Unified Locality-sensitive Signatures for Transactional Memory

R. Quislant, E. Gutierrez, O. Plata and E.L. Zapata

Department of Computer Architecture, University of Málaga,
ETSI Informática, Campus Teatinos, Málaga, E 29071, Spain
{quislant,eladio,oplata,zapata}@uma.es

Abstract. Transactional memory systems coordinate the execution of concurrent transactions by committing non-conflicting ones. Transaction conflicts are detected by recording on-the-fly the memory locations issued by the threads. Some implementations use two per-thread Bloom filters (signatures), one for reads and another for writes, for that purpose. Signatures summarize sets of memory addresses accessed inside a transaction in bounded hardware. However, fixed-sized hardware introduces the address aliasing problem that results in false positives during the conflict checking process.
It is known that the false positive rate increases with the size of the transactions, which has a strong negative impact in the performance of their concurrent execution. In a previous work, authors developed a technique with the aim of reducing the probability of false positives by exploiting spatial locality. In this paper we propose a new technique based on joining the two Bloom filters into a single one and partially sharing the hash function mappings for reads and writes. This unification technique is combined with the locality-sensitive one and it is proved that the false positive rate is further reduced.
This paper proves that unified locality-sensitive signatures improve the execution performance of large concurrent transactions in most tested codes compared to separate signatures, without increasing significantly the required hardware area and with a small increment of power consumption.

Keywords: Hardware transactional memory, memory locality, signatures, Bloom filters.

1 Introduction

Transactional Memory (TM) [8, 7] emerges as an alternative to the conventional multithreaded programming to ease the writing of concurrent programs. TM introduces the concept of transaction, a block of computations which appears to be executed with atomicity and isolation. Transactions replace a pessimistic lock-based model by an optimistic one which solves the abstraction and composition problems.

TM systems coordinate the execution of concurrent transactions by committing non-conflicting ones. A conflict occurs when concurrent transactions access the same memory location and, at least, one of the accesses is a write. Transaction conflicts are detected by recording on-the-fly the memory locations issued by the threads. Some TM implementations use two per-thread Bloom filters [1] (signatures), one for reads and another for writes, for that purpose. Signatures summarize sets of memory addresses accessed inside a transaction in bounded hardware. However, fixed-sized hardware introduces the address aliasing problem (different memory addresses with the same signature representation) that results in false positives during the conflict checking process. Examples of systems that use signatures are BulkSC [4], LogTM-SE [19], SigTM [12], FlexTM [17], and STMlite [11].

It is known that the false positive rate increases with the size of the transactions, and this have a strong negative impact in the performance of their concurrent execution. In a previous work [14], authors developed a technique with the aim of reducing the probability of false positives. This technique defines new hash function mappings so that nearby located addresses share some bits in the Bloom filters, that is, it exploits spatial locality. In this paper we propose a new technique based on joining the two Bloom filters into a single one and partially sharing the hash function mappings for reads and writes without adding significant hardware complexity. The rationale behind this technique is the uneven cardinality that transactional read/write sets exhibit, where read sets are usually larger than write sets. As a result, the signature for reads populates much more than the one for writes and, consequently, the false positive rate for the read signature may be high while, at the same time, the write filter has still a low occupation, with negligible false positive rate. This unification technique is combined with the locality-sensitive one and it is proved that the false positive rate is further reduced.

We use the Wisconsin GEMS LogTM-SE simulator [10] to implement and evaluate the performance of the proposed unified locality-sensitive signatures. Besides, we use CACTI [18] to evaluate the hardware area and power requirements. Experimental results show that the proposed approach is able to reduce the false positive rate and improve the execution performance in most of the benchmark codes, with an insignificant increase in hardware area and a slight increase in power consumption.

The rest of the paper is organized as follows. Next section presents a background on signatures, describing how they are usually designed and implemented, and a brief review of the related work. In Section 3 we introduce and discuss our proposed unified signature design, including their implementation, and a comparison with the separate signature design. Section 4 analyzes the hardware area and power requirements of our signature designs. In section 5 we show an analysis of our proposed signatures and we determine the false positive rate in different cases. Section 6 presents the implementation of unified signatures on the GEMS simulator, and discusses how our novel signature design may improve the execution performance. Finally, Section 7 concludes the paper.

