Experimental Framework for Injecting Logic Errors in a Virtual Machine to Profile Applications for Soft Error Resilience

Nathan DeBardeleben[1], Sean Blanchard[1], Qiang Guan[1,2], Ziming Zhang[1,2], and Song Fu[2]

[1] Los Alamos National Laboratory, Ultrascale Systems Research Center
High Performance Computing Division, Los Alamos NM 87544, USA
{ndebard,seanb}@lanl.gov
[2] University of North Texas, Dependable Computing Systems Lab
Department of Computer Science and Engineering, Denton TX 76203, USA
{QiangGuan,ZimingZhang}@my.unt.edu, Song.Fu@unt.edu

Abstract. As the high performance computing (HPC) community continues to push for ever larger machines, reliability remains a serious obstacle. Further, as feature size and voltages decrease, the rate of transient soft errors is on the rise. HPC programmers of today have to deal with these faults to a small degree and it is expected this will only be a larger problem as systems continue to scale.

In this paper we present SEFI, the Soft Error Fault Injection framework, a tool for profiling software for its susceptibility to soft errors. In particular, we focus in this paper on logic soft error injection. Using the open source virtual machine and processor emulator (QEMU), we demonstrate modifying emulated machine instructions to introduce soft errors. We conduct experiments by modifying the virtual machine itself in a way that does not require intimate knowledge of the tested application. With this technique, we show that we are able to inject simulated soft errors in the logic operations of a target application without affecting other applications or the operating system sharing the VM. We present some initial results and discuss where we think this work will be useful in next generation hardware/software co-design.

Keywords: soft errors, resilience, fault tolerance, reliability, fault injection, virtual machines, high performance computing, supercomputing.

1 Introduction

Reliability is recognized as one of the core challenge areas for extreme-scale supercomputers by a number of studies including the Defense Advanced Research Projects Agency (DARPA)[8] and the International Exascale Software Project[7]. Additionally, the US Department of Energy's Office of Advanced Scientific Computing Research (ASCR) has held several workshops that produced reports on this subject. Furthermore, several studies specifically on reliability

M. Alexander et al. (Eds.): Euro-Par 2011 Workshops, Part II, LNCS 7156, pp. 282–291, 2012.

have found that major undertakings would be required to create resilient next-generation systems[6,5]. High performance computing (HPC) systems of today already struggle with reliability and these concerns are expected to only amplify as systems are pushed to even larger scales.

The high performance computing (HPC) field of resilience aims to find ways to run applications on often unreliable hardware with emphasis on making timely progress toward a correct solution. The goal of resilience is to move beyond merely tolerating faults but coexisting with failure to a point where failure is recognized as the norm and not the exception.

One of the more daunting areas of resilience research is soft errors - those errors which are generally transient in nature and difficult or impossible to reproduce. Often these errors cause incorrect data values to be present in the system. While soft errors are generally rare, there is evidence to believe that the rate is increasing as feature sizes and voltages decrease[10]. Not only will these increasingly common errors negatively impact performance while hardware corrects some of them, we believe these errors will occur not only in the more familiar memory but in logic circuits where traditional techniques will neither detect or be able to correct the error. This leads us to believe that next generation systems will either have to be *hardened* to get around these errors or application programmers will have to learn to design for systems that give incorrect answers with some noticeable probability.

In this work we present SEFI, the Soft Error Fault Injection framework, a tool aimed at quantifying just how resilient an application is to soft errors. While our goal is to look at both corrupted data in memory and corrupted logic circuits, we start our research by examining the latter. We choose to focus on logic errors as faults in memory have been studied in the past and, to a large extent, hardware to detect and correct such errors exists. Our software tools inject soft errors in the logic operations at known locations in an application which allows us to observe how the application responds to faulty behavior of the simulated hardware.

The rest of this paper is organized as follows: Section 2 presents an overview of the logic soft error injection framework and then Section 3 outlines an initial experiment and discusses the results. In Section 4 we discuss the importance of this work and its intended uses. Section 5 compares our approach with other work in the field. Finally, Section 6 discusses the future work and we conclude with our findings in Section 7.

