Hindawi Publishing Corporation
VLSI Design
Volume 2016, Article ID 8093614, 16 pages
http://dx.doi.org/10.1155/2016/8093614

Research Article

A Cache System Design for CMPs with Built-In Coherence Verification

Mamata Dalui[1] and Biplab K. Sikdar[2]

[1]*Department of Computer Science and Engineering, National Institute of Technology Durgapur, West Bengal 713209, India*
[2]*Department of Computer Science and Technology, Indian Institute of Engineering Science and Technology Shibpur, West Bengal 711103, India*

Correspondence should be addressed to Mamata Dalui; mamata.06@gmail.com

Received 22 December 2015; Revised 19 May 2016; Accepted 24 May 2016

Academic Editor: A. Postula

Copyright © 2016 M. Dalui and B. K. Sikdar. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

This work reports an effective design of cache system for Chip Multiprocessors (CMPs). It introduces built-in logic for verification of cache coherence in CMPs realizing directory based protocol. It is developed around the cellular automata (CA) machine, invented by John von Neumann in the 1950s. A special class of CA referred to as single length cycle 2-attractor cellular automata (TACA) has been planted to detect the inconsistencies in cache line states of processors' private caches. The TACA module captures coherence status of the CMPs' cache system and memorizes any inconsistent recording of the cache line states during the processors' reference to a memory block. Theory has been developed to empower a TACA to analyse the cache state updates and then to settle to an attractor state indicating quick decision on a faulty recording of cache line status. The introduction of segmentation of the CMPs' processor pool ensures a better efficiency, in determining the inconsistencies, by reducing the number of computation steps in the verification logic. The hardware requirement for the verification logic points to the fact that the overhead of proposed coherence verification module is much lesser than that of the conventional verification units and is insignificant with respect to the cost involved in CMPs' cache system.

1. Introduction

The continual search for performance enhancement in computation has resulted in a variety of modifications in the processor design technique. This ultimately leads to the inevitable transition toward multicore architecture, the Chip Multiprocessors (CMPs), with thousands of processor cores on chip. The increasing number of cores in CMPs, however, puts threats on the reliability and dependability of a design [1]. A number of works [2–5] addressed these issues from different perspectives. Further, the low supply voltage in today's semiconductor technology narrows down the noise margin and increases the susceptibility to various factors causing transient faults [6] in CMPs.

Most of the fault tolerant schemes reported so far for CMPs are based on the spatial redundancy techniques which may not be effective for faults in on-chip hardware components [2]. Although the cache and memory components are protected by Error Correcting Codes and other techniques, the logic circuits commonly serving the multiple cores are error prone [2].

A CMPs memory subsystem is made up of multilayer caches including private cache for each processor core. It demands very efficient realization of cache coherence protocols. The cache coherence controller (CC) is dedicated to ensuring coherency of shared data in the CMPs' cache system. Such a prime hardware component can also be subjected to fault as well as design defect. A fault in the CC has serious effect on the correctness of computation as well as on maintaining power efficiency of a system. The schemes proposed in the literature [2, 4, 7, 8], for ensuring coherency in CMPs with thousands of cores, incur huge communication overhead along the global wires. In [2], a verification logic has been proposed to detect errors in the coherence controller. It targets a system realizing snoopy protocol.

The snoopy protocols are easy to implement but are not so scalable [9]. For large scale CMPs, updating and invalidating caches, following snoop based protocols, become impractical

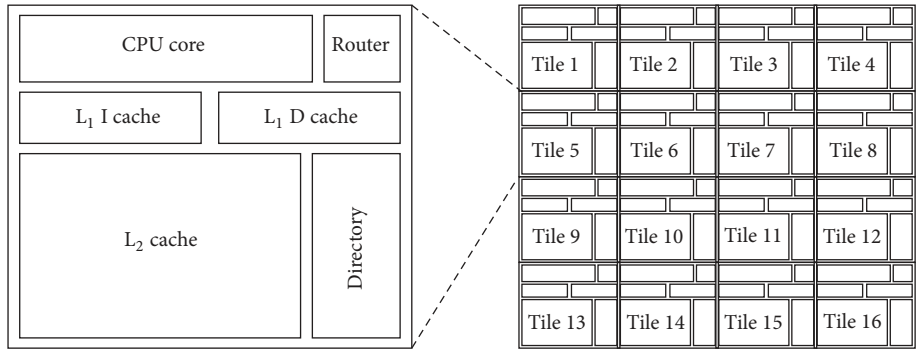


Figure 1: The tiled CMPs organization.

[10]. Several variants of directory based coherence protocols have been proposed in [5, 10, 11]. However, the verification of cache data inconsistencies resulting from defects in such systems is yet to be addressed.

The above concerns motivate us to design an effective cache system for CMPs by developing a scheme to determine the accuracy in maintaining data consistency in the CMPs' cache system realizing directory based protocol. It targets design of built-in logic for verification of cache coherence that can function at speed and can be cost effective. To explore such a design, we consider cellular automata (CA) tool [12] invented by John von Neumann in the 1950s. As CA can handle large volume of data and efficiently be employed to make a decision, a CA based built-in verification logic is proposed to ensure accuracy in functioning of the directory based cache system in tiled CMPs [11]. A special class of CA structure referred to as the single length cycle 2-attractor cellular automata (TACA) is introduced for the design. The TACA analyses the status of CMPs' cache updates and settles to an attractor state (point state) indicating any faulty recording of cache line status and the sharing vector [9] stored in the directory of cache system. The hardware realization of the CA based design enables quick decision on the cache coherency. Further, the introduction of segmentation of the CMPs' processor pool assures better efficiency of the design. It reduces the computation steps while making decision on inconsistency, if any, in each cache update. The basic concept of the CA based solution is reported in [13, 14]. The precise contribution of this paper can be summarized as follows:

(i) A built-in verification logic for CMPs cache system, realizing directory based protocol, is proposed. For ease of understanding, the design is detailed out with the basic 3-state MSI protocol. However, the methodology proposed can be applicable for MESI, MOSI, MOESI, and others.

(ii) The verification logic is developed around an unconventional tool, called cellular automata (CA). The modular structure of the CA is exploited to enable scalable design.

(iii) Design of a high-speed verification unit for cache, harnessing the feature of CA to memorize information, is reported.

(iv) The verification unit is realized for full map as well as for the limited directory based cache systems.

(v) Relevant CA theory has been developed to provide the theoretical basis of the cache system design.

(vi) Experimental result establishing the claim has been reported.

The following section (Section 2) highlights the coherence issues in CMPs' cache system. Section 3 narrates CA theory relevant for the current design. The CA based verification unit is introduced in Section 4, and Section 5 describes the design in detail. The hardware realization and delay overhead reduction through introduction of segmentation are reported in Sections 6 and 7, respectively. A test structure that memorizes the inconsistent recording of cache line states during the processors' references to a memory block is reported in Section 8. The simulation results establishing the effectiveness of the CA based design and its hardware requirement are reported in Section 9. A sketch of the verification unit for limited directory based system is shown in Section 10. In Section 11, we provide a brief on the related works. Section 12 concludes the paper.

2. Cache Coherence in CMPs

In Chip Multiprocessors (CMPs) with a large number of on-chip cores, the shared bus cannot be a good choice due to area overhead and bus contention [11]. The alternative to shared bus is a tiled CMPs architecture. A tile is composed of an array of identical building blocks connecting the cores with point-to-point unordered network [11, 15]. In this work, we consider the ATAC processor architecture [15] which uses a tiled multicore architecture (Figure 1). It is a low-latency, energy-efficient, global, and long distance communication network. Each core in ATAC consists of private L_1 and shared L_2 caches. The L_2 typically follows Nonuniform Cache Access (NUCA). An L_1 cache miss in ATAC generates coherence messages. The other L_1 cache line states of the system are updated in accordance with the coherence messages.

In general, for large scale CMPs, the directory based cache coherence system is desirable [11]. A directory, in directory based system, is a collection of sharing vectors [9]. Each vector corresponds to a data block B and maintains pointers

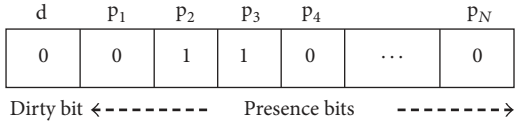


Figure 2: Sharing vector for full-map directory.

to the processors that have the cached copies of B. The directory also stores the state of cached copies in L_1 caches. The organization of a sharing vector is shown in Figure 2. The "d" specifies the dirty bit and d = 1 (true) implies that there is a cache with the latest copy of B and the main memory copy is an invalid copy. The p_i's are the presence bits. $p_i = 1$ implies that the processor P_i is having a cached copy of B. In the current design of coherence verification logic/unit, we consider a distributed directory based system.

Figure 3 is to describe the logical steps followed to maintain the cache coherence in CMPs, realizing the distributed directory based protocol, on a read miss. It consists of processors P_i, P_j, and P_k with local memory modules L_{2i}, L_{2j}, and L_{2k}, respectively. The distributed directories D_i, D_j, and D_k are also local to the corresponding processors P_i, P_j, and P_k. In such a system, if a processor (P_i) requests block B, which is not present in P_i's L_1 (C_i), P_i encounters a read miss and consults its communication assistance unit to find the home (say processor P_j, i.e., L_{2j}) of block B. The request then goes to P_j's site and the directory D_j is consulted. If the sharing vector of B, stored in D_j, shows that d = 0, that is, the L_{2j} has the valid copy of block B, then P_j sends block B to P_i and updates the sharing vector corresponding to B at D_j by setting $p_i = 1$. On the other hand, if d = 1, that is, the copy of block B at L_{2j} is not a valid copy and P_k is having the dirty copy (as shown in Figure 3), then P_k sends block B to P_i and also writes it back to L_{2j} (home site for block B). P_j then updates the sharing vector of block B at D_j. Here we have assumed a 4-hop communication (request to directory → reply with owner → request to owner → reply to requester) to resolve read miss. However, for a system realizing 3-hop communication (request to directory → forward to owner → reply to requester), as in [16], the proposed verification logic is also compatible.

