Fault Injection Experiments With the CLAMR Hydrodynamics Mini-App

Brian Atkinson, Nathan DeBardeleben
Qiang Guan
Ultrascale Systems Research Center
Los Alamos National Laboratory
Los Alamos, NM
bwa@clemson.edu, {ndebard, qguan}@lanl.gov

Robert Robey
Eulerian Codes
Los Alamos National Laboratory
Los Alamos, NM
brobey@lanl.gov

William M. Jones
Department of Computer Science
Coastal Carolina University
Conway, SC
wjones@coastal.edu

Abstract—In this paper, we present a resilience analysis of the impact of soft errors on CLAMR, a hydrodynamics mini-app for high performance computing (HPC). We utilize F-SEFI, a fine-grained fault injection tool, to inject faults into the kernel routines of CLAMR. We demonstrate visually the impact of these faults as they are either benign (have no impact on the results), cause silent data corruption (SDC), or cause the application to crash due to instabilities. We quantify the probability that an injected fault will cause CLAMR to transition to one of the above three states using F-SEFI. Finally, we explore the relationship between the application’s fault characteristics and when the fault is injected in simulation time. Overall, we find that 17% and 24% of the faults propagate into SDC and crashes respectively.

Keywords—resilience; fault-tolerance; fault injection; hydrodynamics; mini-app;

I. INTRODUCTION

The march towards exascale computing poses extreme challenges for leadership-class supercomputers. These challenges are well understood [8] yet the scale of the challenges make the solutions to them somewhat less clear. Three of the chief concerns are power, programmability, and reliability. The U.S. Department of Energy (DOE) is targeting 20 megawatts for an exascale computer, a 35x increase in performance over today’s fastest supercomputer, at essentially no increase in power usage. Exascale systems are expected to have billion-way parallelism, posing challenges to programmability. There is currently much discussion about the programming languages and runtime systems of the future for systems such as these. Finally, several factors are combining to make fault-tolerance and resilience a larger problem than it is today.

As technology advances and circuit feature sizes continue to decrease, we can expect less reliability or sustained reliability at the cost of performance and/or power. Given the need for extreme performance combined with a daunting challenge of radically reducing power consumption, it seems reasonable to assume that building fault-tolerant applications for these machines will be important. Furthermore, while today’s largest systems have about one petabyte of main memory, exascale projections range between 32PB and 128PB. Similar increases in component counts of other portions of the system can be expected. Hard stop faults as well as soft errors causing application silent data corruption (SDC) will become more prevalent and applications intended to run on these systems must begin to prepare.

In this work, we leverage CLAMR [6], a publically available, hydrodynamics mini-application created by Los Alamos National Laboratory (LANL). Given that the application is representative of workloads of importance to LANL, and that LANL remains one of the premier supercomputing centers in the world, we contend that exascale computers will likely run applications similar to CLAMR. As such, we want to explore the resilience and vulnerability of this application to soft errors. To this end, we use the F-SEFI software-based fault-injection tool, also developed by LANL, to study CLAMR.

This research makes several contributions:

- We quantify the resilience of CLAMR to faults injected into a portion of the code where much of the application time is spent.
- We demonstrate using F-SEFI on a complex mini-app and show its usefulness in performing studies of application fault-tolerance.

The rest of the paper is organized as follows. In Section II we introduce CLAMR, a mini-app developed at LANL. In Section III we describe F-SEFI, the fault injection tool used to investigate the resilience of CLAMR, and also discuss our fault model. Results of fault injections are given in Section IV and we show three categories of results in the presence of injected faults. Related work is briefly discussed in Section V and then we conclude and discuss future work in Section VI.

II. CLAMR

CLAMR is a cell-based adaptive mesh refinement (AMR) hydrodynamic mini-app that simulates the shallow water equations. CLAMR was developed at LANL as a test-bed for algorithm development for next-generation architectures. The computational model uses the Eulerian equations to simulate the fluid flow. CLAMR harnesses the three conservation laws of mass, x momentum, and y momentum, that are inherent in the shallow water equations. At each time step, the state variables for height and momentum, in both the x and y
directions are updated. Due to the incompressibility of water, the density of the water can be treated as constant, and thus the height of the water column is essentially the total mass of that column. In the standard test problem of a circular dam break, a cylindrical pulse is created at the center of the mesh and the shallow-water equations are used to calculate the wave moving outward. As the shock reverberates off of the boundaries, the waves generated begin to dampen. This dampening eventually leads to a steady-state where the water settles to a uniform average pool height. The conservation of mass becomes crucially important in validating the consistency of the algorithm, and it is checked periodically during the run.