2 Background and Related Work

In the context of TM, each concurrent thread uses its signatures to record all the memory locations issued when executing inside a transaction. These locations are sorted out into a read set (RS) and a write set (WS). Thus, each thread needs a pair of private signatures. As they are used for conflict detection amongst concurrent transactions, signatures do not tolerate false negatives (undetected true conflicts) but may assume a limited amount of false positives (false conflicts). On the other hand, the RS and WS sizes are unknown in advance, therefore, signatures should not limit the number of addresses to be tracked. In addition, test and insertion of an address should be fast operations.

Fulfilling the requirements above, Ceze et al. [5] proposed a signature implementation with per-thread Bloom filters. These filters were devised to test whether an element is a member of a set in a time and space-efficient way. The Bloom filter comprises a bit array and k different hash functions that map elements into k randomly distributed bits of the array. At first, all the array bits are set to 0. Inserting an element into the Bloom filter consists in setting to 1 the k bits given by the hash functions. Test for membership consists in checking that those k bits are asserted.

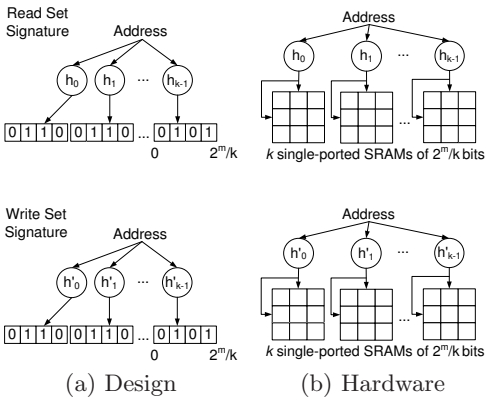
Bloom filters are also known as true or regular Bloom filters. Sanchez et al. [16] proposed the parallel Bloom filter as a hardware-efficient implementation of regular Bloom filters. Whereas the regular filter is implemented as a k-ported SRAM, the parallel one consists of k 1-ported SRAMs, yielding the same or better false positives rate. The same work concludes that Bloom filters should include the H3 class of hash functions [3], instead of bit-selection ones [15], since they are closer to random distribution. However, H3 is more hardware expensive than bit-selection as it needs an XOR tree per hash bit.

An alternative hardware-efficient implementation of hash functions, Page-Block-XOR hashing (PBX), has been proposed in [20]. They use the concept of entropy to find the input bits to the hash functions with high randomness, allowing to reduce the hardware complexity of those functions. Notary also proposes a technique to reduce the number of asserted bits in the signature, based on segregating addresses into private and shared sets. Then, only the shared addresses are recorded in the signature. This solution requires support at the compiler, runtime/library and operating system levels. In addition, the programmer must define which objects are private or shared.

Recently, Choi et al. [6] proposed adaptive grain signatures, that keep the history of transaction aborts and dynamically changes the input bit range to the hash functions based on the abort history. The aim of this design is to reduce the number of false positives that harm the execution performance.

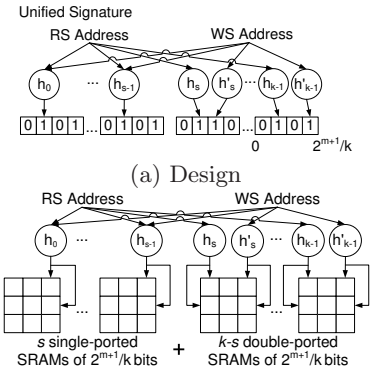
3 Unified Signature Design

Parallel Bloom filters have been proved to yield similar or better performance than regular ones and they require less hardware [16] [14]. Consequently, regular implementation will not be taken into account in this paper.



(a) Design (b) Hardware

Fig. 1: Parallel Separate Signatures.



(a) Design

(b) Hardware

Fig. 2: Parallel Unified Signatures.

Parallel Bloom filters comprise k arrays of $2^m/k$ bits, each of which is only indexed by its own hash function. Figure 1 shows the design and implementation of parallel Bloom signatures. They consist of two separate parallel filters to record the read set and write set addresses. Parallel filters can be implemented as single-ported SRAMs, thus saving in hardware area with respect to regular filters which are implemented as multi-ported SRAMs.

The unified counterpart for the parallel separate signature is depicted in Figure 2. In this case, the bit array is also partitioned into k smaller arrays but $(2^{m+1}/k)$-bit length. Each array is indexed by two hash functions, one for the read set, $h_{[0,k-1]}$, and the other one for the write set, $h'_{[0,k-1]}$. Consequently, parallel unified filters need 2-ported SRAMs instead of single-ported ones taking about twice the area of parallel separate filters. To alleviate this problem, s SRAMs can be made single-ported as Figure 2b shows. This way, an address inserted as a read address is also inserted as a write and vice-versa.

