Statistical Fault Injection

Pradeep Ramachandran†*, Prabhakar Kudva††, Jeffrey Kellington†, John Schumann† and Pia Sanda†

†*IBM Systems and Technology Group, Poughkeepsie, NY.*

††*IBM T. J. Watson Research Center, Yorktown Heights, NY.*

pramach2@uiuc.edu, {kudva,jwellin,johnschu,sanda}@us.ibm.com

Abstract

A method for Statistical Fault Injection (SFI) into arbitrary latches within a full system hardware-emulated model is validated against particle-beam-accelerated SER testing for a modern microprocessor. As performed on the IBM POWER6 microprocessor, SFI is capable of distinguishing between error handling states associated with the injected bit flip. Methodologies to perform random and targeted fault injection are presented.

Keywords – Fault Injection, Soft Errors, SER, SFI

1. Introduction

It is well known that soft errors in logic are a concern in modern VLSI circuits [1]. Recent studies on the IBM POWER6 microprocessor using particle-beam irradiation and full core statistical fault injection (SFI) have shown the importance of microarchitectural "derating" for accurate representation of soft error rate (SER) [3, 15]. There, the remarkable error resiliency of POWER6 was clearly demonstrated by categorizing the destiny of bit flip events in a beam experiment which resulted in over 5,600 fully recovered events including SRAM array events [18]. This paper presents the SFI tool that uses hardware emulation of the system to run many injections while executing realistic workloads. The fast turn-around allows SFI experiments to be practically performed with as many or more bit flips in a few hours as can be had in two days of proton beam experiments. The method has been fully validated against proton beam experiments on a POWER6 microprocessor system. Targeted information can also be gathered with SFI. The calculation speed allows "what-if" questions concerning the resilience of specific circuits, macros, or units within a design. Error handling experiments can be performed to understand the effectiveness of error detection and recovery schemes. This is the first tool that can truly analyze the system effects of SER under realistic conditions. In this paper, we illustrate the use of SFI using POWER6. POWER6 presents an excellent example because of the sophisticated error detection, error handling, and error tracing capabilities [18].

*Pradeep Ramachandran is a graduate student in the University of Illinois at Urbana Champaign. This work was done as a technical co-op at IBM under the Systems and Technology Group.

SFI makes three types of information accessible for the first time:

- Rapid full-system emulation of beam experiments,
- Targeted fault injection in full-system environments, and
- "Cause and effect" tracing of system errors (effect) to the originating bit flip (cause) in a full-system environment.

The purpose of this paper is to present the SFI framework and method, and to illustrate the utility of SFI for unit/macro-level targeted fault injection for detailed SER resiliency analysis.

1.1 Historical Perspective

Fault injection has been used to model SER [6,7,8,17,18]. Simulation techniques that perform fault injections at the RTL level have been limited by speed of computation. They are limited in their ability to capture both the low-level design details (at the latch level) and at the same time model complete error detection and error handling. Hence, few faults are injected and these are typically analyzed in small portions of the design.

Traditional fault injection methods have relied on software simulations for studying the effect of bit-flips. Software simulations use EDA tools, such as NCVerilog and Synopsys, to simulate a hardware design in software. The design is compiled and run in simulation mode by the controlling software, which provides a high degree of controllability and observability of the different elements in the design and provides flexibility to repeat such fault injection experiments on new designs.

Software based simulation mechanisms, however, lack the speed required to perform statistically significant fault injection samples, which may result in drawing inaccurate conclusions about the SER resilience. One way that traditional fault injections dealt with this problem for microprocessor cores has been to perform such injections on smaller components, as opposed to injections on the

entire core. This may result in using test cases that are not representative of real workloads in the full microprocessor environment. Further, they fail to monitor full system behavior and the error handling. The performance of the software based simulation mechanisms limit the size of the designs modeled and the ability to monitor the design both at a fine grain detail (specific latch injection) and at the same time monitor the complete error detection and handling.

