Formal Verification of Distributed Task Migration for Thermal Management in On-chip Multi-core Systems using nuXmv

Syed Ali Asadullah Bukhari   Faiq Khalid Lodhi

September 15, 2014

Abstract

With the growing interest in using distributed task migration algorithms for dynamic thermal management (DTM) in multi-core chips comes the challenge of their rigorous verification. Traditional analysis techniques, like simulation and emulation, cannot cope with the design complexity and distributed nature of such algorithms and thus compromise on the rigor and accuracy of the analysis results. Formal methods, especially model checking, can play a vital role in alleviating these issues. Due to the presence of continuous elements, such as temperatures, and the large number of cores running the distributed algorithms in this analysis, we propose to use the nuXmv model checker to analyze distributed task migration algorithms for DTM. The main motivations behind this choice include the ability to handle the *real* data type and the scalable SMT based bounded model checking capabilities in nuXmv that perfectly fit the stability and deadlock analysis requirements of the distributed DTM algorithms. The paper presents the detailed analysis of a state-of-the-art task migration algorithm of distributed DTM for many-core systems. The functional and timing verification is done on a larger grid size of 9 x 9 cores, which is thermally managed by the selected DTM approach. The results indicate the usefulness of the proposed approach, as we have been able to catch a couple of discrepancies in the original model and gain many new insights about the behavior of the algorithm.

1 Introduction

The ever increasing need of the computing power and technological advances have led to many cores on a chip [1, 2]. This accelerated increase, accompanied with higher power densities, has opened up the challenge of coping with the elevated chip temperatures, which poses serious threats to the reliability of the computing systems. Various Thermal Management (TM) techniques [3, 4, 5] have recently been proposed to overcome these issues. In particular, the Dynamic Thermal management (DTM) [6, 7] for multi-core systems via the task migration mechanism has been identified as a very promising solution to the heating problems in many-core systems with high core integration by the ITRS roadmap of 2013 [8].

The DTM techniques can be broadly classified into two categories: central and distributed [9]. Central DTM (cDTM) is done by a central controller, which is responsible for the overall thermal management of the chip and thus, has the visibility of all the global parameters, such as the core temperatures, of the system [10]. This approach has the inherent issue of scalability as the cDTM often encounters performance degradation while dealing with many-core systems [11, 12, 13]. On the other hand, Distributed DTM (dDTM) manages the heating issues of the chip by employing several thermal management agents as opposed to a single controller [9, 13]. An agent in a distributed system perceives the environment through communication with other agents and can take decisions on its own to a certain extent [14]. The obvious gain in this method is that the need of global knowledge is no longer necessary and thus it resolves the above-mentioned scalability issue of cDTM. Since the dDTM agents are not aware of the overall thermal scenario of the complete chip so it is customary to approximate the required data by information exchange among the neighboring agents only. Based on this information, some dDTM techniques develop an overall thermal model of the system for predicting the core temperatures [15, 16, 17]. If this estimate is above a certain threshold then the task migration is activated. Other dDTM techniques, like [18, 19, 20], make the task migration decisions based on certain algorithms that manipulate the temperature values obtained from the neighboring cores only. For example, a recent task migration technique [20], estimates the average temperature of the complete chip by taking the inputs of the neighboring cores using the distributed signal average tracking algorithm [21, 22].

The need for a thorough analysis of these thermal management techniques is of vital importance as an inefficient task migration decision may lead to the creation of hot spots (regions with excessive temperatures within the chip) and thus endanger the reliability of the chip. Traditionally, the dDTM techniques are analyzed using either simulations or by running on real hardware systems. Both of these methods compromise on the accuracy of the analysis results by analyzing a subset of the possible scenarios only due to their large design-space, which is in turn caused by the distributed nature of DTM techniques and the presence of 100s of cores in the present-age systems where the distrusted DTM techniques are employed. Moreover, choosing the sample set is another major issue while analyzing the dDTM techniques due to the enormous amount of possible options, like the possible temperature values for all the cores are actually infinite due to the continuous nature of temperature. This non-exhaustiveness and incompleteness of the analysis may lead to unwanted scenarios, like the delayed release of the Montecito chip using the Foxton DTM algorithm [23].

Formal verification [24] can overcome the above-mentioned inaccuracy limitations of simulation-based verification due to its inherent soundness and completeness. Given the reactive nature of DTM techniques, model checking has been used for their analysis [25, 26, 27]. Moreover, the SPIN model checker [28] has been recently used in conjunction with Lamport timestamps [29] to analyze the functional and timing properties of the Thermal-aware Agent-based Power Economy (TAPE) [30], which is a state-of-the-art agent-based dDTM scheme. However, this analysis is only done for a 9 core, i.e., 3 x 3, core system and the continuous values of algorithm parameters and the temperate have been abstracted by discrete values in order to cope up with the state-space explosion problem of model checking [31]. These abstractions limit the usefulness of applying model checking for analyzing dDTM techniques as the exhaustiveness of the analysis is somewhat compromised to a certain degree.

