Prevention from Soft Errors via Architecture Elasticity

Yi-Xiao Yin[1,2,3] (尹一笑), Yun-Ji Chen[1] (陈云霁), *Member*, *CCF*, *ACM*, *IEEE*, Qi Guo[4] (郭　崎) and Tian-Shi Chen[1] (陈天石)

[1]*State Key Lab of Computer Architecture and Microprocessor Research Center, Institute of Computing Technology Chinese Academy of Sciences, Beijing 100190, China*

[2]*University of Chinese Academy of Sciences, Beijing 100049, China*

[3]*Loongson Technology Corporation Limited, Beijing 100190, China*

[4]*IBM Research-China, Beijing 100193, China*

E-mail: {yinyixiao, cyj}@ict.ac.cn; joyguoqi@gmail.com; chentianshi@ict.ac.cn

Received November 13, 2013; revised January 21, 2014.

Abstract Due to the decreasing threshold voltages, shrinking feature size, as well as the exponential growth of on-chip transistors, modern processors are increasingly vulnerable to soft errors. However, traditional mechanisms of soft error mitigation take actions to deal with soft errors only after they have been detected. Instead of the passive responses, this paper proposes a novel mechanism which proactively prevents from the occurrence of soft errors via architecture elasticity. In the light of a predictive model, we adapt the processor architectures holistically and dynamically. The predictive model provides the ability to quickly and accurately predict the simulation target across different program execution phases on any architecture configurations by leveraging an artificial neural network model. Experimental results on SPEC CPU 2000 benchmarks show that our method inherently reduces the soft error rate by 33.2% and improves the energy efficiency by 18.3% as compared with the static configuration processor.

Keywords soft error, energy efficiency, architecture elasticity

1 Introduction

As the continuous development of Moore's Law, the functionality and performance of microprocessors have progressed tremendously over the past few decades. However, increasing operating frequencies, lowering threshold voltages[1-3] and shrinking feature size, especially the exponential growth of on-chip transistors[3] are rendering current processors more and more sensitive to errors. Therefore, error resilience has emerged as a key factor in processor design and utilization[1-3].

Soft error is one pervasive cause of computer system failure, induced by energetic particles from cosmic rays and packaging material[1]. Because this type of fault does not result in permanent failure in the hardware, it is termed soft error.

Due to the harmfulness of soft errors, researchers have carried out in-depth investigations on the mitigation of soft errors in the past twenty years. When a soft error has been detected, there are two categories of solutions, i.e., correction and recovery, to cope with it. Plenty of researches, including error correcting codes (ECC)[4] and triple-modular redundancy (TMR)[5], detect the soft errors and subsequently correct them. But most of these studies are limited to memory structures. Different from the correction solutions, when meeting a soft error, the recovery solution rollbacks to the previous fault-free state to evade it. The following are several popular recovery solutions. The high reliability domains, such as the HP Nonstop Advanced Architecture[6] and the IBM zSeries[7], employ large-scale modular redundancy to guarantee excellent fault coverage. Simultaneous redundant threads (SRT) technology[8-9] was proposed to detect soft errors based on simultaneous multithreading (SMT) processors. Muhkerjee *et al.* then introduced chip level redundant threading (CRT)[10] by leveraging CMPs. A number of software approaches[11-12] apply compiler-based instruction duplication in the same execution thread to deliver extremely high fault coverage.

As mentioned above, most traditional schemes of soft error mitigation take action only when a soft er-

Regular Paper

This work is supported by the National Science and Technology Major Project under Grant Nos. 2009ZX01028-002-003, 2009ZX01029-001-003, the National Natural Science Foundation of China under Grant Nos. 61221062, 61100163, 61133004, 61232009, 61222204, 61221062, 61303158, the Strategic Priority Research Program of the Chinese Academy of Sciences under Grant No. XDA06010403, and the Ten Thousand Talent Program of China.

ror has been detected. Instead of passively mitigating soft error when a soft error has happened, this paper proposes a novel mechanism which proactively prevents from the occurrence of soft errors via architecture elasticity[13]. Architecture elasticity is a promising technology which offers the ability to scale down or up the hardware structures when the program is running. By adapting the processor architectures holistically and dynamically, our proposal inherently reduces the soft error rate (SER) and improves the energy efficiency.

