Adaptive Online Testing for Efficient Hard Fault Detection

Shantanu Gupta, Amin Ansari, Shuguang Feng and Scott Mahlke
Advanced Computer Architecture Laboratory, University of Michigan, Ann Arbor
{shangupt, ansary, shoe, mahlke}@umich.edu

***Abstract*— With growing semiconductor integration, the reliability of individual transistors is expected to rapidly decline in future technology generations. In such a scenario, processors would need to be equipped with fault tolerance mechanisms to tolerate in-field silicon defects. Periodic online testing is a popular technique to detect such failures; however, it tends to impose a heavy testing penalty. In this paper, we propose an adaptive online testing framework to significantly reduce the testing overhead. The proposed approach is unique in its ability to assess the hardware health and apply suitably detailed tests. Thus, a significant chunk of the testing time can be saved for the healthy components. We further extend the framework to work with the StageNet CMP fabric, which provides the flexibility to group together pipeline stages with similar health conditions, thereby reducing the overall testing burden. For a modest 2.6% sensor area overhead, the proposed scheme was able to achieve an 80% reduction in software test instructions over the lifetime of a 16-core CMP.**

I. INTRODUCTION

The rapid growth of the silicon process over the last few decades has significantly improved semiconductor integration levels. The transistors today are smaller, faster and cheaper than ever before. However, this aggressive downscaling of dimensions in the forthcoming CMOS technology generations poses critical reliability issues. Leading technology experts have begun to warn designers that future generations of silicon technology will be much less reliable than present ones [7]. In addition to extreme process variations, future processors will likely experience failures in the field due to the device wearout over time. To combat such a scenario, future semiconductor products need to be equipped with cost-effective mechanisms to tolerate in-field (i.e., during usage) silicon defects.

There are a host of mechanisms that lead to silicon defects, the popular ones being negative bias temperature instability (NBTI), oxide breakdown (OBD) and hot carrier injection (HCI) [3]. The challenge of tolerating such permanent hardware faults (i.e., silicon defects) encountered in-field can be divided into three requisite tasks 1) defect detection and diagnosis, 2) recovery to a correct system state after a failure and 3) reconfiguration/repair mechanism to prepare the system for future computation. The focus of this work is on improving the efficiency of the first task: defect detection and diagnosis. Recovery techniques (second task) typically employ a checkpointing mechanism to rollback the system after a failure. These checkpoints are created periodically so that in the event of a failure, not much useful work is lost. SafetyNet [23] and ReVive [20] are two good examples of CMP checkpointing solutions. Finally, the solutions for the repair (third task) typically leverage hardware redundancy to replace broken component(s) or in some cases, merely isolate them. Replacement/isolation techniques exist for a range of granularities: cores [2], pipeline stages [12] and modules within a processor [22].

Defect detection and diagnosis mechanisms can be broadly divided into two broad categories: 1) *continuous*: those that constantly monitor the logic blocks for errors and 2) *periodic*: those that periodically check the processor's logic. A few examples of the *continuous* detection mechanisms are dual modular redundancy (DMR) and DIVA [4]. The common idea between all these solutions is to have some sort of redundant computation (in time or in space) to validate the execution. However, all of them impose significant overheads for area, latency, power and energy. Another means for continuous detection is through sensors that can estimate the amount of device level wearout. Although a variety of low level sensors have been proposed [1], [15], [6], they are limited in their capability to accurately predict/detect a failure.

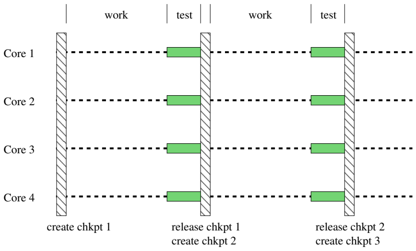


Fig. 1: Periodic testing for fault detection. The vertical stripes represent the checkpoint start/release and the horizontal lines show the progression of threads. At the end of every checkpoint interval, testing is conducted for all processing cores, this is shown as solid horizontal bars.

In contrast, *periodic* detection does not require redundant execution and can give sound guarantees on the fault coverage. These techniques periodically test the system for defects and in case of a failure, they rely on checkpointing and recovery mechanisms. Figure 1 shows snapshot of a system where the tests are conducted at the end of every checkpoint interval. Some of the recent proposals of periodic detection mechanisms are ACE analysis [9] and VAST [13]. Unfortunately, in these proposals, the periodic testing time constitutes as much as 5%-30% of the total system time [9]. This sort of overhead is unacceptable for a high end server that typically apply (virtual machine) consolidation to maintain 100% utilization levels. Even in the case of embedded systems, a great deal of time and energy can be saved by reducing the overhead of periodic testing.

In this work, we propose an adaptive testing framework (ATF) that significantly reduces the overhead of periodic testing in a CMP system. The key insight in ATF is to adapt the testing process to the state of the underlying hardware. For instance, a healthy processor within a CMP can be lightly tested, whereas a weaker counterpart needs thorough testing. In specific, this adaptivity is applicable in three different scenarios:

1) The health of a system varies over its lifetime due to device wearout. Thus, all processors are relatively healthy in the beginning and then deteriorate over time.
2) Manufacture time process variation can form components with differing health levels.
3) Different amounts of stress are experienced by the processors depending up on the workloads assigned.

