An Architecture to Enable Lifetime Testing in CMPs

Rance Rodrigues
Department of Electrical and Computer Engineering
University of Massachusetts at Amherst
email:rodrigues@ecs.umass.edu

ABSTRACT

As CMOS technology scaling continues to low nanometers, aging and device wear-out related degradation is a looming concern. With device wear-out mechanisms such as NBTI and PBTI, catalyzed by higher temperature, transistor performance worsens with time. Similarly with electromigration, interconnects become increasingly resistive until they become non-functional. Such phenomena initially manifest as delay defects, which in the due course of time turn into permanent defects. Online testing is needed to avert situations in which such defects show up as errors during operation. In this paper we propose an architecture that can assist online testing at various levels in a Chip Multiprocessor (CMP). The proposed solution revolves around the incorporation of a small and simple, functionally limited core called the Sentry Core (SC) in the CMP. This core is responsible for assisting with testing of the functional cores in a CMP. Since SC is small, it is assumed to be hardened during design and remain fault-free during useful lifetime of the chip. We explore two of the many possible online testing schemes possible with this architecture. In the first scheme, we explore the use of SC in triggering opportunistic DMR to verify functional correctness of the cores. In the second scheme, we evaluate the effectiveness of the SC in monitoring and potentially detecting errors in operation of the cache coherence protocol. Results indicate that even though incorporation of the SC results in an area overhead in the CMP, the rich functionality it provides with respect to testing and fault tolerance justifies this cost.

Keywords

Online testing, Sentry Core (SC), Dual Modular Redundancy (DMR), Signature, Cache Coherence.

1. INTRODUCTION

The relentless push in technology scaling has led to smaller transistors. However, as device dimensions decrease, current and power densities increase. This results in increased chip operating temperature which accelerates the CMOS wear-out mechanisms such as Negative Bias Temperature Instability (NBTI), Time Dependent Dielectric Breakdown (TDDB) [13] etc., which ultimately results in poor device reliability [4][5].

The CMOS wear-out mechanisms initially manifest themselves as delay defects [14] during operation, resulting in timing violations. As a result, operating frequency must be reduced to ensure correct operation. With time, these defects turn into permanent defects and the number of such defects increases with time. Hence there is need for mechanisms that can monitor operation of the chip and flag any deviation from the expected, online.

In this paper, we propose an online testing scheme for a CMP which can operate at various levels. This scheme is based on incorporation of a small and simple, functionally limited core called the Sentry Core (SC), in the CMP (Figure 1). Since this core is small and simple, its operation can be assumed to be fault free. This assumption is akin to similar assumptions used in watchdog processors [16] and the DIVA checker [1]. The SC is assigned the task of testing operation of the other more complex cores in a CMP. There are a variety of potential tests that can be run by the SC. In this paper, we present two such tests. In the first scheme, SC is used to trigger and monitor opportunistic DMR. In the second scheme, we utilize the SC to monitor and verify the operation of the cache coherence protocol in hardware. Additional details on each scheme follow later in the paper.

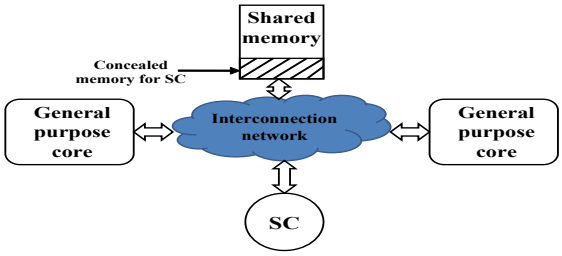

Figure 1: Sentry core (SC) with a dual core processor [19].

The rest of the paper is organized as follows: In section 2, we provide details on the capabilities and features of the SC and also compare it against literature that closely relates to it. In Section 3, we present the two possible online testing schemes using the SC. Each scheme is compared against state-of-the-art and analysis of the results is presented. This is followed by conclusions in section 4.

