Computers, Materials & Continua
DOI: 10.32604/cmc.2023.032822
Article

Shared Cache Based on Content Addressable Memory in a Multi-Core Architecture

Allam Abumwais* **and Mahmoud Obaid**

Computer Systems Engineering, Arab American University, Jenin, 240, Palestine
*Corresponding Author: Allam Abumwais. Email: allam.abumwais@aaup.edu
Received: 30 May 2022; Accepted: 20 September 2022

Abstract: Modern shared-memory multi-core processors typically have shared Level 2 (L2) or Level 3 (L3) caches. Cache bottlenecks and replacement strategies are the main problems of such architectures, where multiple cores try to access the shared cache simultaneously. The main problem in improving memory performance is the shared cache architecture and cache replacement. This paper documents the implementation of a Dual-Port Content Addressable Memory (DPCAM) and a modified Near-Far Access Replacement Algorithm (NFRA), which was previously proposed as a shared L2 cache layer in a multi-core processor. Standard Performance Evaluation Corporation (SPEC) Central Processing Unit (CPU) 2006 benchmark workloads are used to evaluate the benefit of the shared L2 cache layer. Results show improved performance of the multicore processor's DPCAM and NFRA algorithms, corresponding to a higher number of concurrent accesses to shared memory. The new architecture significantly increases system throughput and records performance improvements of up to 8.7% on various types of SPEC 2006 benchmarks. The miss rate is also improved by about 13%, with some exceptions in the sphinx3 and bzip2 benchmarks. These results could open a new window for solving the long-standing problems with shared cache in multi-core processors.

Keywords: Multi-core processor; shared cache; content addressable memory; dual port CAM; replacement algorithm; benchmark program

1 Introduction

Nowadays, multi-core architecture is widely used in processor design, and the cache hierarchy is moved from one level cache to multi-level cache in the modern multi-core system [1–4]. Cache Level 1 (L1) is usually private for each core while other levels are either private or shared. Within a multi-core system, the interconnection network connects cores to the shared level of caches (L 2 or L 3 in most systems). Therefore, shared cache architecture affects overall system performance [5]. This work adopted two levels of the cache hierarchy, private L1 cache and shared L2 cache, to improve cores communication.

As the cores count increases, the contention on the shared cache between cores becomes more intense because only one core can access the shared cache simultaneously. In addition, there are also some other problems that need to be improved. First, a quick replacement of data that is not accessed yet can occur due to the limited shared cache space architecture and competition between cores to access cache lines. This causes shared data access failure and requires, in most cases, reading data from lower level memory [1]. As a result, the system performance will be decreased [6]. Second, In addition to cache architecture, the cache replacement algorithm also has the main function of determining the effective response of the cache. The replacement algorithm's goal is to replace the block that will not be accessed in the near future when the cache becomes full or the data becomes un-useful.

In the previous work [7] a special purpose shared memory architecture based on CAM called DPCAM was presented. In addition, an efficient replacement algorithm, called NFRA, which is based on hardware rather than software executed by the cache controller was also presented. The main purpose of the DPCAM and NFRA design is to allow simultaneous access and achieve less access latency to the shared memory. The DPCAM design was implemented on cyclone V Field Programmable Gate Array (FPGA) family as a standalone memory, and then the performance related to power consumption and access latency was estimated. However, evaluation of the DPCAM as a standalone memory could not reflect its performance within a multi-core system. So, the performance of DPCAM inside a multi-core system should be evaluated.

In this paper, a new L2 shared cache architecture based on DPCAM is embedded inside the multi-core. DPCAM has two dedicated ports, one for reading and the other for writing, to enable simultaneous access and reduce the contention over shared memory. Moreover, This shared cache includes a modified NFRA replacement algorithm based on simple hardware, that reduces the access latency and miss ratio. Performance of the multi-core with DPCAM is measured in terms of throughput and access miss rate. All results are compared with those of traditional multi-core systems, which use the Least Recently Used (LRU) replacement algorithm and set-associative in the L2 cache.

The rest of the paper is organized as follows. In Section 2, the literature review of shared cache in multi-core systems is summarized. In Section 3, a detailed description of DPCAM within a multi-core architecture is presented. In Section 4, a modified replacement policy for DPCAM inside multi-core system is given. In Section 5, the implementation of the DPCAM within a multi-core system and the performance analysis are presented. Finally, Section 6 represents the conclusion of this work.

2 Related Works

Reference [3] states that due to the increasing number of on-chip cores and the deterioration of power-performance penalty of off-chip memory locations, shared Last-Level Caches (LLC) have emerged as among the most critical drivers of multi-core efficiency in today's architecture. A new hardware-software technique was proposed to partition the shared cache between cores to allow simultaneous execution and reduce the contention. Pan's discussion on how multi-cores improve shared LLC performance will be an important piece of literature for this research in understanding shared cache based on content-addressable Memory in a multi-core.

