23]. Model checking allows us to cover all the possible scenarios for a DTM technique by exhaustive state-space exploration, *that is virtually impossible to achieve through simulation or emulation*, especially for larger grid sizes.

## A. Motivation

In the following, we give the motivation for the current work and aver the need of the model checking -based formal analysis of DTM techniques by highlighting the deficiencies in the simulation-based analysis, as shown in Figure 1.

1) **DTM Modeling and Analysis Coverage:** Based on testing methods, it is practically impossible to model and test all the possible scenarios for a DTM scheme or any given system, thus limiting the resulting analysis to a set of benchmarks or parameters. This problem can be alleviated using model checking, as it allows non-deterministic modeling of system's parameters, and the verification is based on exhaustive exploration of the system model against a mathematical property, rather than a few benchmarks vectors. Also, the state-space based nature of model checking favors the modeling of DTM schemes, which are usually represented in a FSM or algorithm [10], [13], and thus can be easily translated to the corresponding modeling language. In contrast, other formal methods like theorem proving etc. require the system to be mathematically expressed as logical functions, which can be more effort requiring in case of DTM techniques that are mostly expressed in a sequential manner.

2) **Property-based Analysis:** The primary requirement of any DTM scheme is to ensure thermal stability of the many-core system, i.e., the deployment of the DTM ensures a safe temperature profile of the chip. However, using simulation and emulation, it is not possible to explicitly define and verify this behavior of DTM. On the other hand, using model checking allows us to mathematically define this desired behavior using temporal logic and then verify it over the system model. In case of DTM verification, this feature is helpful in formalizing different scenarios during DTM execution, e.g., threshold avoidance, thermal balancing etc., as LTL properties. On the other hand, in other formal techniques, like theorem proving, a notion of time or sequence has to be explicitly modeled to allow expression of such verification properties, which results in more modeling effort.

3) **Analysis Time:** The DTM simulation environments involve processor simulators like Sniper [24] and Gem5 [25], etc. for evaluating benchmarks against different DTM schemes. However, the time required for these simulations usually ranges from several hours to a few days [26], whereas, the exhaustive model checking of a DTM model against some desired property can be generally performed in order of minutes to a few hours [20], [27]. Moreover, the automatic verification feature makes model checking an attractive choice for the formal verification of DTM techniques, as other formal methods, e.g., theorem proving with expressive logic, like higher-order logic, require manual effort (and usually more time) to verify a systems property.

4) **Results and Debugging:** The basic output from a DTM simulator environment is the thermal profile of the chip against the execution of the input benchmarks. A conclusion about the DTM scheme has to be drawn by observing and analysing the chip temperature. This process can be simplified using model checking as the output from the tool is a simple yes or no answer for the DTM verification. Moreover, an error trace is provided by the tool in case the DTM scheme violates the desired property. This feature facilitates the DTM designer in identification of bugs at an early stage. In contrast, this facility is not available in other formal methods -based analysis.

## B. Scientific Challenges Targeted in this Paper

A model checking -based analysis of DTM techniques provides a complementary approach to simulation to lead to a reliable working of many-core systems. However, the modeling of DTM techniques and many-core systems, and the corresponding verification problem, poses certain challenges, which are as follows:

1) **Modeling thermal behavior of many-core system**
The most important aspect of modeling a many-core system is to imitate the thermal behavior of the cores. A formal thermal model should capture the necessary physical properties of the cores, as well as, it should be computationally feasible within the available resources. In order to meet both of these requirements, we have a thermal model, given in [28], that includes parameters, like inter-core resistance and capacitance.

2) **Benchmarks' modeling and completeness of analysis**
Traditional DTM analysis report the experimentation results, by giving the cores temperatures against the progress of workloads under test. These workloads mostly consist of predefined standard benchmarks, which present a typical input scenario only, and are executed on the simulator to observe the cores' temperatures. In order to ensure the completeness of our analysis, we have modeled the workloads non-deterministically to allow all possible input workload requirements.

3) **Choice of verification properties**
The extraction and formalization of DTM properties becomes a challenging and important part of verification, since, only an informal description of DTM system and results applicable

to a limited scenario are provided in the literature. We have resolved this issue by formalizing a set of DTM properties equally applicable across different DTM techniques.

4) **Handling state-space explosion issues**

The refinement of the DTM model with precise details (or any other system model for that matter) leads to the infamous problem of state-space explosion [23] during model checking, wherein, the overall system model gets too big to be verified with limited computational resources. We have tried to handle this issue by using appropriate modeling abstractions that minimizes the state-space of the model while capturing the desired system's behavior.

## C. Utility and novel contributions of this work

We elaborate the utility of our approach by integrating it within the design space exploration of DTM techniques. As shown in Figure 2, our framework can be employed at design level and has the following main utilities:

1) To allow the DTM designer to explore the design at early stages and report any bugs, that can then be further inspected using simulation and removed.
2) To help choosing a DTM technique under the given constraints. This is possible by the virtue of formalization of DTM properties that provide a common analysis ground.

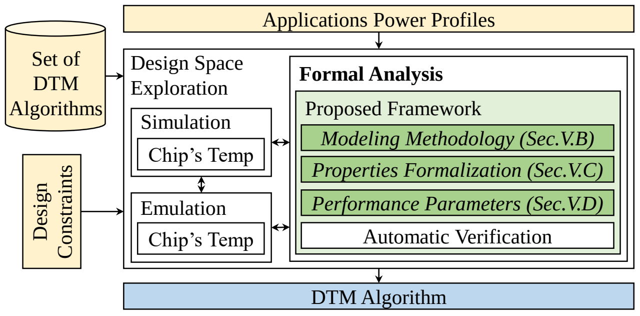

*Figure 2:* Our framework (highlighted green) integrated into design state exploration of DTM schemes.

The distinguishing features of this work are:

1) **Modeling Methodology (Section V)** The proposed framework provides a step-by-step modeling methodology for many-core system and DTM techniques. As compared to earlier approaches [22], [27], this work has the following novel modeling contributions:
   a) *Provision of modeling libraries.* Our framework provides different formal modules or libraries that can be used to model various configurations of DTM techniques and many-core systems.
   b) *Enhanced modeling.* As compared to the existing works, the following new models are included:
      i) *Thermal Model.* A realistic core-temperature model, taken from [28], that includes physical parameters, like inter-core resistance and capacitance, whereas, in earlier works, a simple linear relation of temperature with the task load is assumed.
      ii) *Inter-core communication model.* We have compared DTM techniques by considering the inter-core communication costs.
      iii) *Multi-threaded applications.* As compared to existing works, that assume one thread per application, our approach allows evaluation of DTMs for multi-threaded applications.
      iv) *Different application configurations.* We have modeled applications with and without migration support, allowing us to compare DTM techniques with a no-task migration policy.
2) **Properties Formalization (Section V-D)** We have formalized DTM properties, that express the overall system's behavior and are equally applicable across different DTM techniques. Specifically, this work defines the LTL notation and explanation for the following properties: a) Threshold Avoidance b) Thermal Balancing c) Thermal Safety.
3) **Performance Parameters (Section V-E)** The framework includes a set of performance properties, based on task migration principles, to provide a common ground for analyzing the performance of cDTM and dDTM techniques. In particular, to the best of our knowledge, this work introduces the formal analysis based on the inter-core communication cost for the first time.

To demonstrate the generality and usefulness of the suggested framework, we have conducted a comparison between state-of-the-art central [10] and distributed [8], [13] DTM techniques, up to a $12 \times 12$ many-core system.

**Open-source Contributions:** We have released the open-source code for our proposed methodology and the models for analyzed DTM schemes [29]. It will serve the purpose to encourage application of formal tools in DTM domain and reproduce the verification results.

## II. Related Work

In [19], the SPIN model checker [30] has been used to analyze a dDTM technique, i.e., Thermal-aware Agent-based Power Economy (TAPE) [16]. The functional correctness of TAPE was verified by defining stability condition for the task migration algorithm, and the timing properties were found by integrating Lamport timestamps [31] in the formal model of TAPE. The formal analysis helped to identify a couple of issues, that were overlooked in the simulation based analysis of the same algorithm. However, this analysis was done for a $3\times3$ core grid and continuous parameters, like temperature, were abstracted using a small set of integer values. These parameters could not be relaxed, i.e., the number of cores could not be increased or a larger set of values could not be used, due to state-space explosion. It is note-worthy that such abstractions compromise the exhaustiveness of the analysis.

In [20], some of the above-mentioned issues were alleviated by using the nuXmv model checker for analyzing a dDTM technique [8]. The analysis was done for up to a $9 \times 9$ core grid by leveraging upon the efficient state-exploration algorithms provided in nuXmv. However, the work load was abstracted in [20] by assuming a simple model that consumed a power unit per unit time. The core temperature was also abstracted by assuming it to be directly proportional to the task load. The stability criterion was defined as the state when the temperatures of all the cores in a many-core system are less than the threshold temperature for triggering the task migration. The DTM technique [8] was verified for this stability condition and the number of state transitions required to achieve the stability were also calculated.

The probabilistic model checker, PRISM [32], has also been used for the formal analysis of TAPE, by modeling its behavior as a Markov chain [33]. The analysis was done up to a grid size of $4 \times 4$ and the probability for the maximum chip temperature to be less than the threshold temperature was calculated. The analysis showed that the performance of TAPE depended highly on the weight parameters used in the technique. The verification of TAPE for a higher grid size of $9 \times 9$ was done using approximate probabilistic model checking in [21]. The results showed that TAPE achieved thermal stability but did not perform the temperature balancing across the chip.

A Formal Analysis Methodology for Task Migrations (FAMe-TM), was presented in [22], for the verification of task migration based dDTM techniques. The paper extended the results, presented in [20], by introducing a generic methodology applicable to any dDTM technique. In addition to the number of transitions required for stability, new performance properties, like number of task migrations, task load completed, number of stalled tasks, number of hot spots and time spent in task migration were introduced. The task load and temperature model used were similar to that of [20]. Using FAMe-TM, two dDTM techniques [8], [34] were formally modeled and analyzed and compared up to $9 \times 9$ grid size.