2. THE SENTRY CORE (SC)

Proposed test solutions are built around the SC which verifies the fault-free operation of the general-purpose processor cores. In this section, we cover some of the prior work related to SC and point out the key differences. We then describe the functionality and hardware overhead of incorporating an SC in a CMP.

2.1 Related work

Incorporation of service cores in a CMP is not a new idea. This solution has long been in existence with watchdog processors, where small, functionally limited processors are used to detect control flow and data errors [20][16]. However, these cores are limited in testing capability. For example, in [16], Benso et al. use the watchdog processor to verify control flow correctness by monitoring the system bus. However, their scheme is unable to initiate DMR on the cores in a CMP, something that the SC has capability to do. In [1], Austin proposed the DIVA checker which is a small companion core for a complex core, tasked with verification of functionality in the complex core. The basic principle is that if the results from the DIVA core and the complex core diverge, a fault is detected. However, this scheme can only detect and mask faults in the processor core. It cannot for example, verify cache coherence operation. More recently, lau et al. [21], presented the partner cores concept, where each complex core in a CMP is augmented with a small core for reliability and performance improvements. However, pairing a partner core with each complex core increases overhead. This is not the case with

the proposed sentry core solution, where one SC serves multiple cores on a chip. The sentry core concept was introduced by us earlier in [19]. The focus of that work was to perform opportunistic DMR. In this paper, we include a brief review of that work.

2.2 SC features [19]

The SC is a small and simple core with the objective of enabling quick fault detection in a CMP. The SC needs to have the capability to initialize the test as well as collect and compare results of execution and detect faults if any. To this end it is augmented with a variety of features described next.

2.2.1 Control functions

To assist in fault detection, the sentry core has support for the following operations:

Detect idle cores: SC has the capability to detect if cores are idling. A simple way to do this is by monitoring IPC of the cores.

Duplicate: trigger a DMR/TMR (Dual/Triple Modular Redundancy) configuration by replicating a process and executing it on two or three cores.

Suspend: halt one or all the processor cores to analyze their state.

Resume: resume the operation of a halted core(s).

Terminate: terminate a process on a core.

Monitor the shared bus: SC has access to the shared bus. Hence it has access to all the cache coherence transactions in the CMP.

Message logging: The SC logs messages in its own cache. It can later use this to verify correct operation.

Knowledge of coherence protocol: The SC has knowledge of the cache coherence protocol and hence can predict the next state of a cache line based on its current state and the memory operation being performed.

We have listed only a subset of the many possible functions that a sentry core can provide for fault detection. This functionality can easily be added to any off-the-shelf processor that fits the description of the SC by extending the Instruction Set Architecture (ISA). In many processors, the opcode field size is extensible. For example, in [10], an example of ISA extension is described to incorporate multimedia instructions. The added instructions will be used by the SC to facilitate testing of multicore.

2.3 Hardware considerations for the SC [19]

The SC provides rich fault detection functionality in a CMP with a small hardware overhead. To estimate the overhead of incorporating the SC in a CMP, we consider for example, the EV4 (Alpha 21064), and EV6 (Alpha 21264) cores. The functionality of the EV4 is sufficient to satisfy the requirements of an SC for a CMP comprising of EV6 cores. The EV4 occupies about 11% of the area of an EV6 core [11]. Hence, for a dual/quad/eight core CMP, the area overhead due to the SC reduces to 5.5/2.75/1.375%. This is an acceptable overhead for the added functionality considering that DIVA [1] adds a 6% overhead. While making this comparison in area, it is important to note that the SC scheme is a highly scalable solution unlike DIVA which incurs a fixed overhead for every core in the CMP. A single SC can service a quad, eight or even a sixteen core CMP. However, as the number of cores serviced by the SC increases, the time slice that each core gets for service reduces. For many core systems, additional SCs may be incorporated such that there is one SC for every *m* cores in the CMP. We plan to evaluate this in the future. The above is just an example to illustrate the size of the overhead. In reality, a customized (rather than an off-the-shelf) SC design may have far lower area overhead (< 1%). Incorporation of an SC results in heterogeneity which increases the design time, but heterogeneity is no longer a novel concept [1],[11] and hence may be considered an acceptable practice.

