SymptomTM: Symptom-Based Error Detection and Recovery Using Hardware Transactional Memory

Gulay Yalcin*, Osman S. Unsal*, Adrian Cristal* †, Ibrahim Hur* and Mateo Valero*
* Barcelona Supercomputing Center
† IIIA - Artificial Intelligence Research Institute - CSIC - Spanish National Research Council
Email: (gulay.yalcin, osman.unsal, adrian.cristal, ibrahim.hur, mateo.valero)@bsc.es

***Abstract*—Fault-tolerance has become an essential concern for processor designers due to increasing transient and permanent fault rates. In this study we propose SymptomTM, a symptom-based error detection technique that recovers from errors by leveraging the abort mechanism of Transactional Memory (TM). To the best of our knowledge, this is the first architectural fault-tolerance proposal using Hardware Transactional Memory (HTM). SymptomTM can recover from 86% and 65% of catastrophic failures caused by transient and permanent errors respectively with no performance overhead in error-free executions.**

I. Introduction

In addition to power and performance, reliability is becoming a first-class design constraint for processor designers. Thus it is important to develop simple (in terms of complexity-effectiveness) and powerful (in terms of error detection and recovery coverage) fault-tolerance mechanisms. Recent error detection solutions [3], [6] monitor if there is a symptom of hardware faults in order to provide low-cost fault tolerance schemes. Although these symptom-based error detection schemes provide acceptable fault coverage, they require a simple recovery mechanism to roll back to an error-free state after detecting an error. In this paper, we introduce SymptomTM, an architectural fault-tolerance scheme which utilizes a symptom-based error detection scheme and leverages Hardware Transactional Memory (HTM) mechanisms for error recovery. To the best of our knowledge, this is the first architectural fault tolerance proposal using HTM.

HTM systems provide mechanisms to abort transactions in case of a conflict, thus they discard or undo all the tentative memory updates and restart the execution from the beginning of the transaction. Thus, transaction start can be viewed as a checkpointed stable state. SymptomTM uses the TM abort mechanisms for error recovery.

SymptomTM executes vulnerable code in a special transaction and it monitors if this transaction presents an error symptom (fatal traps in our case). In case of a fatal trap exception, SymptomTM aborts the transaction to recover from the error. Thus, SymptomTM provides a simple recovery mechanism. We designate fatal traps (e.g attempting to execute an undefined instruction code) as our error symptom which is examined by SWAT group for permanent faults in detail [2], [3], [5]. They conclude that it has high error coverage (66%) with no false positive impact unlike other error symptoms such as mispredictions in high confidence branches or high OS activity.

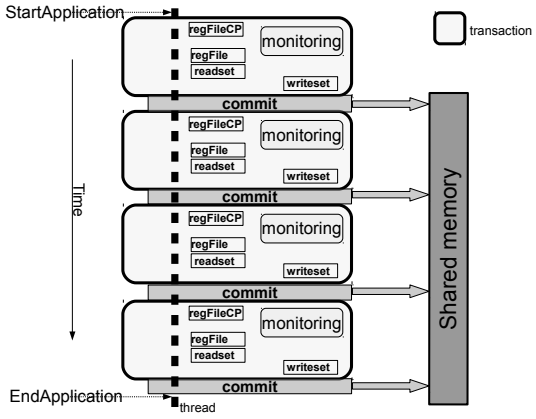

Fig. 1. Design of SymptomTM

II. Basic Design of SymptomTM

The SymptomTM system starts a single special transaction at the beginning of the application (Figure 1). This special transaction is executed atomically and isolated from the system by writing to the local HTM write-set buffer until commit. Hence, errors do not propagate out of the transaction. Also, the execution of the transaction is monitored to detect if there is any symptom of hardware errors, in this case fatal traps. Unless any fatal trap exception is raised in the transaction, the write-set is committed to shared memory at the end of the transaction. Commit process starts whenever write-set size equals the transactional log size. If a symptom is detected, the transaction aborts and restarts the execution from the beginning of the transaction. If there is no symptom at the end of the second restarted execution, that means that the error was transient and that it was corrected. If the second execution raises the fatal trap exception signal again, this could be due to a permanent fault. In this case, SymptomTM allocates another core, copies the checkpointed state of the transaction to the second core and re-executes the transaction. If the second core does not raise an exception, that means the first core had a permanent fault and finally it should be disconnected from the system. Otherwise, either the error is caused by software or SymptomTM can not recover from it. The algorithm used

