Improving Signature Behavior by Irrevocability in Transactional Memory Systems

Ricardo Quislant, Eladio Gutierrez, Emilio L. Zapata, Oscar Plata
Dept. of Computer Architecture
University of Malaga, Spain
Email: {quislant, eladio, zapata, oplata}@uma.es

***Abstract*—Signatures have been proposed in Hardware Transactional Memory (HTM) to represent read and write sets of transactions and decouple transaction conflict detection from private caches. Generally, signatures are implemented as Bloom filters that allow unbounded read/write sets to be summarized in bounded hardware, at the cost of address aliasing that causes false conflict detection. Such conflicts rises exponentially as signature fills so they can lead a parallel program to perform worse than its sequential counterpart (we say that signature saturates).**

In this work, irrevocability is proposed to address the signature saturation problem. When a transaction is near to saturate its signature, the transaction enters an irrevocable state that prevents it from being aborted. Then, such a transaction keeps running while the others are either stalled or allowed to run concurrently. Two variants of irrevocability are analyzed in this paper. Experimental evaluation on an HTM simulator shows the benefits in performance and power consumption of the proposed irrevocability mechanisms.

I. Introduction

Transactional Memory (TM) [1], [2] is a synchronization mechanism that aims to simplify shared memory multiprocessor programming. TM provides the user with the *transaction* construction that ensures atomicity and isolation of the sequence of operations that it encloses. Transactions are allowed to run concurrently unless a conflict (one transaction writes a memory location used by some other one) is detected by the underlying transactional system.

Hardware manufacturers [3], [4] have recently released hardware transactional memory (HTM) support in their commodity off-the-shelf multicore processors. However, such transactional extensions provide a restricted TM system where transactions are unable to survive neither to certain frequent operating system events like migration, paging and scheduling, nor to transactional hardware overflow and cache evictions, amongst others. Many proposals can be found in the literature [5], [6] dealing with virtualization of TM to hide those limitations to programmers. They propose different ways of virtualizing the two main mechanisms of a transactional system: conflict detection and version management.

As far as conflict detection is concerned, it is sought to detach transactional state (essentially, read and write transactional sets) from caches in order to store it in a separate and concise structure that can be easily saved or moved by the operating system. Following such premises, signatures [6], [7] were proposed and adopted as an effective solution to conflict detection virtualization. Signatures, implemented as hash structures, provide a fast and concise way of dealing with large sets of addresses but at the cost of aliasing, that is, different addresses are equally represented in the structure. Aliasing may cause false positives, i.e. positive outcome of signature checks for non-inserted addresses. False positives can give rise to false conflicts between transactions, causing the TM system to waste time and power in resolving such conflicts. As signature fills, false conflict rate can increase to such an extent that the parallel application can perform worse than its sequential version. We refer to this situation as *signature saturation*.

This paper analyzes irrevocability as a mechanism to address the signature saturation problem. Irrevocability is an execution mode of a transaction in which it cannot be aborted. It has been used for different applications [8], [9], [10]. The main one is to assure the execution of irreversible operations, such as, interactive I/O and system calls. But it can be also useful for contention management and for ensuring forward progress. We leverage irrevocable transactions for avoiding parallel application underperforming the sequential one in cases of signature saturation.

The novel contributions of this paper are the following:

- We propose a solution to deal with signature saturation in HTM that consists in making a transaction enter an irrevocable state whenever it reaches a given threshold of occupancy (number of insertions) in its signature. Irrevocability is enforced on foresight of a high probable scenario of false contention. The idea is that a transaction that saturates its signature commits as soon as possible in order to avoid a high false conflict rate scenario with other transactions that could lead to a very poor performance.
- We analyze two irrevocability mechanisms, which are described in the context of a scalable tiled CMP architecture. One mechanism involves that once a transaction becomes irrevocable, all other transactions in the system stall their execution until the irrevocable transaction commits. The other mechanism allow concurrent execution of revocable transactions with the irrevocable one. Irrevocability is a feature that is being included in commercial processors [10] so it could be used at no extra cost.
- We present a performance evaluation of our proposal using the STAMP benchmark suite and the GEMS simulator. Both irrevocability mechanisms and the baseline TM are compared in terms of performance and network power consumption and traffic. For the

analyzed benchmarks, we find a single signature occupancy threshold for triggering the irrevocable state that maximizes parallel performance under the signature saturation situation.

II. Background

Hardware signatures implemented as per-thread Bloom filters was first proposed by Ceze et al. [7] to support operations that efficiently process a set of memory addresses in the context of TM. Bloom filters [11] are time and space-efficient hashing structures that allow an unbounded number of element insertions with the inconvenience of aliasing, that is, a set of different elements are equally represented in the filter. Aliasing causes false positives on membership queries, i.e. the filter says that we inserted an element that we did not (an alias was actually been inserted). The filter is false negative free.

With those characteristics, signatures were subsequently adopted by different TM systems, like LogTM-SE [6] or FlexTM [12], to detach conflict detection from caches. Signatures accelerate conflict detection and provide a means of virtualization for the transactional system, since having the data set of a transaction condensed in a single-purpose structure eases the task of migrating a thread, changing context and surviving cache victimization. Usually, signature-based TM systems have one Bloom filter per transactional data set: read set (RS) and write set (WS).

Regarding irrevocability in the context of hardware transactional memory, two main applications can be identified: supporting operations that cannot be rolled back within transactions (like I/O and system calls), and ensuring transaction forward progress due to hardware resource contention or overflowing. About the first application, Blundell et al. [9] introduces the concept of unrestricted transactions, similar to irrevocability. Unrestricted is an exclusive state (only one transaction at a time may enter such execution mode) and it is used by a transaction to gain the ability to perform I/O operations and system calls. The paper proposes two implementations: unoptimized, where the unrestricted transaction causes the stall of all other threads in the same process until the transaction commits, and optimized, where concurrent execution of multiple restricted transactions and a single unrestricted one is allowed. In the optimized case, unrestricted transactions require to extend the architectural state of the HTM system by means of extra metadata maintained in the caches, at all levels, and main memory, as well as some modifications in the coherence protocol.