In [35], a theorem proving -based framework for the analysis of dDTM was proposed to alleviate the scalability issues in verification. This work mathematically modeled the dDTM technique [8] and proved certain DTM properties by defining them as proof goals in the HOL theorem prover. The verified properties were presented as theorems, that were valid for a generic $\mathrm{n} \times \mathrm{n}$ grid size, thus overcoming the scalability issues reported by model checking. However, since the DTM technique was modeled using higher-order-logic, the verification was not automatic and required user interaction to prove the desired properties or goals. Due to the enormous user-guidance required, the properties verified in [35] were very basic ones, like the no-migration conditions and temperature variation bounds per step, and thus the insights provided by the above-mentioned model checking based approaches can be considered more useful.

A generic comparative analysis methodology, i.e., CAnDy-TM, applicable to both cDTM and dDTM techniques, was proposed in [27]. The stability criteria was improved in CAnDy-TM by formalizing different stability conditions like, threshold avoidance, thermal balancing and safety. Inter-core communication overhead was also considered in the evaluation and an equation from Intel processor [36] was used to model the thermal behavior of the cores. Using CAnDy-TM, a comparison between state-of-the-art cDTM [10] and dDTM [8] techniques was presented.

**Limitations of state-of-the-art:** It is important to note that in all of the above-mentioned works, the thermal behavior of the cores has been modeled by assuming rather simple temperature models and is related to task load in a linear fashion. Also, the task loads running on the cores are assigned a linear power consumption with respect to the task duration. This assumption oversimplifies the real work loads or benchmarks that are run during the simulation based evaluation of DTM techniques. Although these assumptions help to reduce the state-space of the DTM model, yet, the fine and practical details, close to a real physical many-core system, cannot be ensured. Moreover, a single thread per application is assumed as opposed to multi-threaded applications, which are more suitable for many-core systems. Thus, the behavior of DTM techniques with different multi-threaded applications cannot be analyzed. In the current work, we propose to alleviate these issues by formalizing the thermal and multi-threaded application models for many-core systems, that can be applied to different DTM configurations. Also, the proposed models can be easily integrated into the existing analysis methodologies, i.e., FAMe-TM [22] and CAnDy-TM [27].

## III. Preliminaries

In this section, we give an overview of the nuXmv model checker and briefly describe the selected DTM algorithms [8], [10], [13] to be analyzed.

### A. *nuXmv Model Checker*

nuXmv [37] is a symbolic model checker that supports new data types of *Integers* and *Reals* along with the availability of efficient SAT algorithms and advanced Satisfiability Modulo Theories (SMT) [38], based on MathSAT [39]. The main steps in verification of a system using nuXmv are:

1) Expressing the model for the given system using nuXmv language. This is done by creating several modules that are instantiated in the `MAIN` module of the nuXmv code.
2) Writing the properties to be verified, using the Linear Temporal Logic (LTL) or Computation Tree Logic (CTL) [23]. The LTL properties are written in nuXmv using logical operations like, AND (`&`), OR (|), XOR (`xor`), implication (`->`) etc., and temporal operators, like Globally (`G`), Finally (`F`), Next (`X`) and Until (`U`). Similarly, quantifiers like Exists (`E`) and For all (`A`) are provided for writing CTL specifications.
3) The formal model of the system and the properties are given to nuXmv for automatic verification to find out if the system meets the specifications (properties). In case a property fails, nuXmv provides the corresponding counterexample in the execution trace of the FSM, which can be used to debug the problem to see if it is an actual functional bug in the given system or a modeling issue.

### B. *Choice of nuXmv model checker*

The modeling of DTM schemes involves parameters, like temperature, power, etc. that are modeled using $integer$ or $real$ data types. The introduction of these variables (or first-order logic) in a model requires the use of SMT solvers for verification. These SMT solvers like [39]–[41] employ modulo theories, like $integer$ and $real$ etc., and a SAT solver to find the satisfiability after translating the first order predicates to boolean expressions. *We can safely say that any SMT-based model checking tool can thus be used in our approach for verification of DTM schemes.* We have chosen nuXmv model checker in our methodology due to the following reasons:

1) DTM schemes involve a number of design variables, like power, temperature, and other algorithmic parameters, that

are continuous in nature. nuXmv allows variables of type $real$, that facilitates us to model these values without discretization. The earlier analyses [19], [33] based on other model checkers, used an abstract model of DTM schemes by assigning integer values to these parameters, thus compromising on the completeness of the analysis.

2) Due to the complex nature of DTM schemes and the number of DTM parameters, the verification of DTM models faces the issue of state-space explosion. nuXmv deploys MathSAT SMT solver to efficiently verify large state-space models. Based on this, we are able to verify DTM schemes up to grid sizes of 12 $\times$ 12, as compared to 3 $\times$ 3 and 4 $\times$ 4 in [19] and [33], respectively.
3) The earlier analysis [19] involved explicit use of Lamport timestamps [31] in the formal model for verification of timing properties, however, using nuXmv's built-in support, we are able to analyze these properties without using Lamport timestamps.

Model checking faces the infamous problem of *state-space explosion* as the state-space of a model gets very large. This issue can be resolved by taking a higher abstraction of the given model or verifying the model using *Bounded Model Checking BMC* [42]. BMC allows the verification of a model by searching its state space within $k$-bound levels to find a counterexample for the given property.

### C. Multi-Objective DTM mDTM

mDTM [10] is a proactive cDTM technique that avoids thermal threshold as well as balances the cores temperatures across the chip. The technique is based on two central control units, i.e., Central decision Unit for Avoiding Threshold Temperature (CU-AT) and Central decision Unit for Achieving Thermal Balancing (CU-AB). CU-AT takes a task migration decision, based on the predicted temperature values $R_{AT}$ from each core's temperature predictor module, i.e., AT-Predictors (ATP). If $R_{AT}$ of some core $i$ is greater than or equal to a Proactive Threshold $ProT_{th}$, the task is stopped and moved to a Waiting Queue ($WQ$). The task from $WQ$ is moved to a core having temperature less than a predefined temperature $A$. If there is no task to be migrated by CU-AT and $WQ$ is empty, then the thermal balancing module, i.e., CU-AB gets activated. Similar to ATP, there is a predictor module on each core for temperature balancing, i.e,, AB-Predcitors (ABP). Using the temperature difference $e_k$ between the current core temperature $T_k$ and the average chip temperature $T_{avg}$, APB predicts the future temperature difference $R_{AB,k}$ of a core. CU-AB migrates a task from a core if $R_{AB} \geq DyBal$, where $DyBal$ is a run-time parameter that defines the balancing threshold and is given as $DyBal = Bal \times N/M$, and $M$ is the number of tasks running on a core, $N$ is the total number of cores in the system and $Bal$ is a pre-defined balancing parameter. The algorithmic flow of mDTM is shown in Figure 3 and more details can be found in [10].

### D. Hot Spots Reduction DTM rDTM

rDTM [8] performs distributed task migration with the primary goal of reducing the number of hot spots in the chip. Each core in the system runs an instance of rDTM and task migration is invoked if the average temperature of the chip

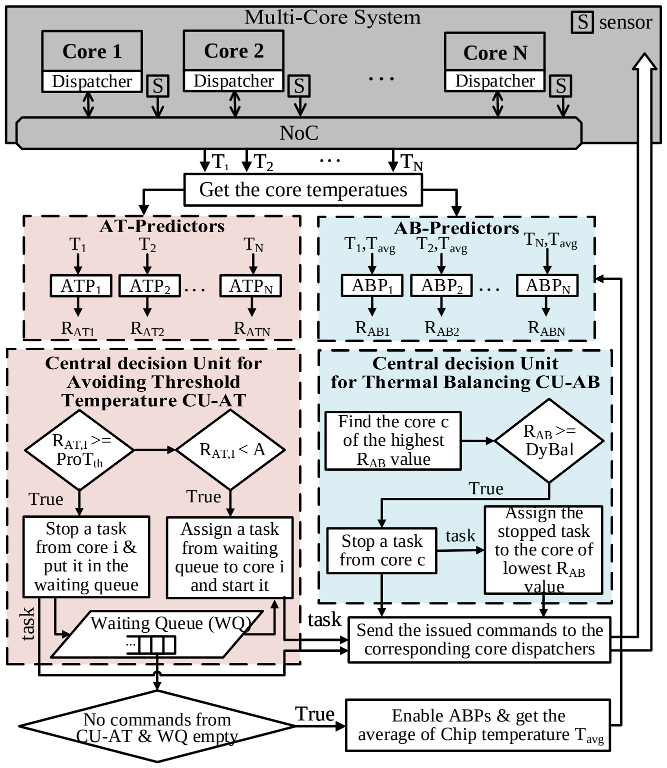
Multi-Core System
S sensor
Core 1
Dispatcher
Core 2
Dispatcher
Core N
Dispatcher
S
NoC
$T_1$ $T_2$ ... $T_N$
Get the core temperatues
AT-Predictors
$T_1$ $T_2$ $T_N$
$ATP_1$ $ATP_2$ ... $ATP_N$
$R_{AT1}$ $R_{AT2}$ $R_{ATN}$
AB-Predictors
$T_1,T_{avg}$ $T_2,T_{avg}$ $T_N,T_{avg}$
$ABP_1$ $ABP_2$ ... $ABP_N$
$R_{AB1}$ $R_{AB2}$ $R_{ABN}$
Central decision Unit for Avoiding Threshold Temperature CU-AT
$R_{AT,I}$ >= $ProT_{th}$
$R_{AT,I}$ < A
True
Stop a task from core i & put it in the waiting queue
Assign a task from waiting queue to core i and start it
task
Waiting Queue (WQ)
Central decision Unit for Thermal Balancing CU-AB
Find the core c of the highest $R_{AB}$ value
$R_{AB}$ >= DyBal
Stop a task from core c
Assign the stopped task to the core of lowest $R_{AB}$ value
Send the issued commands to the corresponding core dispatchers
No commands from CU-AT & WQ empty
Enable ABPs & get the average of Chip temperature $T_{avg}$

*Figure 3:* Overview and Operational flow of mDTM [10].

is above a certain threshold. rDTM calculates this average temperature by using a distributed average signal tracking algorithm [8], which allows the average to be estimated using temperature from neighboring cores only, without the need of global knowledge of the temperature of every core. If a core has a temperature greater than the estimated $T_{avg}$, then the task is migrated from the $current$ core to the appropriate $destination$ core among the neighbors only. The core amongst the neighbors with the maximum temperature and task load difference from the $current$ core is selected as a $destination$ core for migration. The algorithmic flow of rDTM is shown in Figure 4 and more details about rDTM can be found at [8].