The motivation behind unified signatures come from Table 1 which shows the percentage of addresses that have been both read and written inside transactions for each benchmark (a description of the simulation environment can be found in Section 6.1) with respect to the total number of addresses (without repetition). About 50% of locations are both read and written for Bayes, Kmeans and Yada. Overall, about 30% of total locations addressed by each benchmark has been both read and written.

In order to work out the value of s a trade off between hardware requirements and signature performance has to be carried out. On the one hand, if s is set to k, the unified signature implements k single-ported SRAMs. Thus, such a signature requires the same hardware than the parallel separate signature but it is unable to discriminate between read and written addresses and it could degrade the performance. On the other hand, if s is set to 0, the unified signature implements k double-ported SRAMs increasing the hardware requirements

Table 1: Percentage of addresses that have been both read and written inside transactions.

Bench	%	Bench	%
Bayes	51.0	Labyrinth	15.3
Genome	16.0	SSCA2	25.0
Intruder	7.1	Vacation	8.4
Kmeans	48.6	Yada	45.0

Table 2: Area (mm^2) and dynamic energy per access (nJ) requirements of parallel separate and parallel unified signatures. 32nm technology. $k = 4$.

	Area		Energy	
Filter size (2^m)	**4Kbit**	**16Kbit**	**4Kbit**	**16Kbit**
Separate	0.0084	0.0292	0.0020	0.0047
Unified $s = 0$	0.0191	0.0640	0.0030	0.0081
Unified $s = 3$	0.0098	0.0331	0.0026	0.0068

but maximizing the probabilities of discrimination between read and written addresses. Section 6.2 explores every possible scenario.

Finally, hash functions are implemented as H3 XOR functions [3] that only comprise a set of XOR gate trees per function. XOR gate trees do not require significant area and, moreover, they can be replaced by a single line of XOR gates by using PBX hashing [20].

4 Hardware Evaluation

Table 2 compares the area required by unified and separate signatures for several filter sizes. "Filter size" row is the size of one set filter, i.e. 4Kbit means two filters of 4Kbit (for RS and WS) for separate signatures and one filter of 8Kbit for unified ones. We used CACTI 6.5 [13] to model the SRAMs using the 32nm technology node. Parallel separate signatures comprise eight single-ported SRAMs (4 for the RS and 4 for the WS) as $k = 4$, while parallel unified $s = 0$ signatures have four double-ported SRAMs. Separate read/write ports are used. Parallel unified $s = 3$ signatures have three single-ported SRAMs and only one double-ported SRAM. Ports are dual-ended which means that two lines are required per bitline.

Table 2 shows that parallel separate signatures yield the best area and energy numbers. Regarding the parallel unified $s = 0$ signature, it is about twice larger than the parallel separate signature due to its double-ported SRAMs. The parallel unified $s = 3$ configuration, is the closest to the parallel separate one in terms of area. It is only a 13% larger because of the double-ported SRAM. However, parallel unified $s = 3$ signatures outperforms parallel separate ones as seen in Section 6.3. Regarding energy, Table 2 shows a 30% increment in dynamic energy consumption for parallel unified $s = 3$ signatures.

Concerning the hashing logic area, Sanchez et al. [16] worked out one-fifth of the SRAM area for 4 XOR hash functions. This area can be halved using PBX hashing [20] without impact in the performance.

5 False Positive Analysis

Let A be a sequence of addresses, to be inserted in a single Bloom filter of 2^m bits with k hash functions, whose cardinality is $n = Card(A)$. The false positive

probability is commonly calculated [16] [14] as:

$$p_{\text{FP}}(m,k,n) = \left(1 - \left(1 - \tfrac{1}{2^m}\right)^{nk}\right)^k. \quad (1)$$

Eq. (1) can be adapted to the locality-sensitive signature scheme of [14] by considering two supplementary parameters: f which is the probability of an address to be *local*, that is, near to another one in the sequence, and b which measures the average number of bits asserted by a *local* reference with respect to its closest neighbor in the sequence. The value of f will depend on the spatial locality of the program. The value of b can be estimated as $b = \frac{1}{2}+2\cdot\frac{1}{4}+3\cdot\frac{1}{8}+4\cdot\frac{1}{8}$ for the locality-sensitive signatures defined in [14] with $k = 4$ hash functions. For such signatures the false positive probability is given now by:

$$p_{\text{FP LOC}}(m,k,n,f) = \left(1 - \left(1 - \tfrac{1}{2^m}\right)^{n(1-f)k+nfb}\right)^k. \quad (2)$$