Technically, we firstly construct a machine learning model which can quickly and accurately predict the simulation target across different program execution phases on any architecture configuration by employing Artificial Neural Network (ANN) which represents one of the most powerful machine learning algorithms for generalized nonlinear regression. When the program turns into a new phase of execution, we gather a small amount of hardware counters that are capable of characterizing the new phase in a short profiling period. These characteristics combined with several sets of processor architecture configurations are fed into our predictive model to select the best architecture configuration for guiding the architecture elasticity. After the processor has been reconfigured, we proceed to run the program until the next new phase is detected. An example of proactive prevention of soft errors via architecture elasticity is illustrated in Fig.1. The significant increase of data cache miss rate generally means that the current processor architecture configuration has not been able to satisfy the need of the new phase. Owing to cache misses, long delays detain the instructions in the vulnerable structures, such as ROB and issue queue, which probably increases the soft error rate. Moreover, long delays often result in energy wastage and performance losses. Therefore, the SER, performance and power of processor may be better if the cache size is augmented. In fact, it is intractable to decide how to tune the processor architectures for better efficiency and reliability in the situation of involving complex relationships among variable elements.

We have carried out empirical evaluations on our scheme for SPEC CPU 2000 benchmarks. Experimental results show that our proposal reduces the processor SER and energy-delay-squared product by 33.2% and 18.3% respectively, compared with the static configuration processor.

In summary, the main contributions of this paper are the followings:

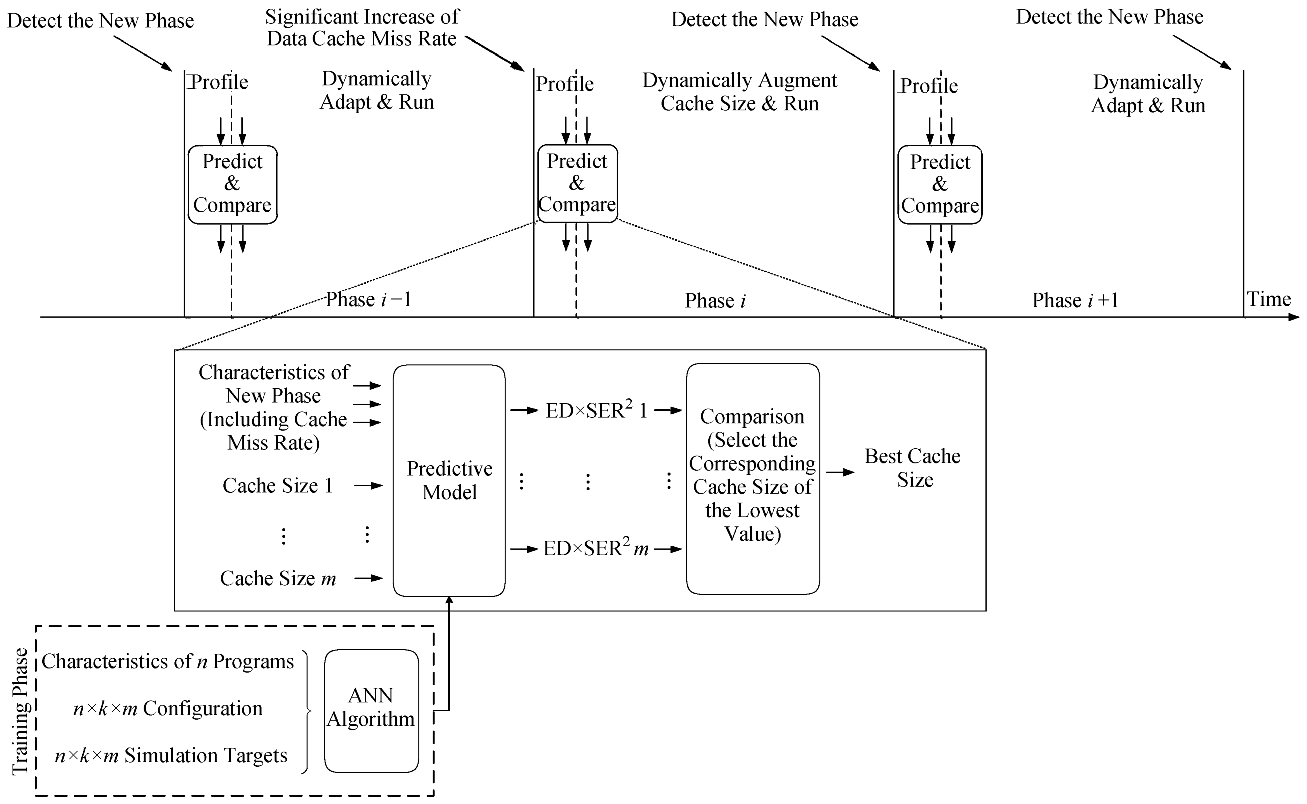

Fig.1. Framework of proactive prevention of soft errors. ED: energy-delay. ED×SER^2: a metric that represents the trade-off between soft error vulnerability and energy efficiency.

• We propose a novel mechanism which proactively prevents from the occurrence of soft errors, significantly different from the traditional correction and recovery mechanisms.

• It is demonstrated that reducing the SER of a single processor structure may increase the SER of other processor structures[14], while our proposal reduces the holistic processor SER, not only a single processor structure.