978-1-4244-5028-2/09/$25.00 ©2009 IEEE

In all the aforementioned cases, the proposed ATF can deliver significant savings on the periodic testing effort while providing the same level of fault coverage. Essentially, our system assesses the health of different processors in a CMP, and appropriately conducts tests. To enable the assessment of processor health, we employ a population of low level sensors [15], [16]. These sensors can predict the mean time to failure (MTTF) with about 25% error for less than a 3% area overhead. We further extend the ATF for application to the StageNet (SN) CMP fabric [12], a highly flexible computing substrate. SN allows arbitrary grouping of stage-level resources from different pipelines to form logical pipelines. We exploit this feature of SN to group together weaker resources from different pipelines and conduct concurrent testing.

The main contributions of this work can be summarized as follows:

1) The proposed ATF introduces the use of low level sensors to guide the online testing process.
2) The ATF achieves a significant reduction in the overhead of periodic testing by adaptively matching the testing process to the underlying hardware's health.
3) An extension of the ATF to StageNet, a flexible CMP fabric, for achieving larger benefits.
4) Lifetime reliability experiments to measure the fraction of time devoted to periodic testing. This setup models process variation, sensor error, device wearout, and testing overhead.

II. BACKGROUND

Here we provide a brief overview of the latest techniques for assessing system health and conducting online tests. Both of these form integral part of the adaptive testing framework proposed later in Section III. Due to the space limitations, we do not provide a background on conventional periodic testing based fault detection [13], [9].

A. *Wearout Sensors*

Wearout monitoring for on-chip devices is a challenging problem and has been an active area of research. Circuit-level designs have been proposed for in-situ sensors that detect the progress of various wearout mechanisms with a reasonable accuracy [15], [18]. A trade-off exists between their accuracy and the area overhead from using them. These sensors are usually designed with area efficiency as a primary design criteria, allowing a large number of them to be deployed throughout the chip for monitoring overall system health. A different approach to sensor design has been to examine the health of on-chip resources at a coarser granularity. Research has involved simple temperature sensors, two dozen on the POWER6 [11], to more complex designs such as the wearout detection unit [6]. These sensors can effectively approximate the useful life remaining in a microarchitectural module.

B. *Online Testing*

The goal of online testing is to detect fault effects, or errors, while the system is in-field. A number of test methodologies exist for online testing, the three important categories being: 1) built-in self test (BIST) based, 2) functional test, and 3) software based self-test (SBST). While BIST addresses the testing problem comprehensively by providing a high fault coverage, it introduces significant hardware overheads [10]. For low-cost embedded systems, such an overhead can not be justified. On the other hand, functional tests use a software program to conduct the testing. The challenge there is the generation of high fault coverage program instructions and automating the process for the same. Most functional testing solutions achieve low fault coverage because they do not consider the RTL structure and are not based on a gate-level fault model (like s-a-fault) [5].

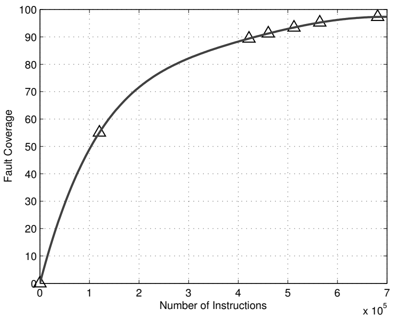


Fig. 2: Fault coverage achieved (in percentage) for varying number of software based self test instructions.

SBST links the instruction-level tests with low-level fault models to achieve good fault coverage while introducing no hardware overhead. SBST starts off by generating module specific deterministic tests patterns and then uses processor instructions as a vehicle for delivering the patterns to module inputs and collecting their responses. The processor simply executes the test program at-speed from the on-chip memory. The test program length is chiefly determined by the module/structure that needs the maximum number of tests. The advantages of SBST are its low cost, ease of application and extensibility. A variety of proposals have been made for SBST [8], [19], [17] with a considerable success. The latest being [17] that reports up to 97.3% fault coverage. The test generation algorithms (for SBST and functional testing) can comfortably trade-off the test size with the amount of fault coverage. Figure 2 illustrates this trade-off between the amount of fault coverage and the number of software test instructions executed for a ARM9-v4 compatible RISC processor using data from [17]. As seen in the figure, the last few percentages of the coverage require the maximum testing effort (number of test instructions).

III. ADAPTIVE ONLINE TESTING

Periodic test based fault detection approaches suffer from the constant overhead of the full test application for all available processing components. In view of the increasing process variation, and the differing amounts of component wearout over the lifetime, an effective optimization is to match the testing thoroughness with the health of a component. We propose an adaptive online testing methodology that builds upon this key insight. Our technique leverages low level sensors to assess the probability of failure in various system components, and appropriately decides the quality of tests applied. The primary benefits from this strategy are the savings in the test time and energy. In addition to the traditional CMP, we extend this adaptive testing philosophy to StageNet(SN) [12], a highly flexible CMP fabric. The advantages of the proposed technique are further magnified while using the SN architecture. The rest of this section provides the details of the adaptive testing framework and discusses its application to a traditional CMP and the SN architecture.