3. ONLINE TESTING SCHEMES USING SC

There are various testing schemes possible using the SC. In this paper, we explore a couple of those. In the first, we leverage the SC to trigger opportunistic DMR in the CMP to verify functional correctness of the cores. In the second, SC is used to verify correct operation of the cache coherence protocol by monitoring the shared bus. Next, we describe operations in each scheme in detail.

3.1 Opportunistic DMR

A multicore chip features more than one functional core, creating opportunities for functionally verifying the operations in one core using the other. In double modular redundancy (DMR) each resource needs to be duplicated. However in a CMP, such resources are already in existence. They just need to be exploited. In this sub-section, we illustrate the use of SC in opportunistic DMR. We first present the recent state-of-the-art in DMR schemes and then delve into the details of the proposed scheme. We would like to remind the reader that this sub-section is a review of our earlier work published in [19].

3.1.1 Related work

DMR/TMR for fault detection has been for long been in existence [15]. Modulo redundancy has high area and power overhead (200/300%). Redundant Multi Threading (RMT) [2][3] is an improvement over such schemes. However, the major source of inefficiency in these schemes is the amount of architectural state that must be compared. Smolens et al. [18] proposed a RMT solution in which the fingerprint of instructions between checkpoints is compared for error detection. Hence state comparison is replaced by comparison of a simple 32 bit word significantly reducing the overheads of RMT. However, comparison of states is assumed to be done by an error-free core. As will soon be seen, the proposed DMR scheme using SC will overcome all of the shortcomings of prior art.

3.1.2 Opportunistic low overhead DMR using SC

We try to reduce the overheads of DMR using SC in 2 ways (i) initiating DMR *opportunistically* and (ii) reduce the amount of state for comparison. Multicore chips are designed to support peak performance, but their average utilization is low [9]. These idle cores are ideal candidates for redundant execution. As mentioned earlier, SC has the capability to detect idling cores. When an idle core is detected, SC initiates DMR by replicating a functional thread. To reduce the amount of architectural state that needs to be compared, signatures are used. Specifically, two signatures are of importance. One signature captures the program execution path by

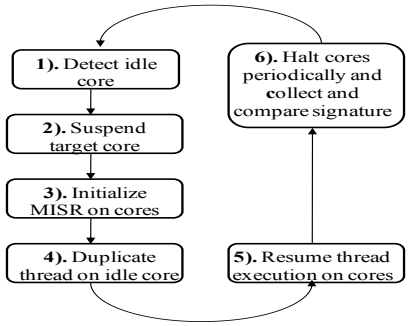

Figure 2: Sequence of steps followed by SC for test scheduling [19].

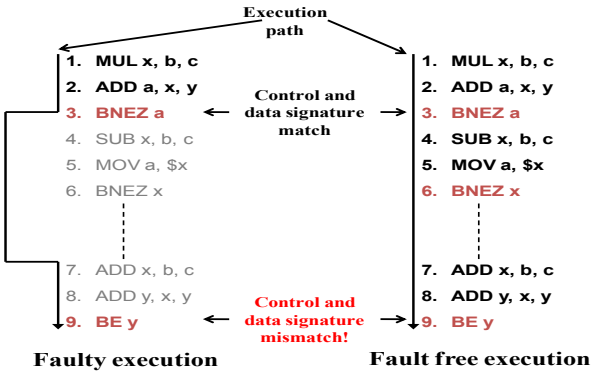

Figure 3: An illustration of the fault detection mechanism [19].

compressing the address of the committed branch instructions and the other signature compresses address and data values of the memory store operations. Since it is expected that any fault in operation will result in a difference in the signatures generated by the fault free and faulty cores, this scheme enables efficient comparison. The sequence of steps followed by SC for test scheduling is shown in Figure 2 and an example illustrating the fault detection mechanism is shown in Figure 3. We evaluated this scheme in [19] and found that the proposed opportunistic DMR results in fault coverage of ~87%. This is encouraging, considering that the workloads used for test purposes were functional workloads and not specifically engineered test codes.

