TMbox: A Flexible and Reconfigurable 16-core Hybrid Transactional Memory System

Nehir Sonmez[*†], Oriol Arcas[*†], Otto Pflucker[*†], Osman S. Unsal[*],
Adrián Cristal[*‡], Ibrahim Hur[*], Satnam Singh[§] and Mateo Valero[*†]
[*] *Barcelona Supercomputing Center (name.surname@bsc.es)*
[†] *Universitat Politècnica de Catalunya* [‡] *CSIC - Spanish National Research Council*
[§] *Microsoft Research Cambridge (satnams@microsoft.com)*

Abstract

In this paper we present the design and implementation of TMbox: An MPSoC built to explore trade-offs in multicore design space and to evaluate parallel programming proposals such as Transactional Memory (TM). Our flexible system, comprised of MIPS R3000-compatible cores is easily modifiable to study different architecture, library and operating system extensions. For this paper we evaluate a 16-core Hybrid Transactional Memory implementation based on the TinySTM-ASF proposal on a Virtex-5 FPGA and we accelerate three benchmarks written to investigate TM.

1. Introduction

A recent alternative for exploring new generations of multicores is based on building a multiprocessor system-on-chip (MPSoC). This approach enables the emulation of large parallel architectures on top of a reconfigurable FPGA platform whose speed and process technology (currently 28 nm) are evolving faster than ASIC. Today's FPGA systems can integrate multiple hard/soft processor cores, multi-ported SRAM blocks, high-speed DSP units, and programmable I/O interfaces with configurable fabric of logic cells.

With the abundance of pre-tested Intellectual Property (IP) cores available, nowadays it is possible to prototype large architectures in a full-system environment which allows for faster and more productive hardware research than software simulation. Over the past decade, the RAMP project has already established a well-accepted community vision and various novel FPGA architecture designs [4], [6], [8], [13], [17], [22]. Another advantage of FPGA emulation over software simulation is the reduced profiling overhead and the possibility for a variety of debugging options.

One direction is to choose a well-known architecture like MIPS and utilize the commonly-available toolchains and library support. Although running a minimal OS might be acceptable, a deeper software stack could have many advantages by providing memory protection, performing scheduling, aiding debugging and file system support. Full OS support can also be accomplished with a nearby host computer which serves system calls and handles exceptions, instead of implementing them in the FPGA model [6].

A proposal that has drawn considerable attention for programming shared-memory Chip Multi-Processors (CMP) has been the use of Transactional Memory (TM), an attractive paradigm for deadlock-free execution of parallel code without using locks. Locks are prone to deadlock or priority inversion while TM provides optimistic concurrency by executing atomic transactions in an all-or-none manner. The programmer encapsulates critical sections inside the `atomic{}` construct and the underlying TM mechanism automatically detects data inconsistencies and aborts and restarts one or more transactions. If there are no inconsistencies, all side effects of a transaction are committed as a whole.

Transactional Memory can be implemented in hardware (HTM) [3], [16], which is fast but resource-bounded while usually requiring changes to the caches and the Instruction Set Architecture (ISA), or software (STM) [9] which can be flexible, run on off-the-shelf hardware, albeit at the expense of lower performance. To have the best of two worlds, there are intermediate Hybrid TM (HyTM) proposals where transactions first attempt to run on hardware, but are backed off to SW when HW resources are exceeded, and Hardware-assisted STM (HaSTM) which aims to accelerate a software-controlled TM implementation by architectural means [7], [2].

Despite the fact that FPGA emulators of many complex architectures of various ISAs have been proposed, only a few of these are on TM research, and only up to a small number of cores. Furthermore, the majority of these proposals are based on proprietary or hard processor cores, which imply rigid pipelines that can prevent an architect from modifying the ISA and the microarchitecture of the system.

In this paper, we present TMbox, a shared-memory CMP prototype with Hybrid TM support. More specifically, our contributions are as follows:

- A description of the first 16-core implementation of a Hybrid TM that is completely modifiable from top to bottom. This implies convenience to study HW/SW tradeoffs in topics like TM.
- We detail on how we construct a multicore with MIPS R3000-compatible cores, interconnect the components in a bi-directional ring with backwards invalidations and adapt the TinySTM-ASF Hybrid TM proposal to our infrastructure.
- Experimental results and performance comparisons of STM, HTM and Hybrid TM on three benchmarks designed to investigate trade-offs in TM. We also discuss the strengths and weaknesses of our approach.