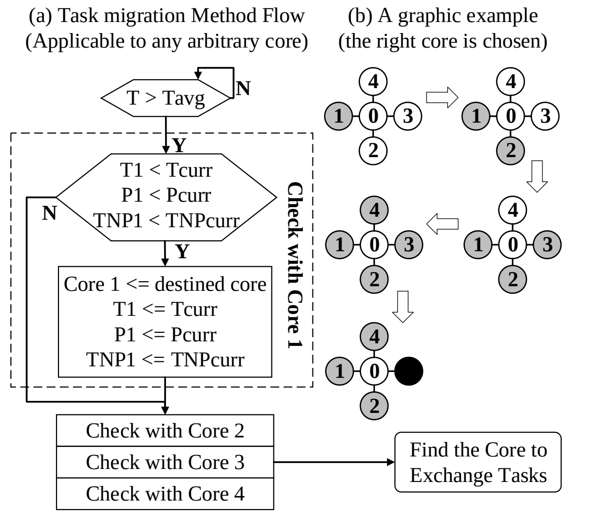

*Figure 4:* A typical execution of rDTM algorithm [8].

### E. Deadline Based Task Allocation Scheme DBTAS

DBTAS [13] aims to satisfy the task deadlines ($D_j$) while maintaining maximum temperature by allocating tasks ($T_j$)

according to earlier deadline first (EDF) algorithm at every migration interval ($t_{mg}$). The scheme finds the coolest core ($C_c$) and its surrounding area ($Z_c$), and selects it for task allocation, if 50% of the cores in $Z_c$ have their temperatures ($T_c$) less than the threshold temperature ($T_{th}$). Also, the product of rate of change of temperature $dT/dt$ and $t_{mg}$ should be equal to or less than the difference of current temperature $T_j$ and the threshold temperature $T_{th,j}$ of that particular core. Next, a minimal voltage and frequency is chosen for the cores $C_j$ in $Z_c$ and the tasks are assigned in the EDF order. If a task is missing its deadline, shown by its parameter lateness $L_{f,j}$, then DVFS is applied to that core such that it deadline can be met. In case, the application of DVFS cannot guarantee the deadline safety, an exception is raised. Figure 5 gives the flowchart for DBTAS and further details can be seen in [13].

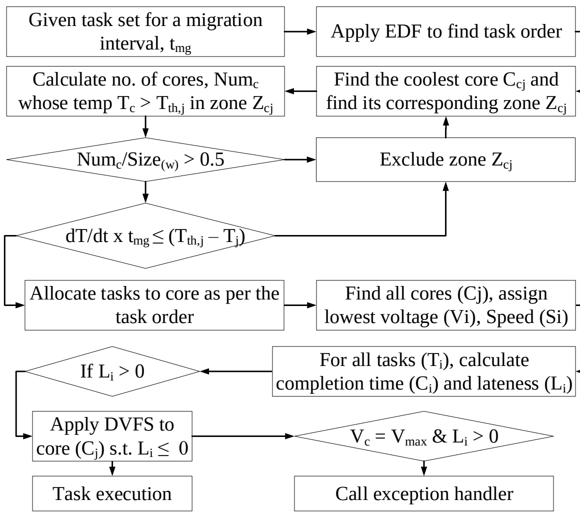
Given task set for a migration interval, $t_{mg}$
Apply EDF to find task order
Calculate no. of cores, $Num_c$ whose temp $T_c > T_{th,j}$ in zone $Z_{cj}$
Find the coolest core $C_{cj}$ and find its corresponding zone $Z_{cj}$
$Num_c/Size_{(w)} > 0.5$
Exclude zone $Z_{cj}$
$dT/dt \times t_{mg} \le (T_{th,j} - T_j)$
Allocate tasks to core as per the task order
Find all cores (Cj), assign lowest voltage (Vi), Speed (Si)
If $L_i > 0$
For all tasks ($T_i$), calculate completion time ($C_i$) and lateness ($L_i$)
Apply DVFS to core ($C_j$) s.t. $L_i \le 0$
$V_c = V_{max}$ & $L_i > 0$
Task execution
Call exception handler

*Figure 5:* Flowchart of DBTAS [13].

## IV. Proposed Methodology

The main idea behind the proposed methodology is to have a set of formal DTM and many-core parameters, that can be reused to formally model the behavior of any arbitrary DTM system. Figure 6 gives an overall flow of the proposed methodology and each of these steps is outlined below:

1) **Core Modeling.** The first step in the proposed methodology is to model the behavior of a core, that can be instantiated to develop the many-core grid. The core model is mainly developed based on its physical parameters, like `thermal`, `power`, and `task queue`, in the `core` module, which are independent of the DTM. The `thermal predictor` and the `migration policy` modules in the `core` model are inherited from the DTM being verified. The inter-core communication mode in the `comm. model` can be central or distributed and is thus also dependant on the DTM. The details of each of these modules is given in Section V.

2) **Modeling of Many-Core Grid.** The second step is to generate a many-core grid from the `core` modules and map the DTM agents to the grid. In case of cDTM, a `Central Agent` module from the library is added to one core module and the rest of the `core` modules are used to model the remaining cores of the system without a DTM agent. In case of dDTM, each `core` module is accompanied by a `Distributed Agent` module that manages thermal conditions in its neighborhood.

3) **Complete System Model.** The next step is to map the task load to the many-core grid. Using the parameter *thread*, a single- or multi- threaded tasks can be assigned to the cores in the system. Similarly, the threads can be fixed to a core using the *with migration* parameter.

4) **Formalization of DTM Properties.** After modeling the complete many-core system with DTM agents and application models, the next step is to write the verification properties for the system model. We have also formalized a set of generic functional properties for DTM techniques, using Linear Temporal Logic (LTL). These properties check the functional correctness of the given DTM scheme and verify that the many-core system indeed achieves a thermally safe or stable state with the given DTM scheme. More details about these properties can be found in Section V-D.

5) **Verification.** Both the many-core system model and the formal properties are given to the model checker for automatic verification. During the verification process, a number of performance parameters, implemented as counters in each core module, are also calculated. *The most important performance parameter for a DTM is the number of state transitions it takes to reach a thermally stable state.* More details about these properties are given in Section V. In case, the verification fails, the tool generates a counterexample which can be analyzed to revisit the modeling of the many-core system.

## V. Formal Modeling

In this section, we formalize the verification problem and provide the details for the formal modeling of our many-core

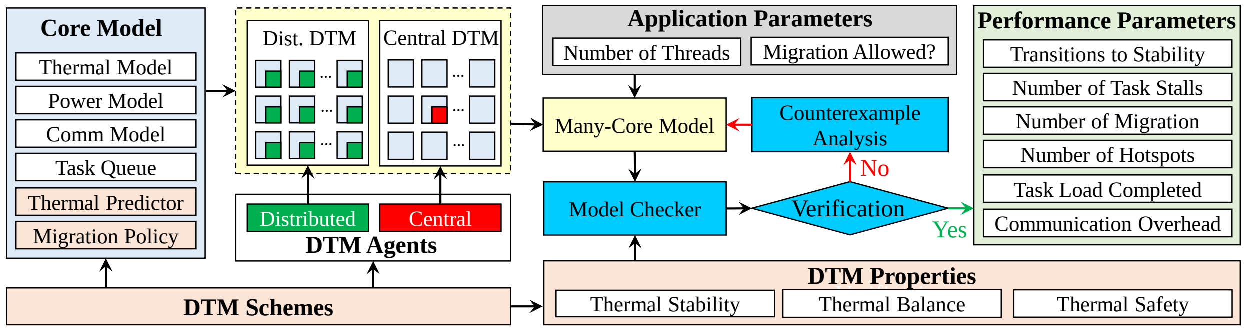

*Figure 6:* An overview of our proposed framework. A complete many-core system model is created using the core's parameters and behavior of the given DTM scheme. Next, a set of generic DTM properties is formalized. The model and the properties are passed to a model checker for automatic verification, which evaluates performance parameters or generates a counterexample.

system along with the modeling assumptions.

### A. Problem Formalization

Consider a many-core system S, with the parameters as shown in Figure 7(a). The basic verification problem for any DTM scheme running on this system can be defined as:

$$\forall c_i \in C,\ T_i(t)\ > Tth\ \ \rightarrow T_i(t+x) < Tth \qquad (1)$$

where $i$ is from 1 to $N$ and $t$ represents a point in time. Eq. 1 gives the basic criterion for a DTM scheme that if a core's temperature goes beyond a predefined threshold (at some point in time $t$), then the DTM scheme should ensure that the core's temperature would eventually (at some point in time $t+x$) be brought to the allowed level as a result of migration.

### B. Problem Modeling

In order to facilitate the formal modeling of the above problem, we have developed the following generic modules:

**Core's Parameters.** We have generated the following modules to capture the physical parameters of the core.

`Power Model.` We have assumed a linear relation between the task load $L_m$ assigned to a core $m$ and the power $P_m$ required by the core to execute that task. For evaluation purpose, we have assigned a unit power to each unit load, i.e.,

$$P_m = L_m \qquad (2)$$

`Thermal Model.` The thermal model relates the cores's temperature to its power consumption. We have modeled the thermal behavior of a core based on the Equation 3 [28]. The results in [28] show that this model closely relates to thermal simulators like Hotspot [43] and therefore realistically models the thermal behavior of the core.

$$T_m = T_A + \sum_{n \in M} C_{mn} \times P(n), \forall m \in M \qquad (3)$$

where $T_m$ is the temperature of a core $m$, $P(n)$ is the average power consumption during task execution time on core $n \in M$ ($M$ being the set of all the cores) and $T_A$ is the ambient temperature. The matrix $C$ is computed based on the thermal parameters of the processor. Equation 3 takes into account the effects of the surrounding cores on a core's temperature. For example, the amount of change in temperature of the $m^{th}$ core caused by the $n^{th}$ core is given by $C_{m,n}$ times the change in power of the $n^{th}$ core. For experimental purposes, we have taken the thermal effects of the 8-neighborhood only, as shown in Figure 7(b). The reference values for $C_{m,n}$ are taken from [28] after scaling them to per power unit. For example, in Figure 7(b), $C_{0,2}$ means that temperature of Core 0 will rise by 0.156°C when Core 2's power increases by 1 Watt [28]. The values for $C_{m,n}$ for one-hop and diagonal neighbors are 0.156°C/W and 0.131°C/W, respectively, whereas the intra-core $C_{m,n}$, where $m = n$, is taken as 0.366°C/W [28].