First, consider separate filters, where the read and write sets are stored separately, in order to compare their false positive rates to those of the unified filter. Let us define $p_R = \frac{Card(R-R\cap W)}{Card(R\cup W)}$ and $p_W = \frac{Card(W-R\cap W)}{Card(R\cup W)}$ as the probability of an address of the sequence being only read or written, respectively, using the cardinality function of the read (R) and write (W) sets. Consequently, $n = Card(R\cup W)$. Also an address in the sequence can be both read and written with probability $p_{RW} = \frac{Card(R\cap W)}{Card(R\cup W)}$. Therefore, the false positive probability in each filter, assuming locality-sensitive signatures, can be expressed as:

$$p^{read}_{\text{FP LOC}} = \left(1 - \left(1 - \tfrac{1}{2^m}\right)^{n(p_R+p_{RW})\bar{k}}\right)^k, p^{write}_{\text{FP LOC}} = \left(1 - \left(1 - \tfrac{1}{2^m}\right)^{n(p_W+p_{RW})\bar{k}}\right)^k, \quad (3)$$

where $\bar{k} = (1-f)k + fb \leq k$ is the average number of hash insertions in the locality-sensitive scheme.

The effective false positive rate will finally depend on how many checks take place on each separate filter. This way, a mathematical expectation of the false positive rate for the separated locality-sensitive signatures can be expressed as:

$$E[p^{\text{SEPARATE}}_{\text{FP LOC}}(m,k,n,f)] = c_R p^{read}_{\text{FP LOC}} + c_W p^{write}_{\text{FP LOC}}. \quad (4)$$

Here c_R and c_W denote the probability of each filter being checked during the sequence of references. This checking pattern is directly linked to the way in which the threads inspect the potential data dependencies. It remains unknown until run-time, being very dependent on the parallelization strategy and the input data. Other important issues having influence on the checking pattern are the coherence protocol and the abort/resume policy of transactions.

Regarding unified filters, Eq. (2) is still valid as long as the k hashing functions used by reads and writes are disjoint. To make a fair comparison, the size of the unified locality-sensitive filter must be the sum of the sizes of the separate filters. Thus, the false positive probability for this unified locality-sensitive filter is given by

$$p^{\text{UNIFIED}}_{\text{FP LOC}}(m,k,n,f) = p_{\text{FP LOC}}(m+1,k,n(1+p_{RW}),f). \quad (5)$$

In Table 3 several scenarios are shown for different values of the parameters defined above. Eqs. (4) and (5) have been evaluated with high and low values for the given parameters: locality (f), only read addresses (p_R), read and written addresses (p_{RW}), and number of checks in the read filter (c_R). Note that $p_R + p_{RW} + p_W = 1$ and $c_R + c_W = 1$. Labels in the table point out the scheme (separate or unified) with the lowest false positive rate according to equations. In the 66% of the explored scenarios the unified scheme beats the separate one. Nevertheless, the scenario which is closer to real workloads is $c_R = 0.5$, i.e. read and write filters are evenly checked, because the TM system assures strong atomicity [8] and data requested to main memory (out of the bounds of TM) must be checked in both filters. Notice that, in this case, the unified scheme yields better false positive rates until the filter gets filled in about $\frac{2}{3}$ of its total capacity. With high locality such a limit shifts to $\frac{3}{4}$ or even disappears.

Table 3: Signature scheme, separate (SEP) or unified (UNI), with the lowest false positive rate according to Eqs. (4) and (5) for several values of the given parameters (Bloom filters with $m = 10$ and $k = 4$).

