Capturing Vulnerability Variations for Register Files

Javier Carretero, Enric Herrero, Matteo Monchiero, Tanausú Ramírez, Xavier Vera
Intel Barcelona Research Center, Intel Labs
E-mail: {javier.carretero.casado, enric.herrero, matteo.monchiero,tanausu.ramirez, xavier.vera}@intel.com

***Abstract*—Soft error rates are estimated based on worst-case architectural vulnerability factor (AVF). Therefore, it makes tracking real-time accurate AVF very attractive to computer designers: more accurate AVF numbers will allow turning on more features at runtime while keeping the promised SDC and DUE rates. This paper presents a hardware mechanism based on linear regressions to estimate the AVF (SDC and DUE) of the register file for out-of-order cores. Our results show that we are able to have a high correlation factor at low cost.**

I. Introduction

The exponential growth rate of on-chip transistors, the lower voltages, and the shrinking feature size make current processors vulnerable to transient faults caused by alpha particles and neutrons [1]. Since these transient errors occur due to an incorrect charge or discharge of an intermediate capacitive node, they do not cause permanent failure in the hardware and hence are termed *soft errors* (SER) in the literature.

The FIT rate caused by soft errors of a chip or its components can be measured via accelerated beam tests, which require a functioning chip. A more flexible approach that allows correcting the design to address any possible reliability issue is *modeling* the FIT rate. The model measures the faults that result in an error through the Architectural Vulnerability Factor (AVF) [2]. The AVF of a hardware structure is the probability that a fault at any place in the component will result in an erroneous behavior in the executed program.

Interestingly, FIT and AVF vary over time. For instance, lower voltage increases the soft error rate, while different programs may use resources differently and have different AVF. Therefore, dynamically tracking the AVF instead of using worst-case upper bounds would allow trading reliability for power and performance, while still meeting the FIT requirements. For instance, based on the observed FIT rate, one can switch lockstep modes, or parity and ECC mechanisms *on* and *off*, or even turn *on* and *off* more cores if there is enough available FIT budget.

Previous works have studied how to predict AVF at runtime by correlating different metrics like IPC, number of cache misses, TLB misses, etc, with simulated AVF [3], [4], [5], [6]. These solutions are based on linear regressions and boosted regression trees that run the analysis with more than 200 different variables and pick the most important 10-20 variables. These works show that predictions can get very accurate.

However, previous works do not consider the register files. In this paper, we propose a methodology based on linear regressions to effectively estimate the AVF of register files at runtime in order to guide processor configuration. In summary, the main contributions of this work are:

- **AVF prediction for register file:** we propose a methodology based on linear regressions to predict the AVF for register files.
- **Choice of metrics:** we show how linear regressions based on common performance metrics do not work and identify few simple parameters that can easily predict the AVF.
- **Extensive training and validation:** previous works used only the SPEC programs. We show how training and validating against SPEC programs is not enough in terms of AVF variability. Instead, we use more than 1000 different programs to show the applicability of our methodology.

The rest of the paper is structured as follows: Section II reviews the methodology to calculate AVF. Section III reviews our methodology to estimate AVF of register files based on linear regressions. Section IV discusses the different results and configurations, and demonstrates the efficiency of our approach. Section V reviews some relevant related work. We summarize our main conclusions in Section VI.

II. Background and Experiment Setup

In this section we review how the impact of soft errors is modeled in modern processors.

A. FIT, SER and AVF

Mukherjee et al. [2] introduced the concept of AVF, which is defined as the probability that a bit flip at any place in a processor component will result in an erroneous behavior in the executed program.

Using AVF, we can calculate the $FIT(i)$ for a processor component i (e.g., functional units) as follows:

$$FIT(i) = RawErrorRate \times TVF \times \#bits_i \times AVF_i \quad (1)$$

As a consequence of the complexity of the AVF calculation and its dependency on applications, current FIT and AVF analysis are pessimistic. This implies that in many scenarios, systems are overprotected, wasting resources and power.

