Path-Based Error Coverage Prediction

Joakim Aidemark, Peter Folkesson, and Johan Karlsson
Chalmers University of Technology
S-412 96 Göteborg, Sweden
+46 31 772 5225, 46 31 772 3663 fax
{aidemark, peterf, johan} @ce.chalmers.se

Abstract

Previous studies have shown that error detection coverage and other dependability measures estimated by fault injection experiments are affected by the workload. The workload is determined by the program executed during the experiments, and the input sequence to the program. In this paper, we present a promising analytical post-injection prediction technique, called Path-Based Error Coverage Prediction, which reduces the effort of estimating error coverage for different input sequences. It predicts the error coverage for one input sequence based on fault injection results obtained for another input sequence. Although the accuracy of the prediction is low, Path Based Error Coverage Prediction manages to correctly rank the input sequences with respect to error detection coverage, provided that the difference in the actually coverage is significant. This technique may drastically decrease the number of fault injection experiments, and thereby the time, needed to find the input sequence with the worst-case error coverage among a set of input sequences.

1. Introduction

Computer systems are increasingly being used in applications that require high dependability. To be confident that these systems deliver the correct service, they need to be validated. Both analytical and experimental techniques are required to fully validate a dependable system. Fault injection [1] is an experimental technique that has shown to be an important means for validating dependable systems. It can be used to study a system in the presence of faults and to identify dependability bottlenecks. One main goal is to determine the coverage of the error detection mechanisms. The error detection coverage can then be used in an analytical model to calculate the reliability, availability and safety of a system.

Studies have shown that the results from a fault injection experiment do not only depend on the hardware, but also on the program executing on the system [2]. Further, the result may also be affected by the input sequence to the program [3], [4], [5]. Therefore, when validating a dependable system, a workload similar to the one used when the system is in operation should be chosen. However this may pose a problem, since the number of input sequences that need to be considered can be very large.

One way to reduce the effort of validating dependable computer systems is through pre-injection analysis and post-injection analysis. Pre-injection analysis is performed before any fault injection is performed to focus fault injection to specific parts of the fault space [6], [7], [8]. In post injection analysis, the results from fault injection experiments are used to predict the outcome of other experiments in order to speed up the validation process [2].

We have previously investigated two post-injection techniques for predicting error coverage called Execution Profile Based Prediction and Data Usage Based Prediction [5]. These techniques can predict variations in error coverage for different input sequences. They predict the error coverage for one input sequence using fault injection results obtained for another input sequence. Execution Profile Based Prediction relies on the fact that workload input variations cause different parts of a program to be executed different number of times. Data Usage Based Prediction takes into account that the input variations alter the usage of data. We used these techniques for predicting the coverage for several hardware implemented error detection mechanisms included in a microprocessor with respect to single bit-flips injected in the microprocessor. Our experiments showed that the prediction techniques could correctly rank input sequences according to error coverage provided that the difference in the actual coverage is significant, although the accuracy of the predicted coverage figures were low. In order to achieve the ranking it was necessary to combine the two techniques, using Execution Profile Based Prediction for bit-flips in registers and flip-flops, and Data Usage Based Prediction for bit-flips in the cache. These prediction

techniques were primarily intended for fault injection techniques such as scan-chain implemented fault injection (SCIFI) or software implemented fault injection (SWIFI), where the observability of the fault activation is low.

In this paper, we present a new post-injection prediction technique called Path-Based Error Coverage Prediction. This technique is intended for simulation-based fault injection and utilizes the high observability available in simulations. In contrast to our previous prediction techniques, Path-Based Error Coverage Prediction show promising results for predicting the coverage for errors occurring in all parts of a microprocessor. Thus, error coverage prediction can be made using a single method.

Path-Based Error Coverage Prediction can be used in the following way. Assume that we want to identify those input sequences that give extremely high or low error detection coverage among a given set of input sequences. (The set of input sequences of interest is typically determined by studying the usage of the evaluated system.) Instead of conducting a fault injection campaign for each input sequence, we conduct a single fault injection campaign for an arbitrary input sequence and then apply Path-Based Error Coverage Prediction to rank the input sequences according to error detection coverage. Once the ranking is done, we can conduct fault injection campaigns with the "interesting" input sequences in order to accurately determine the coverage figures of interest.

