Locality Sensitive Hardware Signature Variants for Hardware Transactional Memory

sangeetha r (sangeethavinashi@gmail.com)
National Institute of Technology Tiruchirappalli https://orcid.org/0000-0001-8734-089X
Satyanarayana Vollala
IIIT–NR: Dr Shyama Prasad Mukherjee International Institute of Information Technology
Ramasubramanian N
National Institute of Technology Tiruchirappalli

Research Article

Keywords: Transactional memory, Hardware signature, Bloom filter, False positive, False negative, Locality sensitive hardware signature

Posted Date: April 26th, 2021

DOI: https://doi.org/10.21203/rs.3.rs-439549/v1

Noname manuscript No.
(will be inserted by the editor)

Locality Sensitive Hardware Signature Variants for Hardware Transactional Memory

Sangeetha R · Satyanarayana Vollala · N. Ramasubramanian

Received: date / Accepted: date

Abstract Lock based techniques have its own limitations like priority inversion, convoying, and deadlock. Lock free techniques overcome those mentioned limitations. Transactional memory (TM) is one leading lock free technique used in recent multi core processors like Intel Haswell and IBM BlueGene/Q. TM has to do data versioning and conflict detection. For conflict detection probabilistic data structure called Bloom Filters are used. Bloom filter based hardware signatures are used in TM. In TM shared memory conflicts like RAW, WAR, and WAW hazards are handled by Bloom Filter (BF). Hardware signatures store memory addresses in hashed form on Bloom filters. Bloom filters are easy to use, performance efficient data structures lead to false positive but never support false negative. Locality sensitive hardware signatures reduce filter occupancy by sharing bits for the contiguous memory addresses, in turn reduces the false positive rate. This paper implements existing H3 – HS and LS – HS proposed by Ricardo Quislant et al. [13]. Also this paper proposes RS – HS, CS – HS, and RO – HS. RO – HS equally spreads addresses among bloom filters thereby reduces filter occupancy. In turn reduced filter occupancy leads to better False Positive Rate.

Keywords Transactional memory · Hardware signature · Bloom filter · False positive · False negative · Locality sensitive hardware signature.

Sangeetha R and N. Ramasubramanian
Department of Computer Science and Engineering, National Institute of Technology, Tiruchirappalli - 620015, India
Tel.: +91-9751662321
E-mail: sangeethavinashi@gmail.com, nrs@nitt.edu

Satyanarayana Vollala
Department of Computer Science and Engineering, IIIT, Naya Raipur, India
Tel.: +91-8220172181
E-mail: satya4nitt@gmail.com, satya@iiitnr.edu.in

1 Introduction

Accesses among shared resources are controlled by locks or lock based techniques. Critical section handling is done by lock based techniques. But it limits the performance by disadvantages like priority inversion, convoying, and deadlock [1], [15]. Lock free techniques like Transactional memory overcome the limitations of lock based techniques. Transactional memory is a parallel programming paradigm supports concurrent ex-ecution of threads. It needs to maintain the following properties like transaction: atomic-ity, consistency, and isolation. But it need not maintain durability property like transac-tion since it is transient. It replaces all sizes of locks from coarse - grained locks to fine – grained locks. Transactional memory mainly needs to do two operations. Former one is data ver-sioning and the later one is conflict detection. Data versioning can be done in two ways. First is eager/undo data versioning, works based on logs. Directly changes the memory and does undo operation when needed. Second is lazy/write data versioning, works based on buffers. Maintain the changes in buffer, till the commit operation is performed. Reflects the changes on memory location during commit operation. Later operation is Conflict detection. Handling shared memory conflicts like RAW, WAR, and WAW hazards. There are two kinds of conflict detection in transactional memory. First one is eager/encounter/pessimistic conflict detection. Conflict detection is done during loads/stores. Second one is lazy/commit/optimistic conflict detection. Con-flict detection is done during commit operation [6]. For conflict detection transactions must maintain its readset addresses and writeset addresses. In order to carry on readset and writeset addresses BF is used as a data structure mostly. Probabilistic data structure called bloom filter that stores memory addresses after hashing at the cost of false positives [12]. There are three types of transactional memory available. They are a) software transactional memory, b) hardware transactional memory, and c) hybrid transactional memory. Software TM is implemented through software constructs like atomic and so on. Recent implementations of Software TM are Intel C++ STM, Intel Java STM, HASTM, and Microsoft OSTM and so on. Full implementation of TM is done through in HTM. Hardware transactional memory performs better than software transactional memory. STM is more flexible than HTM by allowing wider variety of algorithms. Latest processor implementations like Intel Haswell and IBMBlue-Gene/Q processors use Hardware transactional memory. Since it is done on hardware transactional memory, it is called as hardware signatures [2]. Hybrid TM implementations are done to eliminate limitations of STM and HTM and to provide a better TM implementation [10], [22],[21]. In case if read and write bits are added to the cache tag to carry on readset and writeset addresses. It is called as cache tag augmentation and it leads to several limita-tions like block replacement, thread switching, and modification of cache structures. In order to maintain read-set and write-set addresses both cache tag augmentation and hardware signatures or anyone of them could be used. Using hardware signatures doesn't need more changes in the existing cache structures. Locality sensitive hardware signatures share signature bits for the contiguous memory locations thereby reduces filter occupancy that leads to low false positive rate. H3 hashing behaves well for memory address sequences [13]. Contributions of this paper are as follows,

- Compared the newly proposed hardware signatures with H3-HS (Hardware signature using H3 hashing) and LS-HS (Hardware signature using H3 hashing with nullification or classical locality sensitive hardware signature).
- Proposes new hardware signatures
  - Hardware signature using H3 hashing with reverse nullification : RS-HS
  - Combined version of LS-HS and RS-HS : CS-HS
  - Rotational version of LS-HS : RO-HS
- Implemented the H3-HS, LS-HS, RS-HS, CS-HS and RO-HS using Verilog hardware description language.
- Micro benchmarks of STAMP benchmark has been used in this work for comparison.
- Parameters used for comparison in this work are Filter Occupancy Rate and False Positive Rate (FPR).