Reference [6] contributes to the research topic by Chip Multiprocessor (CMP) has emerged as the de-facto standard for next-generation scalable multiprocessor architecture. Having better cache utilization by selective data storage is Tiled CMP (TCMP) is becoming common technological advancement. A large number of cores typically share the Last Level Cache in CMP. In contrast to static Non-Uniform Cache Architecture (NUCA), which has a set address mapping strategy, Dynamic NUCA (DNUCA) permits blocks to be relocated closer to the processor cores when the workload

demands it [6]. In LLC, the NUCA is used to partition it into many banks; each may be accessed separately from the others. The DNUCA-based CMP may consistently distribute workloads to each bank, resulting in improved worldwide utilization.

The previous article [7] findings support incorporating DPCAM as tiny shared cache memory inside multi-core CPUs to improve performance. Many related works improve the shared level of caches in multi-core systems. By coming up with a new design that addresses the multi-core systems that address the issue of gaping memory speed and processor, there is a fundamental aspect of solving the issue of our research on shared cache based on content-addressable memory in a multi-core architecture. Abumwais contributes to the discussion by proposing that DPCAM is a novel architecture for a specialized pipeline cache memory for multi-core CPUs that are being shown DPCAM [7]. Also included a novel replacement algorithm based on hardware, referred to as an NFRA, which is intended to lower the cost inefficiency of the cache controller while simultaneously improving the delay of cache accesses. It was discovered via the experiments that the delay for writing and read operations is much smaller when compared to a predefined cache memory. Furthermore, it has been demonstrated that the latency of a write operation is almost constant irrespective of the size of the DPCAM array used.

However, low-power advantages come at the expense of a high write latency. Some research focused on reducing write latency or minimizing its effects on power. Reference [8] proposed an adaptive shared cache that allows LLC configurations to be modified to applications in multi-core systems during runtime execution. A fairness access method to the shared cache between cores was proposed in [9]. The fairness method assigns a shared cache to multiple cores to achieve balance access and reduce the contention.

With the Internet of Things being able to generate high amounts of data creating high energy and performance in the traditional CPUs and Graphics Processing Unit (GPUs), there is an important improvement in cache and memory bandwidth. Reference [10] constitutes our research topic by proposing a Customizable Associative Processor (CAP), which speeds computing by utilizing several parallel memory-based cores capable of estimated or precise matching, which can be configured to meet specific requirements.

Through associative co-processors, there is always a chance to have special-purpose computers run heavy programs, as [11] explained. Reference [11] proposes using current hardware components to construct a multi-core processor for particularly unique computer systems. This work aims to develop an associative co-processor centered on the FPGA platform for high-performance multi-core processors, specifically for systems that perform associative operations and digital storage functions. The procedures of retrieval and sorting are commonly utilized in both user- and system-level systems.

The use of promising technology is fundamental in solving the issue of having a multi-core architecture that replaces the Static Random Access Memory (SRAMs) that have a low cache. Reference [12] contributes to the topic of research by stating that one of the most serious issues with Spin Torque Transfer RAM (STT-RAMs) is the significant error rate that results from stochastic switching during write operations. Cache management algorithms have a significant influence in determining the number of write operations performed into caches. As a result, it is required to develop cache replacement methods to consider the additional issues posed by STT-RAM caches. For L2 caches, he offers a cache replacement mechanism dubbed Least Error Rate (LER), which he claims will cut the error rate by 50%.

Attempts to address the problem have taken different research directions, leading to major developments in parallel search and Artificial Intelligence (AI) applications for non-volatile Ternary

Content Addressable Memory (TCAM). Attempts to implement SRAM-based TCAM in a shared cache for the parallel are also broadly discussed in the literature [13]. Efforts to implement a new design of non-volatile TCAM (nvTCAM) cells have targeted multiple outcomes, such as a reduction in area overhead and power consumption [14], promising great improvements in TCAM performance.

A novel hybrid memory system composed of Dynamic RAM (DRAM) and Non-Volatile Memory (NVM) has also been proposed to improve the access latency and throughput of the computer system [15]. DRAM-NVM presents a Multi-hash (MuHash) algorithm to solve the problem of limited size in series write operation and to decrease the unnecessary read accesses. DRAM-NVM has been implemented using Intel Optane memory on a single-core [16].

This literature gives input to the research topic: pipeline shared cache based on DPCAM memory in a multi-core architecture. The main contribution of this article is to improve multi-core performance by exploiting DPCAM and the NFRA technique. The comparison points between the proposed work and the existing works are shown in Table 1.

