Testing High Performance Numerical Simulation Programs: Experience, Lessons Learned, and Open Issues

Xiao He*
Xingwei Wang
Jia Shi
hexiao@ustb.edu.cn
School of Computer and Communication Engineering,
University of Science and Technology Beijing
Beijing, China

Yi Liu*
liuyi@cert.org.cn
National Computer Network Emergency Response
Technical Team/Coordination Center of China
Beijing, China

ABSTRACT

High performance numerical simulation programs are widely used to simulate actual physical processes on high performance computers for the analysis of various physical and engineering problems. They are usually regarded as non-testable due to their high complexity. This paper reports our real experience and lessons learned from testing five simulation programs that will be used to design and analyze nuclear power plants. We applied five testing approaches and found 33 bugs. We found that property-based testing and metamorphic testing are two effective methods. Nevertheless, we suffered from the lack of domain knowledge, the high test costs, the shortage of test cases, severe oracle issues, and inadequate automation support. Consequently, the five programs are not exhaustively tested from the perspective of software testing, and many existing software testing techniques and tools are not fully applicable due to scalability and portability issues. We need more collaboration and communication with other communities to promote the research and application of software testing techniques.

CCS CONCEPTS

• **Software and its engineering → Software testing and debugging**.

KEYWORDS

Software testing, Experience, Numerical simulation, High performance computing

ACM Reference Format:
Xiao He, Xingwei Wang, Jia Shi, and Yi Liu. 2020. Testing High Performance Numerical Simulation Programs: Experience, Lessons Learned, and Open Issues. In *Proceedings of the 29th ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA '20), July 18–22, 2020, Virtual Event, USA.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3395363.3397382

*Corresponding Authors

1 INTRODUCTION

High performance numerical simulation programs (HPNSPs) are a sort of computer programs that are widely used to simulate complex real-world systems on high performance computers according to mathematical and physical models. They enable us to reproduce, analyze, and predict the behavior of a system in physics, chemistry, biology, economics, psychology, and social science. They play a crucial role in modern science and engineering. Many spectacular findings [4, 26, 48, 56] are made with the help of HPNSPs. For example, Qing Peng et al. [48] revealed the formation mechanism of ⟨100⟩ dislocation loops, the defects that greatly decrease the strength of alloyed iron, by adopting large-scale molecular dynamics simulation. HPNSPs also contribute to our daily lives from many aspects. For example, they have been applied to weather forecast [6, 45], earth quake analysis [18, 28], and traffic simulation [17].

The quality of these simulation programs is critical to their successful application. If these programs have defects in mathematical models, numerical algorithms, code implementation, and input data, then any conclusion, decision, or scientific finding made from the simulation results may become unreliable.

Software testing has been demonstrated as an effective way of the verification and validation of software programs. The theories, the techniques, and the tools of software testing have been intensively discussed in all mainstream software engineering forums, and they have been successfully applied to various sorts of systems, such as Web and cloud services [14, 58], mobile applications [41, 43, 61], AI systems [36, 47], and compilers [9, 10, 37].

Surprisingly, there is very limited discussion on how to test high performance numerical simulation programs in the software testing community. Existing research efforts on numerical program analysis [5, 7, 30, 38, 44, 55, 59, 66, 68, 70, 71] mainly focus on simple or preliminary math functions. It is unknown for a software testing practitioner how HPNSPs are tested now, what obstacles there are, and whether the testing techniques that are intensively discussed in the software testing community can improve the testing process.

The major objective of this paper is to present our real-world experience of testing high-performance numerical simulation programs from the perspective of software testing practitioners in the context of a cross-discipline research and development (R&D) team who collaboratively work for a national project. The R&D team aims to establish a multi-physics numerical simulation system for nuclear power plants deployed on high performance computers. The system comprises of several numerical simulation programs.

Experts of nuclear physics, material science, applied mathematics, numerical analysis, high performance computing (HPC), computer graphics and software engineering, coming from 10 universities and institutes all-over the country, participated in this project. We joined this team as software experts two years ago, and are responsible for testing these HPNSPs. After struggling for two years, we realized that the testing of HPNSPs is far more complicated than we expected. There are many challenges and pitfalls in practice.

This paper mainly focuses on the following research questions.

RQ.1 What testing approaches are applicable and effective?
RQ.2 Where can we find/reuse existing test inputs? Can we randomly generate new test inputs?
RQ.3 What kinds of test oracles are in use and how to obtain them?
RQ.4 What are the most frequently occurring bugs?

In our practice, we adopted five testing approaches and found 33 bugs in total. Based on our experience, our major conclusions are summarized as follows. (1) Property-based testing and metamorphic testing are two effective approaches to testing HPNSPs. (2) Domain knowledge is crucial for testing HPNSPs, but how to express and exchange domain knowledge for testing is an open issue. (3) Due to the lack of test cases/inputs, test oracles, and automation, our programs under test are not exhaustively tested from the perspective of software testing. (4) Hindered by many practical issues, e.g., scalability, many cutting-edge techniques are not fully applicable. We believe that software testing community should have more collaboration and communication with other communities and domain experts to develop new or adapt existing testing techniques for complex computation systems.

The rest of this paper is structured as follows. Section 2 introduces the numerical simulation programs under test and their general structures. Section 3 presents our real experience of testing these programs in detail. Section 4 answers the research questions. Section 5 summarizes the lessons learned, open issues, and threats to validity. Section 6 discusses the related work. The last section presents the conclusion.

2 BACKGROUND

2.1 Units of Analysis

This paper concentrates on the following five high performance numerical simulation programs as the units of analysis in this study because their developers believed they were ready for testing.

- **ANT-MOC** is a simulation program of neutral particle transport based on the method of characteristics. ANT-MOC originates from openMOC [8], but differs from openMOC in the parallelization strategy and the geometrical model supported. ANT-MOC has 31K lines of C++ code and 18 parameters, e.g., a geometrical model that is specified by CAD software, a material data file, and the number of azimuthal and polar angles. Given a geometrical model, ANT-MOC will generate many characteristic lines. Each characteristic line represents the movement trace of a neutral particle. Then, ANT-MOC iteratively updates the movements and states of all particles along these characteristic lines, till result converges. ANT-MOC outputs the spatial distribution of reaction rates that may be visualized as 3D images.
- **MISA-MD** is a molecular dynamics simulation program that is designed and developed by our R&D team. Its functionality is partially comparable with LAMMPS [50]. MISA-MD has 10K lines of C++ code and 16 parameters, e.g., a table file of a potential function, the initial system temperature, and Fe content. At first, MISA-MD will divide the simulation space into many grids and initialize the state of each atom in every grid. Then, for each grid, the program iteratively updates the movement and the state of every atom, and communicates the information of the atoms that move in/out with neighbor grids during each iteration, till the simulation time is reached. The output is the data of atom movement. A 3D animation may be created to show the movement of all atoms.
- **MISA-SCD** is a spatially resolved stochastic cluster dynamics simulation program that computes defect evolution in irradiated materials using a synchronous parallel kinetic Monte Carlo algorithm. MISA-SCD adapts SRSCD [21] for the simulation of I-V-Cu defect species, so SRSCD is not comparable. MISA-SCD has 14K lines of Fortran code and 30 parameters, e.g., a geometrical model file, a material data file, initial temperature, and dislocation density. At first, MISA-SCD will divide the given space into many grids. Then, for each grid, MISA-SCD iteratively updates the defect information and communicates it with neighbor grids, till the simulation time is reached. The outputs are the numbers of defects of every defect species in each grid.
- **MISA-RT** is a deterministic mean-field cluster dynamics simulation program that computes the density of material defects (i.e., clusters). It has 489 lines of C++ code and 4 parameters, i.e., the simulation time, the length of time step, the initial cluster density, and the maximum cluster diameter. At first, MISA-RT generates many ordinary differential equations (ODEs) and sub-divides them into many groups. For each equation group, the program solves the ODEs repeatedly till the simulation time is reached. The output is a density distribution table of material defects.
- **ATHENA** is a simulation program of steady state thermal-mechanical behavior of oxide fuel rods. ATHENA currently has 9K lines of C++ code, and is a rewrite of a Fortran program FRAPCON [1]. It has 132 parameters, including the total number of simulation steps, an array of linear heat generation rates for every step, and the number of rods. First, each rod is split into many grids. For each grid, ATHENA uses a nested loop to realize the calculation, where the inner loop stops when the result converges and the outer loop stops when the simulation reaches a certain number of steps. Finally, the program outputs the temperature, pressure, and deformation of a fuel rod as functions of time-dependent fuel rod power and coolant boundary conditions. The output file may contain more than 20,000 lines of text.

