

INJECTION OF FAULTS IN COMPLEX COMPUTERS

Henrique Madeira, João Carreira, and João Gabriel Silva
Dep. Engenharia Informatica
Pinhal de Marrocos - Polo II - Univ. Coimbra
3030 Coimbra - PORTUGAL
Email: henrique@mercurio.uc.pt

The Problem

How to inject faults in the very complex processors available today and how to monitor the effects of the faults?

The growing complexity of both the hardware and software of contemporary computers makes the evaluation of the dependability properties of the systems more and more difficult. This is clearly true for methods based on analytical modelling, but it is also true for experimental methods based on fault injection.

Traditional physical fault injection approaches such as pin-level fault injection [Arlat 90, Madeira 94b], heavy-ion radiation [Karlsson 94], and power supply disturbances [Miremadi 92] are very difficult to apply in contemporary systems. In fact, the high complexity and the very high speed of the processors available today make the design of the special hardware required by the above techniques very difficult, or even impossible. The main problem is not in the injection of the faults itself but it is related to the difficulties of controlling and observing the fault effects inside the processor. Furthermore, even the detection of the activated faults is very complex. For example, the injection of faults in processor pins requires the use of complex monitoring hardware to know whether the injected faults have produced internal processor errors or not [Madeira 94a]. Similarly, techniques such as heavy-ion radiation and power supply disturbances require the target chip outputs to be compared pin-by-pin and cycle-by-cycle with a gold unit in order to know whether the injected faults has produced errors inside the target chip or not. Simulation based fault injection (e.g., [Choi 92, Jenn 94]) also has serious problems in being used in very complex processors, as the development effort (of the fault injection tools) and the time consumed by the experiments dramatically increase with the complexity of the processors.

Software fault injection techniques, also known as fault emulation (e.g., [Segall 88, Chillarege 89, Han 93, Young 93, Kanawati 92, Kao 94, Echtele 92]) are being increasingly used as an alternative to the other methods. It basically consists of interrupting the application execution in some way (usually by inserting a software trap or executing the application in trace mode) and executing specific fault injection software code that emulates hardware faults by inserting errors in different parts of the system such as the processor registers, the memory, or the application code. The main advantages of software fault injection are its low complexity, its low development effort, its and low cost (no specific hardware is needed). However, in spite of the large number of software fault injection works and the considerable advances proposed in

the last years, existing software fault injection tools still have several problems and limitations:

1) The software fault injector tools have considerable impact on the target system behaviour, either because part or the whole of the code of the tool has to be executed in the target system (i.e., is part of the target workload) or because the target program may have to be executed in a trace mode;

2) Existing software fault injection tools have a restricted range of fault triggers. Faults are injected either by corrupting the memory image of the application, by inserting traps, or by replacing one set of instructions by another set of instructions. All these methods are related to the instructions execution or to a temporal trigger and no fault triggers related with data manipulation can be defined;

3) The target system monitoring is very limited in existing software fault injection tools. Monitoring is required either for detecting the activation of some faults or to collect relevant information on the fault impact. Only few proposals handle target monitoring, either by using extra instrumentation [Young 92], by using software monitors [Kao 93], or by inserting trap instructions in the adequate locations [Kao 94]. With the last method it is not possible to achieve detailed monitoring, while the other methods require extra hardware or cause great execution overhead.

One Solution: Xception

The use of software fault injection in conjunction with the debugging and performance monitoring hardware already existing inside modern processors

Our proposal consists of the use of the advanced debugging and performance monitoring features existing in modern processors to inject more realistic faults by software, and to monitor the activation of the faults and their impact on the target system behaviour in detail. A new software fault injection and monitoring environment based on this idea has been built at the University of Coimbra. This new tool, called Xception, is already being used in a parallel system based on the PowerPC processor for the evaluation of error detection techniques, diagnosis algorithms and checkpointing techniques.

The huge complexity and integration scale of contemporary processors led the manufacturers to put sophisticated hardware for performance monitoring and debugging inside processors. Some examples of processors having (at different degrees) these kind of features are the HP Precision architecture, the PowerPC family, the Pentium, the Alpha AXP architecture, the Power2 architecture, and the MIPS R4400 - R10000 processor family. These new performance monitoring and debugging features consist mainly of performance counters and breakpoint registers. The former count user defined events such as load, store, or floating point instructions, while the latter enable the programmer to specify breakpoints for a wide range of situations such as load, store or fetch from a specified address or even some instruction types (e.g., floating point instructions). These features are not commonly used either by normal applications or by the operating system (they are mostly used by debugging and

performance analyser tools), which facilitates their usage to build new and advanced fault injector tools such as the Xception.