The next section presents the TMbox architecture, Section 3 explains the Hybrid TM implementation, Section 4 discusses the limitations and the results of running three benchmarks on TMbox. Related work is in Section 5 and Section 6 concludes the paper.

2. The TMbox Architecture

The basic processing element of TMbox is the Honeycomb CPU core, a heavily modified and extended version of the Plasma soft core [20]. The synthesizable MIPS R2000-compatible soft processor core Plasma was designed for embedded systems and written in VHDL. It has a configurable 2/3 stage pipeline, a 4 KB direct-mapped write-through L1 cache, and can address up to 64 MB of RAM. It was designed to run at a clock speed of 25 MHz, and it includes UART and Ethernet IP cores. We chose it because it is based on the popular MIPS architecture, it is complete and it has a relatively small area footprint on the FPGA. Such RISC architectures with simpler pipelines are more easily customizable and require fewer FPGA resources compared to a deeply-pipelined superscalar processor, so they are more appropriate to be integrated into a larger multiprocessor SoC.

To effectively upgrade the MIPS R2000-compatible Plasma to our MIPS R3000-compatible Honeycomb, we designed and implemented two coprocessors: CP0 that provides support for virtual memory using a Translation Lookaside Buffer (TLB), and CP1 encapsulating an FPU. We optimized the cores to make better use of the resources on our Virtex-5 FPGAs where it can run at twice the frequency (50 MHz); we modified the memory architecture to enable virtual memory addressing for 4 GB and caches of 8 KB; we implemented extra instructions to better support exceptions and thread synchronization (load-linked and store conditional) and we developed system libraries for memory allocation, I/O and string functions [21]. The Honeycomb core (without an FPU and the DDR controller) occupies 5827 LUTs (Table 1) on a Virtex-5 FPGA including the ALU, MULT/DIV and Shifter units, the coherent L1 cache and the UART controller, a comparable size to the Microblaze core.

The Virtex5-155T FPGA contains 98K LUTs, 212 BRAMs, and 128 DSP blocks. The DDR2 controller that occupies a small portion of the FPGA (around 2%) performs calibration and serves requests [23]. Using one controller provides sequential consistency for our multicore since there is only one address bus, and loads are blocking and stall the processor pipeline.

2.1. Interconnection

To interconnect the cores, we designed and implemented a bi-directional ring as shown in Figure 1. Arranging the components on a ring rather than a bus requires shorter wires which eases placement on the chip, relaxing constraints, and is a simple and efficient design choice to diminish the complexities that arise in implementing a large crossbar on FPGA fabric. Apart from increased place and route time, longer wires would lead to more capacitance, longer delay and higher dynamic power dissipation. Using a ring will also enable easily adding and removing shared components such as an FPU or any application-specific module, however this is out of the scope of this work.

CPU requests move counterclockwise; they go from the cores to the bus controller, eg. CPU_i - CPU_{i-1} - ... - CPU_0 - Bus Ctrl. Requests may be in form of read or write, carrying a type field, a 32-bit address, a CPU ID and a 128-bit data field, which is the data word size in our system. Memory responses also move in the same direction; from the bus controller to the cores, eg. Bus Ctrl - CPU_n - CPU_{n-1} - ... - CPU_{i+1} - CPU_i. They use the same channel as requests, carrying responses to the read requests served by the DDR Ctrl. On the other hand, moving clockwise are backwards invalidations caused by the writes to memory which move from the Bus Ctrl towards the cores in the opposite direction, eg. Bus Ctrl - CPU_0 - ... - CPU_{i-1} - CPU_i. These

