Application-Specific Signatures for Transactional Memory in Soft Processors

Martin Labrecque, Mark Jeffrey, and J. Gregory Steffan

Department of Electrical and Computer Engineering, University of Toronto
{martinl,markj,steffan}@eecg.toronto.edu

Abstract. As reconfigurable computing hardware and in particular FPGA-based systems-on-chip comprise an increasing number of processor and accelerator cores, supporting sharing and synchronization in a way that is scalable and easy to program becomes a challenge. *Transactional memory* (TM) is a potential solution to this problem, and an FPGA-based system provides the opportunity to support TM in hardware (HTM). Although there are many proposed approaches to HTM support for ASICs, these do not necessarily map well to FPGAs. In particular in this work we demonstrate that while *signature*-based conflict detection schemes (essentially bit vectors) should intuitively be a good match to the bit-parallelism of FPGAs, previous schemes result in either unacceptable multicycle stalls, operating frequencies, or false-conflict rates. Capitalizing on the reconfigurable nature of FPGA-based systems, we propose an application-specific signature mechanism for HTM conflict detection. Using both real and projected FPGA-based soft multiprocessor systems that support HTM and implement threaded, shared-memory network packet processing applications, relative to signatures with bit selection we find that our application-specific approach (i) maintains a reasonable operating frequency of 125MHz, (ii) has an area overhead of only 5%, and (iii) achieves a 9% to 71% increase in packet throughput due to reduced false conflicts.

1 Introduction

As reconfigurable computing systems and in particular FPGAs become more dense, they are increasingly used to implement larger and more complex systems-on-chip composed of multiple processor and acceleration cores that must synchronize and share data. While systems based on shared memory can ease communication between cores, the programmer's job of inserting correct lock-based synchronization can be error-prone and difficult to debug, and the resulting *critical sections* of code within locks are serialized, thus reducing the overall parallelism and efficiency of the system.

Transactional memory (TM) [1–3] can potentially address both challenges. First, TM provides an easier programming model for synchronization, allowing programmers to specify more coarse-grain critical sections (transactions) to be executed atomically. Second, TM reduces contention on these larger critical sections by executing transactions in parallel so long as their memory accesses do not conflict. Hence we are motivated to implement TM for multiple-core reconfigurable computing systems; in this paper we focus on implementing TM

for an FPGA-based soft multiprocessor. While TM can be implemented purely in software (STM), an FPGA-based system can be extended to support TM in hardware (HTM) with much lower performance overhead than an STM. There are many known methods for implementing HTM for an ASIC multicore processor, although they do not necessarily map well to an FPGA-based system.

In this paper we focus specifically on the design of the conflict detection mechanism for FPGA-based HTM, and find that an approach based on *signatures* [4] is a good match for FPGAs because of the underlying bit-level parallelism. A signature is essentially a bit-vector [5] that tracks the memory locations accessed by a transaction via hash indexing. However, since signatures normally have many fewer bits than there are memory locations, comparing two signatures can potentially indicate costly false-positive conflicts between transactions. Hence prior HTMs employ relatively large signatures—thousands of bits long—to avoid such false conflicts. One important goal for our system is to be able to compare signatures and detect conflicts in a single pipeline stage, otherwise memory accesses would take an increasing number of cycles and degrade performance. However, as we demonstrate in this paper, implementing previously proposed large signatures in the logic-elements of an FPGA can be detrimental to processor operating frequency. Or, as an equally unattractive alternative, one can implement large and sufficiently fast signatures using block RAMs but only if the indexing function is trivial—which can itself exacerbate false-positive conflicts and negate the value of larger signatures.

1.1 Application-Specific Signatures

To summarize, our goal is to implement a moderately-sized signature mechanism while minimizing the resulting false conflicts. We capitalize on the reconfigurable nature of the underlying FPGA and propose a method for implementing an application-specific signature mechanism that achieves these goals. An application-specific signature is created by (i) profiling the memory addresses accessed by an application, (ii) using this information to build and optimize a *trie* (a tree based on address prefixes) that allocates more branches to frequently-conflicting address prefixes, and (iii) implementing the trie in a conflict detection unit using simple combinational circuits.

