Characterizing Vulnerability of Network Interfaces in Embedded Chip Multiprocessors

Yong Zou, *Student Member, IEEE*, Yi Xiang, *Student Member, IEEE*, and Sudeep Pasricha, *Member, IEEE*

***Abstract*—In Networks-on-Chip (NoC), with ever-increasing design complexity and technology scaling, soft errors have become a key design challenge. In this work, we extend the concept of architectural vulnerability factor (AVF) from the microprocessor domain and propose a network vulnerability factor (NVF) to characterize the susceptibility of Network Interfaces (NIs) to soft errors. For the first time, a detailed characterization of vulnerability is performed on a state-of-the-art AXI-based NI architecture using full system simulation. Our studies reveal that different NI buffers behave quite differently in the presence of transient faults and each buffer can have different inherent tolerance to faults. Our analysis also considers the impact of thermal hotspot mitigation techniques on the NVF estimation.**

***Index Terms*—Fault tolerance, network interface, networks-on-chip.**

I. Introduction

A major challenge facing the designers of embedded chip multiprocessors (CMPs) today is the increased likelihood of failure due to the rise in transient faults (or soft errors). Soft errors cause the deposit or removal of enough charge to invert the state of a transistor, wire, or storage cell, which can lead to catastrophic system failure. Techniques to ensure fault tolerance have thus become a critical design concern.

Fortunately, not all faults eventually affect the final program outcome. For example, a bit flip in an empty translation lookaside buffer (TLB) entry will not effect the program execution. Based on this observation, Mukherjee *et al.* [1] defined a structure's architectural vulnerability factor (AVF) as the probability that a transient fault in the structure finally produces a visible error in the output of a program. At any point of time, a structure's AVF can be derived via counting the important bits required for architecturally correct execution (ACE) in the structure, and dividing them by the total number of bits of the structure. Such ACE analysis has helped to derive an upper bound on AVF using performance simulation for processor buffers and caches in recent years.

Mirroring the trend with CMPs, the complexity of network-on-chip (NoC) fabrics has also been on the rise. Prior work on NoC reliability has been limited to studying techniques for fault tolerant routing, using redundant resources, and employing coding schemes to protect links and buffers. However, the susceptibility of individual NoC fabric structures to soft errors remains unexplored. In this paper, for the first time, we propose to evaluate the vulnerability of a key NoC component: the network interface (NI) that provides the glue for computing cores to communicate with the NoC fabric. We first develop a sophisticated NI architecture that enables a translation between the processor side AXI communication protocol and the native packet-switched NoC data transfer protocol. We then present the concept of network vulnerability factor (NVF) and perform detailed NVF analysis for subcomponents in the state-of-the-art NI architecture, using a full system simulation with a real OS, and considering the impact of thermal variations. Finally, we illustrate how knowledge of the NVF for NIs can enable low overhead NI implementations with high reliability.

Manuscript received August 28, 2011; accepted March 21, 2012. Date of publication April 04, 2012; date of current version June 15, 2012. This manuscript was recommended for publication by F. Ferrandi.

The authors are with Colorado State University, Electrical Engineering, Fort Collins, CO 80523 USA (email: yong.zou@colostate.edu; yi.xiang@colostate.edu; sudeep@colostate.edu)..

Digital Object Identifier 10.1109/LES.2012.2193553

II. Related Work

The concept of AVF was originally discussed by Mukherjee *et al.* in [1]. There are two main approaches to calculating AVF values: ACE analysis and statistical fault injection (SFI). The former provides a tight lower bound on the reliability level of various processor structures, and has been adopted in many research works on reliability modeling. Fu *et al.* [2] characterized vulnerability phase behavior of four processor microarchitecture structures. Zhang *et al.* [3] performed a similar analysis on SMT architectures. Biswas *et al.* [4] measured AVF for L2 cache structures. None of these or other prior works have explored vulnerability for NoC components.

Several works have addressed the problem of making NoC fabrics more reliable. For instance, Kim *et al.* [5] explored error correction/detection and retransmission techniques to handle errors in links and router logic. Yu *et al.* [6] proposed using Hamming codes to protect router buffers while Frantz *et al.* [7] proposed using TMR together with Hamming codes to address errors. Refan *et al.* [8] proposed adding spare links from a core to its neighboring router to allow packets to bypass any failed switches. None of these existing works attempt to understand the underlying vulnerability of NoC components to guide efforts that can improve reliability, with minimal overhead.