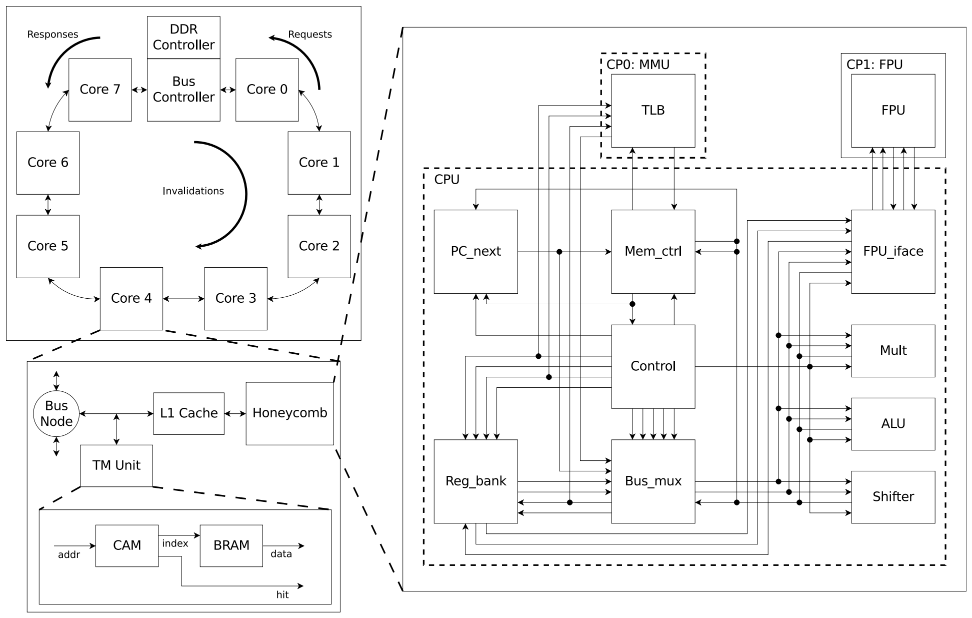

Figure 1. An 8-core TMbox infrastructure showing the ring bus, the TM Unit and the processor core.

Table 1. LUT occupation of components

Component	5-LUTs	Component	5-LUTs
PC_next	138	Mem_ctrl	156
Control	139	Reg_File	147
Bus_mux	155	ALU	157
Shifter	201	MULT	497
Pipeline	112	Cache	1985
TLB	202	TM_unit	1242
Bus_node	619	DDR_ctrl	1119
UART	77	**TOTAL**	**6946**

carry only a 32-bit address and a CPU ID field. When a write request meets an invalidation to the same address on any node, it gets cancelled. Moreover, the caches on each core snoop and discard the lines corresponding to the invalidation address. We detail how we extend this protocol for supporting HTM in the next section.

3. Hybrid TM Support for TMbox

TinySTM [9] is a lightweight and efficient word-based STM library implementation in C and C++. It differentiates from other STMs such as TL2 and Intel STM mainly by its time-based algorithm and lock-based design. By default, it compiles and runs on 32 or 64-bit x86 architectures, using the atomic_ops library to implement atomic operations, which we modified to include Compare and Swap (CAS) and Fetch and Add (FAA) primitives for the MIPS architecture using load-linked and store conditional (LL/SC) instructions. TinySTM-ASF is a hybrid port that enables TinySTM to be used with AMD's HTM proposal, ASF [5], which we modified to work with TMbox. This version starts the transactions in hardware mode and jumps to software if (i) hardware capacity is exceeded, (ii) there is too much contention or (iii) the application explicitly requires it. Our hardware design closely follows the ASF proposal with the exception of nesting support.

A new processor model (-march=honeycomb) was added by modifying GCC and GAS (the GNU Assembler). This new ISA includes all the R3000 instructions plus RFE (Return from Exception), LL, SC and the transactional instructions in Figure 2. All GNU tools (GAS, ld, objdump) were modified to work with these new instructions.

To enable hardware transactions, we extended our design with a per-core TM Unit that contains a transactional cache that only admits transactional loads and stores. By default it has a capacity of 16 data lines (256 bytes). If the TM cache capacity is exceeded, the transaction aborts and sets the TM register $TM2 to ABORT_FULL (explained in the next section) after which the transaction reverts to software and restarts.

A transactional LD/ST causes a cache line to be written to the TM Unit. An invalidation of any of the lines in the TM Unit causes the current transaction to be aborted. Modifications made to the transactional lines are not sent to memory until the whole transaction successfully commits. The TM Unit provides single-cycle operations on the transactional read/writeset stored inside. A Content Addressable Memory (CAM)

Table 2. HTM instructions for TMbox

Instruction	Description
`XBEGIN (addr)`	Starts a transaction and saves the abort address (addr) in TM register $TM0. Also saves the contents of the $sp (stack pointer) to TM register $TM1.
`XCOMMIT`	Commits a transaction. If it succeeds, it continues execution. If it fails, it rolls back the transaction, sets TM register $TM2 to ABORT_CONFLICT, restores the $sp register and jumps to the abort address.
`XABORT (20-bit code)`	Used by software to explicitly abort the transaction. Sets TM register $TM2 to ABORT_SOFTWARE, restores the $sp register and jumps to the abort address. The 20-bit code is stored in the TM register $TM3.
`XLB, XLH, XLW, XSB, XSH, XSW`	Transactional load/store of bytes, halfwords (2 bytes) or words (4 bytes).
`MFTM (reg), (TM_reg)`	Move From TM: Reads from a TM register and writes to a general purpose register.

is built using LUTs both to enable asynchronous reads and since BRAM-based CAMs grow superlinearly in resources. Two BRAMs store the data that is accessed by an index provided by the CAM. Additionally, the TM Unit can serve LD/ST requests on an L1 miss if the line is found on the TM cache.