Our evaluation system is built on the NetFPGA platform [6], comprising a Virtex II Pro FPGA, 4 1GigE MACs, and 200MHz DDR2 SDRAM. On it we have implemented a dual-core multiprocessor (the most cores that our current platform can accommodate), composed of 125MHz MIPS-based soft processors, that supports an *eager* HTM [7] via a shared data cache. We have programmed our system to implement several threaded, shared-memory network packet processing applications (packet classification, NAT, UDHCP, and intrusion detection).

We use a cycle-accurate simulator to explore the signature design space, and implement and evaluate the best schemes in our real dual-core multiprocessor implementation. For comparison, we also report the FPGA synthesis results for a conflict detection unit supporting 4 and 8 threads. Relative to signatures with bit selection (the only other signature implementation that can maintain a

reasonable operating frequency of 125MHz), we find that our application-specific approach has an area overhead of only 5%, and achieves a 9% to 71% increase in packet throughput due to reduced false conflicts.

1.2 Related Work

There is an abundance of prior work on TM and HTM. Most prior FPGA implementations of HTM were intended as fast simulation platforms to study future multicore designs [1, 2], and did not specifically try to provide a solution tuned for FPGAs. Conflict detection has been previously implemented by checking extra bits per line in private [1, 2] or shared [3] caches. In contrast with caches with finite capacity that require complex mechanisms to handle cache line collisions for speculative data, signatures can represent an unbounded set of addresses and thus do not overflow. Signatures can be efficiently cleared in a single cycle and therefore advantageously leverage the bit-level parallelism present in FPGAs. Because previous signature work was geared towards general purpose processors [5, 8, 9], to the best of our knowledge there is no prior art in customizing signatures on a per-application basis.

1.3 Contributions

This paper makes the following contributions: (i) we describe the first soft processor cores integrated with transactional memory, and evaluated on a real (and simulated) system with threaded applications that share memory; (ii) we demonstrate that previous signature schemes result in implementations with either multicycle stalls, or unacceptable operating frequencies or false conflict rates; (iii) we demonstrate that application-specific signatures can allow conflict detection at acceptable operating frequencies (125MHz), single cycle operation, and improved false conflict rates—resulting in significant performance improvements over alternative schemes.

2 Previous Signature Implementations for HTM

A TM system must track read and write accesses for each transaction (the read and write sets), hence an HTM system must track read and write sets for each hardware thread context. The signature method of tracking read and write sets implements Bloom filters [5], where an accessed memory address is represented in the signature by asserting the k bits indexed by the results of k distinct hashes of the address, and a membership test for an address returns true only if all k bits are set. Since false conflicts can have a significant negative impact on performance, the number and type of hash functions used must be chosen carefully. In this paper we consider only the case where each one of the k hash functions indexes a different partition of the signature bits—previously shown to be more efficient [5]. The following reviews the four known hash functions that we consider in this paper.

Bit Selection [5] This scheme directly indexes a signature bit using a subset of address bits. An example 2-bit index for address $a = [a_3 a_2 a_1 a_0]$ could simply be $h = [a_3,\ a_2]$. This is the most simple scheme (i.e., simple circuitry) and hence is important to consider for an FPGA implementation.

H_3 [5] The H_3 class of hash functions is designed to provide a uniformly-distributed hashed index for random addresses. Each bit of the hash result $\mathbf{h} = [h_1,\ h_0]$ consists of a separate XOR ($\oplus$) tree determined by the product of an address $a = [a_3 a_2 a_1 a_0]$ with a fixed random matrix H as in the following example with a 4-bit address and a 2-bit hash [9]:

$$[h_1,\ h_0] = \mathbf{a}H = [a_3 a_2 a_1 a_0] \begin{bmatrix} 1 & 0 \\ 1 & 1 \\ 0 & 1 \\ 1 & 0 \end{bmatrix} = [a_3 \oplus a_2 \oplus a_0,\ a_2 \oplus a_1] \tag{1}$$

Page-Block XOR (PBX) [8] This technique exploits the irregular use of the memory address space to produce hash functions with fewer XOR gates. An address is partitioned into two non-overlapping bit-fields, and selected bits of each field are XOR'ed together with the purpose of XOR'ing high entropy bits (from the low-order bit-field) with lower entropy bits (from the high order bit-field). Modifying the previous example, if the address is partitioned into 2 groups of 2 bits, we could produce the following example 2-bit hash: $[a_2 \oplus a_0,\ a_3 \oplus a_1]$.