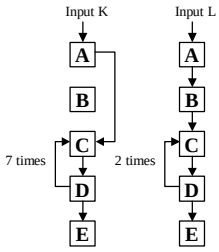
This procedure significantly reduces the time it takes to identify input sequences with extremely high or low coverage, as prediction is much faster than conducting fault injection campaigns. The prediction uses information about the usage of the program's basic blocks for the input sequence for which the prediction is made. This information can be collected during a single fault-free simulation of the program execution. The time needed for the prediction is then determined by the time it takes to run this simulation. This time is approximately equal to the time it takes to make one fault injection experiment, i.e. to observe the effect of a single fault.

We have used this technique for estimating the coverage of several hardware implemented runtime checks included in a microprocessor specially designed for use in highly dependable space applications. We injected single bit-flips inside the microprocessor to emulate the effects of Single Event Upsets.

The paper is organized as follows: In the next section, the prediction technique is described. In section 3 the experimental set-up used to evaluate the prediction technique is presented and in section 4 the prediction technique is applied and evaluated. Finally, in section 5, a discussion and conclusion is given.

2. Predicting error coverage

The Path-Based Error Coverage Prediction technique is an enhanced version of the Execution Based Prediction technique presented in [5]. It predicts the error coverage for an arbitrary input sequence based on the results from fault injection experiments with another input sequence, called the *base-sequence*. The technique is based on the fact that the execution path of the program varies for different input sequences. Depending on the input sequences, different parts of the program are executed different number of times. This is illustrated in Figure 1, where a program is divided into *basic blocks* [9] (A-E) and each basic block is executed a different number of times depending on the input sequence.



Block (i)	No. of times executed for input K ($x_{K,i}$)	No. of times executed for input L ($x_{L,i}$)	Probability of non-covered error for input L $\left(\hat{P}_{nce,i} = \frac{n_{nc,i}}{n_i}\right)$
A	1	1	0/30
B	0	1	0/50
C	8	3	3/10
D	8	3	1/12
E	1	1	0/6

Figure 1: Workload execution path differs for different input data

Let, $P_{nc,i}$, denote the probability that a fault results in a non-covered error, given that the fault is activated during execution of block *i*. The prediction is based on the assumption that $P_{nc,i}$ is constant for all input sequences. This is motivated by the fact that the activity of the system during execution of a basic block is the same regardless of the input sequence used, i.e. the same instructions are always executed in each basic block. However, this probability is likely to vary for different basic blocks in a program.

The error coverage for an input sequence *p*, c_p, can be calculated using the following equation for a workload program containing *n* basic blocks:

$$c_p = 1 - \sum_{i=1}^{n} P_{nc,i} \cdot w_{p,i} \qquad (2.1)$$

where $w_{p,i}$ is a weight factor for block *i*, corresponding to the proportion of faults activated during execution of block *i* for input sequence *p* out of the total number of faults activated for input sequence *p*. $P_{nc,i}$ is estimated from fault injection experiments conducted with the base input sequence as:

$$\hat{P}_{nc,i} = \frac{n_{nc,i}}{n_i} \qquad (2.2)$$

Where $n_{nc,i}$ is the observed number of faults activated when block *i* is executing resulting in non-covered errors, and n_i is the total number of faults activated during execution of block *i*. The weight factor $w_{p,i}$ is estimated using the following equation:

$$w_{p,i} = \frac{n_{p,i}}{n_p} \qquad (2.3)$$

Where $n_{p,i}$ is the number of faults activated during execution of block *i* for input sequence *p* and n_p is the total number of faults activated for input sequence *p*. Now, $n_{p,i}$ is unknown, but it can be estimated based on the number of activated faults for the base sequence as:

$$\hat{n}_{p,i} = \frac{n_i}{x_i} \cdot x_{p,i} \qquad (2.4)$$

Where n_i is the number of faults activated during execution of block *i* for the base input sequence, and x_i is the number of times block *i* is executed for the base input sequence. Thus, n_i divided by x_i gives the number of activations per execution in block *i* for the base input sequence. This is multiplied with $x_{p,i}$, which is the number of times block *i* is executed for input sequence *p*, thereby giving an estimation of, $n_{p,i}$, the number of faults activated during execution of block *i* for input sequence *p* [1].

n_p can be estimated as the sum of the activated faults for all the *n* basic blocks of the base input sequence:

$$\hat{n}_p = \sum_{j=1}^{n} \frac{n_j}{x_j} \cdot x_{p,j} \qquad (2.5)$$

Where n_j is the number of faults activated during execution of block *j* for the base input sequence, $x_{p,j}$ is the number of times block *j* is executed for input sequence *p*, and x_j is the number of times block *j* is executed for the base input sequence.