2 Overview of Methodology

SEFI's logic soft error injection operational flow is roughly depicted in Figure 1. First, the guest environment is booted and the application to inject faults into is started. Next, we probe the guest operating system for information related to the code region of the target application and notify the VM which code regions to watch. Then the application is released, allowing it to run. The VM observes the instructions occurring on the machine and augments ones of interest. A more detailed explaination of these techniques follows.

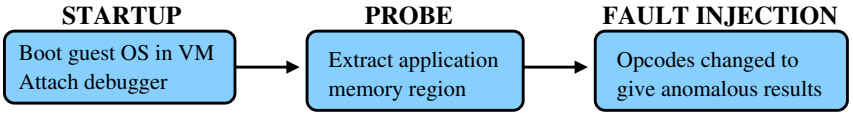


Fig. 1. Overview of SEFI

2.1 Startup

Initial startup of SEFI begins by simply booting a debug enabled Linux kernel within a standard QEMU virtual machine. QEMU allows us to start a gdbserver within the QEMU monitor such that we can attach to the running Linux kernel with an external gdb instance. This allows us to set breakpoints and extract kernel data structures from outside the guest operating system as well as from outside QEMU itself. This is a fairly standard technique used by many Linux kernel developers. Figure 2 depicts the startup phase.

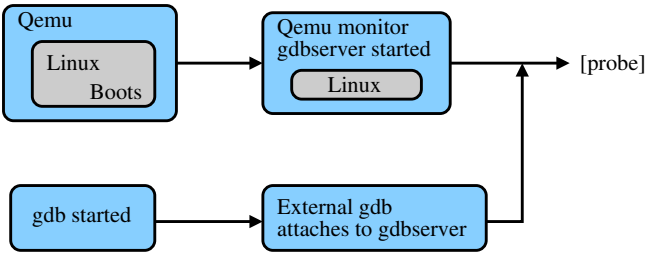


Fig. 2. SEFI's Startup Phase

2.2 Probe

Once the guest Linux operating system is fully booted and sitting idle we use the attached external gdb to set a breakpoint at the end of the `sys_exec` call tree but before an application is sent to a cpu to be executed. We are currently focused on only ELF binaries and have therefore set our breakpoint at the end of the `load_elf_binary` routine. This is trivial to generalize to other binary formats in future work. With the breakpoint set we are free to issue a `continue` via gdb to allow the Linux kernel to operate. The application of interest can now be started and will almost immediately hit our set breakpoint and bring the kernel back to a stopped state. By this point in the `exec` procedure the kernel has already loaded an application's text section into physical memory in a memory region denoted by the `start_code` and `end_code` elements of the task's `mm_struct` memory structure. We can now extract the location in memory assigned to our application by the kernel by walking the task list in the kernel. Starting with the symbol `init_task`, we can find the application of interest either by comparing a binary name to the `task_struct`'s `comm` field or by searching for a known pid which is also contained in the `task_struct`. The physical addresses within the VM of the application's text region can now be fed into our fault

injection code in the modified QEMU virtual machine. Currently this is done by hand but we have plans to automate this discovery and transfer using scripts and hypervisor calls.

Figure 3 depicts the probe phase of SEFI.

