Scientific Report for STSM project "Enhancing an HTM system (TMbox) with HW monitoring capabilities"

Philipp Kirchhofer
philipp.kirchhofer@student.kit.edu

August 30, 2011

Reference Number	COST-STSM-IC1001-8420
Grant Number	COST-STSM-ECOST-STSM-IC1001-200611-008435
COST Action	IC1001
Action Description	Transactional Memories: Foundations, Algorithms, Tools, and Applications (Euro-TM)
STSM Period	2011-06-20 to 2011-07-31
Participant	Philipp Kirchhofer Chair of Prof. Dr. Wolfgang Karl Computer Architecture and Parallel Processing Institute of Computer Science & Engineering (ITEC) Karlsruhe Institute of Technology (KIT)
Host	Dr. Osman Unsal Department for Computer Architecture for Parallel Paradigms Barcelona Supercomputing Center

1 Purpose of the STSM

Transactional Memory (TM) is supposed to simplify parallel programming. Recent research shows that current state-of-the-art TM implementations are on a good way to achieve this goal. Previous work [1] shows that programming with TM semantics exhibits a much smaller error rate compared to programming with traditional fine-grained explicit locking.

But another issue remains: Performance and scalability are both important for a successful adoption of TM. The 90/10 law, originating from software engineering, states that about 90% of application runtime is spent in 10% of code. It is, according to this law, important to know which bottlenecks exist in a given environment to allow the development of scalable and fast applications.

However, in current HTM systems the programmer is often unaware of the application's behavior which makes the optimization a trial-and-error process. As a consequence the TM applications do not run as efficiently as possible. In STM systems generating event logs at run time or profiling on a per-atomic block level are methods that capture and preserve the dependencies between transactions. Typically these techniques come with software overhead and may influence the application runtime characteristics.

The purpose of this short term scientific mission (STSM) was to address these shortcomings and develop a monitoring infrastructure for an HTM system. This will allow the programmer to get insights into the interaction between application and HTM system, to detect bottlenecks during analysis and to optimize the application for the underlying system. Furthermore the knowledge gained can help to develop new designs leading to faster and more efficient HTM systems. The key design aspects include multi-core-scalability, high extensibility, zero runtime overhead and no influence on application runtime characteristics. The system should also be easily usable by an application developer.

2 Description of the work carried out during the STSM

In order to get a complete overview of HTM behavior it is vital to have a system with no impact on application runtime characteristics and application behavior. The optimization hints gathered could otherwise be influenced in some sort and cause a misguided optimization attempt. For HTM systems a separate hardware monitor is therefore the method of choice to non-intrusively gather and preserve run time information.

The TMbox system [2] was used as the base implementation of an HTM system for this project. It uses two ring-buses to connect the soft-core processors. The ring-bus is especially suited to connect one or more monitoring elements. Moreover, the TMbox makes use of FPGAs and thus offers the space and the flexibility to add and synthesize new hardware components. TMbox runs on the BEE3 [3] multi-FPGA research platform equipped with Xilinx Virtex 5 Series FPGAs.

During this STSM an event-based logging system was designed, tested and implemented for the TMbox HTM subsystem. The TMbox system allows up to 16 MIPS-compatible computation cores to be fitted on one FPGA chip. A design decision was to try to preserve this feature by creating a system with a low overhead in terms of look-up-tables and flip-flops used on the FPGA. In combination with the monitoring infrastructure presented here the design, analysis and optimization of future many-core architectures is accordingly possible. The extensibility of the designed system enables future projects to easily implement additional monitoring techniques.