B. Runtime AVF Calculation

Runtime AVF calculation can be used to decide if we are meeting the FIT (and AVF) limits at runtime, instead of using the worst-case FIT and AVF scenarios. This opens the door for many runtime reconfigurations; for instance, if the current FIT is very low, one could opt to turn on more cores. Contrary, if

978-3-9815370-0-0/DATE13/©2013 EDAA

the FIT rate is too high, one could decide to turn on an error protection scheme or shut down a core.

In order to take a decision at time *A*, we need to answer the question *"what is the AVF at time A?"*. Biswas et al. [8] proposed the quantized AVF (Q-AVF) approach: instead of using instantaneous AVF (which can introduce too much fluctuation to lead reconfigurations) or the regular average AVF (which would lose the fine-grained variation in AVF), Q-AVF averages the AVF over short intervals (quantums), such as thousand or few millions of cycles.

C. Linear Regressions

Our goal is to correlate at runtime the given Q-AVF of register files with microarchitectural events. The classical linear regression model for Q-AVF uses first order monomials. It yields a set of weights β_i, one for each predictor event f_i:

$$AVF\ \beta_0 + \beta_1 f_1 + \beta_2 f_2 + \ldots + \beta_k f_k \qquad (2)$$

It also yields a coefficient of variations R^2, which measures how well the generated function fits the observed data (larger R^2 indicates a better fit, with 1 taken to mean perfect correlation).

D. Experiment Setup

We have conducted experiments with a processor that resembles the Intel®Core™Micro-Architecture. The particular processor is a 6-way processor (micro-ops), with an instruction cache and first level cache of 32KB, and a second level cache with 512KB. The register files have 160 physical registers, whereas the ROB has 128 entries and the issue queue 32.

To obtain the correlations, we will use three different set of benchmarks. We will start using the well known set of SPEC CPU 2000 benchmarks to describe our proposal. For each benchmark, we pick the most representative simpoint [9].

When assessing our methodology, we will use two different larger set of benchmarks. Out of 6000 different traces, we create two different sets:

- *Training* set. We randomly pick 500 traces representing different kind of benchmarks (it includes SPEC CPU2000, productivity, kernels, office, multimedia, server and workstation applications).
- *Test* set. In this case, we randomly pick another 500 traces that represent the same kind of benchmarks as the *training* set, although they belong to different programs.

Linear regressions analysis have been performed with the R software [10].

III. AVF Prediction of Register Files

This section reviews our proposal for an efficient mechanism to perform runtime AVF prediction of the register file. For the set of experiments shown in this section, we take the whole SPEC2000 suite. We run 10 million instructions, and use a small quantum size of 1024 cycles. We will discuss more results for more configurations in the evaluation section.

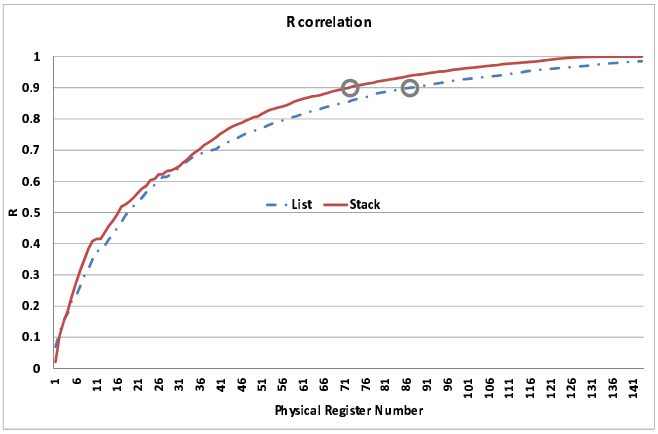


Fig. 1. Correlation factor of Q-AVF of physical registers with Q-AVF of the whole integer register file

A. Tracking Microarchitectural Events

We start by analyzing how well the Q-AVF of the register files correlate with microarchitectural events. For the rest of the section we will run the discussion for the integer register file, and later summarize the results for the floating point (FP) register file.