2.2 General Structure and Characteristics of High-Performance Simulation Programs

By summarizing the units of analysis, the general structure of high performance numerical simulation programs can be depicted in Fig. 1. Despite of some variations, our units of analysis follow this

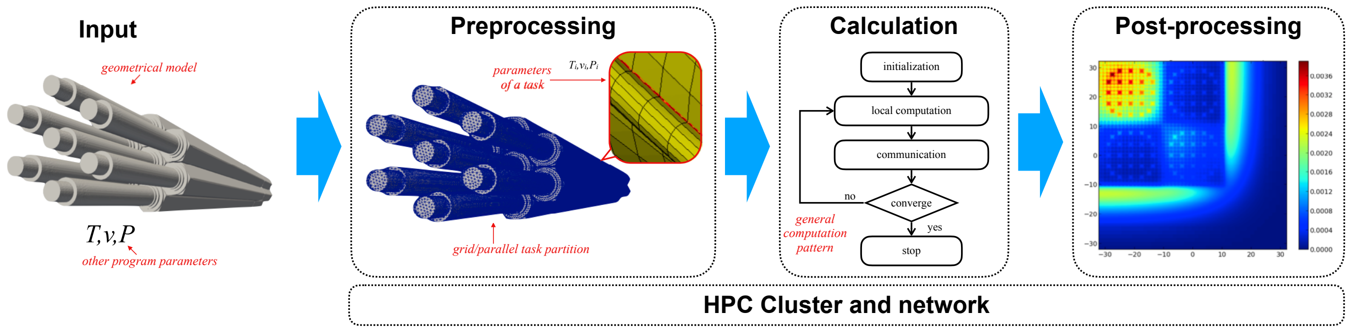


Figure 1: General structure and workflow of high performance numerical simulation

general structure. As shown in Fig. 1, anan HPNSP HPNSP can be divided into three parts as follows.

- **Preprocessing** module is responsible for dividing the entire computation into a number of parallel tasks (e.g., a characteristics line in ANT-MOC and a grid in MISA-MD and MISA-SCD) and preparing the execution parameters for each parallel task, according to the program input. The program input usually consists of a geometrical model, some physical parameters and constants (e.g., the number of molecules, and the initial pressure and temperature), and some boundary parameters (e.g., the total simulation time). If the geometrical model is available, the commonest way of task partition is to divide the geometrical space into many grids that can be further mapped onto parallel tasks.
- **Computation** module is the implementation of every parallel task that is produced by preprocessing. This module contains several local computation steps and communication steps. At runtime, a process (i.e., an instance of the computation module) is created for each parallel task with its own execution parameters. A process is deployed onto a node of a cluster (or a supercomputer). All processes are run in parallel and communicate mutually via high speed network.
- **Post-processing** module is responsible for gathering the outputs from parallel tasks. Optionally, post-processing visualizes the simulation result using proper graphical representations (e.g., for ANT-MOC and MISA-MD).

According to the general structure above, we identify the following characteristics that are shared by most HPNSPs.

- *The general pattern of the computation module is a loop (or a loop of loops).* The reason is twofold. First, the simulation of a physical phenomenon usually involves a simulation time, and the program updates the system state for each time step till the simulation time is reached (e.g., MISA-MD and MISA-SCD). Second, some numerical algorithms iteratively calculate and refine solutions until convergence errors are lower than a threshold (e.g., ANT-MOC and ATHENA).
- *HPNSP is usually a long-term calculation.* The simulation of complex natural systems requires tremendous computations, even the program is parallelized and deployed on high performance clusters. For instance, ANT-MOC requires at least 2300 core hours to compute a median-sized problem.
- *The computation module may be stochastic.* The randomness mainly comes from the numerical algorithms (e.g., Monte Carlo algorithm in MISA-SCD) that are based on the probability theory. In those programs, computation usually adopts random sampling or decisions. To handle randomness, multiple executions and more computation resources are required.
- *Numerical simulation is by nature inaccurate.* There are many factors besides bugs that contribute to the inaccuracy, including the inaccurate input data, physical and mathematical models, discretization errors and numerical algorithms, floating-point arithmetic, and compiler and library issues. Hence, it is sometimes hard to determine whether or not there is bug in the program when its output is inexact.
- *Program input and output are extremely complex.* The input complexity causes many difficulties in test case development and generation. The following factors contribute to the complexity. First, there are in general a large quantity of input parameters. For instance, MISA-SCD has 30 parameters (see Appendix A) and ATHENA has 132 parameters. Second, some parameters may be very complex in data structure. For instance, ANT-MOC takes a 3D model (just like the model in Fig. 1) as its input and MISA-SCD uses a file to describe composition of a metal material. Third, some parameters are domain specific so that their meanings, ranges, and internal constraints may be unclear to testers. Besides, an HPNSP may output a huge amount of data. For instance, MISA-MD outputs a file for each time step, which contains the information of all atoms at the specific time. In one test, there were 40 time steps and 2×80^3 atoms. Consequently, MISA-MD produced over 4×10^7 data points and 3GB data as its output. The output complexity causes many difficulties in automated result assertion.

3 TESTING PROCESS

When we first joint the project, all the HPNSPs were under development. If domain experts and developers believed that *a program was ready for testing*, then they would share their program with us. Afterwards, we began to design a test plan in collaboration with them. We collected and tried possible testing approaches, test inputs, and test oracles from literature and other members in our R&D team. Then we tested the programs; domain experts and developers also tested the programs by themselves. If a program failed any test, then its developers were notified. After the developers fixed their program, we continued testing the fixed program.

3.1 Testing Approaches

To test the five units of analysis, we first collected the testing approaches that may be used by reviewing the literature on software testing and numerical simulation and by asking the domain experts, the numerical program developers, and the HPC developers in our R&D team *how did you test your program*. Supposing that the program under test (PUT) is denoted as p, we found the following five approaches that may be applicable in our context.

- **Classical Testing** (CT) refers to the standard testing approach in which p is fed with a certain input and the actual output is checked by comparing it with an expected result. To apply CT, we must collect adequate test cases, each of which contains a program input and an expected result.
- **Differential Testing** (DT) tests p by comparing the output of p with anther program p' that is comparable with p (or p' is another implementation of p). For a certain test input i, if $p(i) \neq p'(i)$, then a bug is detected. Although the term *differential testing* was proposed by [24, 39], the basic idea had already been widely applied for decades in HPC community who often verify the parallel implementation by comparing the sequential program. DT is also frequently used in the numerical simulation community, in which PUT is often compared with an existing program that is considered well tested. The key to the application of DT is the availability of a well-tested comparable program.
- **Property-based Testing** (PT) refers to the popular testing approach that checks whether the actual output of a single test case i satisfies a necessary property U of PUT, i.e., checking $U(i, p(i)) = true$, rather than directly comparing the actual output with the expected. The key to the application of PT is to define a set of high-quality program properties. The properties may be defined according to the theory and the model of the natural system (e.g., Newton's laws), and the expected behavior of the numerical algorithm and parallelization strategy. For instance, *in an isolated physical system, the final kinetic energy is equal to its initial kinetic energy.*
- **Metamorphic Testing** (MT) is a promising testing approach to test case generation and alleviating oracle problem [57, 62]. It has been applied to the testing of numerical programs [11, 19, 20, 64, 65] since it was proposed. The basic idea of MT is to check the satisfaction of a metamorphic relation (MR) R between a group of source test cases $i_1, i_2, ..., i_k$, a group of follow-up test cases $i_{k+1}, i_{k+2}, ..., i_n$ that are generated from the source test cases, and their corresponding output $p(i_j)$ [13], i.e., checking $R\big(i_1, ..., i_n, p(i_1), p(i_2), ..., p(i_n)\big) = true$. For instance, for a program that computes the acceleration of an object from a given force (i.e., $p(F, m) = F/m$), a possible MR is $F_1 < F_2 \Rightarrow p(F_1, m) < p(F_2, m)$; From a source test case (F_1, m), according to this MR, we are able to generate many follow-up test cases by increasing F_1. Similar to PT, MT requires high quality MRs to be used as test oracles.
- **Richardson's Extrapolation** (RE) is used to verify the convergence of numerical programs [51]. Mathematically, if some desired quantity A can be computed by an approximation $A(h)$, where h is a step size, k is a known constant, and K is an unknown constant, then $A = A(h) + Kh^k + O(h^{k+1})$ holds. If we ignore the higher order term $O(h^{k+1})$, then for h, $h/2$, and $h/4$, it is not difficult to prove that
$$\frac{A(h) - A(h/2)}{A(h/2) - A(h/4)} = \frac{Kh^k - K(h/2)^k}{K(h/2)^k - K(h/4)^k} = 2^k$$
If the program $p(h)$ correctly implemented the mathematical model $A(h)$, then it must satisfy
$$\frac{p(h) - p(h/2)}{p(h/2) - p(h/4)} \approx 2^k \tag{1}$$
RE is a special case of MT and is applicable only if the program involves a step size h and the constant k is known.

