Testing Research Software: A Case Study

Nasir U. Eisty[1], Danny Perez[2], Jeffrey C. Carver[1], J. David Moulton[2], and Hai Ah Nam[2]

[1] University of Alabama, Dept. of Computer Science, Tuscaloosa, AL, USA
neisty@crimson.ua.edu, carver@cs.ua.edu
[2] Los Alamos National Laboratory, Los Alamos, NM, USA danny_perez@lanl.gov, moulton@lanl.gov, hnam@lanl.gov

Abstract. *Background*: The increasing importance of software for the conduct of various types of research raises the necessity of proper testing to ensure correctness. The unique characteristics of the research software produce challenges in the testing process that require attention. *Aims*: Therefore, the goal of this paper is to share the experience of implementing a testing framework using a statistical approach for a specific type of research software, i.e. non-deterministic software. *Method*: Using the *ParSplice* research software project as a case, we implemented a testing framework based on a statistical testing approach called **Multinomial Test**. *Results*: Using the new framework, we were able to test the *ParSplice* project and demonstrate correctness in a situation where traditional methodical testing approaches were not feasible. *Conclusions*: This study opens up the possibilities of using statistical testing approaches for research software that can overcome some of the inherent challenges involved in testing non-deterministic research software.

Keywords: Research Software · Testing · Software Engineering.

1 Introduction

Research software can enable mission-critical tasks, provide predictive capability to support decision making, and generate results for research publications. Faults in research software can produce erroneous results, which have significant impacts including the retraction of publications [8]. There are at least two factors leading to faults in research software: (1), the complexity of the software (often including non-determinism) presents difficulties for implementing a standard testing process and (2) the background of people who develop research software differ from traditional software developers.

Research software often has complex, non-deterministic computational behavior, with many execution paths and requires many inputs. This complexity makes it difficult for developers to manually identify critical input domain boundaries and partition the input space to identify a small but sufficient set of test cases. In addition, some research software can produce complex outputs whose assessment might rely on the experience of domain experts rather than on an

ICCS Camera Ready Version 2020
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-50436-6_33

objective test oracle. Finally, the use of floating-point calculations can make it difficult to choose suitable tolerances for the correctness of outputs.

In addition, research software developers generally have a limited understanding of standard software engineering testing concepts [7]. Because research software projects often have difficulty obtaining adequate budget for testing activities [11], they prioritize producing results over ensuring the quality of the software that produces those results. This problem is exacerbated by the inherent exploratory nature of the software [5] and the constant focus on adding new features. Finally, researchers usually do not have training in software engineering [3], so the lack of recognition of the importance of the corresponding skills causes them to treat testing as a secondary activity [10].

To address some of the challenges with testing research software, we conducted a case study on the development of a testing infrastructure for the *ParSplice*[3] research software project. The goal of this paper is to *demonstrate the use of a statistical method for testing research software*. The key contributions of this paper are (1) an overview of available testing techniques for non-deterministic stochastic research software, (2) implementation of a testing infrastructure of a non-deterministic parallel research software, and (3) demonstration of the use of a statistical testing method to test research software that can be a role model for other research software projects.

2 Background

In a non-deterministic system, there is often no direct way for the tester (or test oracle) to exactly predetermine the expected behavior. In *ParSplice* (described in Section 2.1), the non-determinism stems from (1) the use of stochastic differential equations to model the physics and (2) the order in which communication between the procedures occurs (note however that even though the results from each execution depends upon message ordering, each valid order produces a statistically accurate result, which is the key requirement for the validity of *ParSplice* simulations).

In cases where development of test oracles is difficult due to the non-determinism, some potentially viable testing approaches include metamorphic testing, runtime assertions, and machine learning techniques [6]. After describing the *ParSplice* project, the remainder of this section explains these techniques along with their possible applicability to *ParSplice*.

2.1 ParSplice

ParSplice (Parallel Trajectory Splicing) [9] aims at overcoming the challenge of simulating the evolution of materials over long time scales through the timewise parallelization of long atomistic trajectories using multiple independent producers. The key idea is that statistically accurate long-time trajectories can

[3] https://gitlab.com/exaalt/parsplice

ICCS Camera Ready Version 2020
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-50436-6_33

be assembled by splicing end-to-end short, independently-generated, trajectory segments. The trajectory can then grow by splicing a segment that begins in the state where the trajectory currently ends, where a state corresponds to a finite region of the configuration space of the problem. This procedure yields provably statistically accurate results, so long as the segments obey certain (relatively simple) conditions. Details can be found in the original publication [9].