To ensure transaction forward progress in the face of hardware resource contention or overflowing, different solutions were proposed. Some best-effort systems, like Intel Haswell [13], does not provide any special progress guarantees in hardware. To ensure progress, a software fallback path must be provided by the programmer. Other proposals [14], [15] modify the conflict resolution method in such a way that older transactions are favored over younger ones when they conflict. In this way, when a transaction becomes the oldest in the system, it will never be aborted due to resource contention. This is similar to an irrevocable transaction, though it could be aborted and rolled back many times before being the oldest one. Systems like Vega Azul [16] provide in hardware non-speculative re-executions. When a transaction is aborted once, it rolls back and executes again non-speculatively. Finally, IBM Blue Gene/Q [10] extends this support as follows. When a transaction aborts, the TM runtime determines how to retry the transaction. It may decide to re-execute the transaction normally several times until some threshold is reached. After that, the transaction is re-started one last time in an irrevocable mode. Moreshet et al. [17] and Ferri et al. [18] focus on energy-aware TM systems and propose serialization schemes that serialize transactions after the abort rate exceeds a given threshold.

TABLE I. Irrevocability states

TS	XS	Description
N	N	Normal mode
N	I	Not possible
I	N	Irrevocability triggered by other thread. Conflict resolution favors (I,I) transaction
I	I	Transaction runs as irrevocable (exclusive state)

We propose irrevocability mechanisms to deal with signature saturation, so a transaction becomes irrevocable when we predict that there is going to be a high contention scenario due to signature filling.

III. Irrevocability Mechanisms

A. Irrevocability

This irrevocability mechanism is inspired by the one described in [10]. We define two irrevocability states, one for the thread and another for the transaction in the thread. Table I summarizes the different combinations of the states: TS stands for Thread State and XS is the Transaction State. Both states can be set to either N, Normal, or I, Irrevocable, and TS can be set for all threads by the thread that is in the irrevocable mode. Figure 1 shows the state diagram of the irrevocability mechanism where the states are pairs (TS, XS). We assume an initial state, Out Xact, for a thread that is executing non-transactional code. Notice that non-transactional code can run in parallel with irrevocable code.

Once a thread enters a transaction, Begin Xact, it checks whether its state, TS, is set to I or N. In case TS=N, the thread enters the (N,N) state where the TM system resolves conflicts normally. However, if another thread got irrevocable and set our TS to I, the thread moves to (I,N) state, where we are allowed to run but the conflict manager changes to favor the thread that got irrevocable. Now, we cannot abort the irrevocable transaction in case of conflict. Notice that transactions running in (I,N) and (N,N) states can commit in parallel with an irrevocable transaction, provided that they do not conflict with the latter and they do not go beyond the signature saturation threshold. These mechanism encourages the parallelism of transactions below the threshold, although it increases contention.

When a transaction reaches the signature saturation threshold it moves to a transition state (I,N→I) that arbitrates the irrevocability state. Only one transaction can be in state (I,I) at a time. So, if several transactions saturate their signatures simultaneously, only one of them is granted the irrevocability state, whereas the others must stall until the former commits (TS=N).

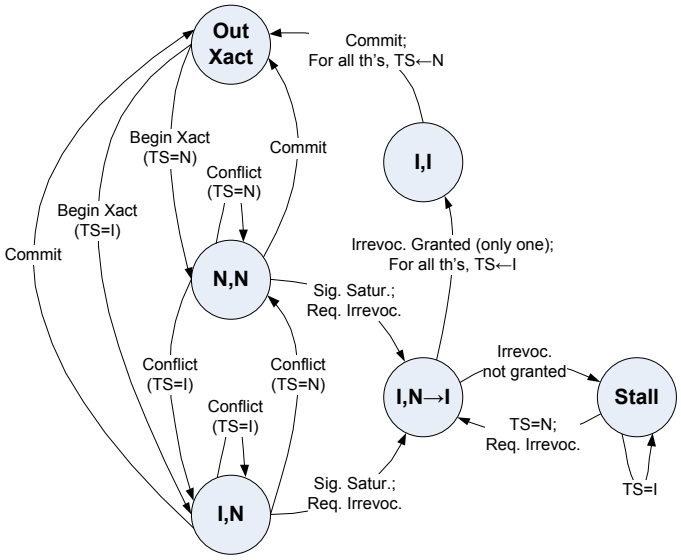

Fig. 1. State diagram of irrevocability.

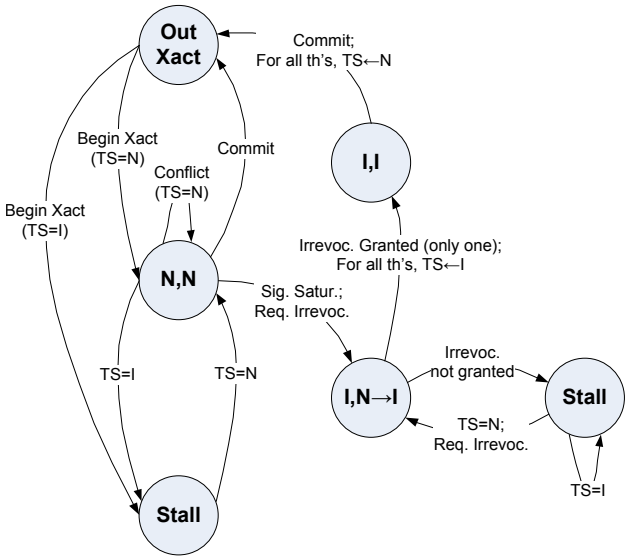

Fig. 2. State diagram of irrevocability with exclusion.

B. Irrevocability with exclusion

The other irrevocability mechanism we have analyzed in this work implies exclusion of other transactions running in the system. A priori, it would seem that this approach limits parallelism, but we will see in Section IV-E that performance loss is negligible in most cases and other collateral benefits spring in terms of energy and network contention.