The main focus of the current paper is to alleviate the above-mentioned issues encountered in [30]. For this purpose, we propose to use the recently released nuXmv model checker [32] to analyze dDTM systems. The distinguishing features of the nuXmv model checker include the ability to handle *real* data types and implicit handling of state counters. Thus, the continuous values in dDTM approaches can be modeled more appropriately and the timing properties of the DTM approached can be analyzed without using the Lamport timestamps explicitly. Moreover, the SAT and SMT based engines of the nuXmv model checker facilitate analyzing larger models and we can thus analyze large grids of multi-core systems.

In order to illustrate the usefulness of the proposed approach, this paper presents the formal analysis of a recently proposed task migration algorithm for hot spot reduction in many-core systems [20]. The algorithm executes the task migration based on a simple criterion of comparing the temperature of the core(s) with the neighboring cores and the average temperature of the chip. The average temperature of the chip in turn is computed using the recently proposed technique of distributed average estimation for time-varying signals [22]. Besides the generic and simplistic nature of this algorithm (as it just manipulates the temperature values from its neighbor to make decision for task migration), another main motivation for choosing this as our case study is its close relationship with other advanced task migration algorithms, such as [33]. Moreover, model checking is not suitable for dDTM techniques like [15, 16, 17], due to their predictive nature.

2 Preliminaries

In this section, we give a brief introduction to the nuXmv model checker and the task migration algorithm for many-core systems [20], which we have formally verified in this paper. The intent is to facilitate the understanding of the rest of the paper for both the dDTM technique design and the formal methods communities.

2.1 nuXmv Model Checker

The nuXmv symbolic model checker [32, 34] is a very recent formal verification tool that extends the NuSMV model checker [35], which in turn is a finite state transitions model checker. nuXMV extends the capabilities of the NuSMV by complementing NuSMV's verification techniques by SAT algorithms for finite state systems. For infinite state systems, it introduces new data types of *Integers* and *Reals* and also provides the support of Satisfiability Modulo Theories (SMT), using MathSAT [36], for verification.

The system that needs to be modeled is expressed in the nuXmv language, which supports the modular programming approach where the overall system is divided into several modules that interact with one another in the `MAIN` module. The properties to be verified can be specified in nuXmv using the Linear Temporal Logic (LTL) and Computation Tree Logic (CTL). The LTL specifications are written in nuXmv with the help of logical operations like, AND (`&`), OR (|), Exclusive OR (`xor`), Exclusive NOR (`xnor`), implication (`->`) and equality (`<->`), and temporal operators, like Globally (`G`), Finally (`F`), next (`X`) and until (`U`). Similarly, the CTL specifications can be written by combining logical operations with quantified temporal operators, like exists globally (`EG`), exists next state (`EX`) and forall finally (`AF`). In case a property turns out to be false, a counter example in the execution trace of the FSM is provided.

2.2 Task Migration Algorithm for Hot Spot Reduction

The main goal of any dynamic DTM technique is to maintain an acceptable average temperature across all the cores. This reduction in the temperature does not always guarantee a balanced distribution that is actually required for the reduction of thermal hot spots. The algorithm proposed in [20], which is under consideration in this paper, overcomes this limitation by performing distributed task migration with the primary goal of achieving thermal reliability and reduced temperature variance across the chip. The algorithm makes use of the recently proposed distributed average signal tracking algorithm [21, 22], which shows that the states of all the distributed agents converge to the average value of the time-varying reference signals. The following equation is used to estimate the average:

$$\dot{z}_i(t) = \alpha \sum_{j \in N_i} sgn[x_j(t) - x_i(t)]$$

$$x_i(t) = z_i(t) + r_i(t) \tag{1}$$

where $sgn(x)$ is the signum function defined as:

$$sgn(x) = \begin{cases} -1 & \text{if } x < 0 \\ 0 & \text{if } x = 0 \\ 1 & \text{if } x > 0 \end{cases} \tag{2}$$

and $z_i(t)$ is the estimated average signal, $x_i(t)$ and N_i are the states of the distributed agent i and its neighborhood, respectively, $r_i(t)$ is the reference signal with bounded derivatives in a finite time and α is a constant value greater than 0. The task migration algorithm makes use of this fact to estimate the average temperature of a core, without the need of global knowledge of the temperature of every core. The task migration policy is then executed only on the cores having a temperature greater than the estimated average temperature T_{avg}. As a result, a considerable amount of data exchange is avoided among the cores and only necessary task migration is done for effectively reducing the temperature. If a core has a temperature greater than the estimated T_{avg}, then the following task migration criterion is used to check if the task can be migrated from the *current* core to some *destination* core among the neighbors:

1. $T_{destination} < T_{current}$, where $T_{destination}$ and $T_{current}$ are the temperatures of the *destination* and *current* cores, respectively.
2. $P_{destination} < P_{current}$, where $P_{destination}$ is the task load of the destined core and $P_{current}$ is the counterpart of the *current* core.
3. $TNP_{destination} < TNP_{current}$, where $TNP_{destination}$ and $TNP_{current}$ are the workloads of the *destination* and *current* cores, respectively.