By combining Equations 2.1-2.5, the predicted error coverage for input sequence *p*, c_p, can be estimated from fault injection experiments conducted using a chosen base input sequence as:

$$c_p = 1 - \sum_{i=1}^{n} \left(\frac{n_{nc,i}}{x_i} \cdot \frac{x_{p,i}}{\sum_{j=1}^{n} \frac{n_j}{x_j} \cdot x_{p,j}} \right) \qquad (2.6)$$

3. Experimental set-up

In this section the experimental set-up used to evaluate the prediction technique is described. First, the target system and the fault injection environment are presented and then the error classifications and definitions used in the experiments are given.

3.1. The Thor microprocessor

The experiments have been conducted on the Thor processor [12]. The Thor microprocessor is developed by SAAB Ericsson Space AB and is intended for highly dependable space and military applications. It is a 32-bit Reduced Instruction Set Computer (RISC) with a four-stage pipe based on a stack-oriented instruction set architecture. The data cache is a direct mapped, write-back cache of 128 bytes (see Figure 2).

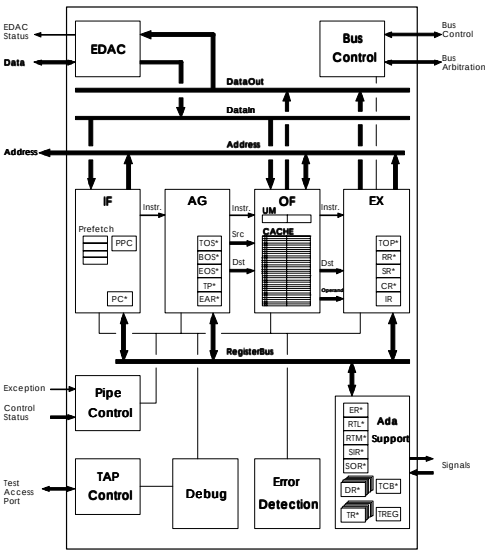


Figure 2: The Thor Microprocessor

There are several error detection mechanisms implemented in the Thor processor such as *run-time checks*, *control flow checking* and *main memory error checking*. There are 12 hardware exceptions included in the run-time checks of Thor, see Table 1. The control flow checking is used to check that the processor's order of execution is correct. In this paper, the coverage of the run-time checks and the control flow checking mechanisms

1 Note that this definition requires that all basic blocks are executed at least once for the chosen base input sequence (otherwise several input sequences are required so that all basic blocks can be accounted for).

with respect to single event upsets (bit-flips) occurring inside the microprocessor is investigated. The main memory checking mechanisms are not evaluated since no faults were injected into the main memory.

Table 1: Hardware exceptions in Thor

Exception	Description
Bus error	Bus time out of external memory
Address error	Operand effective address larger than 2 Gbyte
Data error	Chip input signal DE (Data Exception) is asserted by the EDAC
Instruction error	Illegal instruction or trying to execute a privileged instruction in user mode
Jump error	Attempt to jump, call or return to a target address outside memory address space
Constraint error	A constraint of run-time assertion instructions is not satisfied
Access error	Attempt to follow a null pointer
Storage error	Attempt to access memory outside the task's stack in user mode
Overflow check	Overflow of signed integer and float arithmetic operations
Underflow check	Underflow or denormalized result of float arithmetic operations
Division check	Division by zero
Illegal operation	Illegal operation for float and double arithmetic instruction involving 0 and ∞

3.2. Fault injection environment

Fault injection tool: The experiments were performed using MEFISTO-C, which is an elaboration of the MEFISTO tool [10]. MEFISTO-C injects faults in VHDL models by utilising simulator commands that affect signals and variables.

Fault injection with MEFISTO-C is divided into three phases: *set-up*, *simulation* and *data processing*. In the *set-up phase*, the VHDL target and workload is chosen. The experiments are defined by the location of the injected faults in the VHDL model, the activation time of the faults and the fault model to be used. A set of fault injection experiments is called a fault injection campaign. The s*imulation phase* is conducted using a VHDL simulator, which can be run on one or several Unix workstations. In this case, the Vantage Optium^{TM} simulator was used on two workstations. The *Simulation phase* starts by making a fault free run, called the reference simulation, and then each of the experiments are performed. In the *data processing phase* the trace data from the reference simulation is compared with the trace data from the fault injection experiment. The result from this comparison is analysed to extract information and evaluate the fault injection experiments.