Figure 2 shows the state diagram for the irrevocability mechanism with exclusion. The scheme is very similar to that of the irrevocability mechanism shown in the previous section. Notice that we still allow non-transactional execution in parallel with irrevocable transactions. However, when a thread moves to the (N,N) state, it can run transactionally only while its TS is N. If (TS=I), then other thread is running an irrevocable transaction so the other transactions stall, and the conflict manager changes so that the stalled transactions are aborted whenever the irrevocable transaction conflicts with them. When the irrevocable transaction commits, it changes the TS of the other threads (TS←N) and these resume execution.

C. Example of operation

Figure 3 shows an example of operation of the irrevocability mechanism described in Section III-A. We have three threads, Th 0-2, running non-transactional code, OutXact. First, Th 1 begins a transaction followed by Th 2. Both happen to saturate the signature at the same time, so they move to the transition state where an arbiter grants irrevocability to one of them (Th 1). Th 2 waits stalling until Th 1's transaction commits.

In the meantime, Th 0 begins a transaction and moves to (I,N) state. It is allowed to continue, although a conflict with the irrevocable transaction makes it abort and restart the transaction. Once Th 0 restarts, Th 1's transaction commits and all threads move to normal state. Then, Th 2's transaction, that was stalling because of signature saturation, is granted irrevocability and resumes execution setting all threads to I state, TS←I.

Figure 4 depicts an example of the operation of the irrevocability mechanism with exclusion. Now, we have the same three threads but only Th 1's transaction gets to signature saturation. Irrevocability is granted to such a transaction and all threads move to (I,N) state. Th 0 is out of transaction so it keeps on going. However, Th 2 was running a transaction so it has to stall because this irrevocability mechanism excludes all transactions running in the system except the irrevocable one.

After a while, Th 0 begins a transaction, but it stalls because of the exclusion (TS=I). Also, Th 1 conflicts with the stalled transaction in Th 2 so Th 2's transaction has to abort. We suppose that the TM system has eager version management, therefore Th 1 has to stall until the abort is consumed and the conflicting data released. In case of lazy version management, it is not necessary to stall the transaction. Finally, Th 1's transaction commit and the other transactions resume in normal mode.

D. HTM implementation

We have implemented the irrevocability schemes described in the previous sections in a HTM system simulator (see Section IV-A) as a token-based distributed mechanism for experimentation purposes. However, irrevocability mechanisms are being included in commercial processors [10] so it could be used at *no extra cost*. Or it could be also implemented as a runtime in software. In any case, the token-based mechanism used in our simulations introduces negligible overhead (see Section IV-D).

The baseline architecture used to implement the irrevocability mechanisms is a tiled multicore processor where each tile (core) comprises a CPU, a primary cache of instructions and data, and a second level cache divided into banks. Cores are connected each other by means of a point-to-point grid interconnect which is accessed by L1 and L2 cache controllers via request and response queues [19]. Each core is to be augmented with a Counter (that includes a zero hardware

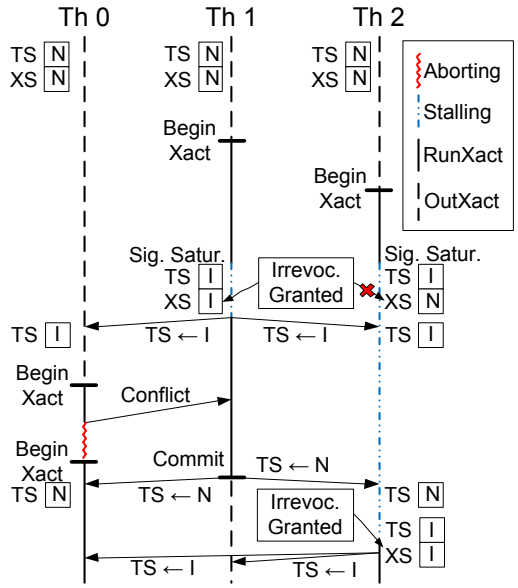

Fig. 3. Example of irrevocability.

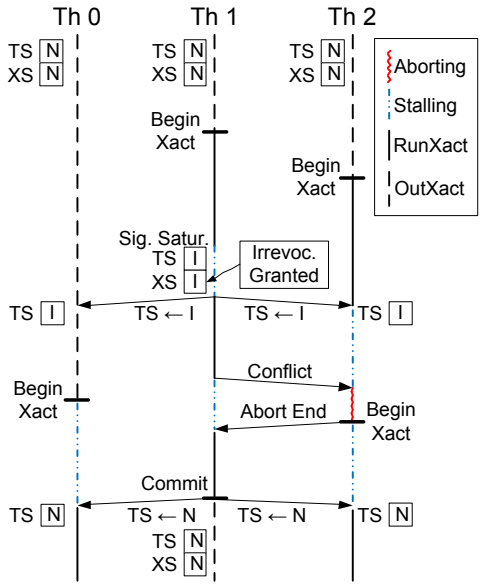

Fig. 4. Example of irrevocability with exclusion.

signal, Z), a flag that holds the thread state, TS, and another flag that holds the transaction state, XS.

Counter keeps count of the number of insertions into the RS/WS signature (in any of the filters). Initially, it is set to a *threshold* value which might be fixed or configured. Such a value could be hard-wired depending on the size of the signature and the amount of false positives allowed (a hard-wired threshold is a good solution, see Section IV-C). When the CPU reads or writes a memory location within the boundaries of a transaction the counter is decremented by 1. If the counter value after an insertion is 0, then the signal Z is raised, indicating that the signature reached the maximum number of insertions before saturation. Then, if Z=1 and we are granted to enter the irrevocable state (arbitrated by the token-based mechanism), the thread sets its XS flag (XS←I) and the TS flag of the rest of the threads (TS←I).