A. *Adaptive Test Framework*

A conceptual illustration of the adaptive test framework (ATF) is shown in the Figure 3. The baseline CMP system is enhanced with the capabilities to assess component health,

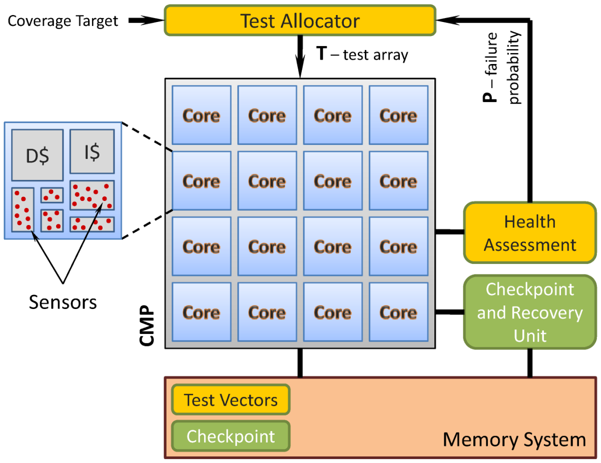


Fig. 3: Adaptive testing framework. A generic CMP system is shown along with the enhancements needed to enable adaptive testing. Health assessment is responsible for gathering sensor readings and producing a fault probability array (**P**). This array is taken up by the test allocator, along with the target coverage, to generate appropriate tests (**T**) for different processing cores.

apply suitable tests, recover from faults (if any) using a checkpointing mechanism [20], and, finally, isolate the faulty core (if and when found). At the end of every periodic checkpoint interval, a health assessment is conducted for all the components in the system, and the corresponding probabilities of failure (**P**) are determined. This array is in turn used by the test allocator to generate suitable tests (**T**) for all components. In the early lifetime, when most of the components are healthy (have low probability of failure), a fewer number of tests are required to make sure the system operates correctly. As the components grow older, their failure probabilities are expected to rise, resulting in a need for more thorough tests. Later in this section, we use this intuitive argument to derive a fault coverage metric (C), that measures the probability of the system to be in a *safe state*. Given a system-wide fault coverage target C, the ATF decides the optimal number of tests required at a per component level (core in this case). This way, testing effort is reduced for the healthy components in the system. The functioning of the important blocks in the Figure 3 is detailed below:

Health Assessment: The lack of knowledge of the underlying component health is the primary reason for applying full tests throughout the component's lifetime. We alleviate this problem by deploying low level sensors that can measure degradation at the transistor level. The primary requirement for such a sensor is to accurately measure the device level characteristics taking into account the process variation and the wearout accumulated over its lifetime. Furthermore, a single sensor won't be enough to provide statistically significant results for an entire core health, and therefore 10-100s would need to be deployed. In such a scenario, low area overhead becomes a favorable feature for such sensors. In this work, for the purpose of illustration, we use the oxide breakdown sensors proposed by E. Karl et al. [15], [16]. These are close to ideal sensors in behavior and have an extremely small area footprint. The results in [16] demonstrate that 500 such sensors are enough to estimate the MTTF (mean time to failure) for an entire chip with less than 10% error. Note that the proposed methodology is not tied to any one sensor type, and a variety of other sensor designs [1], [6] are equally applicable to the methodology proposed here. The ongoing research on NBTI sensors and IDDQ based wearout sensors can also be easily integrated within the ATF. Nevertheless, our choice here was directed by the available data on the accuracy of the oxide breakdown sensor [16].

All cores in our system are enhanced with these sensors. The data from these sensors is gathered and processed in software to generate the MTTF [16], [14] with an error based on the number of sensors. Using the current mean sensor reading, the projected MTTF, and the error in MTTF, we calculate the probability of failure for a core. Note that a higher error in the MTTF estimation translates into a more conservative value of the probability of failure. The discussion of this derivation has been left out in the interest of space. This process is repeated for all the cores in the system to generate the probability of failure array (**P**).

Test Allocator: The task of the test allocator is to prepare suitable test programs for all the cores in the system. At every checkpoint interval, the test allocator is provided with two inputs: 1) a coverage target C (ranges from 0 to 1), and 2) a probability of failure array **P**. Using these two values, it determines the test *fault coverage* (FC) needed by each individual core, such that the coverage target C is always met for that core. Here, the term *fault coverage* implies the fraction of hardware faults covered by test patterns.

For a given core i, and a checkpoint interval t, if the:

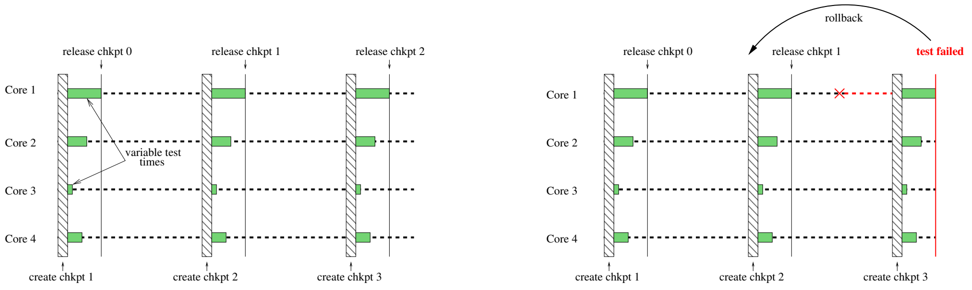
$$\begin{aligned} probability\ of\ failure &= P_i(t),\ and \\ fault\ coverage &= FC_i(t),\ then \\ 1-C &= P_i(t)[1-FC_i(t)] \end{aligned}$$