Workload: The workload program is based on an Ada package implementing a recursive quick-sort algorithm, which sorts seven data elements of the predefined Ada type float. Fault injection is performed during the actual sorting of the seven data elements, which takes different number of cycles depending on the input sequence. After the sorting is finished, the result is written to memory.

Fault model: The fault model used is the single bit-flip fault, which affects state elements by changing a logical value from zero to one or from one to zero. This fault model is reasonably accurate for SEUs (Single Effect Upsets), caused by e.g. heavy ion radiation in the space environment or by neutrons at high altitude [11].

Fault locations: Faults are injected in 4410 internal state elements of the Thor microprocessor, i.e. flip-flops and latches. These state elements are divided in three parts; *Registers*, *Cache* and *Other registers* (811, 1824 and 1775 locations respectively). The registers belonging to the Register part are marked with an asterisk in Figure 2. The Cache is located in the Operand Fetch (OF) stage. All other state elements are grouped into the Other registers part. No faults are injected in the Debug or the TAPI (Test Access Protocol Interface) block.

3.3. Error classification and definitions

Errors are classified in two main categories, Non-effective and Effective errors. The Non-effective errors correspond to errors, which were either latent or overwritten:

- *Latent errors* indicate that the injected fault had no effect on the program execution, but the observable state of the CPU differed from the fault-free state when the program finished.
- *Overwritten errors* indicate that the injected fault was overwritten without causing any other effect on the system.

The Effective errors correspond to errors, which were either detected by the error detection mechanisms of the Thor CPU, caused a CPU crash, or lead to incorrect result:

- *Detected errors:* Errors that were detected by the Thor error detection mechanisms (See Table 1).
- *Other errors:* Errors that caused the CPU to crash. These errors are assumed to be detected by external error detection mechanisms such as I'm alive signals.
- *Escaped errors:* Errors that escaped the error detection mechanisms causing a failure i.e., the sort-algorithm produced an incorrect result.

The Error coverage is the percentage of errors that were detected by an error detection mechanism of the total number of errors. The *Error detection coverage* is the percentage of errors that were detected by an error detection mechanism of the total number of effective errors.

The *activation time* of an injected fault is defined as the time of the first propagation of a fault. If a fault is injected in the data cache, then the activation time is the time when the incorrect data is first used by the CPU. If a latent fault does not propagate, then the activation time is equal to the time when the fault was injected. If an overwritten fault does not propagate, then the activation time is equal to the time when the fault was overwritten.

4. Applying and evaluating the prediction technique

In this section, the Path-Based Error Coverage Prediction technique is applied and evaluated. The technique is applied by first selecting set of input sequences. The set of input sequences used in this study, called input sequence *a–x*, are 24 randomly chosen permutations of the same seven elements to be sorted by the quick-sort program. A fault-free simulation of the program execution was then performed for each of the chosen input sequences to determine their execution paths. In addition, a fault injection campaign was performed on the arbitrarily chosen input sequence *x*. The results from the fault injection campaign conducted on the base input sequence *x*, and the information about the execution paths of input sequences *a–w* were then used to predict the error coverage for each of the input sequences *a-w* according to equation 2.6. Figure 3 shows that the prediction technique identifies input sequences *i* and *p* to have the highest error non-coverage and input sequence *o* to have the lowest error non-coverage. The final step would be to conduct fault injection campaigns on the input sequences that were predicted to have the most extreme error coverage, to accurately estimate the results.

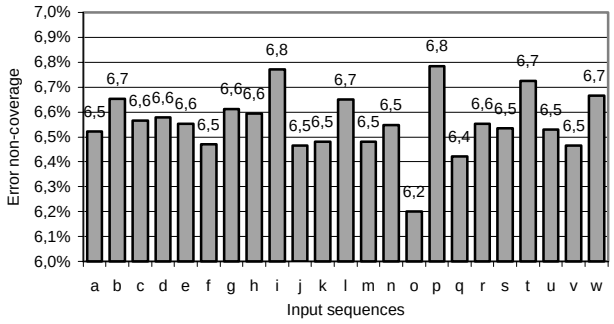


Figure 3: Predicted error non-coverage using input sequence *x* as the base sequence.

