Understanding Soft Error Resiliency of BlueGene/Q Compute Chip through Hardware Proton Irradiation and Software Fault Injection

Chen-Yong Cher, Meeta S. Gupta, Pradip Bose
IBM T. J. Watson Research Center
Yorktown Heights, New York, USA

K. Paul Muller
IBM Systems and Technology Group (STG)
Poughkeepsie, New York, USA

***Abstract*—Soft Error Resiliency is a major concern for Petascale high performance computing (HPC) systems. Blue Gene/Q (BG/Q) is the third generation of IBM's massively parallel, energy efficient Blue Gene series of supercomputers. The principal goal of this work is to understand the interaction between BlueGene/Q's hardware resiliency features and high-performance applications through proton irradiation of a real chip, and software resiliency inherent in these applications through application-level fault injection (AFI) experiments. From the proton irradiation experiments we derived that the mean time between correctable errors at sea level of the SRAM-based register files and Level-1 caches for a system similar to the scale of Sequoia system. From the AFI experiments, we characterized relative vulnerability among the applications in both general purpose and floating point register files. We categorized and quantified the failure outcomes, and discovered characteristics in the applications that lead to many masking improvement opportunities.**

Keywords—soft error rate, co-design, chip irradiation, fault injection, high-performance applications

I. INTRODUCTION

Soft Error Resiliency is a major concern for Petascale high performance computing systems. Blue Gene/Q (BG/Q) is the third generation of IBM's massively parallel, energy efficient Blue Gene series of supercomputers. An overview of the BG/Q compute chip is given in [11]. In designing Blue Gene/Q (BG/Q), many mechanisms are deployed to target soft errors including extensive use of Silicon-On-Insulator technology (SOI), radiation-hardened latches [2][18][19], and detection and correction for on-chip arrays, register files and caches. On the other hand, it is well known that many applications are inherently resilient to soft error. The principal goal of this work is to understand the interaction between BG/Q's hardware resiliency features and high-performance applications through chip irradiation, and software resiliency inherent in these applications through application-level fault (AFI) injection experiments.

Soft errors are transient fails that do not damage the hardware, and there is a short window of opportunity to detect them. Soft errors are caused by high-energy particle incidence. These energetic particles have the ability to deposit charge in a transistor body. If the charge is sufficiently large, a non-conducting transistor can become inadvertently conducting for a short period of time. This can have the detrimental effect of inadvertently causing unwarranted bit flips in computation, control and data. Detailed soft error rate (SER) assessment is necessary to quantify a chip's reliability.

A soft error can be a single event upset (SEU) or a multi bit upset (MBU). In today's modern microprocessor designs, architectural states held on-chip, typically in the form of SRAM-based register file and cache, are often protected from SEU in the form of parity detection from SEU, or single-error correct double-error detect (SECDED) error correction code (ECC) from MBU. When a soft error is detected, it can be recoverable or unrecoverable. A recoverable error has typically no impact other than temporary performance degradation. If the soft error is detected but unrecoverable, it leads to a checkstop (no forward progress) in the form of a hang, a partition outage or a system outage. SRAM cells and latches built in SOI have a significantly reduced soft error rate compared to their bulk counterparts. In addition, BG/Q uses radiation-hardened stacked-latch [2][18][19], and detection and correction circuitry to enable high reliability at HPC scale.

Fault injection through accelerated irradiation is an effective way to evaluate the overall soft error resiliency of microprocessors [6][7]. By irradiating a microprocessor chip running an application with high-energy particles, an accurate assessment of fault masking and behavior under faults can be obtained without being biased by the designer expectation or existing fault injection facilities at the level of pre-silicon models. The results can then be analyzed to project actual long term failure rate for larger-scale HPC systems.

Application-level fault injection (AFI) has also been shown to be very effective in estimating resiliency inherent in the applications. Application behavior under faults can be gathered and analyzed by repeating runs and injecting faults (e.g., bit flips) statistically in time and space (e.g., in register) during the application run. Typically, faults are injected in software-based methodology such as using GNU GDB or inserting injection code into the applications.

We performed our proton irradiation and AFI experiments on the BG/Q compute chip over a variety of selected applications [1] as shown in Figure 1.

In order to study the software behavior under soft errors in the register file, we designed our software-based AFI methodology to intentionally bypass hardware detection in the register files. If the hardware detection had not been bypassed, a potential bit flip in the register files would have been detected and corrected by the BG/Q hardware before propagating into software-visible states. Thus, it should be put in perspective that the results of these AFI experiments do not directly extrapolate to silent data corruption (SDC) rate on any hardware.

We selected *AMG2006, UMT, LAMMPS, WUPWISE, QCD* ad *LINPACK* as our software applications because they are widely used in the HPC environments. Of the selected

978-1-4799-5500-8/14 $31.00 © 2014 IEEE
DOI 10.1109/SC.2014.53

applications, *AMG2006, UMT* and *LAMMPS* were chosen from the full-application versions of the Sequoia Benchmark [1]. *AMG2006* (Algebraic MultiGrid) is a parallel algebraic multigrid solver for linear systems arising from problems on unstructured grids. *UMT* (Unstructured-Mesh deterministic radiation Transport) is a 3D, deterministic, multigroup, photon transport simulation program for unstructured meshes. *LAMMPS* (Large-scale Atomic and Molecular Massively Parallel Simulator) is a simulation program for classical molecular dynamics. *WUPWISE* (Wuppertal Wilson Fermion Solver) is a simulation program in the area of lattice gauge theory in quantum chromodynamics [14]. *QCD* (Quantum Chromo Dynamics) is an application for simulating the dynamics of quarks and gluons [15]. *LINPACK* is a benchmark that solves a dense system of linear equations and is widely used to measure performance of HPC systems [16].

In order to rationalize results across experiments, we modified the applications as described below while maintaining their behavior. To focus our application study at the chip level, we modified multi-node MPI (Message Passing Interface) applications to run as stand alone, single chip instances to exclude the effects on MPI software stacks. In order to collect more statistics, all applications were made to execute in roughly one minute, so that many different applications could be run in a short time under similar conditions. The applications are configured to use all 64 threads on the BG/Q chip to maximize the utilization of BG/Q cores and interconnect.

Table I: Applications for the BG/Q compute chip functional irradiation test and AFI (DP: Double-precision; SIMD: Single-Instruction-Multiple-Data).