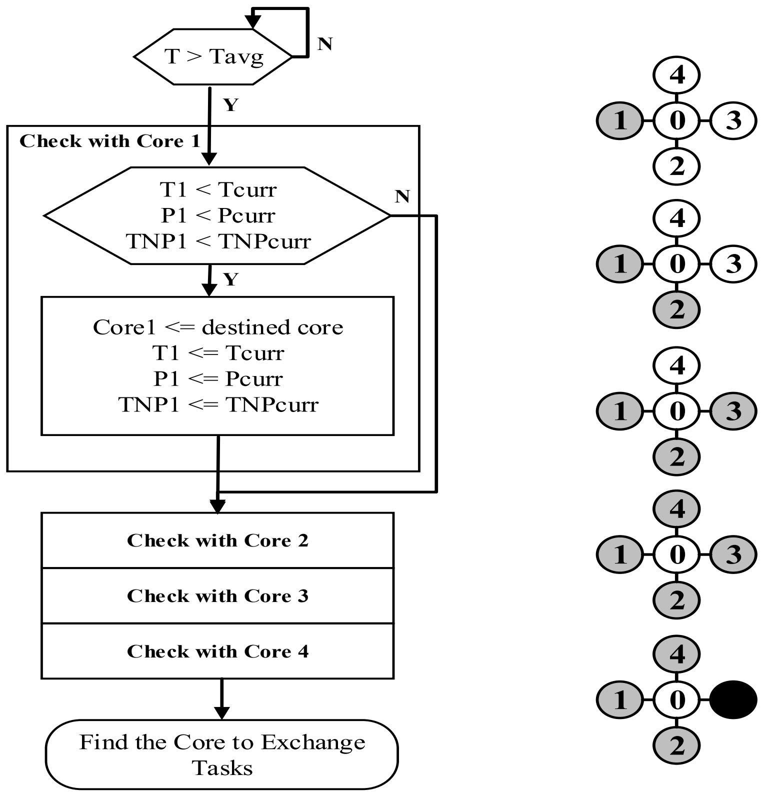
If the temperature T of the core is less than the T_{avg}, then the task migration policy is not activated and the core retains its temperature, otherwise the above mentioned conditions are checked to decide if the task migration is done for a core or not. All the 4 neighbors are passed through the criterion and tasks are exchanged if the conditions are met and then checked with the next neighbor. By the end of the algorithm execution, the most appropriate core is found for task exchange. The pseudo code for this algorithm is given in Algorithm 1 [20] and Fig. 1 presents a typical execution of the algorithm to illustrate the above-mentioned behavior. The node 0, in Fig. 1, represents the current core and the neighboring cores are denoted by 1, 2, 3 and 4. Each core is checked for the satisfiability of the task migration conditions and the right core (shown black) is chosen. The results from MATLAB implementation of the this DTM technique on a 6 x 6 grid show a 30 percent hot spot reduction and smaller temperature variance [20].

3 Modeling the DTM algorithm in nuXmv

In this section, we explain the FSM for Algorithm 1 and its modeling in the language of the nuXmv model checker.

3.1 Our Refinements to the original Task Migration Algorithm

While modeling Algorithm 1 in the nuXmv language, we had to handle some of the scenarios that were not mentioned in the paper [20] where the original algorithm was published. Before going into the implementation details of the model, we find it appropriate to point out the discrepancies in the existing algorithm and our proposed solutions.



(a) Task migration Method Flow
(Applicable to any arbitrary core)

(b) A graphic Example
(the right core is chosen)

Figure 1: A typical execution of the selected algorithm [20].

1. Since the migration algorithm executes concurrently on all the nodes, so it may happen that two different nodes node A and B want to migrate their task to the same name node C at the same time. The algorithm, proposed in [20], does not resolve this conflict. In our model, we have resolved this conflict by giving priority to the node that has a lower value of estimated T_{avg}. This means that all the nodes not only need to know the temperatures of their neighboring nodes, but also of the nodes that could possibly migrate tasks with their neighbors. This revision caters for the conflict resolution but increases the complexity of the algorithm.

2. Another conflict of a similar nature arises when any node A desires to retain its value, because its current temperature is lesser than the estimated T_{avg}, while one of its neighboring nodes wants to exchange the

Algorithm 1 Distributed thermal management Algorithm for avoiding hot spots [20]

```
Require: Task loads, many-core processor 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 the many-core microprocessor, and
    estimate average temperature using distributed state tracking algorithm
    if migration criteria is met then
      Perform distributed task migration using the proposed scheme in Fig. 1
      core by core.
    end if
  end for
```

tasks, based on the execution of its task migration policy. This situation is also resolved by priority assignment based on T_{avg} in our refinement.

3.2 FSM for the Revised Algorithm

The FSM, depicted in Fig. 2, details the working of the refined algorithm for core 0. The temperature, task load, workloads including the neighboring cores and esitmated average temperature by each core are represented by Ts, Ps, $TNPs$ and T_{avg}, respectively. The temperature of the core is compared with the estimated average temperature, i.e., T_{avg}. If the core temperature is greater then the task migration policy is activated and the migration criterion is executed on the neighboring cores one by one to select the core for the migration. Once, the destination core is selected, the improved condition for the task migration is checked to finalize the core selection. The respective conditions are shown in the FSM.