We believe that a *test case* used in CT is a pair of program input and expected output. For DT, PT, MT, and RE, we also need test cases to execute PUT. In fact, we only used the inputs that are defined in the test cases, because the comparable program (in CT), the properties (in PT), MRs (in MT), and Eq. (1) (in RE) play the role of test oracle. In practice, we both reused the test cases developed for CT, and defined/generated new test cases/inputs if possible. For simplicity, when discussing DT, PT, MT, and RE, we do not differentiate the term *test case* and *test input.*

3.2 Overall Results

The general information of the approach application and bugs found is presented in Table 1. We successfully applied four testing approaches to ANT-MOC, MISA-RT, and ATHENA; we applied three approaches to MISA-MD; and we only applied two approaches to MISA-SCD. Hence, each unit was tested using multiple approaches to maximize the possibility of bug detection. Please notice that we worked in a real-world context. We are generally not allowed to freely explore all possible testing approaches and tools due to the limited resource and the time pressure. In practice, we tended to spend more time on the approaches that seemed to be more applicable and effective when testing a program.

CT, DT, and PT were applied to four units, while MT was applied to three and RE was applied to two units. This reflects the usability and applicability of each testing approach. Regarding the possibility of finding bugs, PT and MT revealed bugs in all the units they are applied to; CT seems less likely to detect bugs in our experience.

Table 1 also shows the number of bugs that were found during the testing process. A full list of bugs is available online[1]. A bug in this paper is not a single line of wrong code. Instead, a bug may involve many lines of code that cause the failure of a test. The bugs were counted by the developers who fixed the programs. Presently, we believe that all the bugs were severe because they resulted in incorrect simulation results. It will be the future work to investigate a severity classification for HPNSPs.

In total, we have found 33 bugs, including 7 pre-processing bugs and 24 computation bugs, as shown in Fig. 2. There are also some hidden bugs that fail the test but have not been located. ATHENA has more bugs than others. It is because developers must understand the code of FRAPCON first, and then translate it into the equivalent

[1]Please visit https://github.com/ustbsoftlang/issta2020-hpnsp for the details of all PUTs, test cases/inputs, properties, metamorphic relations, and bugs.

Table 1: Information of approach application and bugs

Unit	CT	DT	PT	MT	RE	Total
ANT-MOC	0	1	6	2+	-	9+
MISA-MD	-	0	8	5	-	9
MISA-SCD	?	-	3	-	-	3?
MISA-RT	0	1	-	1	+	1+
ATHENA	3	7+	1+	-	+	11+
Sum	3?	9+	18+	8+	+	33+?

Notation. *- means that the corresponding testing approach was not applied to the program. Otherwise, the cell tells the bugs found by using the approach: A number indicates how many bugs were already found, and 0 means the program did not fail the tests when an approach was applied; + means that there were some known unrevealed bugs that made the program fail the tests; ? means that we cannot determine whether the failure of the tests was caused by a bug (e.g., it may be attributed to an error in the physical model).*
Note. *The total number of bugs in the last column may not be the sum of the previous columns because a bug may be revealed by multiple approaches.*

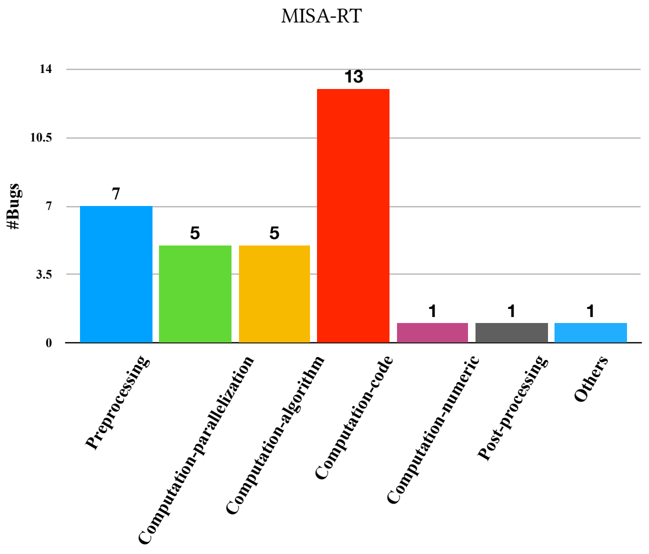


Figure 2: Bug distribution

Table 2: Numbers of test cases (#TC), properties (#Pr), metamorphic test groups (#MTG) and metamorphic relations (#MR) used in our practice

Unit	CT	DT	PT		MT			RE
	#TC	#TC	#TC	#Pr	#MTG	#TC	#MR	#TC
ANT-MOC	6	17	8	5	8	90	3	-
MISA-MD	-	11	4	7	4	17	6	-
MISA-SCD	4	-	4	3	-	-	-	-
MISA-RT	1	3	-	-	1	1000	1	6
ATHENA	1	4	4	2	-	-	-	5

C++ code to develop ATHENA. Many bugs were introduced due to the misunderstanding of the original code.

For all units except ATHENA, PT and MT are more effective in bug detection than CT and DT. DT is very effective for ATHENA because ATHENA is a rewrite of FRAPCON so that we can apply very fine-grained DT. We will discuss this case in Section 3.3.

Adequate test cases/inputs are essential to systematic testing. To test a program, we developed and applied test cases/inputs as many as possible by reusing existing test cases/inputs and generating new test cases/inputs (we discussed this issue in Section 4.2 in detail), under the restriction of the oracle problem, human efforts, and computation costs. Because the test case/input may contain tens of domain-specific parameters and complex geometry model, as discussed in Section 2.2, *it is infeasible to present and explain the definition of any concrete test case* in the paper. Please refer to our online data[1] for more information.

Table 2 shows the numbers of test cases/inputs (#TC), properties (#Pr), metamorphic test groups[2] (#MTG), and metamorphic relations (#MR) that were used in the testing process. For all units except MISA-RT, we can only create a small number of test cases. Specifically, in the process of CT, we created a very limited amount of test cases because of the complexity of program inputs and the difficulty in obtaining test oracles; in the process of DT, we used more test cases, but double efforts might be required to provide test cases for both PUT and its comparable program; MT allows us to generate follow-up test cases (inputs only) automatically, and the major restriction is the computation cost—performing more test cases requires more computation time that may not be affordable.

Table 2 also shows the number of properties and metamorphic relations used. We defined 17 properties for 4 units and 10 MRs for 3 programs. Compared with metamorphic relations, it is easier for us to define program properties. The reason is twofold: (1) high-quality metamorphic relations must be derived from physical and mathematical models, but it was beyond our capability; (2) it is more difficult for domain experts and HPC developers to understand the idea of metamorphic testing, so they did not provide much help.

3.3 Application of Testing Approaches

This section presents our practice of testing the five units of analysis with the five testing approaches.

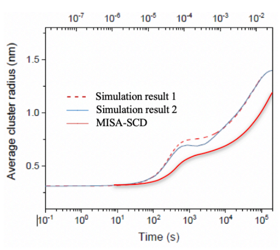
Application of Classical Testing. We applied CT to the testing of ANT-MOC, MISA-SCD, MISA-RT, and ATHENA. To perform CT, we need test cases that have both inputs and expected outputs.

Due to the input/output complexity, we cannot construct test cases, especially test oracles, from scratch. Domain experts advised that we could search test cases from domain benchmark problems, literature, and existing programs.

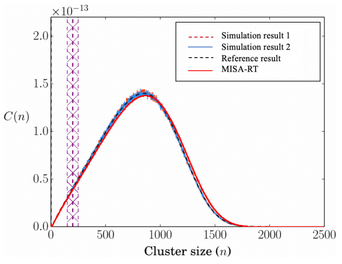
- For ANT-MOC, we found six benchmark problems [22, 32, 63] in the domain of neutral particle transport and nine test cases from openMOC. We developed six test cases from the domain problems, where the expected outputs were formulated by k_{eff}[3] values. We did not reuse the test cases of openMOC because they are either (the simplified version of) the benchmark problems or lacking oracles.
- For MISA-SCD, we found two test cases from literature [49], where the expected results were presented as diagrams. We

[2]A metamorphic test group is comprised of a set of source test cases and the follow-up test cases that are derived from the source test cases.

