Received 16 April 2023, accepted 30 April 2023, date of publication 5 May 2023, date of current version 22 May 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3273219

RESEARCH ARTICLE

MSI-A: An Energy Efficient Approximated Cache Coherence Protocol

ANANT SARASWAT[1], KUMAR ABHISHEK[1], HITESHWAR KUMAR AZAD[2], AND S. SHITHARTH[3]

[1]Department of Computer Science and Engineering, National Institute of Technology Patna, Patna, Bihar 800005, India
[2]School of Computer Science and Engineering, Vellore Institute of Technology, Vellore, Tamil Nadu 632014, India
[3]Department of Computer Science, Kebri Dehar University, Kebri Dehar 250, Ethiopia

Corresponding author: S. Shitharth (shitharths@kdu.edu.et)

ABSTRACT Energy consumption has become an essential factor in designing modern computer system architecture. Because of physical limits, the termination of Moore's law and Dennard's scaling has forced the computer design community to investigate new approaches to meet the requirements for computing resources. Approximate computing has emerged as a promising method for reducing energy consumption while trading a controllable quality loss. This paper asserts that an approximated cache coherence protocol preserves overall energy for computation. We can approximate the cache coherence protocol by adding approximated cache lines to a certain level without hindering the output. This paper introduces an enhanced approximated version of the MSI (Modified Shared Invalid) cache coherence protocol MSI-A (Modified Shared Invalid-Approx). We have verified MSI-A and MSI by employing LTL specifications in the NuSMV model checker. To illustrate the benefits of MSI-A, we have added DTMC (Discrete-Time Markov Chain) with PCTL (Probabilistic Computational Tree Logic). Although the PCTL proves the theory of approximation, we have also simulated the MSI-A in the TEJAS hardware simulator on PARSEC 3.0 to investigate the energy gains and cycle gains of MSI-A in varied applications. The cache lines considered to be approx are between 10 and 30 percent. Each application benefited from approximation according to its nature, and VIPS has indicated a total energy gain of 30.18 percent.

INDEX TERMS Approximation, cache coherence, NuSMV, formal verification, temporal logic, computation.

I. INTRODUCTION

Designing modern computing systems inducts a high priority on reducing energy consumption. Now, there is a need to design a new paradigm of computation. The motivation behind these are two famous concepts, the first of which is the breakdown of Dennard scaling [1]. The second one is the end of Moore's law [2]. So, Dennard's scaling refers to the observation made by Robert Dennard, an IBM researcher, in 1974 that as the dimensions of transistors are scaled down, their power density remains constant, allowing for a proportional increase in the number of transistors that can be placed on a chip while keeping power consumption constant, and this allowed for a significant increase in the performance of computer processors over several decades, known as Moore's Law. However, as transistor dimensions continue to shrink, they are approaching the physical limits of materials and device physics. Dennard's scaling is no longer applicable. Reducing transistor size no longer leads to a proportional reduction in power consumption and heat dissipation. Instead, smaller transistors suffer from leakage current and other physical effects that increase power consumption and generate more heat, limiting the performance and scalability of future processors. The end of Dennard's scaling has significant implications for the future of computer architecture and design. The focus is shifting towards more efficient use of resources, such as energy and memory, and exploring new technologies, such as quantum computing, neuromorphic computing, and photonics. Additionally, there is an increased emphasis on developing algorithms and software to optimize performance on existing hardware platforms rather than relying on hardware improvements alone to drive

The associate editor coordinating the review of this manuscript and approving it for publication was Khursheed Aurangzeb.

performance gains.The Moore Law is reasonably related to Dennard's scaling.

Moore’s law is a term coined by Gordon Moore, co-founder of Intel, in 1965. The law states that the number of transistors on a microchip doubles every two years, significantly increasing processing power and decreasing the cost per transistor. This trend has held for several decades, leading to a rapid increase in computing power and technological advancement. The origins of Moore's law can be traced back to the development of the first microchip by Intel in 1971. At that time, the microchip had 2,300 transistors. However, by the mid-1970s, the number of transistors on a microchip had already increased to over 5,000. The trend continued to accelerate throughout the following decades. The doubling of transistors on a microchip every two years has led to a tremendous increase in computing power, resulting in faster and more efficient computers, smartphones, and other electronic devices. It has also driven the development of new technologies, such as cloud computing, artificial intelligence, and the Internet of Things, which rely heavily on the processing power of microchips. One of the most significant effects of Moore'slaw has been decreased cost per transistor. As the number of transistors on a microchip has increased, the cost per transistor has decreased, making it more affordable to produce increasingly powerful devices, leading to a democratization of technology, and allowing people worldwide to access and benefit from the latest technological advances. However, the rapid increase in computing power and decreased cost per transistor have also presented challenges. As the number of transistors on a microchip has increased, so has the amount of heat generated by the microchip. It has made it more difficult to dissipate heat and increased the overheating risk, which can cause system failures and reduced performance.

As Moore's law and Dennard's scaling come to an end, computing paradigms are shifting in other directions, and approximate computing is one of those new paradigms. Cache-coherence protocol is a safety-critical system, and as the number of cores increases, so does the energy required to maintain coherence. We can use approximate computing with cache coherence, and the idea is to arrange cache lines into approximate and precise ones [3]. The overall output will not suffer if we ignore the approximated cache lines and give the processor any approximated value for computation, and this will save the computation cycles and save energy. A new MSI protocol that reduces power consumption and improves overall performance is presented in this research. This advanced protocol is MSI-A (MSI with approximation), and to prove its correctness, we have verified it using LTL (Linear Temporal Logic) [4] specifications.

Developing a modern processing architecture will continue to benefit from cache coherence because it enables accessible shared-memory programming models that make it easier to create efficient parallel applications. After growing these processors’ cores to hundreds, energy consumption [5] becomes a significant design restriction. Methods to increase the efficiency of coherent multi-cores concerning energy consumption are essential for designing a modern computer architecture with advanced cache coherence. We have designed an advanced approximate cache-coherence protocol MSI-A emanated from MSI. Furthermore, to prove the correctness of the noble MSI-A protocol, we have incorporated Formal Methods [6], a unique mechanism for verifying safety-critical systems.

In this paper, we have developed an advanced approximated version of the MSI cache coherence protocol and infused approximation for saving energy. Now, to prove the correctness of the new protocol (MSI-A), we have to verify all the corners. Verification is an essential part of the software development life-cycle, and verification is necessary for safety-critical systems. Many techniques are available for verification, but Formal Methods are best suited for safety-critical systems. We need a glimpse of safety-critical systems to understand the need for Formal Methods. Safety-critical systems can be defined as [7]:

‘Software modules and subsystems whose malfunction imperils the user or could lead to unintended consequences are known as safety-critical systems.’

Formal Methods are among the best practices standards of the International Electrotechnical Commission (IEC) [8] and the European Space Agency (ESA) [9] standards. Both standards list formal methods as ‘‘strongly recommended’’ verification strategies for the software development of safety-critical systems. NASA[1] not only applies Formal Methods as a validation strategy but also recommends that every aspiring computer scientist be accustomed to formal methods and has their research program for Formal Methods. Formal Methods are helpful even in verifying machine-learning approaches and self-driving cars [10]. Many authors [11], [12] have considered the verification of the cache-coherence protocol by Formal Methods necessary because the computation of a parallel processing system is related to the cache-coherence protocol, and we can label cache-coherence protocols as safety-critical systems. In addition, researchers have used many techniques and tools to verify cache coherence protocols or new computer architectures. For example, Pang et al. [13] provide an overview of formal verification techniques for cache coherence protocols, including model checking and theorem proving. Burenkov et al. [14] present a framework for modelling and verifying cache coherence protocols using the Promela language and the Spin model checker. Joshi et al. [15] describe a formal model of the Intel cache coherence protocol using the TLA+ specification language and the TLC model checker. Xupeng et al. [16] describe the design and verification of the Arm Confidential Compute Architecture (CCA), a security extension for Arm processors that enables hardware-based confidential computing. Chen et al. [17] propose a new approach to verify the equivalence of two quantum circuits using partial equivalence checking (PEC) techniques. Quantum circuits are a collection

[1]https://shemesh.larc.nasa.gov/fm/

of quantum gates that operate on qubits, and verifying the equivalence of two such circuits is essential in ensuring the correctness of quantum algorithms and quantum hardware designs. Our proposal also requires a hardware simulator simulation, and we have simulated MSI-A to demonstrate the energy and cycle gain. We have discussed the simulation in the Proposed Work section of this paper.