The rest of this paper is organized as follows. Section 2 introduces a measure method of SER. Section 3 describes our proposed mechanism which reduces the processor SER and improves the energy efficiency. Section 4 presents our experimental setup and Section 5 evaluates our approach. Section 6 lists the related work and finally Section 7 concludes this paper.

2 Background

A chip's raw soft errors are the bit flips or the logic wrong results arising from energetic particle strikes. It depends on the process technology and the number of bits in the structure. Fortunately, not all raw soft errors affect the final outcome of a program. There are numerous researches which show that a large fraction of radiation-induced faults are masked[3,15]. Mukherjee *et al.* introduced architectural vulnerability factor (AVF)[2] to quantify the architectural masking effect of raw soft errors. A hardware structure's AVF is the probability that a fault in the structure will result in a visible error in the final output of a program. Because of the masking capability of the AVF, the effective SER of a processor structure is defined as (1).

$$\text{effective SER}=\text{raw SER} \times \text{AVF}. \tag{1}$$

In this study, we measure the AVF of processor microarchitecture structures using Architecturally Correct Execution (ACE) analysis[2,16]. A bit in a structure can be classified as either ACE or un-ACE at a certain point in time. An ACE bit is one whose correctness is required for the architecturally correct execution. The two types of un-ACE bits are microarchitectural un-ACE bits and architectural un-ACE bits. Microarchitectural un-ACE bits arise from idle or invalid state and bits in predictor structures. Architectural un-ACE bits contain NOP instructions, prefetch instructions, predicated-false instructions, and dynamically dead instructions. The AVF of a structure is the ratio of the number of ACE bits per cycle in the structure to the total number of bits in the structure. The overall SER of the processor is obtained by adding the SER of all processor structures. For hardware structure E with size m, its AVF over a period of n cycles can be expressed as (2).

$$AVF_E = \frac{\sum_{i=1}^{n} \text{Number of ACE bits in } E \text{ at cycle } i}{m \times n}. \tag{2}$$

3 Mechanism

We construct an artificial neural network model which could predict the simulation target (the output of the ANN model) quickly and accurately by inputting a small amount of hardware counters into the ANN model. The hardware counters are capable of characterizing a new program phase. Based on the predicted results, we select the best architecture configuration from an architecture design space for the program phase. We assume that the simulation target value of the best architecture configuration is the lowest.

The artificial neural network (ANN) is a machine learning model that automatically learns to predict a target (simulation result in our case) based on n puts. Fig.2 shows the most common type of fully connected, feed-forward ANN. The network consists of three layers: input layer, output layer, and hidden layer. Each unit operates on its inputs to produce an output that is passed to the next layer. An input unit passes its input value to all hidden units presented at the hidden layer via a set of weighted edges. Hidden and output units calculate their output by first taking a weighted sum of their inputs based on the edge weights, and transferring this sum to a non-linear activation function. The target prediction is obtained from the output layer. The commonly used activation functions must be nonlinear, monotonic and differentiable, which include sigmoid, linear threshold function, and radial basis function. Fig.3 shows a hidden unit using the sigmoid activation function that is applied to our predictive model. The symbol w in Fig.3 is the model edge weight, I is the input value, x is the weighted sum and $f(x)$ is the sigmoid activation function. ANN represents one of the most powerful machine learning models for generalized

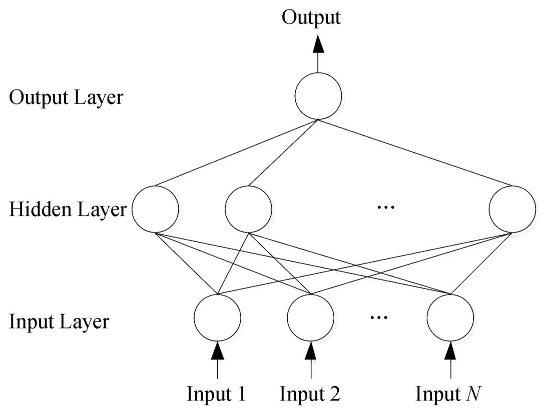

Fig.2. Basic architecture of a fully connected, feed-forward ANN.

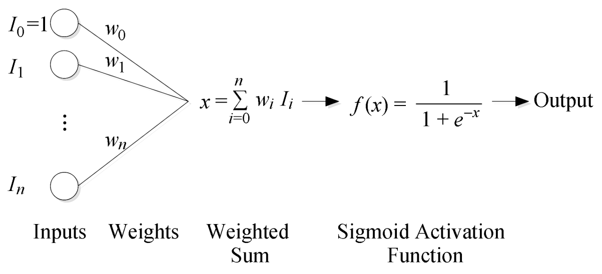
I_0=1
I_1
I_n
w_0
w_1
w_n
$x = \sum_{i=0}^{n} w_i I_i$
$f(x) = \frac{1}{1+e^{-x}}$
Output
Inputs
Weights
Weighted Sum
Sigmoid Activation Function

Fig.3. Single unit of an ANN.

nonlinear regression, and delivers accurate result in the situation of involving complex relationships among variables.