III. F-SEFI

F-SEFI [3] is a fine-grained software-based fault injector developed by Los Alamos National Laboratory. The tool is built on QEMU [1], a processor emulator virtual machine. QEMU is an actively developed VM hypervisor that exists under the popular VM, KVM.

Through F-SEFI, we are able to emulate soft errors at a virtual architecture level and target specific application binaries. While F-SEFI allows coarse-grained injection experiments without knowledge of source code, for this experiment we utilized source code knowledge to place faults with precision at specific line numbers of the CLAMR mini-app. In this way, we were able to precisely study how CLAMR responds to faults.

A. Fault Model

We used F-SEFI to perform campaign studies of fault injection experiments. Results were collected in a database for offline analysis. Also, intermediate states of the mesh were saved for correctness checking as well as to create movies which demonstrate how faults propagated through the mesh.

While F-SEFI is capable of injecting faults in any pattern, frequency, and target desired, for this work each run included a single bit flip in the exponent of a double precision floating point variable. We targeted the calc_.finite_difference function of state.cpp and the FADD (floating point addition). Once the fault was injected, CLAMR was simulated to a predetermined number of timesteps.

One of several outcomes can occur. The fault may not manifest in any final change to the application (benign). Secondly, the fault may cause the system to become unstable and crash. This often happens when physical properties become corrupted to the point where CLAMR is unable to continue, usually creating numerous NaNs. Finally, and most dangerously, the change may cause silent data corruption (SDC) which causes CLAMR to simulate to an incorrect state but not one that causes the system to abort.

It is this third state, that of silent data corruption, that is most worrisome. As CLAMR is a proxy of applications of interest to Los Alamos National Laboratory (LANL), it is also a simplified though meaningful vehicle for experimentation. Reproducible and verifiable computer simulations are of great importance to the U.S. Department of Energy (DOE) and studying the behavior of applications to known faults is valuable. Using F-SEFI we can (and have in follow-on work) experiment with detection and correction techniques that can be empirically shown to improve the reliability of CLAMR.

IV. RESULTS

A. Visualization of Injected Faults

We performed 500 separate fault injection experiments into CLAMR on a 32x32 grid. This grid is relatively small, but large enough that we were able to study the impact of faults. Each cell in the grid is solved independently and refined based on a 3-level refinement criteria adaptive mesh. In order to better study the impact of the faults, a splash was stimulated in the center of the grid and the symmetry was retained during the surface gravity wave propagating away from the center spot and reflecting off the boundary of the grid. In this way, we expected most SDCs to break the symmetry of this sample problem and be easier to study. Figure 1 shows the fault-free simulation execution after 400 iterations and the colors from blue to red represent the depth of water from low to high.

After fault injection, we categorized the behavior of CLAMR into three different classes: benign, silent data corruption (SDC), and crash. The output is benign if both the final result and all the intermediate states at different iterations are the same as the fault-free execution. The results of our experiments are shown in Figure 2. 59% of faults injected into CLAMR have no effect on the intermediate and final states (are benign), 17% of faults injected corrupt the intermediate states of CLAMR and produce incorrect results. Finally, 24% of faults were observed to cause the application to crash immediately or after several iterations.

The SDC and crash cases can be further classified into two sub-categories each, and are shown in Figure 3. Figures 3(a) and 3(b) demonstrate SDC while Figures 3(c) and 3(d) demonstrate crashes.

In Figure 3(a), an extra splash is introduced by the injected fault and the contamination quickly propagates from faulty cells to healthy cells, which is independent from the other wave of water from the center. These waves then begin interacting and the symmetry of this sample problem is destroyed. In Figure 3(b), the shadow area (upper-right) caused by an
(a) An injected fault causes a silent data corruption manifesting as an extra splash.

(b) An injected fault causes a silent data corruption manifesting as a small ripple which dampens over time.

(c) An injected fault causes an inconsistent state to some cells and quickly propagates the inconsistency to the neighbors until the simulation crashes. The tainted cells do not affect the wave propagating from the center before crashing.

(d) An injected fault causes an intolerable splash that hinders the movement of water from the center and crashes the simulation immediately without propagation.