The following contributions are made in this paper:

- A state-space system for MSI has been described, which focuses on approximate computing applications.
- To accommodate the approximate data at the cache lines, we implemented a modified MSI protocol MSI-A (a lazy cache coherence protocol [18]).
- We have designed a working model for the simulation of MSI-A for a hardware simulator with a data spectrum ranging from 10% to 30% approximation.
- We used NuSMV [19] to establish LTL specifications for verifying the MSI-A.
- We evaluated the limiting distribution of MSI-A and calculated its probabilities by PCTL.
- We have evaluated the energy gain and cycle gain of MSI-A over MSI using TEJAS simulator [20].

The remainder of this paper is structured as follows. Section II discusses related work, and Section III describes the proposed approach, which includes protocol verification, formal verification, and the structure of MSI-A. This section also covers the theoretical aspects of MSI as well as the detailed analysis using Discrete Time Markov Chains. Section IV discusses the result and discussion, which includes the outcome of hardware simulation. The conclusion is covered in the last section V.

II. RELATED WORK

Approximate computing has a long history of being used to gain energy and cycles in the signal and information processing communities. Recent work in approximate computing has focused on the development of new paradigms that can ignore a small number of functions while still producing an output that conserves energy and computation cycles. Our previous work [21] focused on MESI-A, and we had good results with it. These findings motivate us to look for other cache coherence protocols that we can tweak for approximable cache lines. The majority of the researchers suggested storing exact and approximation data in two distinct formats: Quaternary and binary format. Quaternary format data can be used for storing approximated data [22], while accurate data is collected in binary format. Quaternary storage permits the keeping of more data but is less dependable. Researchers proposed partitioning approximation and exact data per cache line in the paper Directory-based cache coherence in large-scale multiprocessors [23]. The system requires an extra bit to distinguish between the two. Researchers enforced approximation in DRAM by decreasing the amount of energy used to refresh cells, known as Flikker [24]. Banks et al. [18] suggest lowering synchronisation and data movement for little value, as work on multi-core processors has drawn much attention through snooping [25]. Alur et al. [26] proposes to use hybrid automata for verifying complex systems, and they also suggest that verification using formal methods is essential for safety-critical systems. Chatterjee et al. [22] proposed using stale data for estimated loads and introducing a rudimentary Stale Victim Cache (SVC) to store an approximate cache line following d-L1 eviction. Verification and proof of cache coherence protocols are essential because these protocols are critical for the computation of the system, and therefore these protocols are safety-critical systems. As formal verification is necessary for a safety-critical system, many investigators have validated important cache coherence algorithms [27]. Modelling the approximated cache-coherence protocol conceptually and formally validating it ensures the protocol is error-free. Although experimenters have validated the MESI protocol [28], our work MESI-A has also shown energy gains and cycle gains with approximation. Researchers also illustrated energy gain in the protocol, known as Doppelganger [3], although protocol verification is still lacking. Employing Markov's chains, researchers [29] solved infinite-state problems. Lyu et al. [30] endorse the verification process in the cache-coherence protocol, proposing scalable on-the-fly test-generating methods utilising quotient state space to consider all feasible states. Developing a cache coherence protocol can save computation energy with new paradigms [3]. Verification is a prerequisite for safety-critical systems. Bingham and Lyu [30] have taken scalability into account. However, simulation is required for our method to give a result, which was absent in many articles. Bankes et al. [18] proposed a weak memory cache-coherence strategy, and Chatterjee et al. [22] have not discussed the influence on computation in their work. However, we have taken motivation from that work.

III. PRELIMINARIES

Before proceeding with the MSI-A algorithm, this section illustrated the baselines for this article, which are the MSI cache coherence algorithm, Formal Methods for verification and Kripke structure used in model checking.

1) THE MSI ALGORITHM

The Requests in MSI algorithm are

- PrRd: It is the request from the processor for reading a cache line.
- PrWr: It is the request from the processor for writing a cache line.
- BusRd: When there is a read miss in a processor's cache line.
- BusRdX: When there is a write miss in a processor's cache line.
- BusUpgr: When a write hit occurs in any of the processor's cache line, it responds a ‘BusUpgr’ command into the bus, which invalidates the status of the particular cache line.

- Flush: The entire cache is written back into the memory.

There are three states in the MSI algorithm, and from these states, the transaction is described as follows:

1) Invalid:
   - On command PrRd, BusRd instruction executed and state changes to Shared.
   - On command PrWr, BusRdX instruction is executed, and state changes to Modified.
   - On command BusRd, BusRdX or a BusUpgr, the state remains Invalid.
2) Shared:
   - On command PrRd, the state of cache-line remains in the Shared state.
   - On command PrWr, BusUpgr is executed, and state changes to Modified.
   - On command BusRd, the state of the cache line remains in the Shared state.
   - On command BusRd, BusRdX or a BusUpgr, the state remains Invalid.
3) Modified:
   - On command PrRd or PrWr, the cache-line state remains in the Modified state.
   - On command BusRd, the cache state is flushed on the bus and the state of cache-line changes to Shared.
   - On command BusRdX, the cache line state is flushed on the bus and the state of the cahe-line changes to the Invalid state.
   - A BusUpgr is not going to execute if the cache line is in a modified state

Figure 1 is the illustration of the MSI algorithm. We have verified this algorithm with LTL specifications in NuSMV. The figure resembles the model we have created using the NuSMV model checker. The blue connectors are the request initiated by the processor, and the yellow ones are the requests initiated by the bus on cache lines.

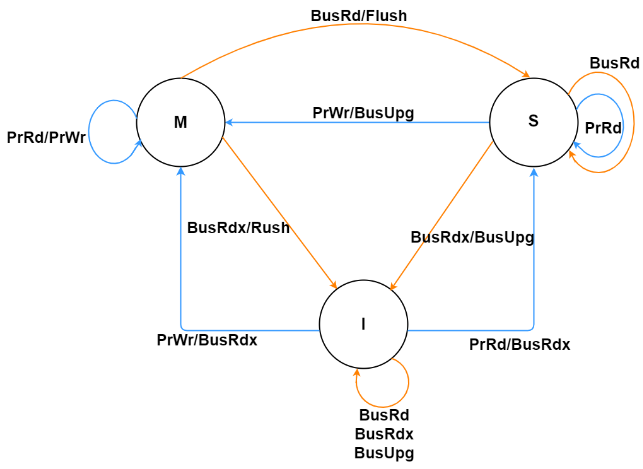


FIGURE 1. **MSI algorithm.**

2) FORMAL METHODS

Formal Methods are a combination of Finite State Machines and mathematical formulas. The Finite State Machine (FSM) is an early paradigm of computation. The great success of machine learning is also linked to FSM. Thus, the urged system achieves accuracy, consistency, and correctness through formal methods, which are necessary when dealing with safety-critical systems. Mathematical designs for any particular specification are both precise and free of ambiguity, and these specifications are coded in a tool using temporal logic.

3) KRIPKE STRUCTURE IN MODEL CHECKING

Kripke structure is the initial illustration of model checking where we give labels to an FSM. There are four components to a Kripke structure: the set of states (S), the transition relation between states (R), the initial state set (I), and a state labelling function ($L : S \implies 2^{AP}$). Researchers have developed many theories in automaton for verifying real-time systems using formal methods [31]. The Formal Method combines formal specification, formal verification and temporal logic. Formal specification is mathematical formulae that define the system, and the formal verification verifies the correctness of the system using temporal logic.

Temporal logic proves the correctness of any expression throughout the execution; for example, Consider the statement "I am in NIT Patna". Though its meaning is constant in time, the statement's truth value can vary. There are many approaches by which temporal logic can be coded in a computer. Examples of temporal logic are Linear Temporal Logic, Computational Tree Logic and many others. We can use Probabilistic Computational Tree Logic to create and analyse Discrete-Time Markov Chains.