The set of architectural and microarchitectural events considered is similar to previous works [6], [8]; it includes:

1) Number of executed and committed uops
2) Cache hits and misses to first and second level cache
3) Branch miss-predictions
4) Number of committed load, store, integer and FP instructions
5) Number of stalls at different pipeline stages
6) Utilization of the issue queue and ROB

The results show a very poor correlation between the Q-AVF of the register files with the different events. For instance, if we combine all events we obtain a correlation factor $R^2 = 0.35$ for the integer register file.

B. Tracking Physical Registers

Programs written for a particular instruction set specify operations based on the architectural (or logical) registers; however, out-of-order processors use larger physical register files and renaming mechanisms that hold many copies of a logical register allowing out-of-order execution of instructions.

The Q-AVF of the register file of an out-of-order processor is the Q-AVF of all physical registers. Therefore, we first check if there is a small subset of physical registers that contribute most of the Q-AVF of the whole register file. We show the results in Figure 1. Each point *n* in the chart represents the correlation factor R^2 when correlating the Q-AVF of the whole register file with the Q-AVF of registers up to register number *n*. We analyzed two different implementations of the free list: (i) as a list, and (ii) as stack. We highlight with two circles the points where $R^2 = 0.9$ is achieved. As one can see, we would need to consider between 70-90 registers to correlate

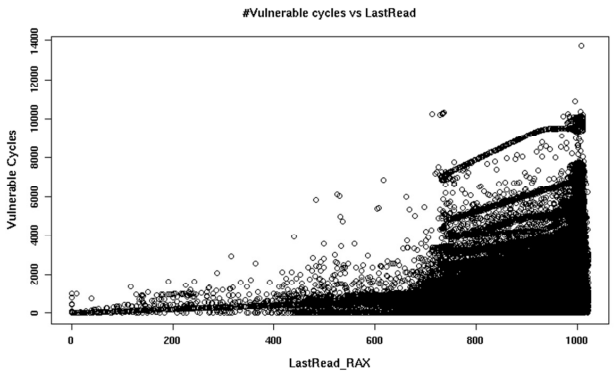


Fig. 2. Q-AVF (in terms of number of vulnerable cycles) distribution for RAX vs LastRead_Entry event

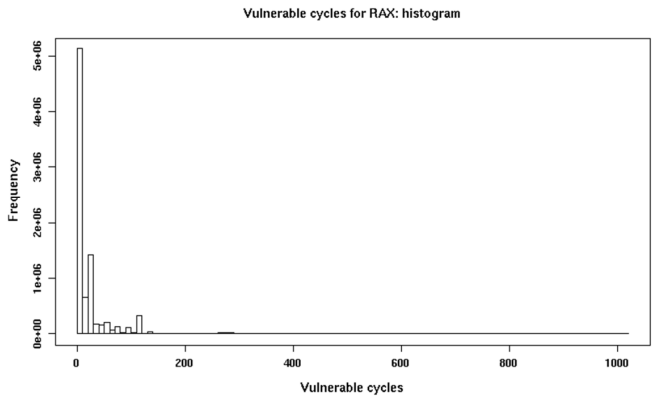


Fig. 3. Histogram for the vulnerability cycles of all physical registers that rename RAX

with reasonable accuracy the Q-AVF of the whole register file, which is impractical.

C. Tracking Architectural Registers

Due to the cost of tracking a large number of physical registers, we considered the possibility of tracking the architectural registers (notice that for an out-of-order processor, the vulnerability of an architectural register is the vulnerability of all physical registers that have renamed it). We started by answering the following question: *can we correlate the Q-AVF of the integer register file with the Q-AVF of few architectural registers?*

Unlike physical registers, we need very few architectural registers to correlate the Q-AVF of the integer register file. Our experiments showed that just tracking RAX, RCX, RDX, RSP, RDI, and TMP0 gives a correlation factor $R^2 = 0.90$.