Benchmark	Threading	FP Characteristics
AMG2006	16 * 4-thread	DP
UMT	16 * 4-thread	SIMD DP
LAMMPS	16 * 4-thread	DP
WUPWISE	16 * 4-thread	DP
QCD	1 * 64-thread	SIMD DP
LINPACK	1 * 64-thread	SIMD DP

Table I provides details of the applications studied. The applications were compiled with the highest performance compiler options available at the time. To ensure that the modified applications are representative of larger-scale, multi-node runs, we collected and computed statistics such as performance throughput and cache miss rates using the BG/Q on-chip performance counters. The statistics are shown in Table II; they are comparable to larger-scale multi-node runs. Table II also shows that the applications cover a representative range of architectural behaviors, namely cache miss rates, instruction-per-cycle throughput (IPC), as well as floating point (FPU) and integer (XU) operations per cycle.

We observe the following results:

- We quantified BG/Q compute core's hardware resiliency in the register files and Level-1 caches.
- We quantified BG/Q application's inherent software resiliency to potential single and multiple bit flips in the register files.
- We identified GPR1, GPR2 and GPR13 to be the most vulnerable over other GPRs.
- We identified signage and exponent bits in the FPRs to be most vulnerable over other bits, and normalization as the mechanism for resiliency in the FPR.
- We identified vulnerable functions in each application.

Table II: Performance counters and statistics.

	AMG2006	***UMT***	***LAMMPS***	***WUPWISE***	***QCD***	***LINPACK***
	Mean Statistics are per core					
Instructions /cycle	0.71	0.52	0.65	0.97	0.28	1.24
Loads/cycle	0.18	0.14	0.23	0.20	0.08	0.48
L1 misses/cycle	0.00	0.02	0.03	0.01	0.04	0.08
L1 miss ratio	0.03	0.11	0.12	0.03	1.36	0.56
L1p miss/cycle	0.00	0.01	0.02	0.00	0.00	0.00
XU instructions/cycle	0.68	0.42	0.45	0.81	0.16	0.64
FPU instructions/cycle	0.03	0.10	0.20	0.15	0.16	0.64
Flops/cycle	0.04	0.36	0.29	0.27	0.92	4.08
	Mean Statistics are per chip (16 user cores)					
L2 hits/cycle	1.38	1.51	1.06	1.37	0.64	0.67
L2 misses/cycle	0.01	0.06	0.01	0.01	0.06	0.01
L2 miss ratio	0.00	0.04	0.01	0.01	0.09	0.02

II. BG/Q 150 MeV Proton Irradiation

A. Background

Fault injection through chip irradiation[6][7][10][12] is an effective way to assess reliability and soft error resiliency of microprocessors and applications within a short period of time, typically a day. By placing a functional chip running applications under irradiation, the resiliency and behavior under errors of the system can be quantified, and the collected data can be extrapolated to model long term mean-time-between-failures (MTBF) for large-scale HPC systems. Highly energetic cosmic rays have the ability to generate a variety of daughter particles in the Earth's atmosphere. Of these, predominantly neutrons with energies > 10 MeV have the ability to cause soft errors in terrestrial microelectronics, since they are still energetic enough to cause secondary particles in a spallation event within the chip. In addition, alpha particles from packaging materials, as well as thermal neutrons in combination with ^{10}Boron, can cause soft errors. However, due to better process control, the contributions from alpha particles and thermal neutrons are diminished today.

The cosmic ray flux depends on the Earth's magnetic field (i.e. latitude, longitude, sunspot activity) and most critically on altitude (atmospheric depth). The resulting energetic neutron flux is customarily quoted for the reference location New York City (40.7 deg N, 74 deg W) at sea level, and is 12.9 neutrons per cm2 per hour. Separate experiments [7] have shown that the effects of the cosmic ray-induced neutron flux are modeled well by a 150 MeV proton beam. In the proton beam experiment, the average proton flux is approximately 1.0 x 10^{10} protons / cm^2 / hour, achieving an acceleration factor of 775 Million. To bridge the modeling gap of the neutron energy spectrum (from 10 MeV to 1 GeV) being replaced by a proton beam, we make use of a previous analysis [7] for the same 150 MeV proton beam as we used for our experiment.

B. Methodology

The experiment was carried out at the Massachusetts General Hospital Proton Therapy Center. We chose to beam a single BG/Q chip in a fan-cooled I/O drawer because it provided maximum flexibility for aligning with the proton beam (Figure 1). The chip we selected for the experiment represented nominal voltage setting (VDD=0.88V). In order to accurately assess the SER of the BG/Q chip, we protected other components of the system such as main memory, power supply and FPGA controllers from the proton beam and daughter particles resulting from collisions between the beam and the BG/Q chip. To that end, we employed lead bricks outside the system (Figure 2) and lighter-weight polymethyl metacrylate (Plexiglas™) shielding inside the system enclosure around the chip, including the DRAM memory system (Figure 3). We aligned the beam such that the entire BG/Q chip, plus an extra 1 mm on all sides, was exposed perpendicularly and uniformly. The proton beam was positioned to enter through the metal heat spreader and exit though the backside (substrate side) of the processor module. Figure 4 shows that proton-sensitive film was used to calibrate the beam to irradiate only the compute chip, confirmed by the shaded area on the film.

In the experiments, we enabled the chip diagnostic mode where all detected-and-corrected errors were recorded in the RAS (Reliability, Availability and Serviceability) logs. This enabled us to validate the design of on-chip fault tolerance features including hardware autonomous detection and correction in the register files. The application output files, diagnostic output files and RAS logs from the runs were saved for post-processing. To determine if applications ran correctly, application output files of completed runs were compared with a known-good output file that was obtained in a test run before the proton irradiation test. In determining if an application had hung, we set a timeout period for the runs at twice the average execution times for each application.

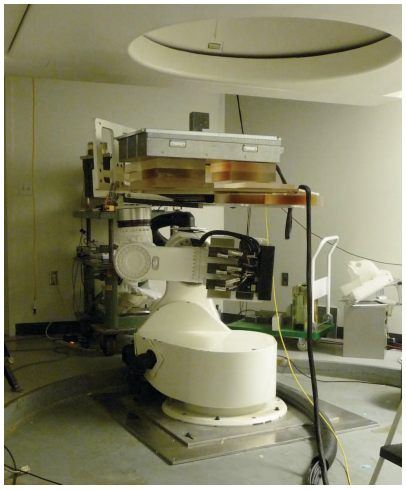

Figure 1: BG/Q IO drawer on stage for beam alignment.