Fig. 3. Comparative outputs with four different consequences of executing CLAMR under fault injection with F-SEFI. Each shows the results of a single \textit{FADD} fault. Figures 3(a) and 3(b) show examples of SDCs while Figures 3(c) and 3(d) show examples of catastrophic data corruption that result in crashes.

In contrast to SDCs, some faults are drastic enough that they cause the application to become unstable and crash the simulation prematurely. Two examples of these are shown in Figures 3(c) and 3(d). In Figure 3(c) the injected fault forces several of the cells into an inconsistent state and this inconsistency spreads to neighbor cells before crashing the simulation. In Figure 3(d), the injected fault develops into another splash, the mesh refines, and several iterations later the application becomes unstable and crashes.

B. Impact of Faults With Respect to Simulation Time

CLAMR’s ability to tolerate faults depends on when the faults are injected. For the sample problem used in this work, CLAMR was run to 5000 iterations and around iteration 1500 the waves hit the wall and begin to bounce back inward. We studied four distinct iterations as targets for fault injection and one seen in Figure 3(a).

In contrast to SDCs, some faults are drastic enough that they cause the application to become unstable and crash the simulation prematurely. Two examples of these are shown in Figures 3(c) and 3(d). In Figure 3(c) the injected fault forces several of the cells into an inconsistent state and this inconsistency spreads to neighbor cells before crashing the simulation. In Figure 3(d), the injected fault develops into another splash, the mesh refines, and several iterations later the application becomes unstable and crashes.

Fig. 2. CLAMR’s fault tolerance over 500 injected faults.
the results are shown in Figure 4. From the results, we can see that early in the simulation, faults cause more damage than they do later in simulation time. For this application and problem setup, as the application progresses, the rate of SDC and crashes is reduced to under ten percent each. One possible reason for this is that early in the simulation much of the grid is quiet and large perturbations to that quiet space are so radically different from the surrounding areas. In contrast, as more of the grid becomes active, these perturbations get smoothed by neighboring cells.

V. RELATED WORK

While there are many related tools and studies of application vulnerability, due to space limitations, we briefly discuss a sampling of some closely related work.

Li et al. [5] investigate the impacts of soft errors on mission critical scientific applications deployed in the Jaguar supercomputer at Oak Ridge National Laboratory using the fault injection tool BIFIT. They show that applications react differently to fault injections at different data objects sites and different time bins. This is similar to our findings in Figure 4.

Fang et al. [2] study the resilience of GPU/CPU OpenMP programs using LLFI, which is developed based on LLVM [4]. Faults are injected into master and slave threads to characterize the error resilience separately. The injection results show that the SDC rates are highly correlated with the applications’ type. While we have previously studied kernels and benchmark applications like this related work, in this paper we are studying a complex physics application.

In order to address the impact of transient errors in GPUs, Yim et al. [9] analyze the vulnerability problem in GPGPU scientific 3D simulation programs: particle-to-grid and N-Body programs that are applied to molecular modeling. A software implemented fault injector is used for injecting single-bit faults into the GPU kernels at run-time. Unlike Yim’s work, CLAMR’s sample problem symmetry provides an easy visualization of corruption that doesn’t require domain knowledge to interpret.

Parasyris et al. [7] use GemFI, a full system emulator, to profile the behaviors of real-world kernels and applications on unreliable hardware. Similar to F-SEFI, GemFI injects faults at the Instruction Set Architecture (ISA) level. To the best of our knowledge, GemFI has not been used to study a complex physics application yet.

VI. CONCLUSIONS AND FUTURE WORK

In this work we experimentally evaluated the fault-tolerance of CLAMR to soft errors injected into the simulation kernel at a fine granularity. We have shown visual examples of how the faults manifest in the application and propagate through to cause silent data corruption or crash the application. For the problem we studied in CLAMR, we saw that around 41% of injected faults caused some form of unfavorable behavior. We also saw that this behavior varied with how far into the application the fault was introduced.

In this brief paper we have outlined only a portion of the work we are doing in this area. We have separately experimented with conservation of mass checks to detect data corruption. In future work we intend to demonstrate new features we have added to CLAMR to detect and roll-back to a previously known good state of the simulation.

As reliability continues to be a challenge moving forward, getting a deep understanding of an application’s tolerance and vulnerability to faults will be vitally important. Through using representative important applications like CLAMR combined with a fault-injection tool like F-SEFI, we have shown we are able to do this type of experiment and obtain meaningful, actionable results.

REFERENCES