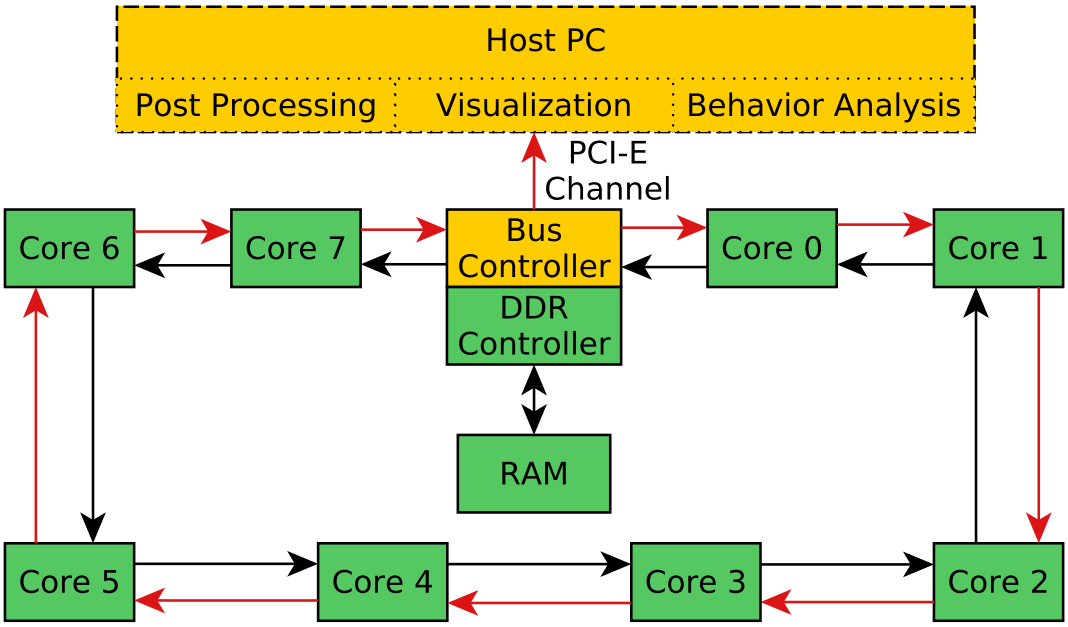

Figure 1: 8 Core TMbox System Block Diagram

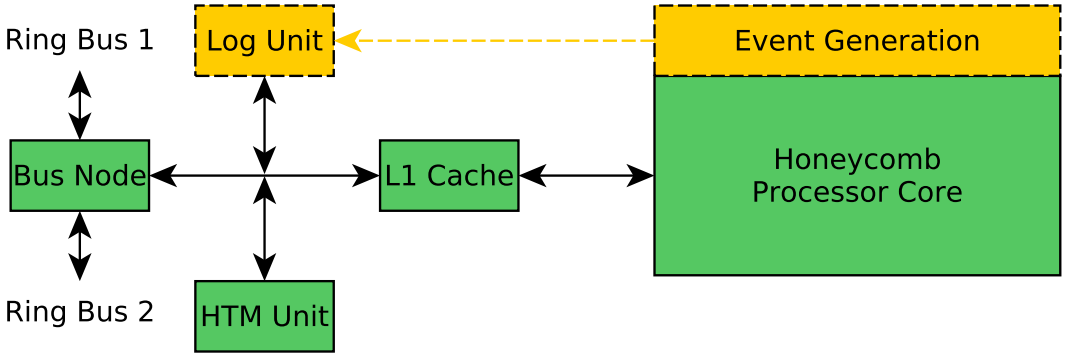

Figure 2: TMbox Core Unit

Figures 1 and 2 show the modified hardware components of the TMbox system in orange color and the added components additionally with dashed lines. The black ring bus in Figure 1 transfers memory read/write requests and responses while the red ring bus transfers invalidation and event messages.

HTM and application behavior can be decomposed into a stream of small events containing information about state changes. The events are later re-composed during post-processing. This design allows running the monitoring infrastructure with low transfer bandwidth needs. The events can be sent one by one during application phases with no invalidation ring bus activity. This does not adversely influence the application behavior and runtime characteristics. An alternative would be to collect and send the whole HTM state each time it changes, causing a large amount of data to be transfered and a consumption of more bandwidth than the chosen approach.