We utilize cross validation to set the tunable parameters of the ANN model for predicting the simulation target accurately. Cross validation is a mechanism which takes advantage of all available data to train predictive model so that it reduces the risk of overfitting. In cross validation, the dataset is divided into N equal-sized folds. The model is trained on $N-1$ folds and tested on the remaining fold, then the selection of folds as testing data is rotated.

4 Experimental Methodology

4.1 Architecture Design Space

This paper proposes a scheme to quickly and accurately select the best architecture configuration from an architecture design space for any new phase of a program. The configurable architectural parameters that we have considered are shown in Table 1 and are similar to the parameters other researches have thought over[17-19]. Table 1 shows the variation ranges and steps of these parameters and the number of different values they can take where "+" and "$*$" represent steps. The baseline machine configuration used in this study represents a high-performance out-of-order processor and is shown in Table 2.

Table 1. Architectural Design Parameters Variation for Model Training

Parameter	Value Range	Step	Number of Values
Machine width	2, 4, 8		3
Fetch queue size	4, 8, 16		3
ROB size	32∼160	16+	9
Issue queue size	8∼72	16+	5
Load/store queue size	8∼72	16+	5
Gshare size	1 K∼32 K	2*	6
BTB size	1 K, 2 K, 4 K		3
L1 Icache size	8 K∼128 K	2*	5
L1 Dcache size	8 K∼128 K	2*	5
L2 Ucache size	256 K∼4 M	2*	5

We vary 10 different parameters across their range of values in a superscalar simulator. However, it would be impractical to select the best architecture configuration online from a design space as large as this. Therefore, due to the implementation and online search overheads, we vary 5 important parameters in a small range and remain the other parameters as same as in the baseline configuration to construct an online design space of 162 points. The online space shown in Table 3 contains typical and feasible design points.

Table 2. Baseline Architecture Configuration

Parameter	Configuration
Pipeline width	4
Fetch queue	8 entries
Issue queue	32 entries
ROB	56 entries
Load/store queue	32 entries
Gshare	2 K entries
BTB	2 K entries, 4-way
L1 Icache	16 KB, 4-way, 32 B blocks, 1 cycle latency
L1 Dcache	16 KB, 4-way, 32 B blocks, 1 cycle latency
L2 Ucache	2 MB, 4-way, 64 B blocks, 6 cycle latency

Table 3. Architectural Design Parameters Variation for Online Adaptation

Parameter	Value Range	Number of Values
ROB size	32, 56, 80	3
Issue queue size	16, 32, 48	3
L1 Icache size	16 K, 32 K, 64 K	3
L1 Dcache size	16 K, 32 K, 64 K	3
L2 Ucache size	512 K, 2 M	2

4.2 Simulator and Benchmarks

Our cycle-accurate simulator is based on Wattch[20] (an extension to SimpleScalar) and includes detailed energy model for the processor. We split RUU of SimpleScalar into a reorder buffer and an issue queue. Moreover, we add a fetch queue and register files. In order to model soft error vulnerability, we integrate the AVF computation methods proposed in [2, 16] into Wattch. Our reliability simulator covers a wide range of significant architecture components: an issue queue, function units, a reorder buffer, and a load/store queue.

We use 21 SPEC CPU 2000 benchmarks[21] compiled with the highest optimization level to evaluate our methods with the reference input set. For each benchmark, we abstract the representative traces using SimPoint[22] with an interval size of 10 million instructions. At the end of each interval, the soft error vulnerability estimate along with performance and energy values can be gained.

4.3 Hardware Counters

We extract several hardware counters used to characterize a program phase. The most important hardware counters are listed as follows: Il1 accesses, Il1 miss rate, Dl1 accesses, Dl1 miss rate, branch instructions rate, branch mis-prediction rate, load/store queue occupant rate, ROB occupant rate, fetch queue occupant rate, issue queue occupant rate, load/store instructions rate, and ACE instructions rate. For example, the high queue occupant rate usually means the soft error rate of the queue is high, because the ratio of ACE bits per cycle in the queue may be high.

4.4 Metric

The energy-delay (ED) product is widely used in architecture design[18]. We use ED $\times$ SER^2 as a metric that represents the trade-off between soft error vulnerability and energy efficiency. The lower the metric value, the better.