Table 1: Comparison between the proposed work and the existing works

Work	[3,8,9]	[12]	[14]	[16]	[10]	[7]	Proposed work
Memory types	Set-associative	STT-RAM	SRAM-based TCAM	Hybrid DRAM and NVM	Resistive CAM (RCAM)	CAM LLC	DPCAM
Replacement algorithm	Standard and modified LRU	LER	-	MuHash	-	NFRA	Modified NFRA
Implementation	LLC cache within multi-core system	The implementation is performed on the Advanced Reduce Instruction Set Computer (RISC) machine (ARM) processor provided by gem5.	The use of TCAM and nvTCAM for search operations and AI applications is investigated.	Has been implemented using Intel Optane memory on a single-core	As a stand-alone and as a hybrid computing unit besides CPU and GPU.	FPGA standalone memory	L2 shared cache within multi-core system
Aimed	Proposed a new hardware-software replacement technique to partition a shared cache among the tasks of the parallel application.	Proposed a LER replacement algorithm that improves the throughput and power consumption compared LRU.	By outlining the current state of SRAM-TCAM and nvTCAM research, this paper aims to inspire more advanced research.	Improve the access latency and throughput of the computer system.	Accelerate computation approximately in CPU and GPU.	Allow simultaneous access, achieve less access latency and reduce the power consumption.	Improve performance of the multi-core systems.

3 Proposed DPCAM Inside Multi-Core System

3.1 DPCAM Architecture

To the best of our knowledge, DPCAM that was proposed in [7] is the first work that addresses using the content addressable memory as a shared cache and taking advantage of its features. Fig. 1 shows DPCAM within a multi-core system architecture.

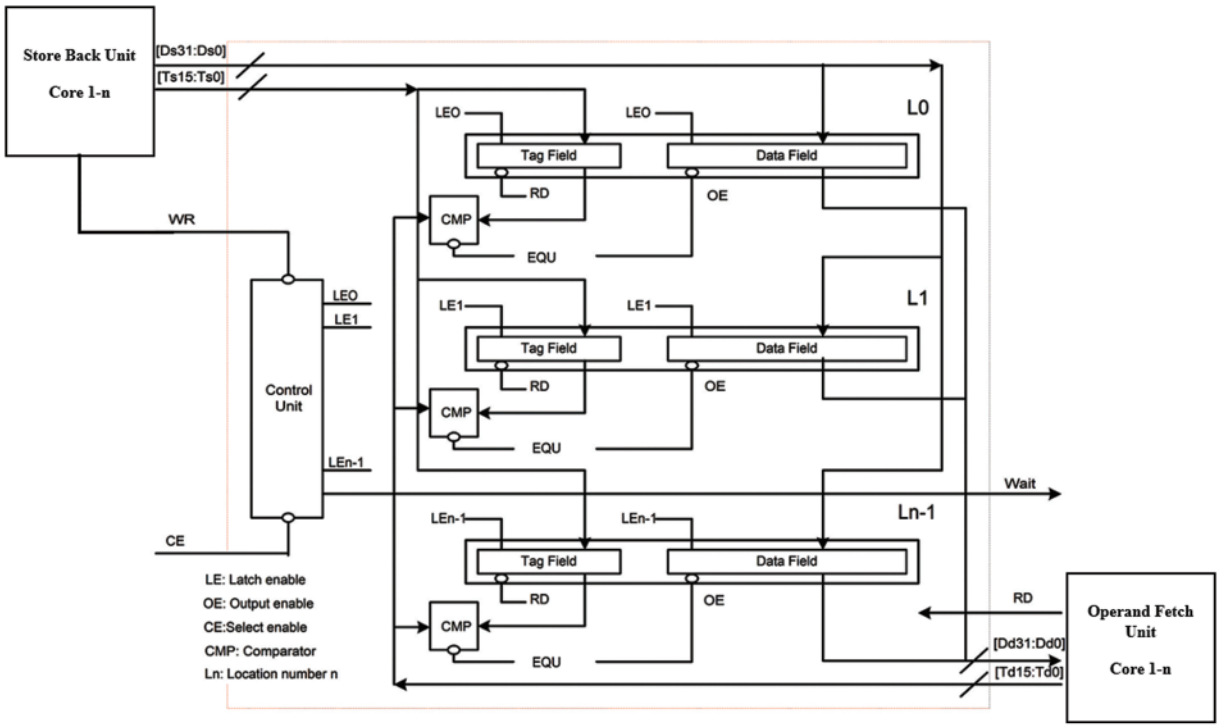

Figure 1: DPCAM architecture within a multi-core system

DPCAM was redesigned for a special purpose to be employed as a shared cache in the multi-core system. This architecture allows simultaneous access to the same shared module based on dual port architecture. It reduces the access latency due to the simple architecture and simple NFRA technique. The DPCAM shared cache is mainly divided into two techniques; dual port and NFRA. The main points are as follows:

A. The Store Back (SB) unit of all cores in the multi-core system can use the first port (write port) of DPCAM to write the shared data where each new write is controlled by the Control Unit (CU).
B. The Operand Fetch (OF) unit of all cores in the multi-core system can use the second port (read port) of DPCAM to read the shared data, which has been stored by other cores.
C. For the writing mechanism, the DPCAM memory architecture is built to store the shared data in the first available empty location or to the oldest location written in case of no empty location. Therefore, there is no unused DPCAM location.
D. The shared data in any location will not be overwritten before at least n times writing elapses, where n is the number of DPCAM locations or lines. All principles of DPCAM in Fig. 1 are applied in this work using the Gem5 simulator.