In order to communicate to other threads any TS changes and to implement the token-based arbitration mechanism, some new message types have to be added to the *L1 Cache Controller* protocol. However, there is no need to change the cache coherence protocol, which is a critical element of a multicore processor, difficult to specify/verify [19]. These new messages are control messages and they are not related to addresses unlike coherence messages. Therefore, as such messages do not include addresses, they will be short packets including the message type, the source and the destination processor ID.

E. Data set count considerations

A problem that we have overlooked so far is related to how we count insertions into the signature. One method would be to decrement the counter on each insertion. However, load and store repetitions could decrement the counter too fast if signature insertions are not previously checked for a hit. So, our scheme comprises an insertion operation that checks if the address was previously inserted and then the counter is decremented if not. Conventionally, insertion operations always insert the address into the signature, regardless of whether the address is already inserted or not. Since signatures are implemented as SRAMs, our proposal could use a read/modify/write SRAM cycle for the insertion operation without compromising performance significantly, where the address is read early in the cycle and written late in the cycle if applies.

Counting insertions trusting the signature can lead to miscounting due to false positives. If we inserted an address X that is an alias of an address Y and then we insert Y into the filter, Y will hit the signature so the counter will not be decremented. Then, irrevocability mode might not be engaged when false positive rate is too high. Note that false positives and saturation threshold are dependent each other. The higher the threshold the more false positives. However, we will see in next section that a low threshold is a good choice, so the miscounting problem is negligible.

Also, we have considered only one counter for both read set and write set signatures, while a counter per data set could be a possibility. However, [20], [21] suggest that unifying in terms of signatures is a good choice to deal with asymmetry in data sets.

IV. Experimental Evaluation

A. Methodology

We have used the Simics [22] full system execution-driven simulator to evaluate the irrevocability mechanisms, together with the Wisconsin Multifacet Project's GEMS [23] toolset module for Simics. GEMS includes an implementation of the LogTM-SE HTM system [6] in its Ruby module that we have modified to include the irrevocability schemes.

The target system is a tiled CMP that comprises 16 in-order single-issue cores, private 32KB split L1 caches; a unified, shared, banked 8MB L2 cache; and a full bit-vector L2 directory. For the network time and power modeling we have used Garnet [24] and Orion [25] from Princeton. The parameters for the interconnect are 128bit flits, 4 virtual

TABLE II. WORKLOADS: INPUT PARAMETERS AND TM CHARACTERISTICS

Bench	Input	# xact	% time in xact	XactID(avg|RS|/avg|WS|)
Bayes	-v32 -r1024 -n2 -p20 -i2 -e2 -s1	624	98%	0(2/1) 1(22/3) 2(6/3) 3(22/11) 4(53/20) 5(2/1) 6(74/33) 7(34/12) 8(4/1) 9(484/274) 10(55/34) 11(75/44) 12(21/3)
Genome	-g512 -s64 -n16384	30304	93%	0(507/9) 1(4/1) 2(8/4) 3(6/4) 4(6/3)
Intruder	-a10 -l64 -n2048 -s1	91374	94%	0(4/1) 1(36/6) 2(2/1)
Kmeans	-m15 -n15 -t0.05 -i random-n1024-d1024-c16	2760	15%	0(134/65) 1(1/1) 2(2/1)
Labyrinth	-i random-x32-y32-z3-n48	126	100%	0(3/1) 1(160/223) 2(7/3)
Vacation	-n16 -q60 -u90 -r16384 -t4096	4095	97%	0(149/14) 1(33/6) 2(132/14)
Yada	-a20 -i633.2	5286	100%	0(8/2) 1(1/0) 2(351/225) 3(1/1) 4(9/2) 5(2/2)

channels per virtual network, and 4 flits per buffer. For the power calculations, we configured Orion with a 32nm process, NVT transistors and 1.0 Vdd.

We used all the codes of the Stanford's STAMP suite [26] for experimentation, except SSCA2, which has been proved to be signature-insensitive [27] because of its small transactions (around 3 and 2 blocks read and written on average) and because it spends most time outside transactions. Table II summarizes input parameters and main transactional characteristics of the benchmarks. Namely, the number of transactions that successfully commits "# xact", the percentage of time running transactions "% time in xact", and the average cardinality of the RS and the WS for each transaction, in cache blocks. Each transaction is identified by its ID, starting from 0 and numbered in order of appearance in the code.

B. Results

Figure 5 shows the execution time normalized to perfect signatures (lower is better) of the sequential application, the base TM system and the system with the irrevocability mechanism described in Section III-A. The threshold beyond which the irrevocability begins is shown as a fraction of the size of the signature (RS and WS included), and it varies from a half to 1/32nd. Since the signature size ranges from 128 to 16K (abscissa axis), the minimum threshold is 4 insertions and the maximum 8K insertions.

Notice that, due to the GEMS slow simulations, the benchmark workload sizes are much smaller than in real scenarios. Real problems demand much larger signatures than in our experiments to avoid saturation.

As said in Section I, signatures can lead the parallel version of an application to perform worse than the sequential. This can be seen in Figure 5 as the signature size decreases. False positives increase and the base TM system performance falls down beyond that of the sequential for most benchmarks. For Genome and Intruder the performance degrades significantly, but the base TM system does not cut the sequential, yet it stays closely. However, Kmeans is almost unaffected by the size of the signature. Although Kmeans exhibits large transactions on average, it spends short time in transactions (see Table II) and gets a high speedup difficult to degrade to such an extent.

As far as the irrevocability mechanism is concerned, we can see a trend towards the base TM system as threshold increases. However, irrevocability tends to sequential as threshold decreases, which was expected. Presumably, a one insertion threshold ($Counter = 1$) would yield the performance of the sequential application plus a slight overhead due to irrevocability control messages. This trend can be perfectly noted in Labyrinth where Irrevoc 1/2 is quite similar to Base whereas Irrevoc 1/32 almost matches Sequential.