Once we had a positive answer, the next step was trying to correlate the Q-AVF of an architectural register with some events. First, we tried with the microarchitectural events described earlier. Results were not very encouraging; for instance, if we consider all events, we get a correlation factor $R^2 = 0.54$ for RAX, or $R^2 = 0.19$ for RSP.

We also tried to calculate the time of last read (LastRead_Entry) for every architectural register (which is the max(LastRead_Entry) of all physical registers that rename the architectural register). The results were also very fuzzy; we show in Figure 2 the Q-AVF distribution versus LastRead_Entry for RAX.

D. Tracking Architectural Registers: Histograms

In order to compute the vulnerability of one architectural register of an out-of-order processor we need to track the vulnerability of the physical registers that rename it. We tracked an histogram of the vulnerability cycles for the different renames of architectural registers. We show in Figure 3 the histogram for RAX. As one can see, the vulnerability cycles for all different renames are much clustered, with most of the renames being vulnerable between 0 to 5 cycles.

Therefore, we explored the possibility of correlating the histogram with the vulnerability for the different architectural registers. Later we will discuss how we could implement this in hardware.

We have tried with different bins (ranges) for the histogram. We observed that having just two bins was not giving very good results. Therefore, we moved to four bins. After trying different configurations, we observed that for short-lived architectural registers (like RAX, RDX, etc) bins defined by 0 to 50 cycles, 50 to 75 cycles, 75 to 100 cycles, and larger than 100 cycles worked pretty well. We obtained $R^2 = 0.94$ for RAX and $R^2 = 0.85$ for RDX. For longer-lived registers (registers that may hold values for longer periods of time), like RSP, we used different intervals that reflect their behavior.

Now, with all pieces in place, we can try to answer the last question: *can we use the bins for the different architectural registers to correlate the Q-AVF of the whole register file?* We ran the correlations using the bins from the registers discussed earlier in Section III-C: RAX, RCX, RDX, RSP, RDI, and TMP0. The results showed that we can correlate the Q-AVF of the integer register file with $R^2 = 0.90$.

E. Implementation

One option is to use two counters for each physical register, a *write* and a *read* counter, and one counter per bin. *Write* counter is set once it is written. *Read* counter is set every time the register is read. At commit time, once the physical register is released, we first calculate the distance between the *write* and the *read* counter. Then, we check which architectural register was renaming (this information is available in the ROB entry), and increment the counter of the corresponding bin. At the end of the quantum, we would read the bin counters and use them to correlate the Q-AVF of the register file.

Sampling. One possible optimization consists in tracking only few registers instead of all 160 physical registers. We explored different options:

- Random configurations (only for potential analysis); at commit time, we randomly decide if the register write and read time are considered.
- Fixed registers; we randomly select few registers that are the only ones sampled for the whole execution. We tried 5 (5R) and 10 (10R) registers.

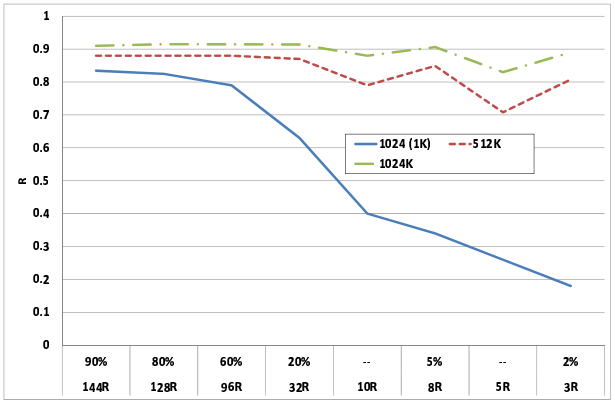


Fig. 4. Impact on the correlation factor of the number of physical registers tracked

We show the results in Figure 4; for the random configurations, we also show the approximate number of registers that are sampled. We first can observe that for the considered quantum (1K cycles), sampling randomly 90% of the register commits still gives good results, with R^2 higher than 0.8. However, quality of the correlation drops as we decrease the number of sampled committed registers. It is also interesting to see that random sampling gives better results that fixed sampling; for instance, randomly sampling 3% of the registers that commit is better that just sampling always the same 5 registers. This is reasonable, since the random selection of the registers allows tracking higher variability.