		$p_R = 0.15$						$p_R = 0.25$						$p_R = 0.5$					
		$p_{RW} = 0.2$			$p_{RW} = 0.5$			$p_{RW} = 0.2$			$p_{RW} = 0.5$			$p_{RW} = 0.2$			$p_{RW} = 0.5$		
f	n \ c_R	0.2	0.5	0.8	0.2	0.5	0.8	0.2	0.5	0.8	0.2	0.5	0.8	0.2	0.5	0.8	0.2	0.5	0.8
0.2	128	UNI	UNI	SEP	UNI	UNI	SEP	UNI	UNI	SEP	UNI	UNI	UNI	SEP	UNI	UNI	SEP	UNI	UNI
	256	UNI	UNI	SEP	UNI	UNI	SEP	UNI	UNI	SEP	UNI	UNI	UNI	SEP	UNI	UNI	SEP	UNI	UNI
	512	UNI	UNI	SEP	UNI	UNI	SEP	UNI	UNI	SEP	UNI	UNI	UNI	SEP	UNI	UNI	SEP	UNI	UNI
	768	UNI	SEP	SEP	UNI	SEP	SEP	UNI	SEP	SEP	UNI	UNI	UNI	SEP	SEP	UNI	SEP	SEP	UNI
	1024	UNI	SEP	SEP	UNI	SEP	SEP	UNI	SEP	SEP	UNI	UNI	UNI	SEP	SEP	UNI	SEP	SEP	UNI
0.8	128	UNI	UNI	SEP	UNI	UNI	SEP	UNI	UNI	SEP	UNI	UNI	UNI	SEP	UNI	UNI	SEP	UNI	UNI
	256	UNI	UNI	SEP	UNI	UNI	SEP	UNI	UNI	SEP	UNI	UNI	UNI	SEP	UNI	UNI	SEP	UNI	UNI
	512	UNI	UNI	SEP	UNI	UNI	SEP	UNI	UNI	SEP	UNI	UNI	UNI	SEP	UNI	UNI	SEP	UNI	UNI
	768	UNI	UNI	SEP	UNI	UNI	SEP	UNI	UNI	SEP	UNI	UNI	UNI	SEP	UNI	UNI	SEP	UNI	UNI
	1024	UNI	UNI	SEP	UNI	SEP	SEP	UNI	UNI	SEP	UNI	UNI	UNI	SEP	UNI	UNI	SEP	SEP	UNI

6 Evaluation

6.1 Methodology

To evaluate the performance of our unified locality-sensitive signatures we used Simics [9] full system execution-driven simulator along with the TM module GEMS [10] from the Wisconsin Multifacet Project. Simics simulates the SPARC architecture and it is able to run an unmodified copy of a Solaris operating system. Solaris 10 was installed on the simulated machine and all workloads run on top of it. GEMS' Ruby module implements the LogTM-SE TM [19] and also includes a detailed timing model for the memory system. Ruby was modified to include the proposed unified signature design described in Section 3.

The base CMP system consists of 16 in-order, single-issue cores with a 32KB split, 4-way associative, 64B block private L1 cache each. L2 cache is unified,

Table 4: Workloads: Input parameters and TM characteristics

Bench	**Input**	**#xact**	**Time in xact**	avg $\|RS\|$	avg $\|WS\|$	max $\|RS\|$	max $\|WS\|$
Bayes	-v32 -r1024 -n2 -p20 -s0 -i2 -e2	523	94%	76.9	40.9	2067	1613
Genome	-g512 -s64 -n8192	30304	86%	12.1	4.2	400	156
Intruder	-a10 -l128 -n128 -s1	12123	96%	19.1	2.5	267	20
Kmeans	-m40 -n40 -t0.05 -i rand-n1024-d1024-c16	1380	6%	99.7	48.5	134	65
Labyrinth	-i rand-x32-y32-z3-n64	158	100%	76.5	62.9	278	257
SSCA2	-s13 -i1.0 -u1.0 -l3 -p3	47295	19%	2.9	1.9	3	2
Vacation	-n4 -q60 -u90 -r16384 -t4096	24722	97%	19.7	3.6	90	30
Yada	-a20 -i 633.2	5384	100%	62.7	38.4	776	510

8MB, 16-bank, 8-way associative, and 64B block size. A packet-switched interconnect with 64B links connects the cores and cache banks. Cache coherence implements the MESI protocol and maintains an on-chip directory which holds a bit vector of sharers. Main memory is 4GB.

Simulation experiments use perfect signatures (no false positives, hardware unimplementable) as the reference. Filter size ranges from 64 bits, which matches the word length in SPARC architecture, to 8K bits length, which matches the performance of perfect signatures for the simulated workloads. All filters use 4 hash functions of the H3 family [3]. Same H3 matrices of Ruby were used.

The benchmarks belong to the Stanford's STAMP suite [2] which is designed for TM research and includes a wide range of applications with emphasis on large read and write sets. STAMP benchmarks have been adapted to GEMS by applying Luke Yen's patches from the University of Wisconsin, Madison. Table 4 summarizes the input parameters and main transactional characteristics of the benchmarks.

