Fault Injection, Detection, and Correction in CLAMR using F-SEFI

Brian Atkinson, William M. Jones
Clemson University
Department of Electrical and Computer Engineering
Clemson, SC, USA
{bwa, wjones2}@clemson.edu

Nathan DeBardeleben, Qiang Guan
Los Alamos National Laboratory
HPC-5
Los Alamos, NM, USA
{ndebard, qguan}@lanl.gov

I. Introduction

As we continue the march towards exascale machines, the number of hardware components within the machines continue to increase. An unfortunate consequence of increaseing the number of hardware components, is that this also leads to an increase in hardware failure rates. While architects have made tremendious strides in detecting and correcting hardware faults, certain faults inherentenly still presist. When these hardware faults cause data corruption, a soft error has occurred. These soft errors, when undetected, can lead to silent data corruption (SDC), and these types of errors cannot be detected or corrected at the hardware level.

This means that designers of hight performance applications need to be aware of these types of faults in order to build better fault taulerant algorithms. These SDC's can have dramatic affects when undetected, because the corrupted data can progates throughout the run of the application. In high preformance scientific applications, data consistency is fundamental to achieving correct results. With our work, we used F-SEFI, a fine-grained soft error fault injector, to inject soft errors into CLAMR. CLAMR, a hydrodynamic application, provided the perfect testing ground for our soft error fault injections, because it was designed as a test bed for algorithm designs on next generation architectures.

Using F-SEFI with CLAMR, we were able to discover vulernabilities to soft errors within the application, and to visualize these errors. Using the information we gathered from the fault injections, we added functionality to the CLAMR code to help detect and recover from SDC errors induced from the faults that were injected.

II. CLAMR

CLAMR [1] is a cell-base adpative mesh refinement (AMR) hydrodynamic mini-app, that was developed at Los Alamos National Laboratory (LANL). CLAMR simulates a cylindrical shock in a level height pool of water. The shock creates waves, which reverberates off of boundaries and eventually dampen out. The simulation uses shallow water equations with Eulerian equations to simulate the water flow. CLAMR uses the conservation laws of mass, x momentum, and y momentum, which are kept in a state variables. These quantities are updated as the shock waves move throughout the mesh. The conservtion of mass is crucial to checking the consitency of the alogorithm, and it is periodically checked during the simulation. Since water is incompressible, the mass of each cell within the mesh is just the height of the water column within the cell due to uniform density. The total mass of water just a summation of each individual cells masses.

III. F-SEFI

F-SEFI [2] is a fine-grained software fault injector tool developed by LANL that is built on top of the QEMU. QEMU is a processor emulator viritual machine, which allows F-SEFI to experiment with injecting faults into different processors. Using the hyprovisor in QEMU, F-SEFI can monitor a program as it executing within the virutal machine. This allows for the F-SEFI to intercept instructions designated to run on the hardware and apply bit flips, soft errors, to specific targeted instructions.

This fine-grained approach leads to being able to target not only a specific instruction type, but also an instructions within a specfic target site within the application. F-SEFI applies soft errors based on parameters feed in through a configuration file. These bit flips can occur in specified bit ranges within floating point operations, or within integer logic operations. The exact site of the injection is recorded within a log, so it can reviewed for how the soft error manipulated the data.

These soft error injections can cause several different outcomes to occur. The fault injection may not cause any errors to be present at the completion of the application run, and these faults are catorgorized as beign. The injection can cause the application to reach an unstable state from which it can not recover, ultimately leading to the application crashing. Finally, the injection can cause data corruption, which leads to incorrect result at the end of the applications execution. This last case of an incorrect result is the most worriesome of all three cases, because this results in SDC.

IV. Fault injection and detection

Using F-SEFI on CLAMR [3] , we chose to inject the soft errors into the calc_finite_difference function. This function preforms most of the calculations for updating the state variables for each cells within the mesh. As the simulation progresses, calc_finite_difference is called repeatedly to update

the cells in the mesh based upon information from neighboring cells. After running numereous injections on different instructions types within calc_finite_difference, floating point adds (FADD's) had the most dramatic impact to the overall results of the calculations. Spefically, the exponent bit fields within the FADD's gave a good representation of the three different outcomes that can occur from soft error injections. Figure 1 shows the rate of these different outcomes for FADD's for 500 fault injections into the exponent bit fields.