To the best of our knowledge, ours is the first work to explore NI vulnerability in an effort to gain a deeper understanding of how faults in the NI can affect correct program execution. Unlike most prior works, we use full system simulation with a complete software stack that includes real OS to drive the NoC traffic. Therefore our study obtains a more realistic analysis of vulnerability in NI structures.

III. Network Interface Architecture

In this section we present a brief overview of our NI architecture that connects processing cores using the popular AXI protocol from ARM, Cambridge, UK[9]. We developed two NI architectures: a processor NI and a memory NI. A processor

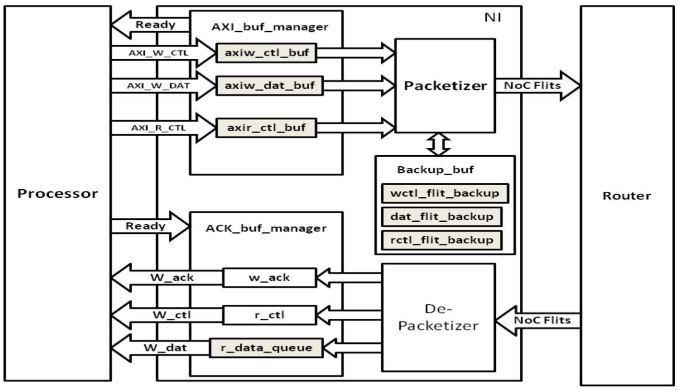

Fig. 1. Processor network interface (NI) block diagram.

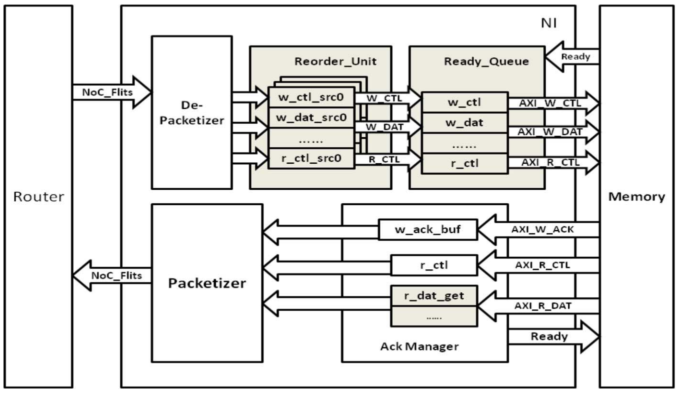

Fig. 2. Memory network interface (NI) block diagram.

NI is responsible for sending write or read data requests to the memory modules. A memory NI interface with memory modules can only accept write or read data requests from other cores, but cannot initiate any requests.

Fig. 1 shows the processor NI architecture. The processor sends write or read signals to the NI through an AXI interface. For a memory write request from the processor, the NI receives *AXI_W_CTL* signals which contain the control information for the request. The NI accepts the request if there is sufficient available data buffer space (*axiw_dat_buf*) inside the NI to save the forthcoming data. After all the data for this write transaction has been stored in the NI buffers, it is transferred to the *Packetizer* unit which converts the request into a NoC packet comprised of multiple flits. The NI supports a low performance overhead end-to-end ACK/NACK flow control which requires flits to be backed up until an ACK signal is received from the destination. Read requests are sent out in a similar manner (except that the outgoing flit does not contain any data). On receiving the read data packet from the NoC, a *De-packetizer* unit is used to convert the packet into data and control signals in the AXI protocol. The de-packetized output is stored in a set of buffers managed by the *Ack_buf_manager* unit.

Fig. 2 shows the memory NI architecture. After a read or write request packet arrives at this NI, the *De-packetizer* unit removes NoC-specific routing information and then sends the data into a *Reorder_Unit* to put in order multiple flits that may possibly be received out of order. This unit has separate buffers dedicated to saving incoming packets from a cluster of cores. After all the data for a memory transaction has arrived, the data is moved into the *Ready_Queue*, which buffers requests to be sent to memory. When the memory is ready, it accesses the queue to obtain pending requests and operates on them. If the request is to write data, an acknowledge signal is sent from the memory after the data has been written. This signal is saved in the *w_ack_buf* buffer. The *Packetizer* unit transforms the acknowledgement into a single ACK flit that is sent back to the source. For a read request, after the data has been read from memory, it is stored into the *r_dat_get* buffer in the NI. A round robin contention resolver is used if a write acknowledge and read data simultaneously request access to the *Packetizer*.

