POSTER: Fault-tolerant Execution on COTS Multi-core Processors with Hardware Transactional Memory Support

Florian Haas, Sebastian Weis, and Theo Ungerer
Department of Computer Science
University of Augsburg, Germany
{haas,weis,ungerer}@informatik.u-augsburg.de

Gilles Pokam, Youfeng Wu
Intel Corporation, Santa Clara
{gilles.a.pokam,youfeng.wu}@intel.com

ABSTRACT
Software-based fault-tolerance mechanisms can increase the reliability of multi-core CPUs while being cheaper and more flexible than hardware solutions like lockstep architectures. However, checkpoint creation, error detection and correction entail high performance overhead if implemented in software. We propose a software/hardware hybrid approach, which leverages Intel’s hardware transactional memory (TSX) to support implicit checkpoint creation and fast rollback. Hardware enhancements are proposed and evaluated, leading to a resulting performance overhead of 19 % on average.

1. INTRODUCTION & RELATED WORK
Intel introduced the Transactional Synchronization Extensions (TSX) as part of the Haswell instruction set architecture [2]. TSX uses the L1 cache to hold back transactional data, which have not yet committed. Its implicit rollback mechanism can be utilized for fault tolerance to recover from detected errors. Loosely coupled lockstep architectures like [4] support multi-core processors, but complex changes to the hardware are required. A similar, but software-only approach is PLR [3]. An approach which leverages transactional memory for error correction was proposed in [1].

This paper presents an evaluation for the Intel Haswell architecture and with respect for proposed hardware enhancements. The instrumentation mechanisms provide an increased code coverage with less performance overhead than other approaches. Combining the flexibility of software-based redundant execution with existing hardware transactional memory allows efficient checkpointing and rollback for fault-tolerant execution of individual applications.

2. ERROR DETECTION & RECOVERY
The main principle of our approach is to execute each user process redundantly and to instrument signature-based comparison on function level. The loosely coupled execution of both processes allows error detection. With the encapsulation of blocks in transactions, error recovery is realized. The instrumentation is done on the binary and the process duplication takes place at runtime. Thus, the executed code of both processes is almost identical.

Comparison of both processes is done on the level of function-based blocks. For an efficient comparison, we create signatures of these blocks which then are exchanged from one process to its duplicate. Fig. 1 shows a schematic of the redundant execution and block-based comparison. A process is duplicated at a user-defined entry point function call. All code within this function is executed redundantly (step 1). Binary instrumentation divides this code into blocks, which are aligned at function boundaries (step 2). Signatures of these blocks are calculated in each process and transferred from the leading process to the trailing process through a shared-memory FIFO queue (step 3). Only the trailing process instruments transaction begin and end instructions around each block. Before committing the transaction, the locally calculated signature and the leading process’ signature are compared. In case of a detected error, the transaction is explicitly aborted, leading to a restart at the beginning of the block. To ensure a correct state of the leading process, it is killed by means of the operating system and a fork creates a new leading process. The program execution then continues at the beginning of the previously faulty block.

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored.

PACT ’16 September 11-15, 2016, Haifa, Israel
© 2016 Copyright held by the owner(s).
DOI: http://dx.doi.org/10.1145/2967938.2974051

Figure 1: Instrumentation on function-based blocks b_n and redundant execution in processes p and p'.
3. HARDWARE ENHANCEMENTS

We propose hardware enhancements for the Intel Haswell micro-architecture, which could further improve our approach in terms of code coverage and performance. The potential impact of these enhancements is investigated by emulation:

Signature Generation.

Since the most relevant bottleneck is the signature generation, our approach benefits from hardware signature generation. A possible hardware extension may calculate the signature implicitly, by issuing a hash calculation on every read from a register and on every write to a register or to memory. The signature can be stored in a dedicated register, which can be reset and read by the software.

Hardware Queue.

Further improvement is possible by supporting signature exchange in hardware. This requires a mechanism to send data uni-directional between individual cores, additionally with buffering to form a FIFO queue. It is sufficient to connect pairs of cores, a queue between every single core is not required. The assignment of processes to cores is handled by the software library.

Transactional Memory.

Enhancements to TSX can also increase the performance of our approach. Increased robustness of TSX lowers the overhead related to aborting transactions. In our approach, transactions should not abort in the error-free case, since conflicts do not occur. Robust transactions which survive events like cache-overflows, interrupts and system calls allow a guaranteed transactional execution, resulting in a better coverage and less overhead for instrumented programs. Furthermore, the communication between transactions will always lead to conflicts between both processes, and thus prohibits parallel transactional execution of both redundant processes. With escape actions, signatures can be exchanged during the commit phase. The transaction will only be allowed to commit if the signatures match. Otherwise, both transactions roll back and try again. This reduces overhead of error-correction.

4. EVALUATION

Fault-tolerant execution of instrumented benchmarks leads to varying amount of runtime overhead. The code coverage for all selected benchmarks is 92.5 % on average. To determine the sources of the overhead, parts of the instrumentation have been measured individually. Results are shown in Fig. 2. One source of overhead is the transaction instructions, which wrap

the dependable blocks in the trailing process. Since TSX does not guarantee a non-conflicting transaction to commit eventually, random aborts may occur. Within Fig. 2, the fraction of the performance overhead related to transactional memory instructions is shown in the black bars at the bottom. The overhead of individual, per-block generation of signatures depends on the size of the blocks and on the number of blocks. The impact of signature generation can be seen in the third gray bar from top in Fig. 2. The execution time overhead related to signature exchange through the FIFO queue is shown in the second gray bar from the top in Fig. 2. Remaining overhead is due to additional instructions inserted during instrumentation. The gray bars on top in Fig. 2 represent the instrumentation overhead, which is 3.95 % on average.

The gray bars in Fig. 3 show the relative execution time of the fault-tolerant execution compared to the original benchmark execution. On the Haswell Core i7, the redundant execution on two cores takes more than twice the time for some benchmarks, and 132 % on average. The emulation of hardware support for signature generation and exchange leads to significantly improved performance. The overhead of our approach could be decreased to reach a relative performance overhead of 19 % on average (see black bars in Fig. 3). The overhead can be further reduced with enhancements to TSX, e.g. escape actions or guaranteed transactional execution. This also increases the overall applicability.

5. REFERENCES


