The Effects of Workload Input Domain On Fault Injection Results

Peter Folkesson and Johan Karlsson[1]

Extended Abstract

1 Introduction

It is well known that workload has significant impact on dependability measures obtained from fault injection experiments [3][5]. The effects on the outcome of fault injection experiments when the input data to be processed by the workload is altered, should also be considered [2]. To investigate the effects of workload input domain on fault injection results (e.g. error coverage), an experimental set-up using the FIMBUL tool [4] was employed. The problem of accurately estimating error coverage, if the effects of input domain are significant, could be solved by performing several fault injection campaigns using different input sequences. Since this would be very time consuming, a methodology to speed up the process is presented. The methodology involves predicting error coverage for different input sequences based on fault injection experiments conducted using another input sequence.

2 Experimental set-up

The target system for the fault injection experiments was the Thor microprocessor, a 32 bit stack oriented RISC processor developed and sold by Saab Ericsson Space AB [6]. The Thor processor uses several advanced error detection mechanisms for fault-tolerance and provides access to the internal logic of the CPU via internal and boundary scan chains using a Test Access Port (TAP).

1. Laboratory for Dependable Computing, Department of Computer Engineering, Chalmers University of Technology, S-412 96 Göteborg, Sweden, email: peterf@ce.chalmers.se, johan@ce.chalmers.se

The FIMBUL (Fault Injection and Monitoring using BUilt-in Logic) tool was chosen to conduct the fault injection experiments on the Thor processor. FIMBUL is able to inject faults into the internal logic of Thor via the scan chains, i.e., scan-chain implemented fault injection (SCIFI).

Three different workloads written in Ada were utilized. Two of the workloads are implementations of the Quicksort and Shellsort algorithms, both sorting an array of 7 elements. The third is an implementation of an algorithm solving the Towers of Hanoi puzzle. The complexity of each workload was about 1 KB of code executing for a few thousand clock cycles (depending on the input data used).

Twenty-five permutations of the initial sort order of the same seven elements were used as input sequences for the sort workloads and seven different tower configurations were used as the input sequences for the Towers of Hanoi workload.

3 Results using varying input data

Almost 4000 single bit-flip faults were injected for each of the different input sequences using the Quicksort workload. The injected faults were selected randomly by sampling the fault space using a uniform sampling distribution. We define the fault space as the Cartesian product $F = L \times T$, where L is the set of all fault locations (2250 internal state elements of Thor) and T the set of all time points when faults can be injected (using a time resolution of one machine instruction).

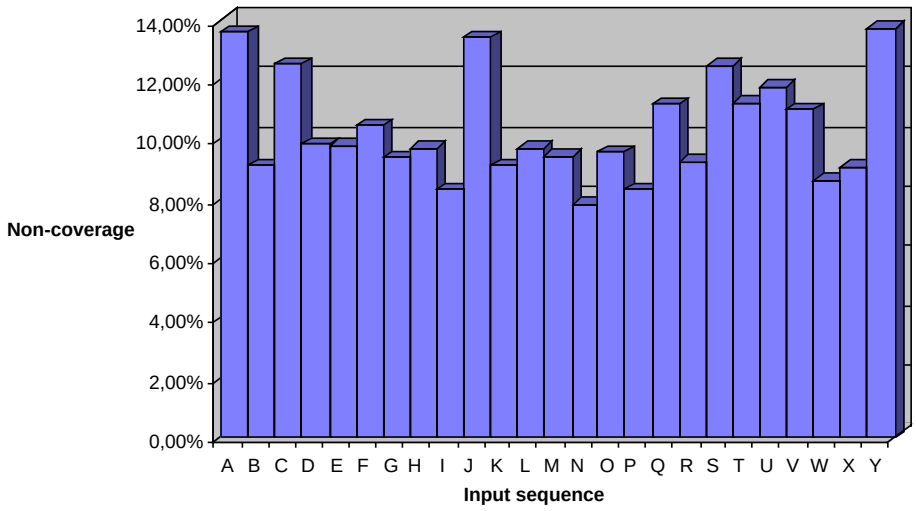

Figure 1: Error non-coverage for Quicksort using different input sequences.