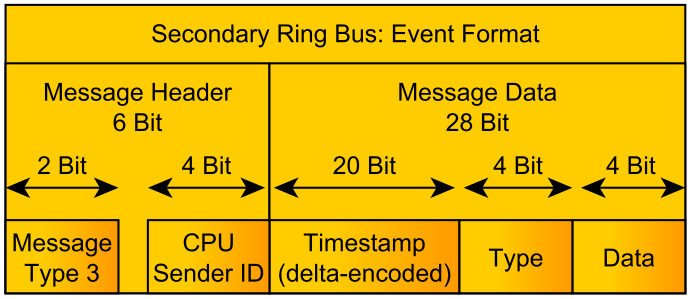

Figure 3: Event Diagram

Figure 3 shows the format of an event in a detailed way. The division into two parts, message header and message data, is given by the fixed format of the secondary (invalidation) ring bus. The special message type 3 distinguishes event messages from already defined invalidation messages. The timestamp, e.g. the time when an event occurred, is delta-encoded. This means that only the difference between consecutive event timestamps is saved. This space efficient encoding allows a temporal space of about $2^{20} = 1$Mio. cycles between two events occurring on one processor. The data field stores additional data available for an event, for instance the cause of an abort (software induced, capacity, invalidation).

Some of the event types defined for transactions include:

- Start
- Commit
- Abort
- Invalidation
- Try locking ring bus for commit
- Succeeded locking ring bus

The generation and capture of these events allows to rebuild the HTM state during post-processing. Additionally subsets of events can be selected later during analysis, allowing a focus on specific types of transactions (for example to analyse only committed transactions). Currently up to 16 different event types can be defined, allowing an easy addition of new event types in future projects. (See also section 7.3)

The following paragraphs describe some of the units created or modified during this STSM projects:

The event generation unit monitors the TM unit state, generating events whenever the state changes. The generated events cover all state changes possible during runtime. They are augmented with additional data that is useful later on for behavior analysis.

A log unit captures events sent by the event generation unit located in the processor core. These events are timestamped and saved using delta encoding in memory blocks located in each core unit. The events are later transferred via the secondary ring bus (invalidation bus) to the bus controller utilizing idle phases.

The bus controller unit gathers all events and transfers them over a PCI Express channel to a host PC. After the supervised application has finished a post processing program (BusEvent-Converter) reads the event stream, rebuilds HTM and application states and generates statistics suitable for later processing with Paraver [4]. The mature and scalable program Paraver was originally designed for the processing of MPI program traces, but in this project it was adapted to analyze and visualize HTM behavior.

Additionally a hardware unit capturing read- and write-sets of transactions running in HTM mode has been designed and tested during the STSM. Future work may integrate this unit into the TMbox system and allow the analysis of an application on a fine-grained statement-by-statement level.

3 Description of the main results obtained

An implementation with no overhead on application runtime or change in application timing characteristics has been achieved by this STSM. With this new infrastructure in place a complete cycle accurate overview of the HTM behavior of an application running on several processor cores can be obtained and analyzed. Some of the available metrics include contention/abort rate, contention between specific threads, time spent in committed and aborted transactions, HTM system overhead and overall scalability.

In addition a visual depiction of the runtime behavior of a TM program can be shown, enabling the viewer to easily identify application parts with different characteristics and to detect parts with sub-optimal behavior. This allows to specifically optimize the poorly performing parts for the underlying HTM system.

Note: The following two figures 4 and 5 have been created using the post processing tool BusEventConverter and the visualization and analysis tool Paraver. They only show a small selection of the views available.

