Phaser: Phased methodology for modeling the system-level effects of soft errors

J. A. Rivers
P. Bose
P. Kudva
J.-D. Wellman
P. N. Sanda
E. H. Cannon
L. C. Alves

This paper presents an overview of Phaser, a toolset and methodology for modeling the effects of soft errors on the architectural and microarchitectural functionality of a system. The Phaser framework is used to understand the system-level effects of soft-error rates of a microprocessor chip as its design evolves through the phases of preconcept, concept, high-level design, and register-transfer-level design implementation. Phaser represents a strategic research vision that is being proposed as a next-generation toolset for predicting chip-level failure rates and studying reliability–performance tradeoffs during the phased design process. This paper primarily presents Phaser/M1, the early stage of the predictive modeling of behavior.

Introduction

As the trend toward smaller device and wire dimensions continues, we are entering an era of increased chip integration, reduced supply voltages, and higher frequencies. An inescapable consequence of this development is the fact that transient, or *soft*, errors will continue to be a serious threat to the general technology of robust computing. Soft errors may be caused by various events including neutrons from cosmic particle incidence, alpha-particles from trace radioactive content in packaging materials, and inductive noise effects ($L\mathrm{d}i/\mathrm{d}t$) on the chip supply voltage resulting from aggressive forms of dynamic power management.

As technology scales from 65 nm toward 45 nm and beyond, current soft-error rate (SER) projections for SRAM cells and latch and logic elements are noteworthy. As Borkar [1] has discussed, the SER per-bit rate for SRAM cells appears to be leveling off, but the bit count per chip is increasing exponentially in accordance with Moore's Law; latch SER is catching up with SRAM per-bit rates with a steeper slope of increase; and logic combinational SER is projected to increase at a much faster pace, although the absolute numbers are significantly smaller than SRAM or latch numbers at the present time. For silicon-on-insulator technology, going forward from 65 nm to 45 nm, the latch SER per-bit rate is predicted to increase from two to five times [2], and the number of latches per chip is expected to increase with integration density. Again, storage-cell SER will still dominate, and latch errors will also be of increasing relevance at 45-nm technologies and beyond.

Chip design must begin with a consideration of system-level mean-time-to-failure (MTTF) targets, and the design methodology must be able to estimate or set bounds for chip-level failure rates with reasonable accuracy in order to avoid in-field system quality problems. A balanced combination of circuit- and logic-level innovations and architecture- and software-level solutions is necessary to achieve the required resiliency to single-event upsets (SEUs). In particular, there is the need for a comprehensive understanding of the vulnerabilities associated with various units on the chip with regard to workload behavior. When such information is available, appropriate approaches—such as selective duplication, SER-tolerant latch design adoption, and error-correcting code (ECC) and parity protection of SER hotspots[1]—may be used for efficient error resiliency.

Our motivation for this presilicon modeling work, therefore, hinges on the premise that an accurate methodology driven by target workloads would enable

[1]*SER hotspot* refers to a region of the chip that is deemed to be highly vulnerable to bit flips in an element (latch, combinational logic, or SRAM). An upset in this region is much more likely to cause a program failure than other adjoining areas. The SER hotspot profile across the chip floorplan may vary with the executing workload; however, it is quite likely that some of the SER hotspots may be largely invariant with regard to the workload. For example, portions of the instruction decode unit (through which all program instructions must pass) may well turn out to be SER hotspots across a wide range of input workloads.

0018-8646/08/$5.00 © 2008 IBM

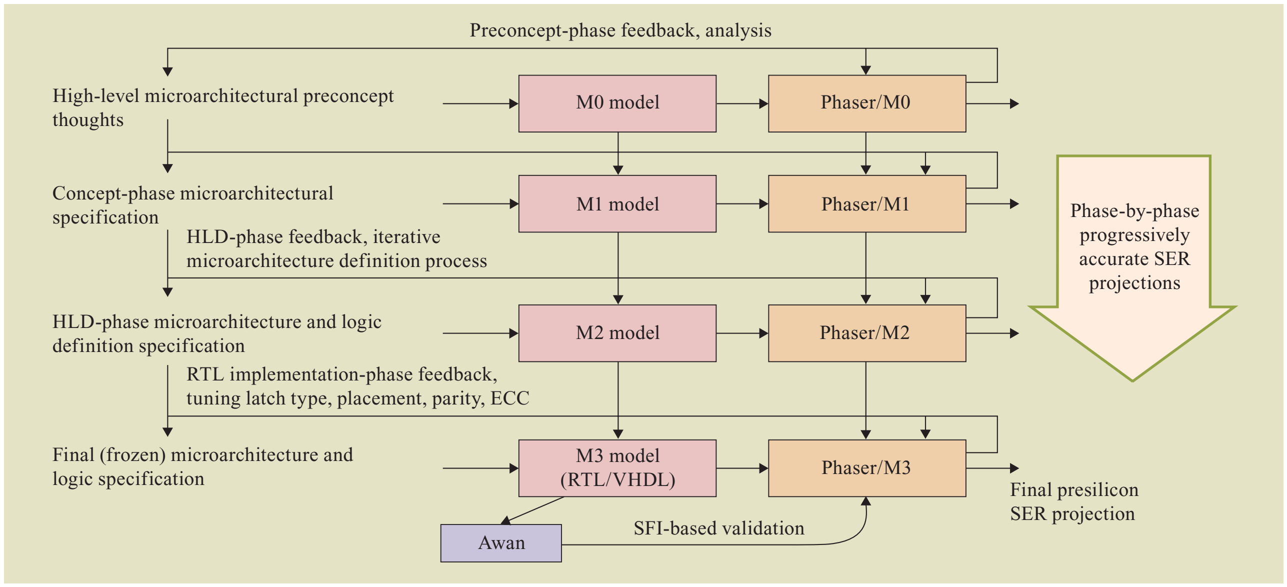
Preconcept-phase feedback, analysis
High-level microarchitectural preconcept thoughts
M0 model
Phaser/M0
Concept-phase microarchitectural specification
M1 model
Phaser/M1
HLD-phase feedback, iterative microarchitecture definition process
Phase-by-phase progressively accurate SER projections
HLD-phase microarchitecture and logic definition specification
M2 model
Phaser/M2
RTL implementation-phase feedback, tuning latch type, placement, parity, ECC
Final (frozen) microarchitecture and logic specification
M3 model (RTL/VHDL)
Phaser/M3
Final presilicon SER projection
Awan
SFI-based validation

Figure 1

High-level view of Phaser.

cost-effective SER solutions during the design process. In this paper, we present the Phaser modeling approach to enable early estimation of SER. This toolset and methodology is helpful for deriving the following:

- SER vulnerability maps of the chip that can be optionally visualized as color-coded chip floorplans that indicate the SER hotspot regions of the chip.
- Iteratively improved designs as the chip goes through design phases, successively enhancing the protection level of the most vulnerable units.
- Design guidance to enable investment in error detection and recovery hardware for optimal error resiliency.

During the concept phase, the integration of device-level and component-level SER analysis with microarchitecture-level performance analysis tools would enable us to study key tradeoffs between performance, power, and reliability. In particular, besides projecting derating[2] factors and overall SER sensitivity values for the chip and its various components, this framework allows us to undertake what-if evaluations and comparisons of the soft-error sensitivities of various latches and cells.

It also enables the architecture definition team to decide on the exact style and level of microarchitectural redundancy that may be needed to achieve the per-chip SER targets. In later stages, as the register-transfer-level (RTL) model becomes available, the chip SER profile is refined as more accurate information about the unit-wise latch distributions, latch types, and SER vulnerabilities of logic and latch elements becomes available. In these later design phases, major microarchitecture paradigm changes are generally not possible, but the Phaser-derived analysis can help adjust the relative protection level and latch types across highlighted units.