Locality-sensitive XOR [9] This scheme attempts to reduce hash collisions and hence the probability of false conflicts by exploiting memory reference spatial locality. The key idea is to make nearby memory locations share some of their k hash indices to delay filling the signature. This scheme produces k H_3 functions that progressively omit a larger number of least significant bits of the address from the computation of the k indices. When represented as H_3 binary matrices, functions require an increasing number of lower rows to be null. Our implementation, called LE-PBX, combines this approach with the reduced XOR'ing of PBX hashing. In LE-PBX, we XOR high-entropy bits with low-entropy bits within a window of the address, then shift the window towards the most significant (low entropy) bits for subsequent hash functions.

3 Application-Specific Signatures

All the hashing functions listed in the previous section create a random index that maps to a signature bit range that is a power of two. In Section 5 we demonstrate that these functions require too many bits to be implemented without dramatically slowing down our processor pipelines. To minimize the hardware resources required, the challenge is to reduce the number of false conflicts per signature bit, motivating us to more efficiently utilize signature bits by creating application-specific hash functions.

Our approach is based on *compact trie hashing* [10]. A trie is a tree where each descendant of a node has in common the prefix of most-significant bits associated with that node. The result of the hash of an address is the leaf position found in the tree, corresponding to exactly one signature bit. Because our benchmarks can access up to 64 Mbytes of storage (16 million words), it is not possible to explicitly represent all possible memory locations as a leaf bit of the trie. The challenge is to minimize false conflicts by mapping the most contentious

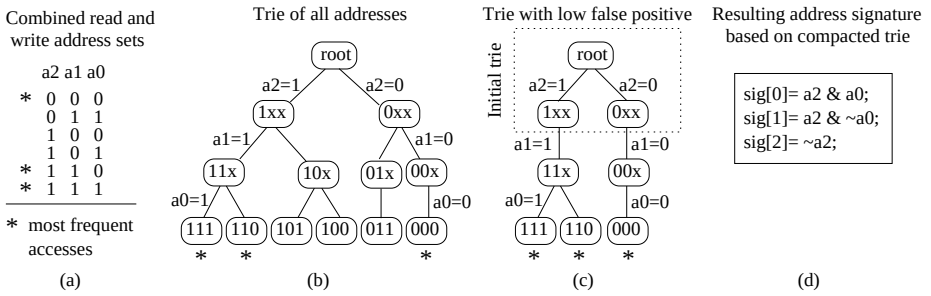


Fig. 1. Example trie-based signature construction for 3-bit addresses. We show (a) a partial address trace, where * highlights frequently accessed addresses, (b) the full trie of all addresses, (c) the initial and final trie after expansion and pruning to minimize false positives, and (d) the logic for computing the signature for a given address (i.e., to be AND'ed with read and write sets to detect a conflict).

memory locations to different signature bits, while minimizing the total number of signature bits.

We use a known greedy algorithm to compute an approximate solution to this NP-complete problem [11]. In the first step, we record in our simulator a trace of the read and write sets of a benchmark functioning at its maximum sustainable packet rate. We organize the collected memory addresses in a trie in which every leaf represents a signature bit. This signature is initially too large to be practical (Figure 1(b)) so we truncate it to an initial trie (Figure 1(c)), selecting the most frequently accessed branches. To reduce the hardware logic to map an address to a signature (Figure 1(d)), only the bits of the address that lead to a branch in the trie are considered. For our signature scheme to be safe, an extra signature bit is added when necessary to handle all addresses not encompassed by the hash function. We then replay the trace of accesses and count false conflicts encountered using our initial hashing function. We iteratively expand the trie with additional branches and leaves to eliminate the most frequently occurring false-positive conflicts (Figure 1(c)). Once the trie is expanded to a desired false positive rate, we greedily remove signature bits that do not negatively impact the false positive rate (they are undesirable by-products of the expansion). Finally, to further minimize the number of signature bits, we combine signature bits that are likely ($> 80\%$) to be set together in non-aborted transactions.

4 Evaluation Infrastructure

Table 1 describes our evaluation infrastructure including the platform and compilation, and how we do timing, validation, and measurement. Due to stringent timing requirements (there are no free PLLs after merging-in the NetFPGA support components), and despite some available area on the FPGA, (i) our caches are limited to 16KB each, and (ii) we are also limited to a maximum of two processors. These limitations are not inherent in our architecture, and would be relaxed in a system with more PLLs and a more modern FPGA. The rest of this section describes our system architecture and benchmark applications.