3. CA Preliminaries

A cellular automaton (CA) consists of a number of cells organized in the form of lattice. It can be viewed as an autonomous finite state machine (FSM). In a two-state 3-neighborhood CA, each CA cell stores either 0 or 1 that refers to the present state (PS) S_i^t at time t of the cell i and the next state (NS) of the cell i at $(t + 1)$ is

$$S_i^{t+1} = f_i\left(S_{i-1}^t, S_i^t, S_{i+1}^t\right), \tag{1}$$

where S_{i-1}^t and S_{i+1}^t are the present states of the left neighbor and right neighbor of ith cell at time t and f_i is the next state function (Figure 4). The state of all the cells $S^t = (S_1^t, S_2^t, \ldots, S_n^t)$ at t is the present state of the CA. Therefore, the next state of an n-cell CA is $S^{t+1} = (f_1(S_0^t, S_1^t, S_2^t), f_2(S_1^t, S_2^t, S_3^t), \ldots, f_n(S_{n-1}^t, S_n^t, S_{n+1}^t))$.

Table 1: Next state functions.

PS	111	110	101	100	011	010	001	000	Rule
RMT	(7)	(6)	(5)	(4)	(3)	(2)	(1)	(0)	
NS	1	1	1	1	1	1	1	1	254
NS	1	1	1	1	1	1	1	0	255

The next state function f_i of the ith CA cell can be expressed in the form of a truth table (Table 1). The decimal equivalent of the 8 outputs (NS) is called "rule" R_i of the cell [12]. There are 256 rules in 2-state 3-neighborhood CA. Two such rules 254 and 255 are illustrated in Table 1. The first row lists the possible 2^3 (8) combinations of present states S_{i-1}^t, S_i^t, and S_{i+1}^t. The last two rows indicate the next states of the ith cell at time $(t + 1)$, defining the rules 254 ($f_i = S_{i-1} + S_i + S_{i+1}$) and 255 ($f_i = 1$), respectively.

The rule vector $R = \langle R_1, R_2, \ldots, R_i, \ldots, R_n\rangle$ configures the cells of a CA. If all the R_i's are the same, that is, $R_1 = R_2 = \cdots = R_n$, the CA is a uniform CA; otherwise, it is a nonuniform/hybrid CA [17]. In Figure 4, the left (right) neighbor of the leftmost (rightmost) terminal cell is permanently fixed to 0-state. It is a null boundary CA.

Definition 1 (RMT). A combination of present states shown in the 1st row of Table 1 is the Min Term of a 3-variable S_{i-1}^t, S_i^t, and S_{i+1}^t switching function and is referred to as the Rule Min Term (RMT).

Column 011 of Table 1 is the 3rd RMT. The next states corresponding to this RMT are 1 for both rules 254 and 255.

Definition 2 (reversible and irreversible CA). A CA is reversible if its states form only cycles in the state transition diagram (all states are reachable); otherwise, it is irreversible (Figure 5).

Definition 3 (attractor and attractor basin). A set of states of a CA forms loop (cycle) and is called *attractor*. An attractor (α) forms an α-basin with the states that lead to the attractor.

The cycles (7→7 and 9→1→9) of Figure 5 are the two attractors of the CA $\langle 1, 236, 165, 69\rangle$. The 7-basin of the CA contains 12 states including the attractor state 7.

Definition 4 (depth). The depth of a CA is defined as the length of the longest path from a state to an attractor in the state transition diagram.

The depth of the CA shown in Figure 5 is 5 (2→10→6→4→5→7).

Definition 5 (active and passive RMT). An RMT x0y (x1y) in a CA rule is called passive (self-replicating) if the RMT x0y (x1y) is 0 (1). On the other hand, if an RMT x0y (x1y) is 1 (0), it is active (non-self-replicating).

For example, the RMT 0 (000) is 0 and RMT 2 (010) is 1 in 254 (Table 1); that is, these two RMTs are passive. However, RMT 0 in 255 is active as it is 1 (Table 1).

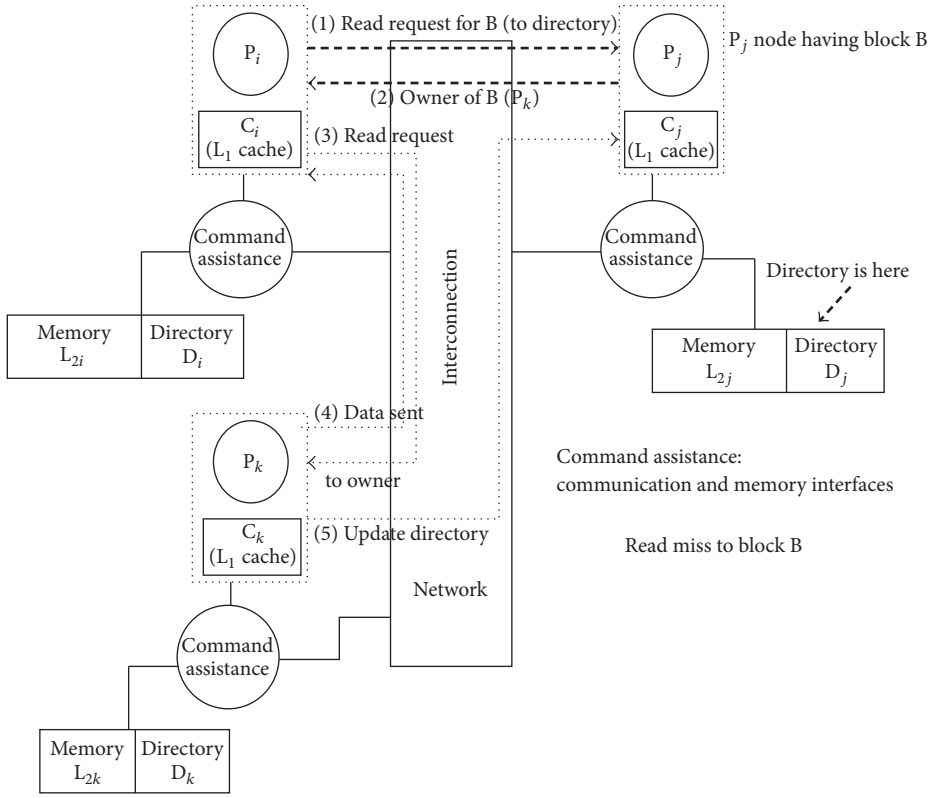


Figure 3: Directory based protocol: read miss to a block.

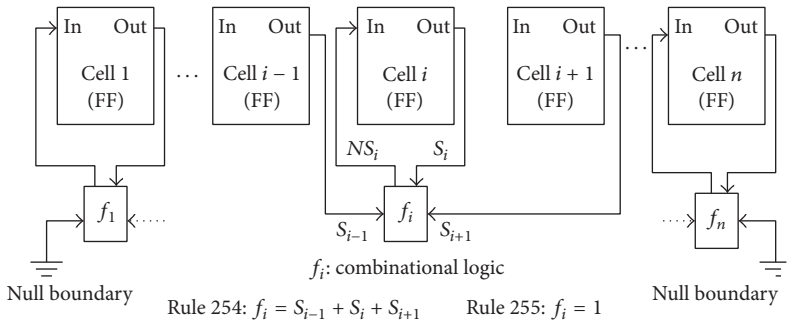


Figure 4: An n-cell null boundary CA.

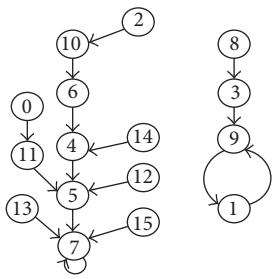


Figure 5: A 4-cell irreversible CA $\langle 1, 236, 165, 69\rangle$.

4. Overview of Verification Logic

A defect in the computing logic of the cache coherence controller (CC) can lead to faulty next state computation in CMPs. Even if the computing logic operates correctly, faulty recording of state(s) may also result due to fault(s) in the communication network of the CMPs cache system. These faulty recordings of state(s) introduce inconsistencies in the cache data states. The identification of cache data inconsistencies in such a system, while maintaining the cache coherence, is addressed in [2]. The solution reported detects errors in a CC. It targets a snoopy protocol based cache system and involves complex data structures as well as computation intensive steps. In [18], we also report design of verification unit for the CC working in a snoopy protocol based system.

The cache coherence protocol (ACKwise) in ATAC processor architecture is the coupling of directory and snoopy protocol named ACKwise [15]. In addition to the probable defects in CC and faults in the communication network, the faulty update of sharing vector is a major concern in ATAC.

Table 2: Cache sharing vector update.

Current sharing vector (1)	Event (2)	Desired sharing vector (3)	Faulty sharing vector (4)	Fault effect (5)	Cases (6)
All p's are 0s	P_i reads	p_i is 1 and all others are 0s	All p's are 0s	Faulty (F)	Case 1
			p_j is 1 and all others are 0s	Faulty (F)	Case 2
	P_i writes	p_i is 1 and all others are 0s	All p's are 0s	Faulty (F)	Case 1
			p_j is 1 and all others are 0s	Faulty (F)	Case 2
All p's are 1s	P_i writes	p_i is 1 and all others are 0s	p_j is 1 and all others are 0s	Faulty (F)	Case 2
			p_i & p_j are 1 and others are 0s	Faulty (F)	Case 3
			p_i is 1 and others are 1s & 0s	Faulty (F)	Case 3
p's are 1s & 0s	P_i reads	p_i is 1 and all others are 1s & 0s	p_i is 0 and others are 1s & 0s	Faulty (F)	Case 1
	P_i writes	p_i is 1 and all others are 0s	p_i is 1 and others are 1s & 0s	Faulty (F)	Case 3
p_j is 1 and all others are 0s	P_i writes	p_i is 1 and all others are 0s	p_i & p_j are 1 and others are 0s	Faulty (F)	Case 3

The sharing vector may also be subjected to faults, even if the sharing status of a cache block is recorded correctly thus making the coherence verification process hard to realize. Incorporating separate verification units for sharing status and sharing vector would be extremely costly. Therefore, we formulated the problem of coherence verification in ATAC like architectures by modelling it as the verification of the compatibility of sharing status and sharing vector on each cache state update. To prove the effectiveness of the proposed cellular automata (CA) based verification logic, we consider the ATAC (tiled CMPs) architecture realizing the directory based cache coherence system with MSI protocol. However, this scheme is also applicable for MESI/MOSI/MOESI protocol based designs.