Figure 1 is a high-level view of the phased SER evaluation methodology pursued by the Phaser approach.

The particular models around which the modeling is targeted at various design phases are the M0 model at the preconcept phase; the M1 model at the concept phase; the M2 model during the high-level design (HLD) phase; and the M3 model during the RTL implementation phase. Typically, the M0 model is an analytical (spreadsheet) performance model or a very early cycle-approximate simulator that is adapted from an earlier-generation M1 cycle-accurate performance model. As the definition progresses to the concept phase, the architecture team arrives at a more definite view of the core and chip-level microarchitecture; at this time, Phaser makes use of the M1 cycle-accurate performance model of the core to build the SER analysis tool. Later, during the HLD phase, the M1 performance model is replaced by a latch-accurate M2 model in which the inter-unit interfaces are accurately

[2]As used here, *derating* refers to the portion of time a microprocessor unit or structure (whether logic or storage) is not in use or operation in a manner that affects program correctness; therefore, it can be said that it is not potentially susceptible to errors. For example, a unit with a derating factor of 75% over a program run implies that such a unit is susceptible to errors 25% of the total execution time of the program run.

modeled in terms of the exact latch counts. The intra-unit execution semantics are still written in a behavioral format using C/C++ language types, as in the M1 model. During this stage of the design, Phaser/M2 is able to model the inter-unit error propagation effects more accurately because the interface latches and their switching activities are directly observable during the simulation of specific workloads. During the RTL implementation phase, Phaser SER analysis moves over to link up with the RTL M3 model that contains detailed logic, latch, and timing information for the full processor. In addition to VHDL (Very high-speed integrated circuit Hardware Description Language) cycle-accurate software simulation (which is rather slow), we have the facility of using significantly accelerated Awan [3, 4] hardware simulation of the RTL, which enables us to run full benchmarks if necessary at RTL detail. At this stage of the design, as the RTL approaches full functionality, the Phaser methodology makes use of validation and calibration support from statistical fault injection (SFI) [5] approaches.

Background

Li et al. [6] present a microarchitecture-level technique, SoftArch, for modeling and estimating architectural masking, which leads to derating, in a microprocessor. In that work, the authors adopt a methodology that tracked and assigned a probabilistic failure rate attribute to every useful variable as the variable traveled through and resided in the various units and structures in the processor pipeline. The accumulated probabilistic failure rate of the given variable as it tracked through the microprocessor pipeline and datapaths to a committed execution stage, including program outputs and permanent memory state, was used to project and calculate the microprocessor error sensitivity values (along with the corresponding unit-wise derating factors). In later work [7], it was shown that for practical ranges of the native per-bit SERs observable at sea level and for modeled systems with tens of components (units), a simpler, post-processing approach can yield sufficiently accurate per-unit and total system SER. Such an approach is based on two steps: first estimating a per-unit average architectural vulnerability factor (AVF) [8] and multiplying that by the unit maximum (unmasked[3]) SER to project the real (derated) SER of the unit as actually manifested in program behavior; second, adding unit-level error rates to derive the chip-level SER value, which is referred to generally as the *sum of failure rates* (SOFRs) [7, 8]. However, it is important to stress that the accuracy

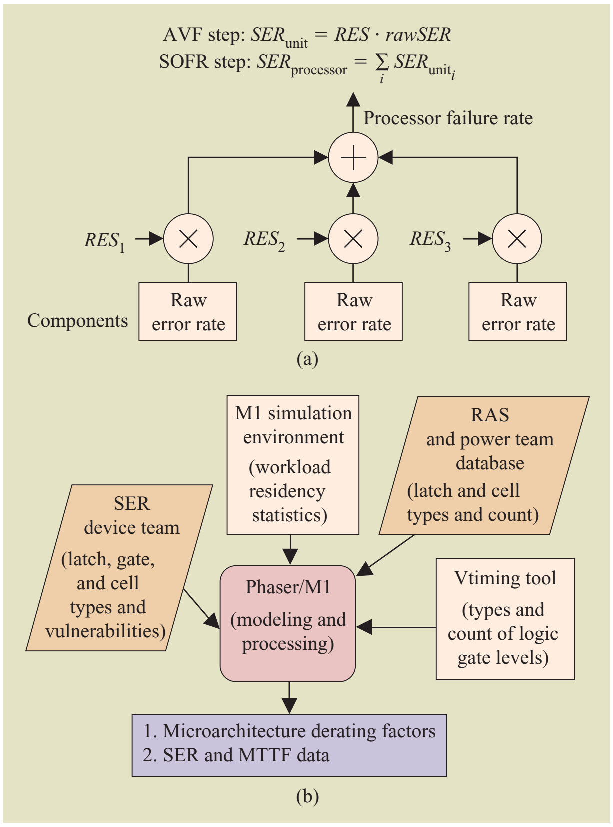
AVF step: $SER_{\text{unit}} = RES \cdot rawSER$
SOFR step: $SER_{\text{processor}} = \sum_i SER_{\text{unit}_i}$
Processor failure rate
RES_1
RES_2
RES_3
Components
Raw error rate
Raw error rate
Raw error rate
(a)
M1 simulation environment (workload residency statistics)
RAS and power team database (latch and cell types and count)
SER device team (latch, gate, and cell types and vulnerabilities)
Phaser/M1 (modeling and processing)
Vtiming tool (types and count of logic gate levels)
1. Microarchitecture derating factors
2. SER and MTTF data
(b)

Figure 2

(a) AVF/SOFR approach for estimating SERs. (b) Phaser/M1 methodology overview. (AVF: architectural vulnerability factor; SOFR: sum of failure rate; RES: residency; RAS: reliability, availability, and serviceability.)

of the unit-wise and total manifested failure rates depends on how the AVFs are collected.

In Phaser, we use such a post-processing approach wherein a simulator software code is modified with inserted monitoring instructions to yield all required average residency statistics at the end of a workload run; those statistics are then combined with the detailed information of per-unit latch distributions of specific types and protection levels, along with native technological data related to raw per-bit SER values. The AVF/SOFR approach estimates the SER of a processor or system in two steps, as shown in **Figure 2(a)**. The first step, the AVF step, estimates the SER of the individual components under the basic assumption that the probability of failure is uniform across a program execution. Hence, the SER of a given component in a processor is simply the fraction of time it holds useful work multiplied by the raw SER of the component. We refer to that fraction of time as the component value of data residency. The second step, the SOFR step, estimates

[3]We use the terms *unmasked SER*, *native SER*, and *raw SER* interchangeably throughout this paper. Any of these terms can be used to mean the maximum SER that would manifest in the absence of any derating at all.

the SER of the entire processor or system by adding together the individual SER values of the constituent components under the general assumption that the inter-arrival time for failures is exponentially distributed. Depending on the particular design phase, the methodology derives the RES factors from the corresponding simulation model (M0, M1, M2, or M3; see Figure 1).

Phaser [**Figure 2(b)**] uses methods to calculate the architectural derating factors that are different from the prior published methods [8, 9] and can be implemented more practically while preserving accuracy. For example, a systematic method to monitor only useful register file residencies (i.e., those that contribute to actual instruction completion and modification of the architected register and memory state) is implemented. (*Architected state* refers to the state components of the machine that are accessible by software, including memory, register files, and special-purpose registers.) The measured residency data is combined with the various latch, logic, and cell raw SERs of the targeted chip in a systematic manner to project the derating factors as well as the overall SER. We use the term *residency* (RES) in this paper where prior work [8] has used the term *AVF*, but for all practical purposes, the intent in estimating this entity is the same: namely, to arrive at a derated SER value for a given modeled component or unit in the system. However, the conceptual differences in how the estimate is derived led to our decision to use *RES* in preference to *AVF*.