Table I summarizes details of ten buffers (shown shaded in Figs. 1 and 2) from both NI architectures that we chose for our vulnerability analysis.

TABLE I
NETWORK INTERFACE BUFFER DESCRIPTION

Buffer Name	Size (entries)	Width (bits)	Function
Reorder_Unit (*mem*)	32	32	reorder request transaction data
Ready_Queue (*mem*)	10	32	queue pending *mem* requests
r_dat_get (*mem*)	20	32	store received read data from *mem*
axiw_ctl_queue (μP)	32	54	store write request AXI control signals from μP
axiw_dat_queue (μP)	32	46	store write data from μP
axir_ctl_queue (μP)	32	54	store read request AXI control signals from μP
r_data_queue (μP)	5	32	store read data from *mem*
wctl_flit_backup (μP)	10	32	backup write control signals sent from μP
rctl_flit_backup (μP)	1	32	backup read control signals sent from μP
dat_flit_backup (μP)	15	32	backup data sent from μP

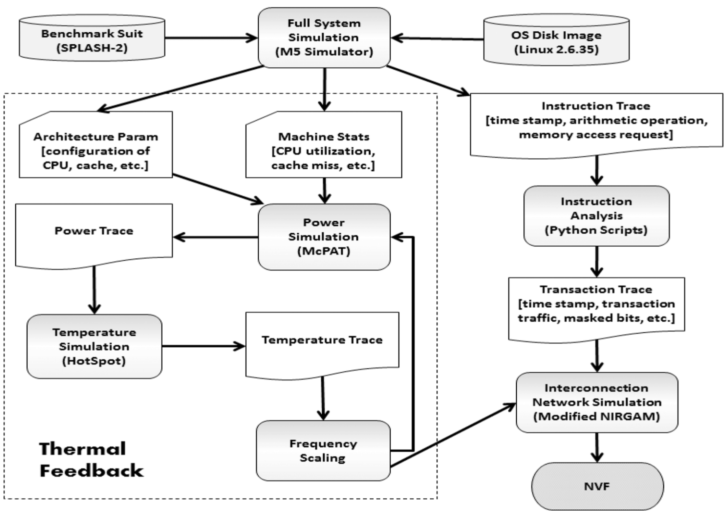
Benchmark Suit (SPLASH-2)
Full System Simulation (M5 Simulator)
OS Disk Image (Linux 2.6.35)
Architecture Param [configuration of CPU, cache, etc.]
Machine Stats [CPU utilization, cache miss, etc.]
Instruction Trace [time stamp, arithmetic operation, memory access request]
Power Trace
Power Simulation (McPAT)
Instruction Analysis (Python Scripts)
Transaction Trace [time stamp, transaction traffic, masked bits, etc.]
Temperature Simulation (HotSpot)
Temperature Trace
Interconnection Network Simulation (Modified NIRGAM)
Thermal Feedback
Frequency Scaling
NVF

Fig. 3. NVF estimation flow for NI buffers.

IV. NETWORK VULNERABILITY FACTOR (NVF)

We define a network vulnerability factor (NVF) that aims to describe the vulnerability of structures found in NoC fabrics. We propose to use the idea of architecturally correct execution (ACE) bits to calculate NVF for the network interface (NI) architecture, which is a key component of all NoCs. ACE bits in our case are the subsets of buffer entries that contain useful data during flit transmission. Any errors that involve these bits will cause a visible error in the final application results. In contrast,