3.2 Cache coherence protocol verification using SC

In the previous section we presented a review of earlier work. In this section, we present new work in application of SC in verifying cache coherency.

With the growing popularity of multicores and multiprocessor systems, functional correctness of shared memory applications is of paramount importance [8][22][23][24]. For example, in multiprocessor systems, the data may be shared among processors. If a core sharing the data modifies it, other cores that share the data must be made aware of this modification, else it could lead to stale data usage. Coherence protocols are used to avoid this situation. Protocols such as the MESI and MOESI define states of each cache line for various operations such that data is consistent across cores. They ensure usage of up-to-date data. There are various flavors of protocols, such as those based on snooping [23] or on the use of directories [26]. In this work, we focus on the systems that employ snooping to ensure cache coherence. An illustration of such a system with the SC incorporated in the CMP is shown in Figure 4. In these systems, memory accesses by any core in the CMP are sent on a common shared bus. Other cores snoop this bus, and if they observe an access made to a cache line that is present in their cache, appropriate actions are taken, as defined by the protocol. Since SC is part of the CMP, it has access to the shared bus and as mentioned earlier, is also aware of the coherence protocol. Hence, by observing the current state of the cache line being addressed and the type of operation (read, write etc.), SC knows the expected next state for that cache line. If due to some error the cache line transitions into a state other than that predicted by SC, it will be detected. This is the basic working of the proposed system. Even though this system shares similarities with some other cache coherence verification schemes [23][22], the benefits of using SC for coherence verification will soon be clear. A brief

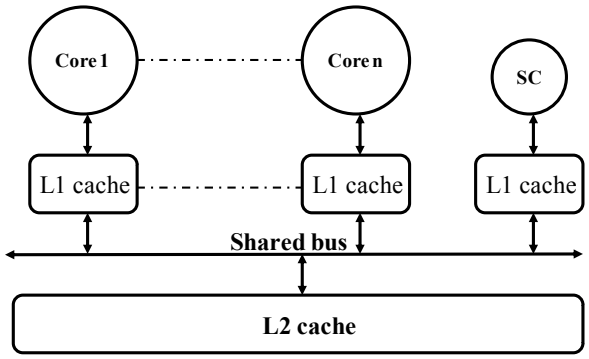

Figure 4: SC in a CMP that employs use of a shared bus.

overview on recent state-of-the-art in cache coherence protocol verification is presented next.

3.2.1 *Related work*

With the growing popularity of shared memory multicores, there have been quite a few proposals on the verification of cache coherence protocols. In [23], Cantin et al. presented a variation of the DIVA checker for cache coherence verification. Just like DIVA does for functional correctness of the cores, cache coherence transactions were verified using simpler logic. However, this scheme required the use of a separate network for global verification of coherence states. In [24], Fernandez-Pascual et al. present a scheme for cache coherence verification in presence of network failures. However, the scheme cannot be used to ensure correct transition of coherence states. In [22], Borodin et al. present a distributed system to verify cache coherence. In their solution, each cache that participates in the coherence protocol is assigned a checker that verifies its operation. This checker is a replica of the entire cache being monitored. Checker logic present in each replica then enables verification. This scheme is closest to ours. However, this scheme incurs an overhead that increases linearly with the number of cores in the CMP unlike ours where a single SC services a number of cores. Further, unlike SC which enables many potential testing schemes, their scheme only provides assistance to cache coherence verification. In [8], DeOrio et al. present a scheme to verify cache coherence post silicon. In this scheme, during the verification, part of the cache is used to log accesses made to memory. Every so often, a post processing algorithm is used to process the logs which detects errors based on discrepancies in the logs of various cores. However, this scheme cannot be employed online and hence cannot counter errors due to aging. Hence it can be seen that even though there exists literature on cache coherence verification, they have a few shortcomings.