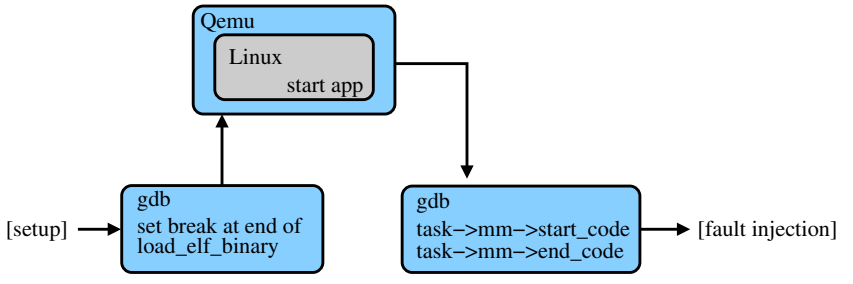


Fig. 3. SEFI's Probe Phase

2.3 Fault Injection

In figure 4 we see that once QEMU has the code segment range of the target application, the application is resumed. Next, when any opcode is called in the guest hardware that we are interested in injecting faults into, QEMU checks the current instruction pointer register (EIP). If that instruction pointer address is within the range of the target application (obtained during the probe phase), QEMU now is aware that the application we are targeting is running this particular instruction. At this point we are able to inject any number of faults and have confidence that we are affecting only the desired application.

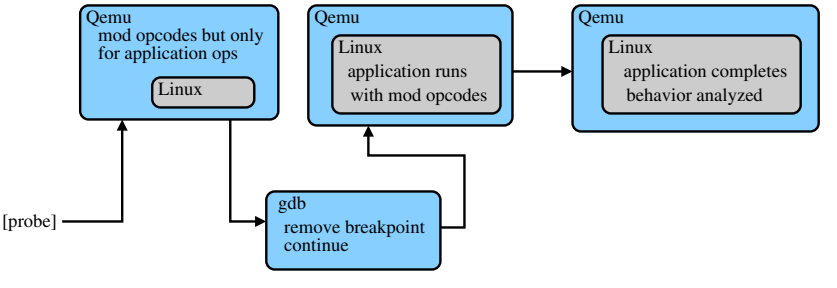


Fig. 4. SEFI's Fault Injection Phase

The opcode fault injection code has several capabilities. Firstly, it can simply flip a bit in the inputs of the operation. Flipping a bit in the input simulates a soft error in the input registers used for this operation. Secondly, it can flip a bit in the output of the operation. This simulates either a soft error in the actual operation of the logic unit (such as a faulty multiplier) or soft error in

the register after the data value is stored. Currently the bit flipping is random but can be seeded to produce errors in a specified bit-range. Thirdly, opcode fault injection can perform complicated changes to the output of operations by flipping multiple bits in a pattern consistent with an error in part but not all of an opcodes physical circuitry. For example, consider the difference in the output of adding two floating point numbers of differing exponents if the a transient error occurs for one of the numbers while setting up the significant digits so that they can be added. By carefully considering the elements of such an operation we can alter the output of such an operation to reflect all the different possible incorrect outputs that might occur.

The fault injector also has the ability to let some calls to the opcode go unmodified. It is possible to cause the faults to occur after a certain number of calls or with some probability. In this way the fault can occur every time which closely emulates permanently damaged hardware or can be used to emulate transient soft errors by causing a single call to be faulty.

3 Experiments

To demonstrate SEFI's capability to inject errors in specific instructions we provide two simple experiments. For each experiment we modified the translation instructions inside of QEMU for each instruction of interest. Once the instruction was called, the modified QEMU would check the current instruction pointer (EIP) to see if the address was within the range of the target application. If so, then a fault could be injected. We performed two experiments in this way, injecting faults into the floating point multiply and floating point add operations.

3.1 Floating Point Multiply Fault Injection

For this experiment we instrumented the floating point multiply operation, "`mulsd`", in QEMU. We created a toy application which iteratively performs Equation 1 40 times. The variable `y` is initialized to 1.0.

$$y = y * 0.9 \tag{1}$$

Then, at iteration 10 we injected a single fault into the multiplication operation by flipping a random bit in the output. Figure 5 plots the results of this experiment. The large, solid line, represents the output as it is without any faults. The other five lines represent separate executions of the application with different random faults injected. Each fault introduces a numerical error in the results which continues through the lifetime of the program.

In Figure 6 we focus on two areas of interest from the plot in Figure 5. In Figure 6(a) the plot is zoomed in to focus on the point where the five faults are injected so as to make it easier to see. Figure 6(b) is focused on the final results of the application. In this figure it becomes clear that each fault caused an error to manifest in the application through to the final results.