The results show that error non-coverage, i.e. the percentage of the injected faults which cause wrong results to be produced by the workload without being detected, can be estimated with a difference of almost 6 percentage units (from 7.80% to 13.67%) if the input data for the workload is altered, see *Figure 1*. Thus, workload input data can have a major impact on the results from fault injection campaigns and must be considered.

Two possible reasons for the observed differences were investigated:

(i) Varying the input data will cause different parts of the workload, i.e. *basic blocks* [1], to be executed different number of times, i.e. the *execution profile* will differ. If most non-covered errors are produced when fault injecting the system during execution of certain basic blocks, the results from fault injection campaigns using different input sequences may vary.

(ii) The contents of the fault injected target locations will differ when the system processes different data as a result of altered data usage, i.e. the *data usage profile* will differ. The fault injection results may vary when altering the input data if most non-covered errors are produced when fault injecting certain *critical* data and some input sequences lead to the usage of a higher amount of critical data than others.

4 Error coverage prediction methodology

A methodology for estimating error coverage considering workload input domain variations can be formulated based on the observations above. The results for a particular input sequence are predicted based on experiments conducted using an arbitrary chosen *base-*, or *reference-*, input sequence, by comparing the execution and data usage profiles of each input sequence.

In execution profile based prediction, the predicted non-coverage $\bar{c}_{ep}$ for a particular input sequence *p* is calculated using the following equation:

$$\bar{c}_{ep} = \sum_{i=1}^{n} \hat{P}_{nce,\,i} \cdot w_{i,ep} \qquad (1)$$

where the basic blocks of the workload are numbered 1 to *n*, $\hat{P}_{nce,\,i}$ is the probability that an in-

jected fault is activated and results in a non-covered error, given that basic block *i* executes, estimated using the base input sequence, and $w_{i,ep}$ is the weight factor for block *i*, i.e. the proportion of the whole execution time spent executing block *i*, for input sequence *p*. Assume that block *i* executes for k_i clock cycles and that the workload executes for l_p clock cycles for input sequence *p*. $w_{i,ep}$ is then calculated as $w_{i,ep} = \frac{k_i}{l_p} \cdot x_{i,p}$ where $x_{i,p}$ is the number of executions of block *i* for input sequence *p*.

In data usage based prediction, the data usage, i.e. the percentage of the total fault space that contains critical data, is measured for the various types of critical data. Let $\hat{P}_{ncd,i}$ denote the probability that a fault injected into critical data item *i* will lead to a non-covered error estimated using the base input sequence, and $w_{i,dp}$ the usage of critical data item *i* for input sequence *p*. The predicted non-coverage $\bar{c}_{dp}$ for input sequence p is then calculated using the following equation:

$$\bar{c}_{dp} = \sum_{i=1}^{n} \hat{P}_{ncd,i} \cdot w_{i,dp} \tag{2}$$

where the critical data items are numbered 1 to *n*.