**Assumption:** Since the above model does not take into account the transient thermal effects, therefore, we have assigned task durations to be long enough to ignore these effects. This assumption is justified as the current temperature of a core at time $t$, approaches the steady-state temperature with increasing $t$, as given by equation 4 [28].

$$T(t) = P \times R + T_A - (P \times R + T_A - T_i)e^{\frac{-t}{RC}} \qquad (4)$$

where $T_A$ and $T_i$ represent the ambient and the initial temperature respectively, and $R$ and $C$ are the thermal resistance and the thermal capacitance respectively.

`Communication Model.` The core arrangement in the many-core system is shown as a 2-D grid as shown in Figure 7(c). In case of a dDTM scheme, each core communicates with its neighbors only, through a simple core to core communication bus as shown in Figure 7(c). Whereas in case of a cDTM, each core talks to a central core only. This communication includes 32-bit values for temperature *(Core_temp)*, core ID *(Core_ID)* and destination core ID *(Dest_ID)* for migration. One bit each is used to indicate a threshold violation *(Tth_viol)* and whether a destination core has been found or not *(Core_found)*.

**Assumptions:** We have assumed a shared memory model for the many-cores in our system. In case of a migration, only a minimal data structure, i.e., pointers (32-bit each) to the program *(Prog_ptr)* and data memory *(Data_ptr)*, is transferred from one core to another. Therefore, for modeling task migrations, we assume a minimum time effort, i.e., one transition, to carry out migration once a migration core has been identified by the task migration policy in the given DTM.

`Task Queue.` We have modeled a simple task queue to handle the tasks executing on each core in the many-core system. The task (thread) queue is allocated the thread of a task with the parameters chosen non-deterministically from the task model, explained later. In addition to these threads, a migrated thread may also be assigned to the queue depending upon the migration request.

**DTM Parameters.** We have formalized the following generic modules, that can be instantiated to build the model for the given DTM technique.

`Central DTM Agent.` The `Central Agent` module implements the central or global controller. This module communicates with all the other cores or a set of cores in the system and exchanges information related to the temperature and task loads. In case, if one of the cores violates the

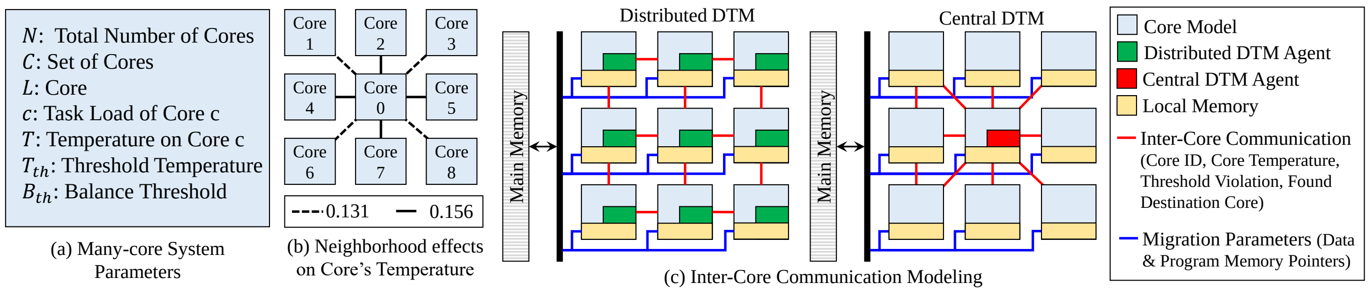
$N$: Total Number of Cores
$C$: Set of Cores
$L$: Core
$c$: Task Load of Core c
$T$: Temperature on Core c
$T_{th}$: Threshold Temperature
$B_{th}$: Balance Threshold
(a) Many-core System Parameters
Core 1
Core 2
Core 3
Core 4
Core 0
Core 5
Core 6
Core 7
Core 8
0.131
0.156
(b) Neighborhood effects on Core's Temperature
Main Memory
Distributed DTM
Main Memory
Central DTM
(c) Inter-Core Communication Modeling
Core Model
Distributed DTM Agent
Central DTM Agent
Local Memory
Inter-Core Communication (Core ID, Core Temperature, Threshold Violation, Found Destination Core)
Migration Parameters (Data & Program Memory Pointers)

*Figure 7:* Formal Modeling of different parameters. (a) and (b) show parameters for a many-core system and inter-core thermal effects, respectively. (c) shows inter-core communication, where the cores communicate with their neighborhood (dDTM) or a central core (cDTM).

threshold temperature, then it sends request to the `Central Agent` to find an appropriate core for task migration. The `central Agent` module then runs the task migration algorithm provided by the selected DTM technique to find a destination core for task migration.

`Distributed DTM Agent`. The `Distributed Agent` modules implement the behavior of a distributed controller and are associated with every core in the system while the information parameters can be exchanged with the neighboring cores only. The `Distributed Agent` modules in a neighborhood take the DTM decision locally whenever task migration is required.

`Thermal Predictor`. Certain DTM techniques [10] use a predictor to estimate the core temperature in the future to take a pro-active decision. The `thermal predictor` module models this aspect of the given DTM technique. The details of this module are governed by the specific DTM and it is not used in case a DTM technique does not employ a thermal predictor. In our case study, mDTM [10] employs a predictor module and its details can be found in Section III-C.

`Migration Policy`. The `migration policy` module implements the task migration behavior of the DTM technique. This module contains the information of the threshold violation condition and the migration criterion from one core to another. In case of the cDTM technique, this module runs only on the central DTM agent. In case of the dDTM technique, each distributed DTM agent runs this module, as each core has to find the appropriate destination core on its own for migration.

**Application Parameters.** In order to generate an application model to be run on our many-core system, we have developed the following modules:

`Single-thread` and `Multi-thread`. In our application model, we have generated separate modules for single and multi -threaded tasks. As the name indicates, these modules are used to model the behavior of the tasks to be run on many-core system by generating single and multi-threaded tasks.

`With migration` and `Without Migration`. Some applications do not allow task migration and thus always reside on the cores that are assigned to them during the initial mapping on a many-core system. If such an application is chosen to run on the system, then, in case of a thermal emergency, the application is halted until the core cools down, i.e., the many-core system runs without a software-level DTM technique. However, in case of a malleable application, the application threads can be moved from one core to another as required with the help of the DTM technique.

**Tasks load and assignment** In contrast to simulation, which test the DTM for the given benchmarks only, model checking exhaustively explores the state-space of the given DTM against the required specification. Therefore, in our models, the mapping of tasks and threads to the cores is done in a non-deterministic fashion and each thread is assigned a non-deterministic load. This approach allows us to verify the functionality of DTM for any arbitrary task load. We have defined a task by three parameters, i.e., number of threads, load and duration.

**Assumptions:** For experimental purpose, we have considered up to 5 threads for an application with load per thread ranging from 1 to 10 load units and duration from 1 to 5 time units. Moreover, we run a total of 400,000 load units on our system model. These values are taken for evaluation purposes only and our task model is not restricted to these values.

## C. Interaction between modules

In this subsection, owe describe the interactions between the different modules mentioned above with the help of their state diagrams, given in Figure 8. The conditions for transitions are shown in blue color and italicized blue is used for internal events or local conditions. The statements in green show the assertions and the updates in the values of different variables (the local updates are given in italics), as a result of these transitions. These modules interact in a synchronous manner to model the overall behavior for a many-core system running a DTM scheme.

The module for `task queue`, shown in Figure 8(a), provides the tasks to be executed on the core. Initially, the queue is filled with a non-deterministic set of loads or tasks. This module checks for the condition *Task_req* to provide a next task for execution. The next task could be a new task (Core_task) with non-deterministic parameters (Task_param = nondeterministic) or a migrated task (Migrated_task and Task_param = Migrate_task_param) from another core depending upon the Migrate condition. Once a task is completed and checked using the condition Task_completed, the queue returns to its initial state to generate the next task.

The task parameters are passed on to the `application` module, shown in Figure 8(b), that models the execution of thread(s). In case of a multi-threaded application, several instances of this `application` module are invoked to model a task with several threads. When the New_task condition is met, the task starts executing if the core's temperature *Core_temp* is less than the threshold *Tth*. *Core_temp* is calculated based on Equation 3, using the core's power consumption, which is linearly related to the Load from `task queue`. The core's temperature can also be passed to an optional (depending upon the chosen DTM scheme) `thermal predictor` module to predict the core's future temperature (Temp_predict) for defining *Tth*. In case of a threshold violation, the `application` module generates the Invoke_migration signal, which is passed to the `migration` module (Figure 8(c)). The `application` module waits for migration, based on the Migrate = 0 condition. When the suitable core is found (Migrate = 1), the task execution resumes. The `application` module also updates the values for local variables like *Time_rem* to keep record of task completion.