Aspect	Description
Compilation	Modified `gcc` 4.0.2, `Binutils` 2.16, and `Newlib` 1.14.0
Instruction set	32-bit MIPS-I ISA without delay slots [12], with software mul and div
FPGA	Virtex II Pro 50 speed grade 7ns
Platform	NetFPGA 2.1 [6] with 4 x 1GigE Media Access Controllers (MACs)
Synthesis	Xilinx ISE 10.1.03, high effort to meet timing constraints
Off-chip memory	64 Mbytes 200MHz DDR2 SDRAM, Xilinx MIG controller
Processor clock	125MHz, same as Ethernet MACs
Validation	Execution trace generated in RTL simulation and online in debug mode, compared against cycle-accurate simulator built on MINT [13]
Measuring host	Linux 2.6.18 Dell PowerEdge 2950 with two quad-core 2GHz Xeon processors
Packet source	Modified `Tcpreplay` 3.4.0 sending packet traces from a Broadcom NetXtreme II GigE NIC to an input port of the NetFPGA
Packet sink	NetXtreme GigE NIC connected to another NetFPGA port used for output
Max. throughput	Smallest fixed packet inter-arrival rate without packet drop, obtained through bisection search (we empirically found 5 second runs to be sufficient)

Table 1. Evaluation infrastructure.

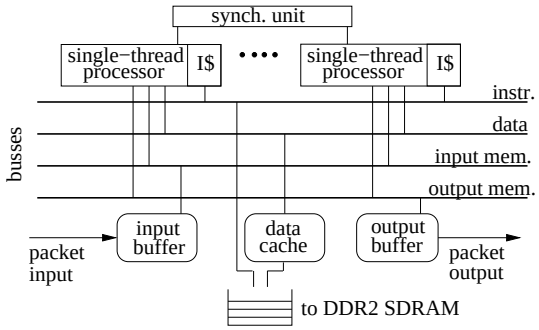


Fig. 2. The architecture of our soft multiprocessor with 2 single-threaded processor cores.

System architecture Our base processor is a single-issue, in-order, single-threaded, 5-stage pipelined processor. To eliminate the critical path for hazard detection logic, we employ static hazard detection [14] in our architecture/compiler. The processor is big-endian which avoids having to do network-to-host byte ordering transformations. Each processor in Figure 2 has a 16 KB private instruction cache. The SDRAM controller services a merged load/store queue of up to 64 entries in-order; since this queue is shared by all processors it serves as a single point of serialization and memory consistency, hence threads need only block on pending loads but not stores. As described in Table 2, our multiprocessor architecture is bus-based and sensitive to the two-port limitation of block RAMs available on FPGAs. In its current form it will not easily scale to a large number of processors. However, as we demonstrate later in Section 5, our applications are mostly limited by synchronization and critical sections, and not contention on the shared buses; in other words, the synchronization inherent in the applications is the primary roadblock to scalability.

Transactional memory support The single port from the processors to the shared cache in Figure 2 implies that memory accesses undergo conflict detection one by one in transactional execution, therefore a single trie hashing unit suffices for both processors. Our transactional memory processor uses a shadow register file to revert its state upon rollback (versioning [16] avoids the need for register

Table 2. On-chip memory hierarchy.

Memory	Description
Input buffer	Receives packets on one port and services processor requests on the other port, read-only, logically divided into ten fixed-sized packet slots
Output buffer	Sends packets to the NetFPGA MAC controllers on one port, connected to the processors via its second port
Data cache	Connected to the processors on one port, 32-bit line-sized data transfers with the DDR2 SDRAM controller (similar to previous work [15]) on the other port
All three	16KB, single-cycle random access, arbitrated across processors, 32 bits bus

Table 3. Applications and their mean statistics.

Benchmark	Description	Dyn. Instr. /packet	Dyn. Instr. /transaction	Uniq. Sync. Addr. /transaction	
				Reads	Writes
`Classifier`	Regular expression matching on TCP packets for application recognition.	2553	1881	67	58
`NAT`	Network address translation plus statistics.	2057	1809	50	41
`UDHCP`	Modified open-source DHCP server.	16116	3265	430	20
`Intruder`	Network intrusion detection [17] modified to have packetized input.	12527	399	37	23

copy). Speculative memory-writes trigger a backup of the overwritten value in an undo-buffer [7] that we over-provision with storage for 2048 values per thread. Each processor has a dedicated connection to a synchronization unit that triggers the beginning and end of speculative executions when synchronization is requested in software.

