Fault Tolerance in OpenSPARC Multicore Architecture Using Core Virtualization

Kavitha Chandrasekar, Revathi Ananthachari,Sangeetha Seshadri and Ranjani Parthasarathi
Department of Computer Science & Engineering,
College of Engineering Guindy, Anna University, Chennai, India

[1]Abstract- In multicore architecture, fault tolerance issues will increase with increase in number of cores. Size scaling has been considered the source of dramatic performance gains. This scaling has lead to mounting reliability concerns due to increasing power densities and on-chip temperatures. Most wear-out mechanisms that plague semiconductor devices are highly dependent on these parameters. Traditional techniques for dealing with device failures have relied on the coarse-grained replication of structures, ignoring the inherent redundancy in CMP architectures. Here we propose to make effective use of the redundancy of functional units in the 8 cores in OpenSPARC multicore processor to overcome faults by core virtualization. Here we specifically make use of virtualization at functional unit-level instead of core-level virtualization. In addition, some intra core execution resources may be extended to provide functionally equivalent results.

Keywords*-CMP, OpenSPARC processor,Core Virtualization, Emulation*

[1] [1] Kavitha Chandrasekar, Revathi Ananthachari, Sangeetha Seshadri, Ranjani Parthasarathi

1. Introduction- Fault tolerance and reliability are important factors in computer architecture that need to be taken into consideration for addressing hard faults. Faults in-field and manufacturing defects need to be addressed and corrected. If we develop architectures and design methodologies to identify faulty logic and recover by replacing the faulty unit alone, it would be more efficient than replacing the entire core.

To tolerate faults in multicore architecture, dedicated spare (reconfigurable core) cores can be used for recovery in case one of the cores has a faulty unit. A reconfigurable unit inside each of the cores will also help in fault tolerance. However, in these cases the reconfigurable unit will be idle most of the time. The redundancy due to multiple homogenous cores present on the same die is not exploited in the above cases.

In this work, we propose to implement fault tolerance at functionality level by introducing a software layer to carry out fault detection and tolerance. We propose to implement virtualization for effectively making use of partially damaged cores. The virtualization layer will act as a liaison between the operating system and the hardware.

This software acts as a virtualizing agent which can mask failure or functionalities and create the abstraction of logical processor cores capable of performing required execution. These logical cores are obtained by mapping units from partially damaged physical cores. Logically mapping out of failed units and mapping in of same functionality from other cores, the virtualization software can hide failures from application programs without need of having reconfigurable units inside the same core or using dedicated spare cores. We also propose the usage of emulation of a functionality of failed unit [1] by extending the functionality of another unit within the same core.

The rest of this paper is organized as follows. Section 2 examines prior related work in architectural fault tolerance schemes. Section 3 explains in detail the existing architecture. Section 4 and 5 explains our proposed modifications to the system. Section 6 concludes.

2. Related Work:

Many ideas have been developed in order to address hardware failure in computer architecture. Some of them have been explained below.

Bower et al. detected hard faults using an on-line mechanism [5]. A hardware checker is employed that maintains a counter for each error identified. When a threshold value is reached, it is considered to be a hardware fault.

Some design for test (DFT) techniques to detect manufacturing defects and to map out the pipeline portions which are faulty has also been innovated [7].

CASP is a self-test mechanism where the system tests itself using tests stored in non volatile memory while continuing normal operation simultaneously. Hence this will not affect the operations of the entire system [2].

A fingerprinting technique for error detection in cores has also been invented [6]. Architectural state is stored in form of architectural fingerprints. A periodic comparison is done across redundant cores using these fingerprints for detecting errors.

After fault detection, it is necessary to map out the faulty units and map in alternative execution resources. Coarse grain duplication of execution resource has been employed to recover from faults [4].

Migration of thread from one core to another in order to bring about fault tolerance has also been suggested [1].

Some limitations in the prior works are:

- Using redundant units for fault tolerance inside single core does not exploit redundancy due to multiple cores on the same die.
- Usage of dedicated cores as spare cores for fault tolerance is inefficient.
- Migration of entire threads to another core during fault tolerance will be less efficient than mapping of only a functionality which will be only a subset of functionalities in the thread.
- In most cases software intervention is not used for fault recovery.

In this work, we observe that intra core redundancy or usage of spare cores is not the most efficient way to handle fault recovery. Hence, we propose to address fault tolerance by virtualization of cores by mapping of functional units which makes redundancy sharing across cores possible or by emulation [1] of the failed functionality.