Figure 6 describes the basic 3-state MSI protocol. Table 2 displays the effect of faulty noting in sharing vector, resulting in faulty ("F") state (column 5) with full-map directory. The entry "All p's are 0s," in column 1 of the first row, represents that none of the processors'/tiles' L_1 caches has a copy of block B. On the other hand, the entry "All p's are 1s," in column 1 of the second row, represents that all the processors' caches have a copy of B. Similarly, the entry "p_i is 1 and all others are 0s" indicates only processor P_i has a copy of block B and there is no other cached copy of B.

On a read/write operation (event) of a processor P_i (Figure 3), the corresponding sharing vector for block B is noted in column 3 of Table 2. The contents of column 4 represent the possible faulty noting at the sharing vector. For example, row 1 of column 1 notes the sharing vector prior to "P_i read". Initially, B does not have a cached copy (represented by "All p's are 0s"). On an event of read by P_i (noted in column 2 of row 1), the desired sharing vector is "p_i is 1 and all others are 0s" (column 3 of row 1). All the possible faulty recordings of the sharing vector are shown in column 4. Column 5 notes the effect of fault ("faulty (F)"). Classification of faulty cases is noted in column 6. The consideration of columns 1, 4, and 5 of Table 2 indicates that the proposed fault detection unit should respond as "F" for the states of cached copies of B (cache lines for B) when the following cases occur:

Case 1. Processor P_i reads/writes block B, but P_i's presence bit (p_i) in sharing vector for B is not updated to "1"; that is, it is still "0".

Case 2. P_i reads/writes and some other processors' (P_j's) presence bit(s) (p_j's) is (are) set to "1," instead of P_i's presence bit (p_i).

Case 3. P_i writes and some other processors' (P_j's) presence bits are not updated; that is, p_j's remain "1".

It is to be noted that Cases 2 and 3 include all cases of all possible faults that affect more than one bit. The proposed CA based logic (shown in Figure 7), to realize the verification in cache coherence system, therefore, is designed so that it can correctly respond with either "NF" (nonfaulty) or "F" (faulty) following the above three cases. It employs an n-cell CA for CMPs with n private (L_1) caches ($C_1, C_2, \ldots, C_n$, where C_i is the cache attached to processor P_i and ith CA cell corresponds to the cache C_i). Now, with each read/write operation, the cache line state (sharing status $Q_{i0}Q_{i1}$ assuming MSI protocol) as well as the presence bits (p's) in the sharing vector are updated. In a fault-free system, the update of sharing status and sharing vector should be compatible. Therefore, the sharing status, to be more specific, the MSB of sharing status (in the current design), and the presence bits of the sharing vector are fed as input to the verification unit to form n-bit compatibility status (CS). The n-cell CA of the verification unit is then run for certain number (t) of time steps with CS as the seed. The CA settles either in an attractor designated as X_1, corresponding to "NF" for nonfaulty recording, or in an attractor X_2, corresponding to "F" (faulty) for the instance of a fault. Therefore, by observing the attractor ("NF" or "F"), the fault in coherence controller logic can be detected. Observing an attractor, however, is reduced to sensing of LSB of the attractor (least significant cell of the CA) to detect fault in the system. The steps in realizing the verification unit for a full-map directory are summarized in the following algorithm.

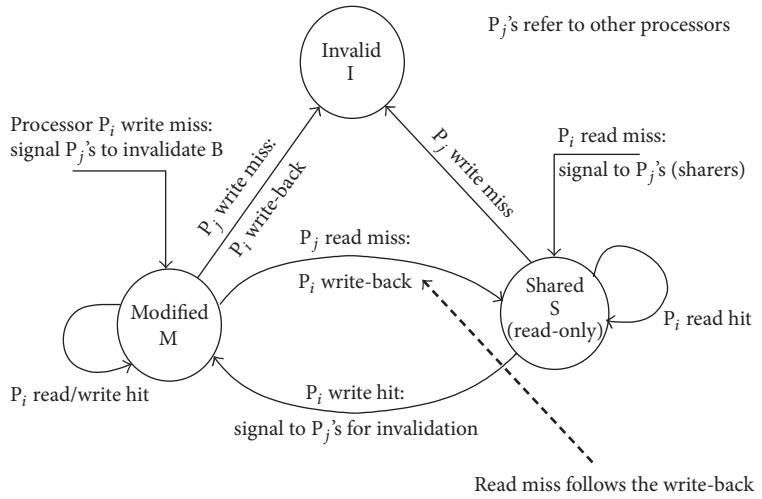


Figure 6: State transition diagram of MSI protocol.

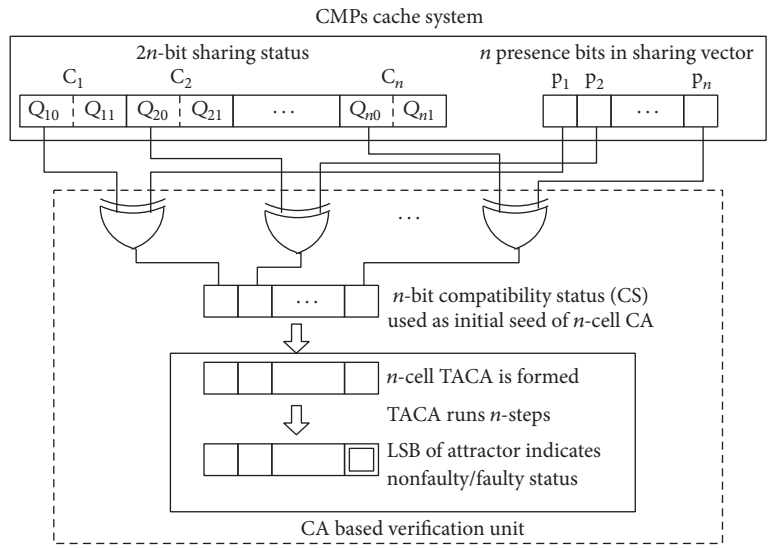


Figure 7: CA based verification unit for full-map directory.

Algorithm 1 (FUNCTION-CAVU$_{\text{FULL-DIR}}$).

Input. Presence bits and MSBs of sharing status of cache lines.

Output. Decision on presence of fault.

(1) Form the n-bit compatibility status (CS) where $CS_i = Q_{i0} \oplus p_i$, $\forall i = 1$, to n.

(2) Run the CA selected for verification logic for t time steps ($t = (n - 1)$ for the current design) with CS as the initial state (seed). It reaches the attractor state.

(3) Check LS cell of the CA (LSB of attractor). If it is "0", then record of sharing status and sharing vector is fault-free; otherwise, it is faulty.

5. The Design of CAVU$_{\text{FULL-DIR}}$

The cellular automata (CA) based solution for the verification of cache inconsistencies, introduced in the earlier section, demands the following:

(i) The CA constructed for the verification unit CAVU$_{\text{FULL-DIR}}$ should form single length cycle attractors.

(ii) The number of attractors should be minimum, preferably two attractors: one (X_1) corresponds to "NF" (nonfaulty) and the other (X_2) corresponds to "F" (faulty) and the CA should correctly report all the faulty cases of Table 2.

(iii) The attractors X_1 and X_2 should differ at least at one position (say, LSB) so that the decision on "NF" or "F" (sensing a single bit of the attractor, that is, least significant cell of the CA) can be taken at speed.

The effect of fault propagates through the CA as it strides over time and induces LSB of the attractor. Therefore, the incidence of fault is translated as a switch from X_1 attractor basin to X_2-basin (say from 0-basin of Figure 8 to its 1-basin). The following subsections characterize the single length cycle attractor CA that can be employed for the current design.

5.1. Single Length Cycle Attractor. The next state of a single length cycle attractor is the attractor itself. In a single length cycle attractor CA, for at least one RMT (Section 3) of each cell rule R_i of R (CA), the cell i is passive (Definition 5). It implies that the state change in cell i is $d \rightarrow d$. This is summarized in the following property.

Property 1. A rule R_i can lead to the formation of single length cycle attractor(s) if at least one of its RMTs is passive; that is, at least one of the RMTs 0(000), 1(001), 4(100), or 5(101) is 0 and/or at least one of the RMTs 2(010), 3(011), 6(110), or 7(111) is 1 [19].

In [19], based on Property 1, the 256 CA rules are classified in 9 groups (groups 0–8). Rule 254 (11111110) is in group 5 as it follows Property 1 for 5 RMTs (RMTs 0, 2, 3, 6, and 7 are passive (Table 1)). A CA configured with the rules that maintain Property 1 for its RMTs is a probable CA with the single length cycle attractors [19]. The construction of single length cycle 2-attractor CA (Figure 8) can be followed from the theory of Reachability Tree for attractors introduced in [20].

5.2. Reachability Tree for Attractors. Reachability Tree for attractors (RTA) is a binary tree representing single length cycle attractors of a CA [20]. Each node is constructed with RMT(s) of a rule that follows Property 1. The left edge is the 0-edge and the right edge is 1-edge (Figure 9). For an n-cell CA, the number of levels in the tree is $(n + 1)$. Root node is at level 0 and the leaf/terminal nodes are at level n. The nodes at level i are constructed from the RMTs of $(i + 1)$th CA cell rule R_{i+1}. The decimal numbers within a node at level i represent the RMTs of the CA cell rule R_{i+1} based on which the cell $(i + 1)$ can change its state. The RMTs of a rule for which we follow 0-edge or 1-edge are noted in the bracket. The number of leaf nodes in an RTA denotes the number of single length cycle attractors of the CA and a sequence of edges from the root to a leaf node, representing an n-bit binary string, is the attractor state. The 0-edge and 1-edge represent 0 and 1, respectively. For example, the number of single length cycle attractor states in the CA $\langle 254, 254, 254, 254\rangle$ of Figure 9 is 2 (X_1 and X_2). The root node (level 0) of the RTA is constructed from passive RMTs 0, 2, and 3 as cell 1 (rule 11111110) can change its state following any one of the passive RMTs (null boundary). As the state of left neighbor of cell 1 is always 0, the passive RMTs 6 and 7 of rule 254 are the do not care RMTs for cell 1. Similarly, as the right neighbor of cell 4 is always 0, the passive RMTs 3, 5, and 7 are do not care for cell 4.

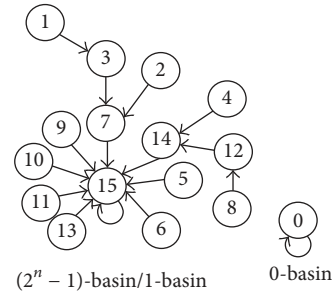


Figure 8: State transition diagram of $\langle 254, 254, 254, 254\rangle$.