The `migration` module, shown in Figure 8(c), receives the Invoke_migration condition from `application` module to trigger the task migration algorithm. Once a core is found (*Core_found = 1*), the `migration` module sends the (Migrate =1) condition to the `communication` module Figure 8(d) along with the information of source and the destination core. The `communication` model transfers 32-bit values for temperature (Core_temp), core ID (Core_ID), one bit for threshold violation (Tth_viol) between the neigh-

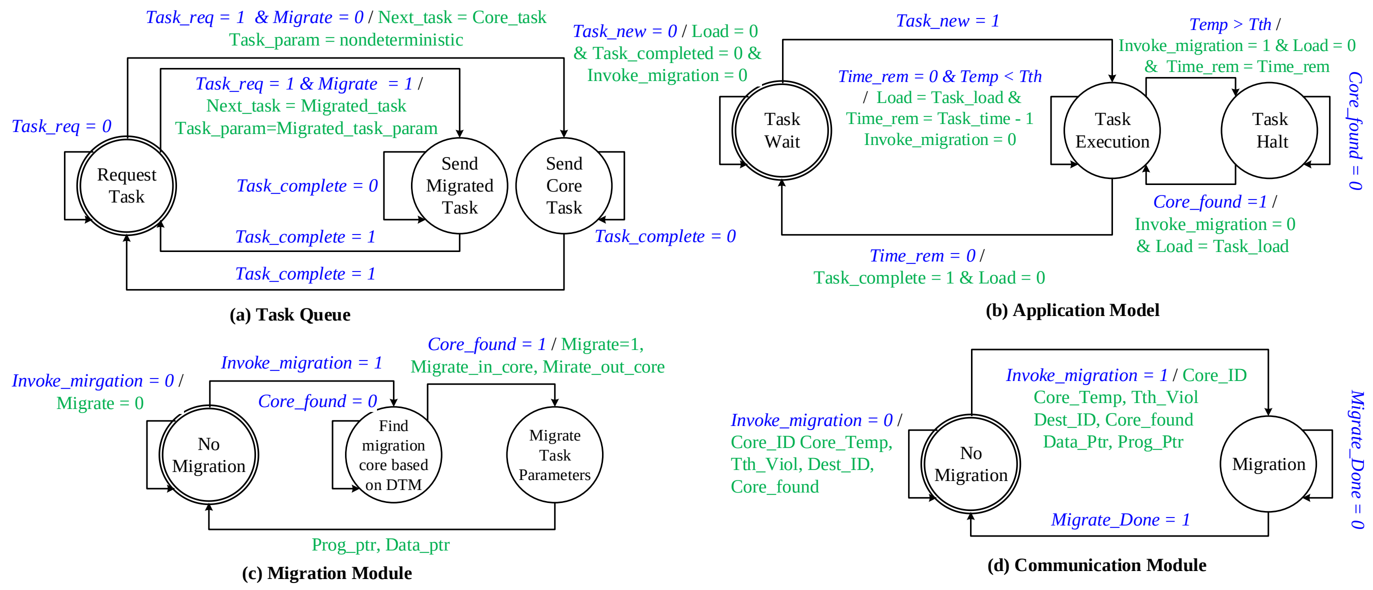

*Figure 8:* State diagrams for different modules used in the modeling. Task Queue (a) passes a new or migrated task to Application module (b), which executes the task or invokes a migration request, depending upon the core's temperature. In case of a migration request, Migration module (c) tries to find a destination core. Communication module (d) transfers the respective core and migration parameters.

boring cores or to a central core. In case of a migration *(Migrate = 1)*, the id of the destination core (Dest_ID = Migrate_in_core) and pointers for data and program, i.e., Data_ptr and Prog_ptr are also transferred between the cores. The internal condition *Migrate_done = 1* is checked to determine the completion of the migration.

## D. Functional Properties

The functional properties check the desired working of DTM schemes and verify the stability of a many-core system with a given DTM scheme. We have defined different variants of the stability condition as a DTM scheme should ideally keep the average chip temperature below a certain threshold as well as minimize the thermal gradient across the chip.

**Stability w.r.t. Threshold Avoidance:** $GF(T_{avg} < T_{th})$
Where G and F represent the LTL operators Global and Future, respectively. This LTL property describes the scenario when the average chip temperature, i.e., $T_{avg}$, eventually becomes less than the predefined threshold temperature $T_{th}$. In other words, the thermally stable state of many-core system is achieved with respect to threshold avoidance. The $GF$ operator ensures that our property is valid somewhere in future $F$, across all the execution paths (globally) $G$ of the state-space model of the DTM scheme.

**Stability w.r.t. Thermal Balancing:** $GF((T_1 - T_{avg} < T_{bal}) \wedge (T_2 - T_{avg} < T_{bal}) \wedge ... \wedge (T_n - Tavg < T_{bal}))$
Besides avoiding the threshold for the average temperature of the many-core system, we also check if a given DTM scheme maintains thermal balance across the chip. The overall system is said to be in a stable state with balanced temperature if the temperature difference of each core in the system with $T_{avg}$ is less then an allowed threshold variation $T_{bal}$.

**Thermal Safety:** $GF((T_{avg} < T_{th}) \wedge (T_1 - T_{avg} < T_{bal}) \wedge (T_2 - T_{avg} < T_{bal}) \wedge ... \wedge (T_n - Tavg < T_{bal}))$
A many-core system is said to be in a thermally safe condition when its $T_{avg}$ is below $T_{th}$ as well as the respective temperature difference of each core with $T_{avg}$ is less than $T_{bal}$. The LTL specification shows that such a DTM scheme satisfying this formal property will eventually reach a thermally safe state in its course of execution for all possible paths.

## E. Performance Properties

We have taken the performance metrics presented in [22] for dDTM techniques and extended them to cDTM. We have also added communication bandwidth as one of the performance metrics. All the performance parameters have been calculated by executing the system traces (using the nuXmv command `execute_trace`) or runs for worst-case scenario, i.e., till the system achieves the thermal stable state. The performance properties are evaluated by implementing counters in nuXmv model that run till the DTM scheme reaches a thermally stable state. The performance properties are detailed as follows:

**Number of transitions required for stability:** For a thermally stable DTM, the most important aspect about its performance is the measure of how quickly the DTM scheme reaches the stable state. Transitions to stability indicate the maximum (worst-case) number of transitions required by the many-core system to reach the thermal stability state. Using such a formal comparison, an easy choice can be made between the given DTM schemes for a many-core system.

**Task load completed:** To compare the performance of DTM schemes in terms of completed task load, while achieving thermal stability, we calculate this parameter by adding the task load completed on all individual cores. A DTM scheme resulting in more number of completed tasks indicates an efficient underlying task migration policy.

**Number of task migrations:** The task migration based DTM schemes, mitigate thermal emergencies in a system by migrating a task from a hot spot to an appropriate core or by redistributing load to achieve thermal balance across the many-core chip. Therefore, a performance efficient DTM scheme should carry out minimum task migrations while achieving a stable state, resulting in a lesser performance overhead.

**Number of task stalls:** DTM schemes stall the execution of a task, if it is running on a hot spot and a destination core cannot be found for task migration. A task stall is necessary to avoid the over heating of the chip, however, it compromises the performance of the system because of the waiting time spent by the heated core to cool down before starting a new task. Thus, a performance efficient DTM scheme should result in fewer task stalls.

**Number of hot spots created:** In addition to task migration aspects, an important performance metric for DTM evaluation is the number of times a many-core system crosses the thermal threshold or becomes a hot spot, under a given DTM scheme. An inefficient DTM scheme may result in a task migration decision with adverse effect on the number of hot spots. Therefore, analyzing a DTM scheme in terms of hot pots created is critical, as continued heating of a chip might result in physical damage and reliability issues.

**Communication overhead:** We have also calculated the communication bandwidth required by a DTM scheme and the total data transferred between the cores till the stability is achieved. This parameter is important as the communication overhead may significantly hinder the performance of a DTM. Also, a comparison based on the communication load can be helpful in choosing a DTM configuration for different grid sizes. For calculating the communication overhead, we have modeled a shared memory system for the many-core systems. Please see `Communication Model` in Subsection V-B for further details on the communication model.

## VI. Case Study

We have taken state-of-the-art DTM algorithms [8], [10], [13] from both the domains and analyzed them using the suggested framework. In this section, we provide the details for our experimental setup followed by functional and performance verification results. For details of the selected DTM algorithms, please see Section III.

### A. Experimental Setup

For verification purpose, we have used the bounded model checking (BMC) approach in nuXmv, with the help of `msat_check_ltlspec_bmc` command [37]. This command employs MathSAT SMT solver [39], under the hood of nuXmv model checker, to ensure the verification of LTL properties up to a desired bound $k$ over infinite state models, i.e., containing $integer$ and $real$ variables, as in case of DTM modeling. In our experimental setup, we have taken $k = 100$. However, our proposed approach is not restricted to the use of BMC for verification. Our approach would result in a complete or *unbounded* verification, if the tool employed can completely check LTL properties over the models containing $real$ and $integer$ parameters.

The verification of the selected DTM schemes has been done for different grid sizes of $3 \times 3$, $4 \times 4$, $6 \times 6$, $9 \times 9$ and $12 \times 12$. We have used nuXmv 1.0.1, running on a Intel Core i7-6700T Quad-Core server operating at 3.06 GHz with 32 GB of RAM. We have run the models for DTM techniques at two different values of threshold temperature for invoking task migration, i.e., 41.5°C and 80°C. The two values have been chosen to analyze the behavior of DTMs w.r.t. different temperature thresholds. The results have been produced for both the application models, i.e., the ones that support and do not support task migration.

### B. Functional Verification

We have verified the selected DTM techniques mDTM [10], rDTM [8] and DBTAS [13] for the functional properties introduced in Section V-D and the verification results are given in Table I. It shows that mDTM meets the three stability criteria defined by functional properties. However, rDTM and DBTAS achieve stability only in terms of threshold avoidance and do not satisfy thermal balancing and safety properties. Also, if a No DTM policy is applied (equivalent to running an application model without migration allowed), then the system still achieves thermal stability w.r.t. threshold avoidance. However, this stability comes at a heavy cost in terms of other performance parameters as discussed in the following Subsection. A thermally unsafe DTM technique may cause reliability issues and even result in physically damaging the chip. mDTM ensures thermal safety of the many-core system by attaining stability w.r.t. both threshold avoidance and thermal balancing, due to separate controllers for each purpose. Therefore, *mDTM seems a better choice than rDTM, DBTAS and No DTM as the foremost objective of a DTM scheme is to achieve thermal stability in a many-core system.*

*Table I:* Verification of Functional Properties

| **DTM Scheme** / **Functional Property** | **mDTM [10]** | **rDTM [8]** | **DBTAS [13]** | **No DTM** |
|---|---|---|---|---|
| **Threshold Avoidance** | ✓ | ✓ | ✓ | ✓ |
| **Thermal Balancing** | ✓ | ✗ | ✗ | ✗ |
| **Thermal Safety** | ✓ | ✗ | ✗ | ✗ |

### C. Performance Evaluation Results

We have also compared the DTM techniques for performance parameters by monitoring their respective counters embedded in each core module. These performance counters are evaluated till the DTM achieves thermal stability and provide further insights into working of the given DTM schemes. Figure 9 shows the number of transitions required by each DTM technique and a No DTM policy for different grid sizes to reach a thermally stable state with threshold avoidance.

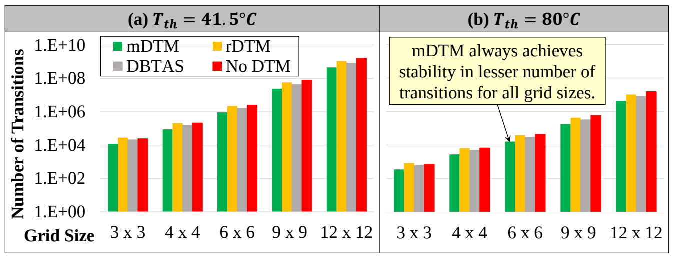

*Figure 9:* Number of transitions required by given DTM schemes to achieve threshold stability.