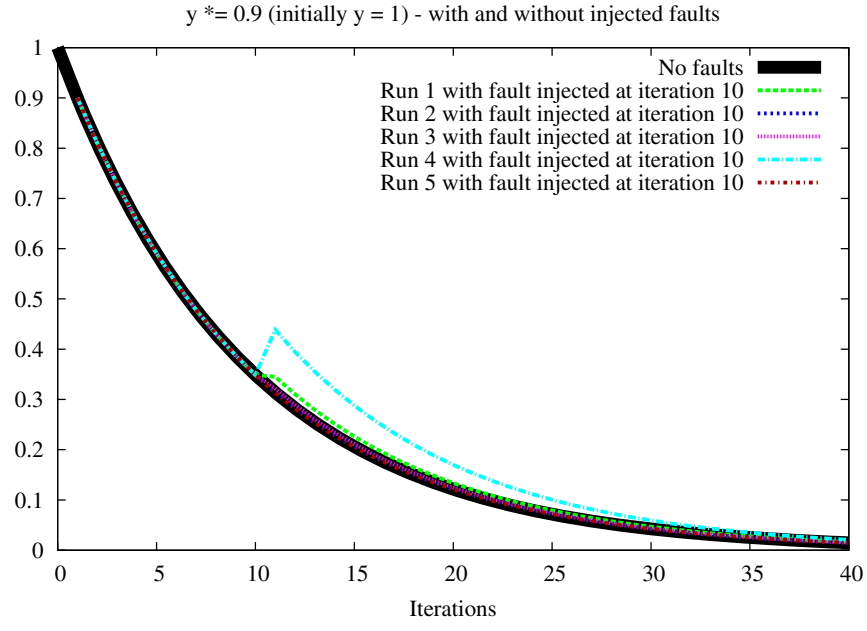
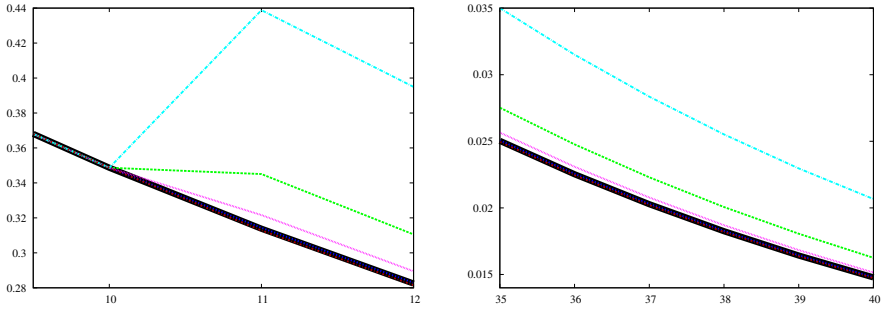


Fig. 5. The multiplication experiment uses the floating point multiply instruction where a variable initially is set to 1.0 and is repeatedly multiplied by 0.9. For five different experiments a random bit was flipped in the output of the multiply at iteration 10, simulating a soft error in the logic unit or output register.



(a) Multiply Experiment - area of interest: injected faults

(b) Multiply Experiment - area of interest: final results

Fig. 6. Experiment #1 with the focus on the injection point (a) and the effects on the final solution (b). In (a) it can be seen that each of the five separately injected faults all cause the value of y to change - once radically, the other times slightly. In (b) it can be seen that the final output of the algorithm differs due to these injected faults.

Table 1. Results of Addition Tests

A	B	C	D	E	F	G	H
30.0	30.0	30.0	30.0	30.0	30.0	30.0	30.0
31.0	31.125	32.0	481.0	23.0	8.5	128849018881.0	1966081.0
32.0	32.125	33.0	482.0	24.0	9.5	128849018882.0	1966082.0

3.2 Floating Point Addition Fault Injection

To demonstrate SEFI's capability to inject faults into different instructions, we provide another simple experiment which uses the floating point add operation, "`addsd`". This experiment simply added the value 1.0 repeatedly, as in Equation 2. At iteration 31 we had SEFI inject an error into the resulting `addsd` instruction. As can be seen from Table 1, the error is varied and sometimes appears in the exponent and other times in the mantissa of the binary representation. In the table we focus only on the iterations of importance for brevity. Column `A` represents the correct answer while the remaining columns all contain an error on the second row (31^{st} iteration).

$$y = y + 1.0 \tag{2}$$

These experiments were crafted to demonstrate the capability of SEFI to inject errors into specific instructions and clearly do not represent interesting applications. The next steps will be to inject faults into benchmark applications (such as BLAS and LAPACK) to study the soft error vulnerability of those applications.