[3]k_{eff} is a statistical value calculated from the program output, which is an indicator of the reaction rate. The higher k_{eff} is, the faster the reaction is.



(a) Actual output of MISA-SCD



(b) Actual output of MISA-RT

Figure 3: Checking actual outputs using diagrams
The solid red curves are the outputs of MISA-SCD (Fig. 3a) and MISA-RT (Fig. 3b); other curves are expected results copied from literature.

also derived two new test cases by densifying the mesh and assuming the outputs to be closer to the expected results.

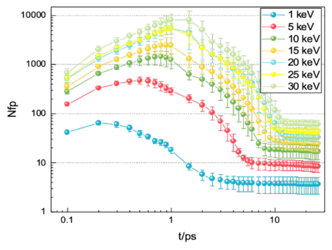
- For MISA-RT, we found one test case from literature [3] whose expected result was also a diagram.
- For ATHENA, we did not find any public test case, so the domain experts created one from their experimental data.

Then, we applied CT and compared the actual outputs with the expected as follows.

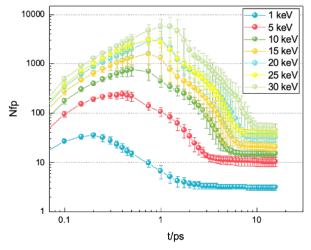
- For ANT-MOC, because of the fact that numerical simulation is by nature inaccurate, the actual outputs are not exactly equal to the expected results. For instance, the expected k_{eff} values of three test cases are 1.128, 1.077, and 1.143, while the actual values are 1.125, 1.074, and 1.141, where the relative errors are less than 0.5%. Both domain experts and developers agreed that the errors are tolerable.
- For MISA-SCD and MISA-RT, because the expected results were defined as diagrams and numeric results are not provided in corresponding literature, we cannot check the program outputs directly. To solve the issue, we drew new curves based on the actual outputs upon the original diagrams, and checked whether the new curves were consistent with the existing ones. Fig. 3 shows two examples of MISA-SCD and MISA-RT, respectively. Fig. 3a shows that the new curve deviates from the expected results of MISA-SCD. However, we cannot judge whether there was a bug. It is because that MISA-SCD adopted a computation method that is different from the method used previously. The deviation might also result from this difference. Fig. 3b shows the curve of the actual output of MISA-RT perfectly fitted the expectation.
- For ATHENA, the major barrier is that the expected output (i.e., the experimental result) is confidential such that it is not shareable. We must send the actual output to domain experts for result checking. Based on domain experts' feedbacks, we found three bugs.

Application of Differential Testing. We applied DT to ANT-MOC, MISA-MD, MISA-RT, and ATHENA. To perform DT, we need a program comparable to PUT and some test cases.

- When applying DT to ANT-MOC, we compared the k_{eff} values of ANT-MOC and openMOC with 17 test cases, including 4 test cases used in CT and 13 derived test cases. We did not

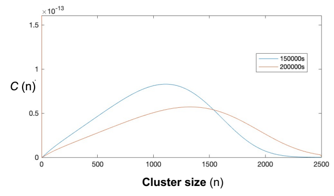


(a) Partial results of MISA-MD

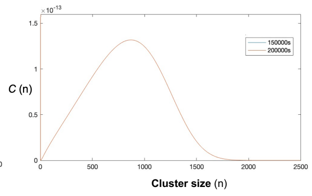


(b) Partial results of LAMMPS

Figure 4: Differential testing of MISA-MD



(a) Results of MISA-RT



(b) Results of Matlab program

Figure 5: Differential testing of MISA-RT
The two curves in Fig. 5b coincided, while the curves in Fig. 5a did not. This implies that MISA-RT did not reach a fixed pointed as the Matlab program did after the simulation time of 150000 s.

use/create more test cases because for any additional test case, we must prepare two versions for ANT-MOC and openMOC due to their difference of input formats, respectively. For most test cases, the relative error between ANT-MOC and openMOC is very small ($< 0.01\%$ on average). There is only one exception, where the difference is 0.21%. From our view, 0.21% is also a small error, but domain experts regarded that the difference is large. Finally, we found a bug of the characteristic line generation in the preprocessing module. After the bug was fixed, the error dropped to 0.042%. As mentioned above, ANT-MOC supports new geometrical models that cannot be handled by openMOC. DT did not cover those problems.
- To apply DT to MISA-MD, we compared MISA-MD and LAMMPS with 11 test cases (containing inputs only) that were developed by ourselves. Because the programs are stochastic, we ran each test multiple times to obtain an average output. Due to the output complexity, as mentioned in Sect. 2.2, it is infeasible to check the outputs of MISA-MD and LAMMPS directly. Hence, we visualized the outputs as figures and animations, and then compared the visual results. For instance, Fig. 4 shows some visual results of MISA-MD and LAMMPS. Obviously, there are significant differences between Fig. 4a and Fig. 4b. However, domain experts regarded that the two figures are similar in their trends and shapes. The divergence was attributed to the different computation methods adopted by MISA-MD and LAMMPS. Finally, MISA-MD passed the tests.

- For MISA-RT, no open-source comparable program existed. To apply DT, we manually developed a sequential Matlab program that realized the same numerical algorithm. We reused the single test case in CT, and the two programs produced the same result. Then, we derived two new test cases by increasing the simulation time. MISA-RT and its Matlab version produced diverse outputs as shown in Fig. 5 which helped us find one floating-point bug about cancellation.
- For ATHENA, because ATHENA is a rewrite of FRAPCON, we conducted a fine-grained differential testing—we compared the output of each function in detail with the single test case used in CT. The domain experts also created several new test cases. They performed DT themselves because the new test cases were not shareable. In this phase, we found seven bugs in total.

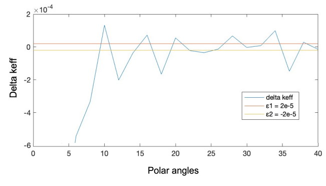
Application of Property-Based Testing. We applied PT to ANT-MOC, MISA-MD, MISA-SCD, and ATHENA. The key of PT is to identify necessary properties. We defined 17 properties (P1-P17), and a complete list is presented in Appendix B.

Obviously, it is best to define properties according to domain knowledge and physical model. For ANT-MOC, we identified one physical property *P5: the flux distribution must be geometrically symmetric*; For MISA-MD, we identified three physical properties (P9-P11), e.g., *P9: the action and the reaction force between any two molecules must be identical* and *P10: the kinetic energy of the entire system must be conserved.* We also defined a property (P12) for MISA-MD according to domain knowledge, i.e., *P12: within a small time span—two consecutive time steps, the change of the force exerted on a molecule should not be greater than ϵ, where $\epsilon = 20$.* These properties reflect the domain requirements that must be fully satisfied.

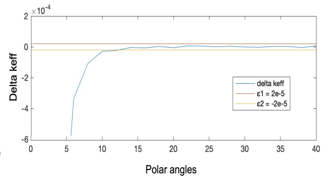
However in our practice, it is very hard for us to define the domain and physical properties due to the lack of domain and mathematical knowledge. Hence, we also attempted to find properties from other aspects with which we are more familiar, as follows.

- Four properties (P1-P4) for ANT-MOC and three properties (P6-P8) for MISA-MD were defined based on the implementation, e.g., *P1: the ID of a characteristic line is unique*, and *P7: the positions and the velocities of molecules must not be NAN.*
- Three properties (P13-P15) for MISA-SCD were identified according to the computation method with respect to the output variables Cu, V, SIA_m, and SIA_im (i.e., the numbers of Cu atoms, vacancies, mobile SIAs, and immobile SIAs), e.g., *P13:* $\neg(Cu < 0 \land V < 0 \land SIA\_m < 0 \land SIA\_im < 0)$.
- Two properties (P16-P17) for ATHENA were identified based on the numerical algorithm, i.e., *P16: the pressure of the system should not change within the inner loop*, and *P17: the result of the inner loop must converge.*

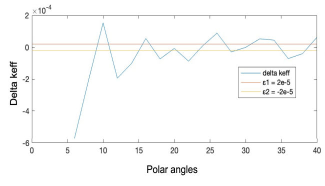
We reused the test cases/inputs used in CT/DT to test our programs with the 17 properties. These properties helped us find many bugs, i.e., 6 bugs in ANT-MOC, 8 bugs in MISA-MD, 3 bugs in MISA-SCD, and > 1 bugs in ATHENA. For instance, with the help of P5, we found a bug that the equation $v' = v/d$ was wrongly realized as $v' = v/\sqrt{d}$ in MISA-MD. In ATHENA, with the help of P17, we found a bug of the incorrect variability of a variable (i.e., a global variable is declared as a local variable). There were more bugs in ATHENA that broke the properties but have not been located.