2. The SFI Framework

The environment for fault injection is based on hardware accelerated simulation of the entire chip. An RTL model of the system of interest, in this case a POWER6, is synthesized and loaded onto a hardware accelerator. The accelerator then behaves as though it were a hardware chip that implemented the processor model although operating at a much lower frequency. Awan is a programmable acceleration engine which has been used extensively for performance analysis of IBM systems [9]. In Figure 1, the center box shows the Awan hardware emulator which has been loaded with VHDL of the design under test. Awan consists of a large number of programmable Boolean function processors with a highly optimized interconnection network. It uses a massive network of Boolean-function processors, each of which is loaded with more than 100,000 logic instructions. The model of a chip is loaded onto these function processors. Typically, each run through the sequence of all instructions in all logic processors constitutes one machine cycle, thus implementing the cycle-based simulation paradigm.

The throughput of the Awan verification is limited by model load, setup, results analysis, and most significantly by the amount of interaction between the engine and the computing host. Fault injection at specified latches and the monitoring of the fault isolation registers is performed via a communication layer between the Awan engine and the communication host at pre-specified intervals in the cycle simulation of the chip. The fault injection methodology attempts to minimize the communication overhead in order to increase the overall simulation performance. The simulation speed of such hardware can be orders of magnitude faster than software based simulation [9, 10].

A chip processor model, say POWER6, is loaded onto the accelerator and applications and vectors are run directly on the programmed hardware which will behave like a POWER6 in a cycle based simulation mode. The code and testing will run significantly faster than software based event simulation but, as expected, slower than running on the actual POWER6 system. Our SFI model replicated the system configuration that was used for the beam experiments for the processor. To be ready for injection, the POWER6 system model is installed on AWAN and the architectural verification program (AVP) test environment is run. Then, latches were randomly selected for injection among all the latches in the processor core. The boxes to the right in Figure 1 illustrate the set of latches within the HW emulator. Faults were injected at selected locations in the model using a communication interface to the simulation acceleration hardware. The communication between the controlling host and the hardware accelerator is shown to the bottom of the Figure.

The fault may exist for the duration of a cycle (toggle mode) or for a larger number of cycles (sticky mode). The effect of the fault is evaluated by checking the system/processor status registers which flag errors such as checkstops, recoveries and machine errors. Errors not normally visible to the machine can be detected by the Architectural Verification Program (AVP) when they result in incorrect architected state. Faults are injected in a given cycle and then clocked for a fixed number of cycles (500,000) to ensure that all possible effects of the fault including recoveries and the extremely rare occurrences of silent data corruption (SDC) have been identified and serviced. After the fault injection has completed, the model is reloaded from a checkpoint.

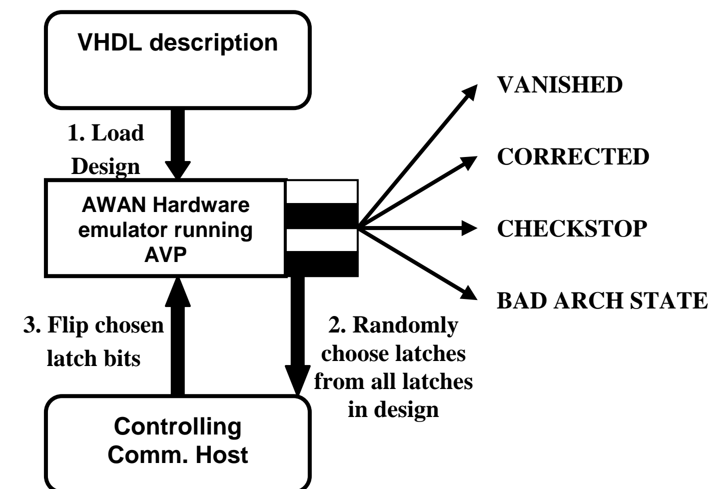

Figure 1: Schematic of SFI framework.