3.2.2 *The proposed solution*

We propose the use of the SC for verification of coherence protocols in snooping bus systems. The fact that SC has access to the shared bus makes this scheme very feasible. In the proposed scheme, SC maintains a log for each and every access that a core in the CMP makes to memory, in its own cache. Whenever a core requests access to a cache line, it is seen on the shared bus, and hence it is also seen by SC. We assume that along with the address of the memory being requested, its current coherence state is also broadcast. The same assumption has been made by Borodin et al. [22]. SC logs this memory request and also its current state. Based on the type of operation involved i.e. read, write or invalidate, SC knows the expected next state of the cache line. Whenever the same line is accessed again, SC verifies that

the cache line has transitioned to a coherence state as expected. If it does, SC retires that entry from the log. If due to an error it does not, SC flags a possible error in operation.

In the cache coherence protocol, there are two types of state transitions that must be verified. Firstly, within a local cache, a cache line must transition from one state to another deterministically. Cache lines shared in multiple caches must also be coherent globally. For example, a line that is in the shared state in one cache cannot be in the modified state in another cache. This will result in a violation of the protocol. The verification of the state transition of a local cache line is trivial, as every memory access made by a core is acknowledged, either by main memory, or by a core that has the line in its cache. However, global verification is a more difficult problem. One way to perform global verification is by a global broadcast of the cache line state, whenever it changes as proposed by Cantin et al. [23]. However, this will pose a significant performance overhead. We want to avoid this overhead. A cheap way for verification is by observing the shared bus. Whenever a cache line that is shared by multiple cores is accessed by one of the cores, there is a good chance that it will appear on the shared bus. For example whenever a core writes to a cache line in the shared state, all other cores sharing that line will be notified via the shared bus. Also whenever a write/read operation is done to a line in the invalid state, the local L1 cache controller will broadcast its request on the shared bus. All cores that have a copy of that line in their cache, will respond to such a broadcast based on the state of that line in their own cache. Such broadcasts are a good chance for global verification. However there are problems with this scheme. If a line in the shared state is erroneously stored as modified or exclusive locally by a core, any future write to this line by the core will result in no broadcasts. This will lead to data inconsistency. The error may only be detected if in the future, another core has written to this line, which results in a broadcast. Hence the latency to verification is an important parameter. In this study, we present in addition to percentage of transactions verified, the average latency to verification. The experimental setup is now described, followed by the results

3.2.3 Experimental setup

In order to model a cache coherent CMP, we used the SESC simulator [6] and the SPLASH-2 benchmarks [25]. The simulator was modified to enable cache coherence verification. We used 6 workloads and each was run for 10M instructions. We present two entities, (i) percentage of the cache transactions that may be verified by observing the bus and (ii) average time for verification of each transaction. We also present a study on the effect of increasing number of cores and their L1 caches in the CMP on the results. A study on the optimal size of the SC cache, such that all requests may be logged is part of future work. To get an upper bound on the capability of this scheme, unlimited storage space is assumed in this study. The characteristics of the core used for the experiments is shown in Table 1. From the table, it can be seen that we have used a core that is representative of an Intel® Core™ i5.

Table 1: Parameters used for the experiments

Parameter	Value
Frequency	2GHZ
Fetch/Issue/Retire	6
INTREG	96
FPREG	80
INTISQ	36
FPISQ	24
ROB	128
L1$ Data/Instr	32K
L2$	2M
# of cores	4
Protocol	MESI

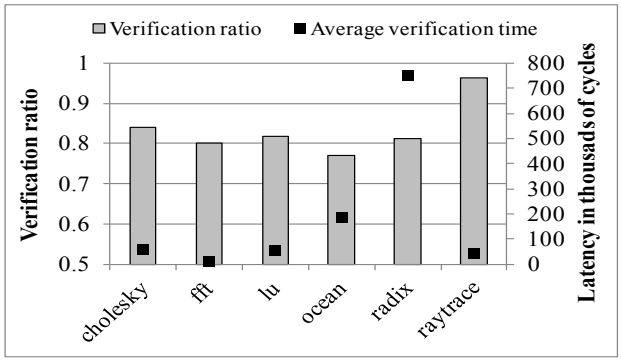

Figure 5: Verification ratio of cache coherence transactions and average latency to verification obtained by SC. Number of cores is set to 4 and cache size to 32KB.