As Figures 2(a) and 2(b) show, a considerable amount of data and a number of factors from various sources must be factored together to achieve a realistic SER and derating prediction model. Generally, our framework revolves around two major approaches: estimating the raw SER of the targeted chip or system, and deriving the average residency of the typical workload executing on the same system.

Methodology

As shown in Figure 1, Phaser is a multistage evaluation and estimation framework of soft-error vulnerability through the various stages or phases of system design. The main methodology behind each of the phases in the framework is similar except that as we move from the preconcept phase to the later phases, the information available and its accuracy improve, making it possible for the overall modeling accuracy to also improve. Since the various design phases are not fundamentally different and only the modeling detail is refined over time, we illustrate the methodology of the framework by discussing a single phase in depth. We focus our discussion on Phaser/M1 in which it is assumed that the design has an M1-level cycle-accurate microarchitecture performance simulator and possibly some preliminary design VHDL code available. It is also assumed that we have clear knowledge and choices of the various circuit elements: latches, combinational logic, and memory cells along with the technology parameters that govern their behavior.

As already indicated, we define the *raw SER* of the microarchitecture or chip as the expected total SER assuming that the microarchitecture or chip is busy 100% of the time and that every bit upset that occurs during its operation leads to a manifested error. Accurate raw SER modeling of a chip or its components requires an in-depth knowledge of the constituent latches, array cells, and combinational logic with respect to counts and types as well as their associated vulnerabilities to soft errors. We obtain latch count and type data from the design database, estimate combinational logic-level gate counts from available design RTL with the Vtiming tool [10], and obtain associated SER vulnerabilities of technology elements, such as latches and cells, from circuit-level SPICE simulations [2] as shown in Figure 2(b).

Although we model the raw SER of a system assuming that all bit flips could be of consequence, we know that for the typical microarchitecture, the residency of useful data is well less than 100% across all modeled units within the system. Hence, to better estimate the derating factor of a chip structure, we need to gather statistics on the residency of relevant live data values within the structure. This is where it becomes necessary to collect such residency values as accurately as possible on the platform under study through its available simulator as it executes a typical representative workload.

In this paper, the Phaser/M1 methodology is described with a single focus only on silent data corruption (SDC)-related SER manifested at the program output. We are interested here in predicting an early stage (necessarily conservative) bound on the SDC-specific SER or equivalently the machine derating [5] factor applicable to SDC-specific error rate estimations.

Raw SER modeling

To model the raw SER of a microprocessor, a component structure, or the whole chip, we begin with the complete profile and counts of the different latch, combinational logic gate, and SRAM register cell composition of the corresponding entity. In particular, we must have the types and counts of the latches, the logic gate levels[4], and the SRAM register cells in the structure under study. As depicted in Figure 2(b), the latch and cell types as well as the count estimates of the various chip units come from

[4]For a combinational logic network, the number of logic levels is defined as the maximum number of logic gates in the path from any primary input to any primary output. The logic level of G, a particular gate, can then be defined as the minimum number of gates between the output (receiving) latch and G, as a path from the output is traced back to the primary input (source pipeline latch) through G. The level of the gates that write to the receiving pipeline latch is thus 0, a gate that has one level-0 gate separating it from the receiving latch is at level 1, and so on.

1	2	3	4	5	6	7	8	9
Fnct Unit	LatType	LatCnt	LgcLvl	LgcLvlCnt	RegCnt	Raw SER (logic)	Raw SER (register)	Raw SER (logic and register)
IFU								
	aa	5,100	Lvl 0	22,100	6,100	3.33E+00	8.85E−01	4.22E+00
	bb	2,000	Lvl 1	29,300				
	cc	160	Lvl 2	30,500				
	a	10	Lvl 3	53,500				
	b	5,300	Lvls 0–3	135,400				
	c	1,300	All Lvls	683,500				
	d	700						
	e	260						
	f	10						
	g	10						
	Total latch	14,850						

1	2	3	4	5
IFU Type	RES	SER (logic/latch)	SER (reg)	Net SER
Logic	0.729	2.43E+00		2.60E+00
Storage	0.194		1.71E−01	

Figure 3

Deriving the real (i.e., effective or derated) SER value from the raw SER for a hypothetical IFU running the SPECint** average workload. The circled data highlights the effect of residency and how the raw SER number is derated to the final SER number. (Fnct Unit: functional unit; LatTyp: latch type; LatCnt: latch count; LgcLvl: logic level; LgcLvlCnt: logic-level gate count; RegCnt: register bits count.)

the design database of the specific chip development project. Such data is updated as more accurate design data becomes available with the maturity of the VHDL logic design. As for the levels and counts of the combinational logic gates, we collect such data by applying the Vtiming tool [10] on the evolving RTL model of the chip or unit. Vtiming allows us to scan the VHDL description and gather statistics about the number of logic gates within each level of a given combinational logic chain. Our interest focuses primarily on the gate counts within level 0 through level 3 (counting levels backward from the receiving pipeline latch bank). This is in light of empirically established knowledge that due to logic-level masking effects, SEU events on gates in levels 4 and beyond generally do not contribute to manifested errors in the receiving latch bank in real microprocessor pipelined logic paths.

The upper table of **Figure 3** shows a simple example of a spreadsheet output for a raw SER model of a hypothetical instruction fetch unit (IFU) in a microprocessor. For this illustration, we focus the modeling on only the latches, the relevant combinational logic gates, and the register cells in the unit. Columns 2 and 3 list the different types of latches and their corresponding counts in our IFU example. Columns 4 and 5 give the count of the logic gate levels and corresponding gates. Column 6 lists the count for the register cells in this hypothetical IFU. Column 7 shows the computed raw SER of the unit (reported for illustration, in arbitrary units), considering only the latches and the relevant combinational logic gates. Column 8 shows the computed raw SER of the unit with respect to the register cells. Column 9 is a sum of the values from columns 7 and 8 to give the total raw SER for the unit. The raw SER derived in columns 7 and 8 is basically the summation of the histograms of the latch, cell, and combinational logic type and their corresponding count frequency multiplied by the raw SER vulnerability of the latch, cell, and logic type. We emphasize that the SER vulnerability data we use for the examples in this paper is for illustrative purposes only, reported in arbitrary units. They do not represent real values. In the IFU example, we estimate the raw SER value to be 4.22 (column 9). To find the actual manifested SER (depicted in the lower table in Figure 3), we must ascertain how much derating can be credited to this unit, for a given workload execution. Phaser/M1 attempts to capture most of the derating effect that can be ascribed to the microarchitecture. This is explained in detail below.

Workload residency modeling

Microprocessor chip structures can be broadly classified into two main groups: logic and storage. We define logic structures to be the various data and control processing units on the chip that are made up of combinational logic gates and latches. Typical examples of on-chip logic structures, using our definition, include the fixed-point unit (FXU) pipelined logic datapath (with associated control logic) and the instruction decode unit (IDU) logic. We define storage to be the various structures that hold data values, such as the queues, register files, and other SRAM macros. Of course, latches may also serve as staging and data-hold resources, especially during stalls in a pipeline flow. In this case, depending on how such stalls are implemented in relation to the clock-gating functionality within the pipeline, certain latch banks may also be categorized within the storage class. However, as explained later, the residency modeling for such pipeline latches is simpler than register files and arrays and is better treated under the logic category.