The architectural verification program, AVP, is a proprietary test program which was run on the simulation model to detect microarchitectural errors. The AVP executes numerous small testcases of pseudo-random instructions. Table 1 shows a comparison between the AVP and the SPECInt 2000 benchmark suite. A performance estimation tool was used to derive the

instruction mix and Cycles Per Instruction (CPI) for each of the 11 components of the SPECInt 2000 benchmark and the AVP. For ease of calculation, only the top 90% of the instruction mix was considered for the comparison. The "Low" and "High" columns of Table 1 represent the component that exhibited the lowest or highest CPI or percentage of instructions in that class. The "Average" column represents the mean of all 11 components. It follows from Table 1 that the AVP certainly fits within the bounds of the SPECInt 2000 benchmark and in most cases is close to the average of all components. Note that CPI numbers are approximations and are not truly representative of POWER6 performance.

As the AVP was executed on the POWER6 model, the monitoring environment detected and logged the impact of the events, which were corrected errors, hardware detected checkstops, and incorrect architected state. The corrected and hardware detected checkstops are identified by the checkers states which are part of the POWER6 model. In the rare case of an incorrect architected state, a special flag was raised to make it distinguishable. The AVPs effectiveness at detecting SDC is discussed further in a related work in [3]. The remaining events, which were most of the events, had no effect, or "vanished". The arrows in Figure 1 illustrate the types of events that are categorized using SFI.

	SPECInt 2000			**AVP**
	Low	High	Average	
Instruction Mix (Top 90%)				
Load	18.9%	35.6%	**27.8%**	**29.4%**
Store	6.4%	31.7%	**14.1%**	**23.6%**
Fixed Point	6.2%	35.9%	**22.2%**	**16.7%**
Floating Point	0%	9.1%	**1.2%**	**0%**
Comparison	4.8%	15.1%	**8.8%**	**4.9%**
Branch	6.9%	28.8%	**15.4%**	**14.6%**
CPI	1.2	3.6	**1.9**	**1.8**

Table 1. Comparison of the AVP to SPECInt 2000. (CPI numbers are approximations and are not representative of POWER6 performance)

2.1 Sample size for fault injections

A major concern with any statistical method for fault injection is that of accuracy. Since a typical processor core contains a few hundreds of thousands of latch bits (the simulated model of the IBM POWER6 contains ~350k latch bits across two cores) sampling becomes necessary for realistic simulation times. It is thus important to understand the statistics behind these simulations, to ensure that the inferred results are statistically significant.

In order to evaluate the effect of varying the number of bits flipped, we performed the following experiment. For a given number of bit-flips, say *X* flips, 10 random samples, each consisting of *X* latch bits, are chosen from the entire design. SFI experiments are performed on each of the 10 samples, identifying the number of vanished flips, recoveries, hangs, and checkstops. The mean and the standard deviation of this population are computed. Figure 2 shows the standard deviation as a fraction of the mean of that category for the number of bit-flips that vanished, caused recoveries, resulted in hangs, and invoked checkstops as the bit-flips in each sample (the X value described above) is varied from 2k to 20k bits. The plot shows the standard deviation in the results predicted by SFI as a fraction of the mean in the given category for increasing number of bit-flips. The mean of the different randomly chosen samples for a given number of bit-flips were fairly constant and are hence not shown in the graph. We see from Figure 2 that as the size of the sample grows, the standard deviation as a fraction of the mean in the results considerably falls, reducing the error in estimation.

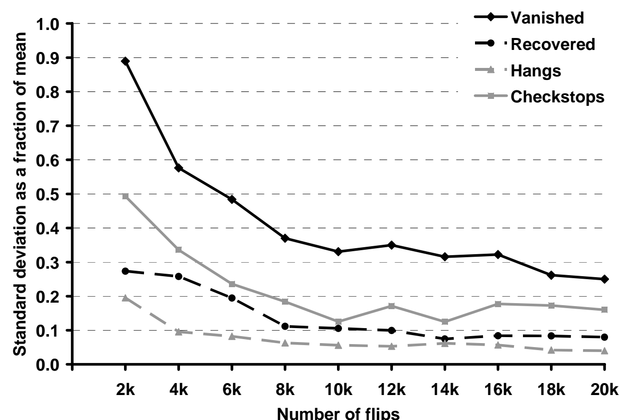

Figure 2: Accuracy of SFI with increasing number of flips.