C. Choosing the right threshold

A problem arises when it comes to choosing the best threshold for our application. As we said in Section III-D, the choice can be determined in different ways: it can be fixed, configured or with a hard-wired threshold. In this case, we tested several thresholds on a per-application basis and we found that a threshold of 1/8th the signature size yields the best results for all benchmarks except for Kmeans, which get the best performance with 1/2nd the signature size (the same results than Base). However, we would get a negligible loss of performance by choosing Irrevoc 1/8 for this benchmark, and it would not get worse results than its sequential version in any case. So, a hard-wired threshold can be a good option, since it would not be necessary to augment the ISA with an instruction to set the threshold, thus hiding the difficulties for a programmer (or compiler) to deal with a new instruction and enhancing the portability of transactional applications.

The theoretical maximum percentage of false positives that we can obtain with a threshold of 1/8th the signature size is given by the equation:

$$p_{\text{FP}}(M,n,k) = \left(1 - \left(1 - \frac{1}{M}\right)^{nk}\right)^k \approx \left(1 - e^{-\frac{nk}{M}}\right)^k \quad (1)$$

where M is the signature size, n the number of insertions and k the number of hash functions. And p_{FP} can be simplified by using the Taylor series expansion of the exponential function, $e^x = \sum_{n=0}^{\infty} \frac{1}{n!} x^n$ [28]. Replacing $n = M/8$ and $k = 4$ in the last equation we obtain $max(p_{\text{FP}}) \simeq 0.023969$. Then, it is better to trigger irrevocability when signatures are yielding more than 2.4% of false positives than allowing parallel progress with increased contention due to false conflicts.

D. Irrevocability mechanism overhead

Figure 6 shows a cycle breakdown for the benchmarks using Irrevoc 1/8 and 128bit signatures. To measure the overhead of our implementation of irrevocability we broke down the cycles into overhead cycles (Irrevoc OH), comprising the cycles spent in requesting and passing the token, the cycles to pass irrevocability state messages, and the cycles that a transaction spends stalling when it is running alone in irrevocable mode. The cycles of the transactions running in irrevocable mode are shown as Run Irrevoc. In addition, non-transactional (Out Xact) and transactional cycles (Run Xact) are shown. Notice that our token-based implementation of the irrevocability mechanism does not incur significant overhead

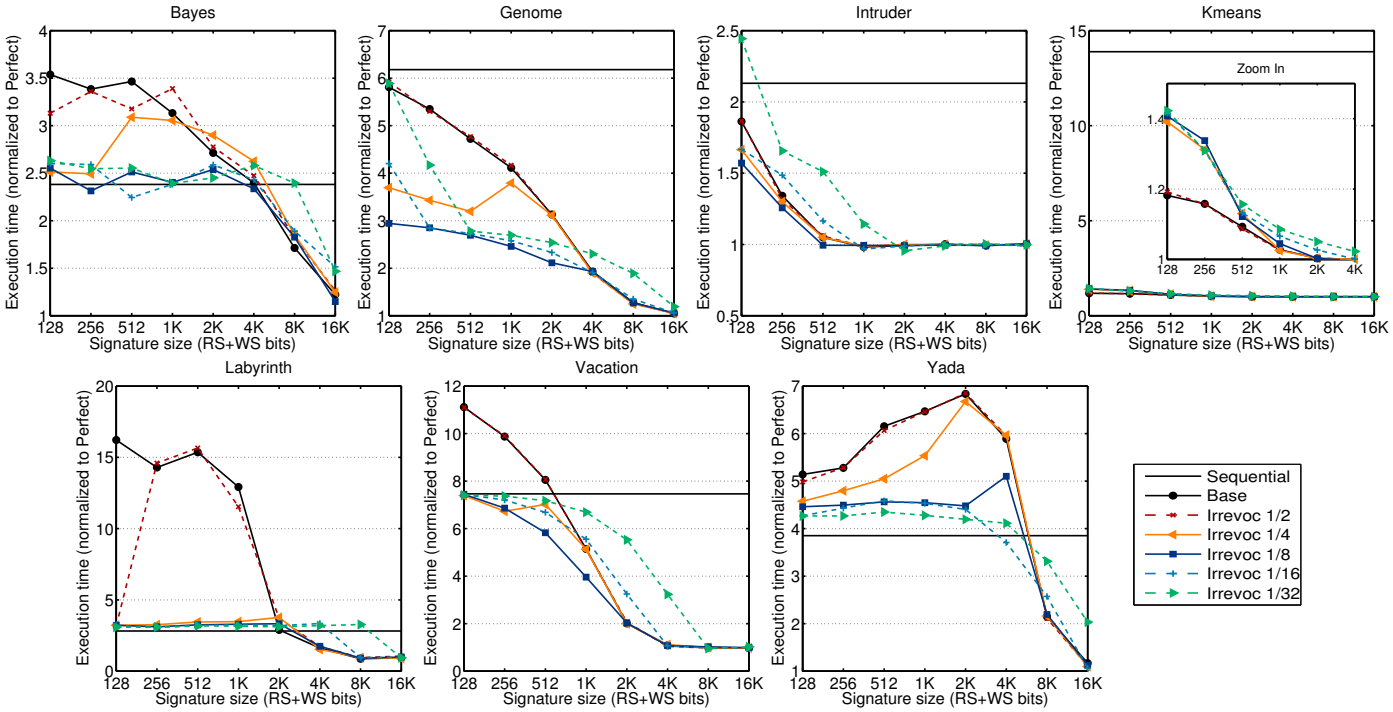

Fig. 5. Results of the irrevocability mechanism (Irrevoc) varying the threshold from 1/2nd to 1/32nd the signature size.

as Irrevoc OH bars can barely be seen in the figure. Practically, the whole of the execution lies in the Out Xact, Run Xact and Run Irrevoc phases of the application.