Table II II shows the results of other performance properties for different gird sizes till the stability w.r.t. threshold avoidance and Figure II presents these results on a logarithmic scale. We give some of the key observations as follows:

**Observation 1:** The verification of the performance properties shows that the central DTM, i.e., mDTM outperforms

*Table II:* Verification Results for Performance Properties

| DTM Techniques | Grid Size | Performance Parameters at $T_{th}$ = 41.5°C | | | | Performance Parameters at $T_{th}$ = 80°C | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | Hot Spots | Task Stalls | Task Load Completed | Task Migrations | Hot Spots | Task Stalls | Task Load Completed | Task Migrations |
| mDTM [10] | 3 × 3 | 214 | 214 | 1870 | 1083 | 55 | 55 | 14947 | 148 |
| | 4 × 4 | 1548 | 1548 | 12879 | 7500 | 430 | 430 | 116970 | 1152 |
| | 6 × 6 | 14834 | 14834 | 133294 | 82562 | 2512 | 2512 | 695067 | 6835 |
| | 9 × 9 | 286935 | 286935 | 2203922 | 1865774 | 27825 | 27825 | 7702786 | 75738 |
| | 12 × 12 | 4806162 | 4806162 | 24051670 | 42370414 | 673051 | 673051 | 186326176 | 1832008 |
| rDTM [8] | 3 × 3 | 99 | 68 | 1019 | 1834 | 26 | 18 | 9220 | 251 |
| | 4 × 4 | 763 | 484 | 8523 | 12701 | 212 | 137 | 72144 | 1949 |
| | 6 × 6 | 6875 | 4704 | 82886 | 139823 | 1166 | 798 | 428698 | 11576 |
| | 9 × 9 | 132970 | 90980 | 1061836 | 3159778 | 12895 | 8823 | 4750857 | 128263 |
| | 12 × 12 | 2227251 | 1523902 | 13229740 | 71756345 | 311904 | 213408 | 114920622 | 3102592 |
| DBTAS [13] | 3 × 3 | 145 | 138 | 1395 | 1482 | 41 | 32 | 12468 | 200 |
| | 4 × 4 | 1164 | 1022 | 10038 | 10254 | 346 | 307 | 100458 | 1551 |
| | 6 × 6 | 11609 | 10448 | 115603 | 118921 | 1967 | 1771 | 600934 | 9846 |
| | 9 × 9 | 224545 | 202091 | 1746365 | 2687414 | 21776 | 19598 | 6659586 | 109090 |
| | 12 × 12 | 3761118 | 3385002 | 19936234 | 61029285 | 526705 | 474034 | 161091726 | 2638778 |
| No DTM | 3 × 3 | 1356 | 1356 | 1130 | 0 | 347 | 347 | 10558 | 0 |
| | 4 × 4 | 11732 | 11732 | 4904 | 0 | 3254 | 3254 | 68692 | 0 |
| | 6 × 6 | 126081 | 126081 | 23918 | 0 | 23143 | 23143 | 366954 | 0 |
| | 9 × 9 | 2931492 | 2931492 | 868913 | 0 | 284272 | 284272 | 3383142 | 0 |
| | 12 × 12 | 53228143 | 53228143 | 17581079 | 0 | 7454026 | 7454026 | 75492433 | 0 |

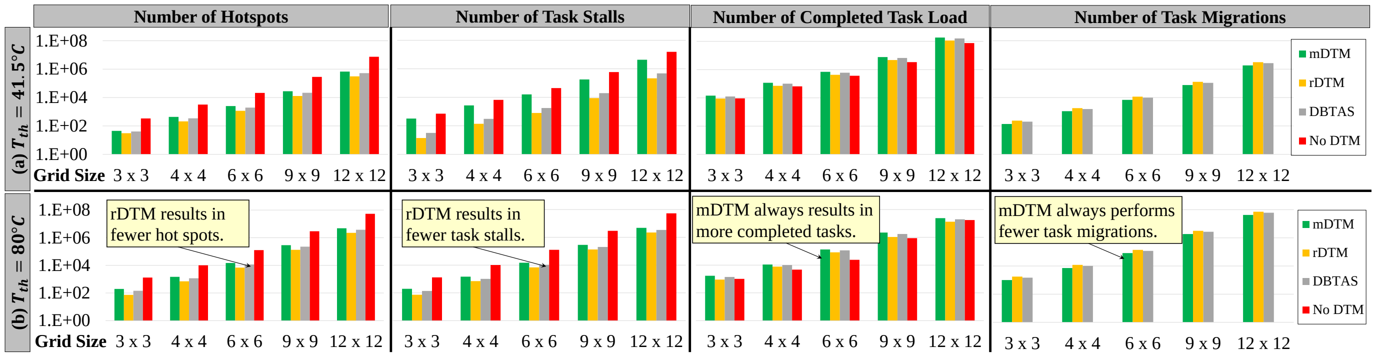

*Figure 10:* Evaluation of performance parameters. DTM schemes are compared based on different parameters derived from DTM principles. These parameters are implemented as counters in DTM models and evaluated till the DTM schemes achieve threshold stability.

rDTM, DBTAS and the case of a No DTM policy. For a 12 × 12 grid size and threshold temperature of 80 °C, the number of transitions required by mDTM to achieve threshold stability are 58.4%, 45% and 72% less than rDTM, DBTAS and No DTM, respectively. This observation shows that mDTM adopts a more efficient DTM policy than rDTM and DBTAS to achieve threshold stability. This comparison is made w.r.t. threshold avoidance as rDTM, DBTAS, and a No DTM policy do not meet the thermal safety criterion.

**Observation 2:** Although, a No DTM policy results in stability w.r.t. threshold avoidance, yet, it behaves poorly in performance metrics. Since a no DTM policy does not invoke any task migration, therefore, it halts the task on the corresponding cores and waits for the core to cool down (while involving temperature effects from the neighboring cores as the tasks running on them are completed) to resume the task. This behavior leads to an increase in the number of transitions to reach a stable state and creates more hot spots. For example, without a DTM, for a 12 × 12 grid size and threshold temperature of 80°C, it takes 3.58, 2.03 and 1.49 times more number of transitions to achieve stability than mDTM, DBTAS and rDTM, respectively. Also, the number of hot spots generated as a result of No DTM policy is 11, 14.15 and 23.8 times greater than mDTM, DBTAS and rDTM, respectively. Similarly, task load completed also decreases in comparison with DTM techniques, as shown in Figure 10. This observation signifies the need for a DTM in a many-core system, as a no DTM policy adversely affects the overall performance of the system.

**Observation 3:** For the same grid size, mDTM performs 1.69 and 1.44 times lesser task migrations as well as completes 62% and 15% more tasks as compared to rDTM and DBTAS, respectively. This observation shows that mDTM achieves stability with more efficiency and lesser overheard than DBTAS and rDTM, owing to an effective task migration policy.

**Observation 4:** Another interesting insight is that *the number of hot spots are equal to the task stalls for mDTM*, however, in case of rDTM, the number of hot spots is different than task stalls. This is because that rDTM discards a task if it cannot find a suitable destination core for the migration whereas mDTM halts the task till a suitable core is found. Due to the same reason, rDTM results in 53.6% lesser hot spots as compared to mDTM. However, this increase in the number of hot spots in mDTM as compared to rDTM does not affect the overall performance of mDTM due to its better DTM controller, as discussed in Observations 1 and 3.

**Observation 5:** Table III shows the number of state transitions required by mDTM to achieve stability according to different criteria mentioned in Section V-D. Since, thermal safety is a more strict LTL property than the other two stability properties and explicitly involves both of them, the number of transitions required to reach a thermally safe state is always greater than that required to achieve the other stability criteria.

*Table III:* Transitions to stability criteria for mDTM [10]

| Grid Size | Threshold = 41.5 °C | | |
|---|---|---|---|
| | Thermal Threshold | Thermal Balancing | Thermal Stability |
| **3 × 3** | 11808 | 59783 | 172521 |
| **4 × 4** | 88165 | 499273 | 1436493 |
| **6 × 6** | 921024 | 5351364 | 15443057 |
| **9 × 9** | 24230016 | 128285512 | 370208551 |
| **12 × 12** | 459095040 | 1369334335 | 3951648723 |
| **Threshold = 80 °C** | | | |
| **3 × 3** | 347 | 59783 | 172521 |
| **4 × 4** | 2755 | 499273 | 1436493 |
| **6 × 6** | 16069 | 5351364 | 15443057 |
| **9 × 9** | 178076 | 128285512 | 370208551 |
| **12 × 12** | 4307513 | 1369334335 | 3951648723 |

**Observation 6:** Figure 11 shows the average number of communication bits transferred by each core till the stability is achieved. This graph shows that for smaller grid sizes, the communication overhead for mDTM is less than rDTM and DBTAS. For example, for 3 × 3 grid size, mDTM requires 4.62 and 3.08 times less number of bits than rDTM and DBTAS, respectively. However, for larger grid sizes, the communication volume for mDTM is greater than rDTM and DBTAS, respectively. For example, for a 12 × 12 grid size, mDTM requires 3.85 and 1.58 times more number of bits than rDTM and DBTAS, respectively. This trend shows that *for smaller grid sizes (up to 6 × 6), the central DTM, i.e., mDTM, performs better than the distributed rDTM and DBTAS, in terms of communication overload.* This is due to the reason that the cDTM technique has a single DTM controller running on a central core and with the increasing grid size, the bandwidth requirement of the central core increases (as the same central core has to communicate with a larger number of cores) more steeply than that of dDTM. This observation also highlights the communication bandwidth as one of the bottlenecks of the centralized DTMs.

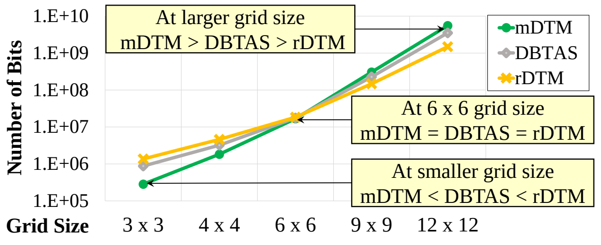

*Figure 11:* Comparison of communication overhead per core.