3.3 Modeling the Average Estimation Algorithm

In order to model Equation (1), we have to first take the integral of the $\dot{z}_i(t)$, The integral of a signum function is given as [37] :

$$\int sign(x)\,dx = |x| \tag{3}$$

Thus the equation for $z_i(t)$ becomes

$$z_i(t) = \alpha \sum_{j \in N_i} |x_j(t) - x_i(t)|$$

and we have

$$x_i(t) = \alpha \sum_{j \in N_i} |x_j(t) - x_i(t)| + r_i(t) \tag{4}$$

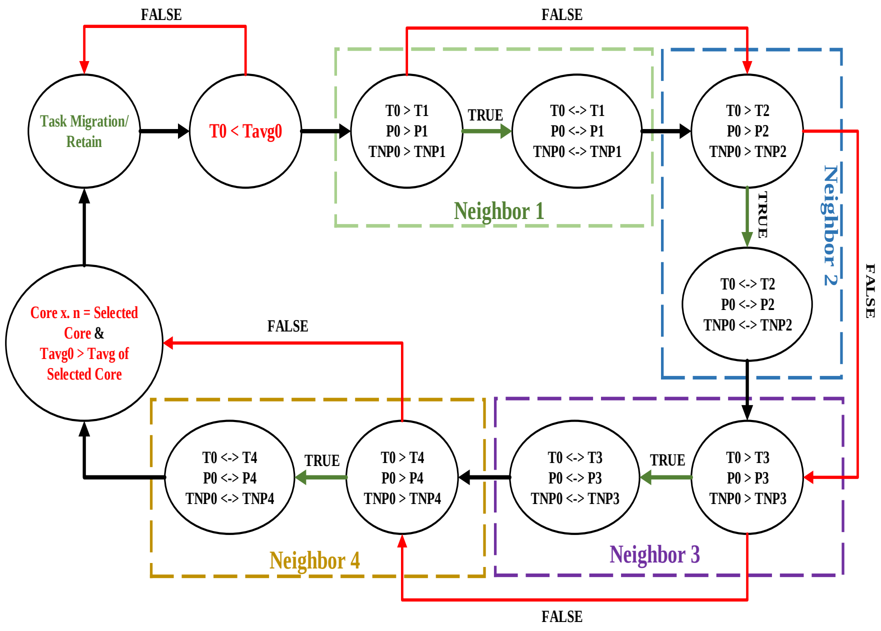


Figure 2: Finite State Machine showing the working of the algorithm for core 0.

In our modeling, $x_i(t)$ becomes equivalent to T_{avg} that a core i estimates, and r_i becomes the core i's temperature.

3.4 Model for the 9 x 9 grid

The algorithm, under verification, allows its nodes to exchange information with a maximum of four neighbors, i.e., north, south, east and west. Information exchange with the diagonal neighboring nodes is not allowed. In order to construct the model of any arbitrary n x n grid, which supports the originally proposed algorithm of [20], we need three distinct types of nodes, i.e., nodes that can communicate with 2, 3 and 4 neighbors, depending on their location in the grid. However, our refinement of the original algorithm requires 6 different types of nodes, as a node with 3 neighbors may need information of 4 or 5 second level neighboring nodes depending on its location in the grid. We have defined second level neighbors of a core x as the cores that can communicate with the neighbors of that core x. Similarly, a four neighbor node may require information of 4, 5 or 6 second level neighboring nodes depending on its location in the grid. Therefore, we have modeled the 9 x 9 grid using six different modules: `n2_3, n3_4, n3_5, n4_6, n4_7` and `n4_8` as shown in Fig. 3. The name of these modules `nx_y` show that the cores modeled by the this module have `x` immediate neighbors and y other second-level neighbors that can exchange tasks with this core or its neighbors. The `MAIN` module calls the instances of these six distinct modules to complete the overall model of a 9 x 9 grid. This model is then used for verifying both functional and timing properties of the given algorithm in the next section.

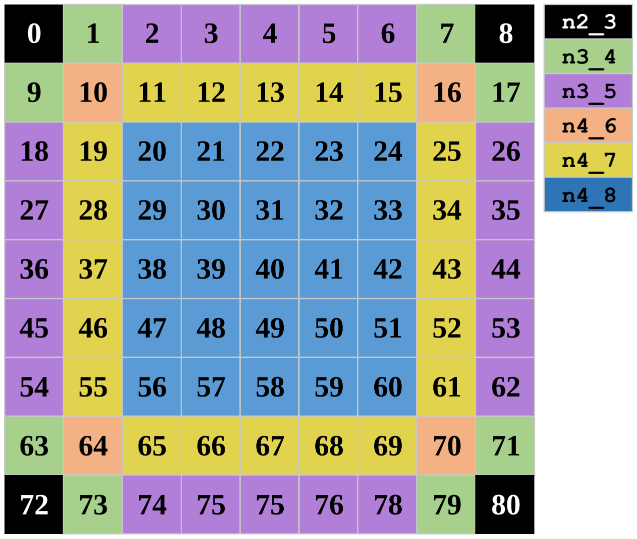


Figure 3: Categorization of cores based on the amount of information exchange.