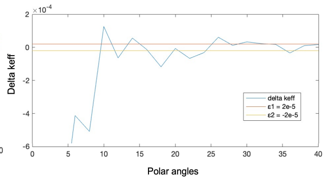
(a) Δk_{eff} of MTG 1 before fixing



(b) Δk_{eff} of MTG 1 after fixing



(c) Δk_{eff} of MTG 2 before fixing



(d) Δk_{eff} of MTG 2 after fixing

Figure 6: Metamorphic testing of ANT-MOC using MR3

Application of Metamorphic Testing. We applied MT to ANT-MOC, MISA-MD, MISA-SCD, and MISA-RT. Assume that p denotes PUT. Similar to PT, MR identification was challenging. Besides, we found that that *it is more difficult for domain experts to understand the concept of MT.* Hence, they did not help us a lot in this phase. We defined 10 metamorphic relations (MR1-MR10), and a complete list is presented in Appendix C. To perform MT, we reused existing test cases as the source test cases and derived more follow-up test cases according to MRs.

- For ANT-MOC, supposing that the k_{eff} value of $p(i)$ is denoted as $k_{eff}(p(i))$, we identified three MRs (MR1-MR3), e.g., *MR1: for any two test case i and i', if i is identical to i' except that i is performed on a single physical core and i' is performed on multiple physical cores, then $k_{eff}(p(i)) = k_{eff}(p(i'))$*, and *MR3: for a source test case i_0, if follow-up test cases i_1, i_2,... are derived by gradually increasing the amount of polar angles, then $|k_{eff}(p(i_k)) - k_{eff}(p(i_{k+1}))| < 2 \times 10^{-5}$* when $k \longrightarrow +\infty$. MR1 is based on the fact that the number of available CPU cores should not affect the program output; MR3 is based on a property of the computation model. ANT-MOC failed MR1 on *Tianhe-2* supercomputer [2] due to the incompatibility of the Intel ICPC compiler. ANT-MOC also failed MR3. Fig. 6a and 6c show the results of two metamorphic test groups. Obviously, $k_{eff}(p(i_k)) - k_{eff}(p(i_{k+1}))$ did not fall below 2×10^{-5}. We found and fixed one bug in the load balance module. After bug fixing, one metamorphic test group satisfied this MR (Fig. 6b), but the other (Fig. 6d) did not yet. We are still finding the problem now.
- For MISA-MD, we defined six metamorphic relations (MR4-MR9) according to the physical models. For instance, *MR6: if two test cases i and i' differ in the amount of Ni atoms only, the final temperatures of $p(i)$ and $p(i')$ must be equal.* MISA-MD failed this MR due to a typo in string constants, where "Ni" was written as "Nii".
- For MISA-RT, we encountered some difficulties during MR identification due to the lack of the domain knowledge, although the program is small. Domain experts only told us

that after a certain time, the physical process will reach a fixed point in reality. We defined an initial MR10 that *if the simulation times of the source and the follow-up test cases i and i' are larger than a certain T_0, $p(i) = p(i')$*. However, MR10 is infeasible because neither domain experts nor we know the value of T_0. To solve this problem, we introduced regression analysis as follows. Given a set of test cases $i_0, i_1, ..., i_n$ that only differ in simulation time, find a best fitted function g from $p(i_0), p(i_1), ..., p(i_n)$ and g must be a function of simulation time. Afterwards, we solve $\mathrm{d}g(t)/\mathrm{d}t = 0$, and the solution is T_0. MISA-RT did not pass MR10 due to the same bug found by DT.

Application of Richardson's Extrapolation. One of the major challenges is the applicability of RE. To avoid misuse, we consulted an expert of mathematics in our university, who identified that RE is applicable to MISA-RT and ATHENA. But for other units, we did not know whether or not RE can be applied due to their complexity.

For MISA-RT, the expert estimated that $\frac{p(h)-p(h/2)}{p(h/2)-p(h/4)} \approx 16$, where h is the length of the time step. However, the actual result was far away from 16—it did not converge. For ATHENA, the variable h refers to the number of grids, but we did not collect enough results to calculate Eq. (1) before ATHENA crashed. We found that when the number of grids increased over a certain value, some calculations in ATHENA threw expected exceptions. We are still working on bug localization.

RE indicated the presence of bugs in MISA-RT and ATHENA. Nevertheless, it did not provide us with useful information to locate bugs. Eq. (1) is a very strict property, so any tiny error in the computation may cause its violation. We as yet do not know the root causes of the violation of Eq. (1) in MISA-RT and ATHENA.

4 ANSWERS TO RESEARCH QUESTIONS

4.1 Answer to RQ.1

Based on our experience described in Sect. 3, our answer to **RQ.1** is that CT, DT, PT and MT are relatively easy to be applied, and PT and MT are more effective in bug detection. The virtues and weaknesses of each testing approach are summarized as follows.

CT is the easiest approach that can be learned and applied, especially by the experts outside software engineering. However, its prerequisite that there are sufficient test cases with oracles is not always satisfied. In several cases (e.g., MISA-RT and ATHENA), we could only obtain one test case. The lack of test cases severely affects the usability and effectiveness of CT.

DT is also an easy to understand approach. It can be applied only if a well verified and comparable program is available. However, such a comparable program does not always exist. For instance, MISA-SCD does not have an available comparable program; openMOC is only partially comparable with ANT-MOC so we cannot use DT to test the new feature of ANT-MOC.

PT and MT alleviate the oracle problems and do not require a comparable program. The major challenge of PT and MT is the identification of program properties and metamorphic relations. This requires a good comprehension of domain theories and knowledge, discretization models, numerical methods, and parallelization strategies. Domain experts, numerical analysis experts, and HPC developers are highly welcome to join this process. Unfortunately, the concepts of PT and MT (especially for MT) are not very popular outside software engineering community. Extra communication and training costs are required to involve other experts.

RE is very sensitive to bugs. However, RE requires that the mathematical model of the program under test involves a step size h and that the order k of the truncate item must be known. Hence, RE is generally not applicable to all HPNSPs. Besides, it does not provide helpful information for bug localization.

Based on our experience and Table 1, we regard that in general PT and MT (and RE is a special case of MT) are more effective than CT and DT, because most bugs were found by PT and MT. The reason why PT and MT are more effective is likely that the properties and the metamorphic relations focus on the expected behaviors of PUT, rather than the program outputs that are by nature inaccurate as discussed in Section 2.2. Properties and relations are more suitable for being test oracles of numerical simulation programs. PT and MT are not comparable because they usually can reveal bugs that were overlooked by each other. To maximize the success rate of bug detection, multiple testing approaches must be applied.

Within the five units, ATHENA is an exceptional case in which DT found most bugs. Because ATHENA is a rewrite of FRAPCON, we are able to compare every function in detail. However, as discussed above, such a comparable program is not always available.

In our practice, we tried several software testing and analysis techniques and tools.Metamorphic testing, as an example of innovative testing techniques, has been successfully applied. However, the application was not plain sailing. As described in Sect. 3.3, we proposed MR10 that is initially infeasible due to an unknown hyperparameter T_0. Besides, to test ANT-MOC, we id a MR that *if we input a mirror of the geometric model, then the final result must be symmetric*. The MR was frequently used in literature [19], but was finally proved invalid because ANT-MOC rejects asymmetric geometric model.

We also noticed that many techniques are applicable in theory but infeasible in practice. First, most existing techniques target at C/C++/Java, however Matlab and Fortran are also frequently used in this field. Second, HPNSPs may be run on a supercomputer (e.g., Sunway TaihuLight [29]) that may have a very special hardware/software environment, where most testing tools cannot be used. Third, due to the long execution time, many tools become unscalable. For instance, we tried fpDebug [7], a dynamical floating-point error analysis tool. When fpDebug is turned on, the execution time becomes around 400-500 times slower than that of the original program. Because our programs usually require hours and days to return a result, such dynamic analysis tools cannot be used.