Applications Network packet processing is no longer limited solely to routing, with many applications that require deeper packet inspection becoming increasingly common and desired. We focus on *stateful* applications—i.e., applications in which shared, persistent data structures are modified during the processing of most packets. Our processors process packets from beginning-to-end by executing the same program, because the synchronization around shared data structures makes it impractical to extract parallelism otherwise (e.g. with a pipeline of balanced execution stages). To take full advantage of the software programmability of our processors, our focus is on the control-flow intensive applications described in Table 3. While we could enforce ordering in software, we allow packets to be processed out-of-order because our application semantics allow it.

5 Results

In this section we first evaluate the impact of signature scheme and length on false-positive conflicts, application throughput, and implementation cost. These results guide the implementation and evaluation of our real system.

Resolution of Signature Mechanisms Using a recorded trace of memory accesses obtained from a cycle-accurate simulation of our TM system that models perfect conflict detection, we can determine the false-positive conflicts that would result from a given realistic signature implementation. We use a recorded trace because the false positive rate of a dynamic system cannot be determined without affecting the course of the benchmark execution: a dynamic system

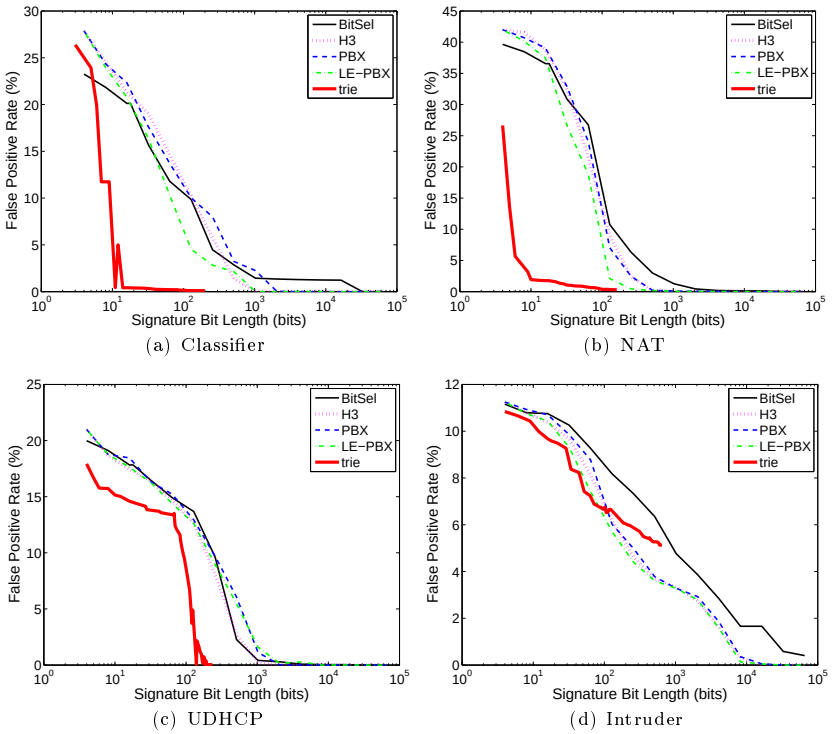
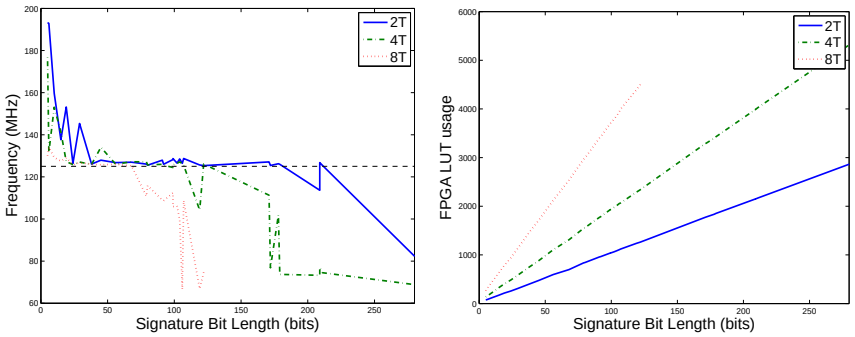


Fig. 3. False positive rate vs signature bit length. Trie-based signatures were extended in length up to the length that provides zero false positives on the training set.

cannot distinguish a false-positive conflict from a later true conflict that would have happened in the same transaction, if it was not aborted immediately. We compute the false positive rate as the number of false conflicts divided by the total number of transactions, including repeats due to rollback.