This article used the NuSMV tool for model checking. In NuSMV, we can encode LTL, but for PCTL, we created DTMC and evaluated it manually. We validated the proposed MSI- A algorithm by employing LTL specifications in the NuSMV model checker. As cache coherence is being considered by researchers as a safety-critical system by researchers [32], [33], [34]. We ensure the correctness of safety-critical systems by encoding LTL formulae with PCTL.

During the experimental analysis, we assumed that 30% of the cache lines are approximated, and the processor knows which cache lines are approximated prior to execution. This article demonstrates that the hardware's strict action for coherency is inefficient for approximate cache lines. The MSI protocol and the new relaxed MSI-A protocol have been formally validated to ensure their safety and liveliness. We basically created a miniature cache coherence protocol with two processors and three cache lines for each processor.

IV. PROPOSED WORK

This article proposed an energy efficient approximated version of the MSI cache coherence protocol known as MSI-A (Modified Shared Invalid-Approx). The proposed MSI-A

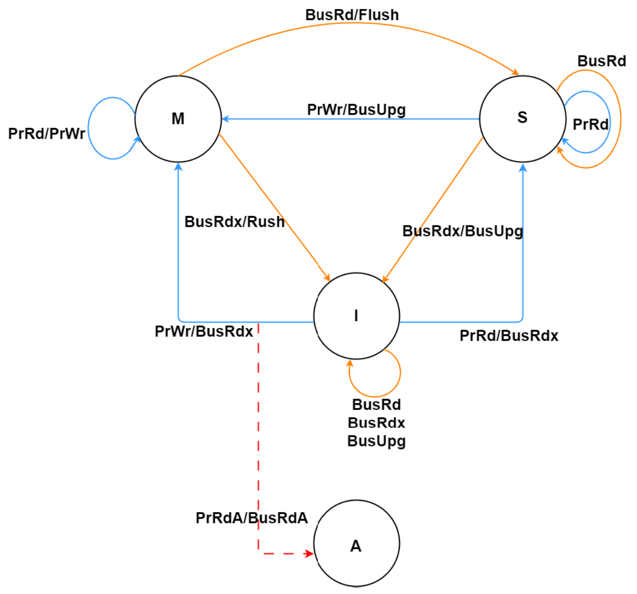


FIGURE 2. MSI-A algorithm.

protocol conserves overall computational energy. The cache coherence protocol can be approximated by adding approximated cache lines to a certain level without affecting the output. The proposed MSI-A cache coherence protocol is depicted in Figure 2. We tested this algorithm in NuSMV using LTL specifications. The figure is similar to the model we created with the NuSMV model checker. The blue connectors represent processor-initiated requests, while the yellow connectors represent bus-initiated requests on cache lines. The following assumptions are made for this protocol:

- Approximate data accumulates at the cache-line distributed only for the approx data. The processor separates the approximate data and creates a memory layout.
- Approximated data must be pre-defined; otherwise, the system will pay some bytes to separate approx data. The processor executes a special move instruction (mova), and the cache-line, which can be approximated, is placed in Approx state.
- All other states are similar to MSI.

We've assumed that a lightweight handheld device is linked to the system via a Local Area Network. For the execution of a specific programme, that device can send data to the processor, informing it how much execution in that programme can be approximated without significantly altering the output. Data that can be approximated will never be fetched from the processors, and the cache line state for that data will be 'APPROX'. In addition, the processor will assign cache lines that are approximable with approximate data, and the status of that cache line will not change during execution. This 'APPROX' state differs from the INVALID state because once a cache lines have been assigned with the approx state, the processor will not modify the state any further.

The handheld device will now be referred to as a 'hand device.' The hand device will assign a probability of approx data, i.e., how much data we can ignore, and will store a random value in the cache line for that data. To analyze the probability of being in the state of 'APPROX' and vice versa, we have created a probabilistic interpretation and designed Discrete-Time Markov Chains (DTMC). For DTMC representation, the future execution depends on the present scenario and not the past execution. The behavior of MSI-A also depends on the current execution because the requests are not dependable on any past executions. If we want to execute this into hardware, we need special instructions executed by the operating system. We have introduced a special appr() function, which will execute when program execution is moderately approximable, and this function takes the cache lines. A 'MOVA' hardware execution is needed to place cache lines in the appr() function.

The appr() function will take a parameter which is the address of approx data present in the program. For example, appr(1) represents a program that can give output without cache-line '1'. Figure 3 depicts the abstract concept as a Petri net. For the sake of illustration, we have only considered four cache lines. If a token is triggered by a handheld device, that cache line will have approx state throughout the program. We assumed that the programmer or user knows which programs respond well to approximate data and which are error-prone when dealing with approximate data. Theoretically, if less data is being fetched, a system will perform faster, and we will gain energy accordingly. Our assumption will consider cycle and energy gain. A cycle involves bringing data from the main memory into the cache, and fewer cycles are referred to as cycle gain. In contrast, energy gain is the total energy saved in executing an approximable program. We have mapped both aspects of the TEJAS simulator using the PARSEC benchmark suite.

A. PROTOCOL VERIFICATION

The MSI algorithm's compliance with the LTL measures has been successfully validated. The semantics of LTL are represented by the states and pathways, respectively. There is no specific reachability condition to consider in the MSI or MSI-A models, but all safety, liveliness, and fairness [35] criteria are taken into account. This article introduced the following lemmas as part of the validation process for the cache coherence protocol:

1) Every cache line will go to the modify state if a write operation is executed on it. This LTL ensures liveness of 'Write' state, i.e. write state, stay live throughout the algorithm. Here's a pseudo-code example of LTL: WrtCmd (Cachestatus = State($Invalid$ $\|$ $Invalid$ $\|$ Shared)) $\Longrightarrow$ State($Write$)
2) If cache lines of any processor have data, and from the bus, the BusRd request executed on other processors for the same data, the cache line state will be 'Shared'. This LTL is for the safety and fairness of 'Shared' state.
3) No two or more processor cache lines will store the same data. This LTL is for the safety and fairness of

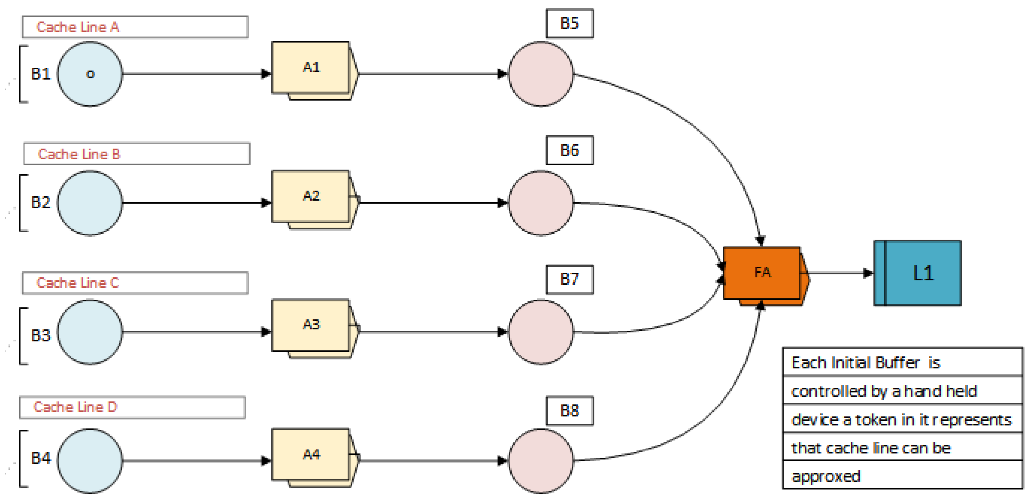


FIGURE 3. **MSI-A approximation illustration.**

cache lines. This LTL also verifies the memory subsystem of the cache.

4) If two or more processors have the same data in their cache lines and one processor has changed it, then the other processor's state of the cache line will go invalid.

Figure 4 depicts the NuSmv model's blueprint.

B. LTL WITH NuSMV

We have used LTL specifications within NuSMV for modelling and Verification. Linear Temporal Logic (LTL) and Computation Tree Logic (CTL) are popular formalisms for specifying and verifying systems' temporal properties. Both have strengths and weaknesses; the choice of which one to use depends on the application and the properties to verify.