4.2 Answer to RQ.2

Based on our experience, our answer to **RQ.2** is as follows: (1) We can find existing test cases/inputs from *domain benchmark problems*, *existing programs*, *literature*, and *experimental data*, though the reusable test cases may be very limited (22 test cases in our practice); (2) Generating new test inputs is possible (but not easy). The sources of test cases and their barriers are summarized as follows.

- *Domain benchmark problems.* Some domains may have defined several benchmark problems (e.g., BEAVRS [32] for

ANT-MOC). However, there are two major barriers to creating test cases from benchmark problems. First, some parameters of the program are described in natural languages, rather than in machine-readable formats. For example, BEAVRS describes that *the reactor core is a pin-by-pin model owning 193 components each of which has* 17×17 *pins*. We exerted CAD software and drew this model manually, which is very time consuming. Second, the problem may not cover all program inputs. Consequently, we must estimate reasonable values for the unspecified parameters.

- *Existing programs.* When there are similar programs, their test cases may also be reusable. Ideally, these test cases can be used to test PUT directly. However in practice, it is very difficult to do so due to the following reasons. (1) The input format of PUT may be very different from that of the comparable program. For instance, the test cases of openMOC were specified using Python scripts, while ANT-MOC adopts a XML-based input format. If we want to reuse test cases of openMOC, then we must manually convert them into XML files. (2) PUT may require some parameters that need to be calculated from the parameters of the comparable program. For instance, MISA-MD requires two parameters e_{pka} and (i, j, k) that must be calculated from the parameters v_x, v_y, v_z of LAMMPS by solving the equation group:

$$\begin{cases} 0.5 \times m_{fe} \times (v_i^2 + v_j^2 + v_k^2) = e2j \times e_{pka} \\ v_i : v_j : v_k = i : k : j \\ v_x = v_i/100, v_y = v_j/100, v_z = v_k/100 \end{cases}$$

where $e2j = 1.60218 \times 10^{-19}$ and $m_{fe} = 9.288 \times 10^{-26}$.
- *Literature.* We used the literature data to test MISA-RT and MISA-SCD. The literature data shares the major barriers of domain benchmark problems and existing programs.
- *Experimental data.* The testing of ATHENA relied on the experimental data. A major barrier in our practice is that the data (and the test cases created) is not fully shareable with us. Consequently, the testing of ATHENA is partially out of our control. Besides, according to the feedback from domain experts, extracting test inputs and expected outputs from the raw data requires tremendous human efforts.

Although we did generate some test inputs in our practice, it is worthwhile emphasizing that test input generation for HPNSPs is not as easy as generating random numbers. First, as discussed in Section 2.2, the input of an HPNSP may be a very complex data structure, e.g., the CAD file of ANT-MOC and the material file of alloy of MISA-SCD. Randomly generating these inputs are challenging. Second, a parameter may have domain specific semantics. In this case, we do not know its range and its impact on computation. Third, some parameters may be correlated under a domain constraint. For instance, MISA-SCD has a parameter that refers to the Cu content in the alloy. If it is increased to a certain threshold value, then the alloy may become another sort and many other parameters must also be altered. Due to these difficulties, we cannot freely generate new test inputs. In practice, we created new test inputs by altering a small subset of the parameter values in the existing ones. Consequently, test adequacy cannot be assured.

4.3 Answer to RQ.3

As an answer to **RQ.3**, we summarized the kinds of test oracles that were used in our practice and their problems as follows.

- *Expected numerical values.* The domain benchmark problems of ANT-MOC include expected numerical values as test oracles. However, these values seldom work because of the inaccuracy and randomness of numerical simulation.
- *Comparable programs.* When DT is applied, the comparable program is used as the test oracle. However, PUT and its comparable program may produce different results if they realized diverse algorithms. On the other hand, the result of the comparable program may also be wrong. These two issues make many difficulties in judging test outputs.
- *Visual results.* The expected outputs may also be depicted as diagrams, images, and even animations. Visual comparison is inevitable in the testing of HPNSPs. First, some literature may only provide the visual results (e.g., MISA-SCD and MISA-RT). Second, due to the output complexity, we may only be able to check the visual results (e.g., MISA-MD). Visual comparison requires extra post-processing efforts. Besides, it also encounters some difficulties in automation, e.g., how to automatically assert that two curves are similar in their trends and shapes could be very challenging.
- *Oracle approximation.* Since the program outputs are by nature inaccurate, an acceptable result range (i.e., oracle approximation [40]) may be used. However, how to determine the bound of the result is another form of oracle problems.
- *Domain experts' opinion.* Domain experts also played a role of test oracles. In the case of ATHENA, a sharp-eyed expert precisely pointed out a tiny error in an 23000-line output file, and helped us find a bug in computation module. However, experts' opinion is not always reliable. When testing MISA-RT, domain experts did not regard the violation of MR10 is due to a bug until we successfully fixed it. Besides, experts may have inconsistent opinions about the same result.
- *Necessary properties.* PT and MT adopt necessary program properties as test oracles. The better we understand the program and the domain knowledge, the better we can define effective properties/MRs. The problem is that learning domain knowledge may be very expensive for testers. For instance, MISA-SCD intended to solve many inter-related equations, which have the form

$$\frac{dN_\theta}{dt} = \widetilde{G}_\theta - \sum_{\theta} \widetilde{R}_{\theta\theta'} N_\theta + \sum_{\theta'} \widetilde{R}_{\theta'\theta} N_{\theta'} - \sum_{\theta,\theta'} \widetilde{K}_{\theta\theta'} N_\theta N_{\theta'} + \sum_{\theta',\theta''} \widetilde{K}_{\theta'\theta''} N_{\theta'} N_{\theta''}$$

But it is too difficult for a tester to derive any necessary property from this equation.

4.4 Answer to RQ.4

Based on our practice, our answer to **RQ.4** is that the most frequently occurring bugs are coding errors (13 bugs, such as incorrect variable variability and incorrect index value/constant), and ATHENA contributed most of them (11 bugs). After that, bugs of preprocessing (7 bugs, e.g., the generation of characteristics lines), bugs of incorrect algorithm implementation (5 bugs, e.g., incorrect implementation of the calculation formula of atom velocities), and bugs of parallel computation and communication (5 bugs) are also very common. Special treatments should be proposed to effectively detect and locate these bugs. Floating-point error (e.g., cancellation) is rare. It does not mean that HPNSPs are free from floating-point errors. As mentioned in our answer to **RQ.1**, the reason may be that we did not find any scalable floating-point analysis tool.

5 DISCUSSION

5.1 Lessons Learned

We summarized several lessons learned as follows.

- **A proposal for the test strategy for HPNSPs.** Based on our experience, an HPNSP (i.e., PUT) may be tested by the following steps. **Step 1**: apply DT to compare PUT and its sequential version to find parallelization errors. The consistency between the sequential and parallel version is also very helpful for debugging, since parallel programs are much harder to be debugged. If PUT is not developed from a sequential program, then we can treat the execution of PUT that has a single parallel task as the sequential version. **Step 2**: apply PT and MT to verify PUT. Because our practice demonstrated that PT and MT are very applicable and effective, we recommend the two approaches as the primary testing approaches for HPNSPs. **Step 3**: if a comparable program p' is available, test PUT by comparing with p'. **Step 4**: validate PUT with domain benchmark problems to address domain experts' concern. The last step is essential because domain experts usually value those domain problems highly (though from the view point of software testing, those problems are generally not sufficient). **In each step**, always test PUT with small-sized test cases that represent simple and primitive physical problems first (we term it *unit data testing* that will be discussed later).
- **Domain experts must be involved in testing.** Frankly, researchers and practitioners of software testing may not be capable to test HPNSPs by themselves. The testing work needs multidisciplinary knowledge, especially the theories and knowledge of application domains. It is impossible for us to quickly manage this knowledge. We must invite domain experts to help us find/develop test inputs and oracles. Nevertheless, domain experts might view this collaboration differently. We regard domain experts as our allies of bug hunting, but they may consider us their employees—we develop and test programs for them. For this reason, domain experts may be unwilling to help us, and this human factor should not be ignored. To improve the present situation, we are trying to establish an industry standard for the testing of nuclear power plant simulation programs, in which the role and obligations of domain experts during the testing process are clearly defined. Regarding technical solutions, we assume that a domain specific language for the specification of domain knowledge should be developed for efficient knowledge transfer across multiple domains.
- **The use of PUT influences the testing.** In our units of analysis, MISA-SCD and MISA-RT are developed based on new domain theories and models. They will be used to research the theories and the models, rather than making a prediction of the real world. For such programs, it is more difficult to define test oracles because the properties of the theories and models may be unknown yet. Test engineers must be aware of the use of PUT, and discuss the correctness criterion with domain experts before testing.
- **Preserve and manage the actual test outputs seriously.** Reusing, rather than re-computing, the outputs of an HPNSP will save considerable time and resources. The preserved output data may be used in the following ways. First, when PUT is tested using multiple approaches with the same test case/input, the preserved output can be reused. Second, if the actual outputs of two versions of PUT are preserved, then we can compare them to identify the impacts of the code changes, since a change to HPNSP does not always improve the accuracy. Third, the preserved output may be used for future examination when new result checking methods are available. For instance, ANT-MOC were checked mainly by the metric k_{eff} for simplicity, and it is our ongoing work to check the outputs by computing *flux distribution*, where the outputs can be reused.