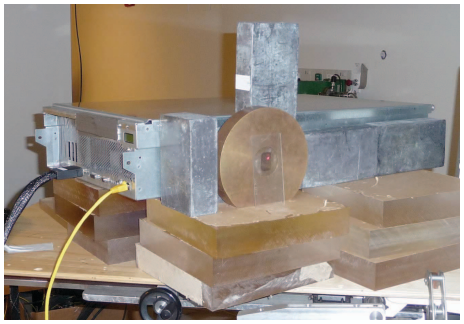

Figure 2: BG/Q IO drawer behind final aperture and lead wall.

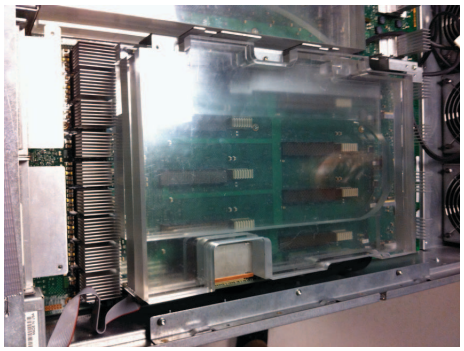

Figure 3: Plexiglas™ was used to protect board components other than the chip.

Figure 4: Proton-sensitive film was used for calibration of the beam.

C. Correctable Errors

On the BG/Q compute cores, many considerations were put into modeling and designing circuitry for soft error resiliency to enable high reliability at the scale of HPC requirements. In the compute core, all the SRAM arrays and registers are protected by either ECC or parity. The compute core also has a lower risk of unscheduled disruption because the architecture circuitry warrants low-latency, autonomous detection and recovery from single-bit upsets. Because L1 data cache and directory has a write-through replacement policy, it can recover from a detected error by copying the correct states from L2 cache. Similarly, the read-only L1 instruction cache and directory can also recover by copying correct states from the L2 cache. As a novel IBM innovation that is first implemented in BG/Q compute core, the FPU and GPR register files are parity-protected but can recover from single-bit errors in hardware: because the register files are already duplicated to provide high bandwidth, the hardware autonomously recovers from a single-bit parity error in one register file array by copying correct states from its duplicate [9].

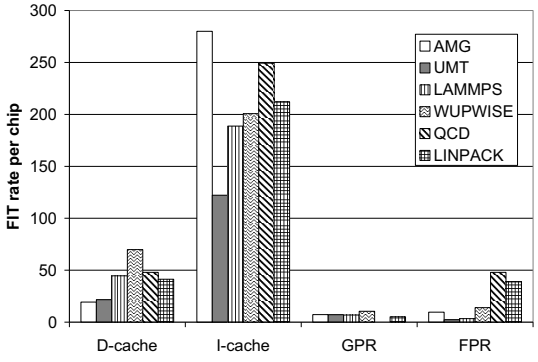

Figure 5: Measured and extrapolated FIT rates of detected-and-corrected errors.

Figure 5 shows the FIT rate per chip (in number-of-failure per billion power-on hours) measured during the chip irradiation test and normalized for New York City sea level for each application. In computing these corrected error rate we collected statistics only from completed runs with no faults such that the irradiation duration can be accurately accounted for with the completion time of each run. In the BG/Q core, both level-1 instruction (I-cache) and data (D-cache) caches are 16kB in effective size.

From Figure 5, even though I-cache and D-cache have the same size, the I-cache detected significantly more bit flip incidents. The reason that I-cache had higher detected error rate can be explained by its cache replacement behavior with respect to D-cache. Bit flips in the caches can be masked if the corresponding physical cache line is replaced where new content are stored over the old, faulty content. In these applications, because data space tends to have a much larger footprint then instruction space, D-cache has a higher replacement rate and therefore higher masking factors. From Figure 5, *LAMMPS* and *WUPWISE* detected more errors in the D-cache because they loaded from D-cache more often, as shown in the performance statistics in Table II, characterized by the higher number of load accesses and L1p misses and relatively lower L1 miss rates. *LINPACK* also had higher detection rate because it had wider SIMD access load width per cache line and its wider access triggers wider parity domain checking. When comparing *LINPACK* to *QCD*, both of which regularly exercised SIMD load instructions, *QCD* had a higher cache miss rate than *LINPACK* and thus a higher masking factor. In FPR, more errors were detected during *QCD* and *LINPACK* because these applications heavily exercised the SIMD register file.

While these correctable errors had no visible performance or functional effects on the applications, it is important to use the results in projecting mean time between correctable errors to validate that the circuitry for detection and correction envisioned at design time properly functions and provides the expected value to the system. Assuming the application mix used in this study, we extrapolated these results to predict the mean time between correctable errors in the GPRs and FPRs of sixteen cores for a system similar to the scale of Sequoia to be 16 days. Similarly, when accounting for instruction cache, data cache, and GPR and FPR register files, the mean time between correctable errors is predicted to be 1.5 days. The high rates of detected-and-corrected errors significantly validated the necessity to include autonomous hardware detection and recovery at the cost of design effort, silicon area and power.

III. Application-Level Fault Injection (AFI)

A. Background

Mukherjee et al. introduced the concept of Architectural Vulnerability Factor (AVF) and ACE Analysis as a means to estimate SER early in the design cycle [17]. This metric evaluates the masking provided at the microarchitecture level. In order to accurately estimate the SER of any given system it is critical to evaluate the masking provided at the application level. In this paper we present a framework to understand the application-level masking using the debugging utility provided in systems.

Ref. [20][22][23] and [24] use circuit-level simulation to profile propagation of low-level errors, particularly due to single-event upset (SEU) in latches and flip-flops, into high-level errors in architectural states. Ref. [23] proposes generation of instruction sequences that mimic the propagation of errors, and [22] focuses on generation of error propagation matrices to drive high-level simulations. Unlike [20][22][23] and [24] which use architectural simulators for high-level error injection, our approach is different because we run the applications at native hardware speed, resulting in efficient profiling of error propagation and application-specific error outcomes on a large number of AFI runs of large scientific programs. Our use of debugging utility also enables profiling of precise application information such as

program counter, program stack and symbol table at the point of injection.

Ref. [20] and [25] quantitatively show inaccuracies in the high-level error injection experiments when modeling flip-flop errors. Our work is fundamentally different because we target soft error in the SRAM-based register files. Our work also differs from prior works in that we compare the results from our high-level error injection experiment to our proton irradiation experiment, whereas [20][22][23] and [24] study flip-flop errors with architectural simulations and compare the results to their respective circuit simulations.