RO-HS performs better than the other four signatures. Because of its rotational nature towards spreading the load among bloom filters. RS-HS works similar as LS-HS in the reverse order. CS-HS combines both LS-HS and RS-HS by spreading addresses that fall in first - half of address range using LS-HS and addresses that fall in second - half of address range using RS-HS. CS-HS avoids overloading a specific bloom filter and thereby reduces filter occupancy in turn improves False Positive Rate. H3-HS performs better than LS-HS, RS-HS, and CS-HS because of its working towards address sequenc-es. Remaining of the paper is arranged in a following manner. Section II elaborates related work about hardware signatures. Section III gives a brief idea about preliminary terms and existing hardware signatures H3-HS and LS-HS. Section IV discusses about newly proposed hardware signature variants RS-HS, CS-HS, and RO-HS. Section V gives an overview of implementation and analyses the results obtained. Section VI gives conclusion about the work done in this paper.

2 Related Works

Bloom filters (BFs) based Hardware signatures are being used widely in different fields like image processing, transactional memory, networking, databases, intrusion de-tection and so on for its compressive nature and speed. Size of the bloom filter doesn't depend on the number of items stored in it. Many systems based on multi core processors use bloom filter based hardware sig-natures like network processors, parallel debugging, deterministic replay[10], Au-to-Memoization Processors, thread level speculation, Transactional memory, and so on. This section discusses about different bloom filter architectures and hardware sig-natures proposed by people. Bloom filter architectures and hardware signatures differ based on the type of hash function used, no. of hash functions used, and how the hashed values are handled. True/standard bloom filters support only insertion of an element and membership checking, it shares the existing bits amid hash functions. Parallel BFs are same as true bloom filters but chops and shares the existing bits amid hash functions applied [19]. Parallel BFs perform well than true BFs by reducing False Positive Rate (FPR). Counting bloom filters perform better than standard bloom filters by reducing filter

occupancy and FPR. But counting bloom filters increase the memory space required based on the size of the counters. If the counter size is 4 bits then the counting BF size is fourfolds the size of standard BF. D left counting BF is similar to counting BF and based on D left hashing technique [5]. Parallel multi-set bloom filters maintain one single parallel bloom filter for readset plus writeset addresses. Parallel multi-set shared bloom filters maintain one single parallel bloom filter for readset plus writeset addresses and treats both the readset plus writeset addresses as read-write address [14]. Parallel BF with BF indexing (PBF BF) uses BF to store inherent dependency values of an item, parallel BF by hash table indexing (PBF HT) utilizes hash table to accu-mulate inherent dependency values of an entity. PBF BF and PBF HT reduce FPR. PBF HT performs better than PBF BF by reducing FPR [20], [24]. Hierarchical bloom filters are used for substring matching, when first substring matches with the first level bloom filter second substring matching is proceeded with the second level bloom filter and it goes on. This type of bloom filter is suitable for diction-ary based applications [5]. In weighted bloom filters, more weighted elements uses more number of hash functions to reduce FPR.Pipelined bloom filter applies several stages of hash functions. Con-firmation with the first stage proceeds with the second and so on [20]. Hamming metric locality - sensitive Bloom filters (HLBF) are proposed to handle updates / changes in hamming distances. Along with HLBF, M – HLBF algorithm to support AMT is also proposed by JiangboQian et al. [12]. Generalized bloom filters are sent as a message in networks. In order to provide se-curity in addition to set hash functions reset hash functions are also introduced at the cost of false negatives [7]. Deletable bloom filter support false negative free deletions at the cost of additional memory [17].Dynamic bloom filters support dynamic data sets and the following operations: insertion, deletion, item check and filter union operations. This type of bloom filters is suitable for applications with dynamic and static data sets [4]. Fast bloom filters are called as Bloom-1. They work faster than other bloom filters by single memory look-up. But they increase FPR slightly [16] .Variable increment counting BF works on the basis of variable increments and improves the performance of counting BFs by reducing FPR [18]. Ternary bloom filters occupy same amount of space like counting bloom filters. These filters minimize the number of bits used for the counters and increase the number of counters to provide low FPR [9]. Two – stage adaptive bloom filters (TSABFs) proposed by Yan Du and Sheng Wang to perform per – flow monitoring in Software Defined Networks. This type of bloom filters keeps the FPR under a threshold value and improves resource utilization, rejection probability [3].Jungwon Lee et al. proposed Circled BF to identify the association of many sets. The following values are being used in circled bloom filter 1, 2, or 3. Value 1 ensures the membership of a set1. Value 2 ensures the membership of a set2. Value 3 represents intersection of sets 1 and 2. Circled bloom filters improve the accuracy of membership querying [8]. C.Y. Tseung et al. proposed a novel Self - learning bloom filter to defend DDOS at-tack. Self-learning bloom filters are used to defend DDOS attack based on features se-lected by ANOVA algorithm [23]. Several signatures discussed in this section are LS-Sig, FlexSig, and Unified signa-tures. Locality sensitive signatures (LS-Sig) works based on spatial locality and shares the signature bits of the nearby locations to reduce the filter occupancy. Reducing filter occupancy automatically improves performance and reduces FPR [13]. Flexible signatures (FlexSig) do

allocate, deallocate, insert, and check operations. Allocate – allocates the resources needed for the signature, deallocate – deallocates the resources needed for the signature, insert – inserts the address, check – checks the avail-ability of the address. It makes flexible signatures on demand [11]. Unified signatures merge read signatures and write signatures. It generates read-read dependency. Helper signatures are used to resolve read-read dependency [2]. This paper proposes three new hardware signatures called RS-HS, CS-HS, and RO-HS. Implementation of H3-HS, LS-HS along with the three new hardware signatures proposed are done in this paper. RO-HS performs better than the other four signatures. Because of its rotational na-ture towards spreading the load among bloom filters. RS-HS works similar as LS-HS in the reverse order. CS-HS combines both LS-HS and RS-HS by spreading addresses that fall in first - half of address range using LS-HS and addresses that fall in second - half of address range using RS-HS. CS-HS avoids overloading a specific bloom filter and thereby reduces filter occupancy in turn improves False Positive Rate. H3-HS performs better than LS-HS, RS-HS, and CS-HS because of its working towards address sequences.