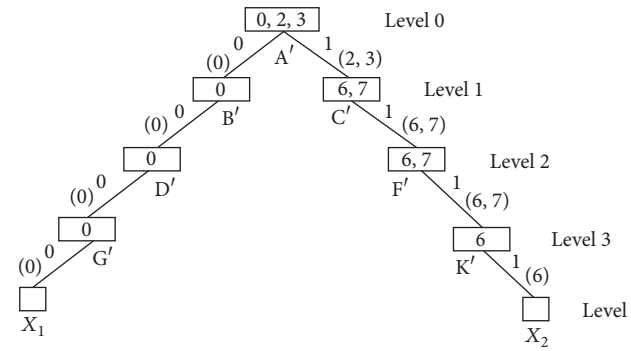


Figure 9: RT for attractors of $\langle 254, 254, 254, 254\rangle$.

The RMTs of two consecutive CA cell rules R_i and R_{i+1} are related while the CA changes its state [20]. This relationship between the RMTs of R_i and R_{i+1} is shown in Table 3. It implies that if the ith CA cell changes its state following RMT K_i, then $(i+1)$th cell changes its state following RMT K_{i+1}. For example, if the 1st cell in Figure 9 changes its state following RMT 0, then the 2nd cell changes its state following RMTs 0 and 1 (from Table 3).

The RTA of Figure 9 can be generalized to depict the attractors for an n-cell CA. The generalized RTA of the CA $\langle 254, 254, \ldots, 254\rangle$ is shown in Figure 10. The 0-edge at B′ of Figure 9 evolves to node D′ with the same set of RMTs ($\{0\}$); that is, nodes B′ and D′ are equivalent and, therefore, transition B′ to D′ is replaced by the transition B′ to B′ (Figure 10). Similarly, the transition C′ $\rightarrow$ F′ in Figure 9 is replaced by the transition C′ $\rightarrow$ C′ in Figure 10. Such transitions between equivalent states are true for level 1 to level $(n - 2)$. For the last cell of a CA ($\langle 254, 254, \ldots, 254\rangle$), some of the RMTs of B′ and C′ (e.g., RMT 1 of B′ and RMT 7 of C′) are do not care RMTs. Therefore, level $(n - 1)$ is shown separately (G′ and K′ in Figure 10). RTA of Figure 10 depicts that the n-cell uniform CA with rule 254 forms 2-single length cycle attractors (all 0s and all 1s).

5.3. CA Rule Selection for the $CAVU_{FULL\text{-}DIR}$. As described in the earlier section, the CA with only two single length cycle attractors X_1 and X_2 (TACA) can be the best choice for the current design. The following properties guide the proper

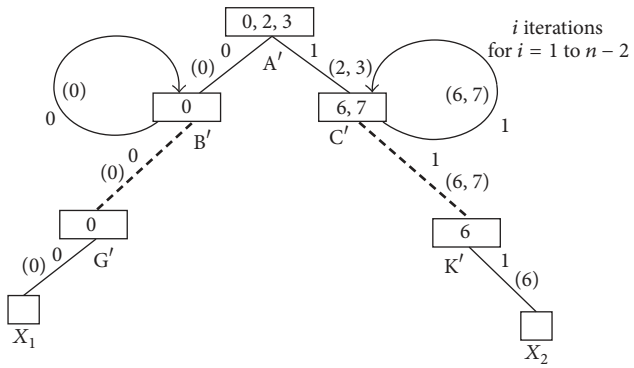


Figure 10: RT for attractors of ⟨254, 254, . . . , 254⟩.

selection of CA rules for the $\text{CAVU}_{\text{FULL-DIR}}$. To reduce the search space, the CA rules that form only single length cycle 2-attractors (TACA) for all lengths (n-cell) are identified. The theory reported in this subsection guides us to select appropriate TACA rules required for the current design.

Property 2. In 3-neighborhood null boundary, the n-cell uniform TACA should have either an all 0s or an all 1s attractor or both. The attractors (say, A_1 and A_2) differ in consecutive 2^0 or 2^1 or $2^{\lfloor \log n \rfloor}$ terminal bits [21].

Property 3 (see [21]). The rules of groups 3, 4, and 5 can only form single length cycle 2-attractor CA (TACA).

Property 4 (see [21]). For all the TACA rules, RMT 5 is an active RMT.

Property 5 (see [21]). For any TACA rule, at least one of RMTs 2 and 3 as well as at least one of RMTs 0 and 7 must be passive.

Theorem 6. *In a single length cycle uniform CA with all 0s attractor, the RMT 0 of the rule selected for CA must be passive.*

Proof. The root node of an RTA can only be formed with one or more RMTs of the RMT set {0, 1, 2, 3}. Among these, RMTs 0 and 1 can be 0 (passive) and RMTs 2 and 3 can be 1 (passive). For all 0s attractor, either of RMTs 0 and 1 should be passive. Now, the next cell RMT for RMT 0 is {0, 1} and it is {2, 3} for RMT 1 (Table 3). If only RMT 1 is passive (RMT 0 is active), its next cell RMTs (i.e., {2, 3}) having value "1" block the 0-edge (edge labelled with RMT value 0) of the RTA. Hence, RMT 0 must be passive for the continuity of 0-edge which corresponds to the all 0s attractor. □

Theorem 7. *In a single length cycle uniform CA with all 1s attractor, RMTs 3, 6, and 7 must be passive.*

Proof. In RTA, among the RMTs of root node, if RMT 0 and/or 1 is passive, the RTA follows a 0-edge and if 2 and/or 3 is passive, it follows 1-edge. So, for an all 1s attractor, the root node RMTs should be 2 and/or 3. For RMT 2 as passive, the next cell RMTs are {4, 5} (Table 3). RMTs 4 and 5, having value 0, cannot contribute to all 1s attractor. Now, for RMT 3 as passive RMT, the next cell RMTs are {6, 7}. So, for continuity of the 1-edge (edge labelled with RMT value 1), the RMT 7 must be passive. However, since for the last cell RMT 7 is do not care, RMT 6 must also be a passive RMT, hence the proof. □

Table 3: Relationship between RMTs of cell i and cell $(i+1)$ for next state computation.

RMT K_i of ith rule	RMTs K_{i+1} of $(i+1)$th rule
0/4	0, 1
1/5	2, 3
2/6	4, 5
3/7	6, 7

Table 4: CA rules for single length 2-attractor CA (TACA).

Group	Rule for TACA
3	38, 52
4	46, 106, 116, 120, 166, 180, 235, 249
5	174, 239, 244, 253, 254

Corollary 8. *In 3-neighborhood null boundary, the uniform CA constructed with only 16 rules (out of 256) can generate all 0s and all 1s single length cycle attractors.*

Proof. The proof is obvious as for all 0s attractor RMT 0 must be passive and for all 1s attractor RMTs 3, 6, and 7 are to be passive. Therefore, such CA rules vary in RMTs 1, 2, 4, and 5, leading to 16 possible combinations, that is, 16 possible CA rules. □

Theorem 9. *The uniform CA with rule 254 has only two single length cycle attractors (all 0s and all 1s states).*

Proof. The self-replicating RMTs of rule 254 (11111110) are RMTs 0, 2, 3, 6, and 7. That is, root node of the RTA of the uniform CA configured with rule 254 (Figure 10) contains the self-replicating RMTs 0, 2, and 3. Now, the RMTs for the next cell rule (following Table 3) are (0, 1) for RMT 0, (4, 5) for RMT 2, and (6, 7) for RMT 3. Since RMTs 1, 4, and 5 are non-self-replicating, they will not appear in the RTA. Only RMTs 0, 6, and 7 appear. So, there can be two sets of self-replicating RMTs, set 0 = {0} and set 1 = {3, 6, 7}, that create two distinct paths from the root to two leaf nodes (X_1 and X_2 in Figure 10), hence producing two attractors: all 0s ($X_1 = 000 \cdots 0$) and all 1s ($X_2 = 111 \cdots 1$). □

The above properties and theorems enable identification of 15 rules as the TACA rules (Table 4). Further, from Property 3 and Corollary 8, it can be concluded that there are only 5 rules 218, 234, 248, 250, and 254 that have only all 0s and all 1s single length cycle attractors. However, from the NSRT diagrams introduced in [21], it can be shown that only rule 254 forms a uniform TACA for all lengths. The other four rules form multilength cycle attractors along with the two (all 0s and all 1s) single length cycle attractors.

From the state transition diagram of the uniform CA configured with rule 254 (Figure 8), it can be observed that

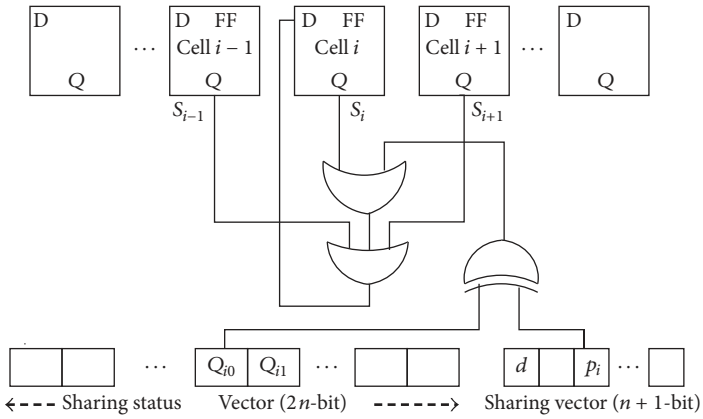


Figure 11: Hardware realization of $\text{CAVU}_{\text{FULL-DIR}}$.

the appearance of "1"(s) in initial state can act as a switch from 0-basin to $(2^n - 1)$-basin (referred to as 1-basin in the figure). The 4-cell CA $\langle 254, 254, 254, 254\rangle$, when initialized with all 0s seed, follows 0-basin (LSB "0" attractor); on the other hand, the CA when initialized with nonzero seed follows the 15-basin (1-basin), that is, with LSB 1 attractor. This behavior of rule 254 matches with the design requirement for $\text{CAVU}_{\text{FULL-DIR}}$. The design is detailed out in the following subsection.