For illustrating the behavior of these modules, the code listing for the `n2_3` module is shown below:

```
VAR
    Tavg  : Real; n : 0 .. 81; T0    : Real; P0   : Real;
    TNP0  : Real; a1: m(Tavg1, Tavg); a2 : m(Tavg2, Tavg);
 ASSIGN
    init(n)          := n0; init(P0)     := T0/10;
    init(TNP0)   :=  P1 + P2;
    next(Tavg)   := a1.c + a2.c + T0;
    next(T0)     :=
    case
        T0 > Tavg :
        case
        ((T0 > T1) & (T1 < T2))&((P0 > P1) & (P1 < P2))&
        ((TNP0 > TNP1) & (TNP1 < TNP2)) & (nb1 != b1) :
        case
        (n1 != b1) & (n3 != b1) & (nb1 != b1) : T1;
        ((n1 = b1) & (Tavg < avg1)) & ((n3 = b1) &
        (Tavg < avg3)) : T1;
        TRUE : T0;
        esac;
        ((T0 > T2) & (T2 < T1))&((P0 > P2) & (P2 < P1))&
        ((TNP0 > TNP2) & (TNP2 < TNP1)) & (nb2 != b2) :
        case
        (n1 != b2) & (n2 != b2) & (nb2 != b2) : T2;
        ((n1 = b2) & (Tavg < avg1)) & ((n3 = b2) &
        (Tavg < avg3)) : T2;
        TRUE : T0;
        esac;
        TRUE : T0;
        esac;
        TRUE : T0;
    esac;
    next(P0)    :=
    case
        T0 > Tavg :
        case
        ((T0 > T1) & (T1 < T2))&((P0 > P1) & (P1 < P2))&
        ((TNP0 > TNP1) & (TNP1 < TNP2)) & (nb1 != b1) :
        case
        (n1 != b1) & (n3 != b1) & (nb1 != b1) : P1;
        ((n1 = b1) & (Tavg < avg1)) & ((n3 = b1) &
        (Tavg < avg3)) : P1;
        TRUE : P0;
        esac;
        ((T0 > T2) & (T2 < T1))&((P0 > P2) & (P2 < P1))&
        ((TNP0 > TNP2) & (TNP2 < TNP1)) & (nb2 != b2) :
        case
        (n1 != b2) & (n2 != b2) & (nb2 != b2) : P2;
        ((n1 = b2) & (Tavg < avg1)) & ((n3 = b2) &
        (Tavg < avg3)) : P2;
        TRUE : P0;
        esac;
        TRUE : P0;
        esac;
        TRUE : P0;
    esac;
    next(TNP0)    :=
    case
        T0 > Tavg :
        case
        ((T0 > T1) & (T1 < T2))&((P0 > P1) & (P1 < P2))&
        ((TNP0 > TNP1) & (TNP1 < TNP2))  & (nb1 != b1):
        case
        (n1 != b1) & (n3 != b1) & (nb1 != b1)  : TNP1;
        ((n1 = b1) & (Tavg < avg1)) & ((n3 = b1) &
        (Tavg < avg3)) : TNP1;
        TRUE : TNP0;
        esac;
        ((T0 > T2) & (T2 < T1))&((P0 > P2) & (P2 < P1))&
        ((TNP0 > TNP2) & (TNP2 < TNP1)) & (nb2 != b2) :
        case
        (n1 != b2) & (n2 != b2) & (nb2 != b2) : TNP2;
        ((n1 = b2) & (Tavg < avg1)) & ((n3 = b2) &
        (Tavg < avg3)) : TNP2;
        TRUE : TNP0;
        esac;
        TRUE : TNP0;
        esac;
        TRUE : TNP0;
    esac;
    next(n)     :=
    case
        T0 > Tavg :
        case
        ((T0 > T1) & (T1 < T2))&((P0 > P1) & (P1 < P2))&
        ((TNP0 > TNP1) & (TNP1 < TNP2)) & (nb1 != b1) :
        case
        (n1 != b1) & (n3 != b1) & (nb1 != b1) : b1;
        ((n1 = b1) & (Tavg < avg1)) & ((n3 = b1) &
        (Tavg < avg3)) : b1;
        TRUE : n0;
        esac;
        ((T0 > T2) & (T2 < T1))&((P0 > P2) & (P2 < P1))&
        ((TNP0 > TNP2) & (TNP2 < TNP1)) & (nb2 != b2) :
        case
        (n1 != b2) & (n2 != b2) & (nb2 != b2) : b2;
        ((n1 = b2) & (Tavg < avg1)) & ((n3 = b2) &
        (Tavg < avg3)) : b2;
        TRUE : n0;
        esac;
        TRUE : n0;
        esac;
        TRUE : n0;
    esac;
INIT
    T0 >= 41.0 & T0 <= 56.0
```

4 Verification of the DTM Algorithm

4.1 Experimental Setup

We used the version 1.0 of the nuXMV model checker along with the Windows 8.1 Professional OS running on a i3 processor, 2.93GHz(4 CPUs), with 4 GB memory for our experiments. In order to assume realistic values of temperatures for our experimentation, we used the temperature range between 41°C and 56°C for a single core as has been reported in [38]. The verification is done for a 9 x 9 grid of nodes (cores) with all of them running processes as described in the previous section. The complete model contains 81 processes.