So, as the overhead of the irrevocability scheme is negligible we can conclude that the reason why performance is better or worse than sequential is because of the Run Xact parallel phase. Depending on the benchmark, the transactional parallel phase may exhibit more or less real contention. In the case of Labyrinth, for example, we obtain a slight performance decrease for Irrevoc 1/8 over Sequential, and the saturation threshold for 128bit signatures is set to 16 insertions, so ID1 transactions are all irrevocable (see Table II) while ID0 and ID2 are not. ID0 and ID2 transactions are in charge of popping a pair of coordinates from a queue and inserting a path to a global list respectively, which are conflict-prone procedures, so the parallel phase is not helping to enhance performance.

On the other hand, Genome shows very good results when using Irrevoc 1/8. Genome comprises 5 transactions (see Table II) where only the first one triggers the irrevocable mode as it goes beyond the 16 insertion threshold for 128bit signatures. The rest are small transactions whose retry count is close to zero, so the transactional phase here is quite uncontended thus enhancing overall performance. Also, when comparing the irrevocability scheme to the base system, the retry count of transaction ID0 reduces drastically from 4 to 1. Therefore, irrevocability maintains the benefit of small uncontended parallel transactions and limits the harm of large signature-saturating transactions in this case.

E. Reducing interconnection network power and traffic

Figure 7 shows the results obtained for both irrevocability mechanisms, Irrevoc and IrrevocEx, described in Section III-A

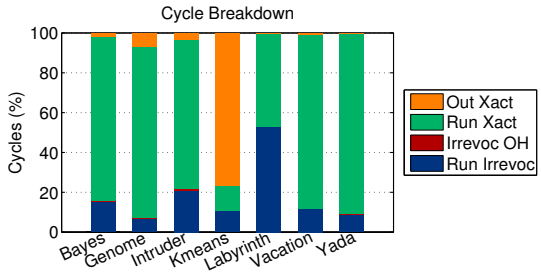

Fig. 6. Cycle breakdown for Irrevoc 1/8 and 128bit signature size.

and Section III-B respectively, with a signature saturation threshold of 1/8th the signature size. Base and Sequential are depicted as well. We can see that irrevocability with exclusion, IrrevocEx, settles very close to irrevocability without exclusion, with two exceptions, Bayes and Intruder.

Bayes is a benchmark that shows high variability because its performance depends on the order in which dependencies are learned [26]. However, we can notice a tendency for IrrevocEx to outperform Irrevoc when using intermediate signature sizes. The number of aborts that IrrevocEx yields is lower than that of Irrevoc for Bayes. This fact points out that Bayes shows high contention, increased by false contention introduced by the signature. Therefore, allowing transactional execution in parallel with irrevocable transactions is not always profitable.

Conversely, Intruder seems to benefit from the increased parallelism offered by Irrevoc. IrrevocEx shows more aborts than Irrevoc. The different scenarios of contention seems to have an impact on performance, as one mechanism forces different transaction interleaving than the other. Thus, Irrevoc seems to arrange transactions 0, 1 and 2 (see Table II) in a

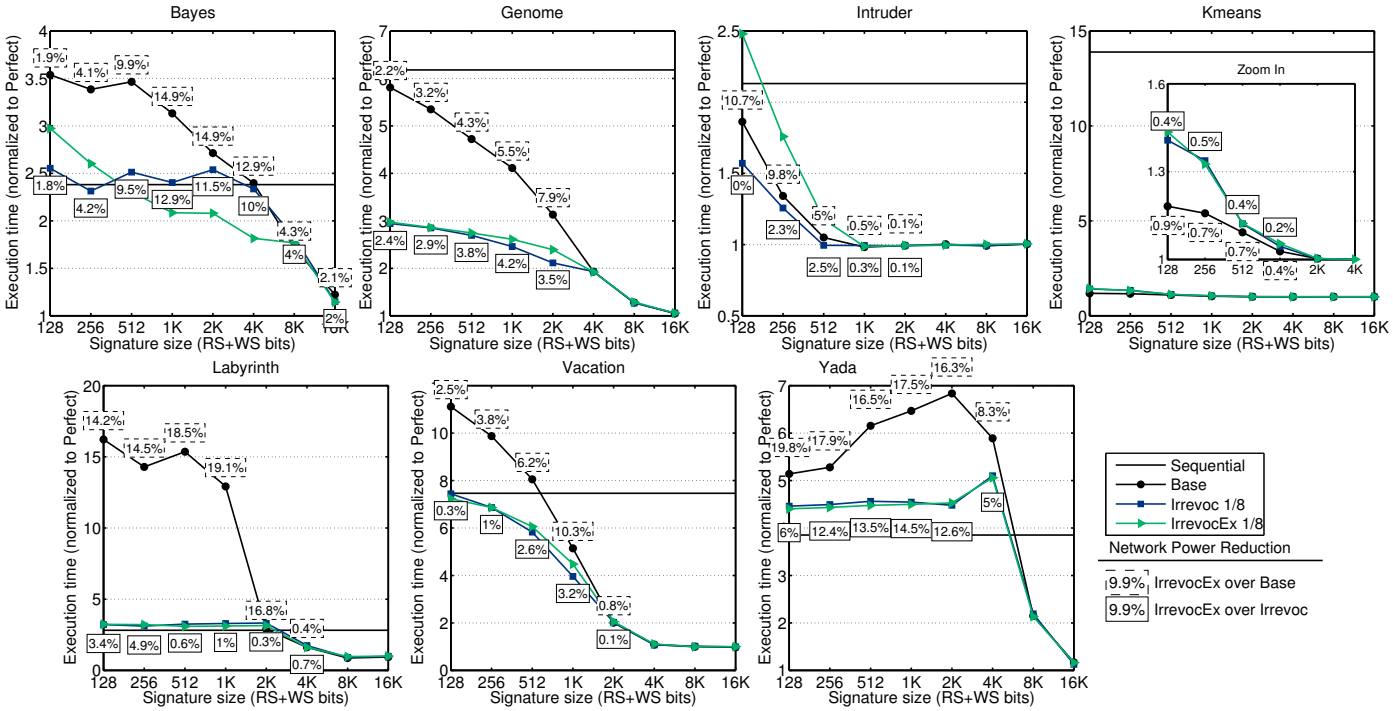

Fig. 7. Results of both irrevocability mechanisms with and without exclusion and a signature threshold of 1/8th. Power reduction is shown in percentage.

way that contention is reduced.