We also explored different quantum sizes. It is interesting to see that as we increase the quantum size, the impact of sampling less registers is minimized. This is due to the fact that although we sample less registers, we are able to track enough variability of renames for *all* the architectural registers due to the larger quantum sizes. For instance, for a quantum size of 1024K cycles, just sampling 5 registers we can have a correlation factor $R^2 = 0.83$. Sampling only one physical register yields an $R^2 = 0.85$ for quantums of 4 million cycles, and $R^2 = 0.91$ for 8 million cycles quantums (not shown in the Figure).

F. FP Register File

We obtained similar results for the FP register file. Registers `MM0`, `MM1`, `MM2`, `MM3` and `FTMP0` correlate with the Q-AVF of the FP register file with $R^2 = 0.92$. When using the 4 bins with the same ranges described for the integer register file, the correlation that we get is $R^2 = 0.93$.

Regarding the impact of sampling for a quantum size of 1024 cycles, sampling one physical register gives $R^2 = 0.39$, which raises to 0.46 when sampling two registers. Consistent with the integer register file, we need to sample 60% of the physical registers to obtain $R^2 = 0.91$.

IV. Accuracy of Prediction

We have shown how our model is able to predict the Q-AVF for the register files. Now, we will examine the conclusions that we obtained with the SPEC2000 benchmarks and small quantums for other workloads and larger quantum sizes. Then, we will discuss the methodology used for training our model. Finally, we will test our model and demonstrate that we can predict the vulnerability of other workloads.

For the set of experiments shown in this section, we will use three different set of benchmarks. We will use again the whole SPEC2000 suite, but this time we will run 300 million instructions since we will use larger quantums. We will also use the *training* and *test* sets described in Section II-D; for every of the 500 traces we will run 20 million instructions.

A. Variability for Large Quantums

The main purpose behind Q-AVF prediction is leading reconfigurations. Context switch for client systems is between 10-16ms, whereas it is about 100ms for servers. If we assume a processor running at 2GHz, it means that for clients, we have a context switch about every 20 million cycles. We believe that using large quantum sizes in the range of 10-20 millions cycles gives the best tradeoffs. While the variability in Q-AVF is still large enough and opens the door for reconfigurations, the implementation is rather simplified: for instance, sampling just one physical register is enough, and we need to calculate just one equation that involves 24 variables (4 bins for each of the 6 architectural registers). Therefore, we chose 16 million cycles quantums.

B. Predicting Larger Set of Benchmarks

In this section, we examine whether the results we obtained for the SPEC2000 and small quantums still works for larger data sets and larger quantums. We will run the discussion for the integer register file and later we will summarize the results for the FP register file.

Accuracy of binning. We start evaluating the accuracy of the bins that we chose in Section III-D. At this moment, to remove any source of inaccuracy, we sample all physical registers and we measure the actual read and write times. Recall that for this configuration, we obtained $R^2 = 0.90$. For the SPEC2000 programs, now we obtain $R^2 = 0.93$, whereas for the training set, we obtain $R^2 = 0.81$.

As we can see, the accuracy when using only the SPEC2000 is not impacted by the larger quantum size (it is even slightly better). However, using a larger set of programs introduces more variability, and whereas the results are still good, the accuracy of the correlation drops. We have tried other bin ranges, but the results were very similar, with R^2 ranging between 0.80 and 0.83. The same bins for the SPEC2000 programs yielded R^2 in the range 0.89-0.93.