We further see that at approximately 10k flips, the sample size is large enough that the error estimation is small. This is because the number of flips that result in a given category (for e.g., the number of flips that result in checkstops) for any given sample stabilizes as the sample size (i.e., the number of bits flipped) is statistically significant. Thus, using only one sample of randomly chosen latch bits gives results with significantly high accuracy. For the results in this paper, we use samples of size of ~10% of the total latch bits in the experiments under consideration to further reduce this estimation error.

2.2 Calibration against beam experiment

SFI is a simulation based methodology and therefore a calibration against a real world experiment such as a

proton beam experiment is helpful to validate its accuracy.

Table 2 shows the percentage of flips that vanished, were corrected, caused checkstops outcomes. The close match between the results validates SFI for POWER6. The proton beam experiment is described in previous work [3,15]. The main benefit of SFI over beam experiments is the controllability of injections and the ability to observe the complete RAS response. In addition, multiple concurrent copies of the simulation environment can be run relatively easily, which is not the case with the beam experiments.

Category	SFI	Proton Beam
Total flips	28815	1748
Vanished	95.48%	95.89%
Corrected	3.62%	3.51%
Checkstop	0.90%	0.60%

Table 2: Error state proportions for SFI and Proton Beam experiments.

3. Results

3.1 Micro-architectural SER resilience

Since the beam cannot be focused on individual components, the SER resilience of the individual micro-architectural components cannot be determined using the beam experiment. We performed fault injection experiments on individual micro-architecture components within the full system AWAN model to study the derating that arises from each micro-architecture unit. Figure 3 presents these results.

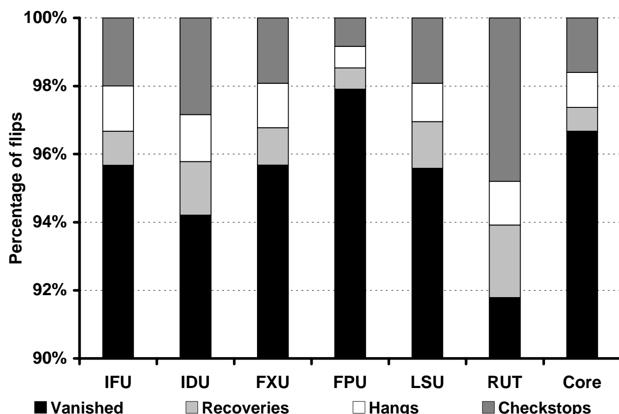

Figure 3: SER of different micro-architecture units.

The bit-flips in the latches of different units are categorized in Figure 3 according to their outcome recorded. Results are shown for the IFU (Instruction Fetch Unit), IDU (Instruction Decode Unit), FXU (Fixed Point Unit), FPU (Floating Point Unit), LSU (Load Store Unit), RUT (Recovery Unit) and Core (Pervasive Logic). The relative outcomes among those recorded during simulation of bit flips namely; Vanished, Recovered, Hangs, and Checkstop are shown. We notice a high rate of architecture-level derating of Soft Errors. However, the outcome of the unmasked faults is dependent on the micro-architecture component and is widely varied across the different units. Bit-flips in a total of 20k latches (out of ~350k total latches) were studied. Owing to the large fraction of bits flipped, the results showed a variation of less than 0.9% as a percentage of the mean for that category, leading to a high confidence in the statistical stability of these numbers for bit-flips in a different set of latches.

Figure 3 shows that a high fraction of the injected bit-flips are masked by the architecture. On an average, 95% of the injected faults are masked by the architecture. However, the variation of this derating across the different units is markedly different. Noticeably, the Recovery Unit (RUT) has the lowest fraction of injected faults that vanish. This is because we have only injected the latches in the RUT. A large portion of the RUT consists of arrays which are protected. Also, the RUT is sensitive to faults in its control logic resulting in only about 92% of the injected faults vanishing without any effect.