3. Existing Architecture:

Overview of OpenSPARC Architecture

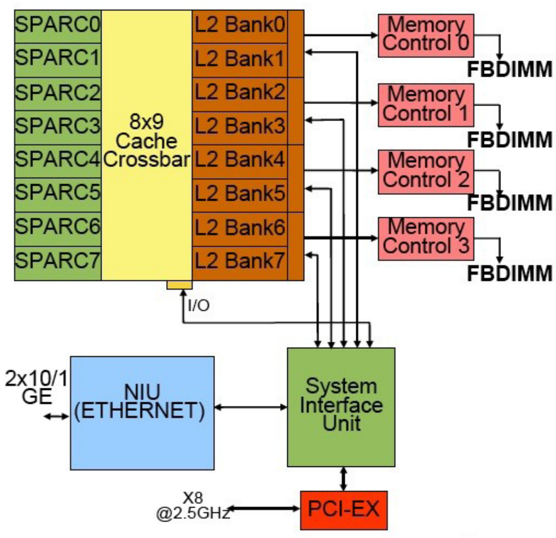

Fig 1: OpenSPARC Architecture [3]

OpenSPARC processor has 8 cores on the same die. The OpenSPARC architecture has reduced temperature gradients which serve to be an effective reliability gradient. By allowing individual threads or even cores to be idled OpenSPARC implements good power control features. Clock gating and power throttling implemented in OpenSPARC reduce the overall power consumption in the cores. OpenSPARC architecture has core level redundancy. This in addition to control over power consumption makes it a reliability-aware processor [3]. Wear out and drift mechanisms enhance the overall chip reliability.

The OpenSPARC architecture also contains effective protection mechanisms for on-chip memory. It supports soft error detection and correction. Single Error Correction/Double Error Detection Error Correcting Codes are used to protect memory arrays greater than 8KB. Parity checking is employed for arrays greater than 2KB[3] .

Multi-bit memory errors within a DRAM device are taken care by Chipkill correction technology which corrects any error contained within a single memory nibble and detects errors in any two nibbles. Checksum is used to detect multi-bit errors in data written in DIMM. In case of a memory error data is recovered using the checksum information. Thus ECC is used to correct single bit errors and checksum and Chipkill for multi-bit errors and DRAM chip failure [3].

Fig 2: Protection mechanisms for On-chip memories [3]

HW Structure	*Protection*	*Recovery*
Instruction Cache Tag	Parity	Hardware
Instruction Cache Data	Parity	Hardware
Windowed Integer Register File	SEC/DED ECC	Software
Floating Point Register File	SEC/DED ECC	Software
Data Cache Tag	Parity	Hardware
Data Cache Data	Parity	Hardware
Store Buffer Data	SEC/DED ECC	Software
Store Buffer Tag	Parity	Software
L2 Cache Data	SEC/DED ECC	Hardware
L2 Cache Tag	Parity	Hardware
L2 Cache Coherence	SEC ECC	Software

4. Proposed Work:

The OpenSPARC processor contains 8 cores on a single die. Each core is capable of running 8 threads. The OpenSPARC architecture includes a para-virtualizing hypervisor layer that exists in between the operating system and the hardware .It is employed to give an illusion of 64 cores to the operating system (8 threads in each of the 8 cores).

Though the OpenSPARC processor acts as a reliability aware processor with enough soft-error detection techniques, it does not tolerate hard-faults. In our architecture we propose to handle the issues involved in detection, isolation and recovery of hard-faults.

The main units in the core that can fail are [8]:

- Instruction fetch unit (IFU)
- Integer execution units (EXU0 and EXU1)
- Load-store unit (LSU)

- Floating-point and graphics unit (FGU)
- Memory management unit (MMU)
- Trap logic unit (TLU)
- Stream processing unit (SPU)

Fault detection:

We propose to use testing techniques specified in CASP [2] for the fault detection phase. There are four phases that have been suggested, namely, test scheduling, pre-processing, performing the CASP tests and resuming normal system operation. The work proposes to select one or more cores for online-test during normal operations of the system. The selected core(s) is then stalled and temporarily isolated from the rest of the system and CASP tests, stored in non volatile memory, are applied. The system then resumes normal operation.

Fault Recovery:

In the recovery stage, the faulty units need to be mapped out, preventing them from further usage. Recent work has also examined ways to apply fine grain micro architectural design for test (DFT) and map out functionality.
To recover from the fault, we propose to introduce a new virtual layer below the hypervisor layer that gives an illusion of eight cores to the hypervisor layer even in case of failure of components of any of the cores. The new layer takes care of mapping out the failed functional unit and mapping in a working redundant functionality from another core and thus presenting a complete logical core to the hypervisor layer as shown in the figure below.
We also propose to implement emulation of functionality that has failed using intra core resources. Software based emulation of failed functionality using resources available inside the same core will also help recover from faults [1].