Sampling. We continue evaluating the impact of sampling. For 16 million cycles quantums, we obtained $R^2 = 0.91$ for the SPEC2000 programs. However, when running the *training* set, the correlation is very poor, just $R^2 = 0.21$; in many cases, we have observed that the register we chose to sample was assigned to an architectural register that was not renamed during the quantum, and therefore, we could not obtain any kind of information about the vulnerability of the register file.

TABLE I
R^2 CORRELATION FACTORS FOR DIFFERENT CONFIGURATIONS

	Set 1: RAX, RCX, RDX, RSP, RDI, TMP0					Set 2: RAX, RCX, RDX, RSP, RDI, TMP0, RSI				
	100% PhR	1PhR	2PhR	3PhR	5PhR	100% PhR	1PhR	2PhR	3PhR	5PhR
SPEC2000	0.93	0.91	NoI	NoI	NoI	0.94	0.91	NoI	NoI	NoI
Training	0.81	0.21	0.60	0.63	0.65	0.81	0.21	0.72	0.76	0.79

(a) Integer Register File

	4 BINS: MM0, MM1, MM2, MM3, FTMP0					8 BINS: MM0, MM1, MM2, MM3, FTMP0				
	100% PhR	1PhR	5PhR	10PhR	30PhR	100% PhR	1PhR	5PhR	10PhR	30PhR
SPEC2000	0.92	0.83	0.97	NoI	NoI	0.98	0.96	NoI	NoI	NoI
Training	0.36	NoI	NoI	NoI	NoI	0.80	0.15	0.55	0.58	0.74

(b) FP Register File

If we increase the number of sampled registers to 2, correlation goes up to $R^2 = 0.60$; with 3 registers, $R^2 = 0.63$, and with 5 registers, $R^2 = 0.65$. Interestingly, we noticed that when we track physical registers the vulnerability provided by RSI that was not considered originally grows in importance.

Therefore, we add RSI on top of RAX, RCX, RDX, RSP, RDI, and TMP0. With this new configuration, when we track all physical registers we obtain $R^2 = 0.81$ (we observe no difference compared to previous case when RSI was not considered). When we sample, we still obtain $R^2 = 0.21$ when sampling 1 physical register, but we increase R^2 to 0.76 (from 0.63) for 3 physical registers, and $R^2 = 0.79$ for 5 registers. From now on, we will use this new configuration.

C. Summary of Results for the FP Register File.

In order to predict the vulnerability of the FP register file, we only need to track 5 different architectural registers: MM0, MM1, MM2, MM3 and FTMP0. This set of registers yields $R^2 = 0.95$ for the SPEC2000 and $R^2 = 0.93$ for the training set and 16 million cycles quantums.

Like the case of the integer register file, we also apply four bins with same ranges. In that case, R^2 is 0.92 for the SPEC2000. However, the correlation is very low for the *training* set, with $R^2 = 0.36$. Our analysis shows that unlike the integer registers, FP registers are used in bursts. Therefore, the distance between reuses of registers like MM0 expands across a large range. We opted to increase the resolution of the histogram and moved to 8 bins, which gave better results raising R^2 to 0.80.

Finally, we assess how many registers we need to sample (we assume the 8 bins configuration). Sampling one single register is good enough for the SPEC2000 suite. However, we need to sample 30 registers to obtain $R^2 = 0.74$ for the *training* suite, which we believe is good enough for our purposes.

D. Discussion

We summarize all results for different analyzed configurations in Table I. We use the term *NoI (not of interest)* for those experiments that do not contribute any insight and therefore were not run.

We have observed that the method we described in Section III-A for small quantums works fine for larger quantums, and we can obtain $R^2 = 0.79$ when studying the integer register file (0.74 for the FP) for our larger *training* set of programs.