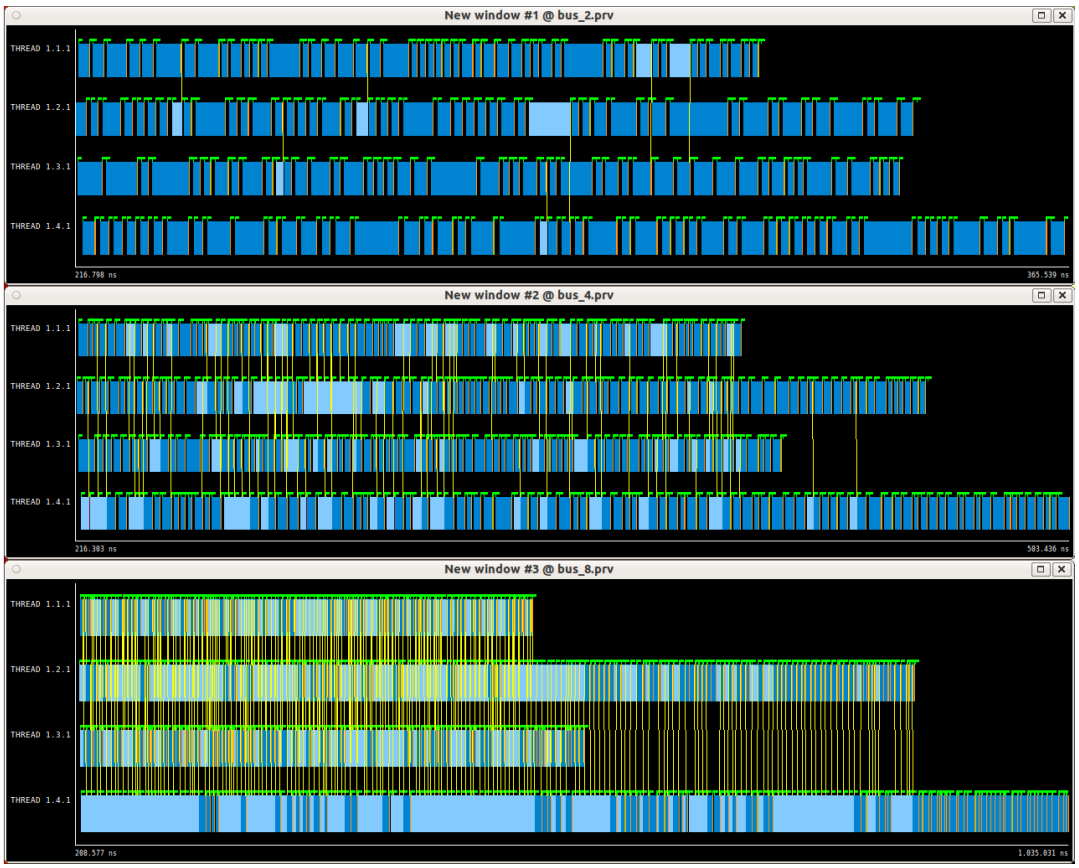

Figure 4: Visualisation of changing contention levels

Figure 4 shows three runs of an application, each with increasing contention levels (amount of aborts). The application simulates a banking house and runs on 4 cores. Dark blue parts indicate computation done in committed transactions, light blue parts on the contrary indicate computation done in aborted transactions, i.e. wasted work. Yellow lines connect transactions being aborted with the transaction causing the abort.

Figure 5 shows another high-level view of the third run, the run with the highest contention level. This time a histogram of the exact time spent in various HTM stages is displayed.

The rows have the following meaning:

- "**Compute**": Time spent doing calculations and transactional reads and writes in successfully commited transactions (e.g. "Useful work")
- "**Compute_Wasted**": Time spent doing calculations and transaction reads and writes in aborted transactions (e.g. "Wasted work")
- "**Try_Lock**": A transaction has finished computing, HTM subsystem tries to prepare the commit phase by locking the ring bus
- "**Commit_Locked**": Ring bus lock was acquired, transactional data is flushed to RAM