3.2.4 Results

The proposed system was evaluated and the results have been plotted in Figures 6, 7 and 8. The percentage of memory transactions verified by SC, by simply observing the shared bus as well as the average time for verification of each transaction is plotted.

In Figure 6, we have plotted the two metrics for all 6 workloads. In that experiment, the number of cores was set to 4 and the L1 cache size to 32KB. It can be seen that on an average a verification ratio of above 0.8 is observed which is encouraging considering that this is obtained by SC simply monitoring the shared bus. This ratio increases if there are many shared memory transactions among the cores. Every time the shared lines are accessed, as mentioned earlier, a broadcast on the shared bus takes place which enables global verification. This also results in lower verification time. This was observed for the benchmark raytrace, which shows very high verification ratio of 0.96 and a relatively low average verification time of 42K cycles. Locality of reference also plays a role. The higher the locality of reference, the more likely it is that the same cache lines appear on the bus, enabling global or local verification. A study on the effect of number of cores in the CMP and the size of the L1 cache of the cores was also conducted. The results obtained for increasing number of cores in the CMP is plotted in Figure 6 for the workload radix. L1

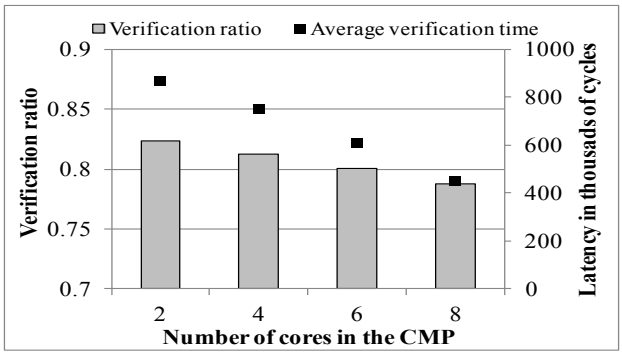

Figure 6: Verification ratio and average latency to verification with an increasing number of cores in the CMP. Cache size set to 32KB and the benchmark is radix.

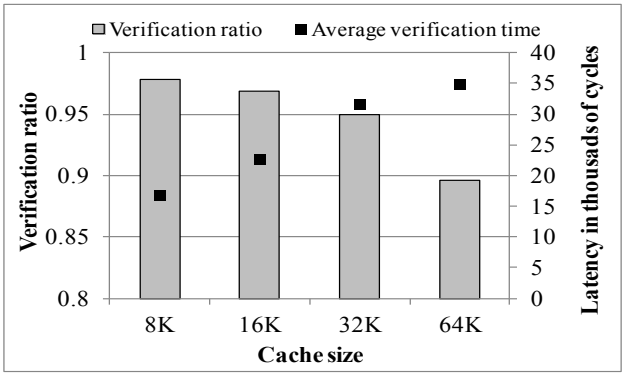

Figure 7: Verification ratio and average latency to verification with an increasing L1 cache size. Number of cores was set to 6 and the benchmark is raytrace.

cache was kept constant at 32KB for this experiment. In line with what can be seen in the figure, in general we observed that with an increase in the number of cores in the CMP, verification ratio decreases and average verification time decreases. We are currently looking into the exact reason for the observed trend. The result obtained with increasing cache size is shown in Figure 7 for the workload raytrace. The number of cores in the CMP was kept constant at 4. It is expected that smaller the cache, more the cache evictions and hence greater the chance for verification. This also results in reduced average verification time, since cache lines reside in the local cache for brief periods. This explains the result plotted in the figure. In all figures, large global verification time is the reason for the relatively high average verification latency.