In other words, the probability of the periodic test not catching a fault $1-C$ in core i is the product of fault occurring $P_i(t)$ and not getting covered $1-FC_i(t)$. From this, we can solve for the required test fault coverage:

$$\begin{aligned} FC_i(t) &= 1-\frac{1-C}{P_i(t)},\ placing\ bounds\ on\ coverage: \\ FC_i(t) &= Min\left\{best\_coverage, Max\left\{0, 1-\frac{1-C}{P_i(t)}\right\}\right\} \end{aligned}$$

Thus, given a coverage target C, a higher probability of failure $P_i(t)$ necessitates an increase in the fault coverage and vice versa. The final equation above also adds bounds to the possible values of the fault coverage, 0 being the minimum and $best\_coverage$ being the best possible coverage using the test generation technique employed. In this work, we propose the use of software based self test (SBST) to conduct the online testing [17]. The advantage of the software based testing is two fold: 1) no hardware overhead, and 2) the fault coverage level is flexible. The proposed methodology in [17] allows generation of test programs to meet different levels of fault coverage. The number of software test instructions are thus tuned on a per core basis to match the fault coverage desired for the same. Figure 2 shows the (single stuck at) fault coverage achievable for an ARM9-v4 compatible RISC core for a range of number of software test instructions. A full set of test instructions is stored in the main memory. The test allocator uses this set of instructions to prepare an array of test programs (**T**) for all the cores. As in the case of sensors, our proposed methodology is not tied to any specific online testing technique.

Checkpoint and Recovery: In the event of a failure, a recovery system is needed to get the system back into an



(a) No fault scenario

(b) Fault occurrence, test failure and rollback scenario

Fig. 4: Checkpointing and adaptive testing for efficient fault detection. Notice that 1) the tests are applied after a new checkpoint is started, and 2) old checkpoint is released once the tests finish successfully.

operational state and isolate the broken component(s). This can be achieved by deploying a CMP checkpoint solution. In this work, we use the ReVive checkpoint system [20]. Revive has a very minimal hardware overhead and maintains the checkpoint in the main memory. The checkpoint interval length can be tuned based on the availability of storage in the targeted system.

Figure 4 shows two scenarios of the ATF in action. The first scenario, illustrated in the Figure 4(a), is for a case with no failures. The horizontal lines show the progression of thread execution, interspersed by the regular checkpoint creations (shaded vertical stripes). The testing phases are shown by solid horizontal bars following each checkpoint creation. The test times vary from core to core, depicting the adaptive nature of the online tests. In this example core 1 runs the longest test (worst health), and core 3 the shortest (best health). The previous checkpoint is released once the tests for all the cores complete successfully. Notice that unlike the traditional practice of testing and then forming a checkpoint (Figure 1), we do the reverse. This design choice is a result of variable test times of the cores in our system. In order to run variable lengths tests on all the cores before a checkpoint, they have to be started at different times. This adds to the complexity of health assessment, test allocation and test scheduling. Thus, in ATF, all tests start concurrently after a new checkpoint is created. Over time, as the cores finish their tests, they are released by the ATF and are made available for job scheduling. However, by creating an additional checkpoint just before running the tests, ATF necessitates two outstanding checkpoints to co-exist while the tests run on the cores. Fortunately, most checkpoint systems, including ReVive, maintain checkpoints as a log of system-wide updates. As the testing phase is very short in length, the additional updates saved in the log due to the second checkpoint are very few, leading to a negligible memory burden. The second scenario, illustrated in the Figure 4(b), shows a case with a failure in core 1 while running a job. The failure is detected during the tests following the creation of the third checkpoint, and system is rolled back to an operational state using the second checkpoint.

System Coverage (SC) Metric: For a system that is periodically tested for faults, there are three distinct categories of events:

1) No failure occurs in the last completed interval
2) Failure occurs and is detected by the test program
3) Failure occurs and is *not* covered by the test program

The first two events maintain the system in the *safe state*, and represent the scenarios where no fault escapes the test. However, the third event is an unwelcome scenario where a fault occurs without being caught. Let us say we have a multi-core chip with n cores. As discussed above, probability of a core i missing a fault in a checkpoint interval t is $P_i(t)[1 - FC_i(t)]$. In other words, the fault occurs and the test is not able to expose it. Continuing along the same lines, the average probability of missing a fault in the entire n core system, within a given checkpoint interval t:

$$Probability\ of\ missing\ fault \quad = \quad \frac{1}{n}\sum_{i=1}^{n} P_i(t)[1 - FC_i(t)]$$

If we sum this over the entire system lifetime, the average probability for the system to miss a fault can be written as:

$$\frac{1}{nT}\sum_{t=1}^{T}\sum_{i=1}^{n} P_i(t)[1 - FC_i(t)]$$

Therefore, the average probability (over the lifetime) of the system *not* missing any faults, i.e. the probability of system being in a *safe state* is:

$$SC = 1 - \frac{1}{nT}\sum_{t=1}^{T}\sum_{i=1}^{n} P_i(t)[1 - FC_i(t)]$$

We refer to SC as the probability of the system being in a safe state. This can also be understood as the effective fault coverage of the system, since it represents the average probability of not missing a fault. We use SC as the metric to specify the target fault coverage in our evaluations.