	Compute	Compute_Wasted	Try_Lock	Commit_Locked	Abort
THREAD 1.1.1	179.719 ns	143.580 ns	214 ns	2.336 ns	2.546 ns
THREAD 1.2.1	268.932 ns	338.118 ns	376 ns	3.844 ns	5.075 ns
THREAD 1.3.1	209.214 ns	142.754 ns	560 ns	4.484 ns	2.749 ns
THREAD 1.4.1	275.827 ns	439.813 ns	709 ns	6.153 ns	6.432 ns
Total	933.692 ns	1.064.265 ns	1.859 ns	16.817 ns	16.802 ns
Average	233.423 ns	266.066,25 ns	464,75 ns	4.204,25 ns	4.200,50 ns
Maximum	275.827 ns	439.813 ns	709 ns	6.153 ns	6.432 ns
Minimum	179.719 ns	142.754 ns	214 ns	2.336 ns	2.546 ns
StDev	40.401,68 ns	128.050,95 ns	186,74 ns	1.368,94 ns	1.627,00 ns
Avg/Max	0,85	0,60	0,66	0,68	0,65

Figure 5: Histogram of time spent in HTM stages

- "**Abort**": Transaction got invalidated, HTM subsystem is reset and transaction is prepared for restart

The values of rows "Try_Lock", "Commit_Locked" and "Abort" can be summed up and compared to the overall runtime to get the amount of overhead incurred by the current implementation of the HTM subsystem. A comparison of rows "Compute" and "Compute_Wasted" gives a rough idea of contention on a sysem level. A more fine-grained analysis down to the contention between two cores can also be done.

Some of the already known bottlenecks of the TMbox system were confirmed and quantified with these analysis capabilities. One of these bottlenecks is the increasing memory access latency when adding more processor cores to the system, an inherent attribute of a ring bus type core interconnect. Future work on the TMbox system may evaluate different approaches to core interconnection, leading to an improved system.

Adding the infrastructure to the TMbox system increased FPGA chip usage by a few percent. Please refer to table 1 for a detailed comparison of a 4 core TMbox system with and without the monitoring infrastructure.

	Monitoring infrastructure		
Type	Without	With	Increase
Slice LUTs	22,340	22,771	1.9 %
Fully used LUT-FF pairs	9,103	9,431	3.6 %
Total BRAM Memory used	864 KB	936 KB	8.3 %

Table 1: FPGA usage of 4 core TMbox system

Furthermore the work done in this STSM, which was about HTM, can be augmented with STM event generation to get more information about the capability, characteristics and optimization possibilities of HybridTM systems.

4 Future collaboration with host institution

Both participant sides (BSC and KIT) are currently discussing the opportunity of publishing a joint paper containing the results of this STSM. The paper will target a high-profile conference like DATE. There are also currently ongoing discussions about extending the collaboration.

5 Confirmation by the host institution of the successful execution of the STSM

The Hosts Osman Unsal and Adrian Cristal from BSC confirm that Philipp Kirchhofer achieved all the targets that we defined for this collaboration with distinction. He developed a very useful monitoring tool for the BSC FPGA-based HTM. In fact, the monitoring tool became a visualization tool as well, enabling the researchers to quickly analyze the execution time line of transactional applications.

6 Other comments

The following sections want to address the points raised by the EuroTM STSM committee. The comments by the committee are printed in bold text.

6.1 Prior Work

Prior work includes Chung et al's stats on TM programs at HPCA 2006, Ansari et al's work at PDP 2009, and (most closely to this proposal) the HW support for profiling from Chafi et al at ICS 2005.

Most of the referred papers are about STM. The general ideas in these papers are transferrable from STM systems but the specific implementation is quite different on HTM systems and is therefore not applicable.