We have shown that the conclusions obtained for the SPEC2000 programs cannot be automatically extended when we consider a larger set of programs (and larger Q-AVF variability). For instance, for the integer register file we need to also track RSI, and instead of sampling a single register, we need to sample at least 5.

For the FP register file changes are even more important; compared to the results that are valid for the SPEC2000 benchmarks, when considering a larger set of benchmarks we need to increase the number of bins and the number of sampled register to deal with the bursty behavior of the FP programs.

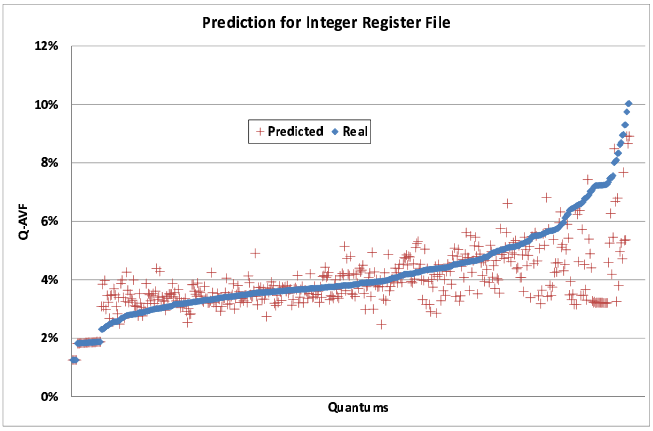
E. Prediction Across Different Benchmark Sets

We further extend our evaluation and assess whether the equations obtained from a set of benchmarks represent the Q-AVF variations for other sets of benchmarks.

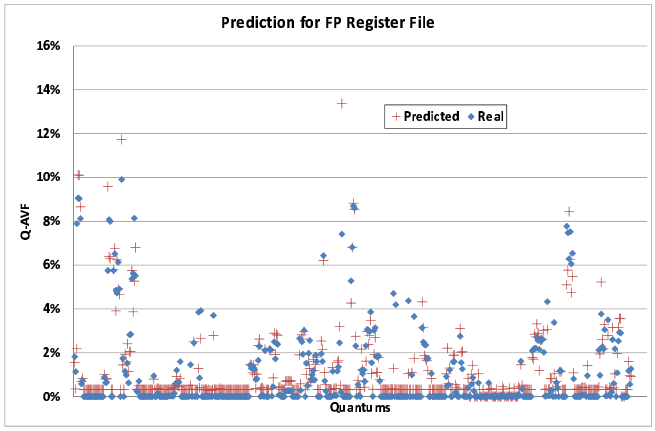
The first observation we make based on previous sections, is that the equations obtained running the SPEC2000 suite are not useful for larger set of benchmarks like the *training* set. As a matter of fact, we even need to tune the methodology to have a good self-correlation.

Next, we evaluate if the equations obtained self-correlating the *training* set can be used to model the Q-AVF for the *test* set. Figure 5 shows the measured and predicted Q-AVF for the test benchmarks using the equations obtained with the training benchmarks. For the integer register file, we sorted the quantums based on the Q-AVF to better show the results due to higher Q-AVF variability. Boxes represent the measured Q-AVF, and crosses the predicted Q-AVF.

As one can see, the results are pretty good, especially if we keep in mind that the idea is to guide reconfigurations. For the integer (FP) register file, we observe an average error of 0.7% (0.4%), with the maximum error being 4.8% (6.0%).



(a) Integer register file



(b) FP register file

Fig. 5. Q-AVF prediction in the out-of-order processor for quantum sizes of 16M cycles for the test benchmarks using the training set equations

V. Related Work

Most studies estimate reliability in terms of architectural vulnerability factor [2]. Most attempts are offline analysis with complex simulators [11], [2], [12]. which are not suitable for online real-time AVF estimation.