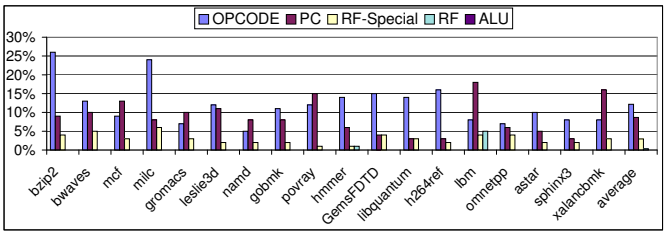

Fig. 2. Fatal Trap Rate of Injected Transient Faults

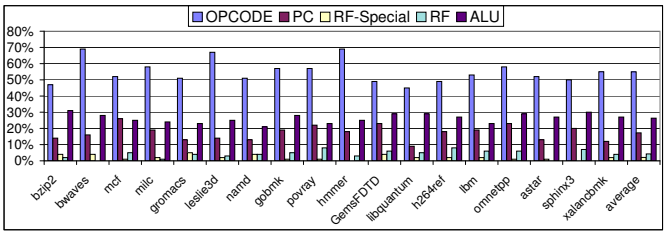

Fig. 3. Fatal Trap Rate of Injected Permanent Faults

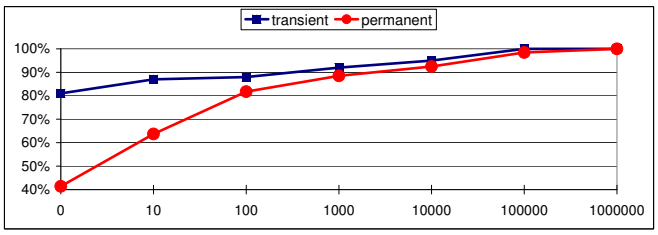

Fig. 4. Number of Stores Executed between Fault Injection and Exception Raise

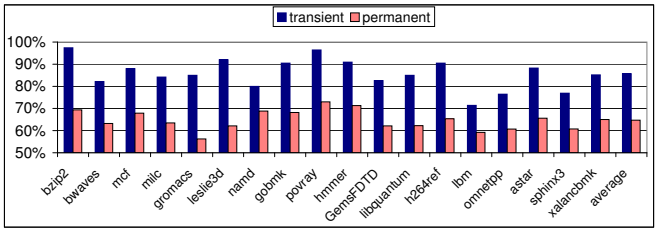

Fig. 5. Error Recovery Performance of SymptomTM

to diagnose the faulty core, is quite similar to the one utilized in TBFD [2].

SymptomTM does not have a perceptible performance degradation in the error free execution that makes it an attractive mechanism for high reliability. It only increases pressure on memory bandwidth since the write operations are convoyed on commit. The main benefit of SymptomTM is that it provides an error recovery mechanism by leveraging the abort mechanism of TM. Atomic transactions guarantees that error does not propagate to the other cores. Thus, rolling back only the core that raises the fatal trap is adequate to recover from faults since the rest of the system is error free. For system-wide checkpointing and data sharing systems, once a fault is detected, no core in the system can be assumed to be fault-free. Due to this problem mSWAT [5] replays all cores in the system up to three times to diagnose the faulty core. SymptomTM replays one core once.

III. Evaluation

We use the M5 full-system simulator [1] with an implementation of a Hardware Transactional Memory system that uses lazy data versioning and lazy conflict detection [4]. We extend this simulator with our SymptomTM implementations. We use fault injection to measure the reliability performance of SymptomTM for both transient and permanent faults.

Figure 2 and Figure 3 shows the hardware exception rate of transient and permanent faults according to our fault injection experiments. As expected, permanent faults are more likely to induce catastrophic failures than transient faults (21.6% vs. 4.8%). We present the exception ratio for different microarchitectural structures, because, in terms of system failures, each structure in the microarchitecture is not equally vulnerable. Note that our fault injection structures are different from SWAT [3] so that our exception ratio for permanent faults is different. Also, SWAT does not present these results for transient faults in detail.

SymptomTM can detect all these critical faults. However, it is able to recover from an error if it raises the exception before the end of the transaction. Therefore, transaction size is a critical issue for recovery. Also, the transaction size is limited by the size of the local buffer area that is correlated with the maximum number of store instructions within a transaction. Figure 4 depicts the number of store instructions executed between the bit flip and exception raise. 87% of catastrophic failures induced by transient faults raise an exception within 32 store instructions (Transaction size in SymptomTM). However, fewer amount of the permanent faults (70%) raise exceptions within 32 store instructions. SymptomTM with a write-set of 32 entries recovers, on average, 86% and 65% of catastrophic failures caused by transient faults and permanent faults respectively (Figure 5). This coverage can be increased with larger write-sets. As it is seen from Figure 4 within 1M stores (~10M instructions which is the checkpoint interval in SWAT) all catastrophic failures can be recovered.

References

[1] Nathan L. Binkert, Ronald G. Dreslinski, Lisa R. Hsu, Kevin T. Lim, Ali G. Saidi, and Steven K. Reinhardt. The M5 simulator: Modeling networked systems. *IEEE Micro*, 26:52–60, 2006.

[2] Man lap Li, Pradeep Ramach, Swarup K. Sahoo, Sarita V. Adve, Vikram S. Adve, and Yuanyuan Zhou. Trace-based microarchitecture-level diagnosis of permanent hardware faults. In *DSN*, 2009.

[3] M. Li, P. Ramach, S. K. Sahoo, S. V. Adve, V. S. Adve, and Y. Zhou. Understanding the propagation of hard errors to software and implications for resilient system design. In *ASPLOS*, 2008.

[4] S. Sanyal, S. Roy, A. Cristal, O. S. Unsal, and M. Valero. Dynamically filtering thread-local variables in lazy-lazy hardware transactional memory. In *HPCC*, pages 171–179, 2009.

[5] Siva Kumar Sastry Hari, Man-Lap Li, Pradeep Ramachandran, Byn Choi, and Sarita V. Adve. mSWAT: low-cost hardware fault detection and diagnosis for multicore systems. In *MICRO*, pages 122–132, 2009.

[6] Nicholas J. Wang and Sanjay J. Patel. ReStore: Symptom-based soft error detection in microprocessors. *TDSC*, 3:188–201, 2006.