3.1. Instruction Set Architecture Extensions

To support HTM, we augmented the MIPS R3000 ISA with the new transactional instructions listed in Table 2. We have also extended the register file with four new transactional registers, which can only be read with the MFTM (move from TM) instruction. $TM0 register contains the abort address, $TM1 has a copy of the stack pointer for restoring when a transaction is restarted, $TM2 contains the bit field for the abort (overflow, contention or explicit) and $TM3 stores a 20-bit abort code that is provided by TinySTM, eg. abort due to malloc/syscall/interrupt inside a transaction, or maximum number of retries reached etc.

Aborts in TMbox are processed like an interrupt, but they do not cause any traps, instead they jump to the abort address and restore the $sp (stack pointer) in order to restart the transactions. Regular loads and stores should not be used with addresses previously accessed in transactional mode, therefore it is left to the software to provide isolation of transactional data if desired. LL/SC can be used simultaneously with TM provided that they do not access the same address.

Figure 2 shows an atomic increment in TMbox MIPS assembly. In this simple example, the abort code is responsable for checking if the transaction has been retried a maximum number of times, and if there is a hardware overflow (the TM cache is full), and in this case jumps to an error handling code (not shown).

3.2. Bus Extensions

To support HTM, we added a new type of request, namely COMMIT_REQ, and a new response type, LOCK_BUS. When a commit request arrives to the DDR, it causes a backwards LOCK_BUS message on the ring which destroys any incoming write requests from the opposite direction, and locks the bus to grant exclusive access to perform a serialized commit action. All writes are then committed through the "channel" created, after which the bus is unlocked with another LOCK_BUS message, resuming normal operation. More efficient schemes can be supported in the future to enable parallel commits [3].

```
  LI    $11, 5         //set max. retries = 5
  LI    $13, HW_OFLOW  //reg 13 has err. code
  J     $TX

$ABORT:
  MFTM  $12, $TM2      //check error code
  BEQ   $12, $13, $ERR //jump if HW overflow
  ADDIU $10, $10, 1    //retries++
  SLTU  $12, $10, $11  //max. retries?
  BEQZ  $12, $ERR2     //jump if max. retries

$TX:
  XBEGIN($ABORT)      //provide abort address
  XLW   $8, 0($a0)    //transactional LD word
  ADDi  $8, $8, 1     //a++
  XSW   $8, 0($a0)    //transactional ST word
  XCOMMIT             //if abort go to $ABORT
```

Figure 2. TMbox MIPS assembly for `atomic{a++}` (NOPs and branch delay slots are not included).

3.3. Cache Extensions

The cache state machine reuses the same hardware for transactional and non-transactional loads and stores, however a transactional bit dictates if the line should go to the TM cache or not. Apart from regular cached RD/WR, uncached accesses are also supported, as shown in Figure 3. Cache misses first make a memory read request to bring the line and wait in WaitMemRD state. In case of a store, the WRback and WaitMemWR states manage the memory write

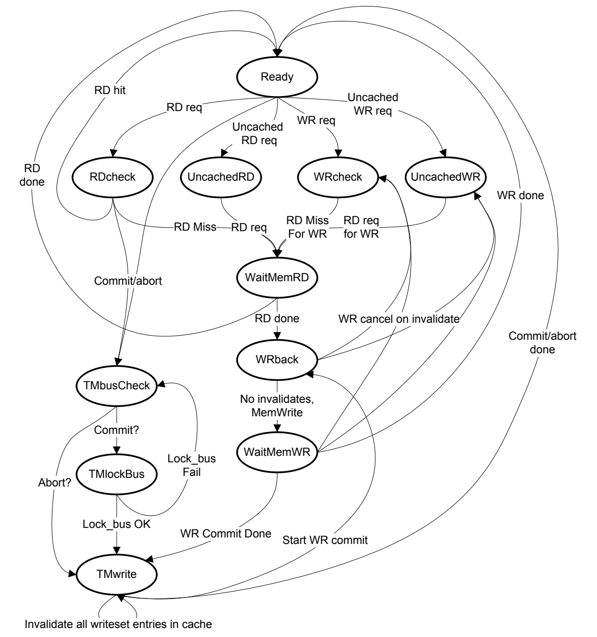

Figure 3. Cache state diagram. Some transitions (LL/SC) are not shown for visibility.