4.5 Model Construction Methods

We use seven integer benchmarks (vpr, gcc, mcf, parser, perlbmk, gap, twolf) and seven floating point benchmarks (wupwise, swim, mgrid, applu, mesa, ammp, lucas). Each benchmark is run for the first SimPoint trace. We uniformly and randomly sample 1 000 architecture configurations from the entire design space. The characteristics of the first SimPoint trace of each benchmark, 1 000 random architecture configurations, and the corresponding simulation target are used to build our ANN model. We apply 7-fold cross validation to build the appropriate ANN model by adjusting parameters, such as learning rate and momentum.

5 Experimental Results

5.1 Accuracy of Our Predictive Model

We use the relative mean absolute error (*rmae*) defined as (3) to measure the accuracy of our ANN model.

$$rmae = \left|\frac{value_{\text{pred}} - value_{\text{real}}}{value_{\text{real}}}\right| \times 100\%, \qquad (3)$$

where $value_{\text{pred}}$ represents the value predicted by our model and $value_{\text{real}}$ represents the real value.

To evaluate the accuracy of our ANN model, we randomly and independently generate 100 sample points for the second SimPoint trace of each benchmark mentioned above (Fig.4(b)), the first SimPoint trace of other two integer benchmarks (eon and vortex), and two floating point benchmarks (sixtrack and galgel) (Fig.4(a)), respectively. Fig.4 demonstrates that our ANN model accurately predicts the simulation target of untrained programs and future execution phases.

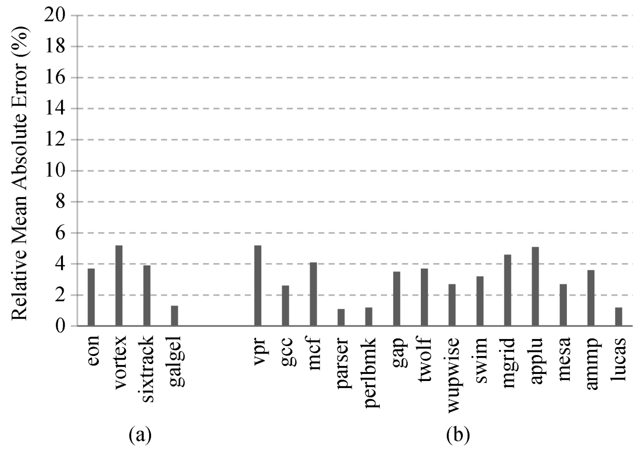

Fig.4. Relative mean absolute error when predicting the target for untrained programs (a) and future execution phases (b).

5.2 Reduction of SER and Improvement of Efficiency Using Our Proposal

When the new phase of a program has been detected, we gather the characteristics in the profiling period and feed them together with sets of architecture configurations within the entire online design space into the constructed predictive model and then select the best architecture configuration. After that, we continue running the current phase with the selected configuration. The process is repeated until all the program's phases have been exhausted. Both processor SER and energy efficiency are evaluated in this subsection. We assume an arbitrary intrinsic fault rate of 0.01 units/bit[23] and compute the SER for most significant processor architecture components. Fig.5 depicts the results achieved by our approach as compared with the baseline configuration. On average, we decrease the processor SER by

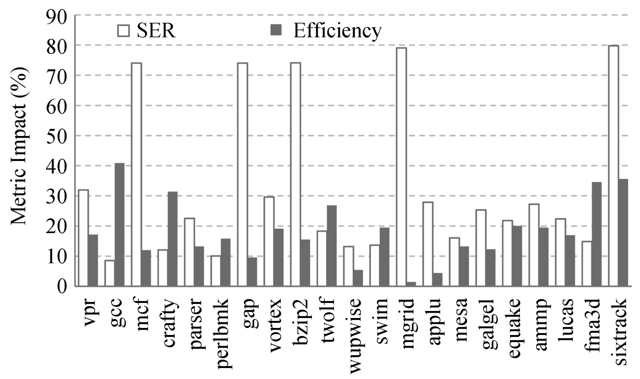

Fig.5. Reduction of SER and improvement of energy efficiency achieved by our proposal compared with the static baseline configuration for SPEC CPU 2000 benchmarks.

more than 33.2%, relative to the static baseline configuration. In some cases, for mcf, gap, bzip2, mgrid and sixtrack, we achieve a remarkable reduction in processor SER even more than 70%. Meanwhile, we improve the energy efficiency by 18.3% on average. Especially for gcc, crafty, fma3d and sixtrack, their energy-delay-squared product has been reduced by over 30%. For these benchmarks, the L2 cache is usually not being fully utilized, so we diminish their L2 cache size from 2 M to 512 K so as to reduce the power consumption.