Some New Xception Features

By directly programming the debugging hardware inside the target processor, Xception can inject faults with minimum interference with the target application. Unlike previous software fault injectors, with Xception the target application is not modified, no software traps are inserted, and it is not necessary to execute the target application in special trace mode (the application is executed at full speed). The sophisticated exception triggers available in most of the modern processors allow the definition of many fault triggers (events that cause the injection of the fault), including fault triggers related with the manipulation of data. Furthermore, it is possible to monitor the activation of latent errors, such as the errors introduced in a specific memory cell, by programming the debugging hardware to cause an exception when the corrupted memory cell is addressed. All the exception code has to do in this case is to report (to Xception) that the corrupted memory cell has been used by the target program. On the other hand, by assessing the performance monitoring hardware inside the processor the Xception can record detailed information on the target processor behaviour after the fault. Some examples are the number of clock cycles, the number of memory read and write cycles, and instructions executed (including specific information on instructions such as branches and floating point instructions) from the injection of the fault until some other subsequent event, for instance the detection of an error (latency). Furthermore, by combining the exception triggers provided by the debugging hardware and the performance monitoring features of the processor, Xception can monitor other aspects of the target behaviour after the fault. For example, it is possible to detect if some memory area has been accessed after the fault or if some program function has been executed. Another important aspect is the fact that, since Xception operates at the exception handler level, and not through any service provided by the operating system, the injected faults can affect any process running on the target system including the operating system. Furthermore, it is also possible to inject faults in application in which the source code is not available.

The target system is regarded by Xception as a system formed by the processor, memory and data/address buses. The target system can be a single computer or a node in a distributed/parallel system. In order to facilitate the task of defining sets of faults, the user can specify faults in the following internal target units: Data Bus, Address Bus, Floating Point Unit, Integer Unit, Memory Management Unit, General Purpose Registers, Condition Code Register, Instruction Decoding, and Main Memory.

Presently, Xception has been implemented on a Parsytec parallel machine build around the PowerPC 601 processor and running PARIX operating system (a Unix alike operating system for parallel machines). The Xception upgrade for the processor PowerPC 604 is being planned and a new version for Pentium based machines is being built.

References

- [Arlat 90] J.Arlat et al., "Fault injection for dependability validation: a methodology and some applications", IEEE Trans. on Software Eng., Vol 16, No 2, Feb. 1990, pp. 166-182.

- [Chilarege 89] R. Chilarege and N. Bowen, "Understanding Large Systems Failures - A fault Injection Experiment", Proc. 19th Int. Symp. Fault-Tolerant Computing, Chicago, June, 1989, pp. 356-363.
- [Choi 92] G. Choi and Ravi Iyer, "Focus: an experimental environment for fault sensitivity analysis", IEEE Transactions on Computers, Vol. 41, No. 12, December 1992.
- [Echtle 92] K. Echtle, M. Leu. "The EFA Fault Injector for Fault-Tolerant Distributed System Testing", in Workshop on Fault-Tolerant Parallel and Dist. Systems, pp 28-35, 1992.
- [Han 93] S. Han, H. Rosenberg, K. Shin, "DOCTOR: an Integrated Software Fault Injection Environment", Technical Report-University of Michigan, 1993.
- [Jenn 94] E. Jenn, J. Arlat, M. Rimén, J. Ohlsson, and J. Karlsson, "Fault Injection into VHDL Models: The MEFISTO tool", Proc. of 24th Int. Symposium on Fault-Tolerant Computing (FCTS-24), pp. 336-344, Austin, TX, USA, 1994.
- [Kanawati 92] G. Kanawati, N. Kanawati, and J. Abraham, "FERRARI: A Tool for the Validation of System Dependability Properties", FTCS-22, Digest of papers, IEEE 1992, pp. 336-344.
- [Kao 93] Wei-lun Kao, R. K. Iyer, D. Tang, "FINE: A Fault Injection and Monitoring Environment for Tracing the UNIX System behaviour under Faults", IEEE Transactions on Software Engineering, Vol 19. No. 11, November 1993.
- [Kao 94] Wei-lun Kao, R. K. Iyer, "DEFINE: A Distributed Fault Injection and Monitoring Environment", Workshop on Fault-Tolerant Parallel and Distributed Systems, June, 1994.
- [Karlsson 94] J. Karlsson, P. Lidén, P. Dahlgren, R. Johansson, and U. Gunneflo, "Using Heavy-ion Radiation to Validate Fault-Handling Mechanisms", in IEEE Micro, Vol. 14, No. 1, pp. 8-32, 1994.
- [Madeira 94a] H. Madeira, M. Relá, F. Moreira, J. Silva. "RIFLE: A General Purpose Pin-Level Fault Injector", Proc. First European Dependable Computing Conference, pp 199-216, Berlin, Germany, October 1994.
- [Madeira 94b] H. Madeira and J. G. Silva, "Experimental Evaluation of the Fail-silent behaviour in Computers without Error Masking", FTCS-24 June 1994.
- [Miremadi 92] G. Miremadi, J. Karlsson, U. Gunneflo, and J. Torin, "Two Software Techniques for On-line Error Detection", proc. of 22th Fault-Tolerant Computing Symposium, FTCS-22, 1992, p. 328-335.
- [Segall 88] Z. Segall, T. Lin, "FIAT: Fault Injection Based Automated Testing Environment". In Proc.. 18th Int. Symp. Fault - Tolerant Computing., June 1988, pp 102-107.
- [Young 92] L. T. Young, R. K. Iyer, K. K. Goswami, and C. Alonso, "A Hybrid Monitor Assisted Fault Injection Environment" 3rd IFIP Working Conference on Dependable Computing for Critical Applications, Sicily, Italy, September 1992.