6.2 Unified Signature Results

Unified signature motivation and design are described in Section 3. Table 3 shows that the percentage of addresses both read and written inside transactions is substantial, so we conducted the experiments to find out the number of hash functions that can be shared by read and write filters without losing performance. For that purpose, shared functions range from $s = 0$, all SRAMs are double-ported, to $s = 4$, all SRAMs are single-ported which means that every insertion into the read set is also an insertion into the write set and vice-versa.

Figure 3 shows the execution time of unified signatures. The more read set and write set hash functions are shared ($s > 0$) the better results are obtained for all the benchmarks. In fact, the best results are obtained for $s = 4$ in every benchmark except Bayes and Genome, which execution is slowed down about 1.25× with respect to separate filters for 8Kbit signatures. Therefore, unified $s = 3$ signatures should be used instead of $s = 4$ ones, as these benchmarks are not pretty sensitive to read and write discrimination but other might be.

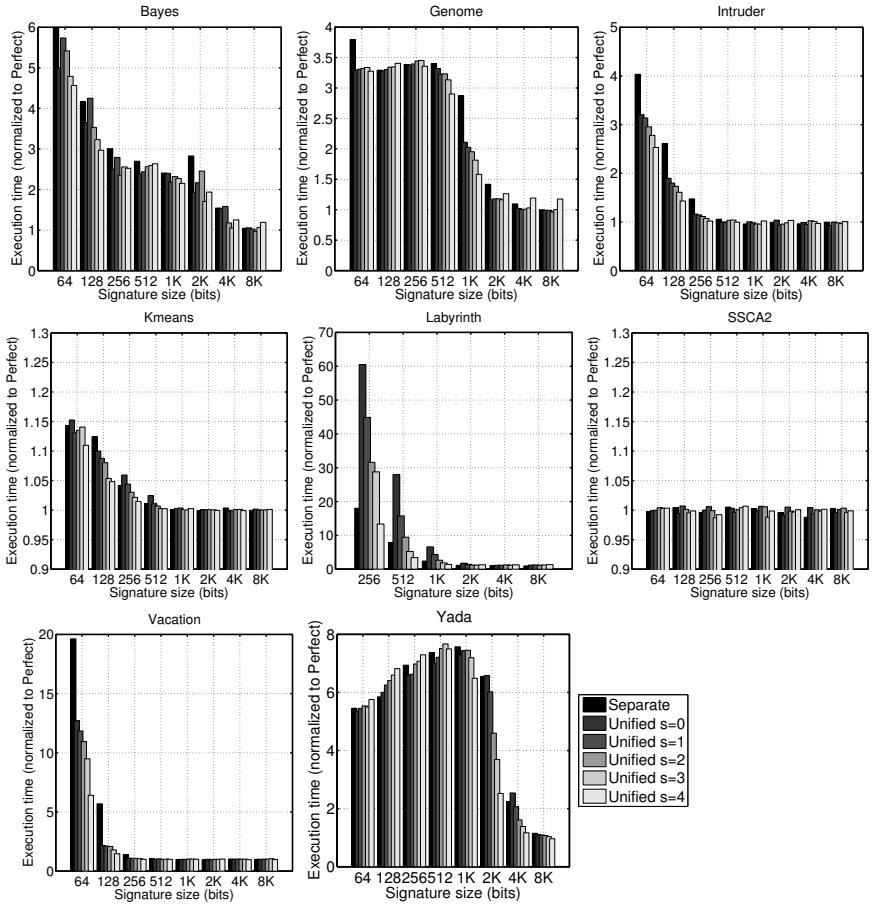


Fig. 3: Execution time normalized to perfect signature comparing separate to unified signatures. Parameter s varies from 0 (2-ported SRAMS) to 4 (1-ported).

6.3 Unified Locality-Sensitive Signature Results

Locality-sensitive hashing [14] takes advantage of locality of reference to store an address stream more concisely in a Bloom filter. Locality-sensitive hash functions store nearby locations sharing some bits of the bit array, thus lowering the occupancy of the filter. For contiguous addresses, the number of hashing outputs with different values is 1. Addresses with distance 2 are different in no more than 2 hashing outputs and, addresses with distance greater than $2^{k-1} - 1$ may have no hashing outputs in common.