Workload residency modeling attempts to measure the opportune proportion of cycles during a workload execution for which bit-flip events could alter program correctness. Hence, to accurately capture such residency data, the focus must include only the true path of program execution. For example, dataflow-centric SERs on a misspeculated path during program execution cannot alter program output. Similarly, rejected or flushed executions, dead instructions, NOPs (no operations), and performance-enhancing instructions (e.g., those related to data prefetch) do not contribute to SER-induced data corruption. In effect, for logic structures, the question we must ask is, During what fraction of the cycles is there an operation that uses a particular logic structure in such a manner as to lead to actual completed instructions? For storage structures, the question is, During what fraction of cycles is the storage resource holding a value that will subsequently be used in the true execution path?

Logic structures: Basic usage residency

For pipelined logic structures, instructions, data values, and control bits stream through the structure in the absence of stalls. Thus, a typical live data value item spends only a single cycle per structural pipeline stage. However, depending on the design of the structure, instruction stall periods may occur, during which resident data may be further exposed to SERs. In such cases, latch banks (registers) that hold the stalled instruction or data components must be analyzed to compute the added SER vulnerability. Microarchitectural M1-class simulators report utilizations for each pipeline stage and these capture stall-related utilizations of that resource, in addition to stall-free single-cycle usage. Residency in pipelined logic structures is, therefore, formally defined as the percentage of cycles during which the corresponding logic structure is busy computing or holding stalled data. In a microarchitecture simulator, we measure the number of cycles that this structure is busy. At the end of the simulation run, the usage residency of the structure may be calculated as the following fraction:

$$RESusage = \frac{(Number\ of\ busy\ cycles)}{(Number\ of\ total\ program\ cycles)}.$$

This basic usage residency calculation does not reflect corrections that must be applied to factor out those cycles that do not have an impact on final program output.

Storage structures: Basic live data residency

For storage structures, a value is written into or read out of the component cells or arrays. Our interest is to be able to compute the average live data residency durations of the component cells of the structure. The *live time* for given data is the period between its write and the last useful read. When data is written into a location and there is no subsequent read before it is overwritten or the program completes, the data is dead and its live time is zero.

Naturally, if a soft error hits a location with a live data value, it will corrupt the data. However, if the error hits a dead data item, it will not have an effect on the program output, and the error will be automatically masked. Hence, for a storage cell, the derating factor depends on the percentage of time that it is holding live data. In a simulator, we measure the percentage of the live data residence time. For a storage structure, we derive the average residency across all of its component cells.

For example, suppose we have an SRAM queue with N entries. To measure the average residencies across all the component entries, we monitor and sum all the live data residence times (cycles) per each queue entry. At the end of the program run, the live residency per each entry is given by the following fraction:

$$live\_res(i) = \frac{(Number\ of\ live\ residency\ cycles\ for\ i\text{th}\ entry)}{(Number\ of\ total\ program\ cycles)}.$$

The average live residency of the structure, therefore, is the accumulation of all the $live\_res(i)$ values, divided by N.

As before, this base residency calculation for storage structures does not take into account adjustments that are required to account for apparent live residency periods that do not actually affect program output.

Correcting factor

The above two effects, the logic and storage residence times, provide the base residencies of the structures. However, using this alone is very conservative as the derating factor. There are many speculative and performance-enhancing activities that occur in a processor that do not contribute to the functional correctness of the program. If a soft error were to hit any of such instructions or their corresponding values, it would not lead to processor failure; the effect would at most be only a degradation of program performance. To obtain more accurate derating factors, statistics are required to augment the calculation. Although we discuss these correcting effects separately, note that such statistics do not have to be collected through a second pass of the simulation run. In fact, for the residency numbers we use for this study, the logic and storage-correcting effect is taken into account while collecting the residency statistics from the simulator in a single simulation run. In effect, all statistics collected from the simulator are those along the true program execution path and of instructions and data values that actually affect the final program outcome.

Branch misprediction rate

Wrong-path speculative instructions are squashed and thus would not contribute to the SER of the microprocessor, even if they have errors. The branch misprediction rate M is defined to be the number of mispredicts per completed instruction. On the basis of the misprediction rate and the branch misprediction penalty, one may formulate a quick estimate of the percentage of squashed instructions as follows:

Let the processor average branch misprediction penalty be P cycles, which means that on average, when a misprediction is detected, there is a pipeline stall of P cycles. Let the number of completed instructions be N and the number of busy cycles of the targeted unit be T_{busy}. The total execution time for completing the N instructions is T. Then the cycles-per-instruction adder resulting from branch mispredicts is $M \cdot P$. Therefore, the total number of cycles lost due to stalls from mispredicts is $T_{\mathrm{waste}} = M \cdot P \cdot N$. The original (conservative) residency value for a structure would be given by $RES_{\mathrm{cons}} = T_{\mathrm{busy}}/T$. If the subject unit always suffers the full stall penalty on each branch mispredict, then the corrected residency of the unit would be

$$\begin{aligned} RES_{\mathrm{real}} &= (T_{\mathrm{busy}} - T_{\mathrm{waste}})/T \\ &= RES_{\mathrm{cons}} \cdot (1 - T_{\mathrm{waste}}/T_{\mathrm{busy}}). \end{aligned}$$

Dead instruction and performance-enhancing instruction percentage

Just as with wrong-path instructions, dead instructions and performance-enhancing instructions will not cause processor failure, even if they are hit by soft errors. It is difficult to measure these two in an early-stage performance simulator precisely without detailed instrumentation of the simulator code. Fortunately, a good estimate of the correction factor due to the above two sources can be derived by simple analysis of the workload trace.

Suppose the fraction of dead and performance-enhancing instructions is determined to be P_dead, then the RES value may be further refined to

$$RES = RES \cdot (1 - P\_dead).$$

Computing the residency factors accurately: A generalized view

When it comes to SER modeling, there are often attempts to use microarchitecture utilization as a proxy for actual residency. However, a close examination of these two metrics in a complex microprocessor pipeline shows a potentially significant difference between the two. Some examples of where and how such corrective factors may be applied via use of average stall event or dead instruction statistics were discussed above. However, in general, there are many more sources of derating imposed by the microarchitecture–workload pair. The effective correction factors to the computed residency (due to all sources) would be awkward and error-prone to derive individually via average statistical behavior alone. In this subsection, we formulate the general problem of extracting true residency statistics directly through more careful, adjusted usage monitoring within the M1 simulator.

The diagrams in **Figure 4** illustrate an important difference between the measurement of residency and utilization during the simulation of a workload. Each of these diagrams represents a single-cycle snapshot of the state of a simplified IBM POWER* microprocessor pipeline model (e.g., loosely based on [11]). In these diagrams, each box represents a location in the processor that could contain the information for an executing instruction, for example, pipeline stage latches or issue queue entries. The general flow of instructions through the pipeline is from left to right, where instructions pass through the fetch logic into the instruction buffer (IBuffer) and then through the decode stages and mapper into the issue queue (IQueue). From the IQueue, instructions are then each issued into one of the different execution units: one of the fixed-point units (FX0, FX1), the load-store units (LS0, LS1), or the branch resolution unit (BRU). This simplified view effectively shows single-