ATF Summary: The ATF primarily benefits in terms of the test application efficiency. In the early lifetime, when the processing cores are healthy, a lot fewer tests suffice for achieving a given fault coverage target SC. With time, and device wearout, this testing overhead gradually rises. Overall, the application of fewer tests has multiple advantages: 1) more time available for actual job execution, 2) power/energy saving, and 3) low fault detection cost visible to the end user. The intended application of the ATF is to detect permanent faults. Another possible application is its use in systems that have variable reliability modes. For instance, a server can tune the coverage target SC of the system based on the job it is running (higher SC for a financial transaction, and lower SC for a regular web page request).

The discussion of the ATF so far has been in the context of a traditional CMP. The key observation that helps the adaptive online testing is the variation in the health of CMP cores (spatially and temporally). The following subsection applies an extension of this to the StageNet CMP fabric, a highly flexible computing substrate.

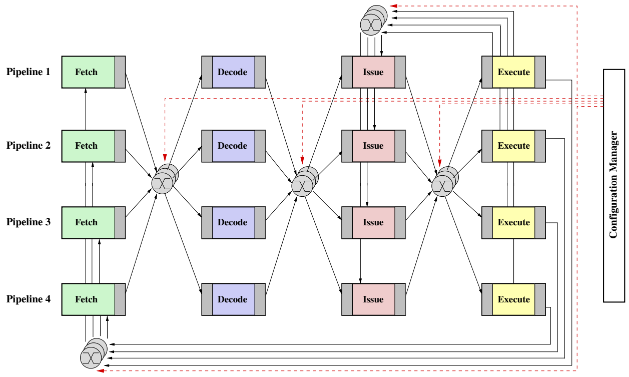


Fig. 5: StageNet fabric with four in-order pipelines woven together using 64-bit full crossbar interconnects. The interconnection configuration is managed by the configuration manager. Within StageNet, logical pipelines, can be constructed by joining any set of unique pipeline stages.

B. Adaptive Testing for StageNet

This section introduces the StageNet (SN) fabric [12], an architectural concept that decouples stages of a pipeline for the purpose of fault tolerance. The real strength of SN fabric is in its ability to isolate broken stages within pipelines. Nevertheless, its flexibility can also assist in forming cores with an even greater variation in their health, thereby magnifying the benefits of the adaptive online testing. The rest of this subsection is broken into two parts, 1) introduction to the SN fabric and 2) application of adaptive testing to the SN.

1) StageNet CMP Fabric: The SN design is a highly reconfigurable and adaptable multi-core computing substrate. It is designed as a network of pipeline stages, rather than isolated cores (Figure 5). A logical core in the SN architecture is referred to as a StageNetSlice (SNS). It is formed by grouping together at lease one pipeline stage of each type. A SNS can easily isolate failures by adaptively routing around faulty stages. In the event of any stage failure, the SN architecture can initiate recovery by combining live stages from different slices, i.e. salvaging healthy modules to form logical SNSs. We refer to this as *stage borrowing*. In addition to this, if the underlying stage design permits, stages can be time-multiplexed by two distinct SNSs. For instance, a pair of SNSs, even if one of them loses its *issue* stage, can still run separate threads while sharing the remaining *issue* stage. We refer to this as *stage sharing*. Thus, a SN system possesses natural redundancy (through borrowing and sharing pipeline stages) and is, all else being equal, capable of maintaining higher throughput over the duration of a system's life compared to a conventional multi-core design.

The SN architecture consists of three prominent components:

a) *StageNetSlice (SNS):* The SNS is a basic building block for the SN architecture. It consists of a decoupled pipeline microarchitecture that allows convenient reconfiguration at the granularity of stages. The decoupling of stages makes the data forwarding and control handling infeasible. Furthermore, the introduction of switches into the heart of a processor pipeline leads to significantly worse performance (4X slowdown over the baseline) due to high communication latencies between the stages. Fortunately, each of these problems can be solved with a few well placed microarchitectural additions (see [12]). With the application of the following optimizations, the performance of the SNS is within 11% of the baseline in-order pipeline.

- *Stream Identification:* Eliminates control hazard.
- *Scoreboard:* Tracks data hazards.
- *Bypass Cache:* Emulates data forwarding.
- *Macro Operations:* Amortizes transfer time of the interconnection network.

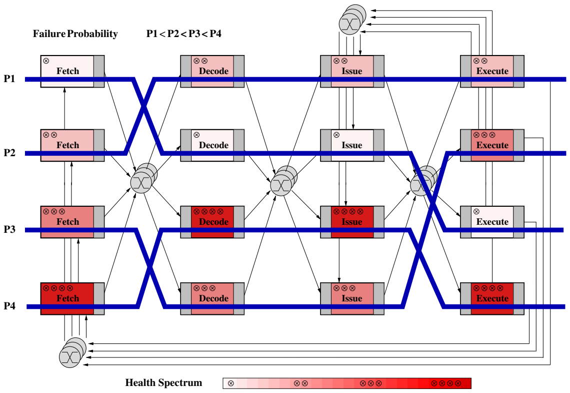


Fig. 6: The shading intensity of stages represents their deterioration. Thus, a darker stage has a higher failure probability and vice-versa. SN flexibility allows connecting stages with similar health, forming logical pipelines.

b) *Interconnection Switch:* The role of the switch is to direct the incoming instruction bundle to the correct destination stage using a routing table. The crossbar switches allows complete flexibility for a pipeline stage at depth *N* to communicate with any stage at depth *N+1*.

c) *Configuration Manager:* Given a pool of stage resources, the configuration manager divides them into a globally optimal set of logical SNSs.