5.4. The $CAVU_{FULL\text{-}DIR}$. Let the attractors of CA formed for all the cases of column 3 ("NF" cases) of Table 2 be X_1 (all 0s) and for other cases ("F" cases) the attractors are X_2 (all 1s). The best selection of CA rules is, therefore, such that

Cond 1: $A_1 = \{X_1\}$ belongs to "NF" and
$A_2 = \{X_2\}$ belongs to "F" are different.

For the current design, we consider a uniform CA with rule 254. In the "NF" case, the CA representing the cache system traces through the X_1-attractor basin (0-basin). Now, if there is a fault in the cache coherence system, the CA traces through the X_2-basin (1-basin). Hence, the LSB of attractor "0" signifies "nonfaulty" recording and "1" signifies a "faulty" recording.

While representing the states "M," "S," and "I" of a cache line in the sharing status, we consider 11 for "M," 10 for "S," and 00 for "I". If a cache line for B is in "M" or "S" state in the C_i (ith processor's cache), the ith bit of the sharing vector (p_i) corresponding to block B is 1. That is, in a nonfaulty case, the MSB of sharing status at C_i and the ith bit of the sharing vector (p_i) both are equal (either 1 or 0).

Theorem 10. *The MSB of sharing status (denoted as "11" for Modified, "10" for Shared, and "00" for Invalid) suffices to be considered for checking compatibility of sharing status with sharing vector.*

Proof. The cache line states when represented in the sharing status vector, the states "M," "S," and "I" are encoded as 11, 10, and 00, respectively. Whenever a block (say, block B) is in "M" state in one cache, the block's state in other caches should ideally be "I" for maintaining coherence. The cache holding the block in "M" state becomes a dirty sharer. Now, whenever, one or more caches have the block in "S" state; on read(s) the cache(s) (processor(s)) become clean sharer(s). In both cases of clean and dirty sharing, the presence bit(s) in the sharing vector should be set to "1". This "1"(s) in the presence bits conforms to the "1"(s) in the MSB of state code for "M" (11) or "S" (10). Hence, checking compatibility of the sharing status and the sharing vector can be performed by checking the MSB of sharing status and the presence bits. □

To ensure the correctness in recording of sharing status and the sharing vector at an update of data block B, the $\text{CAVU}_{\text{FULL-DIR}}$ accepts the compatibility status which is formed by performing XOR of the MSB (Q_{i0}) of 2-bit sharing status ($Q_{i0}Q_{i1}$) of B at C_i and the ith presence bit (p_i) of sharing vector for B as the initial state of ith cell of the CA selected for the test design. The CA is then run for $t = (n-1)$ time steps and the state of its least significant cell (LSB of attractor) defines presence of fault(s) either in sharing status or in sharing vector.

The scheme reported above is described for MSI protocol. However, the scheme also applies for MOSI/MESI/MOESI protocols. For example, if the states "M," "O," "E," "S," and "I" of MOESI are represented by 111, 101, 110, 100, and 000 (001, 010, and 011, do not care), respectively, then the same logic as applied in case of MSI can be effective for verification for MOESI. The MSBs of the state code for "M," "O," "E," and "S" are chosen as 1 (Theorem 10). Accordingly, the corresponding presence bits should be 1 and the compatibility of these two can be checked by applying XOR logic as in MSI (Section 4). Henceforth, for any state code of arbitrary length, if the state(s) representing a sharing (dirty or clean) is represented by a binary code having a "1" in MSB, the proposed logic applies.

6. Hardware Realization

The hardware realization of $\text{CAVU}_{\text{FULL-DIR}}$ is shown in Figure 11. If the state of cached copy of B at ith cache C_i is "Invalid," that is, $Q_{i0}Q_{i1} = 00$ and the ith bit in sharing vector (p_i) is "0," then the XOR of Q_{i0} and p_i is $0 \oplus 0 =$ "0". The uniform CA $\langle 254, 254, 254, \ldots, 254\rangle$, loaded with such all 0s seed, is then run for $(n-1)$ time steps to settle to the

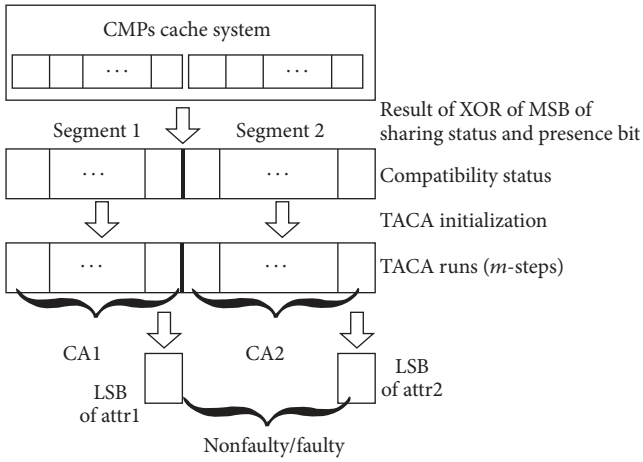


Figure 12: Verification unit with segmentation.

attractor $\langle 00\cdots0\rangle$. However, with B's state at ith cache C_i as "Invalid," that is, $Q_{i0}Q_{i1} = 00$, if the ith bit in sharing vector (p_i) is "1," then the XOR of Q_{i0} and p_i is $0 \oplus 1$ = "1". If uniform CA $\langle 254, 254, 254, \ldots, 254\rangle$ is loaded with nonzero seed ($\langle 00\cdots1\cdots0\rangle$) and run for $(n-1)$ time steps, it settles to the attractor $\langle 11\cdots1\rangle$ (Figure 8). Hence, the LSB of attractor (state of least significant CA cell) "0" indicates the absence of fault and "1" indicates presence of single or multiple faults. The part of the next state logic, shown in Figure 11, is also shared in realizing the next state logic of cell $(i-1)$ and cell $(i+1)$.

The design of $\text{CAVU}_{\text{FULL-DIR}}$ reported in this section requires an n-cell CA for CMPs with n-processor cores and the CA needs to run for t (=depth of CA) time steps to decide on the inconsistency in the recording of cache line states. The computation steps $t = (n-1)$, that is, delay, however, can be reduced through introduction of segmentation of the CMPs processor pool.

7. Delay Minimization

In segmentation, an n-core CMPs processor pool is considered as the collection of 2^q ($q = 1, 2, \ldots$) segments each of $m = n/2^q$ cores. At each transition from a current cache line state to the next state (Table 2) and the corresponding update of the presence bit in the sharing vector and sharing status, the proposed verification unit forms 2^q number of m-cell CA. For example, let us consider the case when $q = 1$ (Figure 12). The n-bit compatibility status of the n-core CMPs is partitioned into $2^q = 2^1 = 2$ halves/segments which are fed to two CA, CA_1 and CA_2, respectively. The compatibility status from $C_1, C_2, \ldots, C_{n/2}$ is fed to CA_1 and that of $C_{n/2+1}, \ldots, C_n$ to CA_2 (assuming n is in powers of 2). The CA_1 and CA_2 are then run parallelly for $m - 1 = (n/2 - 1)$ time steps. The resulting attractors of both CA_1 and CA_2 dictate an inconsistency (Figure 12), if it exists. That is, by sensing the LSBs of two attractors (called check bits) of CA_1 and CA_2, the presence of fault can be detected.

The segmentation effectively reduces the number of computation steps of the verification logic by a factor of 2^q (the number of segments). However, this is achieved at the cost of number of check bits. The number of check bits equals the number of segments (2^q).

The verification unit introduced in the earlier sections decides on the inconsistencies after each transaction. However, instead of verifying each individual transaction, sometimes we need to maintain a log book for a set of transactions. The target is to keep trace of whether one or more transactions are faulty. The CA has the capability to store/memorize information [17] and this feature has been successfully exploited to synthesize a more efficient (high-speed) test design, reported in the next section.

8. High-Speed Verification

The proposed high-speed verification unit ($\text{CAVU}_{\text{HIGH-SPEED}}$) is capable of verifying a set of transactions. For a set of k transactions, if the jth transaction ($j \le k$) is faulty, the $\text{CAVU}_{\text{HIGH-SPEED}}$ keeps a trace of this transaction till completion of all the k transactions. That is, for an instance of a faulty transaction, $\text{CAVU}_{\text{HIGH-SPEED}}$ captures it and memorizes it.

For each transaction in n-processor core CMPs, with compatibility status CS, an n-cell CA is formed at the $\text{CAVU}_{\text{HIGH-SPEED}}$. CS_i (ith bit of CS) is used to set the ith CA cell rule. If the CA is then run for certain number (t) of steps, it settles to an attractor. During the execution of k transactions, if one or more transactions are found to be faulty, the effect of fault propagates to the least significant cell (LSB) of the CA (attractor). For the instance of faulty transaction(s), the CA settles to an attractor with LSB "1" and in cases wherein all the transactions are nonfaulty, the CA settles to an attractor with LSB "0". The precise steps followed to realize the $\text{CAVU}_{\text{HIGH-SPEED}}$ design are noted in the following algorithm.

Algorithm 2 (FUNCTION-$\text{CAVU}_{\text{HIGH-SPEED}}$).

Input. Presence bits and MSBs of sharing status for k transactions.

Output. Decision on presence of single or multiple faults after k transactions.

(1) Initialize the n-cell CA with all 0s.
For $i = 1$ to n

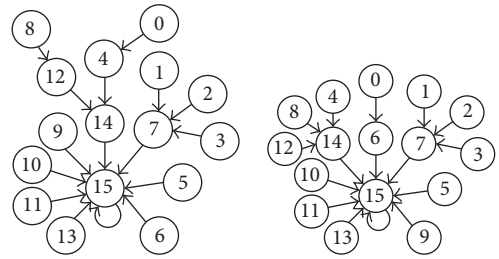
$$S_i^t = 0 \text{ and store it in } S^t[i] \tag{2}$$

(2) For $j = 1$ to k repeat (a) to (e):
(a) Perform jth transaction (read/write).
(b) For $i = 1$ to n

$$CS_i = Q_{i0} \oplus p_i \tag{3}$$

(c) Construct the CA (CS_i sets the ith cell rule-$CS_i = 0$ sets rule R_o and $CS_i = 1$ sets rule R_h).
(d) Run the CA 1-step to compute S^{t+1} from S^t.
(e) Store S^{t+1} to S^t.

(3) Run the CA for $(n-2)$ time steps considering S^t as the present state. It settles to an attractor state A. The LSB (1/0) of attractor indicates the presence/absence of fault(s).