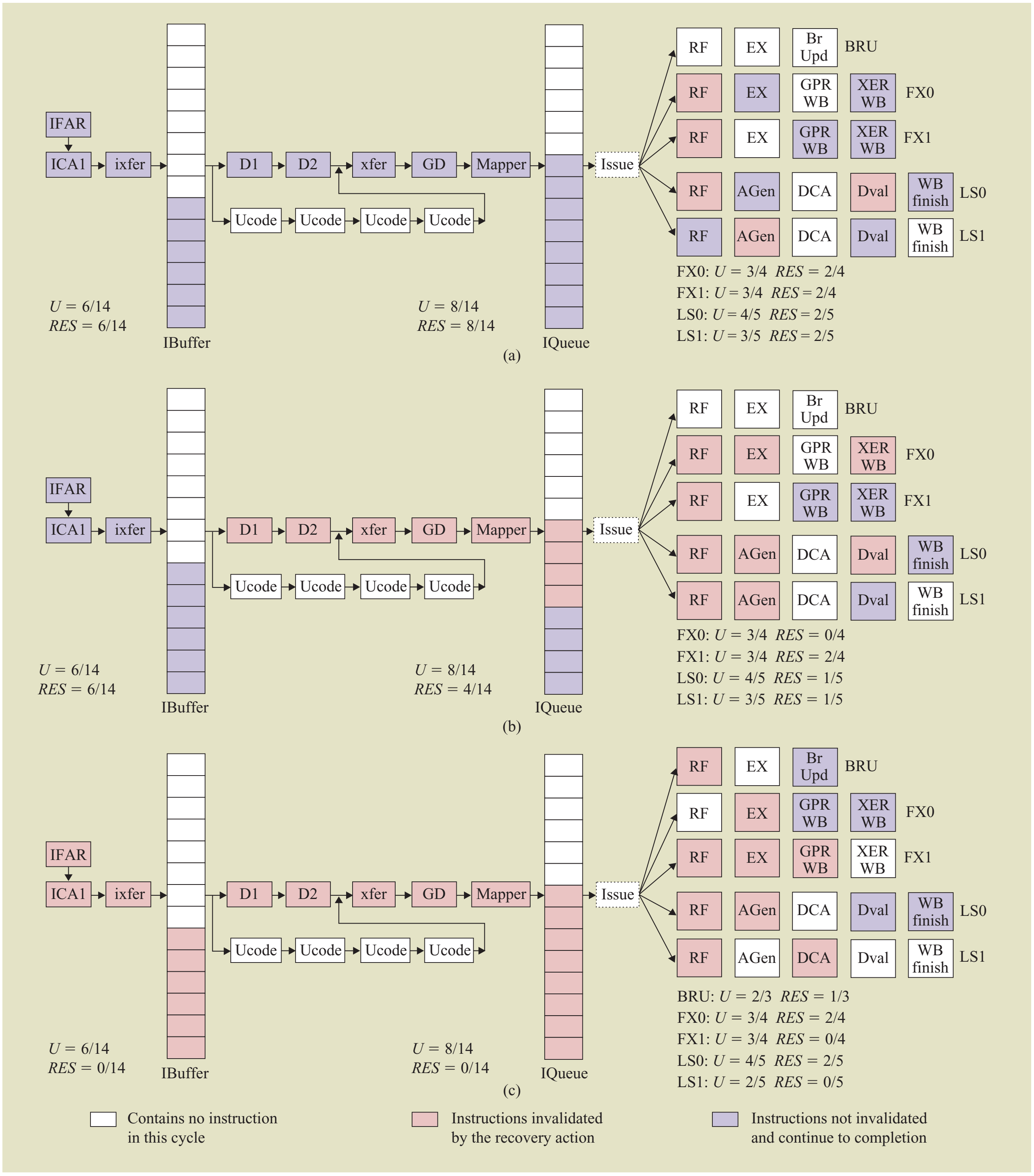

Figure 4

Microprocessor pipeline snapshot state scenarios: (a) state reflecting a reject recovery; (b) state reflecting a flush recovery; (c) state reflecting a mispredict recovery. (AGen: address generation stage; Br Upd: branch update stage; BRU: branch resolution unit; D1, D2: decode stage 1, decode stage 2; DCA: data cache access stage; EX: execution stage; FXU: fixed point unit; GD: group dispatch stage; GPR WB: general-purpose register writeback stage; ICA1: I-cache access stage 1; IFAR: instruction fetch address register access; ixfer: instruction fetch transfer cycle stage; LSU: load/store unit; RF: register file access stage; Ucode: microcode access stage; XER WB: exception register writeback stage.)

instruction decode, and ignores many aspects of a real microprocessor (model) for purposes of clearer explanation of the process. The diagrams of Figure 4 are intended to illustrate the difference between utilization and residency, which is particularly clear when viewed instantaneously (i.e., in a snapshot of the pipeline contents on a given cycle). In these diagrams, each box illustrates a potential location for executing instruction information. The colored boxes indicate locations that are currently holding information related to an executing instruction; the blue boxes indicate instructions that continue through normal execution and completion and, therefore, can affect the final output of the workload, while the red boxes indicate instructions that will not continue through to completion in this execution but will be invalidated (and possibly restarted later).

These three diagrams indicate three different exceptional execution conditions that require some internal adjustment (i.e., some pipeline recovery action) by the microprocessor. Figure 4(a) represents the situation in which an instruction is determined to be one that needs to be rejected for some reason, such as a cache miss. A reject action requires that the rejected instruction and any others that are dependent on it be invalidated from their current locations in the pipeline and reissued from the instruction queue (IQueue). In Figure 4(a) the instructions that are going to be rejected and reissued are colored red while those unaffected are blue. Figure 4(b) represents the situation in which an instruction is being discovered to require a flush recovery action, that is, the instruction and all younger (more recently fetched) instructions must be invalidated in the core pipeline and restarted from the instruction buffer (IBuffer).

Finally, Figure 4(c) represents the situation in which a branch misprediction is identified (the blue box instruction in the BRU), which requires that all instructions younger than the mispredicted branch be invalidated in the pipeline, and execution resume after an instruction fetch from the corrected branch target address. In the diagrams of Figure 4, the computation of the cycles contribution to utilization (U) as against residency (RES) is shown for most of the units by a fraction formula. The utilization is simply the number of colored (occupied) boxes, whether red or blue, divided by the total number of boxes in a given unit; the residency is the number of blue boxes divided by the total number of boxes in the unit.

These formulas characterize what would be used if the utilization or residency calculations were done during each cycle of simulation and well illustrate the difference in these metrics. In practice, residency is not computed on a cycle-by-cycle basis in this manner. Comparing the utilization of the FX0 in Figure 4(b), one finds the utilization to be three colored boxes out of four total and the residency is zero out of four boxes because the three instructions currently utilizing the FX0 pipeline will all be flushed and reexecuted later. Because the current execution of those instructions will not complete, these executions cannot contribute to the final workload output, and thus cannot contribute to errors in the workload output. In practice, the simulator does not try to determine residency on a cycle-by-cycle basis but rather records with each instruction its potential contribution to the unit residencies and adds these contributions only at the time the instruction reaches simulated completion. In this way, the simulator can easily guarantee that residency contributions are taken only for instructions that execute fully to completion along the correct execution path.

Going from raw SER to actual SDC-related SER: An example

Referring to Figure 3, an example can be seen of the basic steps involved in Phaser modeling from raw to real SER for a hypothetical IFU of a microprocessor executing the average Standard Performance Evaluation Corporation SPECint (SPEC integer) workload. Again, we limit our attention here to only SDC-related SER at the program output.

As discussed earlier, we estimate the raw SER due to latches and combinational logic in the IFU to be 3.33 and the raw SER due to the register cells as 0.88, giving a combined total raw SER of 4.22 for the unit. The SER measures are in arbitrary units because the only goal is to quantify the relative reduction of SER due to derating, as explained below.