4.2 Functional Verification

We have done the functional verification of the DTM algorithm by verifying the following properties using the nuXmv's bounded model checking (BMC) support for *real* numbers:

4.2.1 Deadlock

A deadlock state in a system leads to an undesired cyclic behavior. In case of DTM, deadlock happens if the temperature of some core x is greater than the estimated temperature and it is unable to exchange its load with some other core. This behavior could result in the creation of thermal hot spots across the chip. In order to make sure that the DTM algorithm is free of deadlocks, the following property needs to be satisfied:

$$G(core_x.T_0 > core_x.T_{avg} \rightarrow F(core_x.T_0 <= core_k.T_0))$$

This property checks that any core having temperature greater than the average temperature will eventually get a reduction in temperature.

4.2.2 Liveliness

The liveliness property in a system makes sure that the system returns to its good working or desired state. In our verification, we have defined liveliness using the following specification :

$$G(core_x.T_0 < core_x.T_{avg} \rightarrow X(core_x.n = x))$$

This property states that if the temperature of a core is less than the estimated average temperature of the core, then in the very next state, the core does not need to migrate tasks to its neighbors.

4.2.3 Stability

Stability is one of the most important properties for any DTM algorithm. In the given algorithm, stability is attained when the temperature of all the cores will eventually be less than or equal to the estimated average temperature of the chip.

$$GF((core_0.T_0 <= core_0.T_{avg})\&(core_1.T_0 <= core_1.T_{avg})........\&(core_{80}.T_0 <= core_{80}.T_{avg}))$$

For our 9 x 9 grid, it means that eventually, there would be a state where all the cores of the system have a temperature that does not exceed the estimated average temperature.

4.2.4 Verification of Temperature Estimation Algorithm

An interesting observation is the estimation of the average chip temperature using the Eq. 1. The graphs below show the average chip temperature estimation behavior of six cores, each corresponding to one of the six different neighbor configurations in our grid.

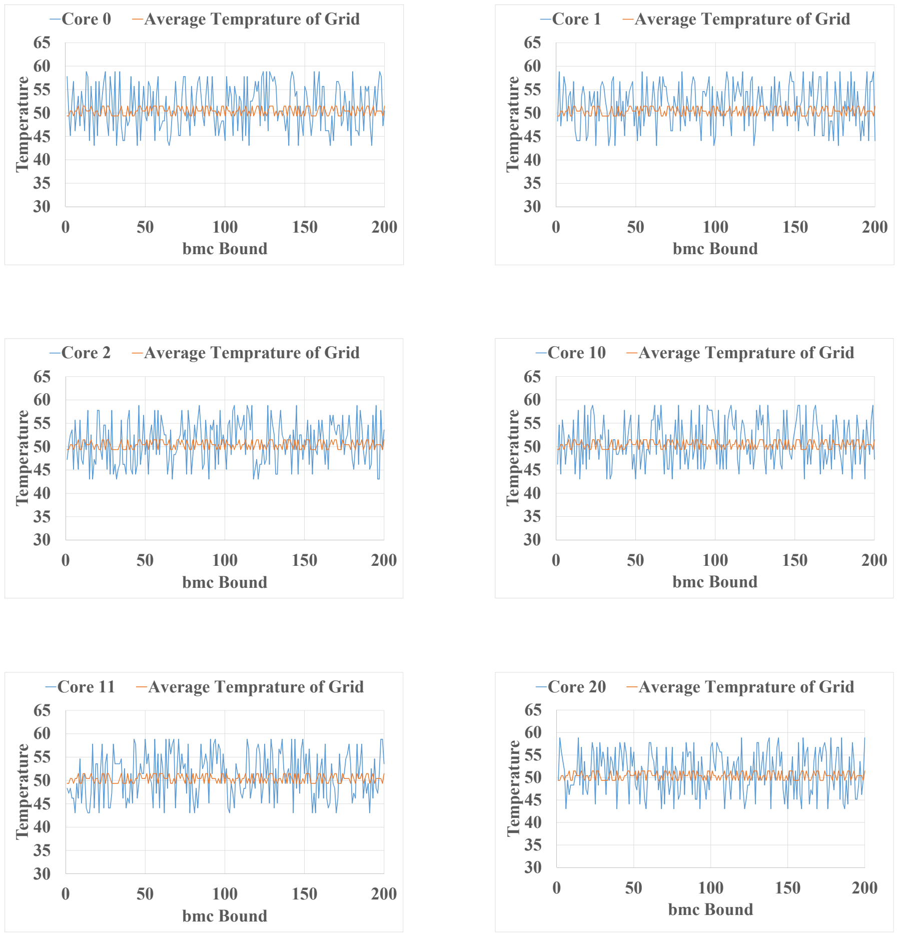


Fig. 4: Temperature Estimation in °C

Initially, each core sees the initial value of the T_{avg} as the average temperature of the core. The cores then estimate the temperature of the chip with the help of underlying average tracking algorithm in the DTM. For illustration purposes, we have shown the actual average temperature of the grid, and the estimated temperature of the selected cores on the same plot. It shows that the average estimation algorithm making use of the temperature information form the neighboring node gives a good average estimate of the overall chip temperature, confirming the functionality of the average estimation algorithm. Moreover the estimated average by different cores is also following a similar pattern.

The verification times and the memory consumption for some of the functional properties, verified in this work, are given in Table 1. The time measurements in Table 1 is done by using nuXmv function `time`.