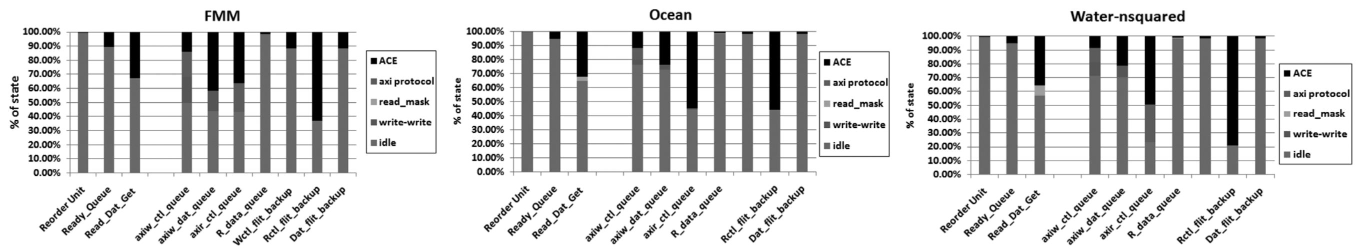

Fig. 4. Breakdown of architectural, microarchitectural states for NI buffers.

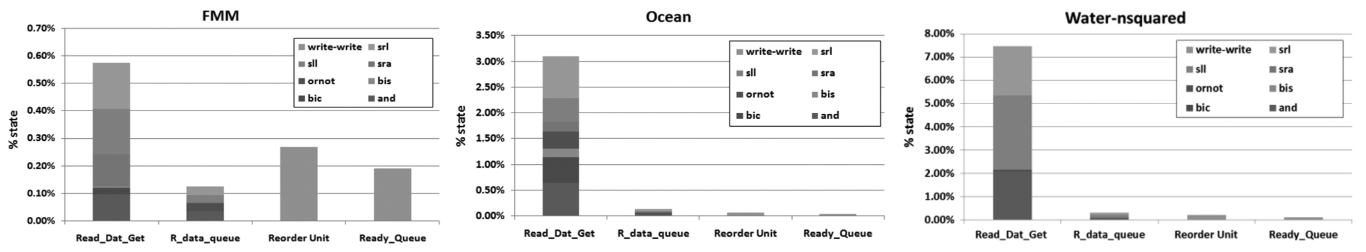

Fig. 5. Breakdown of masking for subset of NI buffers.

unACE bits do not affect the application results, even if soft errors cause changes to these bits. Then the NVF for a hardware component $\boldsymbol{\mathcal{H}}$ of size $\boldsymbol{S_{\mathcal{H}}}$ bits over an analysis window of N cycles can be expressed as

$$NVF = \frac{\sum_{i=l}^{N}(no.\ of\ ACE\ bits\ in\ \mathcal{H}\ at\ cycle\ i)}{N \times S_{\mathcal{H}}}$$

which allows us to calculate soft error rate (SER) for $\boldsymbol{\mathcal{H}}$ as: $SER = \boldsymbol{S_{\mathcal{H}}} \times (FIT/bit) \times NVF \times TVF$, where FIT/bit is the failure in time per bit, and TVF is the timing vulnerability factor encapsulating the fraction of each cycle that a bit is vulnerable.

There are several scenarios that can lead to the presence of unACE bits in a structure that end up making it less vulnerable to soft errors. The major detectable causes of unACE bits that we found to be relevant to NI components in a NoC fabric are: 1) *Idle:* when there are no flits or data saved inside a buffer, even when there are transient errors inside the buffer, as there is no critical information saved inside it at that moment, the final application result will not be affected; 2) *Write-Write:* this scenario corresponds to a write-after-write event. It may so happen that a data that is propagated through the NI buffers and the NoC is overwritten before it is used. 3) *Read Mask:* often, data that is read into a processor may not be used in its entirety. For instance, during an *xor* operation, some bits may not affect the outcome of the operation. These bits can be considered unACE; and 4) *AXI Protocol:* for specific NoC implementations, it is possible that some bits transferred by the AXI protocol (e.g., burst type IDs) may not be relevant.