There are some situations where LTL is preferable over CTL. Here are some reasons:

1) Simplicity: LTL is a more straightforward logic than CTL, with fewer operators and fewer types of temporal modalities. This simplicity makes it easier to use and understand, especially for beginners.
2) Expressiveness: LTL is more expressive than CTL when specifying temporal properties. It allows for specifying complex temporal relationships between events and can express properties that CTL cannot. For example, LTL can express properties like "eventually holds, infinitely often," while CTL cannot.
3) Automata-based model checking: Model checking is a popular technique for verifying the properties of systems, and it is often used with LTL and CTL. One advantage of LTL is that it can be checked using a simple type of automaton called a Buchi automaton, which is easier to construct and use than the automata used for CTL.
4) Natural language specifications: LTL can specify temporal properties in natural language-like syntax, making it easier for non-experts to understand and use. Conversely, CTL has a more complex syntax that can be harder to read and write.

However, it is worth noting that CTL also has its advantages over LTL. For example, it is better suited to verify the properties of systems with branching behaviour, which is common in concurrent and distributed systems. Additionally, some properties are more accessible to express in CTL than in LTL. Therefore, the choice between LTL and CTL ultimately depends on the application's specific needs and properties being verified.

Our model has not been incorporated by branching, so we have used LTL specifications. NuSMV and LTL also work efficiently together. While many tools are available for model checking, we have used NuSMV. NuSMV is a powerful model-checking tool widely used to verify hardware and software systems. Here are some reasons why NuSMV is considered better:

1) Expressive modelling language: NuSMV provides a rich modelling language that easily describes complex systems. It supports both temporal and modal logic, which are commonly used in the formal verification of systems.
2) Scalability: NuSMV can handle large models efficiently, making it suitable for verifying complex systems. It uses sophisticated algorithms to perform model checking, which allows it to scale to large systems.
3) Modular design: NuSMV has a modular design that simplifies adding new features or extending its functionality. It provides a plug-in architecture that allows us to add new verification algorithms or interfaces.
4) Open-source: NuSMV is open-source software, meaning anyone can download and use it for free. It is also actively maintained by a community of developers who provide support and fix bugs.
5) Widely used: NuSMV is widely used in industry and academia. It has been successfully applied to various systems, including hardware and software.

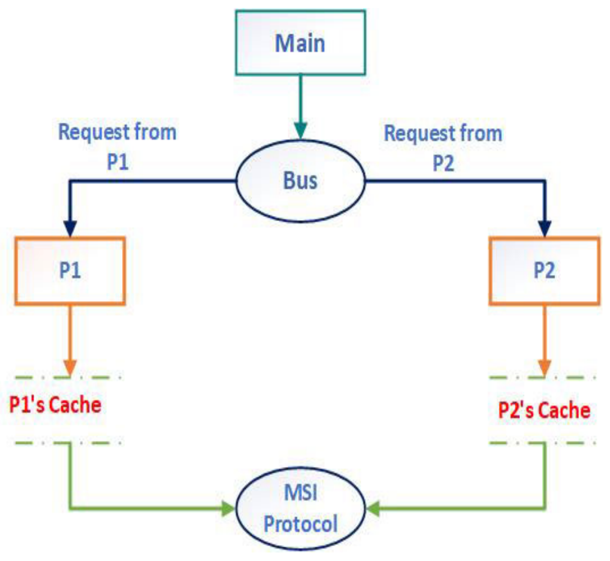


FIGURE 4. NuSMV blueprint of MSI and MSI-A.

Overall, NuSMV is a powerful and flexible tool that can help us verify the correctness of the system. Its expressive modelling language, scalability, modular design, open-source nature and wide adoption make it a better option for many verification tasks.

C. STRUCTURE OF MSI-A

In our simulation of the protocol, we did not take into account any modifications to the local caches. If certain cache lines can be approximated, they are not retrieved for processing, and instead, the processor is given a random number to work with. After calculating, the processor ignores changes into approximable cache lines; this is the reason why we are able to conserve energy and cycles using the MSI-A architecture. It is presumed here that the processor is aware of which cache lines can be approximately reconstructed. When such lines reach the front of the execution queue, the processors realign themselves and begin computing using random values based on a scale that has been previously determined.

The status of more than one cache state in the MSI may change because of the same data. Therefore, we have explained this model using a program graph, an advanced transition system version. We have shown the structure of this program graph in figure 5

Figure 5 illustrates the system for this approximation, and the blue boxes represent the approximable cache lines. The computation can use random values, and the overall outcome is never hampered through this approximation. The blue boxes will not command the processor to fetch the data from the main memory or write it back into the main memory, and the processor will place a random value. As per our assumption, these cache lines will not derail the overall output of the program.

With local writing, we save many cycles, such as a command from the processor that will fetch the data from memory. After the execution, the processor will place the new values in the main memory. Hence eventually, the execution will have cycle gains and energy gains.

D. FORMAL VERIFICATION OF MSI-A

The labelling transition system contains a notation of atomic proposition (AP) to label the system. Generally, AP is the subset of all the states in the model. At an abstract level, we have assumed two atomic propositions, ‘approx’ and ‘non-approx’ MESI-A. The AP ‘non-approx’ is for typical MSI protocol, and ‘approx’ is for MSI-A where only approximable cache lines are present. There must be no effects on MSI's essential properties, and the MSI-A model must sum up the stipulation of MSI protocol. We have added these two specifications for MSI-A:

1) We have verified that if the state of the cache is not approx, then it must change the state to modify state if there is any write operation performed on it.
2) We have verified that if the status of any cache line is ‘approx,’ then during the execution, it will stay on ‘approx,’ and no instruction will change it.

E. THEORETICAL ASPECTS OF MSI FOR PROVING ITS CORRECTNESS

The Atomic Propositions in Labels in Label Transition systems are defined with ($L : S \Longrightarrow 2^{AP}$). We have defined two atomic propositions, which are APPROX and NONAPPROX. The label function in MSI-A for NONAPPRX are L(Modify), L(SHARE) and L(INVALID); furthermore, for APPROX, we have L(APPROX) as the label function. The working of MSI-A and MSI is similar for all the label functions, excluding L(APPROX).

We have used another notation of predecessors and successors in the model to validate the termination of any computation for the APPROX cache-lines. The predecessors in a Transition System TS ($S, Act, \rightarrow I, AP, L$) where $e \in S$ and $\beta \in Act$ is defined as:

$$Post(e, \beta) = \left\{ e' \in S | e \xrightarrow{\beta} e' \right\}$$

$$Post(e) = \bigcup_{\beta \in Act} Post(e, \beta)$$

the successors is defined as:

$$Pre(e, \beta) = \left\{ e' \in S | e' \xrightarrow{\beta} e \right\}$$

$$Pre(e) = \bigcup_{\beta \in Act} Pre(e, \beta)$$

We defined Post(APPROX) as null using the aforementioned factors. As a result, the cache line in consideration cannot change its status further for that particular execution. The Pre(APPROX) state is invalid because we anticipated that cache lines that have been classified as approximable became invalid before the execution commenced. We have anticipated that the cache line may not be approximable for other executions. The bus will therefore invalidate all instances of