3 Preliminaries

3.1 False Positive:

Bloom Filter (BF) says positive for the false address. That is the address checked for matching in the BF, is not previously inserted in the BF but it says positive for the matching.

3.2 False Positive Rate (FPR):

FPR = No. of False positives ÷ Total no. of addresses False Negative: Saying, negative for the address, that is inserted priory. BF never allows false negatives.

3.3 Filter Occupancy:

Total no. of one's present in the BF.

3.4 Filter Occupancy Rate:

Filter occupancy rate = Total no. of one's existing in the BF ÷ Size of a BF

3.5 H3-HS:

H3-HS uses H3 hashing for its implementation. H3 hashing does binary multiplication followed by XOR operation. In order to hash n bit address, it is multiplied with

nxm random binary matrix.

$$h(a) = \begin{bmatrix} a_3 & a_2 & a_1 & a_0 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 1 & 1 \\ 0 & 1 \\ 1 & 0 \end{bmatrix} \tag{1}$$

This paper implements 4 parallel bloom filters for manipulating a 64 bit address. To implement k parallel bloom filters k number of random h3 matrices are used. Bit indices of bloom filter are set by the value of h(a) [13].

3.6 LS-HS:

LS-HS uses H3 hashing with nullification procedure. If the four matrices used are hs0, hs1, hs2, hs3. As per nullification ha0 matrix is kept as such and named as hs0', last one row of hs1 matrix is nullified and named as hs1', last 2 rows of hs2 matrix are nulli-fied and named as hs2', last 3 rows of ha3 matrix are nullified and named as hs3'. Then like H3-HS, binary multiplication followed by XOR operation is done. hs0', hs1', hs2', hs3' are mxn matrices. M0 is amxm matrix. M1 is nxn last row nullified M0 matrix. M2 is nxn last 2 rows nullified M0 matrix. M3 is nxn last 3 rows nullified M0 matrix [13].

$$hs0' = M0.hs0$$

$$hs1' = M1.hs1$$

$$hs2' = M2.hs2$$

$$hs3' = M3.hs3$$

Preprocessed matrices hs0', hs1', hs2', hs3' are multiplied with address A in order to generate the bit indices.

$$hs(a)0' = A.hs0'$$

$$hs(a)1' = A.hs1'$$

$$hs(a)2' = A.hs2'$$

$$hs(a)3' = A.hs3'$$

As per LS-HS concurrent addresses will share most of the bit indices that leads to less filter occupancy and reduced FPR. It is depicted in the Table 1. In the Table 1, 4 – bit addresses are used for manipulation. hs0, hs1, hs2, hs3, hs0', hs1', hs2', hs3' are 4x2 matrices. M0, M1, M2, M3 are 4x4 matrices. Resultant bit indices are 1x2.

4 Proposed Hardware Signature Variants

The following hardware signatures are implemented in this paper: H3-HS, LS-HS, RS-HS, CS-HS, and RO-HS. Among those RS-HS, CS-HS, and RO-HS are the newly proposed hardware signatures. RS-HS uses H3 hashing with reverse nullification procedure. As per reverse nullifi-cation ha3 matrix is kept as such, last one row of ha2 matrix is nullified, last 2 rows of ha1 matrix are nullified, last 3 rows of ha0 matrix are nullified. Table 1 shows the ma-nipulations for sampleaddresses & bit sharing of H3-HS, LS-HS, and RS-HS. In LS-HS, bit indices of Bfr 4 will be reused / shared many times and Bfr 1 will be overloaded or filled more. In RS-HS, bit indices of Bfr 1 will be reused / shared many times and Bfr 4 will be overloaded or filled more. RS-HS is depicted in the Figure 1.

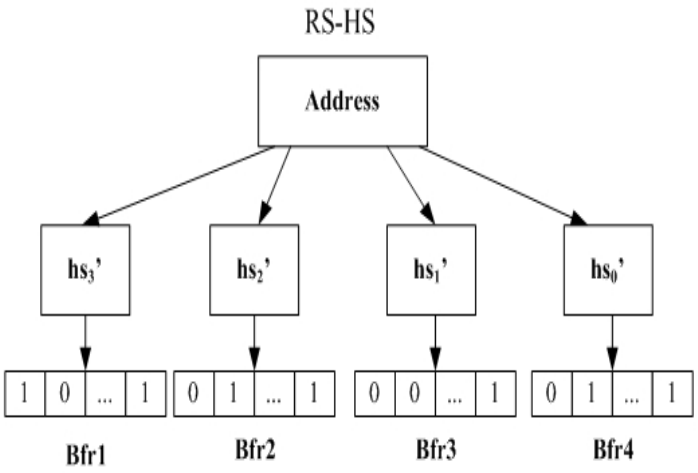

Fig. 1 RS-HS

CS-HS combines both LS-HS and RS-HS. For the first 50% of the address range LS-HS is used and for the next 50% of the address range RS-HS is used. For example if the address range of RS-HS is 2n, then 0 – 2n/2-1 addresses will make use of LS-HS and 2n/2 to 2n-1addresses will make use of RS-HS. CS-HS is depicted in the Figure 2. RO-HS, uses hsx' matrices in a rotational basis. For the first 25% of the address range matrix order used is hs0', hs1', hs2', hs3'. For the second 25% of the address range matrix order used is hs1, hs2, hs3,hs0. For the third 25% of the address range matrix order used ishs2, hs3,hs0, hs1.For the fourth 25% of the address range matrix order used is hs3,hs0,hs1, hs2. RO-HS distributes the load evenly on all the four bloom filters when compared with the other hardware signatures implemented. RO-HS is depicted in the Figure 3.