4 Intended Uses

It is our intention to use SEFI to study the susceptibility of applications to soft errors (logic initially, and later followed by memory). We expect to be able to produce reports on the vulnerability of applications at a fine grain level - at least at the functional level and perhaps at the instruction level. We have demonstrated that we can inject logic faults at specific assembly instructions but translating those instructions back to original higher level language instructions will likely prove complex.

Hardware designers expend a great deal of resources to protect soft errors from propagating into the software stack. While current wisdom is that these protections are necessary, there are a variety of applications that could survive with a great deal less protection and would willingly trade resilience for increases in performance or decreases in power or cost. We believe SEFI begins to present a way to experiment with and quantify the level of resilience of an application to soft errors and might be useful in co-design of future systems.

5 Related Work

The work presented in this paper builds on years of open source research on QEMU[1], a processor emulator and virtual machine. Bronevetsky, et. al[3,4,2]

is probably the closest related work to SEFI in the high performance computing field. In [2] they create a fault injection tool for MPI that simulates MPI faults that are often seen on HPC systems, such as stalls and dropped messages. In [3,4] they performed random bit flips of application memories and observed how the application responded.

It is important to understand the difference between our approach and that presented in the memory bit flipping work of Bronevetsky. Bronevetsky's approach most likely closely simulates a bit flip caused by a transient soft error in that the bit flip happens randomly in memory. While they target these bit flips at a target application, there appears to be no correlation to whether the memory region will be used by the application. As stated, this closely approximates a real transient soft error. Our work, on the other hand, directly targets specific instructions and forces corruption to appear at those lines. This approach is directly targeted more at hardening a code from soft errors. It is our intention to add functionality similar to Bronevetsky's approach as a plug-in to SEFI in future work.

Naughton, et. al, in [9] developed a fault injection framework that either uses `ptrace` or the Linux kernel's built-in fault injection framework. The kernel approach allows injection of three different types of errors: slab errors, page allocation errors, and disk I/O errors. While both approaches in this work are similar to SEFI, our technique allows us to probe a wider range of possible faults.

TEMU[11] is a tool built upon QEMU like SEFI. The TEMU BitBlaze infrastructure is used to analyze applications for "taint" in a security context. This tool does binary analysis using the *tracecap* software. We have not yet had the time to determine of this suite of tools is usable for our interests but it does appear promising that we can build upon TEMU.

NFTAPE[12] is a tool which is similar to SEFI in that it provides a fault injection framework for conducting experiments on a variety of types of faults. NFTAPE is a commercial tool, however, and therefore we have not had the luxury of experimenting with it to this point.

6 Future Work

In order to validate our simulation of soft errors in logic we plan to test the same applications we use in the VM on actual hardware subjected to high neutron fluxes. Neutrons are well known to be the component of cosmic ray showers that causes the greatest damage to computer circuits[13]. Neutrons are known to cause both transient errors due to charge deposition and hard failures due to permanent damage. We will use the neutron beam at the Los Alamos Neutron Science Center (LANSCE) to approximate the cosmic ray induced events in a logic circuit over the lifetime of a piece of computational hardware. Previous work using the LANSCE beam has shown its usefulness in inducing silent data corruption (SDC) in applications of interest.

Future versions of SEFI will include plugins to simulate more sophisticated types of faults. Logic errors are unlikely to consist of simple random bit flips.

We believe the combination of SEFI testing and neutron beam validation will allow us to build realistic models of specific types of logic failures. We also plan on extending SEFI to model multi-bit memory errors which are undetectable by current memory correction techniques.