The signatures that we study are configured as follows. The bit selection scheme selects the least significant word-aligned address bits, to capture the most entropy. For H3, PBX and LE-PBX, we found that increasing the number of hash functions caused a slight increase in the false positive rate for short signatures, but helped reduce the the number of signature bits required to completely eliminate false positives. We empirically found that using four hash functions is a good trade-off between accuracy and complexity, and hence we do so for all results reported. To train our trie-based hash functions, we use a different but similarly-sized trace of memory accesses as a training set.

Figure 3 shows the false positive rate for different hash functions (bit selection, H3, PBX, LE-PBX and trie-based) as signature bit length varies. The false positive rate generally decreases with longer signatures because of the reduced number of collisions on any single signature bit—although small fluctuations are possible due to the randomness of the memory accesses. Our results show that LE-PBX has a slightly lower false positive rate than H3 and PBX for an equal number of signature bits. Bit selection generally requires a

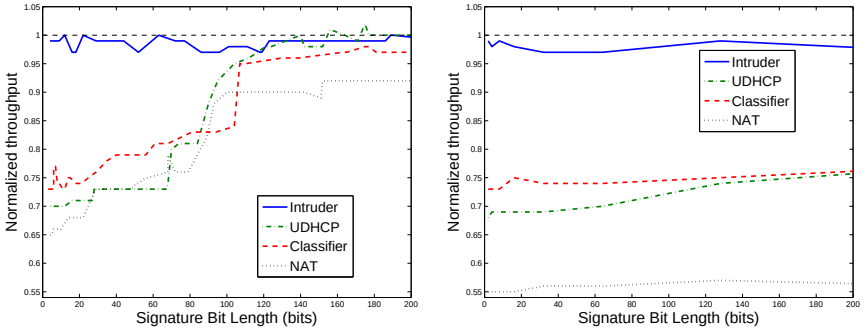


(a) Frequency (b) LUT usage

Fig. 4. Impact of increasing the bit length of trie-based signatures on (a) frequency and (b) LUT usage of the conflict detection unit for 2, 4, and 8-thread (2T,4T,8T) systems. The results for H3, PBX and LE-PBX are similar. In (a) we highlight the system operating frequency of 125MHz.

larger number of signature bits to achieve a low false positive rate, except for UDHCP for which most of the memory accesses point to consecutive statically allocated data. Overall, the trie scheme outperforms the others for CLASSIFIER, NAT and UDHCP by achieving close to zero false positive rate with less than 100 bits, in contrast with several thousand bits. For INTRUDER the non-trie schemes have a better resolution for signatures longer than 100 bits due to the relatively large amount of dynamic memory used, which makes memory accesses more random. Quantitatively we can compute the entropy of accesses as $\sum_{i=0}^{n-1} -p(x_i)\log_2 p(x_i)$ where $p(x_i)$ is the probability of an address appearing at least once in a transaction—with this methodology INTRUDER has an entropy 1.7 times higher on average than the other benchmarks, thus explaining the difficulty in training its trie-based hash function.

Implementation of a Signature Mechanism Figure 4 shows the results of implementing a signature-based conflict detection unit using solely the LUTs in the FPGA for a processor system with 2 threads like the one we implemented (2T) and for hypothetical transactional systems with 4 and 8 threads (4T and 8T). While the plot was made for a trie-based hashing function, we found that H3, PBX and LE-PBX produced similar results. As we will explain later, the bit selection scheme is better suited to a RAM-based implementation. In Figure 4(a) we observe that the CAD tools make an extra effort to meet our 125 MHz required operating frequency by barely achieving it for many designs. In a 2-thread system, two signatures up to 200 bits will meet our 125MHz timing requirement while a 4-thread system can only accommodate four signatures up to 100 bits long. For 8-threads, the maximum number of signature bits allowed at 125MHz is reduced to 50 bits. Figure 4(b) shows that the area requirements grow linearly with the number of bits per signature. In practice for 2-threads at 200 bits, signatures require a considerable amount of resources: approximatively 10% of the LUT usage of the total non-transactional system. When the conflict detection unit is incorporated into the system, we found that its area requirements—by putting more pressure on the routing interconnect of