Table 1 Bit Sharing

Addresses	H3 Hashing				With Nullification LS-HS				With reverse Nullification RS - HS			
	hs0	hs1	hs2	hs3	hs0'	hs1'	hs2'	hs3'	hs0'	hs1'	hs2'	hs3'
0	0	0	0	0	0	0	0	0	0	0	0	0
1	2	3	3	1	2	0	0	0	0	0	0	1
10	1	2	2	2	1	2	0	0	0	0	0	2
11	3	1	1	3	3	2	0	0	0	0	0	3
100	3	1	3	0	3	1	3	0	0	1	3	0
101	1	2	0	1	1	1	3	0	0	1	3	1
110	0	3	1	2	2	3	3	0	0	1	1	2
111	0	0	2	3	0	3	3	0	0	1	1	3
1000	2	0	1	3	2	0	1	3	2	0	1	3
1001	0	3	2	2	0	0	1	3	2	0	1	2
1010	3	2	3	1	3	2	1	3	2	0	3	1
1011	1	1	0	0	1	2	1	3	2	0	3	0
1100	1	1	2	3	1	1	2	3	2	1	2	3
1101	3	2	1	2	3	1	2	3	2	1	2	2
1110	0	3	0	1	0	3	2	3	2	1	0	1
1111	2	0	3	0	2	3	2	3	2	1	0	0

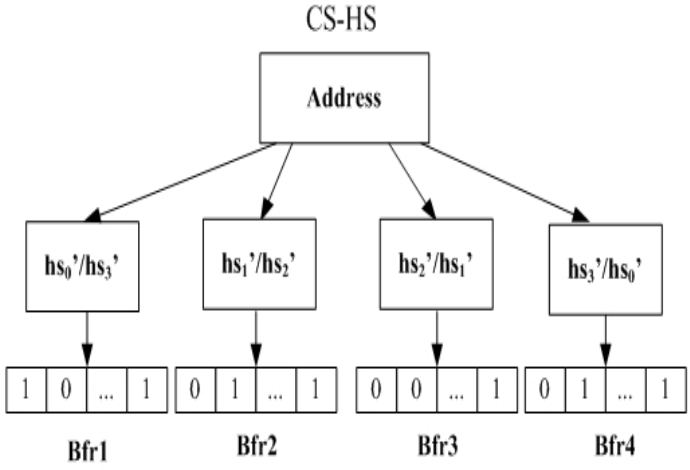

Fig. 2 CS-HS

5 Implementation and Results

Addresses are taken by tracing the execution of micro benchmarks of STAMP. Tracing is done using PIN 3.2 tool. Implementation is done using Verilog HDL in Xilinx ISE design suite 14.7. Results are taken for Bayes, Genome, Intruder, Kmeans, Labyrinth, Ssca2, and Va-cation micro benchmarks of STAMP. By using PIN tool