5.2 Future Research Suggestions

We must make more research efforts to solve the open issues occurring during the testing of HPNSPs. We identify several potential research directions as follows.

- **Dealing with multi-level errors.** The accuracy of the output of an HPNSP is affected by many factors, e.g, errors in physical/mathematical models, errors in discretization, floating-point errors, parallelization errors, and errors in the input data. When an HPNSP produces an inaccurate output, it is very important to locate the root error to know who is responsible for the incorrect result. Nevertheless, how to analyze multi-level errors is a fresh new topic that requires further investigation.
- **Adapt existing testing techniques and tools for high performance computers and programs.** Many existing software testing techniques and tools are inapplicable to HPNSPs due to scalability issue (i.e., high overhead), the issue of process/thread-unsafe, portability issue (e.g., they cannot be run on a supercomputer), and incompatibility of programming languages (e.g., they are only suitable for C/C++ but not Fortran). There is an urgent need to adapt existing techniques and tools for high performance computing environments and programs. However, doing so is non-trivial. For example, while existing dynamic floating-point error analysis tools [7, 70] usually require >400x longer execution time to conduct the dynamic analysis. Minimizing the overheads of these tools is very challenging.
- **Benchmark development and assessment.** Benchmark problems are intensively used in the community of numerical simulation. However, how to develop and assess benchmark problems is an open issue. Benchmark development requires sufficient domain knowledge, so domain experts are more capable of solving this issue. For benchmark assessment,

mutation analysis may be helpful. However, it is required to define a set of mutation operators suitable for HNPSPs.

- **Unit data testing.** We observed that a complex test case can be split or simplified into several simpler test cases. For instance, a test case for a full reactor can be simplified into a test case for a single fuel component. Testing with simplest test cases that represent primitive domain problems is termed *unit data testing* in this paper. How to extract/generate simple test cases from a complex one is an interesting topic.
- **New coverage criterion is required.** Statement coverage is a common indicator of test adequacy. Nevertheless, it may not be useful for testing HPNSPs. For instance, when MISA-RT was tested, just one test case resulted in 100% statement coverage, but we cannot say the testing with one test case is sufficient. As explained in Section 2.2, the computation module of an HPNSP is usually a loop. Statements in the loop are very likely to be covered during a single execution. New coverage criterion for HPNSPs must be defined to better manage the test progress.
- **Test case migration.** In the context of differential testing, there is an urgent need for test case migration. Preparing two versions of the same test case is time consuming and error-prone when PUT and its comparable program have distinct input parameters and formats.

5.3 Threats to Validity

This paper is based our real experience of testing high performance numerical simulation programs. Due to many real-life restrictions, the paper suffers from the following threats to validity.

- **Internal validity.** A major threat to internal validity is that we may not properly exert the testing approaches due to the limitation of our skill and knowledge. Our conclusion about the approach usability and effectiveness may be affected by this factor. To mitigate this problem, we consulted many famous testing experts of metamorphic testing, mutation testing, and floating-point testing.
- **External Validity.** The major threat to external validity is the limited amount of samples (e.g., programs, test cases, and bugs). It will be our future work to enhance our study when more programs in our team are ready for testing.
- **Construct Validity.** This paper uses the number of bugs to evaluate the effectiveness of a testing approach. There are many other factors that may contribute to the evaluation of the effectiveness. Nevertheless, we believe the number of bugs can be used for qualitative analysis.
- **Reliability.** The presented data might be differently interpreted by others. To mitigate this issue, all the interpretations were thoroughly discussed within the development and testing team.

6 RELATED WORK

High performance numerical simulation programs are non-testable. To the best of our knowledge, we are among the first to report real-world experience of testing HPNSPs from the perspective of software testing. Metamorphic testing has been demonstrated to be effective for testing scientific programs, including partial differential equation solvers, Monte Carlo simulation, bioinformatics programs, and learning and AI systems. Most of these efforts [11, 12, 19, 20, 25, 64, 65] described how to apply MT to and proposed some MRs for the programs under test. There are also some efforts discussing how to identify and infer MRs [13, 35, 69]. However, existing work mainly focused on relatively simple numerical functions, rather than large scale simulation programs.

Floating point analysis attracted much attention [27] during the past decade. Various approaches have been proposed to detect and fix floating point errors [7, 16, 38, 44, 46, 55, 67, 68, 70, 71] and floating-point exceptions [5]. The result of error analysis was also combined with precision turning [15, 33, 42, 53, 54]. The major barrier to their application is the scalability and robustness.

Testing numerical programs is also discussed in the community of numerical simulation. Some efforts focused on reporting the test results of certain programs [23, 31]. Others proposed approaches and tools [51, 52, 60] for verifying numerical programs. Due to the complexity of testing, the numerical simulation community tends to treat errors as an intrinsic characteristic of numerical programs and manage them by uncertainty quantification [34]. These efforts have formed a new research branch of numerical analysis that is independent from software testing. We assume that the two communities should have more collaborations and exchange more experience to improve the testing of numerical programs.

7 CONCLUSION

To the best of our knowledge, this paper is among the first to report real-world experience of testing high performance numerical simulation programs from the perspective of software testing. We presented how we applied software testing approaches to five programs in detail. Although we managed to find several bugs for each program under test, we encountered many challenges in test case and oracle development, test automation, and reducing test costs.

We believe that a huge amount of research and engineering efforts are required to make existing testing approaches applicable and more effective to the real-world and complex programs. We also suggest that researchers and practitioners of software testing should have more communication with experts from HPC community and numerical simulation community to share new research achievements with them. We regard this paper as a bridge that brings systematic software testing to numerical simulation.

ACKNOWLEDGMENTS

This work is supported by National Key R&D Program of China (No. 2017YFB0202303) and Beijing Natural Science Foundation (No. 4192036). We thank Prof. T.Y. Chen from Swinburne University, Prof. Xiaoyuan Xie from Wuhan University, Dr. Zhanwei Hui from PLA Academy of Military Science, Prof. Mingyue Jiang from Zhejiang Sci-Tech University, Prof. Xiaoyin Wang from University of Texas at San Antonio, and Prof. Chang-Jun Hu and Prof. Chang-Ai Sun from University of Science and Technology Beijing for their valuable suggestions on this work during the past two years. We also thank the HPC developers from University of Science and Technology Beijing and the domain experts from China Institute of Atomic Energy for helping us collect and analyze data. Finally, we appreciate the constructive review comments from the anonymous reviewers.

A PARAMETERS OF MISA-SCD

The following table that is provided by the developers of MISA-SCD lists all the input parameters of MISA-SCD to demonstrate the input complexity of HPNSPs. The table shows that the input of MISA-SCD is complex in terms of the data format, the amount, and the semantics of parameters. Please refer to https://github.com/ustbsoftlang/issta2020-hpnsp for more detailed information about the input and output of each HPNSP.

Name	Meaning
defectFile	The defect attributes file.
meshFile	The mesh file.
irradiationType	Type of irradiation ('Cascade' for neutron irradiation, 'FrenkelPair' for electron irradiation, 'None' for no irradiation).
implantScheme	Toggle between Monte Carlo defect implantation and explicit defect implantation.
cascadeFile	The cascade defects file.
implantType	Where uniformly implanting defects.
implantFile	The data file containing non-uniform implantation profile.
grainBoundaries	Toggle whether we are going to include the effect of grain boundaries.
pointDefect	Toggle whether point defects are allowed to move only.
temperature	Temperature (K)
soluteConcentration	Initial content of solute atoms (Cu) in iron.
numVac	Initial number of vacancies in the simulation system.
dpaRate	The rate of DPA.
totalDPA	Total DPA in simulation.
firr	Radiation enhanced factor for Cu.
annealTemperature	Annealing temperature (K).
annealTime	Total anneal time.
lattice	The lattice constant (nm).
burgers	Dislocation loop burgers vector (nm).
reactionRadius	Reaction distances (nm).
grainSize	Grain size (nm).
dislocDensity	Dislocation density (nm^{-2}).
cascadeVolume	Volume of cascade (nm^3).
max3D	Maximum size for SIA defect to diffuse in 3D.
numSims	Number of times to repeat simulation.
totdatToggle	Toggle whether the total defects file is output (The file contains number of ever defect species, cluster number densities, average cluster size, average radius of clusters and so on).
minSolute	The minimum size of solute clusters included in the statistics.
minVoid	The minimum size of vacancy clusters included in the statistics.
minLoop	The minimum size of SIA clusters included in the statistics.
minVS	The minimum size of S_Vac clusters included in the statistics.