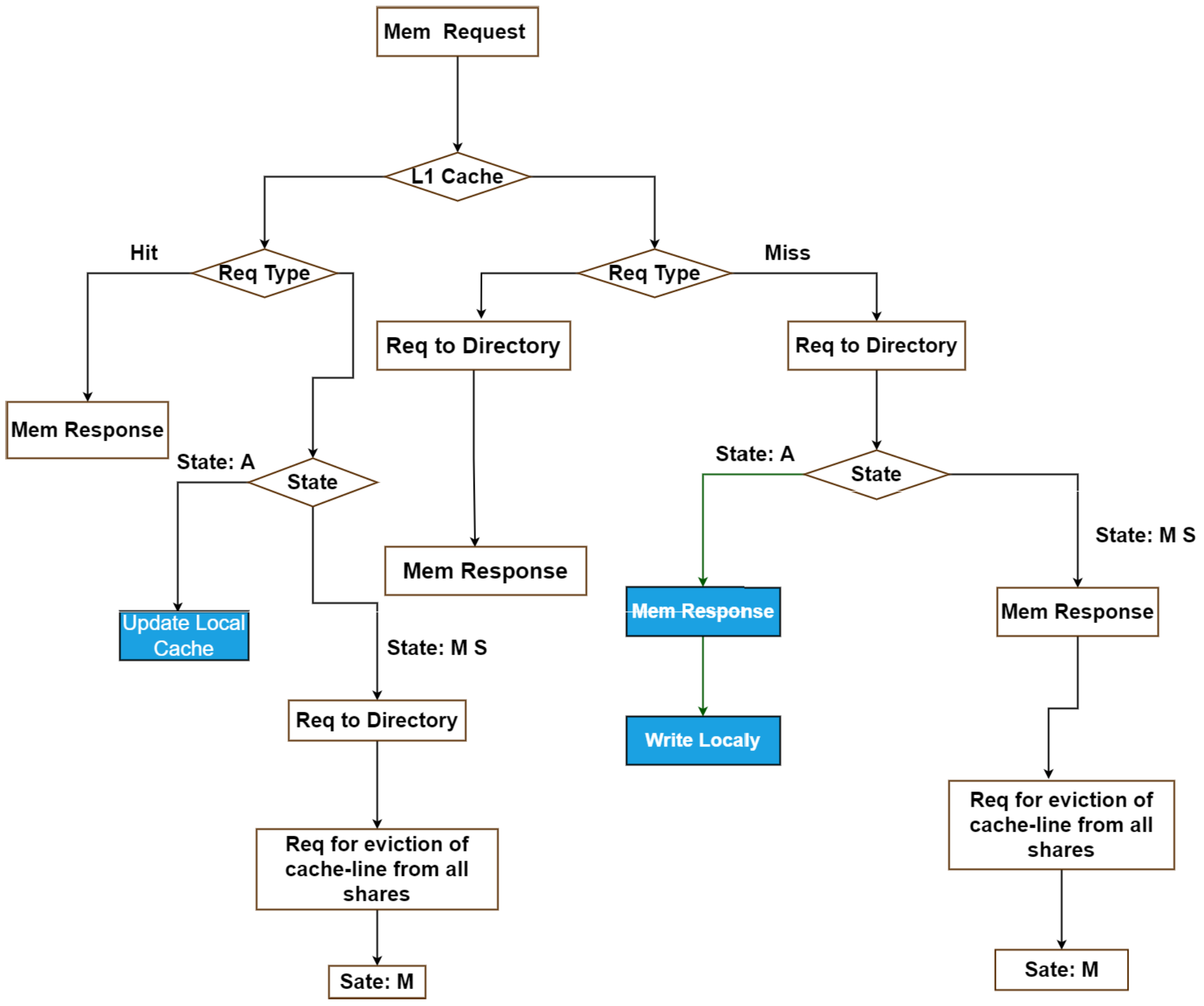


FIGURE 5. MSI-A structure for saving energy and cycle.

that cache line, and its status has been set to APPROX for executions where it is approximable. This assumption will not affect the other cache's existence.

The approximation is non-deterministic because it is possible for us to have an approximable cache line for any distinct execution while at the same time having the same cache line as typical (not approximable) for another execution.

We experienced challenges with state exploration in MSI-A as a result of the increased number of states generated by the number of cache lines. This problem is addressed using the sum operator in the MCRL2 environment, and it is assumed that eight cache lines (two for each CPU) are approximable. We investigated the sum operator using MCRL2 and used the sum operator to demonstrate MSI-A correctness. In this instance, the sum operator is applied to the full MSI-A model, and a correctness proof is necessary to explain the effect of the sum operator on the model, which handles ordinary cache lines.

Axiom-1:

$$\sum_{d:D} X = X$$

So if we apply the sum operation on the whole model, but because of Axiom 1, the expected performance of the protocol, which accepts standard cache lines, will not hinder.

Axiom-2:

$$\sum_{d:D} X(d) = X(e) + \sum_{d:D} X(d)$$

Axiom 2 explains that if there is a choice over many processes over X(d) for concrete d, we can take one of its choices generalized by the sum operator and put it separately. The model will behave as usual if we take out X(e). This Axiom proves that LTL applied to the model always functions with the sum operator or even on a finite part of it.

F. ANALYSIS USING DISCRETE TIME MARKOV CHAINS

The Discrete-Time Markov Chain (DTMC) is compatible with the Probabilistic Computation Tree Logic (PCTL) approach, which determines the probabilities of any discrete run. In order to facilitate a deeper level of comprehension, we have associated our assumption with a stochastic matrix. We considered the read requests in the caches based on our findings. We determined that each cache in the APPROX

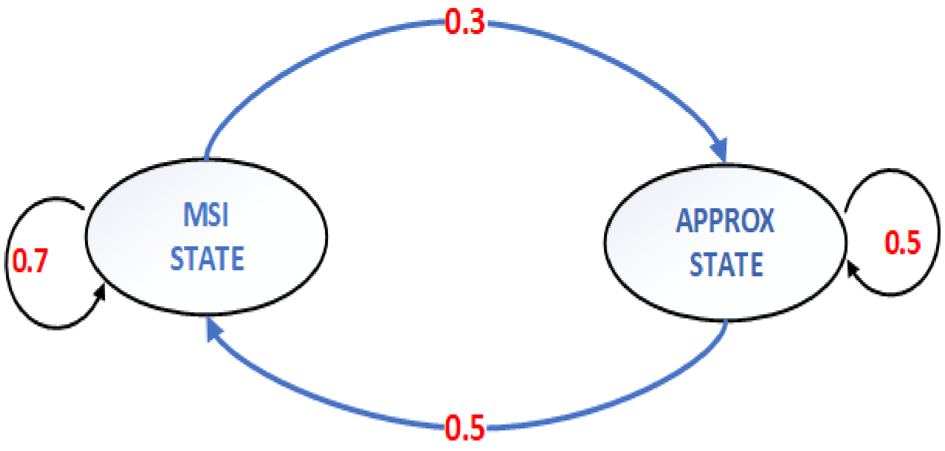


FIGURE 6. The DTMC model for MSI-A.

state ought to be given an equal likelihood of being retrieved once more by the processor during the subsequent execution. A picture of a stochastic matrix is the given Matrix 1. Here, the likelihood of reaching an APPROX state from a non-APPROX state is 0.3, while the probability of accomplishing a loop from a non-APPROX state to itself is 0.7. In the subsequent execution, if the APPROX state was the initial state, then there is a fair likelihood of remaining in the APPROX state, which is equal to 0.5, and there is also an equal probability of obtaining a non-APPROX state, which is also equal to 0.5. Figure 6 illustrates the model where we have described the assumptions for creating DTMC. Markov Chains [29] have been utilised by researchers in order to gain an understanding of the probabilistic performance of verification.

Matrix 1: Stochastic matrix for MSI-A

$$P = \begin{array}{cc} \textit{NonApprox} & \textit{Approx} \end{array} \left(\begin{array}{cc} 0.7 & 0.3 \\ 0.5 & 0.5 \end{array} \right) \begin{array}{c} \textit{NonApprox} \\ \textit{Approx} \end{array}$$

Matrix 2 is an illustration of the limiting distribution. This distribution will not change after the limiting distribution. The model is aperiodic, irreducible and positive recurrent; therefore, we can calculate the steady-state probabilities.

Matrix 2: Limiting Distribution for MSI-A

$$V = \begin{array}{cc} \textit{NonApprox} & \textit{Approx} \end{array} \left(\begin{array}{cc} 0.63 & 0.37 \\ 0.62 & 0.38 \end{array} \right) \begin{array}{c} \textit{NonApprox} \\ \textit{Approx} \end{array}$$

The steady-state probabilities are the final probabilities of the system. Here it conveys that over the long run, the probability of achieving approx state from the non approx state is 0.37 and staying in approx state after the cache-line is in the approx state is 0.38. In equation 2, 'V' is limiting distribution,The initial state probabilities are defined by matrix 1 it is $P(0)$ and P^n is the probability distribution after n commands on Matrix 1. The normalization part of the equation 3 is the right part and the left part of the equation becomes linear system of equations and it is linearly independent.

$$V = \lim_{n\to\infty} P(0) * P^n \quad (1)$$

$$V(P-I) = 0, \quad and \sum_j V_j = 1 \quad (2)$$

TABLE 1. Architecture of the system used for simulation.