3.2 Multi-Core Using DPCAM Architecture

The main parts of the multi-core system are the cache levels, shared cache architecture, cache coherence, Interconnection Network (IN) and the main memory. Fig. 2 shows the main parts of the proposed multi-core architecture. This architecture is built based on two-level cache, IN using crossbar switch, Modified-Exclusion-Shared-Invalid (MESI) cache coherence protocol and main memory access.

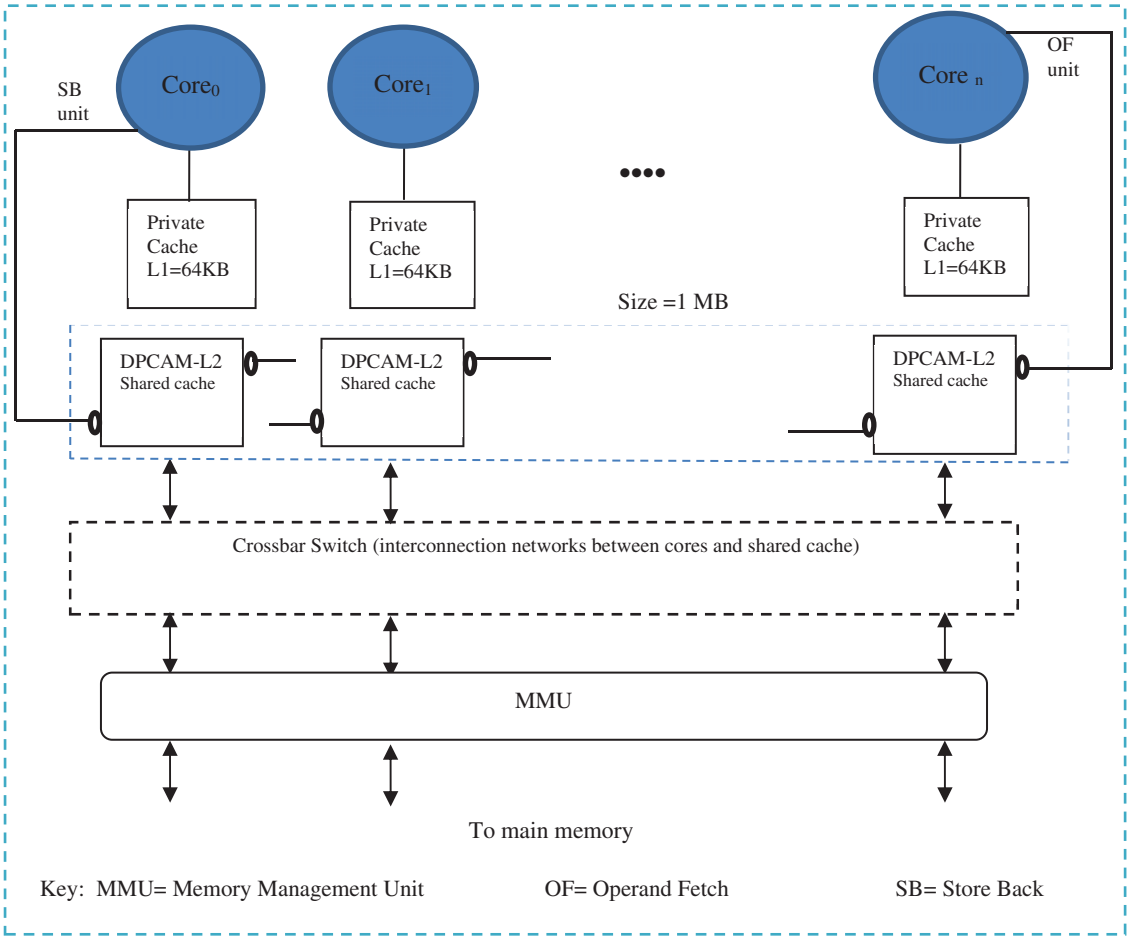

Figure 2: Multicore architecture using DPCAM shared cache

Accessing the shared cache memory still has the problem of contention between the cores. This leads to an increase in the access latency and hence decreases the system throughput. Both cache levels design and INs among cores are the main factors that affect the performance in multi-core systems [1]. In the multi-core Architecture Using DPCAM, the DPCAM is employed as a global L2 (shared cache) and the local data cache is graded as a set-associative L1 cache.

When the Memory Management Unit (MMU) loads the threads to the L1 instruction cache, it loads the local data of the thread to the local data cache, and the shared variables to the DPCAM. A shared variable, of course, is seen and can be accessed by all cores in multi-core system [1,2,4]. During the thread execution, the OF unit can equally access the local and the shared cache to fetch the required operands. The SB unit writes the resulting shared variables to be accessed by other cores. The tag of

the variable includes its address, its version number and its valid bit. In the memory hierarchy access, cores first check L1 cache and L2 cache respectively. A miss is returned if no cache hits are found and the request for data is forwarded to the main memory. On the other hand, shared data should be checked whether another core has changed them or not in order to preserve the consistency between cores. This action has been achieved using MESI cache coherence protocol.