(a) Trie-based hashing (b) LE-PBX

Fig. 5. Throughput with signatures using trie-based hashing with varying signature sizes normalized to the throughput of an ideal system with perfect conflict detection (obtained using our cycle-accurate simulator).

the FPGA—lowered the maximum number of bits allowable to less than 100 bits for our 2-thread system (Table 4). Re-examining Figure 3, we can see that the trie-based hashing function delivers significantly better performance across all the hashing schemes proposed for less than 100 signature bits.

An alternate method of storing signatures that we evaluate involves mapping an address to a signature bit corresponding to a line in a block RAM. On that line, we store the corresponding read and write signature bit for each thread. To preserve the 125MHz clock rate and our single-cycle conflict detection latency, we found that we could only use one block RAM and that we could only use bit selection to index the block RAM—other hashing schemes could only implement one hash function with one block RAM and would perform worse than bit selection in that configuration. Because the data written is only available on the next clock cycle in a block RAM, we enforce stalls upon read-after-write hazards. Also, to emulate a single-cycle clear operation, we version the read and write sets with a 2-bit counter that is incremented on commit or rollback to distinguish between transactions. If a signature bit remains untouched and therefore preserves its version until a transaction with an aliasing version accesses it (the version wraps over a 2-bit counter), the bit will appear to be set for the current transaction and may lead to more false positives. The version bits are stored on the same block RAM line as their associated signature bits, thus limiting the depth of our 16Kb block RAM to 2048 entries (8-bits wide). Consequently, our best bit selection implementation uses a 11 bit-select of the word-aligned least-significant address bits.

Impact of False Positives on Performance Figure 5 shows the impact on performance in a full-system simulation of a varying signature length, when using either a trie-based hashing function or LE-PBX, the scheme with the second-lowest false positive rate. The jitter in the curves is again explained by the unpredictable rollback penalty and rate of occurrence of the false positives, varying the amount contention on the system. Overall, we can see that signatures have a dramatic impact on system throughput, except for INTRUDER for which

Benchmark	Max. Signature bits	Total LUT usage	LUT overhead	Additional throughput
Classifier	92	20492	5%	12%
NAT	68	20325	4%	58%
UDHCP	84	20378	4%	9%
Intruder	96	20543	5%	71%

Table 4. Size, LUT usage, LUT overhead and throughput gain of our real system with the best application-specific trie-based hash functions over bit selection.

the false positive rate varies little for this signature size range (Figure 3(d)). We observe that for Classifier, UDHCP and NAT, although they achieve a small false positive rate with 10 bits on a static trace of transactional accesses (Figure 3), their performance increases significantly with longer signatures. We found that our zero-packet drop policy to determine the maximum throughput of our benchmarks is very sensitive to the compute-latency of packets since even a small burst of aborts and retries for a particular transaction directly impacts the size of the input queue which in turn determines packet drops. The performance of NAT plateaus at 161 bits because that is the design that achieves zero false positives in training (Figure 3(b)). As expected, Figure 5(b) shows that there is almost no scaling of performance for LE-PBX in the possible signature implementation size range because the false positive rate is very high.

Measured Performance on the Real System As shown in Table 4 and contrarily to the other schemes presented, the size of the trie-based signatures can be adjusted to an arbitrary number of bits to maximize the use of the FPGA fabric while respecting our operating frequency. The maximum signature size is noticeably smaller for NAT because more address bits are tested to set signature bits, which requires more levels of logic and reduces the clock speed. In all cases the conflict detection with a customized signature outperforms the general purpose bit selection. This is coherent with the improved false positive rate observed in Figure 3. We can see that bit selection has the best performance when the data accesses are very regular as in UDHCP, as indicated by the low false positive rate in Figure 3(c). Trie-based hashing improves the performance of Intruder the most because the bit selection scheme suffers from bursts of unnecessary transaction aborts.

CAD Results Comparing two-processor full system hardware designs, the system with trie-based conflict detection implemented in LUTs consumes 161 block RAMs and the application-specific LUT usage reported in Table 4. Block-RAM-based bit selection requires one additional block RAM (out of 232, i.e., 69% of the total capacity) and consumes 19546 LUTs (out of 47232, i.e. 41% of the total capacity). Since both kinds of designs are limited by the operating frequency, trie-based hashing only has an area overhead of 4.5% on average (Table 4). Hence the overall overhead costs of our proposed conflict detection scheme are low and enable significant throughput improvements.