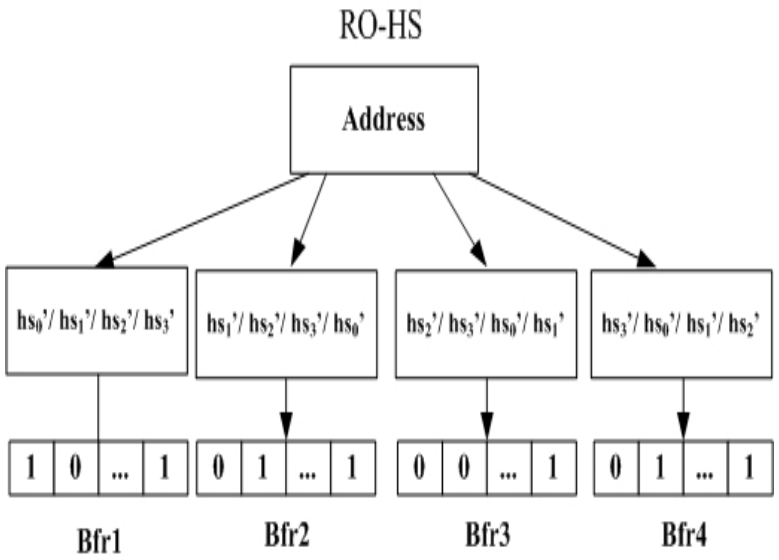

Fig. 3 RO-HS

memory reference addresses are fetched and used. False positive rate (FPR) and filter occupancy of all four bloom filters are calculated based on results for the micro benchmarks of STAMP. Filter occupancy and FPR are directly related. More filter occupancy (FO) leads to high FPR and less filter occupancy leads to low FPR. FPR results of micro benchmarks are indicated in Figure 4. As per Figure 4, RO-HS has low FPR than all other hardware signatures implemented. Initially for addresses up to 200, H3-HS perform better than RO-HS after that RO-HS shows better results. LS-HS and RS-HS implementations show more or less similar results for FPR. CS-HS performs better than LS-HS and RS-HS. H3-HS performs better than CS-HS, LS-HS, and RS-HS. As per the FPR results of STAMP micro benchmarks more of bit sharing and less filter occupancy may not perform better than evenly distributed addresses. RO-HS evenly distributes the addresses in bloom filters. So it performs better than all other hardware signature implementations.H3-HS distributes addresses evenly when compared with LS-HS, RS-HS, and CS-HS. So FPR of H3-HS is less than LS-HS, RS-HS, and CS-HS. Filter occupancy rate of Genome, Ssca2, and Vacation micro benchmarks are listed in Table 2. Filter occupancy rate of other micro benchmarks are similar to the results shown in Table 2 with minor variations. Filter occupancy of Bloom filter 1 H3-HS and LS-HS occupy more. CS-HS occupies less than H3-HS and LS-HS. RO-HS occupies less than CS-HS. RS-HS occupies less than CS-HS. Filter occupancy of Bloom filter 2 H3-HS occupies more filter than the other hardware signatures. RO-HS occupies less than H3-HS.CS-HS, RS-HS, and LS-HS occupy less than RO-HS. Filter occupancy of Bloom filter 3 H3-HS occupy more. RO-HS occupy less than H3-HS. LS-HS, RS-HS, and CS-HS occupy less than RO-HS Filter occupancy of Bloom filter 4 H3-HS and RS-HS occupy more. CS-HS occupies less than H3-HS and RS-HS. RO-HS occupies less than CS-HS. LS-HS occupies less than RO-

HS.H3-HS occupies more space in all the four filters. LS-HS occupies more space of Bloom filter 1. RS-HS occupies more space of Bloom filter4. RO-HS occupies all the four filters moderately.