The proposed multi-core architecture has been compared with the traditional multi-core architecture using Gem5 simulator. Both architectures have the same parts of level of caches, cache coherence, IN and main memory. The shared cache memory is the sole difference, as the proposed design uses DPCAM in shared cache and the traditional design uses a four-way set-associative shared cache. The comparison results for both architectures will be shown in Section 5.

4 A Modified Replacement Policy for DPCAM

In the proposed DPCAM, a NFRA replacement algorithm and a new writing mechanism were presented in [7]. The write operation is based on a pointer is implemented in the CU. If the core writes shared data in Location x (Lx), the next write operation (from the same of other cores) will store shared data into location Lx + 1. This mechanism can be repeated until location Ln−1, then returns to L0 to overwrite the oldest shared data. This technique is practical for implementation in a single-core or a limited number of cores inside a multi-core system. The main problem of this mechanism is that an empty DPCAM location that stores a shared data related to a process recently terminated can be found with any DPCAM location. e.g., if the pointer points to location L10 to write a shared data, then the next coming data from any core will be stored in L11 automatically, whether it is empty or overwriting if it is full. This happens although there may be several empty locations in DPCAM because the CU automatically writes to the next location. Of course, this problem will decrease the system throughput because the shared data may be replaced before its use.

As a solution to this problem, a modified mechanism of the write operation and a replacement policy are proposed. For this purpose, a new bit is added to the tag field in each DPCAM location. Whether a location is valid to be written (if it is empty) or invalid to be written (if it is full) is indicated using the most significant bit in the tag field tag {16}. If tag {16} equals zero, it means that the location is empty and can be written, otherwise, the location is full and it cannot be written except if it is pointed to by the CU and no other empty location in DPCAM exists.

Fig. 3 shows the flowchart of the proposed mechanism, in the beginning, the CU was set to initially point to the first location and test the tag {16}. Because the DPCAM is empty, the tag {16} always equals zero in all locations. Therefore, the first write operation will occur on the first line of the DPCAM, and so on until location n−1 (Ln−1). This means n write operations can happen before there is a need for overwriting any location. When the DPCAM becomes full, the pointer is pointed to the current location (L[k]) and the tag {16} is tested. If tag {16} equals zero then the shared data is directly written to the current pointer location and the pointer is incremented. Otherwise, if tag {16} equals one the tag {16} for another location is tested until finding an empty location. This means that writing the first empty memory location and then return to the current location (L[K]) for the next write operation. In case that no empty location is found, the pointer returns to its current location and overwrites the new shared data then it is incremented to point to location (L[K++]).

Note that tag {16} values are tuned by the scheduler in the compiler which must ensure that data is used during allowed time. Furthermore, if any core requires the read version of shared data for longer than the allowed time, it can store it in its private L1 cache.