**Observation 7:** Figure 12 shows verification time for the stability criteria mentioned in Section V-D, for mDTM. Since, thermal safety is a more strict LTL property that implicitly includes the other two stability criteria, therefore, its verification requires more time than the other two LTL specifications.

**Observation 8:** Table IV shows the memory requirements for verifying the stability property for the selected DTM techniques. It shows that the memory requirement for mDTM, rDTM and DBTAS is almost the same (DBTAS requiring 1% and 12% more memory for 4 × 4 grid size than rDTM and mDTM, respectively). For grid sizes 6 × 6 and greater, all the DTM algorithms make use of the maximum available system memory. This behavior shows the growth in the state-space, and thus complexity, for DTM techniques with the increase

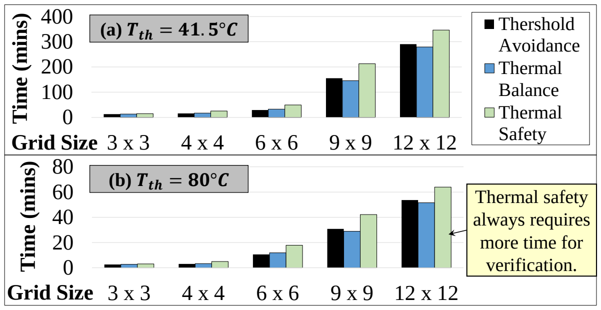

*Figure 12:* Verification time required by mDTM [10]

in number of cores. The differences among the verification times for mDTM, rDTM and BDTAS, are also negligible (a difference of 0.9% between mDTM and rDTM, and 6% between mDTM and DBTAS, for 12 × 12 grid size, and reduce with an increasing grid size. Hence, based on our analysis, we can safely state that *the computational complexity of mDTM, rDTM and DBTAS, for achieving thermal stability with threshold avoidance is the same.* Moreover, this observation also shows that for the same complexity, mDTM performs better in terms of performance parameters (Please see Observations 1, 3 and 4) due to an efficient DTM controller, assenting to mDTM behavior depicted in earlier observations.

*Table IV:* Verification time and memory for threshold stability

| DTM Techniques | Grid Size | Memory (MB) | | Verification time (Sec) | |
|---|---|---|---|---|---|
| | | $T_{th}$=41.5°C | $T_{th}$=80°C | $T_{th}$=41.5°C | $T_{th}$=80°C |
| **mDTM [10]** | 3 × 3 | 18179 | 11294 | 774 | 158 |
| | 4 × 4 | 26197 | 19014 | 943 | 183 |
| | 6 × 6 | 31289 | 31289 | 1761 | 636 |
| | 9 × 9 | 31289 | 31289 | 9330 | 1847 |
| | 12 × 12 | 31289 | 31289 | 17432 | 3218 |
| **rDTM [8]** | 3 × 3 | 17984 | 11204 | 830 | 168 |
| | 4 × 4 | 29056 | 21277 | 992 | 192 |
| | 6 × 6 | 31289 | 31289 | 1831 | 660 |
| | 9 × 9 | 31289 | 31289 | 9478 | 1877 |
| | 12 × 12 | 31289 | 31289 | 17276 | 3189 |
| **DBTAS [13]** | 3 × 3 | 19257 | 11981 | 855 | 174 |
| | 4 × 4 | 29423 | 21455 | 1031 | 200 |
| | 6 × 6 | 31289 | 31289 | 1913 | 691 |
| | 9 × 9 | 31289 | 31289 | 10016 | 1984 |
| | 12 × 12 | 31289 | 31289 | 18483 | 3412 |

## VII. Comparison with State-of-the-Art

A comparison of the proposed framework with the existing works is given in Table V with the following key insights:

1) Our framework provides generic modules that can be used to formally model DTM and many-core systems.
2) We include a realistic thermal model in framework that incorporates the impact of neighboring cores.
3) We have included the inter-core communication cost.
4) Our framework provides for the evaluation of DTM techniques against multi-threaded applications.
5) Our modeling includes different task loads or application configurations that allows us to analyze the scenario of a no DTM policy against different DTM schemes.

## VIII. Conclusion

In this paper, we have proposed the use of model checking for analysis and comparison of central and distributed DTM schemes under multi-threaded workloads considering malleable and without migration models for applications. A complete thermal model capturing effects of the neighboring cores, thermal resistance and capacitance, is integrated in our

*Table V:* Features Comparison of the proposed framework

| **Analysis Framework Features** | | **Ismail et al. [19]** | **Shafaq et al. [21]** | **FAMe-TM [22]** | **CAnDy-TM [27]** | **This Work** |
|---|---|---|---|---|---|---|
| **Provision of generic modules** | | ✗ | ✗ | ✗ | ✗ | ✓ |
| **DTM Configuration** | Central | ✗ | ✗ | ✗ | ✓ | ✓ |
| | Distributed | ✓ | ✓ | ✓ | ✓ | ✓ |
| **Stability Criteria** | Threshold Avoidance | ✓ | ✓ | dDTM Specific | ✓ | ✓ |
| | Thermal Balancing | ✗ | ✗ | ✗ | ✓ | ✓ |
| | Thermal Safety | ✗ | ✗ | ✗ | ✓ | ✓ |
| **Application Model** | Multi-Threaded | ✗ | ✗ | ✗ | ✗ | ✓ |
| **Thermal Model** | Neighbor Effect | ✗ | ✗ | ✗ | ✗ | ✓ |
| | Thermal Resistance | ✗ | ✗ | ✗ | ✓ | ✓ |
| | Thermal Capacitance | ✗ | ✗ | ✗ | ✗ | ✓ |
| **Communication Model** | | ✗ | ✗ | ✗ | ✓ | ✓ |
| **Maximum Grid Size** | | $3 \times 3$ | $9 \times 9$ | $9 \times 9$ | $12 \times 12$ | $12 \times 12$ |
| **Tool Used** | | SPIN | PRISM | nuXmv | nuXmv | nuXmv |
| **Open Source** | | ✓ | ✓ | ✓ | ✓ | ✓ |

model. Moreover, we have presented some key have provided functional properties and performance parameters. The functional properties define different thermal stability conditions for a DTM whereas the performance parameters evaluate a DTM with the help of task migration principles. Using our proposed framework, the state-of-the-art DTM schemes from central and distributed domains have been compared using the nuXmv model checker. Our verification results show that the central DTM technique, i.e., mDTM [10] achieves a thermally safe state by satisfying the threshold avoidance and temperature balancing stability criteria. Whereas, the distributed DTM technique, i.e., rDTM [8] and DBTAS [13] achieve stability only in terms of threshold avoidance. Further analysis with the help of performance parameters shows that if no DTM policy is applied to the many-core system, the system still achieves stability w.r.t. threshold avoidance, but at the cost of degradation in performance. As compared to rDTM and DBTAS, the time required by mDTM to achieve the threshold stability is 58.4% less and 45%. The analysis also reveals that, for smaller grid sizes (up to $6 \times 6$), the centralized mDTM performs better in terms of communication overhead as compared to the distributed rDTM and DBTAS schemes. However, for grid sizes greater than $6 \times 6$, rDTM and DBTAS schemes perform better than the central mDTM. We believe that the traditional analysis methods, like simulation or emulation, cannot provide a fair comparison between various DTM configurations due to their incompleteness. Whereas, this work provides a common comparison ground for DTM analysis and highlights many insights about the given DTM schemes [8], [10], [13]. We have also provided open-source access to the code [29] for our work, to facilitate reproduction of results and designing of DTMs with formally verified properties.

## Acknowledgement

This work was partially supported by the Erasmus+ International Credit Mobility (KA107).

## References

[1] W. Huang, M. R. Stan, K. Sankaranarayanan, R. J. Ribando, and K. Skadron, "Many-core design from a thermal perspective," in *Des. Autom. Conf.*, 2008, pp. 746–749.

[2] W. Huang, K. Skadron, S. Gurumurthi, R. J. Ribando, and M. R. Stan, "Exploring the thermal impact on manycore processor performance," in *Semicond. Thermal Meas. and Manage. Symp.* IEEE, 2010, pp. 191–197.

[3] J. Donald and M. Martonosi, "Techniques for multicore thermal management: Classification and new exploration," in *Computer Architecture*, 2006, pp. 78–88.

[4] J. Kong *et al.*, "Recent thermal management techniques for microprocessors," *ACM Comput. Surv.*, vol. 44, no. 3, pp. 13:1–13:42, 2012.

[5] D. Kudithipudi *et al.*, "Thermal management in many core systems," in *Evolutionary Based Solutions for Green Comput.* Springer, 2013, pp. 161–185.

[6] V. Hanumaiah and S. Vrudhula, "Temperature-aware dvfs for hard real-time applications on multicore processors," *Comput., IEEE Trans.*, vol. 61, no. 10, pp. 1484–1494, 2012.

[7] R. Mukherjee and S. O. Memik, "Physical aware frequency selection for dynamic thermal management in multi-core systems," in *Comput.-Aided Des.*, 2006, pp. 547–552.

[8] Z. Liu, X. Huang, S.-D. Tan, H. Wang, and H. Tang, "Distributed task migration for thermal hot spot reduction in many-core microprocessors," in *ASIC*, 2013, pp. 1–4.

[9] M. A. Al Faruque, J. Jahn, T. Ebi, and J. Henkel, "Runtime thermal management using software agents for multi-and many-core architectures," *IEEE Des. & Test of Comput.*, vol. 27, no. 6, pp. 58–68, 2010.

[10] H. Khdr, T. Ebi, M. Shafique, H. Amrouch, and J. Henkel, "mdtm: multi-objective dynamic thermal management for on-chip systems," in *Des., Autom. & Test in Europe*, 2014, pp. 330–336.

[11] M. Kadin *et al.*, "Central vs. distributed dynamic thermal management for multi-core processors: Which one is better?" in *Great Lakes Symp. on VLSI*, 2009, pp. 137–140.

[12] G. Singla, G. Kaur, A. K. Unver, and U. Y. Ogras, "Predictive dynamic thermal and power management for heterogeneous mobile platforms," in *Des., Autom. & Test in Europe*, 2015, pp. 960–965.

[13] S. K. S. Tyagi *et al.*, "Thermal-aware power-efficient deadline based task allocation in multi-core processor," *J. of Comput. Sci.*, vol. 19, pp. 112–120, 2017.