From the above data, we infer that the SER resilience of the different units are markedly different and are significantly impacted by the functionality of the different units. However, this data cannot be used to directly interpret the relative vulnerabilities of the different units as each unit has a different number of latches.

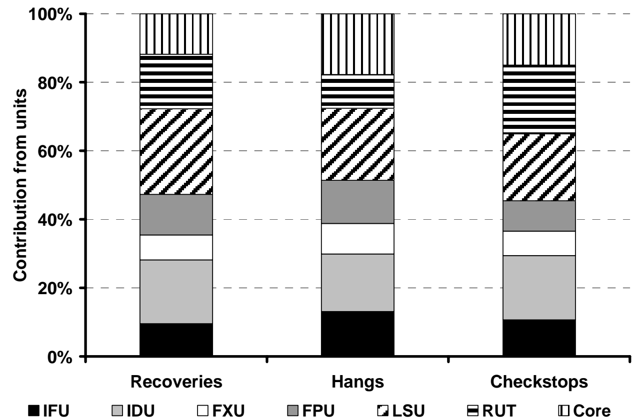

Figure 4: Contribution of each unit to the total percentage of recoveries, hangs, checkstops categories.

Figure 4 shows the contribution from each unit towards the total recoveries, hangs and checkstops seen. The data is calculated from Figure 3 by taking the number of latches in each unit into account.

From figure 4, we see that the contribution towards recoveries is highest from the LSU. The normalized numbers help us to identify the relative vulnerabilities of the different units more accurately.

Since the LSU has the highest number of latch bits that are flipped across all the components, it has the maximum number of recoveries. Additionally, all units have a non-zero contribution to the recoveries due to the existence of checking hardware. For example every unit in the IBM POWER6 core has the ability to detect a fault and invoke recovery through a retry with the help of the Recovery Unit (RUT). Faults in the RUT and the Core (pervasive) logic have the highest individual contribution to the checkstops and hangs seen. Checkers that detects hangs and watchdog timer violations are present in the core pervasive logic and these result in hangs or unrecoverable checkstops. If the Recovery Unit sees a fault when it performs recovery, it results in a core checkstop that can be recovered through software.

This in-depth analysis into the SER characteristics of the different micro-arch components stems from the ability of the SFI framework to perform focused statistically significant bit-flips in a short duration. It is hard to have such a high degree of controllability and observability with real-world beam experiments, making SFI an invaluable tool in SER estimation.

3.2 Identifying the SER of different latches

A typical processor core contains various types of latches, including latches that are used in scan-only mode, pipeline latches, latches that make up the register files, etc. It is important to understand the SER susceptibility of these different latch types to identify areas where additional design effort is needed to improve reliability. Such analyses will help to improve the quality of the processors.

Figure 5 presents the results reflecting the SER of the different latch types. The GPTR (General Purpose Test Register latch) and the MODE latches correspond to latches that are used in a scan-only mode. The REGFILE latches correspond to latches in the various register files while the FUNC latches correspond to the latches in the various pipeline stages. Depending on the location of these latches in the architecture, two latches that are of the same type may have different derating as they perform different architectural functions. Since the previous section examined the effects of these functional differences, in this section, we focus on understanding the differences of the SER resilience between the different latch types. Hence we aggregate the data for each latch type, ignoring the differences in their functions. Approximately 10% of the latches in each scan chain were injected with faults, resulting in significantly low variation across different runs.