The rest of the benchmarks do not show much difference on using either irrevocability mechanisms. Notice that if two transactions that execute the same code in parallel (let say transaction 1 in Intruder), begin at the same time and do not conflict with each other, both would reach the signature threshold almost at the same time. Then, both mechanisms would behave similarly. One scenario that benefits Irrevoc is that there is an irrevocable transaction running in the system and the other threads are running different transactions whose data set size is lower than the signature threshold and do not conflict with each other and with the irrevocable transaction. Also, if the other transactions go beyond the threshold and they do not conflict with the irrevocable transaction, the transactional work done so far would be more than that for the IrrevocEx mechanism. However, those scenarios do not show up frequently for the concerned benchmarks.

Regarding power, Figure 7 shows the network dynamic power reduction, both for links and routers, of IrrevocEx with respect to Base and Irrevoc. Overall, both irrevocability mechanisms reduce the power requirements of the interconnect when compared with the base system. However, the reduction for IrrevocEx is significant, getting up to nearly 20% in Bayes, Labyrinth and Yada. The exclusion of all transactions running in the system, but the irrevocable one, prevent the system from consuming energy on computations that are eventually aborted because of conflicts. When compared with Irrevoc, the power reduction gain is less obvious. Around 3% for Genome, Intruder, Labyrinth and Vacation, but up to 10-15% for Bayes and Yada, which do not get benefit from the increased concurrency of Irrevoc.

Another benefit of the irrevocability mechanisms is their impact in network traffic. When using Irrevoc, the transactions that reach the signature threshold stall until one gets the token. On the other hand, the case of IrrevocEx is more noticeable since all transactions stall except the irrevocable one. That is, whereas in the Base TM system, stalling transactions are continuously probing the network to see if the memory location they are waiting for is available (like a spinlock), in IrrevocEx, the transactions stalling due to irrevocability are sleeping until an IRREVOCEND message is received, thus not feeding the network with new messages.

Figure 8 shows the average number of flits per cycle per link of the network for Base, Irrevoc and IrrevoEx, and a signature size of 512bit. The traffic reduction is substantial when using the IrrevocEx mechanism, which drastically cuts the network traffic for all the benchmarks except Kmeans, which does not spend too much time in transactions (see Table II). The traffic reduction in Irrevoc is less noticeable because threads are not stalled on an irrevocability event.

The reduction in the number of messages running through the network and the subsequent energy consumption savings are important features that make the irrevocability with exclusion a mechanism worthy of consideration. Even more when we have seen that performance is scarcely affected.

V. CONCLUSIONS

This work presents irrevocability mechanisms to deal with signature saturation in TM. A transaction moves to an irrevocable execution mode when the number of insertions into the signature reaches a given saturation threshold. Then, only such a transaction is allowed to continue while the other transactions running in the system are stalled (irrevocability with exclusion) or left to run until they reach the threshold (irrevocability). It is

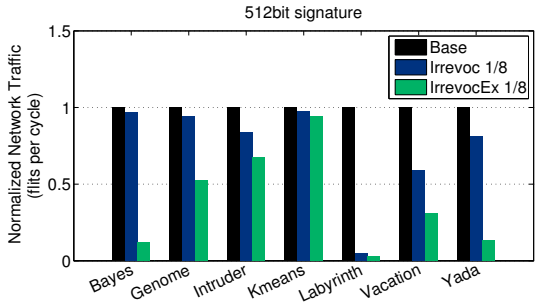

Fig. 8. Interconnect network traffic in average number of flits per cycle per link of the network, normalized to that of the Base system.

a novel application of irrevocability where signature occupancy is used to predict contention situations that may cause a high number of transaction aborts. This helps to save processor cycles and energy due to unnecessary aborts while assuring forward progress.

We implemented our irrevocability proposals in the Wisconsin GEMS simulator and obtained encouraging results from the STAMP benchmark suite and a signature threshold of 1/8th the signature size. Also, we show that our irrevocability mechanism implementation does not incur significant overhead, both in hardware and performance, and that commercial processors are including irrevocability so it could be used at no extra cost. Finally, network power and traffic are reduced because of passive stalling during irrevocability mode, specially when using irrevocability with exclusion.

ACKNOWLEDGMENT

This work has been supported by the Government of Spain with project CICYT TIN2010-16144.

REFERENCES

[1] M. Herlihy and J. Moss, "Transactional memory: Architectural support for lock-free data structures," in *20th Ann. Int'l. Symp. on Computer Architecture (ISCA'93)*, 1993, pp. 289–300.

[2] J. Larus and R. Rajwar, *Transactional Memory*. Morgan & Claypool Pub., 2007.

[3] R. Haring, M. Ohmacht, T. Fox, M. Gschwind, D. Satterfield, K. Sugavanam, P. Coteus, P. Heidelberger, M. Blumrich, R. Wisniewski, A. Gara, G.-T. Chiu, P. Boyle, N. Chist, and C. Kim, "The IBM Blue Gene/Q compute chip," *IEEE Micro*, vol. 32, no. 2, pp. 48–60, 2012.

[4] B. Goel, R. Titos, A. Negi, S. A. McKee, and P. Stenstrom, "Performance and energy analysis of the restricted transactional memory implementation on Haswell," in *28th IEEE Int'l Parallel & Distributed Processing Symp. (IPDPS'14)*, may 2014.

[5] R. Rajwar, M. Herlihy, and K. Lai, "Virtualizing transactional memory," in *32th Ann. Int'l. Symp. on Computer Architecture (ISCA'05)*, 2005, pp. 494–505.

[6] L. Yen, J. Bobba, M. Marty, K. Moore, H. Volos, M. Hill, M. Swift, and D. Wood, "LogTM-SE: Decoupling hardware transactional memory from caches," in *13th Int'l. Symp. on High-Performance Computer Architecture (HPCA'07)*, 2007, pp. 261–272.