The lifetime reliability results for SN demonstrated nearly 50% improvement in the cumulative work compared to a traditional CMP [12]. Furthermore, the high resiliency of the SN fabric can be leveraged to combat process variation and manufacture time defects, in addition to the wearout failures.

2) Adaptive Testing: At any point in the lifetime, because of the manufacture time process variation and the device wearout, different pipeline stages within the SN fabric would exhibit different amounts of degradation. A snapshot of the SN fabric in Figure 6 shows the varying degrees of degradation between pipeline stages of the system. For the sake of illustration, four health levels are shown from lightest shade (best health) to the darkest shade (worst health). Let us say that the health assessments (that provide probability of failure) map to levels 1-4 of test thoroughness (test fault coverage). In the case of a traditional CMP, the ATF decides the test thoroughness on the basis of weakest component in a core/pipeline. For instance, even if only one stage within a pipeline is badly worn out, ATF for a traditional CMP assigns a thorough test program to that pipeline. Going by this principle, pipeline 1 would apply level 2 test, pipeline 2 - level 3 test, and pipelines 3,4 - level 4 tests. In contrast, the SN can make the testing more efficient by grouping together stronger components separately from weaker components. The bold lines in the figure show the pipeline stages that are combined to formed logical SNSs. First logical SNS (P1) would need to apply level 1 test, P2 - level 2, P3 - level 3 and P4 - level 4. Thus, SN achieves a reasonable amount of test reduction over a traditional CMP.

In order to separate out the stronger pipeline resources from the weaker ones, we sort stages of each type on the basis of their health. For instance, in Figure 6, *fetch* stages are already sorted based on their health (from the top to the bottom pipeline). The stages with equal health ranks are connected to form logical pipelines. These health rankings of the stages can vary over the lifetime depending up on the stress experienced by different stages in the system. Fortunately, the flexibility in the SN system allows it to dynamically segregate stronger and weaker components at will (after every checkpoint interval).

IV. EVALUATION

A. *Methodology*

For evaluating the potential of the proposed approach in reducing the testing overhead, we conduct lifetime reliability experiments. This is required in order to measure the cumulative reduction in the amount of test instructions over the system's lifetime. A CMP is modeled consisting of 16 ARM9-v4 compatible RISC processors. The SN CMP is configured as four group of 4-pipeline wide SN blocks. The operating frequency was set to 1GHz at 130nm IBM process. The systematic and random process variations were modeled using VARIUS [21]. Oxide breakdown (OBD) was used as the representative wearout mechanism with degradation equations from [24], [14]. This choice was motivated by the presence of accuracy data for the low level OBD sensors [15]. A variable number of these OBD sensors were deployed within the cores for the health assessment.

The lifetime experiments are conducted as a series of interval simulations. Each interval simulation updates the sensor readings, and allocates the appropriate size of tests to the cores based on their probabilities of failure (P_i). For a given fault coverage (FC_i) (as determined by the test allocator), the number of test instructions executed is extracted from the data plotted in Figure 2. The maximum achievable fault coverage for testing is 97.3% and is bounded by the SBST scheme that we employ [17]. The presented results use the system fault coverage metric SC as derived in the Section III wherever we refer to coverage target.

B. *Results*

Figure 7 shows the number of instructions used (over the CMP's lifetime) by the ATF, normalized to a baseline CMP system which applies a constant amount of test (given a coverage target). The target system coverage is set to 97.3% (best achievable by the chosen SBST scheme [17]), and the test instructions reported are accumulated over the lifetime. For a 5% sensor error in the health assessment, about 96% of the test instructions are saved while using the proposed ATF. As the number of sensors is reduced (thereby making the reading less accurate), only a more conservative estimate of failure probability is possible, forcing the adaptive system into assigning bigger tests to all system processors. However, even with the higher levels of sensor error, the benefits erode gradually, and the proposed scheme can deliver up to 82% test time saving with 25% sensor error. We believe this point offers a good trade-off between the sensor area overhead (2.6%) and the saving in the test instruction count (82%). Thus, our scheme does not depend on high sensor accuracy levels to achieve test reduction.

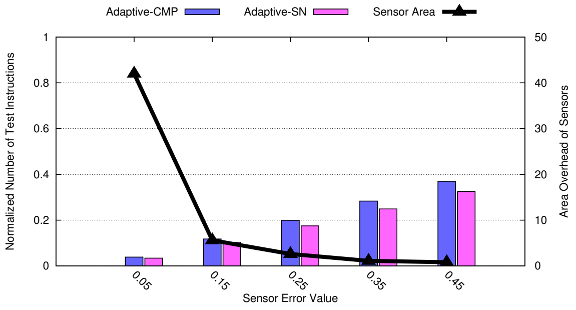


Fig. 7: Number of test instructions for the adaptive online testing in CMP and SN with varying amount of sensor error. The number of test instructions are normalized to a regular CMP with fixed periodic testing. The plot also shows the sensor area overhead used by the proposed approach for health assessment. The coverage target (SC) is fixed at 97.3%.