6 Conclusions

In this paper we have studied several previously-proposed signature-based conflict detection schemes for TM. Among those, we found that bit selection

provides the best implementation that avoids (i) degrading the operating frequency of an FPGA-based soft multiprocessor system or (ii) stalling the processors for multiple cycles. We have presented a method for implementing more efficient signatures by customizing them to match the access patterns of an application. Our scheme builds on trie-based hashing, and minimizes the number of false conflicts detected, improving the ability of the system to exploit parallelism. On a real FPGA-based packet processor, we measured packet throughput improvements of 12%, 58%, 9% and 71% for four applications, demonstrating that application-specific signatures are a compelling means to facilitate conflict detection for FPGA-based TM systems.

References

1. S. Wee, J. Casper, N. Njoroge, Y. Tesylar, D. Ge, C. Kozyrakis, and K. Olukotun, "A practical FPGA-based framework for novel CMP research," in *Proc. of FPGA '07*, 2007, pp. 116–125.
2. S. Grinberg and S. Weiss, "Investigation of transactional memory using FPGAs," in *Proc. of EEEI'06*, Nov. 2006, pp. 119–122.
3. C. Kachris and C. Kulkarni, "Configurable transactional memory," in *Proc. of FCCM'07*. IEEE Computer Society, 2007, pp. 65–72.
4. L. Ceze, J. Tuck, J. Torrellas, and C. Cascaval, "Bulk disambiguation of speculative threads in multiprocessors," in *Proc. of ISCA'06*, 2006, pp. 227–238.
5. D. Sanchez, L. Yen, M. D. Hill, and K. Sankaralingam, "Implementing signatures for transactional memory," in *Proc. of MICRO '07*, 2007, pp. 123–133.
6. J. W. Lockwood, N. McKeown, G. Watson, G. Gibb, P. Hartke, J. Naous, R. Raghuraman, and J. Luo, "NetFPGA - an open platform for gigabit-rate network switching and routing," in *Proc. of MSE '07*, June 3-4 2007.
7. L. Yen, J. Bobba, M. R. Marty, K. E. Moore, H. Volos, M. D. Hill, M. M. Swift, and D. A. Wood, "LogTM-SE: Decoupling hardware transactional memory from caches," in *Proc. of HPCA '07*, 2007, pp. 261–272.
8. L. Yen, S. Draper, and M. Hill, "Notary: Hardware techniques to enhance signatures," in *Proc. of Micro'08*, Nov. 2008, pp. 234–245.
9. R. Quislant, E. Gutierrez, and O. Plata, "Improving signatures by locality exploitation for transactional memory," in *Proc. of PACT'09*, 2009, pp. 303–312.
10. E. J. Otoo and S. Effah, "Red-black trie hashing," Carleton University, Tech. Rep. TR-95-03, 1995.
11. S. Khuller, A. Moss, and J. S. Naor, "The budgeted maximum coverage problem," *Inf. Process. Lett.*, vol. 70, no. 1, pp. 39–45, 1999.
12. M. Labrecque, P. Yiannacouras, and J. G. Steffan, "Custom code generation for soft processors," in *Proc. of RAAW '06*, December 2006.
13. J. Veenstra and R. Fowler, "MINT: a front end for efficient simulation of shared-memory multiprocessors," in *Proc. of MASCOTS '94*, Jan. 1994, pp. 201–207.
14. M. Labrecque and J. G. Steffan, "Fast critical sections via thread scheduling for FPGA-based multithreaded processors," in *Proc. of FPL'09*, Sept. 2009.
15. R. Teodorescu and J. Torrellas, "Prototyping architectural support for program rollback using FPGAs," *Proc. of FCCM '05*, pp. 23–32, April 2005.
16. K. Aasaraai and A. Moshovos, "Towards a viable out-of-order soft core: Copy-free, checkpointed register renaming," in *Proc. of FPL*, 2009.
17. C. Cao Minh, J. Chung, C. Kozyrakis, and K. Olukotun, "STAMP: Stanford transactional applications for multi-processing," in *Proc. of IISWC '08*, Sept. 2008.