Figure 4 shows the results of unified $s = 3$ locality-sensitive signatures. Two different possibilities are shown:

– L1: This scheme makes that the hash functions h_3 and h'_3 assert less bits in their filter. This reduces the false positive rate because of low occupancy,

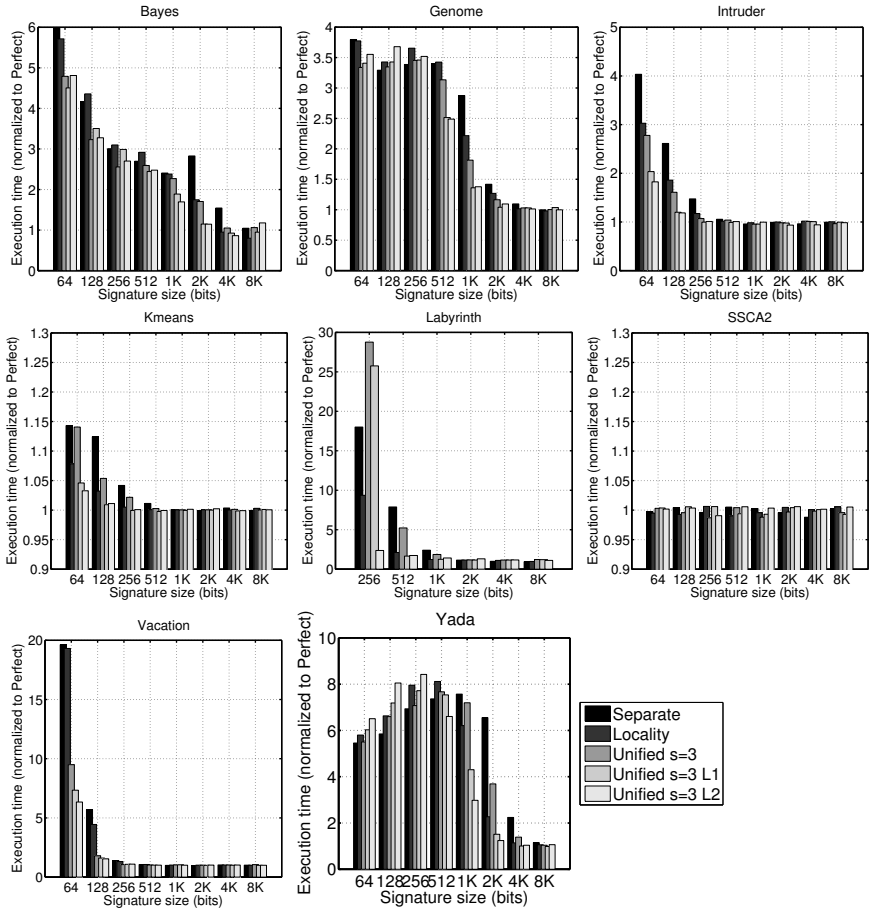


Fig. 4: Execution time normalized to perfect signatures comparing separate, separate locality and unified $s = 3$ signatures enhanced with locality hashing (L1 and L2)

but the filter may fail to discriminate reads/writes from nearby located reads/writes.

– L2: This scheme is the opposite to L1. In this case, h_3 and h'_3 behaves as normal but the others assert less bits. The filter not sharing the hash functions stay the same as in $s = 3$ configuration, discriminating between locations read and written, and the other filters get the locality improvement.

Figure 4 shows similar results for L1 and L2 schemes for all benchmarks except Labyrinth, Genome and Yada. Labyrinth behaves better with L2 for small signatures and, Genome and Yada get slightly worse results for small signatures and L2. Unified locality-sensitive signatures outperform separate ones for the majority of the tested codes.

7 Conclusions

We propose a unified signature design in the context of transactional memory which keeps track of both the read and write sets in the same filter without adding significant hardware complexity. Several configurations of unified signatures are analyzed and evaluated. Additionally, unified signatures are enhanced using locality-sensitive hashing, proposed by the authors in a previous work.

We used the Wisconsin GEMS to implement and evaluate the performance of the proposed unified locality-sensitive signatures. Besides, we used CACTI to evaluate the hardware area and power requirements. Experimental results show that the proposed approach improves the execution performance in most of the benchmark codes, with an insignificant increase in hardware area and a slight increase in power consumption, making of it a good alternative to separate signatures.

Acknowledgment

We would like to thank Dr. Luke Yen (AMD) for providing his patches to adapt STAMP workloads to GEMS simulator. This work has been supported by the Ministry of Education of Spain with project CICYT TIN2006-01078 and by the Junta de Andalucia with project P08-TIC-04341.