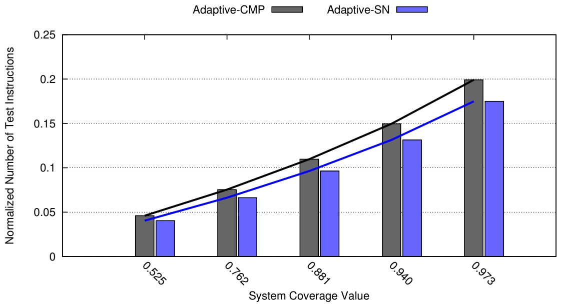


Fig. 8: Number of test instructions for the adaptive online testing in CMP and SN with varying system coverage target (SC). The number of test instructions are normalized to that needed by a CMP with non-adaptive testing.

Figure 8 uses the similar terms as the one before, and presents the test instruction saving for a range of system coverage targets. The sensor error is fixed at 25% for these results. Depending upon the reliability requirements of a system, the coverage target can be dynamically tuned. For instance, a move lower to 88% coverage target can result in an over 90% test instructions saving. The increasing divergence (when going towards higher coverage) between the saving obtained using adaptive CMP and adaptive SN is also noteworthy. We expect the adaptive SN to well surpass the benefits of adaptive CMP in high coverage target scenarios. For future technology nodes, with higher levels of process variation, a SN based system would be capable of extracting even bigger gains by segregating stronger resources from the weaker ones. That way, much fewer pipelines would need a thorough testing.

The result plots so far have presented a cumulative value for the number of test instructions over the entire lifetime, in this next result, we present the data of test thoroughness over time. Figure 9 plots a three dimensional plot with average number of test instruction executed in consecutive simulation intervals for a range of coverage target values. This plot is for the SN system with 25% sensor error. Here, the trend of the number of test instructions over time reveals an interesting behavior of the proposed scheme. For extremely low coverage targets, say 0.5 (or 50%), hardly any test instructions are applied. However, for higher values of coverage target, there is a rhythmic pattern of the test instruction count over the lifetime. The number of test instructions rise to a peak, and then fall-off. This peak formation is representative of a core nearing its time to failure, and then failing subsequently. As a core reaches close to its failure time, the adaptive system ramps up the number of test instructions to guarantee the coverage target. Once the core fails, the system returns to a nominal state since most of the other cores are healthy. There are 16 such peaks in this plot, each representing dying time of a core. Overall average for the number of test instructions is higher later in the lifetime due to the poorer health of many cores in the system. This plot is a clear demonstration of the proposed adaptive testing framework in tuning the testing time with the probability of failure. In contrast, a traditional periodic testing approach will exhibit a flat surface with constant testing intensity.

All the savings that we have reported for the test instructions, can translate into a range of benefits in a target system: 1) *performance* gain from spending less time for test; and 2) *power* and *energy* saving from running fewer instructions. For the system that we simulate (16 core CMP) with a checkpoint interval of 10ms, the performance overheads are 7%, 1.85% and 1.6% for CMP testing, CMP adaptive testing and SN adaptive testing, respectively. According to Revive [20], a 10ms checkpoint interval would require 20MB storage on an average and up to 100MB peak storage requirement. A smaller allocation of storage to the checkpoint mechanism can force the checkpoint intervals to be even shorter, making the testing time even more significant.

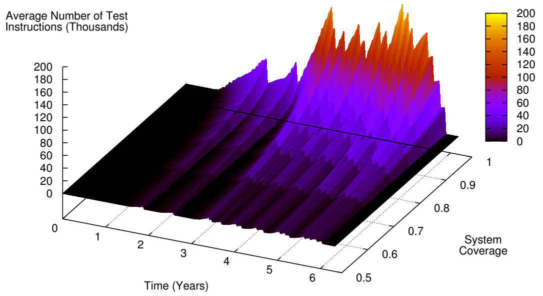


Fig. 9: This plot shows the variation in the average number of test instructions executed in the CMP system over its lifetime for a range of system coverage targets.

V. CONCLUSIONS

With the looming reliability challenges in future technology generations, in-field tolerance to silicon defects will be a necessity in future computing systems. Periodic online testing, although a good fit to this problem, imposes heavy test time overheads. The proposed adaptive test framework significantly reduces this testing overhead. The key insight is to leverage low level sensors to assess failure probability of various system resources, and suitably apply the tests. This way, a healthy system uses a fraction of resources for testing compared to another one nearing its time to failure. Over the lifetime, testing detail is adaptively managed by the proposed solution. The lifetime simulation for a system with 2.6% area devoted to health assessment sensors, resulted in an 80% reduction in the software test instructions while delivering the same fault coverage. We further extend this reduction by 12% when applying the adaptive testing to the StageNet architecture. This test time reduction can translate to varying levels of benefits in power, performance and energy depending up on the attributes of the targeted system. Overall, we believe, that the adaptive online testing offers an economical solution to the challenge of online fault detection.

VI. ACKNOWLEDGMENTS

This research was supported by ARM Ltd., the National Science Foundation grant CCF-0347411, and the Gigascale Systems Research Center, one of five research centers funded under the Focus Center Research Program, a Semiconductor Research Corporation program.