Properties	Core	Module	Memory Usage (MBs)	Time (s)
Liveliness Property	0	n2_3	1015.69	745.67
Liveliness Property	1	n3_4	1051.71	751.25
Liveliness Property	2	n3_5	1025.64	749.65
Liveliness Property	10	n4_6	1041.52	758.75
Liveliness Property	11	n4_7	1031.74	781.85
Liveliness Property	20	n4_8	1033.85	790.65
Deadlock Property	0	n2_3	1351.41	1245.59
Deadlock Property	1	n3_4	1325.35	1235.61
Deadlock Property	2	n3_5	1315.63	1241.91
Deadlock Property	10	n4_6	1359.54	1249.71
Deadlock Property	11	n4_7	1359.54	1239.41
Deadlock Property	20	n4_8	1343.51	1251.11
Stability Property	-		2051.51	2253.56

Table 1: Timing and memory resources for some of the properties verified by our technique.

4.3 Timing Verification

The functional properties, verified in the previous section, have been verified for non-deterministic initial temperature values. In this section, we verified various timing related properties for specific scenarios, with particular initial temperatures. In order to measure the time stamps between the states transition, we have used built-in nuXmv commands `execute_trace` and `execute_partial_trace`. Table 2 shows the time to stability for different selected cores (one from each module) under 16 possible conditions. Here, `n2_3`, `n3_4`, `n3_5`, `n4_6`, `n4_7` and `n4_8` represents different modules. T_0 represents the temperature of the tested core and T_1, T_2, T_3 and T_4 represent the temperature of neighbors of the tested core. The first case `nb0` in Table 2, represents the case when the temperature

Scenarios	Experimental Setup					n2_3	n3_4	n3_5	n4_6	n4_7	n4_8
	T0	**T1**	**T2**	**T3**	**T4**	**0**	**1**	**2**	**10**	**11**	**20**
nb0	56	🟦	🟦	🟦	🟦	49	52	51	63	67	62
nb1	56	56	🟦	🟦	🟦	53	53	54	61	63	61
nb2	56	🟦	56	🟦	🟦	61	57	55	71	73	69
nb3	56	🟦	🟦	56	🟦	-	20	57	72	79	80
nb4	56	🟦	🟦	🟦	56	-	-	-	89	87	91
nb12	56	56	56	🟦	🟦	63	59	53	94	92	85
nb13	56	56	🟦	56	🟦	62	61	54	91	98	96
nb14	56	56	🟦	🟦	56	-	57	51	107	114	113
nb23	56	🟦	56	56	🟦	62	62	59	117	112	105
nb24	56	🟦	56	🟦	56	-	58	57	103	109	112
nb34	56	🟦	🟦	56	56	-	60	65	100	105	102
nb123	56	56	56	56	🟦	-	-	-	115	120	117
nb124	56	56	56	🟦	56	-	-	-	119	124	118
nb134	56	56	🟦	56	56	-	-	-	121	125	131
nb234	56	🟦	56	56	56	-	-	-	132	129	141
nb1234	56	56	56	56	56	-	-	-	129	131	137

Table 2: Time to stability for different test scenarios.

of all the neighbor cores is less than the threshold. Similarly, the `nb1234` represents the case when the temperature of all neighbor cores, i.e., 1,2,3 and 4, exceed the threshold. Whereas, the other cases represent the intermediate possibilities between these extreme scenarios. It can be seen from Table 2 that the maximum time to reach stability, 141s, is taken when the given core, of type `n4_8`, and three of its neighbors are at a temperature of 56°C.

5 Conclusion

This paper presents the formal verification of both functional and timing properties of a recent dDTM technique [20] for on-chip many-core systems using the nuXmv model checker. Due to the ability to handle *real* numbers and the powerful verification methods, based on SAT and SMT solvers, in nuXmv, we have been able to gain many new insights into the given algorithm. The analyzed model has 81 cores and the analysis is done within the range of 41 to 56°C. To the best of our knowledge, such a big model cannot be handled rigorously by simulation based testing. We plan to extend this work by proposing a common ground to analyze and compare dDTM schemes, such as [18, 39, 13, 40], both in terms of functional and timing properties.

References

[1] Schauer, B.: Multicore processors–a necessity. ProQuest discovery guides (2008) 1–14

[2] Wyngaard, J., Inggs, M., Collins, J., Farrimond, B.: Towards a many-core architecture for hpc. In: Field Programmable Logic and Applications. (2013) 1–4

[3] Brooks, D., Martonosi, M.: Dynamic thermal management for high-performance microprocessors. In: High-Performance Computer Architecture, IEEE (2001) 171–182

[4] Donald, J., Martonosi, M.: Techniques for multicore thermal management: Classification and new exploration. In: ACM SIGARCH Computer Architecture News. Volume 34., IEEE Computer Society (2006) 78–88

[5] Kong, J., Chung, S.W., Skadron, K.: Recent thermal management techniques for microprocessors. ACM Comput. Surv. **44**(3) (2012) 13:1–13:42

[6] Yang, J., Zhou, X., Chrobak, M., Zhang, Y., Jin, L.: Dynamic thermal management through task scheduling. In: Performance Analysis of Systems and software. (2008) 191–201

[7] Mukherjee, R., Memik, S.O.: Physical aware frequency selection for dynamic thermal management in multi-core systems. In: Computer-aided Design, ACM (2006) 547–552