There is also prior literature related to correction of soft errors. Ref. [21] proposes selective error correction code (ECC) for SRAM-based registers based on life time usage. However it does not capture the additional masking due to algorithm and floating-point format as we observe in our AFI experiments. Ref. [26] and [27] propose application-based detection and correction of soft errors through analysis of the algorithms. Our work differs from these prior works in that our proposal targets a general, automated approach that relies less on application programmers.

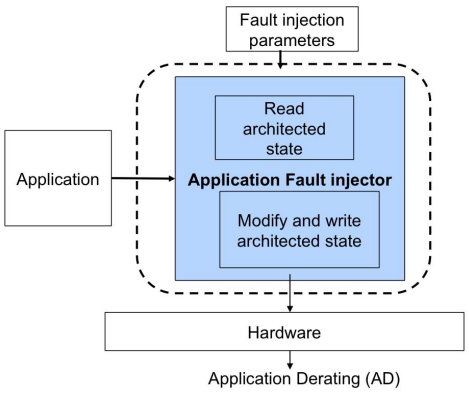

Figure 6: A high-level description of the Application-fault injection Framework.

In this paper we propose a framework which runs on the native hardware and can inject faults into application in a non-intrusive manner. Figure 6 presents the high-level picture of the framework highlighting the different components of this framework. The proposed application-level fault injection (AFI) enables the detailed understanding of application masking using statistical fault injection experiments while the application is executing on native hardware. The application fault injector has the ability to examine and modify the core image and the registers of the application being analyzed.

B. Methodology

By analyzing key statistics on injection points (e.g., register and memory locations, execution time stack, instruction mix), the behavior of the applications under fault can be better understood. By comparing the error rates between different applications, more error tolerant algorithm and systems can be derived in the future.

In the proton irradiation experiment, we obtained the error rates detected and corrected by the BG/Q hardware circuitry. In order to understand the benefit of these detection and correction circuitry, we need to analyze the behavior of the applications without them. Therefore, in the AFI experiments, we intentionally bypass the existing on-chip protections in the hardware. Because the software-based AFI intentionally bypassed hardware detection in our specially controlled experiment setup, it should be put in perspective that the results of these AFI experiments do not directly extrapolate to failure rate on any hardware detection technique or on any particular machine including BG/Q and the chip irradiation results in section II.

The method takes advantage of available program debugger software in providing the fault injection capability. The Compute Node Kernel (CNK) deployed on BG/Q supports a remote debugging facility, *gdbtool.* To provide GDB support for the compute node, the GDB server, *gdbtool*, is first started for each rank to be debugged. The tool can be started by running the *start_tool* command. When using the start_tool command, the *gdbtool* server is attached to a running job at a random time during its execution. Once the *gdbtool* attaches to and stops the execution of the application, any given architected state can be read and modified. The *gdbtool* server then detects and the job continues its execution with the injected fault.

In our experiments, we target pseudo-random fault injections to corrupt the register-state of a workload (or application program) running on the BG/Q compute node. We target register states because unlike the Level-1 read-only instruction cache and write-through data cache that can recover by loading correct states from lower-memory hierarchy, a detected error in the register file cannot be easily recovered and often requires duplicate (as in the case of BG/Q) or adding ECC which often affects critical timing path in the circuit designs. Therefore, we seek to identify application attributes that could improve power and area efficiency of error detection and correction in the register files for future designs.

Pseudo-random fault injection experiments are made into the architected register space in each controlled experiment for the corresponding benchmark. Each such injection can lead to one of five outcomes:

i. **Vanished** - The bit that is flipped via the injection has no effect on the final program output. In this paper, it is used synonymously with the term "Masked".

ii. **Mismatched** - The injected bit-flip results in a mismatch in the program output when compared to a known-good run.

iii. **Mismatched but passed** – Similar to "Mismatched", the injected bit-flip results in a mismatch in the program output. However, the application's build-in algorithm determines that the obtained solution is within tolerable range of the desired solution.

iv. **Crashed** - The injected error results in the operating system terminating the program due to a detected runtime error (e.g., *divide-by-zero*, *segmentation fault,* etc.).

v. **Hung** - The injected bit-flip results in a hung state, where there is no forward progress of the program execution.

In our definition, a *failure* occurs when an injected fault is not masked and results exclusively in one of the outcomes ii, iii, iv and v listed above.

A soft error can be a single event upset (SEU) or a multi bit upset (MBU). In addition to injecting SEU through AFI, we also studied the effects of MBU. Conventional circuit-level detection such as parity and error correction code (ECC) protects against odd number of bit flips, whereas ECC or parity with bit-wise interleaving can also detect two-bit flips at relatively reasonable circuitry cost. The next MBU case of interest is a four-bit flip. Therefore, we evaluated the effects of four-bit flips in our AFI MBU experiments to observe the differences in outcome distributions between single and four-bit flips.

C. Register File Injections

The AFI methodology enabled us to analyze the resiliency of a given application. The application resilience is directly correlated to the temporal and spatial usage of the architected state. Figure 7 shows the fault-distribution of various applications for injection into the 64-bit general-purpose and 64-bit, four-way SIMD floating-point registers. General-purpose registers (GPR) show higher vulnerability as compared to floating-point registers (FPR) for all the benchmarks. This phenomenon can be attributed to the use of general-purpose register in address calculations along with arithmetic operations. For a given application, differences can be observed for the different functions and procedures.

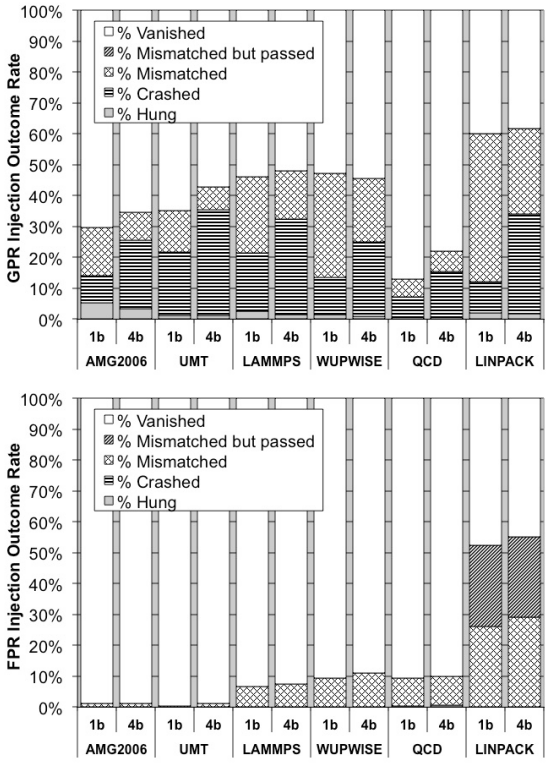

Figure 7: Outcome distribution resulting from AFI injection into GPR and FPR.