[14] Y. Cui, W. Zhang, V. Chaturvedi, and B. He, "Decentralized thermal-aware task scheduling for large-scale many-core systems," *IEEE Trans. VLSI Syst.*, vol. 24, no. 6, pp. 2075–2088, 2016.

[15] B. Yun, K. G. Shin, and S. Wang, "Predicting thermal behavior for temperature management in time-critical multicore systems," in *Real-Time and Embedded Technology and Applications*, 2013, pp. 185–194.

[16] T. Ebi, M. Faruque, and J. Henkel, "Tape: Thermal-aware agent-based power economy for multi/many-core architectures," in *Comput.-Aided Des.*, 2009, pp. 302–309.

[17] B. Wojciechowski *et al.*, "Fast and accurate thermal simulation and modelling of workloads of many-core processors," in *Thermal Investigations of ICs & Syst.*, 2011, pp. 1–6.

[18] S. Ananthanarayanan *et al.*, "Empower: FPGA Based Emulation of Dynamic Power Management Algorithms for Multi-core Systems on Chip," in *Int. Symp. on FPGAs*, 2012, pp. 266–266.

[19] M. Ismail, O. Hasan, T. Ebi, M. Shafique, and J. Henkel, "Formal verification of distributed dynamic thermal management," in *Comput.-Aided Des.* IEEE, 2013, pp. 248–255.

[20] S. A. A. Bukhari *et al.*, "Formal Verification of Distributed Task Migration for Thermal Management in On-chip Multi-core Systems using nuXmv," in *Formal Techn. for Safety-Critical Syst.*, ser. Commun. in Comput. and Inf. Sci., vol. 476, 2015, pp. 32–46.

[21] S. Iqtedar, O. Hasan, M. Shafique, and J. Henkel, "Probabilistic formal verification methodology for decentralized thermal management in on-chip systems," in *Enabling Technologies: Infrastructure for Collaborative Enterprises*, 2015, pp. 210–215.

[22] S. A. A. Bukhari *et al.*, "FAMe-TM: Formal analysis methodology for task migration algorithms in many-core systems," *Sci. of Comput. Program.*, vol. 133, Part 2, pp. 154 – 174, 2017.
[23] E. M. Clarke, Jr., O. Grumberg, and D. A. Peled, *Model Checking*, 1999.
[24] T. E. Carlson, W. Heirman, and L. Eeckhout, "Sniper: exploring the level of abstraction for scalable and accurate parallel multi-core simulation," in *Int. Conf. for High Performance Comput., Netw., Storage and Anal.*, 2011, p. 52.
[25] N. Binkert et al., "The gem5 simulator," *ACM SIGARCH Comput. Architecture News*, vol. 39, no. 2, pp. 1–7, 2011.
[26] A. Akram and L. Sawalha, "× 86 computer architecture simulators: A comparative study," in *Comput. Des., Int. Conf. on*, 2016, pp. 638–645.
[27] S. A. A. Bukhari, F. K. Lodhi, O. Hasan, M. Shafique, and J. Henkel, "CAnDy-TM: Comparative analysis of dynamic thermal management in many-cores using model checking," in *Des., Autom. & Test in Europe*, 2017, pp. 1289–1292.
[28] Z. Wang and S. Ranka, "A simple thermal model for multi-core processors and its application to slack allocation," in *Parallel & Distrib. Processing*, 2010, pp. 1–11.
[29] Sources, http://save.seecs.nust.edu.pk/projects/comp-dtm/, 2019.
[30] G. J. Holzmann, "The Model Checker SPIN," *IEEE Trans. Softw. Eng.*, vol. 23, no. 5, pp. 279–295, 1997.
[31] L. Lamport, "Time, clocks, and the ordering of events in a distributed system," *Commun. ACM*, vol. 21, no. 7, pp. 558–565, 1978.
[32] M. Kwiatkowska *et al.*, "Prism: Probabilistic symbolic model checker," in *Int. Conf. on Modelling Techn. and Tools for Comput. Performance Evaluation*, 2002, pp. 200–204.
[33] S. Iqtedar, O. Hasan, M. Shafique, and J. Henkel, "Formal probabilistic analysis of distributed dynamic thermal management," in *Des. Autom. & Test in Europe*, 2015, pp. 1221–1224.
[34] H. Mizunuma, Y.-C. Lu, and C.-L. Yang, "Thermal coupling aware task migration using neighboring core search for many-core systems," in *VLSI Des., Autom., and Test*, 2013, pp. 1–4.
[35] M. U. Sardar *et al.*, "Theorem proving based formal verification of distributed dynamic thermal management schemes," *J. of Parallel and Distrib. Comput.*, vol. 100, pp. 157–171, 2017.
[36] Intel, "Intel xeon processor 7400 series thermal/mechanical design guidelines," www.intel.com/Assets/en_US/PDF/designguide/320337.pdf, 2017.
[37] R. Cavada et al., "The nuXmv Symbolic Model Checker," in *Comput. Aided Verification*, 2014, vol. 8559, pp. 334–342.
[38] C. Barrett and C. Tinelli, "Satisfiability modulo theories," in *Handbook of Model Checking*. Springer, 2018, pp. 305–343.
[39] A. Cimatti *et al.*, "The mathsat5 smt solver," in *Int. Conf. on Tools and Algorithms for the Construction and Anal. of Syst.*, 2013, pp. 93–107.
[40] L. De Moura and N. Bjørner, "Z3: An efficient smt solver," in *Int. conference on Tools and Algorithms for the Construction and Anal. of Syst.*, 2008, pp. 337–340.
[41] R. Bruttomesso *et al.*, "The opensmt solver," in *Int. Conf. on Tools and Algorithms for the Construction and Anal. of Syst.*, 2010, pp. 150–153.
[42] A. Biere *et al.*, "Bounded model checking." *Handbook of Satisfiability*, vol. 185, pp. 457–481, 2009.
[43] W. Huang, S. Ghosh, S. Velusamy, K. Sankaranarayanan, K. Skadron, and M. R. Stan, "Hotspot: a compact thermal modeling methodology for early-stage vlsi design," *IEEE Trans. VLSI Syst.*, vol. 14, no. 5, pp. 501–513, 2006.

**Syed Ali Asadullah Bukhari** received his B. Eng. and MS. Eng. degrees from National University of Sciences and Technology (NUST), Islamabad, Pakistan. Currently, he is pursuing PhD Eng. from NUST and working as a Research Assistant at System Analysis and Verification (SAVe) Lab, at School of Electrical Engineering and Computer Science (SEECS), NUST. His research interests include formal analysis and verification of embedded systems and dynamic thermal management for Many-Core systems. Mr. Bukhari is a member of Pakistan Engineering Council (PEC).

**Faiq Khalid** (S'18) received the M.S. degree in electrical engineering and the B.E. degree in electronics engineering from NUST, Pakistan, in 2016 and 2011, respectively. He is currently pursuing the Ph.D. degree in hardware security with Computer Architecture and Robust, Energy-Efficient Technologies at Vienna University of Technology, Austria. His research interests include formal analysis and verification of embedded systems, hardware design security and reliability of VLSI circuits and machine learning. He has received the Quaid-e-Azam Gold Medal for his academic achievements, the Best Researcher Award at the SAVe Lab in 2014, and the prestigious Richard Newton Fellowship at DAC 2018.

**Osman Hasan** received the B.Eng. (Hons.) degree from the N-W.F.P University of Engineering and Technology, Pakistan, in 1997, and the M.Eng. and Ph.D. degrees from Concordia University, Montreal, Canada, in 2001 and 2008, respectively. He served as an ASIC design Engineer from 2001 to 2003 at LSI Logic Corporation in Ottawa, Canada and as a Research Associate at Concordia University, for 18 months after his doctoral degree. Currently, he is an Associate Professor at the NUST-SEECS, Pakistan. He is the founder and director of SAVe Lab at NUST, which mainly focuses on the design and formal verification of embedded systems. He has received several awards and distinctions, including the Pakistans Higher Education Commissions Best University Teacher (2010) and Best Young Researcher Award (2011) ) and the Presidents gold medal for the best teacher of the University from NUST in 2015. Dr. Hasan is a Senior member of IEEE, member of Association for Automated Reasoning (AAR) and member of PEC.

**Muhammad Shafique** (M'11 - SM'16) received the Ph.D. degree in computer science from the Karlsruhe Institute of Technology, Germany, in 2011. He is currently a Full Professor with the Department of Informatics, Institute of Computer Engineering, TU Wien, Austria, where he is directing the group on Computer Architecture and Robust, Energy-Efficient Technologies. He holds one U.S. patent and over 200 papers in premier journals and conferences. His research interests include computer architecture, energy-efficient systems, robust computing, hardware security, brain-inspired computing, emerging technologies, and embedded systems. His research has a special focus on cross-layer analysis, the modeling, design, and optimization of computing and memory systems, and their integration in the Internet of Things and smart cyber-physical systems. He is a Senior Member of the IEEE and a member of the ACM, SIGARCH, SIGDA, SIGBED, and HiPEAC. He received the prestigious 2015 ACM/SIGDA Outstanding New Faculty Award, six gold medals, and several best paper awards and nominations at prestigious conferences. He served on the TPM of several conferences and gave several invited talks, tutorials, and keynotes.

**Jörg Henkel** (M'95 - SM'01 - F'15) is the Chair Professor for Embedded Systems at Karlsruhe Institute of Technology. Before that he was a research staff member at NEC Laboratories in Princeton, NJ. He received his diploma and Ph.D. (Summa cum laude) from the Technical University of Braunschweig. His research work is focused on co-design for embedded hardware/software systems with respect to power, thermal and reliability aspects. He has received six best paper awards throughout his career from, among others, ICCAD, ESWeek and DATE. For two consecutive terms he served as the Editor-in-Chief for the ACM Transactions on Embedded Computing Systems. He is currently the Editor-in-Chief of the IEEE Design&Test Magazine and is/has been an Associate Editor for major ACM and IEEE Journals. He has led several conferences as a General Chair incl. ICCAD, ESWeek and serves as a Steering Committee chair/member for leading conferences and journals for embedded and cyber-physical systems. Prof. Henkel coordinates the DFG program SPP 1500 Dependable Embedded Systems and is a site coordinator of the DFG TR89 collaborative research center on Invasive Computing. He is the chairman of the IEEE Computer Society, Germany Chapter, and a Fellow of the IEEE.