operations. While in these two states, if an invalidation arrives to the same address, the write will be cancelled. In case of a store-conditional instruction, the write will not be re-issued, and the LL/SC will have failed. Otherwise, the cache FSM will re-issue the write after such a write-cancellation on invalidation.

While processing a transactional store inside of an atomic block, an incoming invalidation to the same address causes an abort and possibly the restart of the transaction. Currently our HTM system supports lazy version management: the memory is updated at commit-time at the end of transactions, as opposed to having in-place updates and keeping an undo log for aborting. We also provide lazy conflict detection which implies that data inconsistencies are detected only after the speculative data is committed to the memory. Each transactional write successfully committed causes an invalidation signal, which aborts the transactions that already have those lines in the TM cache. So a transaction can only be aborted due to data conflicts during transaction execution (between XBEGIN and XCOMMIT/XABORT).

To support HTM, the cache state machine is extended with three new states, TMbusCheck, TMlockBus and TMwrite. One added functionality is to dictate the locking of the bus prior to committing. Another duty is performing burst writes in case of a successful commit which runs through the TMwrite-WRback-WaitMemWR-TMwrite loop. The TMwrite state is also responsible for the gang clearing of all entries in the TM cache and the writeset entries that are also found in L1 cache after a commit/abort. To enable this, address entries that are read from the TM Unit are sent to L1 cache as invalidation requests, after which the TM cache is cleared in preparation for a new transaction.

4. Experimental Evaluation

TMbox can fit 16 cores in a Virtex-5 FPGA, occupying 86,797 LUTs (95% of total slices) and 105 BRAMs (49%). In this section, we first examine the trade-offs of our implementation, we then discuss the results of three TM benchmarks.

4.1. Architectural Benefits and Drawbacks

On the TM side, the performance of our best-effort Hybrid TM is bounded by the size of the transactional cache of the TM unit. Although for this work we chose to use a small, 16-entry TM cache, larger caches can certainly be supported on the TMbox on larger FPGAs (keeping in mind the extra area overhead introduced).

In pure HTM mode, all 16 lines of the TM cache can be used for running the transaction in hardware, however the benchmark can not run to completion if there are larger transactions that do not fit in the TM cache, since there is no hardware or software mechanism to decide what to do in this case. The largest overhead related to STM is due to keeping track of transactional loads and stores in software. The situation can worsen when the transactions are large and there are many aborts in the system.

In Hybrid TM mode it is desired to commit as many transactions as possible on dedicated hardware, however when this is not possible, it is also important to provide an alternative path using software mechanisms. All transactions that overflow the TM cache will be restarted in software, implying all work done in hardware TM mode to be wasted in the end. Furthermore to enable hybrid execution, TinySTM-ASF additionally keeps the lock variables inside the TM cache. This results in allowing a maximum of 8 variables in the read/writesets of each transaction as opposed to 16 for pure HTM. Of course this is true provided that neither the transactional variables, nor the lock variables share a cache line, in which case, in some executions we observed some transactions having a read/writeset of 9 or 10 entries successfully committing in hardware TM mode.

On the network side, the ring is an FPGA-friendly option: we have reduced the place and route time of an 8-core design to less than an hour using the ring

Table 3. TM Benchmarks Used

TM Benchmark	Description
`Eigenbench[11]`	Highly tunable microbenchmark for TM with orthogonal characteristics. We have used this benchmark (2000 loops) with (i) r1=8, w1=2 to overflow the TM cache and vary contention (by changing the parameters a1 and a2) from 0–28%, and (ii) r1=4 and w1=4 to fit in the TM cache and vary the contention between 0–35%.
`Intruder[15]`	Network intrusion detection. A high abort rate benchmark, contains many transactions dequeuing elements from a single queue. We have used this benchmark with 128 attacks.
`SSCA2[15]`	An efficient and scalable graph kernel construction algorithm. We have used problem scale = 12

network, whereas it took more than two hours using a shared crossbar for interconnection and we could not fit more than 8 cores. However, each memory request has to travel as many cycles as the total number of nodes on the ring plus the DDR2 latency, during which the CPU is stalled. This is clearly a system bottleneck: using write-back caches or relaxed memory consistency models might be key in reducing the number of messages that travel on the ring to improve system performance.