To obtain the actual (manifested) SER when the component is executing the SPECint suite average, we derate the raw SER values by the actual RES of the component. In our simulations, we derive two RES numbers for the IFU; for the logic structure part, we estimate an average RES of 0.729, while for the storage register structure we derive 0.194. These values are generated with the correction effect discussed earlier.

The logic structure RES of 0.729 derates the IFU logic component of SER from 3.33 to 2.43. Likewise, the register storage structure RES of 0.194 derates the IFU storage register component SER from 0.885 to 0.171. Putting the two segments together, we end up with a total derated SER estimate of 2.60. In terms of derating factor, this means that at least 39% of the raw SEUs in the IFU region (caused by soft errors) are not expected to have an effect in the final program output as far as data integrity is concerned. The reason we say “at least” is that what we have captured so far in the Phaser/M1 predictive modeling is a subset of the full range of sources of derating in SDC analysis for a given workload.

SER projections, analysis, and discussion

In this section we describe the various classifications of soft errors from the perspective of the microarchitecture and discuss how the Phaser framework identifies and projects the amount of derating obtainable at the microarchitectural level. We also present a summary view of the Phaser-based SER projection and derating analysis for a full-function microprocessor.

Error classification and microarchitectural derating

The error classification and derating taxonomy adopted in the Phaser framework follow the same lines as those used in Sanda et al. [5]. As a brief review of that taxonomy, any bit flip occurring in a modern microprocessor chip (with architected detection and recovery support for soft-error tolerance) can be placed in one of four classes. (Note again that we limit our attention in this particular paper to SDC-related final program output errors only.)

a. *Vanished error class*—errors that occur as a result of bit flips but have no architectural consequence. For example, if a bit flip occurs in a unit that is not active or is not holding useful data at the time of the bit-flip occurrence, and if that SEU does not trigger any error detector, then that bit flip will be of no consequence.
b. *Corrected class*—bit flips that generate an error condition, but the system is able to detect and correct the error or can recover through re-execution from a prior-saved golden checkpoint. A typical example may be an ECC-protected array or a parity-protected latch that can trigger a hardware recovery action.
c. *Checkstop class*—bit flips that result in errors that are detected in hardware but cannot be corrected or recovered via hardware means. In this case, the detected error is trapped by software and usually results in a machine check. While a checkstop can terminate a job on a system, it does not result in an SDC event.
d. *Incorrect architected state class*—bit flips that manifest as a deviation in the architected state from what would be expected in the absence of an SEU. (An incorrect architected state may not eventually result in an SDC, but the possibility does exist, unlike in the other classes.)

Generally, we refer to the error-masking effects of these four categories as *machine derating* [5], but in the context of Phaser/M1, the scope of derating is primarily the chip microarchitecture, so we may also refer to it here as *microarchitectural derating* (MD). As discussed in Reference [5], there are additional sources of application derating even after an architectural state has been corrupted. Thus, only a subset of the class (d) bit flips actually result in an SDC. In the current Phaser/M1 methodology, we do not attempt to capture the application derating effects; our attempt is to predict the transition probability of bit flips that manifest as class (d) events, and we translate that into a conservative bound on the SDC-specific SER during early-stage microarchitecture definition and analysis. If needed, of course, separate analysis of application-level derating may be done using, for example, a full-system functional simulator (e.g., Mambo, as described in [5]), and the inferred application derating factor can be used to tighten the SDC-related SER bound obtained via Phaser/M1.

Phaser error classification and derating

The SER prediction focus of the Phaser framework is on the MD aspect; the attempt is to predict and allocate derating among the four classes of bit-flip events. In effect, the framework attempts to derate the starting raw SER value of the microarchitecture under study by the first three categories in the MD taxonomy (a, b, and c), while reporting the class (d) bit-flip transition probability as an upper bound for SDC-related SER.

Phaser currently predicts the MD stack in two pairs, namely MD(b,c) and MD(a). The first pair, MD(b,c), combines the SER derating attributable to the recovery and checkstops categories. If the checkers in a chip region are assumed to be active even if that region is not used, then arguably MD(b,c) is workload-independent. MD(a), on the other hand, focuses on the derating contributed by the vanished error class and in general this could be workload-dependent. Then the all-important MD(d) piece used in our SDC SER projection can be computed by subtracting the effects of MD(b,c) and MD(a) from the raw SER value. Given that our current MD(a) fraction of the stack is an underestimate since we are not able to factor in all of the sources of derating attributable to the vanished error class, the MD(d) piece is an overestimate or upper bound of the real value.

Note, by the way, that in our reported methodology, the timing and clock derating factors are not directly addressed. They could in general be implicitly included in the starting raw SER assumptions. Hence Phaser/M1 methodology need not be concerned with these additional sources of derating.

Sample projection and derating analysis for an entire processor core

Figure 5 shows the SER estimation for a two-way multithreaded high-end core running the average SPEC floating-point (SPECfp**) and SPECint workloads, respectively, reported in arbitrary units. In each case, two

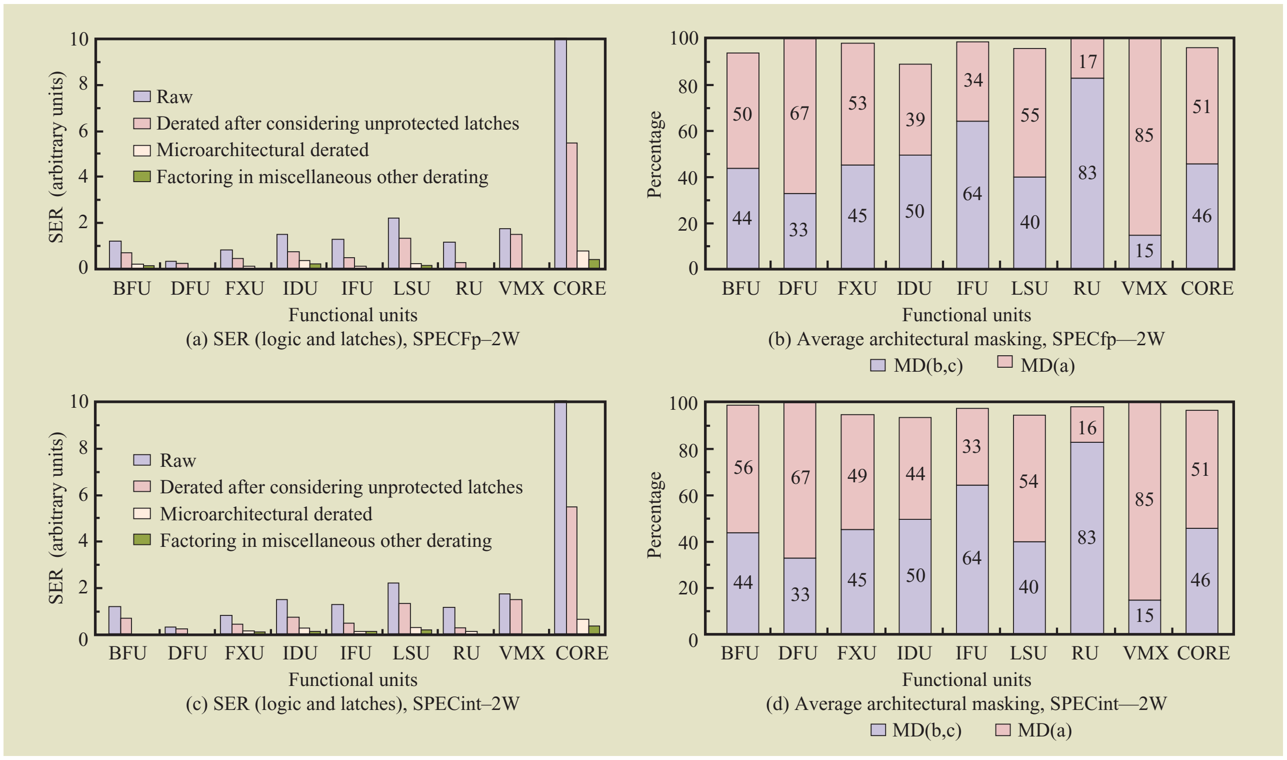

Figure 5

SER estimation for a high-end processor core running the SPECfp (a, b) and SPECint (c, d) average workloads. (2W: two-way; MD: microarchitectural derating.)