In this study, reconfiguration only occurs once every 10 million instructions. Thus, the reconfiguration overheads for the whole program execution are significantly reduced. Moreover, the majority of reconfiguration time is hidden because transistors can be powered up and down when the processor components are being used[17,24]. In addition, the overheads for the implementation of the ANN model used in this study are negligible[25].

6 Related Work

Recovery Solutions. n-Modular Redundancy (nMR)[6] guarantees excellent fault coverage, but it incurs 100% hardware and energy consumption, and loses 50% throughput. Instead of full processor duplication, using simple checkers[26] or duplicating copies of a processor component[27] is often performed. Nevertheless, they are usually hardwired at the design time. Simultaneous redundant threads[8] augment SMT processors for transient fault detection. Because of resource contention between the leading thread and trailing thread, Muhkerjee *et al.* introduced chip level redundant threading (CRT)[10]. Recent studies[9,28] provide partial redundancy mechanisms. For instance, [28] avoids redundant execution for low AVF program phases. There has been a surge of researches[7,29-31] that exploit anomalous microarchitectural behavior to detect soft errors. The symptoms include memory exceptions, branch misprediction, cache misses, etc. Most recovery solutions[29] rely upon the heavyweight, full-system checkpointing mechanisms. Furthermore, the recovery solutions must be based on the detection of soft errors. But our proposal proactively prevents from the occurrence of soft errors and improves the energy efficiency.

Correction Solutions. Memory structures, like caches and main memory, are protected against faults using parity or ECC[4]. A cache line is extended with parity or ECC, then every read requires error detection or correction and every write needs parity or ECC encoding. Most of the correction solutions are only suitable for memory structures, due to the prohibitive costs.

Reducing SER Directly Without Recovery and Correction. There has been a small amount of researches[15-16,32] which reduce the SER directly by keeping valid state out of vulnerable structures. When cache-miss is encountered, [32] squashes instructions that sit in the instruction queue. These methods are limited to a single structure. Nevertheless, our proposal reduces the holistic SER of processor.

Time Varying Characteristic, Phase Detection and Online Estimation. Previous researches[17,23] showed that AVF, performance and power exhibit time varying behavior both across and within programs. Online phase detection techniques[22,33] were proposed to detect and classify program execution that shows homogeneous characteristics of interest. Walcott *et al.*[28] and Duan *et al.*[34] demonstrated that AVF can be estimated online by utilizing a small set of microarchitectural metrics. In this study, we consider reliability, performance, and power consumption simultaneously and predict the specified simulation target quickly and accurately at runtime.

Architecture Elasticity Technology. How to dynamically tune the processor structures or multiprocessor for efficiency has received much attention in the past few years. On the one hand, the studies for issue queue[24], reorder buffer[35] and caches[36], aim at single component of the processor by leveraging control mechanisms. On the other hand, there are numerous proposals that use reconfigurable processor to satisfy both single-threaded programs and multi-threaded programs simultaneously[37-39].

7 Conclusions and Future Work

This paper proposed a novel mechanism which proactively prevents from the occurrence of soft errors. We built a machine learning model which can quickly and accurately predict the simulation target across different program execution phases on any architecture configurations by leveraging an artificial neural network model. Based on the constructed predictive model, we selected the best architecture configuration used to guide the holistic and dynamic adaptation of processor architectures. Compared with the static configuration processor, our proposal reduces soft error rate by 33.2% and improves energy efficiency by 18.3%. Although this paper has targeted a uniprocessor design, our approach probably can be extended to a multiprocessor design.

References

[1] Baumann R. Radiation-induced soft errors in advanced semiconductor technologies. *IEEE Transactions on Device and Materials Reliability*, 2005, 5(3): 305-316.

[2] Mukherjee S S, Weaver C, Emer J, *et al.* A systematic methodology to compute the architectural vulnerability factors for a high-performance microprocessor. In *Proc. the*

36th Annual IEEE/ACM International Symposium on Microarchitecture (*MICRO2003*), December 2003, p.29.

[3] Shivakumar P, Kistler M, Keckler S *et al.* Modeling the effect of technology trends on the soft error rate of combinational logic. In *Proc. the International Conference on Dependable Systems and Networks* (*DSN2002*), June 2002, pp.389-398.

[4] Mitra S, Seifert N, Zhang M, Shi Q *et al.* Robust system design with built-in soft-error resilience. *IEEE Computer*, 2005, 38(2): 43-52.

[5] Lyons R E, Vanderkulk W. The use of triple-modular redundancy to improve computer reliability. *IBM Journal of Research and Development*, 1962, 6(2): 200-209.

[6] Bernick D, Bruckert B, Vigna P D *et al.* Nonstop® advanced architecture. In *Proc. Int. Conf. Dependable Systems and Networks* (*DSN2005*), June 28-July 1, 2005, pp.12-21.