On the processor side, the shallow pipeline negatively affects the operating frequency of the CPU. Furthermore larger L1 caches can not fit on our FPGA, however they could be supported on larger, newer generation FPGAs, which would help the system to better exploit locality. Having separate caches for instructions and data would also be a profitable enhancement.

4.2. Experimental Results

Eigenbench is a synthetic benchmark that can be tuned to discover TM bottlenecks. As Figure 4 shows, the transactions in EigenBench with 2R+8W variables overflow (since TinySTM-ASF keeps the lock variables in the transactional cache) and get restarted in software, exhibiting worse performance than STM. However, the 4 read-4 write variable version fits in the cache and shows a clear improvement over STM.

In the SSCA2 results presented in Figure 5, we get an 1-8% improvement over STM because this benchmark contains small transactions that fit in the transactional cache. Although Intruder (Figure 6) is a benchmark that is frequently used for TM, it is not a TM-friendly benchmark, causing a high rate of aborts and non-scalable performance. However, especially with 16-cores, our scheme achieves in (i) discovering conflicts early and (ii) committing 48.7% of the total transactions in hardware, which results in almost 5x superior performance compared to direct-update STM, which has to undo all changes on each abort. We were unable to run this benchmark on pure HTM because it contains memory operations like malloc/free inside transactions that are complex to run under HTM and are not supported yet on TMbox.

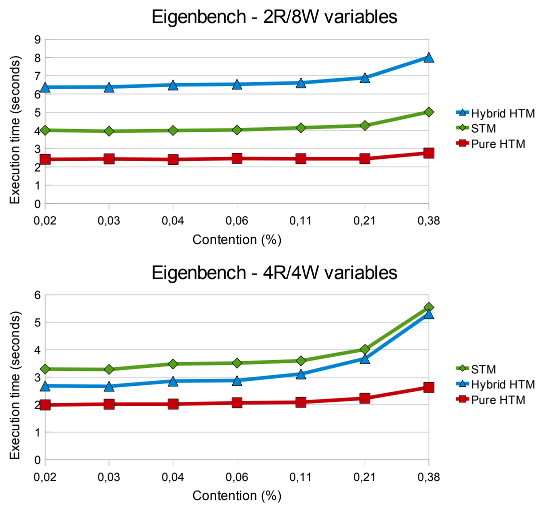

Figure 4. Eigenbench results (16 cores).

These three benchmarks can benefit from our hybrid scheme because they do not run very large transactions, so most of the fallbacks to software caused are due to repeated aborts or mallocs inside transactions. For SSCA2, we see good scalability for up to 8 cores, and for Intruder for up to 4 cores. The performance degradations in STM for Intruder are caused by the fact that the STM directly updates the memory and as the abort rates increase, its performance drastically decreases. Furthermore the system performance is benchmark-dependent: compared to sequential versions, the TM versions can perform in the range of 0.2x (Intruder) to 2.4x (SSCA2). We will be looking more into overcoming the limitations of the ring bus, improving on the TM implementation (serialized commits) and the coherency mechanism.

5. Related Work

Few mostly initial work has been published in the context of studying Transactional Memory on FPGA prototypes. ATLAS is the first full-system prototype of an 8-way CMP system with PowerPC hard processor cores, buffers for read/write sets and per-CPU caches

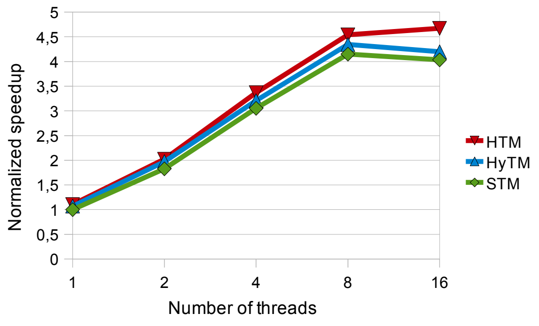

Figure 5. SSCA2 benchmark results.