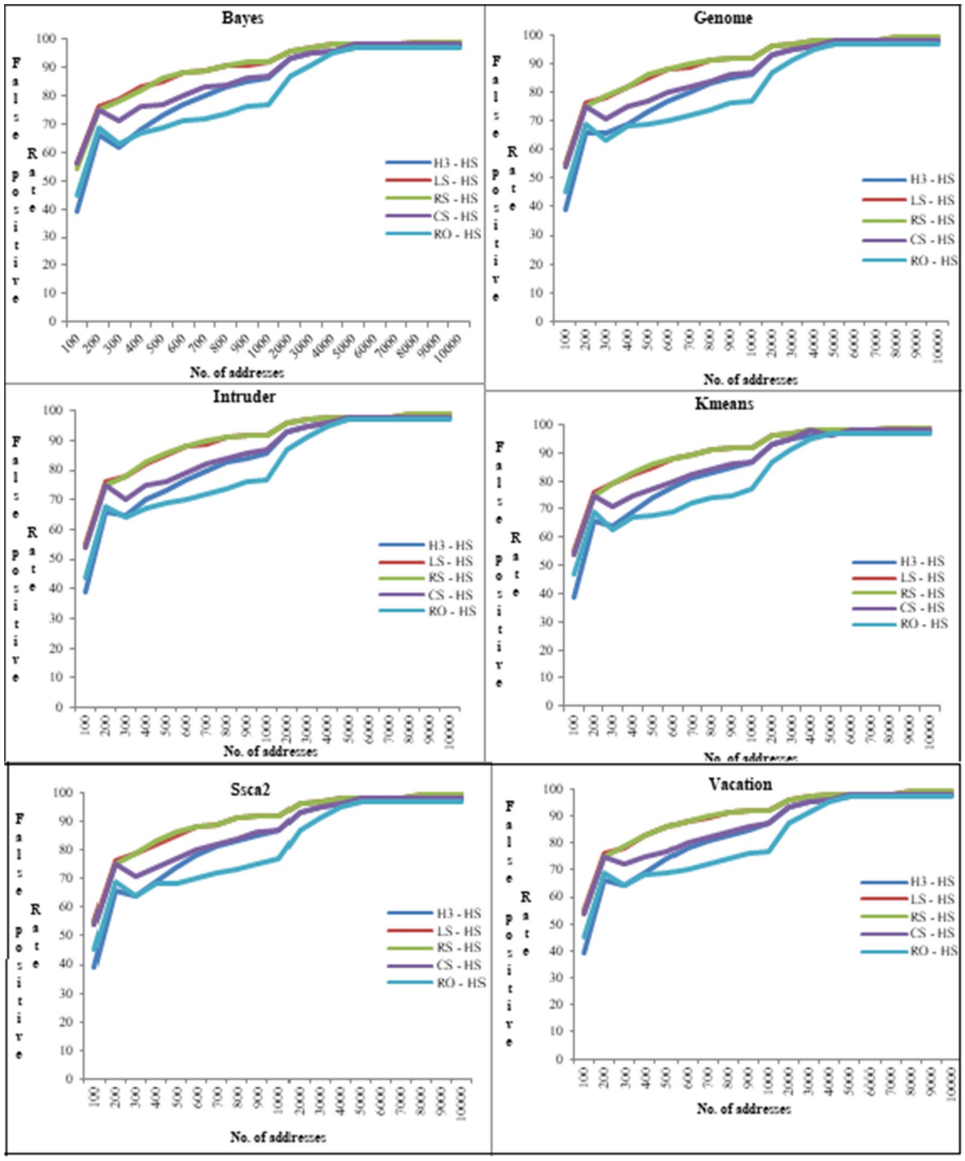

Fig. 4 FPR results

Table 2 Filter Occupancy of micro-benchmarks

Hardware Signature Type	No. of addresses	Bfr 1	Bfr 2	Bfr 3	Bfr 4	Bfr 1	Bfr 2	Bfr 3	Bfr 4	Bfr 1	Bfr 2	Bfr 3	Bfr 4
		Genome micro benchmark				Ssca2 micro benchmark				Vacation micro benchmark			
H3 – HS	100	54	61	40	60	57	61	38	61	56	61	38	57
	400	91	80	85	82	81	78	85	86	77	87	75	75
	700	93	90	95	90	92	87	92	94	96	92	94	93
	1000	96	100	100	96	97	92	98	97	99	95	97	97
	4000	100	100	100	100	100	100	100	100	100	100	97	98
	7000	100	100	100	100	100	100	100	100	100	100	97	98
	10000	100	100	100	100	100	100	100	100	100	100	97	98
LS - HS	100	55	3	3	4	58	2	3	3	57	1	3	2
	400	90	5	3	4	82	3	4	3	76	1	3	2
	700	92	5	3	4	92	4	4	3	96	2	3	2
	1000	98	6	3	4	98	4	4	3	100	2	3	2
	4000	100	6	3	5	100	5	5	3	100	2	3	2
	7000	100	6	3	5	100	7	5	3	100	2	3	2
	10000	100	6	3	5	100	7	5	3	100	2	3	2
RS - HS	100	0	0	2	61	3	2	2	62	3	3	2	60
	400	0	0	5	84	4	3	5	88	3	4	4	80
	700	0	0	5	92	4	3	5	93	3	4	5	97
	1000	0	0	5	98	4	3	6	98	3	4	5	98
	4000	0	0	5	100	5	3	8	100	3	4	7	98
	7000	0	0	5	100	6	3	8	100	3	4	7	99
	10000	0	0	5	100	6	3	8	100	3	4	7	99
CS – HS	100	25	3	4	48	26	2	3	47	25	2	2	50
	400	60	4	5	68	61	2	4	62	62	2	7	60
	700	85	5	6	80	90	8	5	91	88	7	8	92
	1000	95	7	7	98	97	8	7	98	97	7	8	98
	4000	100	7	7	100	100	9	7	100	100	7	9	98
	7000	100	7	8	100	100	9	8	100	100	7	9	98
	10000	100	7	8	100	100	9	9	100	100	7	9	98
RO – HS	100	23	12	30	33	25	12	30	33	23	11	27	32
	400	60	23	40	48	62	22	40	60	50	22	40	58
	700	65	40	68	63	65	70	62	79	68	60	80	82
	1000	95	85	92	91	80	80	91	94	70	94	93	92
	4000	97	95	95	93	98	100	98	96	98	98	97	93
	7000	100	100	100	100	100	100	100	100	98	100	97	97
	10000	100	100	100	100	100	100	100	100	98	100	98	97

6 Conclusion

RO-HS shows better performance by its even distribution of load among all the four bloom filters. Even though the locality sensitive nature of the signature reduces the filter occupancy, more of bit sharing leads to high FPR. Only insertions into the bloom filter increases the filter occupancy and never reduces it. Allowing deletions also into the bloom filter will reduce filter occupancy. Our future work is to implement counting bloom filter version of H3-HS, LS-HS, RS-HS, CS-HS, and RO-HS to reduce the FPR further.

7 Declarations

7.1 Funding:

Not applicable

7.2 Conflicts of interest/Competing interests:

Not applicable

7.3 Availability of data and material:

Not applicable

7.4 Code availability:

Not applicable

7.5 Authors' contributions:

Not applicable

References

1. Cho, H., Ravindran, B., Jensen, E.D.: Lock-free synchronization for dynamic embedded real-time systems. In: Proceedings of the Design Automation & Test in Europe Conference, vol. 1, pp. 1–6. IEEE (2006)
2. Choi, W., Draper, J.: Improving utilization of hardware signatures in transactional memory. IEEE Transactions on Parallel and Distributed Systems **24**(11), 2230–2239 (2012)
3. Du, Y., Wang, S.: Two-stage adaptive bloom filters for per-flow monitoring in software defined networks. In: 2018 IEEE International Conference on Communications (ICC), pp. 1–7. IEEE (2018)
4. Guo, D., Wu, J., Chen, H., Yuan, Y., Luo, X.: The dynamic bloom filters. IEEE Transactions on Knowledge and Data Engineering **22**(1), 120–133 (2009)
5. Gupta, D., Batra, S.: A short survey on bloom filter and its variants. In: 2017 International Conference on Computing, Communication and Automation (ICCCA), pp. 1086–1092. IEEE (2017)
6. Kozyrakis, C.: Transactional memory implementation overview. In: 2006 IEEE Hot Chips 18 Symposium (HCS), pp. 1–31. IEEE (2006)
7. Laufer, R.P., Velloso, P.B., Duarte, O.C.M.: A generalized bloom filter to secure distributed network applications. Computer Networks **55**(8), 1804–1819 (2011)
8. Lee, J., Lim, H.: A circled bloom filter for the membership identification of multiple sets. In: 2019 International Conference on Electronics, Information, and Communication (ICEIC), pp. 1–3. IEEE (2019)
9. Lim, H., Lee, J., Byun, H., Yim, C.: Ternary bloom filter replacing counting bloom filter. IEEE Communications Letters **21**(2), 278–281 (2016)
10. Meenu, S.: Transactional memory: A review
11. Orosa, L., Antelo, E., Bruguera, J.D.: Flexsig: Implementing flexible hardware signatures. ACM Transactions on Architecture and Code Optimization (TACO) **8**(4), 1–20 (2012)