Figure 7 shows the fault distribution resulting from the AFI experiments. For each application, the upper graph refers to single and four-bit injections into the general purpose register file (GPR), and the lower graph refers to single and four-bit injections into the floating point register file (FPR). From Figure 7, *LINPACK* produced the most Mismatches (i.e., outcome ii and iii). In all the applications, going from SBU (1b) to MBU (4b) did not significantly but only slightly increased the overall failure (i.e., not masked) ratio. All applications also produced a higher percentage of crashes (i.e., outcome iv) when going to MBU while Mismatches decreased by similar percentage points. This phenomenon can be understood as follows: if an effective address is changed by a four-bit flip rather than single, it is more likely to become out of bound with the array or TLB page boundaries. Another phenomenon consistent across all applications is that GPR injection caused more failure cases in terms of hung, crashes and mismatches than FPR injection due to the higher masking rate of FPR operation. We further explore the masking in the FPR in Section III.D. In the FPR fault injection experiment *LINPACK* is most vulnerable. *LINPACK* is roughly 50X more sensitive than *AMG2006* and *UMT*, and 5X over other applications, although its built-in validation test is also able to self-check for correctness of the results. *AMG2006* and *UMT* are highly resilient to FPR faults – although the root cause of this resiliency is not understood yet.

In *LINPACK*, a validation test was included as part of the benchmark to check for acceptable accuracy of the computed results. The test, called a *residual test,* checks for magnitude differences between the computed solution and the ideal solution. In our experiments, the validation test would pass if the scaled residual computed within the application is less than 16.0. As discussed in section III.B, because our applications were performed under controlled conditions, a run would be classified as mismatch if its results are different from the known-good run even if the results are accepted by the residual test. However, in the interest of application analysis, we differentiate two types of mismatches for *LINPACK* results in Figure 7: *Mismatched but passed* which shows the percentage of runs passing the residual test despite having produced different output, and *Mismatched* which shows the percentage failing the residual test. Figure 7 shows that almost all of the GPR injection runs failed residual tests, whereas nearly half of the FPR injections still passed the residual tests. This phenomenon shows that *LINPACK* is much less tolerant to faults in GPR than FPR.

As shown in Figure 7, the FPR were less vulnerable compared to the GPR; with around 90% masking on the average across all the registers. This can be attributed to the nature of usage of FPR. Most of the high-performance computing check for error-bounds and slight error in the values due to rounding, therefore errors caused by flipping the lower bits has less effect on the correctness of the overall computation. This is in contrast to the GPRs, which are primarily used in address calculation and increment values.

D. Masking in the floating point format

The IEEE 754 double-precision binary floating point format describes a 64-bit representation of 1-bit signage, 11-bit exponent and 52-bit fraction, as is supported in the BG/Q FPU. Theoretically, although the sign and exponent have fewer bits, they should have a more significant impact if flipped because they will change the magnitude and direction of the represented number. The fraction, although larger in bit count, may have a smaller impact on average because only a few bits represent the significant numbers in the representation. The significant bits are not easily identi-

fiable because their positions depend on the normalization of the number and the precision of the represented numbers. However, when a floating point number is normalized, a bit flip in its fraction is less likely to impact the number representation because the bit flip is more likely to impact the less significant bits in the fraction. Figure 8 shows the probability of failures if an error is injected into one of the portions of an FPR register in the IEEE 754 representation. To clarify, the failure probabilities in Figure 8 do not account for how probable whether signage, exponent or fraction bits would be flipped, which would depend on the number of bits for each portion and how they are computed, but they represent the probability of failures given that a bit flip is injected into each portion. From Figure 8, *LAMMPS* and *WUPWISE* had higher failure ratio when their exponent bits were flipped, but were not sensitive when its signage bit was flipped. The fraction bits, although as expected did have a significant impact more than the exponent bits, still represented a significant probability of failures. In all three cases, bit flips did not lead to crashes except a few occurrences in *LAMMPS*. The observation that the signage and exponent bits are more vulnerable suggests that these bits should be protected by a stronger ECC scheme in a future design. However, one must note that depending on the endian and precision mode, the physical locations of the signage and exponent may change. The envisioned ECC code must also accommodate the different precision and endian modes, or even mixed, dynamically adjustable precisions and endian modes in the floating point register file as specified in the ISA.

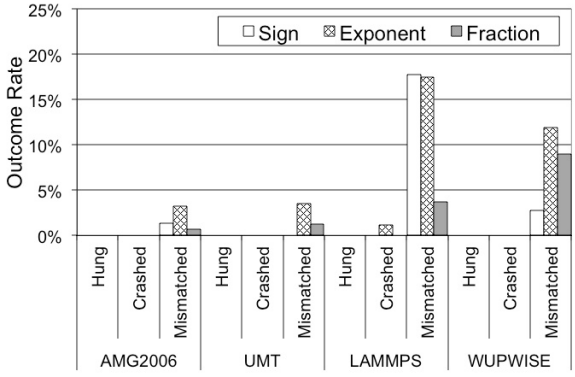

Figure 8: The failure (i.e., not masked) distribution depending on the bit position of injection.

E. Masking in the register allocation

When studying software-level vulnerability and optimization opportunities, register usage and allocation in the application are important because architectural registers are the most commonly used interface between hardware and software. While the *POWERPC* Instruction Set Architecture (ISA) allows any register to be used in the general-purpose sense, compiler specification often assigns specific role to each register for the sake of binary interfacing and standardization. Note that these register usage corresponds to logical names of the register files. In architectures that support register renaming, the most vulnerable registers are dependent on their usage and not its physical location. In this analysis, we quantify the failure rates of these general purpose registers (GPR) and relate the observations to their assigned roles by the compiler specification.