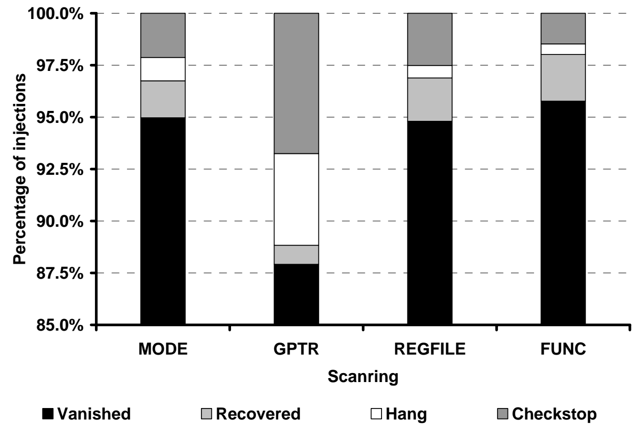

Figure 5: SER of different types of latches.

Figure 5 shows that the scan-only latches have a larger system level impact than the functional latches. MODE and GPTR latches are scan-only latches while the other latches are used in normal operation and are hence read-write latches. The results motivate the hardening of scan-only latches in the core. Since scan-only latches are read-only latches, their data remains persistent through the execution making them highly intrusive to application execution. Bit-flips in functional latches, on the other hand, may be over-written as these are read-write latches. Hence, a flip in a read-write latch (REGFILE and FUNC in Figure 5) is more likely to vanish (about 95% for both cases).

SFI thus helps identify such vulnerabilities in latches early in the design process thus providing opportunities for improved design choices. This is due to the high controllability and observability of the methodology, in addition to performing statistically significant bit-flips to draw the right conclusions.

3.3 Effectiveness of hardware checkers

In order to deal with the heightened SER concerns, modern processors also deploy several forms of hardware checkers that detect the presence of a bit-flip. For example the IBM POWER6 has several such hardware checkers that check for abnormal execution of the hardware.

Owing to the controllability and observability of the design and state latches using SFI, we were able to evaluate the effect of these checkers on the SER of the processor by disabling and enabling checkers in various parts of the core through masking of checkers. Table 3 presents these results.

The flexibility of SFI helps identify the effectiveness of these checkers in the core. Table 3 shows the effect of adding low-level hardware checkers to the core in order to detect abnormal execution caused due to SER. *Raw* represents the SER in the absence of any checkers and

Check represents the SER in the presence of these low-level hardware checkers.

Type	Vanish	Rec	Hangs	Chk
Raw	98.8%	0%	1.2%	0%
Check	95.9%	1.5%	1.1%	1.5%

Table 3: Understanding the effect of checkers

In the presence of hardware checkers, we see an increase in the number of recovery and checkstop events seen by the processor. The number of recoveries increases, indicating that the checkers are very effective at catching the effect of the faults and correcting for them. The reduction in the number of vanished bit flips is due to the fact that some of the errors in the *Raw* mode were not being caught by the processor. These are caught by the checkers and become checkstops and hangs. The checkers are therefore very effective at improving the quality of the design.

4. Conclusions

The ever-increasing SER on modern processors is a major concern for processor designers today. In order to first understand this problem, several techniques that inject bit-flips in today's processors to study SER effects have been proposed. However, one main draw-back of the current techniques is the slow speed of software simulation, which results in erroneous conclusions from using workloads that are not representative of the real world, or from studying too few bit-flips.

This paper presents Statistical Fault Injection (SFI), a tool that uses accelerated hardware based emulation to deal with shortcomings of traditional fault simulation methods. Through the use of full-system models for SER studies, SFI allows characterization of the effects of soft errors on different parts of the modern processor. This paper presents our ability to perform an in-depth study of the RAS characteristics in processors like the IBM POWER6 processor during development through extensive fault injections in the core of the processor and provide feedback to the designers.

As designers devise new techniques to circumvent the problems caused by SER, techniques such as SFI that aid in accurate SER characterization become more important. The flexibility offered by such tools allows designers to perform in-depth studies to understand the derating of these errors by various layers to logic and use this derating to their advantage. Current and future work involves fault injections in the periphery of the core, such as the I/O subsystem, memory subsystem and so on. Future core and system designs will need to be power efficient and therefore require careful analysis of soft error sensitivities to optimally allocate and apportion any additional resources to provide soft error protection.