The programing model of the underlying TMbox system [2] is comparable to the Transactional Coherence and Consistency model [5]. The monitoring techniques used in this STSM are in some parts comparable to the Transactional Application Profiling Environment [6], both developed at Stanford university. Major differences include the use of multiple ring buses

in the TMbox system, while other systems use a switched bus network with different timing characteristics and influence on HTM behavior. A bus network using a crossbar increases the complexity of the synthesized design on the FPGA chip and adds additional constraints for unit placement, ultimately reducing the scalability reached by the TMbox design. The ring bus interconnection used by TMbox allowed an easy addition of monitoring units. Another difference to TAPE is that the event-based monitoring infrastructure presented here allows to not only analyse HTM but also HybridTM behavior.

In addition, the researchers should try to involve Ferad Zyulkyarov at BSC, and/or to build on the tools that Ferad has developed in his PhD work. Ferad's PhD was on debugging and profiling TM programs. He built on STM systems, but it could be good to re-use the tools and workloads that he developed with the HTM profiling proposed here.

Ferad Zyulkyarov had already left BSC before the start of the STSM, therefore direct contact with him was not possible. Fortunately it was possible to attend his PhD defense and watch the presentation he held there about TM techniques. The Bartok compiler, which Ferad used in his thesis, is proprietary closed-source software and thus not adaptable to the TMbox platform. The workloads, which he developed using this compiler, therefore couldn't be reused for this project.

6.2 Novel ideas

The following novel ideas distinguish the work done here from previous research:

The visualization and analysis capabilities of the existing scalable tool Paraver have been leveraged for the purposes of HTM and HybridTM behavior analysis. This novel feature allows an easy interactive in-depth visualization and analysis of HTM behavior.

Using FPGA technology for the implementation of a monitoring infrastructure allows for fast execution and analysis of long and complex applications. As a positive effect this also allows the rapid adaption to new requirements and hardware design changes while providing excellent scalability in terms of performance and functionality.

6.3 Outlook

We would encourage the researchers to look at whether the ideas can be generalized beyond TM - are there other aspects of the system where better support for performance monitoring would be useful?

The event-based design created in this project can be easily extended to enable the analysis of all parts of processor core operations (like cache, ALU, TLB and TMU utilization and memory access patterns). This kind of data is useful for behavior and optimization research in other fields of computer science, for instance operating systems and hardware design, and for the construction of adaptive and self-optimizing systems. The visualization and analysis features

of Paraver can be used for this purpose by adapting the post processing program created in this STSM to new event types.

7 Acknowledgements

This work was supported by a COST Action IC1001 (EuroTM) STSM grant.

References

[1] C. J. Rossbach, O. S. Hofmann, and E. Witchel, "Is transactional programming actually easier?," *SIGPLAN Not.*, vol. 45, pp. 47–56, January 2010.

[2] N. Sonmez, O. Arcas, O. Pflucker, O. S. Unsal, A. Cristal, I. Hur, S. Singh, and M. Valero, "TMbox: A flexible and reconfigurable 16-core hybrid transactional memory system," in *Proceedings of the 2011 IEEE 19th Annual International Symposium on Field-Programmable Custom Computing Machines*, FCCM '11, (Salt Lake City, UT, USA), pp. 146–153, IEEE Computer Society, 2011.

[3] "Bee3 research platform." http://research.microsoft.com/en-us/projects/bee3/.

[4] "Paraver website." http://www.bsc.es/paraver.

[5] L. Hammond, B. D. Carlstrom, V. Wong, B. Hertzberg, M. Chen, C. Kozyrakis, and K. Olukotun, "Programming with transactional coherence and consistency," in *Proceedings of the 11th international conference on Architectural support for programming languages and operating systems*, ASPLOS-XI, (New York, NY, USA), pp. 1–13, ACM, 2004.

[6] H. Chafi, C. C. Minh, A. McDonald, B. D. Carlstrom, J. Chung, L. Hammond, C. Kozyrakis, and K. Olukotun, "TAPE: a transactional application profiling environment," in *Proceedings of the 19th annual international conference on Supercomputing*, ICS '05, (New York, NY, USA), pp. 199–208, ACM, 2005.