In order to evaluate the accuracy of the prediction technique two more fault injection campaigns were conducted on the arbitrarily chosen input sequences *h* and *i*. The predicted error coverage was then compared with the results from the fault injection experiments, called the observed results. To enhance the evaluation, input sequence *h* and *i* were also used as base sequences to predict the results of the other two campaigns. Thereby, six different predictions were performed.

In the rest of this section, three analyses are performed. First, the error detection coverage is predicted, then the error coverage is predicted and finally, as a comparison, the error coverage is predicted using Execution Profile Based Prediction [5].

4.1. Predicting error detection coverage

The results of predicting the error detection coverage are shown in Figure 4-6. The figures show a comparison of the predicted error detection non-coverage and the observed error detection non-coverage using input sequence *h*, *i* and *x* as base sequences respectively. The prediction technique is able to identify the input sequence with the highest error detection non-coverage using base sequence *i* or *x*. However, the prediction is incorrect when using base sequence *h*. This is because the observed error detection coverage for input sequence *i* and *x* are very similar; the difference is only ~2 % for faults injected into the whole CPU and ~0.6 % for faults injected into the cache. Whereas using base sequences *i* or *x*, the differences are more than 8 %.

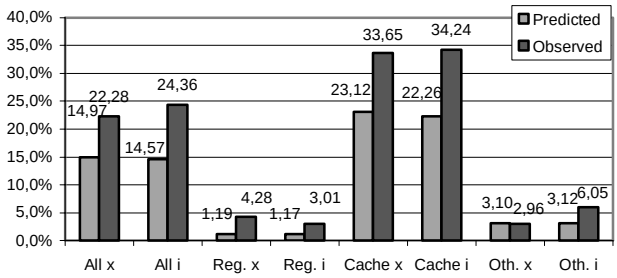


Figure 4: Observed vs predicted error detection non-coverage using input sequence *h* as the base sequence.

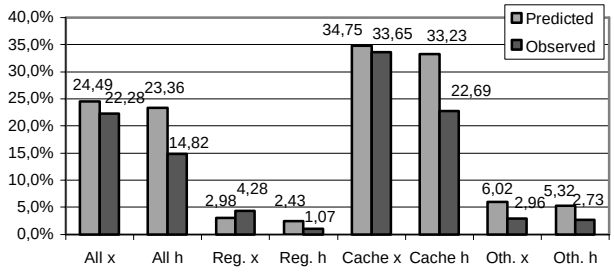


Figure 5: Observed vs predicted error detection non-coverage using input sequence *i* as the base sequence.

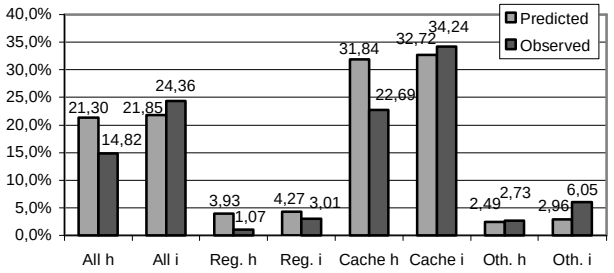


Figure 6: Observed vs. predicted error detection non-coverage using input sequence *x* as the base sequence

4.2. Predicting error coverage

In this section the error coverage is predicted, i.e. the overwritten and latent errors are also included. The results of using the Path Based Error Coverage Prediction (PB) are shown in Figure 7-9. The technique is able to identify the input sequence with the highest error detection non-coverage using base sequence *i* or *x*. However, similar to what was observed in section 4.1, the prediction is incorrect when base sequence *h* is used since the difference in error coverage for sequence *i* and *x* is too insignificant.

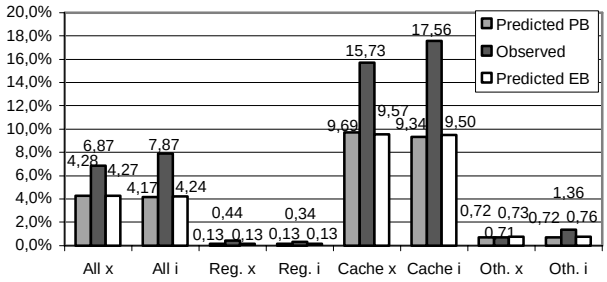


Figure 7: Observed vs. predicted error non-coverage using input sequence *h* as the base sequence.