12. Qian, J., Huang, Z., Zhu, Q., Chen, H.: Hamming metric multi-granularity locality-sensitive bloom filter. IEEE/ACM Transactions on Networking **26**(4), 1660–1673 (2018)
13. Quislant, R., Gutierrez, E., Plata, O., Zapata, E.L.: Ls-sig: Locality-sensitive signatures for transactional memory. IEEE Transactions on Computers **62**(2), 322–335 (2011)
14. Quislant, R., Gutierrez, E., Plata, O., Zapata, E.L.: Multiset signatures for transactional memory. In: Proceedings of the international conference on Supercomputing, pp. 43–52 (2011)
15. Rajwar, R.: Speculation-based techniques for transactional lock-free execution of lock-based programs. Ph.D. thesis, Citeseer (2002)
16. Reviriego, P., Christensen, K., Maestro, J.A.: A comment on "fast bloom filters and their generalization". IEEE Transactions on Parallel and Distributed Systems **27**(1), 303–304 (2015)
17. Rothenberg, C.E., Macapuna, C.A., Verdi, F.L., Magalhaes, M.F.: The deletable bloom filter: a new member of the bloom family. IEEE Communications Letters **14**(6), 557–559 (2010)
18. Rottenstreich, O., Kanizo, Y., Keslassy, I.: The variable-increment counting bloom filter. IEEE/ACM Transactions on Networking **22**(4), 1092–1105 (2013)
19. Sangeetha, R., Ramasubramanian, N.: A survey of hardware signature implementations in multi-core systems. In: 2015 3rd International Conference on Signal Processing, Communication and Networking (ICSCN), pp. 1–5. IEEE (2015)
20. Saravanan, K., Senthilkumar, A., Chacko, P.: Modified whirlpool hash based bloom filter for networking and security applications. In: 2014 2nd International Conference on Devices, Circuits and Systems (ICDCS), pp. 1–6. IEEE (2014)
21. Shahid, A., Murad, M., Qadri, M.Y., Qadri, N.N., Ahmed, J.: Hardware transactional memories: A survey. In: Innovative Research and Applications in Next-Generation High Performance Computing, pp. 47–65. IGI Global (2016)
22. Titos-Gil, R., Fernández-Pascual, R., Ros, A., Acacio, M.E.: Concurrent irrevocability in best-effort hardware transactional memory. IEEE Transactions on Parallel and Distributed Systems **31**(6), 1301–1315 (2019)
23. Tseung, C., Chow, K., Zhang, X.: Anti-ddos technique using self-learning bloom filter. In: 2017 IEEE International Conference on Intelligence and Security Informatics (ISI), pp. 204–204. IEEE (2017)
24. Xiao, B., Hua, Y.: Using parallel bloom filters for multiattribute representation on network services. IEEE Transactions on parallel and distributed systems **21**(1), 20–32 (2009)