7 Conclusion

In this paper we have demonstrated the capability to inject simulated soft errors into a virtual machine's instruction emulation facilities. More importantly, we have demonstrated how to target these errors so as to be able to reasonably conduct experiments on the soft error vulnerability of a target application. This type of experimentation is usually complicated because faults that are introduced cause errors in other portions of the system, especially the operating system, and often results in outright crashes. This makes getting meaningful data about the injected faults difficult. The approach presented in this paper gets around these limitations and provides quite a bit of control.

Acknowledgements. Ultrascale Systems Research Center (USRC) is a collaboration between Los Alamos National Laboratory and the New Mexico Consortium(NMC). NMC provides the enviroment to foster collaborative research between LANL, universities and industry allowing long-term interactions in Los Alamos for professors, students and industry visitors.

This work was supported in part by the U.S. Department of Energys National Nuclear Security Administration under contract DE-AC52-06NA25396 with Los Alamos National Security, LLC.

References

1. Bellard, F.: Qemu, a fast and portable dynamic translator. In: Proceedings of the Annual Conference on USENIX Annual Technical Conference, ATEC 2005, p. 41. USENIX Association, Berkeley (2005)
2. Bronevetsky, G., Laguna, I., Bagchi, S., de Supinski, B., Schulz, M., Anh, D.: Statistical fault detection for parallel applications with automaded. In: IEEE Workshop on Silicon Errors in Logic - System Effects, SELSE (March 2010)
3. Bronevetsky, G., de Supinski, B.: Soft error vulnerability of iterative linear algebra methods. In: Workshop on Silicon Errors in Logic - System Effects, SELSE (April 2007)
4. Bronevetsky, G., de Supinski, B.R., Schulz, M.: A foundation for the accurate prediction of the soft error vulnerability of scientic applications. In: IEEE Workshop on Silicon Errors in Logic - System Effects (March 2009)
5. Cappello, F., Geist, A., Gropp, B., Kale, L., Kramer, B., Snir, M.: Toward exascale resilience. International Journal of High Performance Computing Applications 23, 374–388 (2009)
6. DeBardeleben, N., Laros, J., Daly, J., Scott, S., Engelmann, C., Harrod, B.: High-end computing resilience: Analysis of issues facing the hec community and path-forward for research and development (December 2009), `http://institute.lanl.gov/resilience/docs/HECResilience.pdf`

7. Dongarra, J., et al.: The international exascale software project roadmap. International Journal of High Performance Computing Applications 25, 3–60 (2011)
8. Kogge, P., et al.: Exascale computing study: Technology challenges in achieving exascale systems (2008)
9. Naughton, T., Bland, W., Vallee, G., Engelmann, C., Scott, S.L.: Fault injection framework for system resilience evaluation: fake faults for finding future failures. In: Proceedings of the 2009 Workshop on Resiliency in High Performance, Resilience 2009, pp. 23–28. ACM, New York (2009)
10. Quinn, H., Graham, P.: Terrestrial-based radiation upsets: A cautionary tale. In: Proceedings of the 13th Annual IEEE Symposium on Field-Programmable Custom Computing Machines, pp. 193–202. IEEE Computer Society, Washington, DC (2005)
11. Song, D., Brumley, D., Yin, H., Caballero, J., Jager, I., Kang, M.G., Liang, Z., Newsome, J., Poonsankam, P., Saxena, P.: A high-level overview covering vine, temu, and rudder. In: Proceedings of the 4th International Conference on Information Systems Security (December 2008)
12. Stott, D., Floering, B., Burke, D., Kalbarczpk, Z., Iyer, R.: Nftape: a framework for assessing dependability in distributed systems with lightweight fault injectors. In: Proceedings of IEEE International Computer Performance and Dependability Symposium, IPDS 2000, pp. 91–100 (2000)
13. Ziegler, J.F., Lanford, W.A.: The effect of sea level cosmic rays on electric devices. Journal Applied Physics 528 (1981)