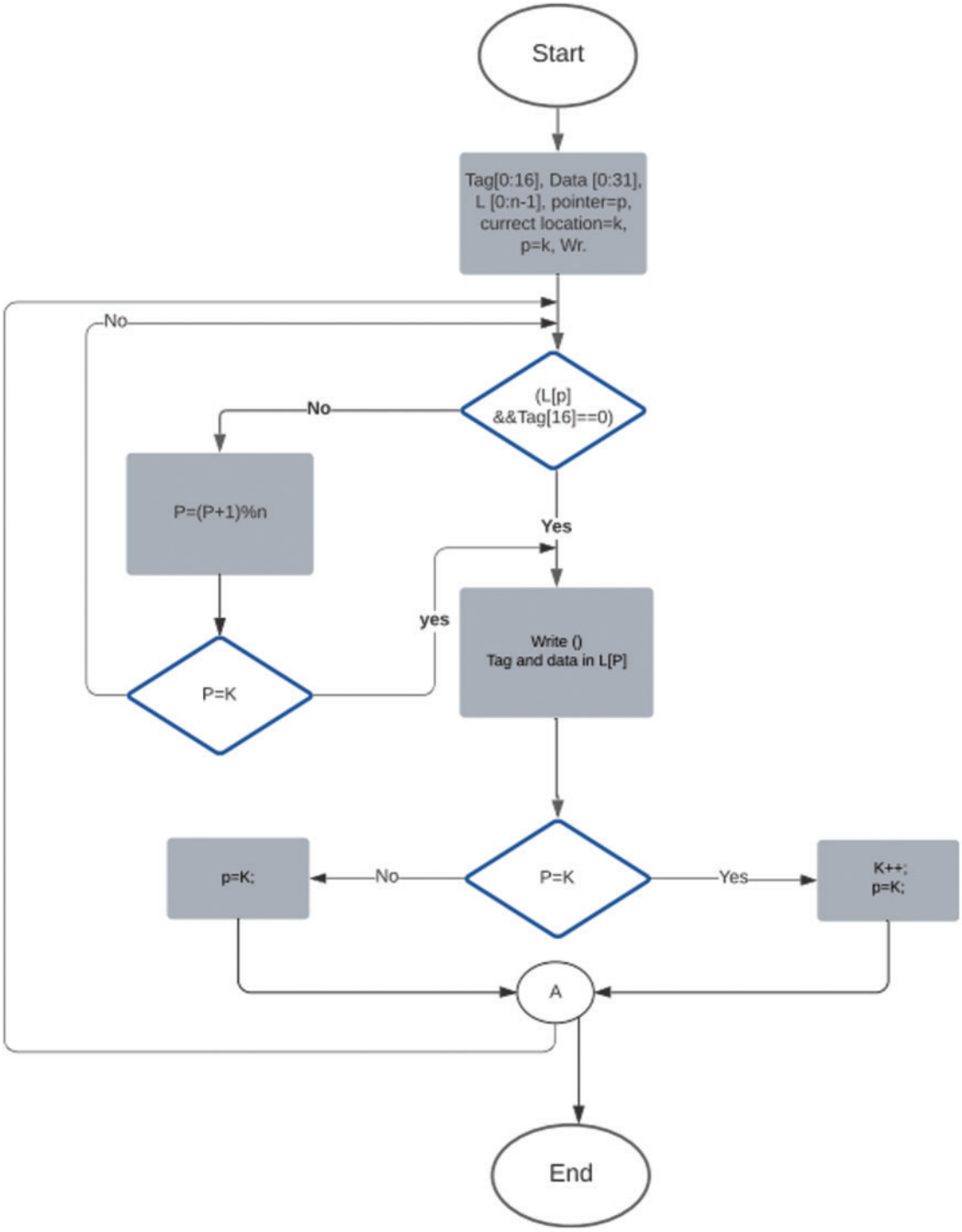

Figure 3: A modified replacement policy for DPCAM

5 Evaluation the MPCAM Within Multi-Core System

5.1 Experimental Setup

The Gem5 simulator and various types of SPEC CPU 2006 benchmarks are chosen to study the performance of multi-core system based DPCAM as the second level of cache (shared cache). In Gem5, we use a full-system functional and timing simulator of multi-core memory system called Ruby. Ruby is a part of Gem5 project. It provides full specifications and flexible cache memory such as cache architecture, cache coherence protocols, cache replacement algorithm and various IN models [17]. Ruby uses a first level of cache as private L1 data and instruction, and a second level of cache as shared L2. In our proposed study, same size and cache coherence protocol type of L1 and new architecture of DPCAM with same size of traditional set-associative L2 cache are used. Table 2 shows the detailed configuration of the multi-core architecture (number of processors, cache hierarchy, cache coherence protocol and main memory) used in our proposed system.

Table 2: The configurations of the experimental multi-core architecture

System component	Values
No. of processor	4
Processor specifications	Out-of-order, 2,6 GHz.
Cache levels	2
L1 cache specifications	2-way set-associative, size = 64 KB and block size = 64 B.
L2 cache specifications	Size = 1MB and size of each Bank = 256 KB.
Cache model	Non Uniform cache Access (NUCA).
Memory	1 GB

5.2 Simulation Result

According to the above configuration, the proposed architecture has been simulated with various test benchmarks. Figs. 4 and 5 show a comparative study of system performance in terms of Instruction per Cycle (IPC) and miss rate between a multi-core system that uses L2 based set-associative shared cache and the proposed multi-core system which uses L2 based DPCAM shared cache. The proposed architecture exploits the features for DPCAM and NFRA replacement algorithm to improve its performance. All of these findings are displayed in Figs. 4 and 5 are normalized to the traditional architecture result for each benchmark.

Table 3 and Fig. 4 shows the performance of the multi-core system of the traditional and the proposed DPCAM design in terms of IPC with various benchmark programs. The average IPC for seven benchmark programs for both designs is shown on Table 4. Following that, Eq. (1) is used to determine the overall performance improvements in terms of IPC.

$$\text{Throughput improvements} = 1 - \left(\frac{mean\ of\ traditional\ design}{mean\ of\ DPCAM\ design}\right) \times 100\% \tag{1}$$

It can be observed that the DPCAM architecture improves performance by about 8.7% from the average of various benchmark programs. In addition, it can be seen that among all test programs, mcf and astar do not produce any improvement. On the contrary, the system performance declines in

terms of IPC. This happens because some benchmark programs are memory intensive and they may not benefit from using the shared cache. This is the case for the mcf and astar programs [18].