[8] ITRS: http://www.itrs.net/Links/2013ITRS/2013Chapters/2013Overview.pdf (2014)

[9] Kadin, M., Reda, S., Uht, A.: Central vs. distributed dynamic thermal management for multi-core processors: Which one is better? In: Great Lakes Symposium on VLSI, ACM (2009) 137–140

[10] Donald, J., Martonosi, M.: Techniques for multicore thermal management: Classification and new exploration. In: Computer Architecture. (2006) 78–88

[11] Singh, A., Shafique, M., Kumar, A., Henkel, J.: Mapping on multi/many-core systems: Survey of current and emerging trends. In: Design Automation Conference (DAC), ACM / EDAC / IEEE. (2013) 1–10

[12] Ebi, T., Kramer, D., Karl, W., Henkel, J.: Economic learning for thermal-aware power budgeting in many-core architectures. In: Hardware/Software Codesign and System Synthesis, ACM (2011) 189–196

[13] Shafique, M., Henkel, J.: Agent-based distributed power management for kilo-core processors. In: Computer-Aided Design, IEEE (2013) 153–160

[14] Weiss, G., ed.: Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence. MIT Press (1999)

[15] Nath, R., Carmean, D., Rosing, T.S.: Power modeling and thermal management techniques for manycores. In: Computers and Communications, IEEE (2013) 740–746

[16] Salami, B., Baharani, M., Noori, H.: An adaptive temperature threshold schema for dynamic thermal management of multi-core processors. In: Computer Architecture and Digital Systems. (2013) 119–120

[17] Yun, B., Shin, K.G., Wang, S.: Predicting thermal behavior for temperature management in time-critical multicore systems. In: IEEE Real-Time and Embedded Technology and Applications Symposium. (2013) 185–194

[18] Ebi, T., Faruque, M., Henkel, J.: Tape: Thermal-aware agent-based power econom multi/many-core architectures. In: Computer-Aided Design. (2009) 302–309

[19] Ge, Y., Malani, P., Qiu, Q.: Distributed task migration for thermal management in many-core systems. In: Design Automation Conference, ACM (2010) 579–584

[20] Liu, Z., Huang, X., Tan, S.D., Wang, H., Tang, H.: Distributed task migration for thermal hot spot reduction in many-core microprocessors. In: ASIC. (2013) 1–4

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

[22] Chen, F., Cao, Y., Ren, W.: Distributed computation of the average of multiple time-varying reference signals. In: American Control Conference. (2011) 1650–1655

[23] Dunn, D.: Intel delays montecito in roadmap shakeup. EE Times, Manufacturing/Packaging (Oct 2005)

[24] Drechsler, R., et al.: Advanced Formal Verification. Volume 122. Springer (2004)

[25] Norman, G., Parker, D., Kwiatkowska, M., Shukla, E., Gupta, R.: Using probabilistic model checking for dynamic power management. Formal Aspects of Computing **17** (2003) 202–215

[26] Shukla, S., Gupta, R.: A model checking approach to evaluating system level dynamic power management policies for embedded systems. In: High-Level Design Validation and Test Workshop, IEEE. (2001) 53–57

[27] Lungu, A., Bose, P., Sorin, D.J., German, S., Janssen, G.: Multicore power management: Ensuring robustness via early-stage formal verification. In: Formal Methods and Models for Codesign, IEEE (2009) 78–87

[28] Holzmann, G.J.: The model checker spin. IEEE Transactions on software engineering **23**(5) (1997) 279–295

[29] Lamport, L.: Time, clocks, and the ordering of events in a distributed system. Commun. ACM **21**(7) (1978) 558–565

[30] Ismail, M., Hasan, O., Ebi, T., Shafique, M., Henkel, J.: Formal verification of distributed dynamic thermal management. In: Computer-Aided Design, IEEE (2013) 248–255

[31] Clarke, Jr., E.M., Grumberg, O., Peled, D.A.: Model Checking. MIT Press (1999)

[32] Cavada, R., Cimatti, A., Dorigatti, M., Griggio, A., Mariotti, A., Micheli, A., Mover, S., Roveri, M., Tonetta, S.: The nuxmv symbolic model checker. In: Computer Aided Verification. Volume 8559 of LNCS. Springer (2014) 334–342

[33] Liu, Z., Xu, T., Tan, S.D., Wang, H.: Dynamic thermal management for multi-core microprocessors considering transient thermal effects. In: Design Automation Conference. (2013) 473–478

[34] nuXmv: https://nuxmv.fbk.eu/ (2014)

[35] NuSMV: http://nusmv.fbk.eu/ (2014)

[36] MathSAT 5: http://mathsat.fbk.eu/ (2014)

[37] Wolfram: http://functions.wolfram.com/ComplexComponents/Sign/21/01/01/ (2014)

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

[39] Henkel, J., Ebi, T., Amrouch, H., Khdr, H.: Thermal management for dependable on-chip systems. In: Asia and South Pacific Design Automation Conference. (2013) 113–118

[40] Khdr, H., Ebi, T., Shafique, M., Amrouch, H., Henkel, J.: mdtm: multi-objective dynamic thermal management for on-chip systems. In: Design, Automation & Test in Europe. (2014) 330