(a) CA⟨254, 255, 254, 254⟩ (b) CA⟨254, 255, 255, 254⟩

Figure 13: State transition after hybridization.

Unlike $\text{CAVU}_{\text{FULL-DIR}}$, the current design demands synthesis of uniform as well as hybrid CA. That is,

(i) the CA formed should be of single length cycle attractor CA;

(ii) when all the transactions are nonfaulty, a uniform CA is formed with LSB of attractor "0" and the CA traces through the 0-basin of the attractor;

(iii) in presence of one or more faulty transactions, a hybrid SACA having LSB of attractor "1" should be formed (say, all 1s attractor) and the CA traces through the 1-basin;

(iv) the hybridization of uniform CA results in conversion of the CA into an SACA so that the occurrence of a fault is translated as a one-way switch from the 0-basin to the 1-basin. This helps in preserving the trace of a faulty transaction, if any.

For example, the CA of Figure 8 can be chosen for the nonfaulty transactions. It then traces through 0-basin (self-loop). In presence of a faulty transaction, the hybrid CA of type shown in Figure 13 is formed. The hybrid CA traces through the 1-basin, that is, attractor with LSB "1".

Theorem 11. *The uniform TACA with rule R can be converted into an SACA, when hybridized with rule 255 at single or multiple positions (cells), if both RMTs 0 and 7 and either of RMTs 1 and 3 and one of RMTs 4 and 6 are passive in R.*

Proof. Let us consider the uniform TACA with rule R is hybridized with rule 255 at $(i + 1)$th cell. Since the passive RMTs of rule 255 are 2, 3, 6, and 7, the passive RMTs for the ith cell can be RMTs 1 and 5 (for which the RMTs of $(i + 1)$th cell are 2 and 3) and RMTs 3 and 7 (for which the RMTs of $(i+1)$th cell are 6 and 7). Since R is a TACA rule, RMT 5 of R cannot be passive (Property 4). Further, the RMTs on which the $(i+2)$th cell (configured with R) can change its state are 4, 6, and 7 (RMT 5 is active). The 0 branch (created with passive RMT 0 of R) of the RTA is stopped due to hybridization, as rule 255 does not have RMT 0 as passive. Now, for the continuity of the nonzero branch, either of RMTs 1 and 3 and one of RMTs 4 and 6 must be passive. With these sequences of passive RMTs of rule R and rule 255, only one path from root to $(i + 1)$th node as well as only one path from $(i + 1)$th node to the leaf is possible in the RTA of the CA; that is, the RTA is having only one path from root to leaf node which corresponds to the only single length cycle attractor. Further, it can be verified from the NSRT diagram [21] of the hybrid CA that it does not introduce any additional multilength cycle in hybridization. Hence, the CA resulting from hybridization is an SACA. □

For example, the uniform CA constructed with TACA rules 174/244/254 when hybridized with rule 255 at single or multiple positions is transformed into SACA.

Corollary 12. *The uniform TACA with rule 254 (R_o) is converted into an SACA when hybridized with rule 255 (R_h) at single or multiple positions.*

Proof. Uniform TACA configured with rule 254 forms only two attractors, $X_1 = 00\cdots0$ and $X_2 = 11\cdots1$ (Theorem 9); that is, the RTA has only two branches leading to two leaf nodes (X_1 and X_2). The all 0s branch is followed on the self-replication of RMT 0. The passive RMTs of rule 254 are RMTs 0, 2, 3, 6, and 7 and those of rule 255 are 2, 3, 6, and 7. Now, if the uniform CA $\langle 254, 254, \ldots, 254\rangle$ is hybridized with rule 255 at any position, it blocks the "0" branch (all 0s attractor), thus leaving only the all 1s attractor. Further, it can be verified from the NSRT diagram [21] of the hybrid CA that it does not introduce any additional multilength cycle in hybridization. Hence, the resulting CA is an SACA. □

For the current realization, we consider hybridization (with $R_h = 255$) of a uniform CA formed with rule $R_o = 254$. Such hybridization allows merger of attractor basins of the uniform CA (0-basin of uniform CA is merged with the 1-basin of hybrid CA). That is, for "NF" case of Section 4, the system traces through the X_1-basin (0-basin, Figure 8) of the uniform CA and for a fault the hybrid CA is formed and the system traces through the merged X_2-basin (15-basin, Figure 13) of hybrid CA.

Figure 14 describes the operation of the $\text{CAVU}_{\text{HIGH-SPEED}}$. Let us consider a system of 4 caches and a set of 5 transactions on the caches among which the 1st transaction results in a single fault and the 2nd transaction results in a double fault (indicated by the 1s in the compatibility status). The 0th transaction is a nonfaulty transaction (compatibility status 0000, as shown in Figure 14). So, a 4-cell uniform CA $\langle 254, 254, 254, 254\rangle$ is formed. The CA is then run for 1 time step with all 0s initial seed. It produces the next state 0000. Since transaction 1 is a faulty transaction, it results in a hybrid

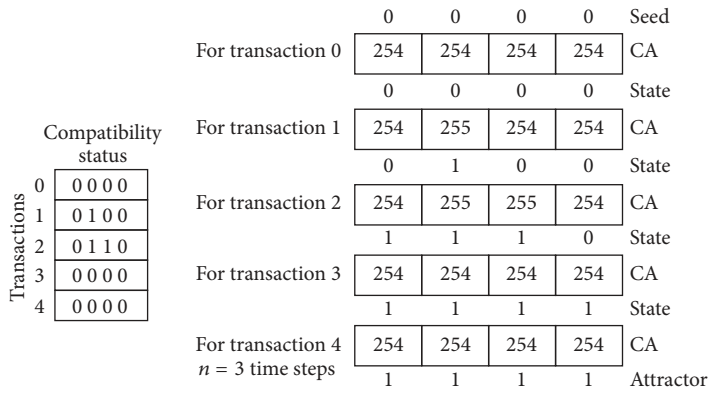


Figure 14: Functioning of the $CAVU_{HIGH\text{-}SPEED}$.

CA configured with rules 254 and 255 ($\langle 254, 255, 254, 254\rangle$). The hybrid CA, when running for 1-step from 0000 state, produces the next state 0100. Transaction 2, being a faulty transaction with double faults at cell 1 and cell 2, also results in a hybrid CA $\langle 254, 255, 255, 254\rangle$. This CA, when running 1-step from 0100 state, produces the next state 1110. Transactions 3 and 4 are nonfaulty transactions. Hence, the CA constructed for transactions 3 and 4 are the uniform CA with rule 254 ($\langle 254, 254, 254, 254\rangle$). The CA for transaction 3, when running for 1-step from 1110 state, results in 1111 state. The CA for transaction 4 is then run for $(n - 1) = 3$-steps and it reaches the attractor 1111 state. The "1" in the LSB of the attractor indicates the presence of faulty transactions.

9. Experimental Results

The performance of proposed verification unit is evaluated in Multi2Sim [22]. A module realizing the verification unit is developed and is augmented in Multi2Sim. The standard programs in SPLASH-2 benchmark suit [23] are used as the workload. The L_1 cache in each core is unified for instructions and data and L_2 is shared. The following test environment and parameters as described in Table 5 are considered for the experimentations. The benchmarks programs are run with input data sets as listed in Table 6:

(i) Operating system: Ubuntu 12.04LTS (64 bits).

(ii) Number of cores used: 4, 8, and 16.

We evaluate the percentage of memory references (load/store) that results in a state change in caches (Figure 15) to determine how frequently the verification unit needs to verify transactions. If the number of memory references resulting in state change increases, it points to system vulnerability even for a single fault in the system. Figure 15 denotes that the memory references increase with the number of cores. This is due to increase in memory traffic resulting from coherence misses and other aspects.

Some faults in the system do not lead to error and hence remain hard to detect. Figure 16 depicts the percentage of faults turning into errors. For our evaluation, we have randomly injected fault at various parts of CC at an interval of 1000 and 10,000 clock cycles. The error coverage (percentage of errors detected) for fault injection interval of 10,000 cycles and 1000 cycles is reported in Figures 17 and 18, respectively. These show that the proposed CA based verification unit ($CAVU_{FULL\text{-}DIR}$) ensures error coverage which is almost the same as that of the scheme reported in [2].

Table 7 reports the comparison of the CA based schemes with and without segmentation (Section 7). Column 1 notes the number of processor cores. Columns 2 and 3 report the number of computation steps and the number of check bits required to decide on the incoherency (without segmentation). The requirements in segmentation based scheme are shown in columns 4–6.

In columns 5–10 of Table 8, we report the area overhead (gate counts, FFs, and the 2-input NAND/XOR gates) of the CA based designs for the CMPs with 16 to 256 processor cores (column 1). The area computation follows units specified in mcnc.genlib [24]. The requirements for the design reported in [2] are given in columns 2–4 for comparison. Columns 5–8 show the overhead (gate count and area) of the verification logic for MSI (Section 4), without the consideration of segmentation of processor pool. The figures of columns 9 and 10 represent the area requirement for MESI and MOESI protocols, respectively. Comparison of the figures noted in columns 4 and 8 reveals the fact that the CA based verification logic achieves a considerable reduction in area.

The overhead of the $CAVU_{HIGH\text{-}SPEED}$ is shown in Table 9. The gate counts are provided in columns 2, 3, and 4. The area computed as per [24] is reported in column 5. Column 6 is reproduced (for $CAVU_{FULL\text{-}DIR}$) for comparison. It can be observed that the hardware overhead of $CAVU_{HIGH\text{-}SPEED}$ is the same as that of the $CAVU_{FULL\text{-}DIR}$. However, the $CAVU_{HIGH\text{-}SPEED}$ can be better accepted when there is a need for verifying a set of transactions rather than individual transactions. For a system of n-processor cores, the design of $CAVU_{FULL\text{-}DIR}$ requires $(n - 1)$ computation steps to make a decision on a defect after each transaction whereas, for k number of transactions, the $CAVU_{HIGH\text{-}SPEED}$ ensures a correct decision on fault in exactly $(k + n - 2)$ computation steps ($(k-1)$-steps for the first $(k-1)$ transactions and $(n-1)$-steps for the last transaction) of the CA hardware. The design thus achieves a speedup of

$$S_k = \frac{k(n-1)}{k+n-2} \tag{4}$$

over the design of $CAVU_{FULL\text{-}DIR}$ for a set of k transactions.