There has been some work on estimating the AVF in real time [5], [6]. Walcott [6] et al. use linear regression to explore the relationship between AVF of instruction queue, load store queue and ROB and various microarchitecture level variables such as structure occupancy, number of instructions executed, etc. Duan et al. [3] propose using boosted regression trees as a predictive model. Later, Biswas et al. [8] extend this work by calculating and estimating vulnerability over short windows of time, providing better opportunities for reconfigurations.

Soundararajan et al. [5] propose a method to estimate AVF for the reorder buffer (ROB) in the processor. This method determines the AVF by estimating the occupancy of the instruction queue.

Fu et al. [4] explore program reliability/vulnerability phase behavior. They also explore the AVF estimation for the issue queue and the reorder buffer in an out-of-order processor.

VI. Conclusions

This paper describes a methodology based on linear regressions to effectively estimate the AVF of register files at runtime.

We reason our implementation based on small quantums and using a small data set such as SPEC2000 benchmarks. Later, we show the impact of quantum size. We also discuss the impact of using a more diverse and larger data set to the correlation accuracy, and identify the sources of inaccuracies.

Finally, we modified the implementation and trained our model with more than 500 different programs from different segments like productivity, data base, SPEC benchmarks, multimedia, etc. We tested our model with another set of 500 different programs and show that the prediction is accurate.

Acknowledgments

This work has been partially supported by the EC FET TRAMS (Terascale Reliable Adaptive Memory Systems), 248789.

References

[1] R. Baumann, "Soft errors in advanced computer systems," in *Proceedings of IEEE Design and Test of Computers*. Los Alamitos, CA, USA: IEEE Computer Society, 2005, pp. 258–266.

[2] S. Mukherjee, C. Weaver, J. Emer, S. Reinhardt, and T. Austin, "A systematic methodology to compute the architectural vulnerability factors for a high-performance microprocessor," in *Proceedings of the 36th International Symposium on Microarchitecture (MICRO)*. New York, NY, USA: ACM Press, 2003.

[3] L. Duan, B. Li, and L. Peng, "Versatile prediction and fast estimation of architectural vulnerability factor from processor performance metrics," in *Proceedings of International Symposium on High Performance Computer Architecture (HPCA)*, 2009.

[4] X. Fu, J. Poe, T. Li, and J. Fortes, "Characterizing microarchitecture soft error vulnerability phase behavior," in *Proceedings of International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*, 2006.

[5] N. Soundararajan, A. Parashar, and A. Sivasubramaniam, "Mechanisms for bounding vulnerabilities of processor structures," in *Proceedings of International Symposium on Computer Architecture (ISCA)*, 2007.

[6] K. Walcott, G. Humphreys, and S. Gurumurthi, "Dynamic prediction of architectural vulnerability from microarchitectural state," in *Proceedings of 34th International Symposium on Computer Architecture (ISCA)*, 2007.

[7] X. Vera, J. Abella, J. Carretero, and A. González, "Selective replication: A lightweight technique for soft errors," *ACM Transactions on Computer Systems (TOCS)*, vol. 27, pp. 8:1–8:30, January 2010.

[8] A. Biswas, N. Soundararajan, S. Mukherjee, and S. Gurumurthi, "Quantized avf: A means of capturing vulnerability variations over small windows of time," in *Proceedings of Workshop on Silicon Errors in Logic -System Effects (SELSE)*, 2009.

[9] T. Sherwood, E. Perelman, G. Hamerly, and B. Calder, "Automatically characterizing large scale program behavior," in *Proceedings of International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2002.

[10] "R software project," http://www.r-project.org/.

[11] X. Li, S. Adve, P. Bose, and J. Rivers, "Softarch: An architecture-level tool for modeling and analyzing soft-errors," in *Proceedings of International Conference on Dependable Systems and Networks (DSN)*, 2005.

[12] N. Wang, A. Mahesri, and S. Patel, "Examining ace analysis reliability estimates using fault-injection," in *Proceedings of International Symposium on Computer Architecture (ISCA)*, 2007.