versions of the same thread were simulated simultaneously. Figures 5(a) and 5(c) present raw and derated SER for each of the major units in the core and for the full core. Figures 5(b) and 5(d) present a stacked bar of where the various deratings originate for the units and finally for the full core. The raw SER is reported in arbitrary units in such a way that it leads to a net core raw SER value of 10. As depicted in Figures 5(b) and 5(d), the MD(b,c) component derates the core raw SER by 46%. This derating is workload-independent. The MD(a) component derates the SER a further 51%, which brings the total derating of the core to 97% for the average SPECfp or SPECint workload. The real numbers show that for the SPECfp average workload, the core SER goes from 10 (raw) to an SDC-specific SER upper bound of 0.37, while for the SPECint average workload, the core SER goes from 10 (raw) to an SDC-specific SER upper bound of 0.32. Note that the MD(a) values across individual units of the processor do exhibit workload-dependence, but when averaged across the full core, the dependence seems to become very small or negligible in this particular case.

The various units of the core make different contributions to SER. However, the IDU and the LSU proportionally stand out as the units that are most vulnerable to soft errors. This is not surprising since both units exhibit relatively higher data residencies and occupy larger chip areas in terms of latch count. The LSU has the largest chip area among the units in terms of latch count. Although it ranks somewhere in the middle with respect to residency among the units, its area exposure still renders it the most vulnerable. The IDU, on the other hand, is the third largest unit in terms of latch count. However, it showed the highest residency, thus making it the second most vulnerable unit in the group.

As is evident from Figure 5, the data and charts produced by the Phaser framework allow the designer to carry out various what-if experiments and evaluations. For example, it can be seen from the Phaser/M1 data that by replacing the latches and cells used in either the IDU or the LSU with latches and cells that are more hardened against bit flips (resulting in an area and power cost), one can derive a microarchitecture core with a lower SER.

Statistical fault injection validation of Phaser

Validation of higher-level predictive models is a very desirable aspect of early estimation systems. In order to justify the decisions made by higher-level models, it is necessary to compare the accuracy of such models against lower-level detailed models as well as actual hardware measurements. Referring to Figure 1, during the implementation phase of the project, when the design RTL model (M3, coded in a hardware description language such as VHDL or Cadence Verilog**) is available, one needs to conduct experiments at this detailed level of abstraction to validate the conclusions and projections derived during an earlier phase of the project.

In other work—for example, that of Saggese et al. [12] and the ongoing IBM work described in References [5] and [13]—SFI has been widely used with simulation to understand the functional effects of soft-error-induced bit flips. Of particular interest is its use to understand the logic and microarchitectural derating of the chip due to bit flips on the model. The propagation of errors through the system as a result of the injected fault is tracked and classified. Clearly, a benefit of the fault injection work described in References [5] and [13] is to be able to use the SFI methodology to calibrate our predictive Phaser modeling. This calibration process, when completed, will yield a Phaser methodology that can be used with confidence in predicting the derating factors and net SER for future IBM processor designs. Of course, as described in [5], the final point of calibration is by means of the data collected from direct neutron and proton bombardment experiments conducted on the actual chip once it becomes available. The steps outlined below show the incremental, phased validation process pursued in the Phaser project.

During the Phaser/M0 or Phaser/M1 stage of projections, the validation reference is principally the detailed data available from a previous project. For example, in the case of the IBM POWER6* microprocessor, prior measured data from the IBM POWER5* microprocessor that is suitably adjusted to factor in design, technology, and operating point (e.g., voltage) forms the best available reference to verify the new predictive model. Also, analytical bounds of utilization and derating derived from specially designed loop test cases can serve as useful test and calibration reference points. This is similar to techniques used in earlier work for the early-stage validation of M1-level performance simulators [14]. Limited SFI experiments at this high level can also be used to obtain bounds on derating that serve as useful best- and worst-case calibration points.

During the Phaser/M2 stage, the inter-unit interfaces are specified through detailed VHDL descriptions (although the units themselves remain specified through behavioral C/C++ coding). At this stage, the inter-unit interface latch counts and relevant derating factors can be directly measured by scanning the model and by using SFI experiments at this level.

As the RTL M3 VHDL model matures, simulation at this level captures detailed function and timing at the logic and latches. To accelerate simulation speed, hardware emulation [4] methods are used (see the Awan block in Figure 1, described in detail in References [5] and [13]). While the Phaser/M3 predictive model is able to exploit the detailed design information available at this stage of the design, the SFI experiments done on the M3/Awan model serve as an accurate reference for derating factors. This calibrated data can be used to refine the earlier models used within Phaser/M1, for example, to analyze the reliability–performance tradeoff experiments with greater precision and to better help tune the design parameters. The detailed methodology for validation and calibration used in the Phaser project is outside the scope of this paper. References [5] and [13] describe elements of the methodology in some detail. The detailed results of the SFI-based validation exercises (with a focus on the POWER6 microprocessor experience) will be separately documented in future publications.

Conclusion

This paper presents an overview of the Phaser toolset and methodology that is being developed to model the effects of soft errors on the microarchitectural and architectural functionality of a system. We have provided a broad overview of the multiphase framework and have presented a detailed description of the Phaser/M1 stage methodology. The Phaser framework will help develop a fundamental understanding of the system-level effects of SER in the context of current and future generation IBM processors and associated systems.

*Trademark, service mark, or registered trademark of International Business Machines Corporation in the United States, other countries, or both.

**Trademark, service mark, or registered trademark of Standard Performance Evaluation Corporation, Cadence Design Systems, Inc., or Sony Computer Entertainment, Inc., in the United States, other countries, or both.

References

1. S. Borkar, "Designing Reliable Systems from Unreliable Components: The Challenges of Transistor Variability and Degradation," *IEEE Micro* **25**, No. 6, 10–16 (2005).
2. A. KleinOsowski, E. H. Cannon, P. Oldiges, and L. Wissel, "Circuit Design and Modeling for Soft Errors," *IBM J. Res. & Dev.* **52**, No. 3, 255–263 (2008, this issue).
3. M. E. Wazlowski, N. R. Adiga, D. K. Beece, R. Bellofatto, M. A. Blumrich, D. Chen, M. B. Dombrowa, et al., "Verification Strategy for the Blue Gene/L Chip," *IBM J. Res. & Dev.* **49**, No. 2/3, 303–318 (2005).