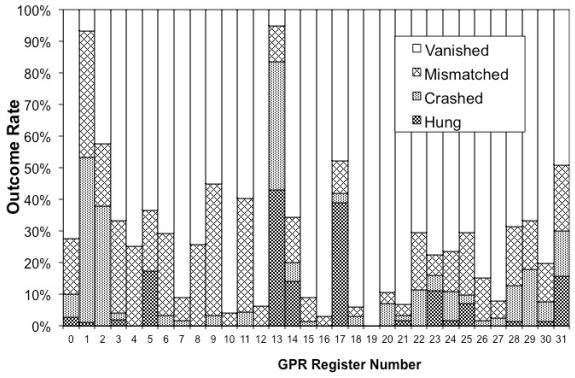

Figure 9: Failure distribution of AMG2006 with GPR injection based on register numbers/names. GPR1 and GPR13 can be seen to be highly vulnerable compared to the remaining GPRs,

Figure 9 shows the fault distribution across the 32 general purpose registers for *AMG2006*. Significant variation in the vulnerability of the registers can be observed, with GPR1, GPR2 and GPR13 having high vulnerability to the single-bit flips. GPR1 is used as a stack pointer and hence any corruption of the stack pointer would most likely lead to a crash or an error by moving the stack to some other point in computation. Such behavior was observed generally for GPR1 across all the benchmarks. Similarly, GPR2 is used as pointer for Table of Content (TOC) of the current running routine to locate its variables. Therefore a corrupted TOC will cause the execution to continuously load values from incorrect location of variables. GPR13 in *AMG2006* was also seen to be highly vulnerable with only 5% masking and this is mostly attributed to the nature of use GPR13. Because GPR13 is the preferred register in compiler stack-based register allocation algorithm among the global variable registers (GPR13-GPR31 are global registers), GPR13 is mostly used for reads and mostly into load instructions. A bit flip in the value of GPR13 will lead to corruption of the load address. The observation that some of these GPR registers are more vulnerable suggests that these registers should be protected by a stronger ECC scheme in a future design, assuming future compilers continue to adhere to the same register allocation schemes.

F. Algorithmic Masking in the Application Functions

Application-level fault injection helps to understand the resiliency of the application towards single or multiple-event upsets. The resiliency depends strongly on utilization and residency of architectural states; making some procedures or functions highly vulnerable compared to others. However, when predicting resiliency of a given function, the *algorithmic masking* in that function should also be taken into account. In this section, we provide a study of the application performance behavior coupled with the failure characteristics of the key functions for the given applications. A vulnerability analysis of each function can help application developers make the frequently called vulnerable functions more resilient by altering the usage patterns of the architected state and hence reduce the overall vulnerability of a given application.

Because GPR is more vulnerable than FPR, we focus our failure analysis of the applications based on the GPR AFI experiments. Table IV - Table VIII show the failure distribution when faults are injected into the GPR coupled with the time spent in the top 5 functions for each application. A breakdown of the fault distribution of each function highlights the vulnerability of any given function. In identifying the most vulnerable code segments in an application for optimization opportunity, both failure rate and execution time of the code segment relative to the application have to be examined together. For example; *hypre_SeqVectorInnerProd* was the most vulnerable function for *AMG2006* because it had the lowest 60.2% masked factor when considered in isolation without accounting for the time spent in the function. However, it accounts for only 5.8% of the runtime of the application; and hence reduced the overall vulnerability of the function to 7.8%. It is also not uncommon in the applications where some functions are way more vulnerable than their portion of execution time would suggest. For example, in Table VII which shows the failure rate breakdowns for *QCD*, the function *_vmx_dwf_dp_lab* was executed only 4.7% of overall execution time but was responsible for 21.3% of the failures in the GPR injection experiment.

Synchronization functionality such as idle loop and thread scheduler account for non-trivial portion of the overall failure percentage. For example, *_pthread_cond_wait* accounts for 10.2% in *AMG2006*, while *_sched_yiel*d and *.SpinWaitTaskSwitchBGQ* account for 3.7% and 2.6% in *LAMMPS*, respectively. This observation shows that there are software-level reliability optimization opportunities in the runtime stack, outside the application's and compiler's regimes.

Table III: Execution time and failure distributions of GPR AFI for AMG2006.

Function name	Timeout	Crash	Mismatch	Vanish	% Time	% failures
hypre_CSRMatrixMatvecT	5.8%	5.7%	16.6%	71.9%	27.0%	25.4%
hypre_BoomerAMGRelax	3.9%	10.1%	24.7%	61.3%	16.6%	21.5%
hypre_CSRMatrixMatvec	5.0%	6.2%	20.4%	68.4%	16.7%	17.7%
__pthread_cond_wait	8.1%	10.0%	5.8%	76.1%	12.8%	10.2%
hypre_SeqVectorInnerProd	3.4%	16.9%	19.5%	60.2%	5.8%	7.8%

Table IV: Execution time and failure distributions of GPR AFI for UMT.

Function name	Timeout	Crash	Mismatch	Vanish	% Time	% failures
Snswp3d	0.7%	22.7%	14.6%	62.1%	67.6%	72.9%
Snqq	0.8%	15.7%	17.4%	66.1%	6.1%	5.9%
.ThdCode	5.1%	14.6%	1.9%	78.3%	8.0%	4.9%
Snneed	2.0%	13.7%	29.4%	54.9%	2.6%	3.3%
Snflwxyz	0.0%	16.7%	30.6%	52.8%	1.8%	2.5%

Table V: Execution time and failure distributions of GPR AFI for LAMMPS.

Function name	Timeout	Crash	Mismatch	Vanish	% Time	% failures
PairEAM::compute	1.4%	20.8%	32.1%	45.7%	71.9%	84.2%
Neighbor::half_bin_newton	1.9%	17.7%	13.5%	67.0%	10.8%	7.7%
.__sched_yield	9.2%	9.8%	0.6%	80.5%	8.8%	3.7%
.SpinWaitTaskSwitchBGQ	8.3%	11.9%	1.8%	78.0%	5.5%	2.6%
FixNVE::initial_integrate	0.0%	16.7%	11.1%	72.2%	0.9%	0.5%

Table VI: Execution time and failure distributions of GPR AFI for WUPWISE.

Function name	Timeout	Crash	Mismatch	Vanish	% Time	% failures
Zgemm	1.1%	15.3%	31.7%	51.9%	33.3%	33.9%
Zaxpy	1.2%	8.0%	38.7%	52.1%	16.8%	17.0%
Memset	1.4%	16.2%	23.9%	58.6%	11.1%	9.7%
Lsame	0.8%	6.2%	27.8%	65.1%	12.1%	8.9%
Gammul	1.8%	13.5%	49.5%	35.1%	5.6%	7.6%

Table VII: Execution time and failure distributions of GPR AFI for QCD.