Table 5: Parameters used in full system simulation.

Parameter	Value
Processors	x86 cores, 1 GHz, single issue, in-order
L_1 cache	64 KB per core, unified, LRU replacement policy 4-way associative, 2-cycle latency, 64-byte line
L_2 cache	2 MB, LRU replacement policy, 4-way associative, 20-cycle latency, 64-byte line
Memory	1 GB, 100-cycle latency
Network	2D mesh topology, 2-cycle router, 1-cycle link latency, 36 bytes wide

Table 6: Benchmark and input data set.

Benchmark	Input parameter
Barnes	16 K particles
FFT	256 K integers
FMM	16 K particles
Ocean	258 × 258
Radix	1024 K keys
LU	512 × 512
Raytrace	car.env

10. Verification for Limited Directory

The design described in Sections 4 and 5 is tuned to full-map directory in which traditionally a sharing vector is maintained for a block B to indicate cached copies of B in the system. This sharing vector is of length $(n + 1)$ for a system of n-processor cores. As a result, the directory storage overhead is quite unacceptable for CMPs with thousands of processor cores. The alternative scheme that considers compact directory organization is the limited directory protocol [25, 26]. In such organization, the $(n + 1)$-bit sharing vector of block B is replaced by the fixed number of pointers to the processors' caches which are having a copy of B. In this section, we address the design of a verification logic for a system with limited directory, considering non-broadcast-based solution to handle the case of directory runs short of pointers [25].

In a system with limited directory protocol, the pointers indicating processor ids/caches need to be decoded. The structure of limited directory, shown in Figure 19, indicates that caches C_2 and C_3 (corresponding to processors P_2 and P_3) are currently sharing a block B and at most four processors can share block B simultaneously (as 4 fields are for the pointers). That is, the update of sharing status and the pointers involves a small and fixed number of entries. Hence, the CA based verification unit for limited directory ($CAVU_{LIM\text{-}DIR}$) can be realized with an r-cell CA ($r \ll n$), where r is the maximum number of sharers allowed for a block and thereby reduces the time to decide the coherence status.

Figure 20 describes an architecture of $CAVU_{LIM\text{-}DIR}$ with four pointers. For each cache transaction, after the pointer updating, the pointer is decoded to access the corresponding cache and the status of the block is read from the cache. Depending on the number of pointers r (in Figure 20, $r = 4$), an r-cell CA is formed. The MSB of status of a block (B) in the cache, pointed to by the ith pointer, is used to set the ith CA cell rule. If the MSB of status is "1", the corresponding CA cell rule is set to 254; otherwise, it is set to rule 255. This differs from the rule selection for $CAVU_{FULL\text{-}DIR}$. Once the CA cell rule is set, the r-cell CA is run for $(r - 1)$ time steps and the LSB of the attractor ("1"/"0") indicates the presence/absence of a fault in limited directory entry.

The overhead of $CAVU_{LIM\text{-}DIR}$, considering a four-pointer representation of the limited directory (as shown in Figure 19), is illustrated in Table 10. Column 1 represents the number of cores and columns 2 and 3 note the gate counts (number of FFs and 2-input NANDs). Column 4 records the area overhead. The area overhead of the $CAVU_{FULL\text{-}DIR}$ has been reproduced in column 5 for comparison. The results show considerable reduction in gate count and the area overhead in $CAVU_{LIM\text{-}DIR}$ compared to those of $CAVU_{FULL\text{-}DIR}$.

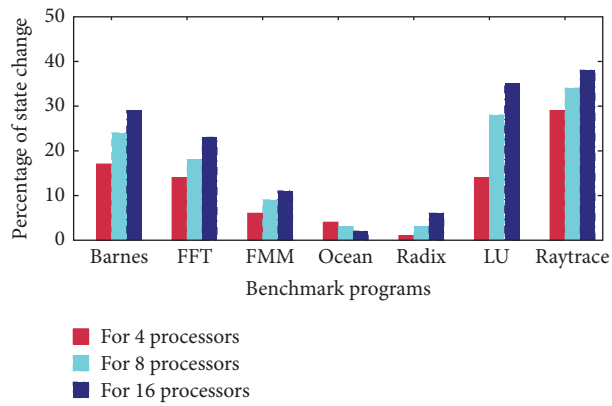


Figure 15: Percentage of memory references causing state change in caches.

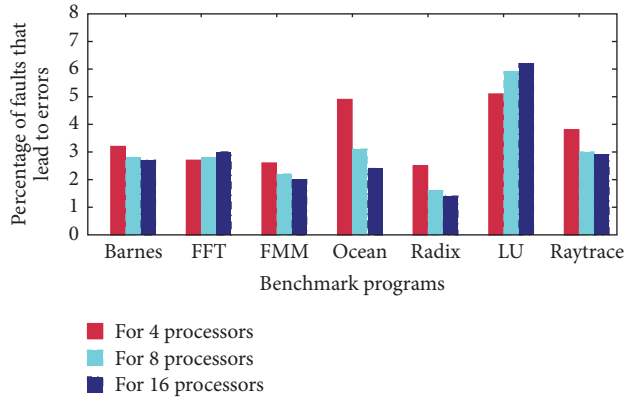


Figure 16: Percentage of faults leading to errors.

11. Related Work

The schemes ensuring coherency in CMPs with large number of cores are reported in [2–5, 11, 27]. These deal with the interactions between on-chip interconnection network and the cache coherence protocol. Liqun et al., in [3], propose

Table 7: Performance in CA segmentation.

	$CAVU_{FULL-DIR}$ without segmentation		$CAVU_{FULL-DIR}$ with segmentation		
Number of cores	Number of comp. steps	Number of check bits	Number of segments	Number of comp. steps	Number of check bits
(1)	(2)	(3)	(4)	(5)	(6)
16	16	1	2	8	2
			4	4	4
256	256	1	2	128	2
			4	64	4
1024	1024	1	2	512	2
			4	256	4
			8	128	8

Table 8: Hardware requirements for $CAVU_{FULL-DIR}$.

	Scheme [2]			$CAVU_{FULL-DIR}$					
				MSI				MESI	MOESI
Number of cores	Number of FFs	Number of NANDs	Area (units)	Number of FFs	Number of NANDs	Number of XORs	Area (units)	Area (units)	Area (units)
(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)
16	12824	159	47824016	16	144	16	259840	259840	259840
32	21152	159	78737552	32	288	32	519680	519680	519680
64	41512	159	154313872	64	576	64	1039360	519680	519680
128	80432	159	298784912	128	1152	128	2078720	519680	519680
256	159288	159	591498384	256	2304	256	4157440	519680	519680

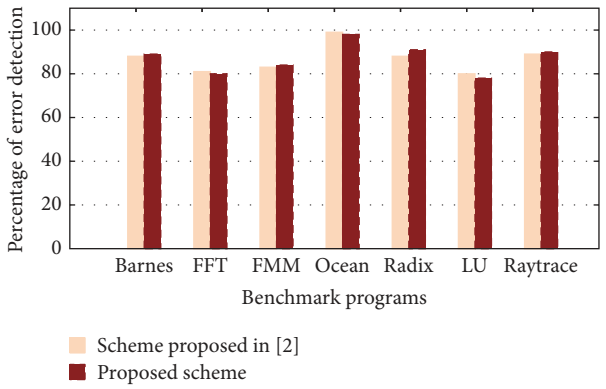


Figure 17: Error coverage with fault injection interval of 10000 clock cycles.

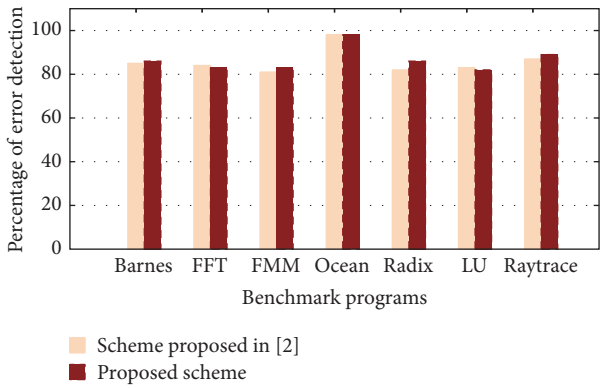


Figure 18: Error coverage with fault injection interval of 1000 clock cycles.

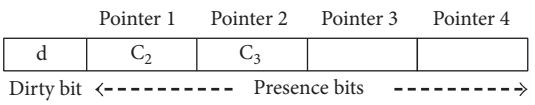


Figure 19: Sharing vector for limited directory.

a solution with interconnection network composed of wires with varying latency, bandwidth, and energy characteristics, and coherence operations are intelligently mapped to the appropriate wires of heterogeneous interconnection. Zhao et al. [28] propose an alternative novel L_2 cache architecture, where each processor has a split private and shared L_2 cache. When data is loaded, it is located in private L_2 or shared L_2 according to its state (exclusive or shared). This scheme efficiently utilizes the on-chip L_2 capacity ensuring low average access latency. This scheme employs a snooping cache coherence protocol and verifies it with formal verification method. A network caching architecture was proposed to

Table 9: Hardware requirements for high-speed design ($CAVU_{HIGH\text{-}SPEED}$).

	$CAVU_{HIGH\text{-}SPEED}$				$CAVU_{FULL\text{-}DIR}$
Number of cores	Number of FFs	Number of NANDs	Number of XORs	Area (units)	Area (units)
(1)	(2)	(3)	(4)	(5)	(6)
16	16	144	16	259840	259840
32	32	288	32	519680	519680
64	64	540	64	1039360	1039360
128	128	1152	128	2078720	2078720
256	256	2304	256	4157440	4157440

Table 10: Hardware requirements for $CAVU_{LIM\text{-}DIR}$.