Hardware	Specifications
Architecture	X86 with clock frequency of 3.4GHz, 16-cores
Type of Pipeline	In-order pipeline of width 1
Branch Predictor	Bimodal, Penalty : 8 cycles
Private L1 cache	8KB, 4 way, 32 byte block, 3-cycle latency
Shared L2 cache	8KB, 4 way, 32 byte block, 32-cycle latency
Directory Cache Coherency	Number of entries 65536, 8 way, 64 byte block, 32-cycle latency
Main Memory	A miss in L2 cache is considered as hit in main memory, miss penalty is 200 cycles
Power Model	McPat [37]

V. RESULT AND DISCUSSION

We have used the Princeton Application Repository for Shared-Memory Computers (PARSEC) [36], a benchmark suite comprised of multi-threaded programs. This benchmark suite is employed within the TEJAS simulator to measure various applications' gain in energy and cycles. The applications we have used are of different kinds. These applications are shown in table 2.

The energy gain and cycle gain are dependent on the nature of applications. The applications requiring frequent read and write operations on approx cache lines are showing better results than others. The energy gain of various applications is shown in figures 7a and 7b. The cycle gain is shown in figures 7c and 7d.

The analysis phase is hard-coded for a certain percentage of cache lines which do not need any write command

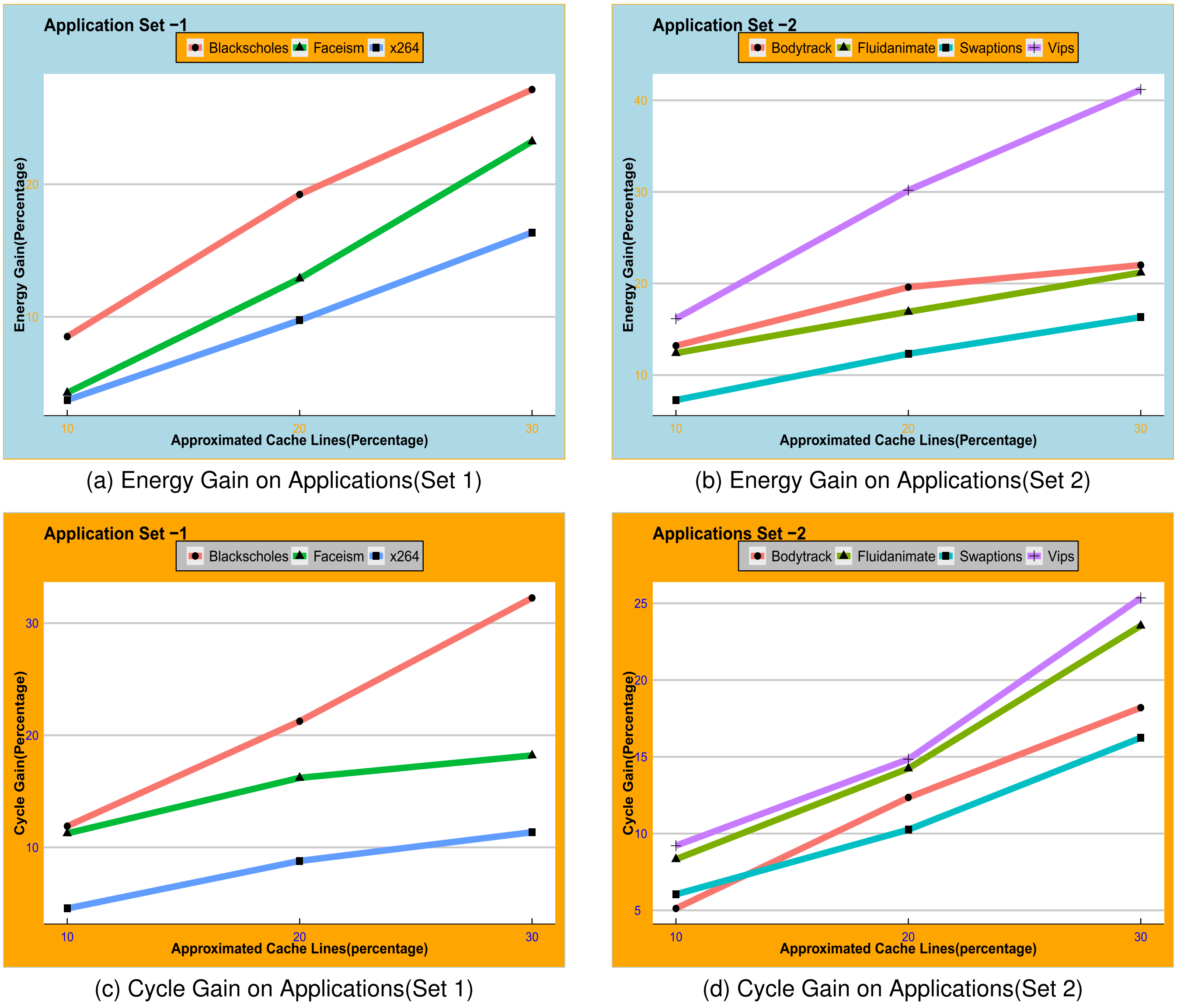


FIGURE 7. **Energy and cycle gain in various applications using approximation and MSI-A.**

TABLE 2. **Various applications for simulations.**

Applications	Nature of Applications
Blackscholes	Financial Analysis
Faceism	Animation
x264	Rendering
Fluidanimate	Animation
Vips	Media Processing
Swaptions	Compilation
Bodytrack	Computer vision

from the processor. Also, the cache lines are not fetched by the processor, and these approximated cache lines are identified by a bit. Approximate cache lines generally do not change the desired output. We have analysed the results with 10 to 30 percent cache lines as approx. We have analysed another cache coherence protocol, MESI, and created an approximable MESI version, MESI-A. We have noted the benefits of the MESI-A protocol (in terms of cycle and energy gain) with 5 to 20 percent of the approximated cache lines. In the following subsection, we have compared MSI-A and MESI-A.

The architecture we have used for simulation is discussed in the table 1, which is the same architecture of our previous work MESI-A, and by using the same structure, we can compare the outcome of approximation on both MSI-A and MESI-A Different applications show different results with approximated cache lines. Our analysis shows 'Vips,' an application for media processing, has shown tremendous energy gain and cycle gain. The energy gain of this application is 16.16, 18.95 and 30.18, for 10, 20 and 30 percent, respectively. The second one to follow is 'Blackscholes' (used for financial analysis), which has an

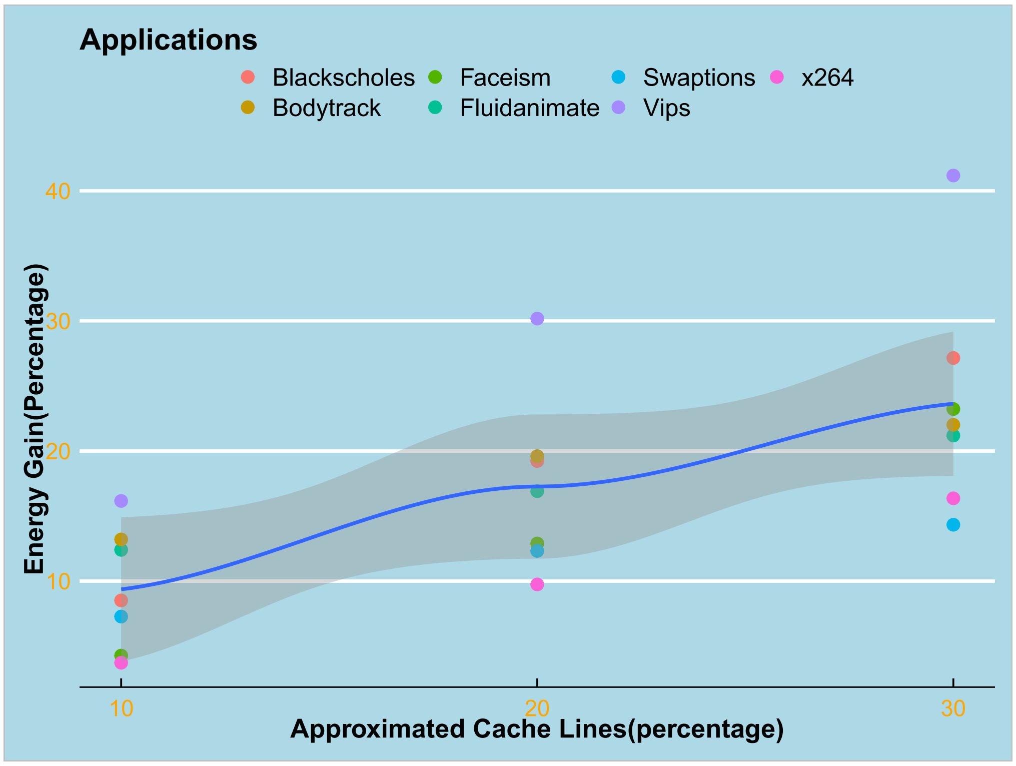


FIGURE 8. Energy gain representation of all applications after MSI-A.

energy gain of 18.15 for the 30 percent approx cache lines, while the cycle gain is 8.51, 11.23, 18.15 for 10, 20 and 30 percent respectively. In our analysis 'X264,' an encoder used for rendering has less energy and cycle gain as compared to other applications. The 'X264' has 3.72, 5.74, 9.36 energy gain for 10, 20 and 30. percent approx cache lines, the cycle gain for 'X264' is 4.55, 8.79, 11.35 for 10, 20 and 30 percent approx cache lines, respectively.