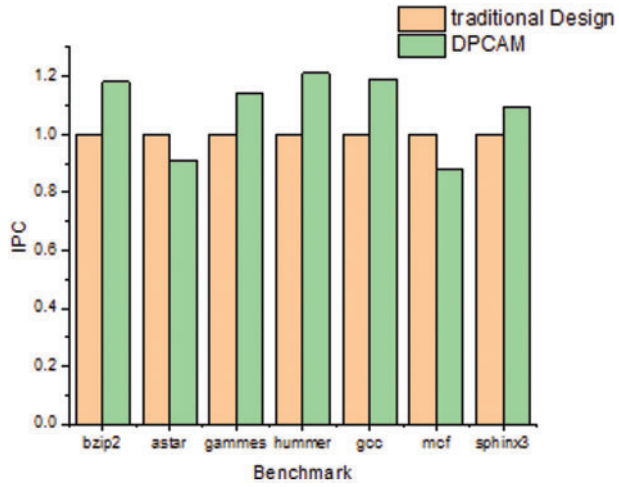

Figure 4: System performance in (IPC) comparison

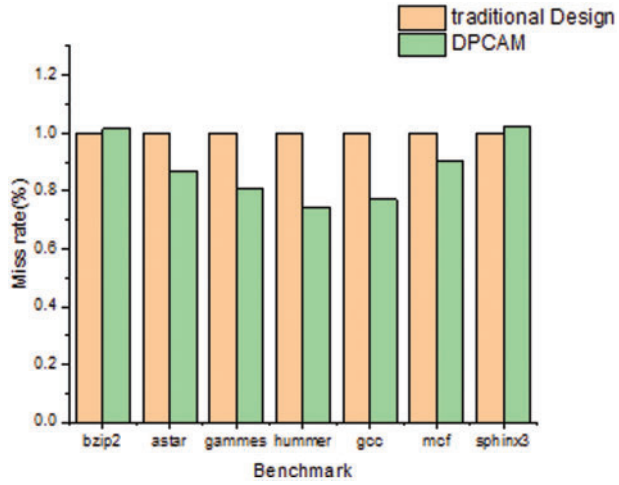

Figure 5: System miss rate comparison

Table 3: The IPC for seven benchmark programs

Benchmark name	IPC for traditional design	IPC for DPCAM design
Bzip2	0.90	1.062
astar	0.50	0.455
gammes	0.70	0.812
hummer	0.78	0.9516
gcc	0.83	0.9877

(Continued)

Table 3: Continued

Benchmark name	IPC for traditional design	IPC for DPCAM design	
mcf	0.92	0.7636	
sphinx3	0.87	0.9918	
	Mean 0.786	Mean 0.861	Improvement 8.71%

Table 4: The access miss rate for seven benchmark programs

	Traditional design			DPCAM design		
Benchmark name	Number of L2 cache references	Number of misses	Miss rate	Number of DPCAM references	Number of misses	Miss rate
Bzip2	30491718	838316	2.7%	30491718	875213	2.87%
astar	5234678	183213	3.5%	5234678	20155	0.38%
gammes	8971498	376802	4.2%	8971498	18840	0.21%
hummer	10967781	570324	5.2%	10967781	20400	0.19%
gcc	9873018	473904	4.8%	9873018	18265	0.185%
mcf	5994666	194826	3.25%	5994666	20981	0.35%
sphinx3	34491313	896716	2.6%	34491313	961917	2.78%

Table 4 and Fig. 5 present the miss rate for the proposed DPCAM with NFRA instead of the LRU replacement policy in traditional design. The number of L2 cache references, number of misses, and the miss rate for seven benchmark programs are shown on Table 4. In the proposed architecture, the DPCAM miss happens if the data has not yet been created due to a scheduling issue or if it has been overwritten due to a size limitation. Otherwise, if the shared data is produced it becomes available to all OFs units.

Generally, it can be observed that the miss rate of the proposed system is clearly reduced with an average of about 13% compared with traditional design. However, the miss rate is reduced for all test programs except for test programs sphinx3 and bzip2. Sphinx3 has produced around a 1.9% increase in miss rate. On the other hand, bzip2 has produced a 1.2% increase in miss rate. This is because of the large number of reference instructions and iterations for these two types that have about three billion load instructions. Finally, it can be observed that in some benchmarks, the shared cache miss rate is not enough to judge the system performance, because it does not take into account the cause of miss, penalty of misses, and the latency of access in hit. This is obvious in bzip2 and sphinx3 where they do not improve the miss rate but the performance is clearly increased.

6 Conclusion

The proposed architecture exploits the features of content addressable memory and NFRA replacement algorithm to improve the system performance. The performance of the proposed system was evaluated using the Gem5 simulator and a set of SPEC CPU 2006 benchmarks and compared to a traditional multi-core system. The comparison of system performance in terms of IPC and miss rate

between a multi-core system with associative shared L2 cache and a multi-core system with L2-based shared DPCAM cache.