Function name	Timeout	Crash	Mismatch	Vanish	% Time	% failures
_vmx_dwf_dp_lab	0.0%	1.3%	67.9%	30.8%	4.7%	21.3%
britney::randFermion(Fermion_t)	0.0%	6.7%	0.0%	93.3%	42.7%	18.6%
_vmx_dwf_dp_dag_lab	1.9%	11.5%	53.8%	32.7%	3.2%	13.8%
drand48	0.4%	11.5%	0.0%	88.1%	15.8%	12.2%
vmx_cg	0.0%	0.0%	39.2%	60.8%	3.2%	8.0%

Table VIII: Execution time and failure distributions of GPR AFI for LINPACK.

Function name	Timeout	Crash	Mismatch	Vanish	% Time	% failures
.slow_stride	2.6%	0.0%	59.3%	38.0%	71.5%	80.6%
.simple_nn_dgemm	3.7%	0.0%	67.9%	28.4%	5.2%	6.7%
.HPL_dlaswp00N	2.4%	0.0%	81.2%	16.5%	4.0%	6.0%
.HPL_dgemv	0.0%	0.0%	65.7%	34.3%	1.9%	2.3%
.Copy_A_NonTranspose	0.0%	0.0%	38.7%	61.3%	1.8%	1.3%

IV. CONCLUSIONS

The principal goal of this work is to understand the interaction between BlueGene/Q hardware and high-performance applications when it comes to SER through chip irradiation experiment and application-level fault injections experiments.

From our irradiation experiments we derived that the mean time between correctable errors at sea level (New York City) of the SRAM-based register files and Level-1 caches for a system similar to the scale of Sequoia system with its roughly 1.6 million cores running HPC workloads to be approximately 1.5 days, an outstanding result for a system of this magnitude. In only the register files are considered, we found the mean time between correctable errors in the GPRs and FPRs of the similar system to be 16 days. The high rate of detected-and-corrected errors significantly validated the necessity to include autonomous hardware detection and correction at the cost of design effort, silicon area and power. In our AFI experiments, we characterized relative vulnerabilities among the applications in both general purpose (GPR) and floating-point (FPR) register files and the software attributes that lead to these vulnerabilities. In order to study software behavior under fault, we designed our software-based AFI methodology to intentionally bypass hardware detection in these register files. If the hardware detection had not been bypassed, a potential bit flip in the register files would have been detected and corrected by the aforementioned hardware before propagating into software-visible states. Thus, it should be put in perspective that the results of these AFI experiments do not directly extrapolate to silent data corruption (SDC) rate on any hardware.

In our results, applications are shown to be more resilient to FPR faults than GPR faults. In FPR fault injection experiments, *LINPACK* is the most vulnerable to SDC, roughly 50X more than *AMG2006* and *UMT* and 5X over other applications, although the built-in residual checking in *LINPACK* is also able to self-check for correctness of computed results. *AMG2006* and *UMT* are highly resilient to FPR faults possibly due to their algorithms. In GPR fault injection experiments, applications show similar resiliencies with masking factors ranging from 40% to 87%. We also examined the register usage in GPR and determined the GPR registers that are the most vulnerable. Going from single-bit upset (SBU) to multi-bit upset (MBU) fault injections, we do not observe significant changes in overall fault rates except MBU produces more detectable faults than SBU in GPR injection experiments. The increase in detectable fault rates is due to the fact that GPR are mainly used for address computations and changes in multiple bits are most likely to cause addresses to be out-of-bound and detected by hardware memory management. We also studied the most vulnerable code segments for each application, which would be useful for future tuning and improvement using software techniques such as resiliency register allocation or hardware transactional memory. Our study reveals that the most vulnerable codes are not often the most executed code segment, that profiling is needed to guide such improvements.

ACKNOWLEDGMENT

This work was supported in part by the Lawrence Livermore National Laboratory (LLNS) under the Blue Gene/Q Project [Subcontract #B554331]. The application-level fault injection (AFI) methodology development was sponsored in part by Defense Advanced Research Projects Agency, Microsystems Technology Office (MTO), under contract no. HR0011-13-C-0022. The views expressed are those of the authors and do not reflect the official policy or position of the Department of Defense or the U.S. Government. This document is: Approved for Public Release, Distribution Unlimited. The authors would also like to acknowledge Ethan Cascio and the Francis H. Burr Proton Therapy Center at Massachusetts General Hospital for their support and the use of their facilities.

REFERENCES

[1] ASC Sequoia Benchmark Codes https://asc.llnl.gov/sequoia/benchmarks/

[2] Kenneth P. Rodbell, David F. Heidel, Jonathan A. Pellish, Paul W. Marshall, Henry H.K. Tang, Conal E. Murray, Kenneth A. LaBel, Michael S. Gordon, Kevin G. Stawiasz, James R. Schwank, Melanie D. Berg, Hak S. Kim, Mark. R. Friendlich, Anthony M. Phan, and Christina M. Seidleck, "32 and 45 nm Radiation-Hardened-By-Design (RHBD) SOI Latches", accepted for December 2011 issue of the IEEE Transactions on Nuclear Science (TNS).

[3] Loveless, T.D., Jagannathan, S., Reece, T., Chetia, J., Bhuva, B.L., McCurdy, M.W., Massengill, L.W., Wen, S.-J., Wong, R., Rennie, D., , "Neutron- and Proton-Induced Single Event Upsets for D- and DICE-Flip/Flop Designs at a 40 nm Technology Node," IEEE Transactions on Nuclear Science, vol.58, no.3, pp.1008-1014, June 2011.

[4] Oldiges, P., Dennard, R., Heidel, D., Ning, T., Rodbell, K., Tang, H., Gordon, M., Wissel, L.,"Technologies to further reduce soft error susceptibility in SOI," Electron Devices Meeting (IEDM), 2009 IEEE International vol., no., pp.1-4, 7-9 Dec. 2009.

[5] B. Sinharoy, R. Kalla, W. J. Starke. H.Q. Le, R. Cargnoni, J. A., Van Norstrand, B. J. Ronchetti, J. Stuecheli, J. Leenstra, G.L. Guthrie, D.Q. Nguyen, B. Blaner, C.F. Marion, E. Retter and P. Williams, "IBM POWER7 multicore server processor", IBM Journal of Research and Development, vol. 55, no. 3, 2011.