Figures

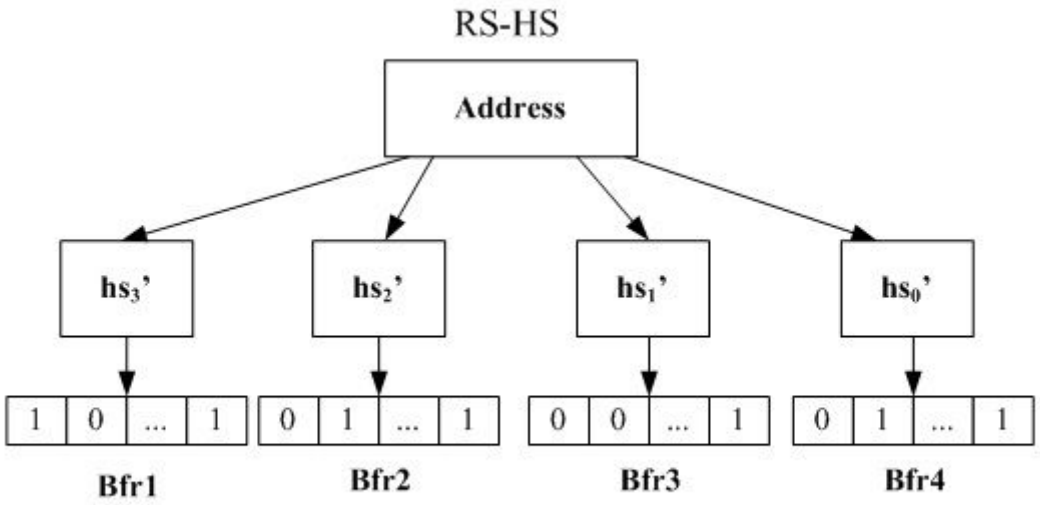

Figure 1

RS-HS

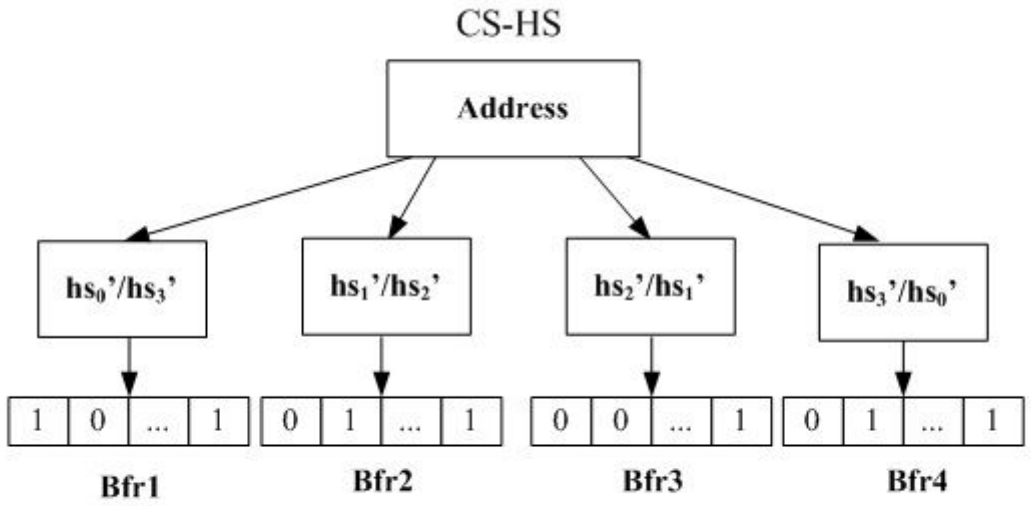

Figure 2

CS-HS

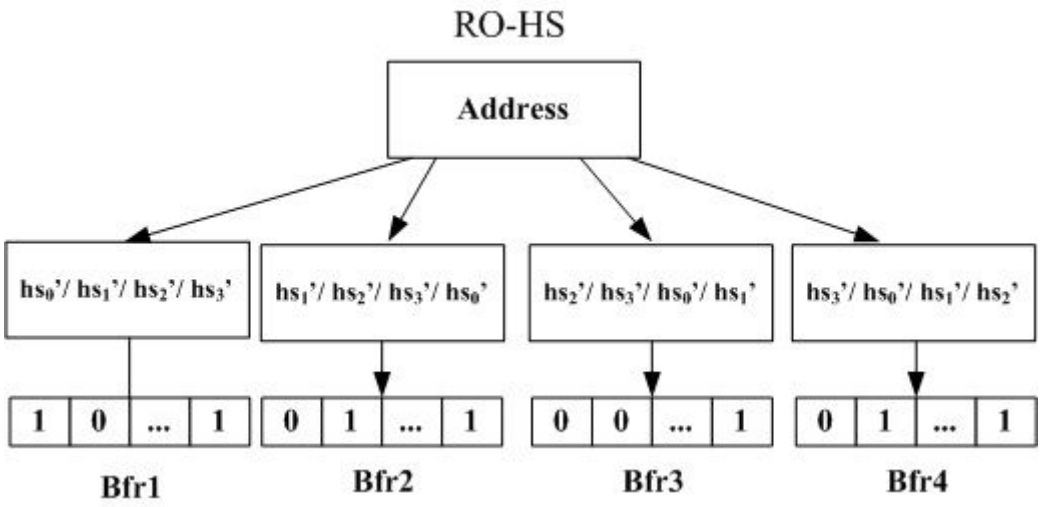

Figure 3

RO-HS

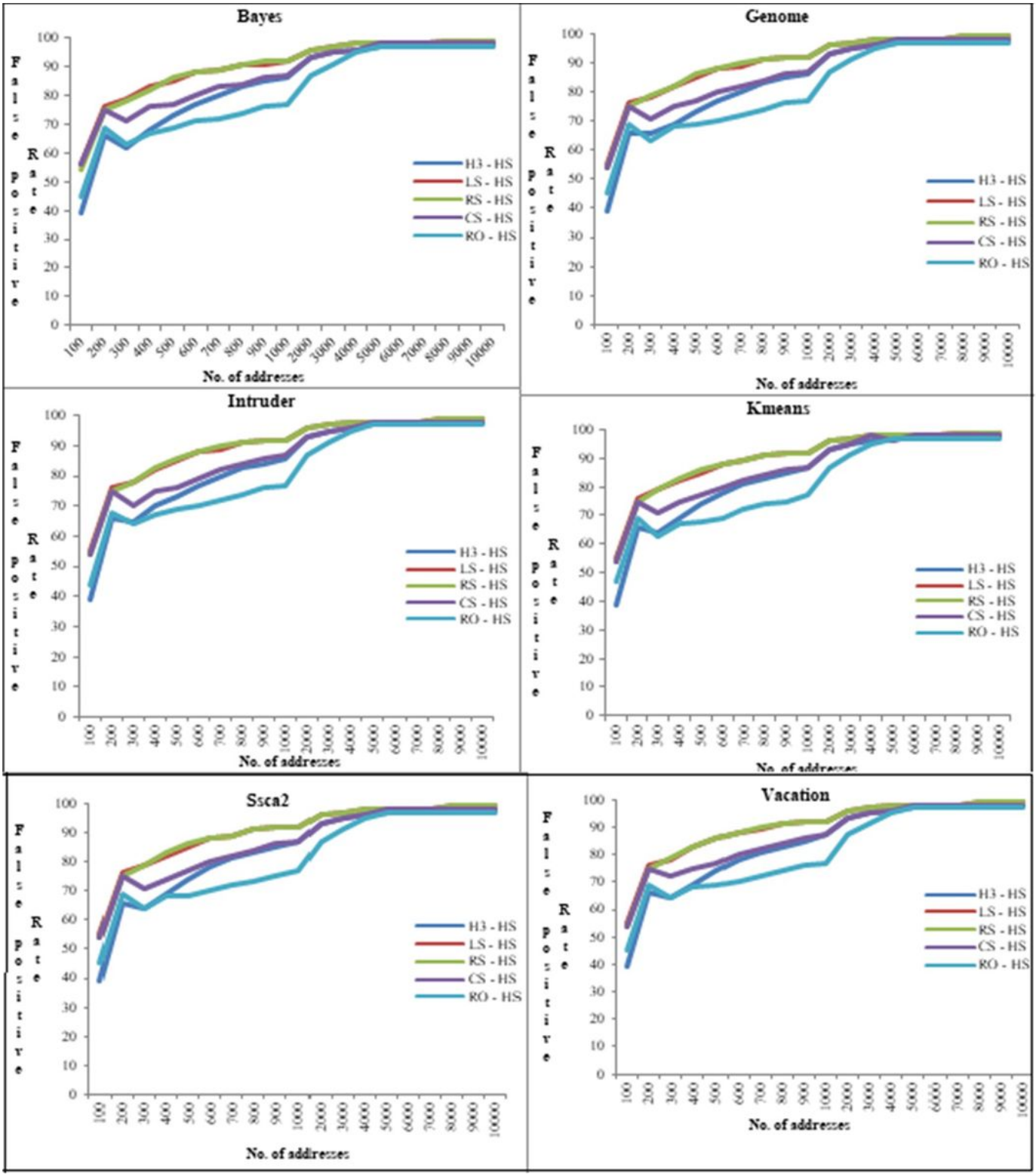

Figure 4

FPR results