3.2.5 Discussion

We have seen that SC enables a reasonable fraction of the memory transaction state changes (>80%) to be verified by simply observing the shared bus. However, there is a small fraction of transactions that remain unverified. One way to ensure that all transactions are verified is by SC forcing cores to reveal the state of the cache line if deemed necessary. Such a procedure would incur a performance penalty, since the shared bus would be occupied during the process. Depending on the fault model chosen, it may be possible to prove that verifying the fraction that we do is sufficient to ensure correct operation. This is a part of future work. Size of the SC cache for logging requests is also an important parameter. Depending on the size of the individual caches and the number of cores in the CMP, it may be the case that SC cannot log any more transactions due to limited storage space. In the worst case, all cores have mutually exclusive contents in their caches, forcing SC to store the cache contents of all cores. However, this is rarely the case. We observed that on an average, the storage space required was less than half of the total cache lines in the CMP. However, for an 8 core system with 64KB caches results in a requirement that SC cache be at least 256KB. But this is the case if all transactions must be logged. Once again depending on the fault model chosen, it may be possible to leave out verification of a few transactions. In that case it may be possible for SC to have a smaller cache and yet achieve verification. Another solution to reduce SC cache size and have complete verification is to have one SC for a set of cores on the CMP. All of this is part of future work. We are also currently exploring other potential uses of the SC, such as online performance guard band tuning and automatic core restart by SC on failure.

4. CONCLUSIONS

We have proposed a new approach for online testing of multicore processors running multithreaded programs. We proposed a diminutive core called sentry sore to provide the basic functions of initiating, halting and querying processors. The sentry core adds less than 3% in area to a 4 core CMP, but enables a rich set of testing features. The sentry core initiates and controls online testing of other cores in the CMP. The major advantage of the proposed scheme is its versatility and adaptability in different testing scenarios. We have presented two of the many possible online tests possible. In one of them (review of our earlier work [19]), the sentry core was used to initiate opportunistic DMR and in the other, it was used for cache coherence protocol verification. We found that greater than 80% of the memory transactions can be verified by the SC by simply monitoring the shared bus. In summary, this is a single scheme that fits the solution for various problems.

5. ACKNOWLEDGMENTS

This work has been supported in part by a grant from the GRC and the NSF (Grant No 1985.001). We would like to thank the ACM for giving us the opportunity to compete in the SRC and also the reviewers' for their time and useful feedback.

6. REFERENCES

[1] Austin, T.M., "DIVA: a reliable substrate for deep submicron microarchitecture design," Proceedings of the 32nd Annual International Symposium on Microarchitecture, MICRO-32 , pp. 196-207, 1999.

[2] Rotenberg, E., “AR-SMT: A Microarchitectural Approach to Fault Tolerance in Microprocessors,” Proc. International Conference on Dependable Systems and Networks (DSN) 1999.

[3] Reinhardt, S.. Mukherje, S. S., “Transient Fault Detection via Simultaneous Multithreading,” Proc. IEEE/ACM International Symposium on Computer Architecture (ISCA), 2000.

[4] Srinivasan, J., Adve, S.V., Bose, P., Rivers, J.A., "The impact of technology scaling on lifetime reliability," Proc. International Conference on Dependable Systems and Networks, pp. 177-186, June- July 2004.

[5] Borkar S. Y., “Designing Reliable Systems from Unreliable Components: The Challenges of Transistor Variability and Degradation,” IEEE Micro, Vol. 25, Issue. 6, pp. 10-16, Nov.-Dec. 2005

[6] Renau, J.; et al., SESC Simulator, January 2005. http://sesc.sourceforge.net

[7] The Standard Performance Evaluation Corporation (Spec CPI2000 suite). http://www/specbench.org/osg/cpu2000

[8] DeOrio, A., Wagner, I., Bertacco, V., "Dacota: Post-silicon validation of the memory subsystem in multi-core designs," Proc. IEEE 15th International Symposium on High Performance Computer Architecture, HPCA 2009, pp. 405-416, Feb. 2009.