References

1. Bloom, B.: Space/time trade-offs in hash coding with allowable errors. Communications of the ACM 13(7), 422–426 (1970)
2. Cao Minh, C., Chung, J., Kozyrakis, C., Olukotun, K.: STAMP: Stanford Transactional Applications for Multi-Processing. In: IEEE Int'l Symp. on Workload Characterization (IISWC'08). pp. 35–46 (2008)
3. Carter, L., Wegman, M.: Universal classes of hash functions. J. Computer and System Sciences 18(2), 143–154 (1979)
4. Ceze, L., Tuck, J., Montesinos, P., Torrellas, J.: BulkSC: Bulk enforcement of sequential consistency. In: 34th Ann. Int'l. Symp. on Computer Architecture (ISCA'07). pp. 278–289 (2007)
5. Ceze, L., Tuck, J., Torrellas, J., Cascaval, C.: Bulk disambiguation of speculative threads in multiprocessors. In: 33th Ann. Int'l. Symp. on Computer Architecture (ISCA'06). pp. 227–238 (2006)
6. Choi, W., Draper, J.: Locality-aware adaptive grain signatures for transactional memories. In: IEEE Int'l. Symp. on Parallel and Distributed Processing (IPDPS'10). pp. 1–10 (2010)
7. Herlihy, M., Moss, J.: Transactional memory: Architectural support for lock-free data structures. In: 20th Ann. Int'l. Symp. on Computer Architecture (ISCA'93). pp. 289–300 (1993)
8. Larus, J., Rajwar, R.: Transactional Memory. Morgan & Claypool Pub. (2007)
9. Magnusson, P., Christensson, M., Eskilson, J., Forsgren, D., Hallberg, G., Hogberg, J., Larsson, F., Moestedt, A., Werner, B., Werner, B.: Simics: A full system simulation platform. IEEE Computer 35(2), 50–58 (2002)

10. Martin, M., Sorin, D., Beckmann, B., Marty, M., Xu, M., Alameldeen, A., Moore, K., Hill, M., Wood, D.: Multifacet's general execution-driven multiprocessor simulator GEMS toolset. ACM SIGARCH Comput. Archit. News 33(4), 92–99 (2005)
11. Mehrara, M., Hao, J., Hsu, P.C., Mahlke, S.: Parallelizing sequential applications on commodity hardware using a low-cost software transactional memory. In: ACM SIGPLAN Conf. on Programming Language Design and Implementation (PLDI'09). pp. 166–176 (2009)
12. Minh, C., Trautmann, M., Chung, J., McDonald, A., Bronson, N., Casper, J., Kozyrakis, C., Olukotun, K.: An effective hybrid transactional memory system with strong isolation guarantees. In: 34th Ann. Int'l. Symp. on Computer Architecture (ISCA'07). pp. 69–80 (2007)
13. Muralimanohar, N., Balasubramonian, R., Jouppi, N.: CACTI 6.0: A tool to model large caches. Tech. Rep. HPL-2009-85, HP Laboratories (2009)
14. Quislant, R., Gutierrez, E., Plata, O., Zapata, E.: Improving signatures by locality exploitation for transactional memory. In: Int'l Conf. on Parallel Architectures and Compilation Techniques (PACT'09). pp. 303–312 (2009)
15. Ramakrishna, M.V., Fu, E., Bahcekapili, E.: Efficient hardware hashing functions for high performance computers. IEEE Trans. on Computers 46(12), 1378–1381 (1997)
16. Sanchez, D., Yen, L., Hill, M., Sankaralingam, K.: Implementing signatures for transactional memory. In: 40th Ann. IEEE/ACM Int'l Symp. on Microarchitecture (MICRO'07). pp. 123–133 (2007)
17. Shriraman, A., Dwarkadas, S., Scott, M.: Flexible decoupled transactional memory support. In: 35th Ann. Int'l. Symp. on Computer Architecture (ISCA'08). pp. 139–150 (2008)
18. Wilton, S., Jouppi, N.: CACTI: an enhanced cache access and cycle time model. IEEE Journal of Solid-State Circuits 31(5), 677 –688 (1996)
19. Yen, L., Bobba, J., Marty, M., Moore, K., Volos, H., Hill, M., Swift, M., Wood, D.: LogTM-SE: Decoupling hardware transactional memory from caches. In: 13th Int'l. Symp. on High-Performance Computer Architecture (HPCA'07). pp. 261–272 (2007)
20. Yen, L., Draper, S., Hill, M.: Notary: Hardware techniques to enhance signatures. In: 41st Ann. IEEE/ACM Int'l Symp. on Microarchitecture (MICRO'08). pp. 234–245 (2008)