5. References

[1] S. Mitra, N. Seifert, M. Zhang, Q. Shi and K. S. Kim, "Robust System Design with Built-In Soft Error Resilience," *IEEE Transactions on Computers*, vol. 38, no. 2, Feb. 2005.

[2] M. Mack, W. Sauer, S. Swaney, and B. Mealey, "IBM POWER6 reliability," *IBM Journal of Research and Development. Vol. 51, No. 6,* 2007.

[3] J. Kellington, R. McBeth, P. Sanda, and R. Kalla, "IBM POWER6 Processor Soft Error Tolerance Analysis Using Proton Irradiation," in *Workshop on Silicon Effects of Logic – System Effects (SELSE)*, 2007.

[4] C. Constantinescu, "Neutron SER Characterization of Microprocessors," in *Proceedings of International Conference on Dependable Systems and Networks (DSN)*, 2005.

[5] S. Cakici, P. Sanda, K. Wright, J. Day, S. Swaney, and, E. Cannon, "Proton Irradiation Studies Single Event Upsets in IBM POWER5 System," in *Workshop on Silicon Effects of Logic – System Effects (SELSE)*, 2006.

[6] G. Kanawati, N. Kanawati, and J. Abraham, "FERRARI: A Flexible Software-Based Fault and Error Injection System," *IEEE Transactions on Computer*, vol. 44, 1995.

[7] T. Tsai and R. Iyer, "FTAPE: A Fault Injection Tool to Measure Fault Tolerance," presented at *Computing in Aerospace*, 1995.

[8] N. Wang, J. Quek, T. Rafacz and S. Patel, "Characterizing the Effects of Transient Faults on a High-Performance Processor Pipeline," in *Proceedings of International Conference on Dependable Systems and Networks (DSN)*, 2004.

[9] J. M Ludden, et. al. "Functional Verification of the POWER4 microprocessor and POWER4 multiprocessor systems," in *IBM Journal on Research and Development*, Vol 46, No. 1 Jan, 2002.

[10] M. Wazlowski et al "Verification Strategy for the Blue Gene/L chip," IBM Journal on Research and Development", Vol 49, No. 2/3, 2005

[11] A. Biswas, R. Cheveresan, J. Emer, S. Mukherjee, and R. Rangan, "Computing architectural vulnerability factors for address-based structures," in *Proceedings of International Sympsium on Computer Architecture (ISCA)*, 2005.

[12] X. Li, S. Adve, P. Bose, and J. Rivers, "SoftArch: An Architecture-Level Tool for Modeling and Analyzing Soft Errors," in *Proceedings of International Conference on Dependable Systems and Networks (DSN),* 2005.

[13] H, Nguyen, Y. Yagil, N. Seifert, and M. Reitsma, "Chip-Level Soft Error Estimation Model," *IEEE Transactions on Device and Materials Reliability*, Vol.5, No.3, 2005.

[14] S. Mukherjee, C. Weaver, J. Emer, S. Reinhardt, and T. Austin, "A Systematic Methodology to Compute the Architectural Vulnerability Factors for a High-Performance Microprocessor," in *Proceedings of International Sympsoium on Microarchitecture (MICRO)*, 2003.

[15] P. Sanda et al., "IBM POWER6 Processor Soft Error Resilience," IBM Journal of Research and Development, to appear, 2008.

[16] P. Kudva, J. Kellington, P. Sanda, R. McBeth, J. Schumann, and R. Kalla, "Fault Injection Verification of IBM POWER6 Soft Error Resilience," in *Workshop on Architectural Support for Gigascale Integration (ASGI)*, 2007.

[17] T. Tsai et al., "Stress-Based and Path-Based Fault Injection," *IEEE Transactions on Computers*, Vol 48, No 11, 1999.

[18] K. Reick, P. Sanda, S. Swaney, J. Kellington, M. Mack, M. Floyd, and D. Henderson, "Fault-Tolerant Design of the IBM POWER6 Microprocessor," *IEEE Micro*, to appear, 2008.