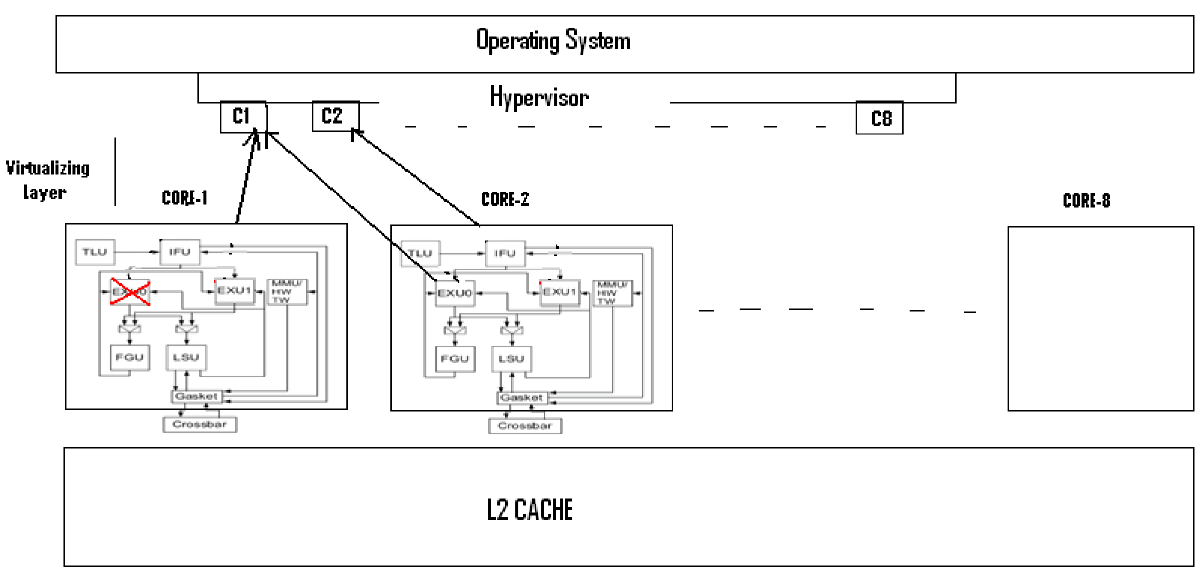

Fig 3: Core-1 has a failed exec unit (EXU0). EXU0 of core 2 replaces the failed unit to form virtual core C1

5. Proposed Implementation:

We propose to implement this fault tolerance technique on the OpenSPARC processor. The IFU of the processor includes a decode unit which decodes the instructions after the fetch stage and the instructions are then assigned to the appropriate units based on their availability. The availability status of every unit in the core is maintained in the decode unit of the core. In addition, we propose to maintain a status of the faulty units in the decode unit of every core and when an instruction to be scheduled on the faulty unit is encountered the decode unit informs the hypervisor layer which takes care of constructing the virtual core with a working unit from another core. Now a new instruction which does not use the faulty unit can be scheduled to work on the faulty core. When the decode unit of a processor itself fails this information is maintained in the hypervisor and the instruction is appropriately decoded by some other core. Either a round robin usage of the other cores' units or a unit of a core that is comparatively lightly loaded can be used instead of the faulty core's unit. The logic for this scheduling of instructions on the cores is maintained in the hypervisor level. All the communication between the processors and the hypervisor layer is effected using the shared L2 caches.

6. Conclusion:

In this paper, we have introduced virtualization of physical cores by introducing a software layer between the hardware and hypervisor layer of OpenSPARC processor to recover functionality in partially damaged CMPs. Redundancy of multiple cores on a single die has been used to efficiently bring about fault tolerance. This helps spread the impact of the failure across cores. Hence, core virtualization is an efficient mechanism as compared to intra-core redundancy or usage of spare cores.

References:

[1] Russ Joseph,"Exploring Salvage Techniques for Multi-core Architectures".Workshop on high performance computing reliability issues(in conjunction with HPCA 2005).

[2] Yanjing Li, Samy Makar, Subhasish Mitra ,"CASP: Concurrent Autonomous Chip Self-Test Using Stored Test Patterns", Proceedings of the DATE , (Design Automation and Test in Europe) March, 2008, published in Issue 16:18:31.0

[3] Ishwar Parulkar, Alan Wood, James C. Hoe, Babak Falsafi, Sarita V. Adve, Joseph Torrellas, Subhasish Mitra," OpenSPARC: An Open Platform for Hardware Reliability Experimentation", published in the Fourth Workshop on Silicon Errors in Logic-System Effects (SELSE), April 2008.

[4] J. Srinivasan, S. V. Adve, P. Bose, and J. A. Rivers. "Exploiting structural duplication for lifetime reliability enhancement". In Proceedings of the 32nd International Symposium on Computer Architecture, June 2005.

[5] [Bower 05]Fred A. Bower 1,3, Daniel J. Sorin2, and Sule Ozev2, "A Mechanism for Online Diagnosis of Hard Faults in Microprocessors", Microarchitecture, 2005. MICRO-38, Proceedings, 38th Annual IEEE/ACM International Symposium on 12-16 Nov 2005. Date Published in Issue: 2005-12-05 08:50:30.0

[6] J.C. Smolens, et al., "Fingerprinting: Bounding soft-error detection latency and bandwidth", IEEE Micro, Nov-Dec 2004, Date Published in Issue: 2005-01-31 08:27:50.0

[7] Schuchman. E.; Vijaykumar, T.N., "Rescue: a micro architecture for testability and defect tolerance",IEEE, Computer Architecture, 2005. Proceedings. 32nd International symposium on Volume , Issue , 4-8 June 2005 Page(s): 160 – 171

[8] OpenSPARC T2-Core Microarchitecture Specification

[9] www.opensparc.net