V. NVF Estimation Flow

Fig. 3 shows the estimation flow we used to calculate NVF for each NI buffer. We booted up Linux OS on an M5 full system simulator [10] to execute SPLASH-2 benchmarks and output traces that recorded details such as time stamps, core IDs, instructions executed, destination addresses, and memory access types, for each memory-related operation. A Python script was used as a trace analyzer to extract memory access behavior and detect masking scenarios. The traces were ported to a cycle-accurate NoC simulator [12] that we enhanced with our NI models, router architectures, and AXI protocols. The trace-driven simulation was then used to obtain NVF for the NI buffers being considered. To account for thermal effects at runtime (e.g., performance slowdown due to hotspots) in our NVF analysis, M5 was used to generate periodic statistics for the open-source McPAT tool [11], to calculate the power trace of the CMP system, which was sent to Hotspot [13] to generate the thermal profile of the die. The thermal trace over time was used to identify points where frequency should be scaled to address hotspot scenarios. This scaling information was fed into the trace driven simulation, to generate thermal-variation aware NVF estimates for the NI buffers

VI. Evaluation Studies

We considered a CMP platform with nine cores and a shared global L2 cache memory, connected together by a 3 × 3 mesh network. The L1 instruction and data cache sizes are set to 64 Kb and the L2 cache is 2048 Kb. Every core runs at a baseline frequency of 2 GHz. The implementation process technology is 32 nm. M5 full-system simulation of parallelized SPLASH-2 benchmarks (*fmm, ocean, water-nsquared*) is used to generate traces that are fed into a cycle-accurate NoC simulator. The detailed execution traces were generated for a span of 4 billion cycles, after fast-forwarding for 750 million cycles to account for an initial warm up period (for OS boot up, etc).

Fig. 4 shows results for the thermal-aware estimation of buffer states. The figures show the percentage of cycles that the buffers contain ACE bits and unACE bits. The NVF for each buffer is essentially the percentage of cycles that a buffer contains ACE bits. It can be seen that NVF varies significantly across buffers from 0.35% to 55.75%. In almost all cases, the buffers hold unACE bits for a majority of the time, with the idle state being the biggest contributor.

A few buffers have notable read and write masking which contributes to the unACE state. Fig. 5 presents read masking effect for a subset of NI buffers with notable masking, broken

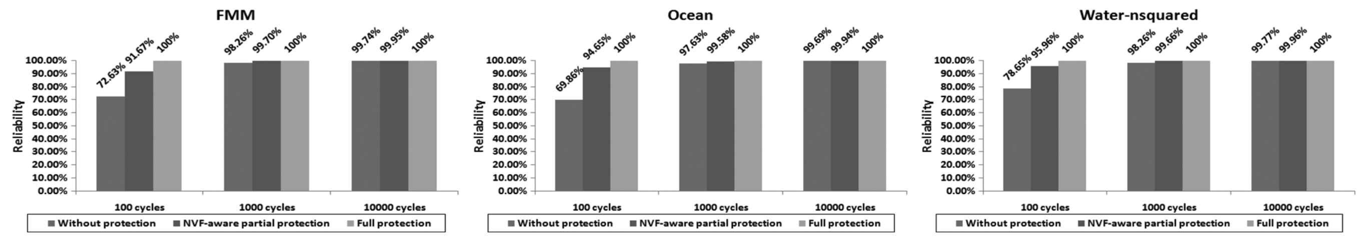

Fig. 6. Reliability vs. fault rate for the three NI design approaches.

down based on the specific instructions that causes the masking, such as shift right logical (*srl*), bit clear (*bic*), etc. The masking effect is most prominent for the *Read_Dat_Get* buffer which holds read data from memory for long periods due to network congestion before packetization and injection. The results provide important insights on how certain NI buffers such as *axir_ctl_queue* and *Rctl_flit_backup* are more susceptible to soft errors than other buffers, because they store critical (ACE) data more frequently. This observation can help guide the optimization of NI architectures for reliable operation.

We conducted a study that explored the reliability and overheads of different NI design strategies. The baseline case is an NI designed without protection mechanisms such as encoding or redundancy. Another approach, employed in recent literature for NoC routers uses redundancy, with spare routers or TMR for each buffer [7] (error encoding/decoding in contrast has greater performance overhead). As prior work has mainly addressed design of reliable routers and ignored reliable NI design, we extrapolate these approaches to NIs, and create a fully protected scenario in which all buffers in the NI are protected using TMR. Such an architecture is expected to have very high reliability. The third approach employs partial protection for a subset of NI buffers guided by our NVF analysis. In this approach, an NI buffer is protected with TMR only if it has a NVF value of 5% or more.