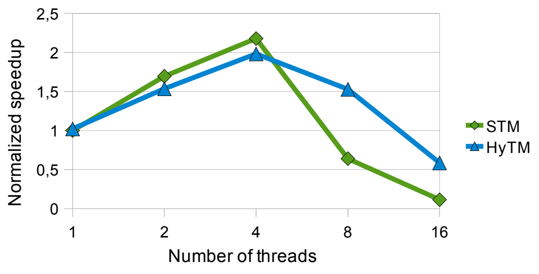

Figure 6. Intruder benchmark results.

augmented with transactional read-write bits and TCC-type HTM support, with a ninth core for running Linux and serving OS requests from other cores [17].

Kachris and Kulkarni describe a TM implementation for embedded systems which can work without caches, using a central transactional controller on four Microblaze cores[12]. TM is used as a simple synchronization mechanism that can be used with higher level CAD tools like EDK for non-cache coherent embedded MPSoC. The proposal occupies a small area on chip, but it is a centralized solution that would not scale as we move up to tens of cores. Similarly, the compact TM proposal, composed by off-the-shelf cores with a software API managing transactions, can be useful for early validation of programs to TM [19].

Recent work that also utilizes MIPS soft cores focuses on the design of the conflict detection mechanism that uses Bloom filters for an FPGA-based HTM [14]. Application-specific signatures are compared to detect conflicts in a single pipeline stage. The design takes little area, reducing false conflicts. The underlying bit-level parallelism used for signatures makes this approach a good match for FPGAs. This proposal was the first soft core prototype with HTM albeit only with 2 cores; it is not clear what is done in case of overflow or how the design would scale. Another approach that uses Bloom filters on FPGAs to accelerate STMs on commodity cores was presented by Casper et al. [2].

Ferri et al. proposed an energy-efficient HTM on a cycle-accurate SW simulator, where transactions can overflow to a nearby victim cache [10]. It is a realistic system with cache coherence, and non-centralized TM support, running a wide range of benchmarks on various configurations, however bus-based snoopy protocol would not scale with more cores, the simulator is not scalable and would suffer from modelling larger numbers of processors, and no ISA changes are possible to the ARM hard CPU core.

Recently, an HTM was proposed by C. Thacker for the Beehive system [24]. In case of overflow the entire transaction is run under a commit lock without using the transactional hardware. We believe that software transactions might have more to offer. The Beehive design also uses a uni-directional ring where messages are added to the head of a train with the locomotive at the end [24]. Ring networks are suggested as a better architecture for shared memory multiprocessors by Barroso et al. [1] and a cache coherent bi-directional ring was presented by Oi et al. [18], but as far as we know, using backwards-propagating write-destructive invalidations is a novel approach. Unlike some of the proposals above, our system features a large number of processors and is completely modifiable which enables investigating different interconnects, ISA extensions or coherency mechanisms.

6. Conclusions

We have presented a Hybrid TM design, where we fit 16 cores on an FPGA providing hardware support and accelerating a modern TM implementation running benchmarks that are widely used in TM research.

The results agree with our insights and findings from other works [15]: Hybrid TM works well when hardware resources are sufficient, providing better performance than software TM. However, when hardware resources are exceeded, the performance can fall below the pure software scheme in certain benchmarks. The good news is that Hybrid TM is flexible; a smart implementation should be able to decide what is best by dynamic profiling. We believe that this is a good direction for further research.

We have also shown that a ring network fits well on FPGA fabric and using smaller cores can help building larger prototypes. Newer generations of FPGAs will continue to present multicore researchers with interesting possibilities, having become so mature as to permit investigating credible largescale systems architecture. We are looking forward to extending the TMbox with a memory directory and to use multiple-FPGAs.

Acknowledgements

We would like to thank Adrià Armejach, Miquel Pericàs and all anonymous reviewers for their comments and valuable feedback. This work is supported by the cooperation agreement between the Barcelona Supercomputing Center and Microsoft Research, by the Ministry of Science and Technology of Spain and the European Union (FEDER funds) under contract TIN2007-60625 and TIN2008-02055-E, by the European Network of Excellence on High-Performance Embedded Architecture and Compilation (HiPEAC) and by the European Commission FP7 project VELOX (216852). The TMbox is available at http://www.velox-project.eu/releases.

References

[1] L. A. Barroso and M. Dubois. Cache coherence on a slotted ring. In *International Conference on Parallel Processing*, 1991.

[2] J. Casper, T. Oguntebi, S. Hong, N. G. Bronson, C. Kozyrakis, and K. Olukotun. Hardware acceleration of transactional memory on commodity systems. *ASPLOS '11*, pages 27–38, 2011.