The *ParSplice* code is a management layer that orchestrates a large number of calculations and does not perform the actual molecular dynamics itself. Instead, *ParSplice* uses external molecular dynamics engines. The simulations used in *ParSplice* rely on stochastic equations of motion to mimic the interaction of the system of interest with the wider environment, which introduces a first source of non-determinism.

A basic ParSplice implementation contains two types of processes: a splicer and producers. The splicer manages a database of segments, generates a trajectory by consuming segments from the database, and schedules execution of additional segments, each grouped by their respective initial state. Producers fulfill requests from the splicer and generate trajectory segments beginning in a given state; the results are then returned to the splicer. The number of segments to be scheduled for execution in any known state is determined through a predictor statistical model, built on-the-fly. Importantly, the quality of the predictor model only affects the efficiency of *ParSplice* and not the accuracy of the trajectory. This property is important because the predictor model will almost always be incomplete, as it is inferred from a finite number of simulations. The unavailability of the ground truth model (which is an extremely complex function of the underlying physical model) makes assessment of the results difficult. In addition, this type of stochastic simulation is not reproducible, adding to the difficulty of testing the code. Therefore, in this case study we create a basis for the *ParSplice* testing infrastructure using various methodical approaches and apply the test framework to the continuous integration process.

2.2 Metamorphic Testing

Metamorphic testing operates by checking whether the program under test behaves according to a set of metamorphic relations. For example, a metamorphic relation R would express a relationship among multiple inputs x1, x2,.., xN (for N $>$ 1) to function f and their corresponding output values *f(x1)*, *f(x2)*,.., *f(xN)* [2]. These relations specify how a change to an input affects the output. These metamorphic relations serve as a test oracle to determine whether a test case passes or fails. In the case of *ParSplice*, it is difficult to identify metamorphic relations because the outputs are non-deterministic. The relationship between the x's and the f's is therefore not direct but statistical in nature.

2.3 Run-time Assertion Checking

An assertion is a boolean expression or constraint used to verify a necessary property of the program under test. Usually, testers embed assertions into the

ICCS Camera Ready Version 2020
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-50436-6_33

source code that evaluate when a test case is executed. Later testers use these assertions to verify whether the output is within an expected range or if there are some known relationships between program variables. In this way, a set of assertions can act as an oracle. In the context of *ParSplice*, assertions can be used to test specific functions, but not to test the overall validity of the simulations, would protect only against catastrophic failures, such as instabilities in the integration scheme.

2.4 Machine Learning Techniques

Machine learning is a useful approach for developing oracles for non-deterministic programs. Researchers have shown possibilities of both black-box features (developed using only inputs and outputs of the program) and white-box features (developed using the internal structure of the program) to train the classifier used as the oracle [1] [4]. It is possible to test *ParSplice* with machine learning techniques. For example, we could fake the molecular dynamics (MD) engine with our own model to produce output data to use as a training set and consider the actual output data as a testing set. Due to the amount of effort required to use this approach in *Parsplice*, we determined that it was not feasible.

3 Case Study

To implement the testing framework, the first author spent a summer at Los Alamos National Laboratory working on the *ParSplice* project. The testing framework is based on the Multinomial testing approach (described in Section 3.2), implemented using a *progress tracking card* (PTC) in the Productivity and Sustainability Improvement Plan (PSIP)[4] methodology. The testing approach is integrated with the CMake/CTest tool for use in the runtime environment and continuous integration. In this section, we describe the PSIP methodology, the Multinomial test approach, and results that verify the implementation of the testing framework.

3.1 PSIP

The PSIP methodology provides a constructive approach to increase software quality. It helps decrease the cost, time, and effort required to develop and maintain software over its intended lifetime. The PSIP workflow is a lightweight, multi-step, iterative process that fits within a project's standard planning and development process. The steps of PSIP are: a) Document Project Practices, b) Set Goals, c) Construct Progress Tracking Card , d) Record Current PTC Values, e) Create Plan for Increasing PTC values, f) Execute Plan, g) Assess Progress, h) Repeat.