[7] Li M L, Ramachandran P, Sahoo S K *et al.* Understanding the propagation of hard errors to software and implications for resilient system design. In *Proc. the 13th Int. Conf. Architectural Support for Programming Languages and Operating Systems* (*ASPLOS2008*), March 2008, pp.265-276.

[8] Rotenberg E. AR-SMT: A microarchitectural approach to fault tolerance in microprocessors. In *Proc. the 29th Int. Symp. Fault-Tolerant Computing* (*FTCS1999*), June 1999, pp.84-91.

[9] Gomaa M, Vijaykumar T. Opportunistic transient-fault detection. In *Proc. the 32nd International Symposium on Computer Architecture* (*ISCA2005*), June 2005, pp.172-183.

[10] Mukherjee S S, Kontz M, Reinhardt S K. Detailed design and evaluation of redundant multi-threading alternatives. In *Proc. the 29th International Symposium on Computer Architecture* (*ISCA2002*), May 2002, pp.99-110.

[11] Wang C, Kim H, Wu Y, Ying V. Compiler-managed software-based redundant multi-threading for transient fault detection. In *Proc. the International Symposium on Code Generation and Optimization* (*CGO2007*), March 2007, pp.244-258.

[12] Rehman S, Shafique M, Henkel J. Instruction scheduling for reliability-aware compilation. In *Proc. the 49th Design Automation Conference* (*DAC2012*), June 2012, pp.1292-1300.

[13] Chen Y J, Chen T S, Guo Q *et al.* An elastic architecture adaptable to millions of application scenarios. In *Proc. the 9th IFIP International Conference on Network and Parallel Computing* (*NPC2012*), Sept. 2012, pp.188-195.

[14] Duan L, Zhang Y, Li B, Peng L. Universal rules guided design parameter selection for soft error resilient processors. In *Proc. Int. Symp. Performance Analysis of Systems and Software* (*ISPASS2011*), April 2011, pp.247-256.

[15] Soundararajan N, Parashar A, Sivasubramaniam A. Mechanisms for bounding vulnerabilities of processor structures. In *Proc. the 19th International Symposium on Computer Architecture* (*ISCA2007*), June 2007, pp.506-515.

[16] Biswas A, Cheveresan R, Emer J *et al.* Computing architectural vulnerability factors for address-based structures. In *Proc. the 32nd International Symposium on Computer Architecture* (*ISCA2005*), June 2005, pp.532-543.

[17] Dubach C, Jones T M, Bonilla E V, Boyle M F P O. A predictive model for dynamic microarchitectural adaptivity control. In *Proc. Annual IEEE/ACM International Symposium on Microarchitecture* (*MICRO2010*), Dec. 2010, pp.485-496.

[18] Dubach C, Jones T, O'Boyle M. Microarchitectural design space exploration using an architecture-centric approach. In *Proc. Annual IEEE/ACM International Symposium on Microarchitecture* (*MICRO2007*), Dec. 2007, pp.262-271.

[19] Guo Q, Chen T S, Chen Y J *et al.* Effective and efficient microprocessor design space exploration using unlabeled design configurations. In *Proc. Int. Joint Conf. Artificial Intelligence* (*IJCAI2011*), July 2011, pp.1671-1677.

[20] Brooks D, Tiwari V, Martonosi M. Wattch: A framework for architectural-level power analysis and optimizations. In *Proc. International Symposium on Computer Architecture* (*ISCA2000*), June 2000, pp.83-94.

[21] Henning J L. SPEC CPU2000: Measuring CPU performance in the new millennium. *IEEE Computer*, 2000, 33(7): 28-35.

[22] Sherwood T, Perelman E, Hamerly G, Calder B. Automatically characterizing large scale program behavior. In *Proc. International Conference on Architectural Support for Programming Languages and Operating Systems* (*ASPLOS2002*), October 2002, pp.45-57.

[23] Nair A A, Eyerman S, Eeckhout L, John L K. A first-order mechanistic model for architectural vulnerability factor. In *Proc. International Symposium on Computer Architecture* (*ISCA2012*), Oct. 2012, pp.273-284.

[24] Buyuktosunoglu A, Albonesi D, Schuster S *et al.* A circuit level implementation of an adaptive issue queue for power-aware microprocessors. In *Proc. the 11th Great Lakes Symposium on VLSI* (*GLSVLSI2001*), March 2001, pp.73-78.

[25] Bitirgen R, Ipek E, Martinez J. Coordinated management of multiple interacting resources in chip multiprocessors: A machine learning approach. In *Proc. Annual IEEE/ACM International Symposium on Microarchitecture* (*MICRO2008*), November 2008, pp.318-329.

[26] Meixner A, Bauer M, Sorin D. Argus: Low-cost, comprehensive error detection in simple cores. *IEEE Micro*, 2008, 28(1): 52-59.