In sections II and III, we have discussed the importance of formal methods in verifying MSI-A and MSI. We have also analyzed the role of the Discrete-Time Markov Chain in establishing the proof of effectiveness for approximated MSI-A. All the LTL specifications are verified. The MSI-A should encapsulate all the LTL specifications of MSI, and we have verified all the LTL specifications of MSI within MSI-A. In addition, we have used a few LTL specifications to ensure the correctness of MSI-A.

We have examined the results of all the applications shown in table 2. By plotting the energy gain of all those applications, we have found that all applications perform better with approximable cache lines and with MSI-A cache coherence protocol. The regression line shown in figure 8 suggest all these applications, on average, are performing 23 percent better, with 30 percent of cache lines. The results will improve if the processor regularly requests the same approximate cache lines. Few applications that get regular updates are far better than the average.

A. MSI-A VS MESI-A

In our prior study with MESI-A [21], we discovered that 'Vips' performed better with 29.58 percent for 20 percent of approximate cache lines. In contrast, MSI-A performed significantly better, with 30.18 percent for the same application. MSI-A is effective, being considerably superior to MESI-A, and therefore, obviously, with MSI-A, the system will save more energy and cycles with approximation infused in it. We have shown the comparison of MESI-A and MSI-A on 20 percent of the cache lines in the table 3.

The fact that the MSI protocol loads all data into the shared state by convention, irrespective of whether the data is not intended to be shared, is the foundation of the MSI protocol's most significant flaw. When we transition the cache block from the shared to the modified state, we are required to send a signal to the other caches, instructing them to invalidate any copies of block X that they may have stored; if these other caches do not store a copy of block X, then we are wasting bus bandwidth and cycles for no reason.

The most likely situation for a program is to read data that is not shared with any other threads and modify that data. We now have a method for differentiating this non-shared (exclusive) data, appreciating the introduction of an exclusive state in MESI. When we modify the vast bulk of our data, we do not need to send out any unnecessary invalidate messages. MESI is basically identical to MSI. However, it is better tuned for the most prevalent scenario.

TABLE 3. Various applications after MESI-A and MSI-A with 20 percent of approximate cache-lines.

Energy Gain of Various Applications	MESI-A	MSI-A
Blackscholes	17.88	19.23
Faceism	11.23	12.89
x264	9.23	9.74
Fluidanimate	17.23	17.91
Vips	29.52	30.18
Swaptions	12.57	12.62
Bodytrack	15.46	19.60

On the other hand, when it comes to approximation, the MSI-A performs significantly better than the MESI-A. The most significant disadvantage associated with this situation is the fact that MESI-A has an extra state that requires a cache line to make more state changes, and this increases the computations. After analyzing both protocols, we determined that the MSI-A protocol offers more potential for energy and cycle savings as compared to MESI-A because energy and cycle gain has increased with the introduction of more approximate cache lines.

VI. CONCLUSION AND FUTURE WORK

One of the modern paradigms of the design of current computing systems is approximate computing. We have developed MSI-A, an approximated version of the MSI protocol. This work illustrates the benefits of using an approximation in the MSI cache-coherence protocol and describes how we have constructed MSI-A. Now considering cache-coherence protocol as a safety-critical system, we need validation of the MSI-A protocol to verify the working of the protocol. We have verified both MSI-A and MSI using LTL specifications in NuSMV. MSI-A is a better protocol if the system contains approximable cache lines. If the cache lines are not approximable, then MSI and MSI-A have trace equivalence. We have also proved the correctness of MSI-A concerning the verification of infinite states. With the assistance of PCTL and DTMC, we have proved that MSI-A is a theoretically enhanced cache coherence protocol, and 0.38 is the steady state probability of achieving the approx state, which will not change any further.

We have employed the TEJAS hardware simulator to examine the cycle gain and the energy gain on various applications that MSI-A offers compared to MSI. With approx cache lines executed in MSI-A ‘Vips ’have shown an energy gain of 30.18 percent, and ‘Blackscholes’have shown 19.23 percent of energy gain. Furthermore, the performance of MSI-A is superior to that of our previous work MESI-A on the cycle gain and energy gain scales. The theoretical and practical applications of MSI-A have shown impressive results. However, it is presumed that the processor is aware of which cache lines may be approximated and which cannot be approximated. We are currently working on coding an approximable and generalised version of the cache coherence protocol on the hardware. Also, we are developing an algorithm to predict the approximable cache lines so we don’t need to hardcode them before the execution. Furthermore, we are working on an approx version write invalidated of cache-coherence protocol.

REFERENCES

[1] R. H. Dennard, F. H. Gaensslen, V. L. Rideout, E. Bassous, and A. R. LeBlanc, “Design of ion-implanted MOSFET’s with very small physical dimensions,” *IEEE J. Solid-State Circuits*, vol. SSC-9, no. 5, pp. 256–268, Oct. 1974.

[2] G. Moore, “Cramming more components onto integrated circuits,” *Electron. Mag.*, vol. 38, no. 8, pp. 114–117, Apr. 1965.

[3] J. S. Miguel, J. Albericio, A. Moshovos, and N. E. Jerger, “Doppelgänger: A cache for approximate computing,” in *Proc. 48th Int. Symp. Microarchitecture*, 2015, pp. 50–61.

[4] G. De Giacomo, A. D. Stasio, F. Fuggitti, and S. Rubin, “Pure-past linear temporal and dynamic logic on finite traces,” in *Proc. 29th Int. Joint Conf. Artif. Intell.*, Jul. 2020, pp. 4959–4965.

[5] O. Mutlu, S. Ghose, J. Gomez-Luna, and R. Ausavarungnirun, “A modern primer on processing in memory,” in *Emerging Computing: From Devices to Systems*. Berlin, Germany: Springer, 2022, pp. 171–243.

[6] A. E. K. Sobel and M. R. Clarkson, “Formal methods application: An empirical tale of software development,” *IEEE Trans. Softw. Eng.*, vol. 28, no. 3, pp. 308–320, Mar. 2002.

[7] S. Liu, V. Stavridou, and B. Dutertre, “The practice of formal methods in safety-critical systems,” *J. Syst. Softw.*, vol. 28, no. 1, pp. 77–87, Jan. 1995.

[8] *Marine Energy-Wave, Tidal and Other Water Current Converters—Part 101: Wave Energy Resource Assessment and Characterization*, Standard 62600-101, 2015.

[9] R. F. V. Preuschen, “The European space agency,” *Int. Comparative Law Quart.*, vol. 27, no. 1, pp. 46–60, 1978.

[10] N. Mehdipour, M. Althoff, R. D. Tebbens, and C. Belta, “Formal methods to comply with rules of the road in autonomous driving: State of the art and grand challenges,” *Automatica*, vol. 152, Jun. 2023, Art. no. 110692.

[11] J. Choi and A. Chlipala, “Hemiola: A DSL and verification tools to guide design and proof of hierarchical cache-coherence protocols,” in *Proc. 34th Int. Conf.* Haifa, Israel: Springer, 2022, pp. 317–339.