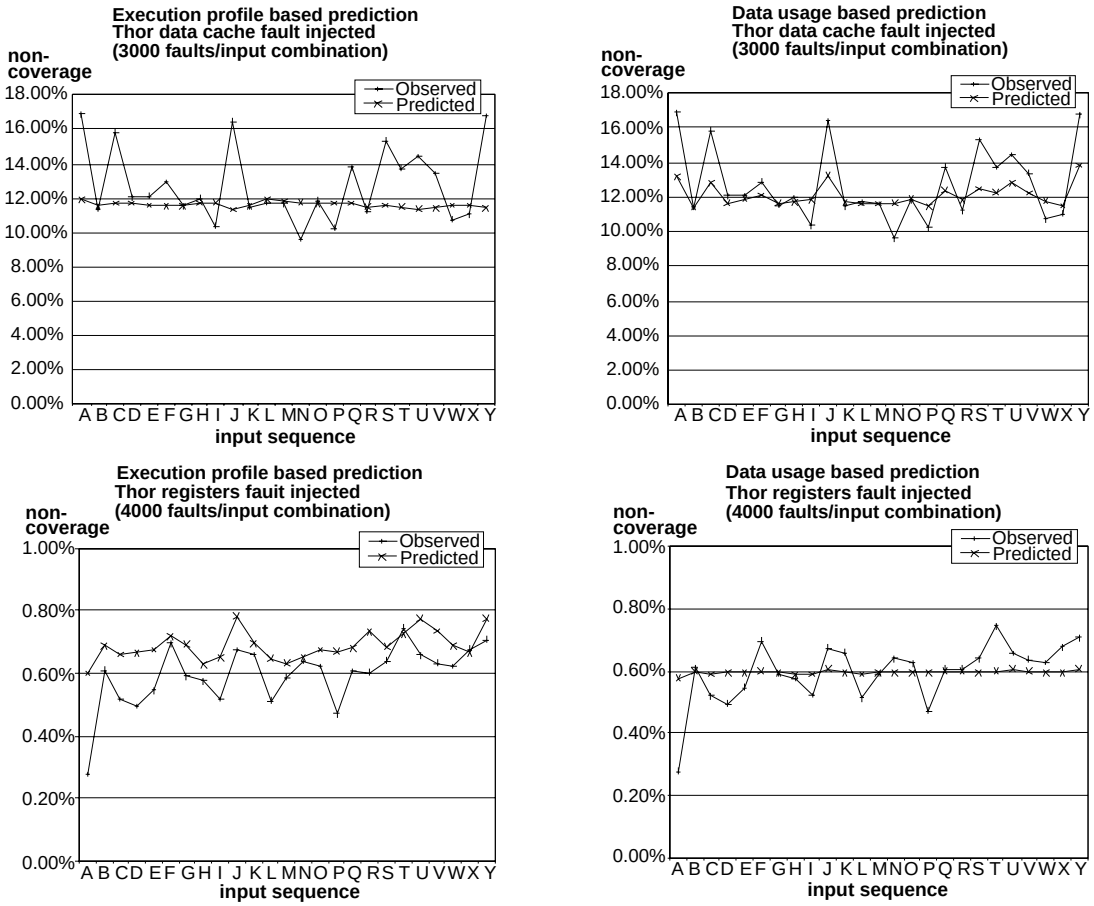

Figure 2: Predicted vs. observed non-coverage for Quicksort.

Results from fault injection campaigns conducted on different parts of the Thor CPU using the Quicksort workload show that error coverage for the data cache is predicted more favorably using data usage based prediction, while error coverage for the other parts of the CPU, i.e. the Thor registers, is predicted more favorably using execution profile based prediction, see *Figure 2* (input sequence *M* was used as the base combination for all predictions). These observations are also supported by results from fault injection campaigns using the Shellsort and Towers of Hanoi workloads.

5 Future work

More research is needed to refine the methodology. Data usage based prediction on the data cache fails to predict the actual error coverage but is nevertheless useful for pointing out the input sequences with the lowest coverage which could be fault injected to estimate the worst-case coverage. In execution profile based prediction, the block activating a fault is assumed to be the same as the one executing when the fault is injected. This approximation would not be needed using, e.g., simulation based fault injection where the higher observability would enable direct identification of the blocks activating the non-covered errors. How the input combinations used for fault injection and prediction should be selected if the input space is large, also remains to be investigated. More experiments are also needed, using different fault injection techniques on more systems and more workloads, in order to verify the generality and usability of the methodology.

References

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

[2] C. Constantinescu, “Using multi-stage and stratified sampling for inferring fault-coverage probabilities”, *IEEE Transactions on Reliability*, 44 (4), pp. 632-639, 1995.

[3] 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.

[4] P. Folkesson, S. Svensson, and J. Karlsson, “A Comparison of Simulation Based and Scan Chain Implemented Fault Injection”, in *Proc. 28th Int. Symp. on Fault-Tolerant Computing (FTCS-28)*, pp. 284-293, (Munich, Germany) June 1998.

[5] U. Gunneflo, J. Karlsson, and J. Torin, “Evaluation of Error Detection Schemes Using Fault Injection by Heavy-ion Radiation”, in *Proc. 19th Int. Symp. Fault-Tolerant Computing (FTCS-19)*, pp. 340-347, 1989.

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