[6] 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.

[7] 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.

[8] James Warnock, Leon Sigal, Dieter Wendel, K Paul Muller,Joshua Friedrich, Victor Zyuban, Ethan Cannon, A.J. KleinOsowski, "POWER7™ Local Clocking and Clocked Storage Elements," ISSCC 2010 Digest of Technical Papers, pp. 178-179.

[9] Michael Karl Gschwind and Robert Philhower, U.S. Patent 7512772 Soft error handling in microprocessors, Filing date: Jan 8, 2007, Issue date: Mar 31, 2009, Original Assignee: IBM Corp.

[10] C-Y. Cher, K. P. Muller, R. A. Haring, D. L. Satterfield, T. E. Musta, T. M. Gooding, K. D. Davis, M. B. Dombrowa, G. V. Kopcsay, R. M. Senger, Y. Sugawara, K. Sugavanam, "Soft Error Resiliency Characterization and Improvement on IBM BlueGene/Q Processor Using Accelerated Proton Irradiation",in Workshop on Silicon Effects of Logic - System Effects (SELSE), 2013.

[11] R.A. Haring, M. Ohmacht, T. W. Fox, M. K. Gschwind, D. L. Satterfield, K. Sugavanam, P. W. Coteus, P. Heidelberger, M.A. Blumrich, R.W. Wisniewski, A. Gara, G. L.-T. Chiu, P. A. Boyle, N. H. Christ and C. Kim, "The IBM Blue Gene/Q Compute Chip", IEEE Micro, vol. 32, no. 2, pp. 48-60, 2012.

[12] Michalak, S., DuBois, A., Storlie, C., Quinn, H., Rust, W., DuBois, D., Modl, D., Manuzzato, A. and S. Blanchard (2012) "Assessment of the Impact of Cosmic-Ray-Induced Neutrons on Hardware in the Roadrunner Supercomputer," IEEE Transactions on Device and Materials Reliability, vol.12, no.2, pp.445,454, June 2012.

[13] Carnes, B., Chan, B., Draeger, E. W., Fattebert, J.-L., Fried, L., Glosli, J., Krauss, W. D., Langer, S. H., McCallen, R., Mirin, A. A., Najjar, F., Nichols, A. L., Oppelstrup, T., Rathkopf, J. A., Richards, D., Streitz, F., Vranas, P. M., Rice, J. J., Gunnels, J. A., Gurev, V., Kim, C., Magerlein, J., Reumann, M., Wen, H.-F., "Science at LLNL with IBM Blue Gene/Q," IBM Journal of Research and Development , vol.57, no.1/2, pp.11:1,11:18, Jan.-March 2013.

[14] Müller, Matthias S. and Kalyanasundaram, Kumaran and Gaertner, Greg and Jones, Wesley and Eigenmann, Rudolf and Lieberman, Ron and Waveren, Matthijs and Whitney, Brian, "SPEC HPG Benchmarks for Large Systems", High Performance Computing, Lecture Notes in Computer Science vol. 2858, Springer 2003.

[15] US Lattice Quantum Chromodynamics http://www.usqcd.org/fnal

[16] The LINPACK Benchmark http://www.top500.org/project/linpack

[17] Shubhendu S. Mukherjee, Christopher Weaver, Joel Emer, Steven K. Reinhardt, and Todd Austin. "A Systematic Methodology to Compute the Architectural Vulnerability Factors for a High-Performance Microprocessor," In Proceedings of the 36th annual IEEE/ACM International Symposium on Microarchitecture (MICRO 36). IEEE Computer Society, Washington, DC, USA.

[18] Ethan H. Cannon, AJ Kleinosowski, K. Paul Muller, Tak H. Ning, Philip J. Oldiges, Leon J. Sigal, James D. Warnock, Dieter Wendel, U.S. Patent US 8354858 B2, Apparatus and method for hardening latches in SOI CMOS devices, Filing date: Jan 8, 2011, Issue date: Jan 15, 2013, Origional Assignee: IBM Corp.

[19] Ethan H. Cannon, David F. Heidel, K. Paul Muller, Alicia Wang, U.S. Patent US 20100301446 A1, In-line stacking of transistors for soft error rate hardening, Filing date: May 28, 2009, Issue date: Dec 2, 2010, Origional Assignee: IBM Corp.

[20] Hyungmin Cho, Mirkhani, S., Chen-Yong Cher, Abraham, J.A., Mitra, S., "Quantitative evaluation of soft error injection techniques for robust system design," Design Automation Conference (DAC), 2013 50th ACM / EDAC / IEEE, vol., no., pp.1,10, May 29 2013-June 7 2013.

[21] P. Montesinos, Wei Liu, J. Torrellas, "Using Register Lifetime Predictions to Protect Register Files against Soft Errors," 37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, 2007 (DSN'07), pp.286,296, 25-28 June 2007.

[22] Z. Kalbarczyk et al., "Hierarchical Simulation Approach to Accurate Fault Modeling for System Dependability Evaluation," IEEE Trans. Software Engineering, vol. 25, no. 5, pp. 619–632, Sept. –Oct. 1999.

[23] G. A. Kanawati, N. A. Kanawati, and J. A. Abraham, "EMAX: An Automatic Extractor of High-Level Error Models," Proc. AIAA Computing Aerospace Conf., pp. 1297–1306, 1993.

[24] M. Maniatakos, N. Karimi, C. Tirumurti, A. Jas, and Y. Makris, "Instruction-Level Impact Analysis of Low-Level Faults in a Modern Microprocessor Controller," Computers, IEEE Transactions on Computers, vol.60, no.9, pp.1260,1273, Sept. 2011.

[25] M. Rimen, J. Ohlsson,, J. Torin,, "On microprocessor error behavior modeling," Twenty-Fourth International Symposium on Fault-Tolerant Computing, 1994 (FTCS-24).

[26] Manu Shantharam, Sowmyalatha Srinivasmurthy, and Padma Raghavan, "Characterizing the impact of soft errors on iterative methods in scientific computing," in Proc. of the ACM international conference on Supercomputing (ICS'11).

[27] Teresa Davies and Zizhong Chen, "Correcting soft errors online in LU factorization," in Proc. of the 22nd ACM international symposium on High-performance parallel and distributed computing (HPDC'13).