[7] L. Ceze, J. Tuck, J. Torrellas, and C. Cascaval, "Bulk disambiguation of speculative threads in multiprocessors," in *33th Ann. Int'l. Symp. on Computer Architecture (ISCA'06)*, 2006, pp. 227–238.

[8] A. Welc, S. Bratin, and A.-R. Adl-Tabatabai, "Irrevocable transactions and their applications," in *20th ACM Symp. on Parallelism in Algorithms and Architectures (SPAA'08)*, june 2008, pp. 285–296.

[9] C. Blundell, E. C. Lewis, and M. M. Martin, "Unrestricted transactional memory: Supporting I/O and system calls within transactions," Dept. Computer and Information Science, University of Pennsylvania, Tech. Rep. TR–CIS–06–09, 2006.

[10] A. Wang, M. Gaudet, P. Wu, J. N. Amaral, M. Ohmacht, C. Barton, R. Silvera, and M. Michael, "Evaluation of Blue Gene/Q hardware support for transactional memories," in *21st Int'l Conf. on Parallel Architectures and Compilation Techniques (PACT'12)*, 2012, pp. 127–136.

[11] B. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Communications of the ACM*, vol. 13, no. 7, pp. 422–426, 1970.

[12] A. Shriraman, S. Dwarkadas, and M. Scott, "Flexible decoupled transactional memory support," in *35th Ann. Int'l. Symp. on Computer Architecture (ISCA'08)*, 2008, pp. 139–150.

[13] M. D. Wang, B. Mihai, L. Li, S. Sharifymoghaddam, G. Steffan, and C. Amza, "Exploring the performance and programmability design space of hardware transactional memory," in *9th ACM SIGPLAN Workshop on Transactional Computing (TRANSACT'14)*, march 2014.

[14] C. Ananian, K. Asanovic, B. Kuszmaul, C. Leiserson, and S. Lie, "Unbounded transactional memory," in *11th Int'l. Symp. on High-Performance Computer Architecture (HPCA'05)*, 2005, pp. 316–327.

[15] K. Moore, J. Bobba, M. Moravan, M. Hill, and D. Wood, "LogTM: Log-based transactional memory," in *12th Int'l. Symp. on High-Performance Computer Architecture (HPCA'06)*, 2006, pp. 254–265.

[16] C. Click, "Azul's experencies with hardware transactional memory," in *Bay Area Workshop on Transactional Memory*, 2009.

[17] T. Moreshet, R. I. Bahar, and M. Herlihy, "Energy-aware microprocessor synchronization: Transactional memory vs. locks," in *Workshop on Memory Performance Issues (held in conjunction with the Int'l. Symp. on High-Performance Computer Architecture, HPCA '06)*, 2006.

[18] C. Ferri, S. Wood, T. Moreshet, R. Iris Bahar, and M. Herlihy, "Embedded-TM: Energy and complexity-effective hardware transactional memory for embedded multicore systems," *J. of Parallel and Distributed Computing*, vol. 70, no. 10, pp. 1042–1052, 2010.

[19] D. J. Sorin, M. Plakal, A. E. Condon, M. D. Hill, M. M. K. Martin, and D. A. Wood, "Specifying and verifying a broadcast and a multicast snooping cache coherence protocol," *IEEE Trans. Parallel and Distributed Systems*, vol. 13, no. 6, pp. 556–578, Jun. 2002.

[20] W. Choi and J. Draper, "Unified signatures for improving performance in transactional memory," in *IEEE Int'l. Parallel Distributed Processing Symp. (IPDPS'11)*, May 2011, pp. 817–827.

[21] R. Quislant, E. Gutierrez, O. Plata, and E. Zapata, "Hardware signature designs to deal with asymmetry in transactional data sets," *IEEE Trans. on Parallel and Distributed Systems*, vol. 24, no. 3, pp. 506–519, 2013.

[22] P. Magnusson, M. Christensson, J. Eskilson, D. Forsgren, G. Hallberg, J. Hogberg, F. Larsson, A. Moestedt, B. Werner, and B. Werner, "Simics: A full system simulation platform," *IEEE Computer*, vol. 35, no. 2, pp. 50–58, 2002.

[23] M. Martin, D. Sorin, B. Beckmann, M. Marty, M. Xu, A. Alameldeen, K. Moore, M. Hill, and D. Wood, "Multifacet's general execution-driven multiprocessor simulator GEMS toolset," *ACM SIGARCH Computer Architecture News*, vol. 33, no. 4, pp. 92–99, 2005.

[24] N. Agarwal, T. Krishna, L.-S. Peh, and N. Jha, "GARNET: A detailed on-chip network model inside a full-system simulator," in *IEEE Int'l. Symp. on Performance Analysis of Systems and Software (ISPASS'09)*, April 2009, pp. 33–42.

[25] A. Kahng, B. Li, L.-S. Peh, and K. Samadi, "ORION 2.0: A fast and accurate noc power and area model for early-stage design space exploration," in *Design, Automation & Test in Europe (DATE'09)*, April 2009, pp. 423–428.

[26] C. Minh, J. Chung, C. Kozyrakis, and K. Olukotun, "STAMP: Stanford Transactional Applications for Multi-Processing," in *IEEE Int'l Symp. on Workload Characterization (IISWC'08)*, 2008, pp. 35–46.

[27] R. Quislant, E. Gutierrez, O. Plata, and E. L. Zapata, "LS-Sig: Locality-sensitive signatures for transactional memory," *IEEE Trans. on Computers*, vol. 62, no. 2, pp. 322–335, 2013.

[28] D. Sanchez, L. Yen, M. Hill, and K. Sankaralingam, "Implementing signatures for transactional memory," in *40th Ann. IEEE/ACM Int'l Symp. on Microarchitecture (MICRO'07)*, 2007, pp. 123–133.