[9] Meisner, D., Gold, B., and T. F. Wenisch, “PowerNap: eliminating server idle power,” Proceeding of the 14th international conference on Architectural support for programming languages and operating systems (ASPLOS '09), pp. 205-216.

[10] Lee R. B., "Multimedia extensions for general-purpose processors," Proc. SIPS 97 - IEEE Workshop on Signal Processing Systems, pp. 9-23, Nov 1997.

[11] Kumar, R., et al. "Single-ISA heterogeneous multi-core architectures: the potential for processor power reduction," Proc. 36th Annual IEEE/ACM International Symposium on Microarchitecture, MICRO-36, pp. 81- 92, Dec. 2003

[12] Yilmaz, M., Tehranipoor, M., Chakrabarty, K., "A Metric to Target Small-Delay Defects in Industrial Circuits," IEEE Design & Test of Computers, vol. 28, no. 2, pp. 52-61, March-April 2011.

[13] Ogawa, E.T., et al., "Leakage, Breakdown, and TDDB Characteristics of porous low-k silica based interconnect materials," Proc. International Reliability Physics Symposium, 2003

[14] Kim K-S., Mitra, S., Ryan, P.G., "Delay defect characteristics and testing strategies," IEEE Design & Test of Computers, vol. 20, no. 5, pp. 8- 16, Sept.-Oct. 2003.

[15] Sieworek D. P., and Swartz, R. S. (Eds.). Reliable Computer Systems: Design and Evaluation. A K Peters, 3rd edition, 1998.

[16] Benso, A,. Di Carlo, S., Natale, G., Prinetto, P., "A Watchdog Processor to Detect Data and Control Flow Errors," Proc. 9th IEEE On-Line Testing Symposium, p. 144, 2003

[17] Slegal, T.J et al., "IBM's S/390 G5 microprocessor design," IEEE Micro, March - April 1999.

[18] Smolens, J.C.; et al. "Fingerprinting: bounding soft-error-detection latency and bandwidth," IEEE Micro, vol. 24, no.6, pp.22-29, Nov.-Dec. 2004.

[19] Rodrigues, R.; Koren, I.; Kundu, S.; , "An Architecture to Enable Life Cycle Testing in CMPs," Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT), 2011 IEEE International Symposium on , vol., no., pp.341-348, 3-5 Oct. 2011

[20] Saxena, N.R.; McCluskey, E.J.; , "Control-flow checking using watchdog assists and extended-precision checksums," Computers, IEEE Transactions on , vol.39, no.4, pp.554-559, Apr 1990

[21] Lau, E.; et al. 2011. Multicore performance optimization using partner cores. In Proceedings of the 3rd USENIX conference on Hot topic in parallelism (HotPar'11). USENIX Association, Berkeley, CA, USA, 11-11.

[22] Borodin, D.; and Juurlink, B.. 2008. A Low-Cost Cache Coherence Verification Method for Snooping Systems. In Proceedings of the 2008 11th EUROMICRO Conference on Digital System Design Architectures, Methods and Tools (DSD '08). IEEE Computer Society, Washington, DC, USA, 219-227.

[23] Cantin, J,; M. Lipasti, H.; and Smith, J.. Dynamic Verification of Cache Coherence Protocols. In Proc. ISCA Workshop on Memory Performance Issues, 2001.

[24] Fernandez-Pascual, R.; et al.; , "A Low Overhead Fault Tolerant Coherence Protocol for CMP Architectures," High Performance Computer Architecture, 2007. HPCA 2007. IEEE 13th International Symposium on , vol., no., pp.157-168, 10-14 Feb. 2007.

[25] Woo, S.; et al. The SPLASH-2 Programs: Characterization and Methodological Considerations. In Proceedings of the 22nd International Symposium on Computer Architecture, pages 24–36, June 1995.

[26] Chaiken, D.; et al., "Directory-based cache coherence in large-scale multiprocessors," Computer , vol.23, no.6, pp.49-58, June 1990.