Enhancing Real-Time Behaviour of Parallel Applications using Intel TSX

Florian Haas, Stefan Metzlaff, Sebastian Weis, and Theo Ungerer
Department of Computer Science, University of Augsburg
Augsburg, Germany

{haas,metzlaff,weis,ungerer}@informatik.uni-augsburg.de

***Abstract*—With the Transactional Synchronization Extensions (TSX), implemented in the current Intel Haswell processor architecture, hardware transactional memory has now arrived at general-purpose computing. Although TSX simplifies parallel programming and speeds up general-purpose parallel applications, it entails timing difficulties for real-time systems in case of conflicts and transaction aborts. In this paper, we propose a software-based contention manager, which enhances the task scheduler of a real-time Linux system and thus allows for using Intel TSX in real-time context.**

Ensuring real-time requirements of parallel applications executed on multi-cores is challenging, because timing interferences and delays may occur on almost every level of the system. At the hardware level the access on shared resources like bus or memory causes timing interferences. The operating system level provides methods for scheduling the different tasks and granting resource access, but the mutual exclusion of accesses to critical sections pose delays that depend on the current workload of the system and are hard to determine. Further at application level, the lock-based concurrency control complicates the application design for the programmer and introduces delays due to serialisation of the critical sections.

Optimistic concurrency control, like Transactional Memory (TM) [1], is used to ease the design of parallel applications, since they suffer no priority inversion and deliver more comprehensive code. Unfortunately, typical TM systems do not consider the timing of the transactions, i.e. it is possible that certain transactions starve, while the TM system is operating properly. However, there are approaches that introduce real-time constraints in transactional memory systems by using priorities for transactions [2], proposing contention management strategies for real-time applications [3, 4, 5], or providing even an analysable TM for hard real-time systems [6, 7, 8]. In contrast to the design of a custom TM system, we will leverage a given TM system to ensure real-time constraints of parallel applications. With our work we target the Restricted Transactional Memory (RTM), which is part of the Intel TSX specification and implemented in the Intel Haswell processor. Nevertheless, our approach is also applicable for other TM implementations.

In a lock-based parallel application consisting of real-time tasks the task scheduler grants the access to shared resources of the different tasks by access serialisation. This causes an unknown delay (or it is at least hard to determine) of the real-time tasks, which may jeopardize the timeliness of each task. Our approach for parallel applications using Intel RTM can mitigate this problem, since different tasks may access the same shared data and only have to serialise their accesses on a conflict. However, the serialisation of conflicting tasks will delay the task on conflict. In contrast to the pessimistic lock-based approach the transactions can be serialised in a dynamic order by a central contention management. To provide a fair distribution of the delays induced by conflicts, we consider the enhancement of the task scheduler by a component that tracks and influences the transactional progress of each task according to its vitality. The tasks to be aborted are selected depending on their dynamic priority, which takes the number of previous aborts into account. For example, on frequent conflicts two tasks with the same priority will be forced to abort their transactions in a way to reach an evenly distributed abortion rate.

Therefore, we intend to enhance the RTM primitives of Intel's TSX specification by code, which informs the task scheduler about the status and progress of the transactions of each task. This makes it possible to extract the characteristics of each task regarding frequent aborts or conflicting transactions during run-time and adjust the dispatching of the transactions to the system. To delay tasks the system can suspend the task before entering a transaction, such that other tasks may advance. When the conflicting task has committed its transaction, the suspended task is waken up and is allowed to proceed.

As evaluation platform we will use a Linux system with the real-time patch PREEMPT_RT [9, 10] applied. With this kernel patch, most parts of the kernel code can be preempted. Priority inversions within the kernel are prevented by using different locks which support priority inheritance. In the following we will show how the TSX instructions (XBEGIN and XEND) and the transaction fallback code need to be enhanced and which additional interface to the task scheduler needs to be provided to ensure the described congestion control:

The enhanced transaction begin requests the dispatch of the transaction at a global instance (by calling a transaction dispatch hook). The transaction may be delayed by suspending the task due to possible conflicts with higher priority tasks or frequent aborts of other conflicting tasks. Once the task may dispatch its transaction, a transaction begin counter is increased, the tasks marks itself to be in transaction (by setting an `inTX` flag), and the transaction is started via XBEGIN. The enhanced transaction end first commits the transaction via XEND. Then, if no abort occurred, the task's commit counter is increased and the task resets its `inTx` flag. The reset of the `inTX` flag triggers the wakeup of any suspended tasks that were waiting on the completion of the transaction. On transaction abort an enhanced abort handler is called. It increases the abort counter and resets the `inTX` flag. Depending on the status of the system the transaction is restarted or (in case that a transaction retry is not promising) the lock-based backup code is executed. For the scheduling of the transactions an additional system function, the Transaction Dispatch System Function, is needed. It observes the abort ratios of the transactions of all tasks. By gathering the characteristics of the transactions and the priority of the tasks it is able to decide, whether a transaction needs to be delayed to grant a fair distribution of the delay times among all tasks. The task suspension can be implemented using conditionals that observe the `inTX` flag of the conflicting transactions. Thus, on transaction commit all waiting tasks are continued.

Using this approach the scheduling of tasks with transactions has more opportunities to prevent deadline misses caused by transactional conflicts, e.g. depending on the laxity of a task to a deadline its transactions may be delayed to grant other tasks a timely commit to reach their deadlines. We are confident that our approach provides a fair distribution of the delays caused by transactional conflicts and thus improves the real-time behaviour of parallel applications. Preliminary results are planed to be presented at the workshop.

Acknowledgment

This work is partly funded by an Intel Germany Microprocessors Lab Research Grant on high-performance computing in safety critical systems.

REFERENCES

[1] T. Harris, J. Larus and R. Rajwar. *Transactional Memory*. 2nd. Synthesis Lectures on Computer Architecture. Morgan & Claypool Publishers, 2010.

[2] J. Manson, J. Baker, A. Cunei, S. Jagannathan, M. Prochazka, B. Xin and J. Vitek. "Preemptible atomic regions for real-time Java". In: RTSS. 2005.

[3] T. Sarni, A. Queudet and P. Valduriez. "Real-Time Support for Software Transactional Memory". In: RTCSA. 2009.

[4] W. Maldonado, P. Marlier, P. Felber, J. Lawall, G. Muller and E. Riviere. "Deadline-aware scheduling for Software Transactional Memory". In: DSN. 2011.

[5] J. Gottschlich and D. Connors. "Extending contention managers for user-defined priority-based transactions". In: EPHAM. 2008.

[6] M. Schoeberl, F. Brandner and J. Vitek. "RTTM: real-time transactional memory". In: SAC. 2010.

[7] M. Schoeberl and P. Hilber. "Design and Implementation of Real-Time Transactional Memory". In: FPL. 2010.

[8] S. Metzlaff, S. Weis and T. Ungerer. "Leveraging Transactional Memory for a Predictable Execution of Applications Composed of Hard Real-Time and Best-Effort Tasks". In: RTNS. 2013.

[9] S. Rostedt and D. V. Hart. "Internals of the RT Patch". In: Proceedings of the Linux symposium. 2007.

[10] *PREEMPT RT kernel patch*. URL: https://rt.wiki.kernel.org/index.php/CONFIG_PREEMPT_RT_Patch.