REFERENCES

[1] M. Agarwal, B. Paul, and S. Mitra. Circuit failure prediction and its application to transistor aging. In *Proc. of the 2007 IEEE VLSI Test Symposium*, page To appear, Apr. 2007.

[2] N. Aggarwal, P. Ranganathan, N. P. Jouppi, and J. E. Smith. Configurable isolation: building high availability systems with commodity multi-core processors. In *Proc. of the 34th Annual International Symposium on Computer Architecture*, pages 470–481, 2007.

[3] J. S. S. T. Association. Failure mechanisms and models for semiconductor devices. Technical Report JEP122C, JEDEC Solid State Technology Association, Mar. 2006.

[4] T. Austin. Diva: a reliable substrate for deep submicron microarchitecture design. In *Proc. of the 32nd Annual International Symposium on Microarchitecture*, pages 196–207, 1999.

[5] K. Batcher and C. Papachristou. Instruction randomization self test for processor cores. In *Proc. of the 1999 IEEE VLSI Test Symposium*, page 34, Washington, DC, USA, 1999. IEEE Computer Society.

[6] J. Blome, S. Feng, S. Gupta, and S. Mahlke. Self-calibrating online wearout detection. In *Proc. of the 40th Annual International Symposium on Microarchitecture*, pages 109–120, 2007.

[7] S. Borkar. Designing reliable systems from unreliable components: The challenges of transistor variability and degradation. *IEEE Micro*, 25(6):10–16, 2005.

[8] L. Chen, S. Ravi, A. Raghunathan, and S. Dey. A scalable software-based self-test methodology for programmable processors. *Proc. of the 40th Design Automation Conference*, pages 548–553, June 2003.

[9] K. Constantinides, O. Mutlu, T. Austin, and V. Bertacco. Software-based online detection of hardware defects: Mechanisms, architectural support, and evaluation. In *Proc. of the 40th Annual International Symposium on Microarchitecture*, pages 97–108, 2008.

[10] R. Das, I. L. Markov, and J. P. Hayes. On-chip test generation using linear subspaces. In *Proc. of the 2006 IEEE European Test Symposium*, pages 111–116, Washington, DC, USA, 2006. IEEE Computer Society.

[11] J. Friedrich et al. Desing of the power6 microprocessor, Feb. 2007. In *Proc. of ISSCC*.

[12] S. Gupta, S. Feng, A. Ansari, J. Blome, and S. Mahlke. The stagenet fabric for constructing resilient multicore systems. In *Proc. of the 41st Annual International Symposium on Microarchitecture*, pages 141–151, 2008.

[13] H. Inoue, Y. Li, and S. Mitra. Vast: Virtualization-assisted concurrent autonomous self-test. In *Proc. of the 2008 International Test Conference*, Sept. 2008.

[14] E. Karl, D. Blaauw, D. Sylvester, and T. Mudge. Multi-mechanism reliability modeling and management in dynamic systems. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 16(4):476–487, Apr. 2008.

[15] E. Karl, P. Singh, D. Blaauw, and D. Sylvester. Compact in situ sensors for monitoring nbti and oxide degradation. In *2008 IEEE International Solid-State Circuits Conference*, Feb. 2008.

[16] E. Karl, D. Sylvester, and D. Blaauw. Analysis of system-level reliability factors and implications on real-time monitoring methods for oxide breakdown device failures. In *Proc. of the 2008 International Symposium on Quality of Electronic Design*, pages 391–395, Washington, DC, USA, 2008. IEEE Computer Society.

[17] T.-H. Lu, C.-H. Chen, and K.-J. Lee. A hybrid software-based self-testing methodology for embedded processor. In *2008 ACM symposium on Applied computing*, pages 1528–1534, New York, NY, USA, 2008. ACM.

[18] S. Mishra and M. P. adn Douglas L. Goodman. In-situ sensors for product reliability monitoring, 2006. http://www.ridgetop-group.com/.

[19] A. Paschalis and D. Gizopoulos. Effective software-based self-test strategies for on-line periodic testing of embedded processors. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 24(1):88–99, Jan. 2005.

[20] M. Prvulovic, Z. Zhang, and J. Torrellas. Revive: cost-effective architectural support for rollback recovery in shared-memory multiprocessors. *Proc. of the 29th Annual International Symposium on Computer Architecture*, pages 111–122, 2002.

[21] S. Sarangi, B. Greskamp, R. Teodorescu, J. Nakano, A. Tiwari, and J. Torrellas. Varius: A model of process variation and resulting timing errors for microarchitects. In *IEEE Transactions on Semiconductor Manufacturing*, pages 3–13, Feb. 2008.

[22] P. Shivakumar, S. Keckler, C. Moore, and D. Burger. Exploiting microarchitectural redundancy for defect tolerance. In *Proc. of the 2003 International Conference on Computer Design*, page 481, Oct. 2003.

[23] D. Sorin, M. Martin, M. Hill, and D. Wood. Safetynet: improving the availability of shared memory multiprocessors with global checkpoint/recovery. *Proc. of the 29th Annual International Symposium on Computer Architecture*, pages 123–134, 2002.

[24] J. Srinivasan, S. V. Adve, P. Bose, and J. A. Rivers. The impact of technology scaling on lifetime reliability. In *Proc. of the 2004 International Conference on Dependable Systems and Networks*, pages 177–186, June 2004.