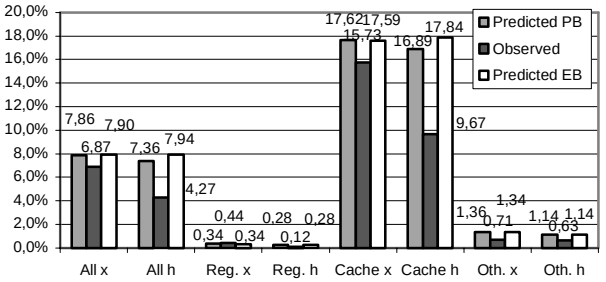


Figure 8: Observed vs. predicted error non-coverage using input sequence *i* as the base sequence.

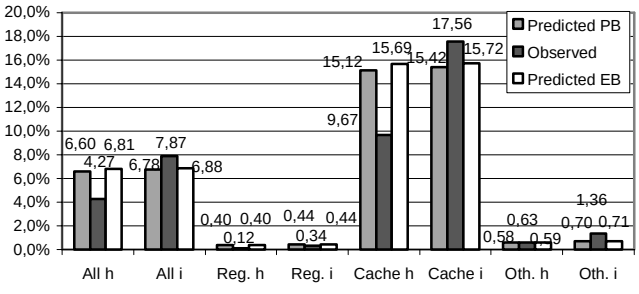


Figure 9: Observed vs. predicted error non-coverage using input sequence *x* as the base sequence.

4.3. Execution Profile Based Prediction

As a comparison, the Execution Profile Based Prediction technique (EB) is also evaluated (see Figure 7-9). The technique is able to identify the input sequence with the highest error non-coverage for the Registers and for the Other registers, but for the Cache, the technique is not able to correctly rank the input sequences. In [5], the reason for this was assumed to be that faults injected in the cache would not be activated for a considerable amount of time, i.e. the activation latencies were long. Since the Execution Profile Based Prediction technique relies on the approximation that the basic block activating a non-covered error is the same as the block executing when the fault is injected, this approximation do not hold for faults injected into the cache due to the long activation latencies. (In the Path-Based Error Coverage Prediction the actual activation is observed). However, for faults injected in the register part the approximation hold, since the activation latencies are short. These assumptions are verified in this study. The results show that the observed median activation latency for the cache is 54 cycles, while the observed median activation latency for the Registers and the Other registers is only 1 cycle (see Figure 10).

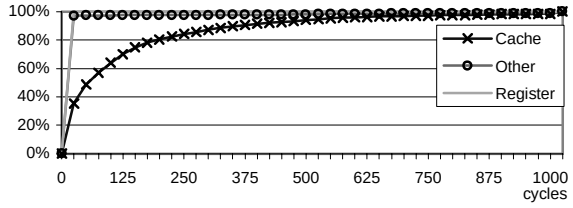


Figure 10: Activation latencies for input sequence *i*.

4.4. Results of fault injection campaigns

The results from the three performed fault injection campaigns are presented in this section. Table 3 shows the results of fault injection campaigns with each of the input sequences. The results show that the input sequence has a significant impact on the estimated error coverage. The error coverage varies from 92.13% to 95.73%.

Table 3: Results of fault injection experiments

Name	Input sequence *h*	Input sequence *i*	Input sequence *x*
Number of injected faults	13928	12831	13682
Non-effective errors	71,17 ± 0,75 %	67,69 ± 0,81%	69,16 ± 0,77%
Effective errors			
Detected error	24,15 ± 0,71%	24,09 ± 0,74%	23,64 ± 0,71 %
Other error	0,41 ± 0,11%	0,35 ± 0,10%	0,33 ± 0,10 %
Escaped error	4,27 ± 0,34%	7,87 ± 0,47%	6,87 ± 0,42 %

5. Discussion and conclusion

In this paper, a post-injection prediction technique, called Path-Based Error Coverage Prediction, suitable for simulation-based fault injection is presented. The technique reduces the effort of estimating error coverage when the input sequence to a workload program varies. The need to consider input sequence variations when estimating the error coverage using fault injection has been shown in previous studies and was also verified in this study, where the estimated error coverage varied more than 4 percentage units for different input sequences. The objective of the technique is to decrease the number of fault injection experiments needed to be performed to estimate the error coverage since it can be very time consuming to conduct fault injection campaigns for each input sequence.