4. J. M. Ludden, W. Roesner, G. M. Heiling, J. R. Reysa, J. R. Jackson, B.-L. Chu, M. L. Behm, et al., "Functional Verification of the POWER4 Microprocessor and POWER4 Multiprocessor Systems," *IBM J. Res. & Dev.* **46**, No. 1, 53–76 (2002).
5. P. N. Sanda, J. W. Kellington, P. Kudva, R. Kalla, R. B. McBeth, J. Ackaret, R. Lockwood, J. Schumann, and C. R. Jones, "Soft-Error Resilience of the IBM POWER6 Processor," *IBM J. Res. & Dev.* **52**, No. 3, 275–284 (2008, this issue).
6. X. Li, S. V. Adve, P. Bose, and J. A. Rivers, "SoftArch: An Architecture-Level Tool for Modeling and Analyzing Soft Errors," *Proceedings of the International Conference on Dependable Systems and Networks*, Yokohama, Japan, 2005, pp. 496–505.
7. X Li, S. V. Adve, P. Bose, and J. A. Rivers, "Architecture-Level Soft Error Analysis: Examining the Limits of Common Assumptions," *Proceedings of the 37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, Edinburgh, U.K., 2007, pp. 266–275.
8. S. S. Mukherjee, C. Weaver, J. Emer, S. K. Reinhardt, and T. Austin, "A Systematic Methodology to Compute the Architectural Vulnerability Factors for a High-Performance Microprocessor," *Proceedings of the 36th Annual IEEE/ACM International Symposium on Microarchitecture*, San Diego, CA, 2003, pp. 29–40.
9. A. Biswas, P. Racunas, R. Cheveresan, J. Emer, S. S. Mukherjee, and R. Rangan, "Computing Architectural Vulnerability Factors for Address-Based Structures," *Proceedings of the 32nd International Symposium on Computer Architecture*, Madison, WI, 2005, pp. 532–543.
10. P. Kudva, B. Curran, S. K. Karandikar, M. Mayo, S. Carey, and S. S. Sapatnekar, "Early Performance Prediction," *Proceedings of the Workshop on Complexity—Effective Design: Held in Conjunction with the 32nd International Symposium on Computer Architecture*, Madison, WI, 2005; see *http://www.csl.cornell.edu/~albonesi/wced05/wced05.pdf*.
11. H. Q. Le, W. J. Starke, J. S. Fields, F. P. O'Connell, D. Q. Nguyen, B. J. Ronchetti, W. M. Sauer, E. M. Schwarz, and M. T. Vaden, "IBM POWER6 Microarchitecture," *IBM J. Res. & Dev.* **51**, No. 6, pp. 639–662 (2007).
12. G. P. Saggese, N. J. Wang, Z. T. Kalbarczyk, S. J. Patel, and R. K. Iyer, "An Experimental Study of Soft Errors in Microprocessors," *IEEE Microprocessors* **25**, No. 6, 30–39 (2005).
13. P. Kudva, J. W. Kellington, P. N. Sanda, R. McBeth, J. Schumann, and R. Kalla, "Fault Injection Verification of IBM POWER6 Soft Error Resilience," *Proceedings of the Workshop on Architectural Support for Gigascale Integration*, San Diego, CA, 2007; see *http://www.ece.cmu.edu/~asgi/F4.pdf*.
14. P. Bose, "Testing for Function and Performance: Towards an Integrated Processor Validation Methodology," *J. Electronic Testing Theory Application* **16**, No. 1/2, 29–48 (2000).

Received July 17, 2007; accepted for publication August 29, 2007; Internet publication April 3, 2008

Jude A. Rivers *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, New York 10598 (jarivers@us.ibm.com).* Dr. Rivers is a Research Staff Member in the Reliability- and Power-Aware Microarchitectures department. He received a Ph.D. degree in computer science and engineering from the University of Michigan at Ann Arbor in 1998. He then joined IBM Research where he has focused on a wide range of high-performance computer architecture issues and innovations, including hypercaching, efficient instruction and data supply for server systems, power-aware microarchitecture design and analysis, embedded systems, and reliability-aware design and analysis. He has authored several refereed publications, is the author or coauthor of six issued patents, and has more than 20 pending patent applications. He has received several IBM Invention Plateau Awards and a Research Technical Group Award. Dr. Rivers is a member of the IEEE.

Pradip Bose *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, New York 10598 (pbose@us.ibm.com).* Dr. Bose received a Ph.D. degree in electrical and computer engineering from the University of Illinois at Urbana–Champaign in 1983. Since then he has been with IBM Research where he currently manages the Reliability- and Power-Aware Microarchitectures department. He has been involved in the design and presilicon modeling of virtually all IBM POWER microprocessor series. His current research interests are in high-performance computers, power- and reliability-aware microprocessor architectures, presilicon modeling and validation, compilers, and design automation. He is the author or coauthor of more than 70 refereed publications, including several book chapters. He has received several IBM Invention Plateau Awards, a Research Accomplishment Award, and an Outstanding Innovation Award from IBM. He served as Editor-in-Chief of *IEEE Micro* from 2003 to 2006. Dr. Bose is an IEEE Fellow.

Prabhakar Kudva *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, New York 10598 (kudva@us.ibm.com).* Dr. Kudva received a Ph.D. degree in computer science from the University of Utah in 1995. He has been a Research Staff Member at the IBM Thomas J. Watson Research Center since then. He works in the areas of design automation, processor architecture, high-end microprocessor circuit design and methodology, and ASICs. He holds several patents, has published numerous papers, and has served on various conference technical program committees in these areas. He has received several awards including the Outstanding Technical Accomplishment and Research Division Awards from IBM as well as an IEEE/ACM William J. McCalla ICCAD Best Paper Award. Dr. Kudva is an Adjunct Professor at Columbia University.

John-David Wellman *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, New York 10598 (wellman@us.ibm.com).* Dr. Wellman is a Research Staff Member in the Systems Technology and Microarchitecture department. He received a Ph.D. degree in computer science and engineering from the University of Michigan in 1996, at which time he joined IBM Research. He has focused on the modeling and analysis of computer performance and the simulation and verification of digital logic to improve aspects of processor performance. He has participated in the design and implementation of the IBM POWER4* microprocessor, and was a principal member of the IBM Research team that worked with Sony and Toshiba to develop the concept design for what is now the Cell Broadband Engine**. Dr. Wellman is an author or coauthor of more than 20 patents and several papers.

Pia N. Sanda *IBM Systems and Technology Group, 2455 South Road, Poughkeepsie, New York 12601 (sanda@us.ibm.com).* Dr. Sanda received a B.S. degree in engineering physics and a Ph.D. degree in physics, both from Cornell University. She is a Senior Technical Staff Member for server development, working on the soft-error reliability of server systems. She is known for her pioneering work in the development of PICA (picosecond imaging circuit analysis) to measure and visualize the actual switching behavior of high-performance microprocessors to verify timing. She also invented algorithms for phase-shift mask generation which are used in integrated circuit autogeneration tools. Dr. Sanda is a member of the IBM Academy of Technology.

Ethan H. Cannon *IBM Systems and Technology Group, 1000 River Street, Essex Junction, Vermont 05452 (cannon1@us.ibm.com).* Dr. Cannon received a B.S. degree in engineering physics from the University of California, Berkeley, in 1994, and M.S. and Ph.D. degrees in physics from the University of Illinois at Urbana–Champaign, in 1995 and 1999, respectively. After postdoctoral studies at the University of Notre Dame, he joined IBM. Dr. Cannon is currently a reliability engineer focusing on soft-error simulations and measurements.

Luiz C. Alves *IBM Systems and Technology Group, 2455 South Road, Poughkeepsie, New York 12601 (alves@us.ibm.com).* Mr. Alves is a Senior Technical Staff Member working in the System z* Reliability, Availability and Serviceability group. He graduated from New York University in 1975 with a B.S. degree in electrical engineering and received an M.S. degree in electrical engineering in 1977 from the Polytechnic Institute of New York. He joined IBM in 1977 working in the advanced system manufacturing engineering organization, where he held various technical and managerial positions. In 1985 he was named field quality assurance manager for the IBM 3090* system, and in 1987 he became the RAS manager for the 9021 processor families. Mr. Alves is currently responsible for defining the RAS requirements for future System z products.