We created and followed a PTC containing a list of practices we were working to improve, with qualitative descriptions and values that helped set and track

[4] https://betterscientificsoftware.github.io/PSIP-Tools/PSIP-Overview.html

ICCS Camera Ready Version 2020
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-50436-6_33

our progress. Our progress tracking card consists of 6 scores with a target finish date to develop the testing framework. The scores are:

- Score 0 - No tests or approach exists
- Score 1 - Requirement gathering and background research
- Score 2 - Develop statistical test framework
- Score 3 - Design code backend to integrate test
- Score 4 - Test framework implemented into ParSplice infrastructure
- Score 5 - Integrate into CI infrastructure

We were able to progress through these levels and obtain a score of 5 by the end of the case study.

3.2 Multinomial Test

The Multinomial test is a statistical test of the null hypothesis that the parameters of a multinomial distribution are given by specified values. In a multinomial population, the data is categorical and belongs to a collection of discrete non-overlapping classes. For instance, multinomial distributions model the probability of counts of each side for rolling a k-sided die n times. The Multinomial test uses Pearson's χ^2 test to test the null hypothesis that the observed counts are consistent with the given probabilities. The null hypothesis is rejected if the p-value of the following χ^2 test statistics is less than a given significance level. This approach enables us to test whether the observed frequency of segments starting in i and ending in j is indeed consistent with the probabilities p_{ij} given as input to the Monte Carlo backend. Our Multinomial test script uses the output file of *ParSplice* as its input and execute the test and post-processes the results by performing Pearson's χ^2 to assess whether to reject the null-hypothesis.

3.3 Results

A key insight from the theory that underpins *ParSplice* is that a random process that describes the splicing procedure should rigorously converge to a discrete time Markov chain in a discrete state space. In other words, the probability that a segment added to a trajectory currently ending in state i leaves the trajectory in state j should be a constant p_{ij} that is independent of the past history of the trajectory. One way to test *ParSplice* would be to verify that the splicing procedure is indeed Markovian (memory-less). However, taken alone, such a test would not guarantee that the splicing proceeds according to the proper Markov chain. A more powerful test would assess whether the spliced trajectory is consistent with the ground-truth Markov chain. A key obstacle to such a test is that this ground-truth model is, in practice, unknown and can only be statistically parameterized from simulation data.

To address this issue, we replaced the molecular dynamics (MD) simulation backend with a simpler Monte Carlo implementation that samples from a pre-specified, Markov chain. That is, we replaced the extremely complex model

ICCS Camera Ready Version 2020
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-50436-6_33

inherent to the MD backend with a known, given model of predefined probabilities. The task then becomes assessing whether the trajectory generated by *ParSplice*, as run in parallel on large numbers of cores, reproduces the statistics of the ground truth model. In this context, this technique is the ultimate test of correctness, as *ParSplice* is specifically designed to parallelize the generation of very long trajectories that are consistent with the underlying model. Statistical agreement between the trajectory and the model demonstrates that the scheduling procedure is functional (otherwise, the splicing the of trajectory would halt), the task ordering procedure is correct, the tasks executed properly, the results reduced correctly, and the splicing algorithm was correct.

The statistical assessment to test *ParSplice* can be conducted using the Multinomial test approach. Our null hypothesis was that the observed counts generated by *ParSplice* are consistent with the probabilities in the model. If the p-value from the multinomial test is less than 0.05, we reject the null hypothesis and conclude that the observed counts differ from the expected ones. Conversely, if the p-value is greater than 0.05, we do not reject the null hypothesis and can conclude that the test passes. For the sake of verifying our Multinomial test, we ran *ParSplice* in different time frames and observed the result. Figure 1 shows the p-values obtained from running *ParSplice* for 1, 2, 5, 10, 20, 40, 60, and 90 minutes. We can see that in all cases, the p-values are greater than 0.05, which indicates that the tests passed during these instances of the execution.