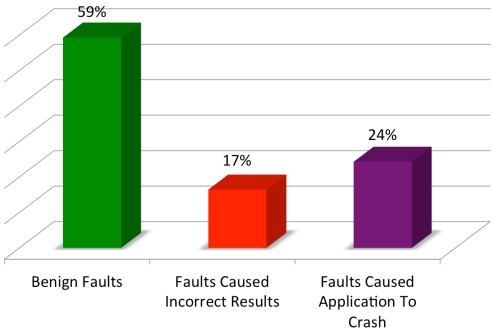

Fig. 1: Fault outcomes for FADD injections

Our results showed that the CLAMR application experienced 17% SDC to these fault injections. To gain a better understanding of the impacts of these injections, we utilized the visualizations built into CLAMR. These visualiztions of the simulation originally could only occur during run-time. To see the faults injections impact on the simulations, we stored the graphics data off to disk while CLAMR was running without visualiztions in the QEMU environment. This allowed us to visually see how the faults were impacting the application during runs with SDC's. Figures 2 shows a normal iteration of the CLAMR application, and Figure 3 shows a SDC caused by a fault injected into the application at the same iteration.

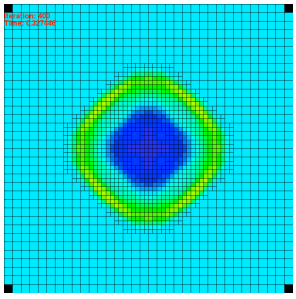

Fig. 2: Normal iteration of CLAMR simulation

Visualing the faults gave us a better understanding what kind of injections caused certain faults to occur. Using this insight, we turned to trying to not only detect SDC's after faults were injected, but also to try and recover from these faults. We did this by taking advantage of the conservation of mass check that was already implemented in CLAMR. During each check for the conservation of mass, a every small, but allowable difference, within the total water mass was checked. If the mass of the water had deviated too far from this allowable difference, then an error had occurred due to a injected fault.

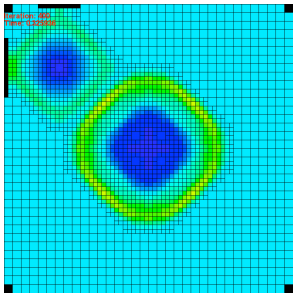

Fig. 2: Second force appears in CLAMR simulation due to fault injection. This was a case of SDC.

To recover from these faults, we implemented a checkpointing scheme, which was configurable from the command line at the start of CLAMR by the user. A user can specify how many checkpoints should be saved at any given point, the intervals at which the simulation should checkpoint it's current state, and the allowed tolerable deviation of mass difference between the checkpoint intervals. These checkpoints were saved to disk at the specified iterations, and were overwritten in a round robin fashion based on the number of checkpoints that were supposed to kept during the executation of CLAMR Using this, we set up a rollback feature to try and recover from any inconsist state that the faults had created.

At the conservation of mass check, if the mass had deviated too far from the allowable percentage, CLAMR would attempt to rollback to the most recent checkpointed state saved to disk. If it could not recover from the most recent checkpointed state, then it would go back to the second to last checkpointed state to try recover from the error. This process would continue for as many checkpointed states that were saved to disk. After implementing this detection and recovery scheme, we were able to successfully detect and recover from 81% of faults that would have caused the application to crash or would have lead to SDC.

V. Conclusion

The performance penalty that was accrued from keeping three checkpointed states to disk was a 1.26% overhead in total execution time. For simulations that required at most two rollbacks to safely recover from a fault, we an increase of 10.81% to the total execution time. By being aware of soft errors and their impacts; fault tolerance techniques can be introduced into scientific algorithms and applications. By doing this they can become resilient in spite of failure.

References

[1] D. Nicholaeff, N. Davis, D. Trujillo, and R. W. Robey. Cell-based adaptive mesh refinement implemented with general purpose graphics processing units. *Tech Report LA-UR-11-07127*, 2011.

[2] Q. Guan, N. DeBardeleben, S. Blanchard, and S. Fu. F-SEFI: A fine-grained soft error fault injection tool for profiling application vulnerability. In *IEEE 28th International Symposium on Parallel Distributed Processing (IPDPS)*, 2014.

[3] B. Atkinson, N. DeBardeleben, Q. Guan, R. Robey, W. M. Jones. Fualt injection experiments with the clamr hydrodynamics mini-app. In the *IEEE 25th International Symposium on Software Reliability Engineering*, Naples,Italy,November,2014.