[3] H. Chafi, J. Casper, B. D. Carlstrom, A. McDonald, C. C. Minh, W. Baek, C. Kozyrakis, and K. Olukotun. A scalable, non-blocking approach to transactional memory. *HPCA '07*, pages 97–108, 2007.

[4] D. Chiou, H. Sunjeliwala, H. Sunwoo, J. D. Xu, and N. Patil. FPGA-based Fast, Cycle-Accurate, Full-System Simulators. *Number UTFAST-2006-01*, 15(5):795–825, November Austin, TX, 2006.

[5] D. Christie, J.-W. Chung, S. Diestelhorst, M. Hohmuth, M. Pohlack, C. Fetzer, M. Nowack, T. Riegel, P. Felber, P. Marlier, and E. Rivière. Evaluation of AMD's advanced synchronization facility within a complete transactional memory stack. In *EuroSys '10*, pages 27–40, 2010.

[6] E. S. Chung, E. Nurvitadhi, J. C. Hoe, B. Falsafi, and K. Mai. A complexity-effective architecture for accelerating full-system multiprocessor simulations using FPGAs. In *FPGA '08*, pages 77–86, 2008.

[7] P. Damron, A. Fedorova, Y. Lev, V. Luchangco, M. Moir, and D. Nussbaum. Hybrid transactional memory. *ASPLOS '06*, 2006.

[8] N. Dave, M. Pellauer, and J. Emer. Implementing a functional/timing partitioned microprocessor simulator with an FPGA. *WARFP*, 2006.

[9] P. Felber, C. Fetzer, and T. Riegel. Dynamic performance tuning of word-based software transactional memory. In *PPoPP*, pages 237–246, 2008.

[10] 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. Parallel Distrib. Comput.*, 70:1042–1052, October 2010.

[11] S. Hong, T. Oguntebi, J. Casper, N. Bronson, C. Kozyrakis, and K. Olukotun. EigenBench: A simple exploration tool for orthogonal TM characteristics. In *IISWC'10*, 2010.

[12] C. Kachris and C. Kulkarni. Configurable transactional memory. In *FCCM '07*, pages 65–72, April 2007.

[13] A. Krasnov, A. Schultz, J. Wawrzynek, G. Gibeling, and P. yves Droz. RAMP Blue: A message-passing manycore system in FPGAs. In *FPL 2007*, pages 27–29, 2007.

[14] M. Labrecque, M. Jeffrey, and J. Steffan. Application-specific signatures for transactional memory in soft processors. In *ARC 2010*, pages 42–54. 2010.

[15] C. C. Minh, J. W. Chung, C. Kozyrakis, and K. Olukotun. STAMP: Stanford transactional applications for multi-processing. In *IISWC*, 2008.

[16] K. E. Moore, J. Bobba, M. J. Moravan, M. D. Hill, and D. A. Wood. LogTM: Log-based transactional memory. In *HPCA 2006*, pages 254–265, 2006.

[17] N. Njoroge, J. Casper, S. Wee, Y. Teslyar, D. Ge, C. Kozyrakis, and K. Olukotun. ATLAS: A chip-multiprocessor with TM support. In *DATE'07*, pages 3–8, 2007.

[18] H. Oi and N. Ranganathan. A cache coherence protocol for the bidirectional ring based multiprocessor. In *PDCS'99*, pages 3–6, 1999.

[19] M. Pusceddu, S. Ceccolini, G. Palermo, D. Sciuto, and A. Tumeo. A compact TM multiprocessor system on FPGA. *FPL'10*, pages 578–581, 2010.

[20] S. Rhoads. Plasma soft core. http://opencores.org/project,plasma.

[21] N. Sonmez, O. Arcas, G. Sayilar, O. S. Unsal, A. Cristal, I. Hur, S. Singh, and M. Valero. From Plasma to BeeFarm: Design experience of an FPGA-based multicore prototype. In *ARC'11*, March 23-25 2011.

[22] Z. Tan, A. Waterman, R. Avizienis, Y. Lee, H. Cook, D. Patterson, and K. Asanović. RAMP gold: An FPGA-based architecture simulator for multiprocessors. In *DAC '10*, pages 463 – 468, 2010.

[23] C. Thacker. A DDR2 controller for BEE3. Microsoft Research, 2009.

[24] C. Thacker. Hardware Transactional Memory for Beehive. In *http://research.microsoft.com/en-us/um/people/birrell/beehive/hardware transactional memory for beehive3.pdf*. MSR Silicon Valley, 2010.