The experimental results showed a significant improvement in the performance of the proposed architecture by up to 8.7% in terms of IPC for average types of benchmarks. This improvement is due to the architectural features of DPCAM that enable concurrent reads and writes in shared memory without any contention. The NFRA design contributes to a significant improvement by enabling parallel seeks and sequential writes to all memory locations. In addition, the error rate is reduced by about 13% for average benchmark types, except for the sphinx3 and bzip2 benchmarks. This improvement is due to the effectively modified NFRA, which increases the lifetime of shared data on DPCAM. As a result, the missing access only occurs when there is a delay in the production of the shared data by the processors or after a premature overwrite of the shared data before reading due to cache capacity limitations.

Funding Statement: The authors received no specific funding for this study.

Conflicts of Interest: The authors declare that they have no conflicts of interest to report regarding the present study.

References

[1] J. Hennessy and D. Patterson, *Computer Organization and Design: The Hardware Software Interface*, 2nd ed., Cambridge, United States: Elsevier, pp. 520–550, 2020.

[2] W. Stallings, "Computer organization and architecture designing for performance," in *Pearson Education International*, 8th ed., New Jersey, USA: Prentice Hall, pp. 111–140, 685–699, 2013.

[3] A. Pan and V. Pai, "Runtime-driven shared last-level cache management for task-parallel programs," in *Proc. of the Int. Conf. for High Performance Computing, Networking, Storage and Analysis*, Austin Texas, USA, pp. 1–12, 2015.

[4] A. Abumwais and A. Ayyad, "The MPCAM based multi-core processor architecture: A contention free architecture," *Wseas Transactions on Electronics*, vol. 9, no. 13, pp. 105–111, 2018.

[5] J. Mars, R. Hundt and N. A. Vachharajani, "Cache contention management on a multicore processor based on the degree of contention exceeding a threshold," *Patent and Trademark Office*, Patent No. 9,268,542, pp. 1–25, 2016.

[6] S. Das and H. K. Kapoor, "Towards a better cache utilization by selective data storage for CMP last level caches," in *29th Int. Conf. on VLSI Design and 15th Int. Conf. on Embedded Systems (VLSID)*, Kolkata, India, pp. 92–97, 2016.

[7] A. Abumwais, A. Amirjanov, K. Uyar1 and M. Eleyat, "Dual-port content addressable memory for cache memory applications," *Computer, Material & Continua*, vol. 3, no. 70, pp. 4583–4597, 2021.

[8] K. Korgaonkar, I. Bhati, H. Liu, J. Gaur, S. Manipatruni *et al.,* "Density tradeoffs of non-volatile memory as a replacement for SRAM based last level cache," in *ACM/IEEE 45th Annual Int. Symp. on Computer Architecture (ISCA)*, Los Angeles-California, pp. 315–327, 2018.

[9] D. Wang and J. Li, "Shared cache allocation based on fairness in a chip multiprocessor architecture," in *Int. Conf. on Advanced Hybrid Information Processing*, Harbin, China, pp. 501–504, 2017.

[10] M. Imani, D. Peroni, A. Rahimi and T. S. Rosing, "Resistive CAM acceleration for tunable approximate computing," *IEEE Transactions on Emerging Topics in Computing*, vol. 7, no. 2, pp. 271–280, 2016.

[11] A. Martyshkin, I. Salnikov, D. Pashchenko and D. Trokoz, "Associative co-processor on the basis of programmable logical integrated circuits for special purpose computer systems," in *Global Smart Industry Conf. (GloSIC)*, Chelyabinsk, Russia, pp. 1–5, 2018.

[12] A. Monazzah, H. Farbeh and S. Miremadi, "LER: Least error rate replacement algorithm for emerging stt-ram caches," *IEEE Transactions on Device and Materials Reliability*, vol. 16, no. 2, pp. 220–226, 2016.

[13] W. Jiang, Q. Wang and V. K. Prasanna, "Beyond TCAMs: An SRAM-based parallel multi-pipeline architecture for terabit IP lookup," in *IEEE INFOCOM the 27th Conf. on Computer Communications*, Phoenix, AZ, USA, pp. 1786–1794, 2008.
[14] K. J. Zhou, C. Mu, B. Wen, X. M. Zhang, G. J. Wu *et al.,* "The trend of emerging non-volatile TCAM for parallel search and AI applications," *Chip*, vol. 1, no. 2, pp. 100012, 2022.
[15] S. Mittal and J. S. Vetter, "A survey of software techniques for using non-volatile memories for storage and main memory systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, no. 5, pp. 1537–1550, 2015.
[16] Y. Li, L. Zeng, G. Chen, C. Gu, F. Luo *et al.,* "A multi-hashing index for hybrid DRAM-NVM memory systems," *Journal of Systems Architecture*, vol. 128, pp. 102547, 2022.
[17] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi *et al.,* "The gem5 simulator," *ACM SIGARCH Computer Architecture News*, vol. 39, no. 2, pp. 1–7, 2011.
[18] T. K. Prakash and L. Peng, "Performance characterization of spec cpu2006 benchmarks on intel core 2 duo processor," *ISAST Trans. Computer Software Engineering*, vol. 2, no. 1, pp. 36–41, 2008.