GA

ANALYZING THE PROPAGATION OF SOFT ERROR CORRUPTION IN HPC APPLICATIONS

Jon Calhoun, University of Illinois at Urbana-Champaign
2016-2017 Graduate Fellow

EXECUTIVE SUMMARY

Because the rate of radiation-induced soft errors impacting application data are expected to increase on future high-performance computing (HPC) systems, analyzing how corruption due to soft errors propagates inside applications becomes critical for developing efficient detection and recovery schemes. Quantifying the latency (the number of instructions or iterations) of common symptoms of soft errors—e.g., crash or detection—allows for measuring the effectiveness of soft error detectors at containing corruption. Containing corruption allows for low-cost localized recovery instead of the high-cost global roll-back recovery checkpoint-restart. To analyze corruption propagation in HPC applications, an LLVM-based compiler tool instruments an application, allowing for tracking of corruption at an instruction and application variable level. Results show that the latency of common symptoms of soft errors along with speed of propagation to other processes varies dramatically. Blue Waters provided us with a platform to run thousands of fault-injection experiments and analyze hundreds of gigabytes of data efficiently.

RESEARCH CHALLENGE

Design constraints, such as cost of procurement and power budget, may make future HPC systems more susceptible to transient radiation-induced soft errors [1]. Soft errors commonly result in bit inversion and are the reason that all HPC main memories uses protection mechanisms such as error-correcting codes (ECC). On future systems, full protection from the effects of soft errors will be prohibitively expense. Thus, HPC applications running for long durations and at large scales may experience data corruption during execution [2]. Furthermore, global rollback-recovery becomes increasingly expensive at larger scales. Analyzing the effectiveness of current software-based detection schemes at containing data corruption is central in facilitating low-cost localized recovery and ensuring correct simulation results.

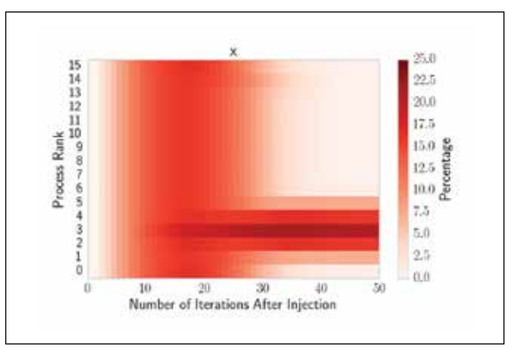

Figure 1: Propagation of corruption (percentage of elements deviated by at least 1e-10) in the solution variable of *x* of the application HPCCG.

METHODS & CODES

To track corruption propagation inside HPC applications, we constructed an LLVM compiler pass to create an executable file that emulates lockstep execution between a nonfaulty version of the application called *Gold* and a version that experiences a single bit-flip fault injection called *Faulty*. *Gold* produces a reference set of loads, stores, and correct program behavior that *Faulty* is judged against. The compiler pass completes the following steps when instrumenting the code: (1) it duplicates compiled code, forming two applications—*Gold* and *Faulty*, (2) it interleaves instructions from Gold and Faulty, emulating lockstep execution, (3) it instruments all loads and stores in *Faulty* with a function call to log deviations, (4) it instruments *Faulty* branches to check for control flow divergence, (5) it instruments faulty code for fault injection with FlipIt [3], and (6) it logs deviation in application-level variables at the end of iterations. This tool is used to investigate the latency (in number of LLVM instructions executed) of common symptoms of soft errors, the speed and extent of propagation between variables and parallel processes, and the impact of local problem size and compiler optimizations on propagation. The applications we test include: Jacobi, CoMD, and HPCCG. The latter two are taken from the Mantevo mini-app collection [4]. Each of these applications is modified with a lightweight soft-error detection scheme to investigate the latency of soft-error detection.

RESULTS & IMPACT

This project tracks corruption propagation at both the instruction level and the application variable level. Investigation at the instruction level shows the latency (in number of LLVM instructions) of common symptoms of soft errors: crash, control flow divergence, and soft-error detection. Application crashes have the shortest latency of the symptoms considered. Results show that 90% of crashes occur within four instructions. During those four instructions, there is little chance of corruption propagation. The latency of control-flow divergence and soft-error detection are noticeably larger. The larger latency allows for corruption to propagate to other variables and processes. To measure the extent and magnitude of corruption propagation in variables and processes, we track propagation at the application variable level.

Tracking propagation in application variables highlights dependencies between variables and processes. The application HPCCG uses the conjugate gradient linear solver to solve the sparse system of $Ax=b$. HPCCG is run 1,500 times with a single bit-flip fault injected on process rank 3 during every run. Fig. 1 shows the average percentage of elements deviated by more than 1e-10 (color) in the solution vector, x, for the application HPCCG across all MPI processes (y-axis) for subsequent iterations after injection (x-axis). As corruption in the solution vector propagates locally, the horizontal color for that row grows darker. As corruption is removed, the color lightens. Propagation between processes can be seen by looking at the color progression of columns at each iteration. Due to an `MPI_Allreduce` in the algorithm, corruption is present on all processes within one iteration after injection. However, the magnitude of the corruption is below our threshold deviation tolerance of 1e-10 and does not appear in Fig. 1. As HPCCG continues to iterate, corruption propagates in all processes. Over time, corruption is removed as HPCCG refines the solution to $Ax=b$. The largest in magnitude and most difficult to remove corruption remains on the process that experienced the fault, rank 3.

Analyzing HPC applications and how they propagate corruption is useful in determining what types of soft-error detectors to use and where they are best placed. Furthermore, knowing which variables are corrupted and on which processes corruption resides when a soft error is detected enables low-cost localized recovery instead of an expensive global rollback recovery, checkpoint-restart. Going forward, as HPC design constraints increase the likelihood of soft errors that can impact HPC application data, efficient detection and recovery schemes are needed to ensure applications obtain correct results.

WHY BLUE WATERS

The Blue Waters system allows us to perform fault injection and track propagation in thousands of HPC application runs and analyze the data generated more efficiently than other available systems. The fast turnaround time in obtaining and analyzing results greatly increased the speed and quality of this project.

PUBLICATIONS AND DATA SETS

Calhoun, J., M. Snir, L. Olson, and M. Garzaran, Understanding the Propagation of Error Due to a Silent Data Corruption in a Sparse Matrix Vector Multiply. *Proceedings of the 2015 IEEE International Conference on Cluster Computing*, (IEEE Computer Society, Chicago, Ill., September 8–11, 2015), pp. 541–542.

Calhoun, J., M. Snir, L. Olson, and W.D. Gropp, Towards a More Complete Understanding of SDC Propagation. To appear in *Proceedings of the 26th International Conference on High Performance and Distributed Computing*, (ACM, Washington, D.C., June 26–30, 2017).

Jon Calhoun was a fifth-year Ph.D. student in computer science working under the direction of Luke N. Olson and Marc Snir at the University of Illinois at Urbana-Champaign when this work was completed. He graduated in August 2017.