[27] Vadlamani R, Zhao J, Burleson W, Tessier R. Multicore soft error rate stabilization using adaptive dual modular redundancy. In *Proc. IEEE Design, Automation and Test in Europe Conference & Exhibition* (*DATE2010*), March 2010, pp.27-32.

[28] Walcott K R, Humphreys G, Gurumurthi S. Dynamic prediction of architectural vulnerability from microarchitectural state. In *Proc. the 34th International Symposium on Computer Architecture* (*ISCA2007*), May 2007, pp.516-527.

[29] Racunas P, Constantinides K, Manne S, Mukherjee S S. Perturbation-based fault screening. In *Proc. the 13th Int. Symp. High Performance Computer Architecture* (*HPCA2007*), Feb. 2007, pp.169-180.

[30] Wang N, Patel S. ReStore: Symptom-based soft error detection in microprocessors. *IEEE Transactions on Dependable and Secure Computing*, 2006, 3(3): 188-201.

[31] Feng S, Gupta S, Ansari A, Mahlke S. Shoestring: Probabilistic soft error reliability on the cheap. In *Proc. the 15th International Conference on Architectural Support for Programming Languages and Operating Systems* (*ASPLOS2010*), March 2010, pp.13-17.

[32] Weaver C, Emer J, Mukherjee S S, Reinhardt S K. Techniques to reduce the soft error rate of a high-performance microprocessor. In *Proc. Annual International Symposium on Computer Architecture* (*ISCA2004*), June 2004, pp.264-275.

[33] Cho C, Zhang W, Li T. Informed microarchitecture design space exploration using workload dynamics. In *Proc. Annual IEEE/ACM International Symposium on Microarchitecture* (*MICRO2007*), December 2007, pp.274-285.

[34] Duan L, Li B, Peng L. Versatile prediction and fast estimation of architectural vulnerability factor from processor performance metrics. In *Proc. the 15th International Symposium on High Performance Computer Architecture* (*HPCA2009*), February 2009, pp.129-140.

[35] Abella J, Gonzalez A. On reducing register pressure and energy in multiple-banked register files. In *Proc. International Conference on Computer Design* (*ICCD2003*), October 2003, pp.14-20.

[36] Balasubramonian R, Albonesi D, Buyuktosunoglu A, Dwarkadas S. Memory hierarchy reconfiguration for energy

and performance in general-purpose processor architectures. In *Proc. Annual IEEE/ACM International Symposium on Microarchitecture* (*MICRO2000*), December 2000, pp.245-257.

[37] Ipek E, Kirman M, Kirman N, Martinez J F. Core fusion: Accommodating software diversity in chip multiprocessors. In *Proc. Annual International Symposium on Computer Architecture* (*ISCA2007*), June 2007, pp.186-197.

[38] Watanabe Y, Davis J D, Wood D A. Widget: Wisconsin decoupled grid execution tiles. In *Proc. Annual International Symposium on Computer Architecture* (*ISCA2010*), June 2010, pp.2-13.

[39] Khubaib K, Suleman M A, Hashemi M *et al.* MorphCore: An energy-efficient microarchitecture for high performance ILP and high throughput TLP. In *Proc. the Annual IEEE/ACM International Symposium on Microarchitecture* (*MICRO2012*), Dec. 2012, pp.305-316.

Yi-Xiao Yin is currently a Ph.D. candidate of Institute of Computing Technology, Chinese Academy of Sciences, Beijing. She received the B.S. degree in computer science from University of Science and Technology of Changsha, China, in 2005. Her main research interests include computer architecture, soft error reliability analysis and prediction, fault tolerance, and reconfigurable devices.

Yun-Ji Chen graduated from the Special Class for the Gifted Young, University of Science and Technology of China, Hefei, in 2002. He received the Ph.D. degree in computer science from Institute of Computing Technology (ICT), Chinese Academy of Sciences (CAS), Beijing, in 2007. He is currently a professor at ICT. His research interests include parallel computing, microarchitecture, hardware verification, and computational intelligence.

Qi Guo received the B.S. degree in computer science from Department of Computer Science and Technology, Tongji University, Shanghai, in 2007, and the Ph.D. degree in computer science from Institute of Computing Technology (ICT), Chinese Academy of Sciences (CAS), Beijing, in 2012. He is currently a staff researcher at IBM Research-China, Beijing. His research interests include computer architecture, VLSI design and verification.

Tian-Shi Chen received the B.S. degree in mathematics from the Special Class for the Gifted Young, University of Science and Technology of China (USTC), Hefei, in 2005, and the Ph.D. degree in computer science from Department of Computer Science and Technology, USTC, in 2010. Currently, he is an associate professor at ICT, CAS. His research interests include computer architecture, parallel computing, and computational intelligence.