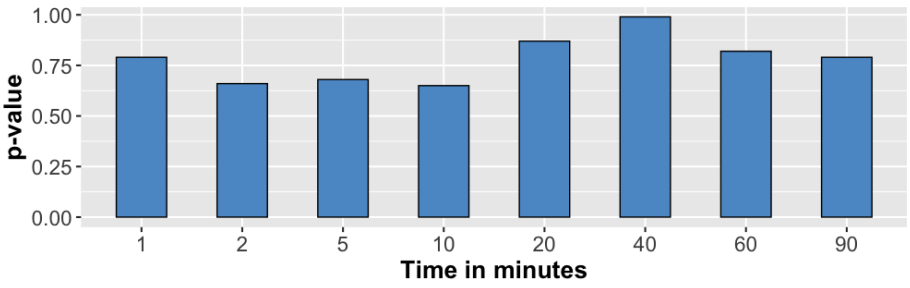

Fig. 1. p-values obtained by executing *ParSplice* for different times.

4 Conclusion

In this paper, we describe a case study of the *ParSplice* project in which we followed the PSIP methodology to develop a testing framework to address the difficulties of testing non-deterministic parallel research software. We first considered applying traditional industrial testing approaches. However, the non-determinism of *ParSplice* made these approaches unusable. Then we identified testing techniques specially designed for non-deterministic software. Once again,

ICCS Camera Ready Version 2020
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-50436-6_33

those techniques did not fit *ParSplice*. Finally, we identified a statistical testing approach, Multinomial Testing, that would work for *ParSplice*.

The Multinomial Testing approach is ideal for *ParSplice* given its constraints, i.e. time, non-determinism, and the existing continuous integration system. The lessons learned from this case study can be valuable to the larger research software community because, like *ParSplice*, many research software projects have stochastic behavior which produces non-deterministic results. The approach we followed to develop the test framework can be a model for other research software projects. We plan to extend the testing infrastructure in a more methodological way with as many possible testing techniques installed in the system.

Acknowledgement

This research was supported by the Exascale Computing Project (17-SC-20-SC), a collaborative effort of the U.S. Department of Energy Office of Science and the National Nuclear Security Administration. (LA-UR-20-20082)

References

1. Chan, W., Cheung, S., Ho, J.C., Tse, T.: Pat: A pattern classification approach to automatic reference oracles for the testing of mesh simplification programs. Journal of Systems and Software **82**(3), 422 – 434 (2009)
2. Chen, T.Y., Tse, T.H., Zhiquan Zhou: Fault-based testing in the absence of an oracle. In: 25th Annual International Computer Software and Applications Conference. COMPSAC 2001. pp. 172–178 (Oct 2001)
3. Easterbrook, S.M., Johns, T.C.: Engineering the software for understanding climate change. Computing in Science Engineering **11**(6), 65–74 (Nov 2009)
4. Frounchi, K., Briand, L.C., Grady, L., Labiche, Y., Subramanyan, R.: Automating image segmentation verification and validation by learning test oracles. Inf. Softw. Technol. **53**(12), 1337–1348 (Dec 2011)
5. Heroux, M.A., Willenbring, J.M., Phenow, M.N.: Improving the development process for cse software. In: 15th EUROMICRO International Conference on Parallel, Distributed and Network-Based Processing (PDP'07). pp. 11–17 (Feb 2007)
6. Kanewala, U., Bieman, J.M.: Techniques for testing scientific programs without an oracle. In: Proceedings of the 5th International Workshop on Software Engineering for Computational Science and Engineering. pp. 48–57. SE-CSE '13 (2013)
7. Kanewala, U., Bieman, J.M.: Testing scientific software: A systematic literature review. Inf. Softw. Technol. **56**(10), 1219–1232 (Oct 2014)
8. Miller, G.: A scientist's nightmare: Software problem leads to five retractions. Science **314**(5807), 1856–1857 (2006). https://doi.org/10.1126/science.314.5807.1856
9. Perez, D., Cubuk, E., Waterland, A., Kaxiras, E., Voter, A.: Long-time dynamics through parallel trajectory splicing. Journal of Chemical Theory and Computation **12** (11 2015)
10. Segal, J.: Some problems of professional end user developers. IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC) pp. 111–118 (2007)
11. Segal, J.: Software development cultures and cooperation problems: A field study of the early stages of development of software for a scientific community. Computer Supported Cooperative Work (CSCW) **18**(5), 581 (Sep 2009)

ICCS Camera Ready Version 2020
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-50436-6_33