Fig. 6 shows the reliability of the NI architecture designed using the three approaches discussed above. Three different error injection rates were considered, with a single soft error being randomly injected into the NI buffers every 100, 1000, or 10000 cycles. Reliability is measured in terms of how often the incident fault causes failure in the program. It can be seen that NI architectures without protection are particularly susceptible to faults, even with lower fault incidence rates. More importantly, the reliability of the NVF-aware partial protection approach is very close to that for a fully protected approach.

Fig. 7 presents the average power dissipation for the three NI design approaches. The NVF-aware partially protected architecture has almost 50% lower power dissipation than the fully protected approach. The advantage of analyzing vulnerability for the components of the NI architecture thus becomes apparent.

By varying the protection threshold in our proposed approach, a trade-off between reliability and power dissipation can be achieved. A higher threshold value than the 5% chosen in this work would allow sacrificing some reliability to achieve NI architectures with even lower power dissipation.

VII. Conclusion

In this paper, for the first time, a detailed characterization of vulnerability was performed on an AXI-based NI architecture

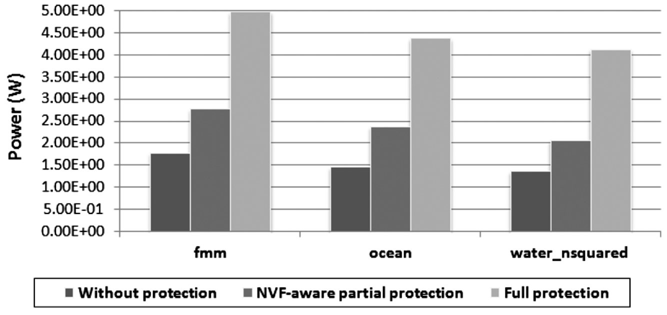

Fig. 7. Average power dissipation for the NI design approaches.

to understand masking effects, using full system simulation and incorporating thermal-aware frequency throttling. Our work has identified dominant masking effects and substructures in NIs that are particularly vulnerable to faults. The insights from this work can enable opportunistic low power protection of NI architectures to meet reliability goals in emerging designs.

References

[1] S. S. Mukherjee, C. Weaver, and J. Emer, "A systematic methodology to compute the architectural vulnerability factors for a high-performance microprocessor," in *Proc. MICRO*, 2003, pp. 29–40.
[2] X. Fu, J. Poe, T. Li, and J. Fortes, "Characterizing microarchitecture soft error vulnerability phase behavior," in *Proc. Int. Workshop Modeling, Anal., Simul. Comput. Telecommun. Syst. (MASCOTS)*, 2006, pp. 147–155.
[3] W. Zhang, X. Fu, and T. Li, "An analysis of microarchitecture vulnerability to soft errors on simultaneous multithreaded architectures," in *Proc. Int. Symp. Perform. Anal. Syst. S/W (ISPASS)*, 2007, pp. 169–178.
[4] A. Biswas *et al.*, "Explaining cache SER anomaly using DUE AVF measurement," in *Proc. IEEE Int. Symp. High-Perform. Comput. Archit. (HPCA)*, 2010, pp. 1–12.
[5] J. Kim, D. Park, and C. Nicopoulos, "Design and analysis of an NoC architecture from performance, reliability and energy perspective," in *Proc. ACM/IEEE Symp. Archit. Netw. Commun. Syst. (ANCS)*, 2005, pp. 173–182.
[6] Q. Yu, B. Zhang, Y. Li, and P. Ampadu, "Error control integration scheme for reliable NoC," in *Proc. Int. Symp. Circuit Syst. (ISCAS)*, 2010, pp. 3893–3896.
[7] A. Frantz *et al.*, "Dependable network-on-chip router able to simultaneously tolerate soft errors and crosstalk," in *Proc. Int. Test. Conf. (ITC)*, 2006.
[8] F. Refan *et al.*, "Reliability in application specific mesh-based noc architectures," in *Proc. Int. On-Line Test Symp. (IOLTS)*, 2008, pp. 207–212.
[9] ARM, AMBA AXI Protocol Specification v2.0 2010.
[10] M5 Simulator System [Online]. Available: http://www.m5sim.org
[11] McPat 0.8 [Online]. Available: http://www.hpl.hp.com/research/mcpat/
[12] Nirgam [Online]. Available: http://nirgam.ecs.soton.ac.uk/
[13] Hotspot 5.0 [Online]. Available: http://lava.cs.virginia.edu/HotSpot/