[12] M. Graf, G. A. G. Andrade, and L. C. V. dos Santos, “EveCheck: An event-driven, scalable algorithm for coherent shared memory verification,” *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 42, no. 2, pp. 683–696, Feb. 2023.

[13] J. Pang, W. Fokkink, R. Hofman, and R. Veldema, “Model checking a cache coherence protocol of a Java DSM implementation,” *J. Log. Algebr. Program.*, vol. 71, no. 1, pp. 1–43, Mar. 2007.

[14] V. Burenkov and A. Kamkin, “Applying parameterized model checking to real-life cache coherence protocols,” in *Proc. IEEE East-West Design Test Symp. (EWDTS)*, Oct. 2016, pp. 1–4.

[15] R. Joshi, L. Lamport, J. Matthews, S. Tasiran, M. Tuttle, and Y. Yu, “Checking cache-coherence protocols with TLA$^+$,” *Formal Methods Syst. Design*, vol. 22, no. 2, pp. 125–131, Mar. 2003.

[16] X. Li, X. Li, C. Dall, R. Gu, J. Nieh, Y. Sait, and G. Stockwell, “Design and verification of the arm confidential compute architecture,” in *Proc. 16th USENIX Symp. Operating Syst. Design Implement. (OSDI)*, Carlsbad, CA, USA, Jul. 2022, pp. 465–484.

[17] T.-F. Chen, J.-H.-R. Jiang, and M.-H. Hsieh, “Partial equivalence checking of quantum circuits,” in *Proc. IEEE Int. Conf. Quantum Comput. Eng. (QCE)*, Sep. 2022, pp. 594–604.

[18] C. J. Banks, M. Elver, R. Hoffmann, S. Sarkar, P. Jackson, and V. Nagarajan, “Verification of a lazy cache coherence protocol against a weak memory model,” in *Proc. Formal Methods Comput. Aided Design (FMCAD)*, Oct. 2017, pp. 60–67.

[19] A. Cimatti, E. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, and A. Tacchella, “NuSMV 2: An opensource tool for symbolic model checking,” in *Proc. Int. Conf. Comput. Aided Verification*. Cham, Switzerland: Springer, 2002, pp. 359–364.

[20] S. R. Sarangi, R. Kalayappan, P. Kallurkar, S. Goel, and E. Peter, “Tejas: A Java based versatile micro-architectural simulator,” in *Proc. 25th Int. Workshop Power Timing Model., Optim. Simul. (PATMOS)*, Sep. 2015, pp. 47–54.

[21] A. Saraswat, K. Abhishek, M. R. Ghalib, A. Shankar, M. Alazab, and B. Nongpoh, "Towards energy efficient approx cache-coherence protocol verified using model checker," *Comput. Electr. Eng.*, vol. 97, Jan. 2022, Art. no. 107482.

[22] P. Chatterjee, H. Sivaraj, and G. Gopalakrishnan, "Shared memory consistency protocol verification against weak memory models: Refinement via model-checking," in *Proc. Int. Conf. Comput. Aided Verification*. Cham, Switzerland: Springer, 2002, pp. 123–136.

[23] D. Chaiken, C. Fields, K. Kurihara, and A. Agarwal, "Directory-based cache coherence in large-scale multiprocessors," *Computer*, vol. 23, no. 6, pp. 49–58, Jun. 1990.

[24] S. Liu, K. Pattabiraman, T. Moscibroda, and B. G. Zorn, "Flikker: Saving dram refresh-power through critical data partitioning," in *Proc. 16th Int. Conf. Architectural Support Program. Lang. Operating Syst.*, 2011, pp. 213–224.

[25] E. E. Bilir, R. M. Dickson, Y. Hu, M. Plakal, D. J. Sorin, M. D. Hill, and D. A. Wood, "Multicast snooping: A new coherence method using a multicast address network," in *Proc. 26th Int. Symp. Comput. Archit.*, 1999, pp. 294–304.

[26] R. Alur, C. Courcoubetis, T. A. Henzinger, and P.-H. Ho, "Hybrid automata: An algorithmic approach to the specification and verification of hybrid systems," in *Hybrid Systems*. Berlin, Germany: Springer, 1992, pp. 209–229.

[27] K. L. McMillan, "Parameterized verification of the flash cache coherence protocol by compositional model checking," in *Proc. Adv. Res. Work. Conf. Correct Hardw. Design Verification Methods*. Cham, Switzerland: Springer, 2001, pp. 179–195.

[28] L. Ivanov and R. Nunna, "Modeling and verification of cache coherence protocols," in *Proc. IEEE Int. Symp. Circuits Syst.*, May 2001, pp. 129–132.

[29] A. Remke, B. R. Haverkort, and L. Cloth, "Model checking infinite-state Markov chains," in *Proc. Int. Conf. Tools Algorithms Construction Anal. Syst.* Cham, Switzerland: Springer, 2005, pp. 237–252.

[30] Y. Lyu, X. Qin, M. Chen, and P. Mishra, "Directed test generation for validation of cache coherence protocols," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 38, no. 1, pp. 163–176, Jan. 2019.

[31] R. Alur and D. Dill, "Automata for modeling real-time systems," in *Proc. Int. Colloq. Automata, Lang., Program.* Cham, Switzerland: Springer, 1990, pp. 322–335.

[32] A. Felty and F. Stomp, "A correctness proof of a cache coherence protocol," in *Proc. 11th Annu. Conf. Comput. Assurance*, 1996, pp. 128–141.

[33] F. Pong and M. Dubois, "Formal verification of complex coherence protocols using symbolic state models," *J. ACM*, vol. 45, no. 4, pp. 557–587, Jul. 1998.

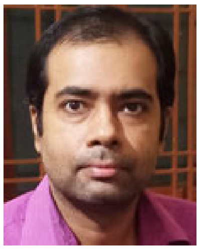
[34] S. Burckhardt, R. Alur, and M. M. Martin, "Verifying safety of a token coherence implementation by parametric compositional refinement," in *Proc. Int. Workshop Verification, Model Checking, Abstract Interpretation*. Cham, Switzerland: Springer, 2005, pp. 130–145.

[35] T. Wahl, "Fairness and liveness," Tech. Rep., 2010. [Online]. Available: http://www.ccs.neu.edu/home/wahl/Publications/fairness.pdf

[36] Princeton. (2011). *Parsec 3.0 Benchmark Suite*. [Online]. Available: https://parsec.cs.princeton.edu/parsec3-doc.htm

[37] N. P. Carter, *Schaum's Outline of Computer Architecture*. New York, NY, USA: McGraw-Hill, 2001.

ANANT SARASWAT received the M.Tech. degree from the National Institute of Technology Meghalaya, and the M.Sc. degree from Pondicherry University. He is currently a Ph.D. Research Scholar with the Department of Computer Science and Engineering, National Institute of Technology Patna. His research interests include theoretical computer science, and formal verification and approximation.



KUMAR ABHISHEK received the Ph.D. degree in computer science and engineering from the National Institute of Technology Patna, India. He is currently an Assistant Professor with the Department of Computer Science and Engineering, National Institute of Technology Patna. He has published more than 100 research papers in various renowned international conferences and SCI indexed journals. His research interests include RDF, semantic web, ontology, semantic sensor web, ontology mapping, and approximation.

HITESHWAR KUMAR AZAD received the Ph.D. degree in computer science and engineering from the National Institute of Technology Patna, India. He is currently a Senior Assistant Professor with the School of Computer Science and Engineering, Vellore Institute of Technology, Vellore, India. He has published several research papers in prestigious international conferences and SCI indexed journals. His research interests include information retrieval, query expansion, data mining, NLP, semantic web, and linked open data.

S. SHITHARTH received the Ph.D. degree from the Department of Computers Science and Engineering, Anna University. He is currently pursuing the Ph.D. (Visiting) degree with The University of Essex. He has worked in various institutions with a teaching experience of seven years. He is also an Associate Professor with Kebri Dehar University, Ethiopia. He has published in more than 51 international journals and 20 international and national conferences. He has even published four patents in IPR. His current research interests include cyber security, blockchain, critical infrastructure and systems, and network security and ethical hacking. He is also an Active Member of IEEE Computer Society and five more professional bodies. He is also a member of the International Blockchain Organization. He is a certified hyperledger expert and a certified blockchain developer. He is an active researcher, a reviewer, and an editor for many international journals.