	$CAVU_{LIM\text{-}DIR}$ (considering four pointers)			$CAVU_{FULL\text{-}DIR}$ (Section 4)
# cores	# FFs	# NANDs	Area (units)	Area (units)
(1)	(2)	(3)	(4)	(5)
16	4	114	345216	259840
32				519680
64				1039360
128				2078720
256				4157440

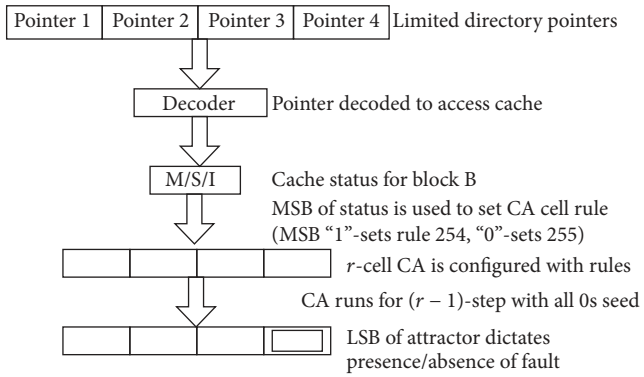


Figure 20: Verification unit for limited directory.

address the issues of on-chip memory cost of directory and long L_1 cache miss latencies in [5]. The directory information is stored in network interface component thus eliminating the directory structure from L_2 cache. A verification logic that can dynamically detect errors in coherence controller (CC) has been proposed in [2]. Ros et al. have proposed the coherence protocol Dico-CMP for tiled CMP architecture [11]. Dico-CMP avoids indirection by providing a block from the owner node instead of home node thus reducing the network traffic compared to broadcast-based protocols. Further, a scalable organization of directory based on duplicating tags has been proposed to ensure that the directory bank size is independent of the number of tiles. However, the verification of cache coherence, an important problem, has not been addressed properly, possibly due to lack of efficient verification tool [29]. A verification logic which caters to snoop based systems only is reported in [2].

12. Conclusion

A solution for quick determination of data inconsistencies in caches as well as in recording of the sharing vectors in a system realizing directory based cache coherence protocol is reported. It avoids rigorous computation and communication overhead assuring robust and scalable design, specially, for a system with thousands of processor cores. The design is proved to be highly flexible to cater to different cache coherence protocols, for example, MSI/MESI/MOESI. The introduction of segmentation of the CMPs' processor pool ensures better efficiency in making decision on the inconsistencies in maintaining cache coherence. Further, the design has been modified to cope up with limited directory based protocol.

Competing Interests

The authors declare that they have no competing interests.

References

[1] K. Olukotun and L. Hammond, "The future of microprocessor," *Magazine Queue—Multiprocessors*, vol. 3, no. 7, pp. 26–29, 2005.

[2] H. Wang, S. Baldawa, and R. Sangireddy, "Dynamic error detection for dependable cache coherency in multicore architectures," in *Proceedings of the 21st International Conference on VLSI Design (VLSI DESIGN '08)*, pp. 279–284, Hyderabad, India, January 2008.

[3] C. Liqun, N. Muralimanohar, K. Ramani, R. Balasubramonian, and J. B. Carter, "Interconnect-aware coherence protocols for chip multiprocessors," in *Proceedings of the 33rd International Symposium on Computer Architecture (ISCA '06)*, pp. 339–350, Boston, Mass, USA, June 2006.

[4] R. E. Ahmed, "Energy-aware cache coherence protocol for chip-multiprocessors," in *Proceedings of the Canadian Conference on Electrical and Computer Engineering (CCECE '06)*, pp. 82–85, IEEE, Ottawa, Canada, May 2006.

[5] J. Wang, Y. Xue, H. Wang, and D. Wang, "Network caching for chip multiprocessors," in *Proceedings of the IEEE 28th International Performance Computing and Communications Conference (IPCCC '09)*, pp. 341–348, Scottsdale, Ariz, USA, December 2009.

[6] R. Gong, K. Dai, and Z. Wang, "Transient fault recovery on chip multiprocessor based on dual core redundancy and context saving," in *Proceedings of the 9th International Conference for Young Computer Scientists (ICYCS 08)*, pp. 148–153, IEEE, Hunan, China, November 2008.

[7] A. Yamawaki and M. Iwane, "Coherence maintenances to realize an efficient parallel processing for a cache memory with synchronization on a chip-multiprocessor," in *Proceedings of the 8th International Symposium on Parallel Architectures, Algorithms and Networks (ISPAN '05)*, December 2005.

[8] C. Fensch, N. Barrow-Williams, R. D. Mullins, and S. Moore, "Designing a physical locality aware coherence protocol for chip-multiprocessors," *Institute of Electrical and Electronics Engineers. Transactions on Computers*, vol. 62, no. 5, pp. 914–928, 2013.

[9] J. L. Hennessy and D. A. Patterson, *Computer Architecture: A Quantitative Approach*, Morgan Kaufmann, 3rd edition, 2003.

[10] A. Ros, M. E. Acacio, and J. M. García, "Scalable directory organization for tiled CMP architectures," in *Proceedings of the International Conference on Computer Design (CDES '08)*, pp. 112–118, July 2008.

[11] A. Ros, M. E. Acacio, and J. M. García, "A direct coherence protocol for many-core chip multiprocessors," *IEEE Transactions on Parallel and Distributed Systems*, vol. 21, no. 12, pp. 1779–1792, 2010.

[12] S. Wolfram, *Cellular Automata and Complexity—Collected Papers*, Addison Wesley, 1994.

[13] M. Dalui, K. Gupta, and B. K. Sikdar, "Directory based cache coherence verifivation logic in CMPs cache system," in *Proceedings of the 1st International Workshop on Many-Core Embedded Systems (MES '13) in Conjunction with the 40th Annual IEEE/ACM International Symposium on Computer Architecture (ISCA '13)*, pp. 33–40, Tel-Aviv, Israel, June 2013.

[14] M. Dalui and B. K. Sikdar, "Design of directory based cache coherence protocol verification logic in CMPs around TACA," in *Proceedings of the 11th International Conference on High Performance Computing and Simulation (HPCS '13)*, pp. 318–325, IEEE, Helsinki, Finland, July 2013.

[15] G. Kurian, J. E. Miller, J. Psota et al., "ATAC: a 1000-core cache-coherent processor with on-chip optical network," in *Proceedings of the 19th International Conference on Parallel Architectures and Compilation Techniques (PACT '10)*, pp. 477–488, Vienna, Austria, September 2010.

[16] D. Lenoski, J. Laudon, K. Gharachorloo, A. Gupta, and J. Hennessy, "The directory-based cache coherence protocol for the DASH multiprocessor," *ACM SIGARCH Computer Architecture News*, vol. 18, no. 2, pp. 148–159, 1990.

[17] P. Pal Chaudhuri, D. Roy Chowdhury, S. Nandi, and S. Chatterjee, *Additive Cellular Automata—Theory and Applications*, vol. 1, IEEE Computer Society Press, Los Alamitos, Calif, USA, 1997.

[18] M. Dalui and B. K. Sikdar, "An efficient test design for verification of cache coherence in CMPs," in *Proceedings of the 9th IEEE International Conference on Dependable, Autonomic and Secure Computing (DASC '11)*, pp. 328–334, Sydney, Australia, December 2011.

[19] S. Das, N. N. Naskar, S. Mukherjee, M. Dalui, and B. K. Sikdar, "Characterization of CA rules for SACA targeting detection of faulty nodes in WSN," in *Proceedings of the 9th International Conference on Cellular Automata for Research and Industry (ACRI '10)*, Ascoli Piceno, Italy, September 2010.

[20] S. Das, S. Mukherjee, N. N. Naskar, and B. K. Sikdar, "Characterization of single cycle CA and its application in pattern classification," *Journal of Electronic Notes in Theoretical Computer Science*, vol. 252, pp. 181–203, 2009.

[21] M. Dalui, *Theory and applications of cellular automata for CMPs cache system protocol design and verification [Ph.D. thesis]*, IIEST Shibpur, Howrah, India, 2014.

[22] R. Ubal, B. Jang, P. Mistry, D. Schaa, and D. Kaeli, "Multi2Sim: a simulation framework for CPU-GPU computing," in *Proceedings of the 21st International Conference on Parallel Architectures and Compilation Techniques (PACT '12)*, pp. 335–344, ACM, September 2012.

[23] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta, "SPLASH-2 programs: characterization and methodological considerations," in *Proceedings of the 22nd Annual International Symposium on Computer Architecture*, pp. 24–36, June 1995.

[24] E. M. Santovich, K. J. Singh, L. Lavagno et al., "SIS: a system for sequential circuit synthesis," Tech. Rep. UCB/ERL M92/41, Electronic Research Laboratory, 1992.

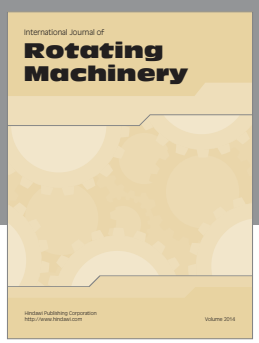
[25] A. Agarwal, R. Simoni, J. Horowitz, and M. Horowitz, "An evaluation of directory schemes for cache coherence," in *Proceedings of the 15th International Symposium on Computer Architecture (ISCA '88)*, Honolulu, Hawaii, USA, May-June 1988.

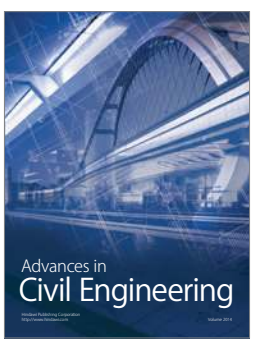
[26] M. Mahmoud and A. Wassal, "Hybrid limited-pointer linked-list cache directory and cache coherence protocol," in *Proceedings of the 2nd International Japan-Egypt Conference on Electronics, Communications and Computers (JEC-ECC '13)*, pp. 77–82, Cairo, Egypt, December 2013.

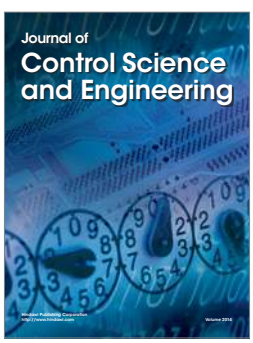
[27] E. Jerger, *Chip multiprocessor coherence and interconnect system design [Ph.D. thesis]*, University of Wisconsin-Madison, 2008.

[28] X. Zhao, K. Sammut, and F. He, "Formal verification of a novel snooping cache coherence protocol for CMP," in *Proceedings of the CMP-MSI: Workshop on Chip Multiprocessor Memory Systems and Interconnects*, 2007.

[29] F. Pong and M. Dubois, "Verification techniques for cache coherence protocols," *ACM Computing Surveys*, vol. 29, no. 1, pp. 82–126, 1997.







Hindawi
Submit your manuscripts at
http://www.hindawi.com