The technique is applied by first performing a fault injection campaign for an arbitrary input sequence. The results of this campaign are then used to predict the results of other input sequences. The prediction is made by calculating error coverage factors for each of the basic blocks executed by the workload program based on the results obtained from the fault injection campaign. The error coverage for an input sequence is then predicted by means of a weighted sum of these coverage factors. The weight factors are obtained by analysing the execution path of the input sequence using a fault-free execution of the workload. Thus, the technique enables the input sequences to be ranked according to error coverage. Once the ranking is done, fault injection experiments can be performed for the input sequences predicted to have the lowest (or highest) error coverage, to estimate the actual error coverage values, thus giving an estimation of e.g., the worst-case error coverage.

Three fault injection campaigns were performed, each with a different input sequence to a workload program executing a quick-sort algorithm. The workload was executed on the Thor microprocessor. The error coverage ranged from 92.13% to 95.73% with corresponding 95% confidence intervals of ±0.47% and ±0.34% respectively. The results of each of these campaigns were used to predict the results of the other two campaigns, thereby; six different predictions were made. The results of the fault injection experiments were compared with the predicted results showing that the technique was able to identify the input sequence with the highest or lowest error coverage provided that the difference in actual coverage was significant.

Only three fault injection campaigns were performed in this study and only one workload was employed. Therefore, more experiments are needed to verify the technique. The technique could also be refined further by e.g., using the results of the fault injection campaigns conducted to estimate the actual coverage values, to make new predictions of the error coverage of the other input sequences in order to verify the original predictions. Also the error coverage for each basic block can probably be estimated more accurately using a mean value calculated from several fault injection experiments. The technique should also be applied on other systems to investigate whether it is hardware dependent. In addition, there is also a need to investigate how to calculate confidence intervals for the predicted values.

References

[1] Iyer, R.K., “Experimental Evaluation,” in Special Issue of *Proc. 25th Int. Symp. on Fault-Tolerant Computing* (FTCS-25), Pasadena, CA, USA, 1995.

[2] E. Czeck, and D. Siewiorek, “Observations on the Effects of Fault Manifestation as a Function of Workload”, *IEEE Transactions on Computers*, 41 (5), pp. 559-566, May 1992.

[3] Constantinescu, C. “Using multi-stage and stratified sampling for inferring fault-coverage probabilities”, IEEE Trans. on Reliability, Volume: 44 Issue 4, Dec. 1995 pp. 632 –639.

[4] Amendola, A.M.; Impagliazzo, L.; Marmo, P.; Poli, F., “Experimental evaluation of computer-based railway control systems” Digest of Papers, *Twenty-Seventh Annual International Symposium on Fault-Tolerant Computing*, 1997, pp 380 –384.

[5] P. Folkesson and J. Karlsson, “Considering Workload Input Variations in Error Coverage Estimation”, in *Proc. Third European Dependable Computing Conference (EDCC-3)*, pp. 171-188, (Prague, Czech Republic) September 1999.

[6] Guthoff, J.; Sieh, V., “Combining software-implemented and simulation-based fault injection into a single fault injection method” Digest of Papers., *Twenty-Fifth International Symposium on Fault-Tolerant Computing*, 1995, pp. 196 –206.

[7] Benso, A.; Rebaudengo, M.; Impagliazzo, L.; Marmo, P., “Fault-list collapsing for fault-injection experiments”, Proceedings Annual Reliability and Maintainability Symposium, 1998, pp. 383 –388.

[8] Tsai, T.K., Mei-Chen Hsueh, Hong Zhao, Kalbarczyk, Z., Iyer, R.K. “Stress-based and path-based fault injection”, *IEEE Trans. on Comp.*, Vol. 48 Issue 11, Nov. 1999, pp 1183 –1201.

[9] A. Aho, R. Sethi, and J. Ullman, *Compilers: Principles, Techniques and Tools*, Reading, MA: Addison Wesley, 1985.

[10] E. Jenn, J. Arlat, M. Rimén, J. Ohlsson, and J. Karlsson, “Fault Injection into VHDL Models: The MEFISTO Tool”, in *Proc. 24th Int. Symp. on Fault-Tolerant Computing (FTCS-24)*, (Austin, TX, USA) June 1994, pp. 66-75.

[11] G. C. Messenger, "Collection of Charge on Junction Nodes From Ion Tracks", *IEEE Trans. on Nuclear Science*, Vol. NS-29, No. 6, Dec. 1982, pp. 2024-2031.

[12] *Saab Ericsson Space AB, Microprocessor Thor, Product Information*, September 1993.