B ALL PROPERTIES

The following table shows the properties that were used in PT for each program. PID stands for *Property ID*.

Program	PID	Description	Reason
ANT-MOC	P1	The ID of any characteristic line must be unique	Implementation
	P2	The ID of any characteristic line must be a positive integer	Implementation
	P3	The length of any characteristic line must be a positive integer	Implementation
	P4	The ID of FSR must be unique	Implementation
	P5	The flux distribution must be geometrically symmetric	Physics & Computation
MISA-MD	P6	The ID of any molecule must be unique	Implementation
	P7	The position and velocity of a molecule should not be NAN	Implementation
	P8	The position of a molecule must be within the simulation space	Implementation
	P9	The action and the reaction force between any two molecules must be identical	Physics
	P10	The kinetic energy of the entire system must be conserved (equal to the input PKA energy)	Physics
	P11	The number of molecules should not change	Physics
	P12	Within two consecutive time steps, the change of the force exerted on a molecule should not be greater than ϵ, where $\epsilon = 20$	Domain knowledge
MISA-SCD	P13	$\neg(Cu < 0 \wedge V < 0 \wedge SIA\_m \wedge 0 \wedge SIA\_im < 0)$	Computation
	P14	$\neg(SIA\_m > 0 \wedge SIA\_im > 0)$	Computation
	P15	$\neg((SIA\_m > 0 \vee SIA\_im > 0) \wedge V > 0)$	Computation
ATHENA	P16	The pressure of the system should not change within the inner loop	Numeric algorithm
	P17	The result of the inner loop must converge	Numeric algorithm

Many properties are related to IDs (e.g., P1, P2, and P4), because ID generation in preprocessing module usually may adopt a very complex algorithm, rather than generating a random unique integer.

C ALL METAMORPHIC RELATIONS

The following table shows the metamorphic relations that were used in MT for each program. MRID stands for *Metamorphic Relation ID*.

Program	MRID	Description	Reason
ANT-MOC	MR1	For any two test case/input i and i', if i is identical to i' except that i is performed on a single physical core and i' is performed on multiple physical cores, then $k_{eff}(p(i)) = k_{eff}(p(i'))$.	Execution
	MR2	For a source test case i, if the follow-up test case i' is derived from i by increasing the depth of a control pin, then $k_{eff}(p(i)) > k_{eff}(p(i'))$.	Physics
	MR3	For a source test case i_0, if follow-up test cases i_1, i_2,... are derived by gradually increasing the amount of polar angles, i.e., by densifying the grids, then $k_{eff}(p(i_k)) - k_{eff}(p(i_{k+1})) < 2 \times 10^{-5}$, when $k \rightarrow +\infty$.	Computation
MISA-MD	MR4	For a source test case i, if the follow-up test case i' is derived from i by increasing the initial PKA, then $T(p(i)) < T(p(i'))$, where T extracts the system temperature from the program output.	Physics
	MR5	For a source test case i, if the follow-up test case i' is derived from i by increasing the simulation space, then $T(p(i)) > T(p(i'))$.	Physics
	MR6	For a source test case i, if the follow-up test case i' is derived from i by increasing Ni content, then $T(p(i)) = T(p(i'))$.	Physics
	MR7	For a source test case i, if the follow-up test case i' is derived from i by increasing Cu content, then $T(p(i)) = T(p(i'))$.	Physics
	MR8	For a source test case i, if the follow-up test case i' is derived from i by increasing the initial PKA, then $Fs(p(i)) < Fs(p(i'))$, where Fs extracts the amount of Frankel defects from the program output.	Physics
	MR9	For a source test case i, if the follow-up test case i' is derived from i by increasing the initial PKA, then $Fp(p(i)) < Fp(p(i'))$, where Fp extracts the peak value of Frankel defects from the program output.	Physics
MISA-RT	MR10	If the simulation times of the source and the follow-up test cases i and i' are larger than a certain T_0, $p(i) = p(i')$.	Physics

REFERENCES

[1] [n.d.]. http://www.swmath.org/software/18294

[2] 2020. https://www.top500.org/featured/systems/tianhe-2/

[3] Hala Ashi. 2008. *Numerical methods for stiff systems.* Ph.D. Dissertation. University of Nottingham.

[4] Stefanie L Baker, Aravinda Munasinghe, Bibifatima Kaupbayeva, Nin Rebecca Kang, Marie Certiat, Hironobu Murata, Krzysztof Matyjaszewski, Ping Lin, Coray M Colina, and Alan J Russell. 2019. Puri Fi Cation By Tuning Protein Solubility. *Nature Communications* (2019), 1–12.

[5] Earl T. Barr, Thanh Vo, Vu Le, and Zhendong Su. 2013. Automatic detection of floating-point exceptions. *POPL: Principles of Programming Languages* (2013), 549–560.

[6] Peter Bauer, Alan Thorpe, and Gilbert Brunet. 2015. The quiet revolution of numerical weather prediction. *Nature* 525, 7567 (2015), 47–55.

[7] Florian Benz, Andreas Hildebrandt, and Sebastian Hack. 2012. A dynamic program analysis to find floating-point accuracy problems. In *Proceedings of the 33rd ACM SIGPLAN conference on Programming Language Design and Implementation - PLDI '12*, Vol. 47. ACM Press, New York, New York, USA, 453.

[8] William Boyd, Samuel Shaner, Lulu Li, Benoit Forget, and Kord Smith. 2014. The OpenMOC Method of Characteristics Neutral Particle Transport Code. *Annals of Nuclear Energy* 68 (2014), 43–52.

[9] Junjie Chen, Jiaqi Han, Peiyi Sun, Lingming Zhang, Dan Hao, and Lu Zhang. 2019. Compiler Bug Isolation via Effective Witness Test Program Generation. In *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering* (Tallinn, Estonia) *(ESEC/FSE 2019)*. Association for Computing Machinery, New York, NY, USA, 223–234.

[10] J. Chen, G. Wang, D. Hao, Y. Xiong, H. Zhang, and L. Zhang. 2019. History-Guided Configuration Diversification for Compiler Test-Program Generation. In *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. 305–316.

[11] T.Y. Chen and T.H. Tse. 2002. Metamorphic testing of programs on partial differential equations: a case study. In *Proceedings 26th Annual International Computer Software and Applications (COMPSAC'02)*. IEEE Computer Society, Washington, DC, USA, 327–333.

[12] Tsong Yueh Chen, Joshua WK Ho, Huai Liu, and Xiaoyuan Xie. 2009. An innovative approach for testing bioinformatics programs using metamorphic testing. In *BMC Bioinformatics*, Vol. 10. 24.

[13] Tsong Yueh Chen, Pak Lok Poon, and Xiaoyuan Xie. 2016. METRIC: METamorphic Relation Identification based on the Category-choice framework. *Journal of Systems and Software* 116 (2016), 177–190.

[14] Tsong Yueh Chen, Changai Sun, Guan Wang, Baohong Mu, Huai Liu, and Zhaoshun Wang. 2012. A Metamorphic Relation-Based Approach to Testing Web Services Without Oracles. *International Journal of Web Services Research* 9, 1 (2012), 51–73.

[15] Wei-Fan Chiang, Mark Baranowski, Ian Briggs, Alexey Solovyev, Ganesh Gopalakrishnan, and Zvonimir Rakamarić. 2017. Rigorous floating-point mixed-precision tuning. In *POPL 2017*. ACM Press, New York, New York, USA, 300–315.

[16] Wei-Fan Chiang, Ganesh Gopalakrishnan, Zvonimir Rakamaric, and Alexey Solovyev. 2014. Efficient search for inputs causing high floating-point errors. In *Proceedings of the 19th ACM SIGPLAN symposium on Principles and practice of parallel programming - PPoPP '14*. ACM Press, New York, New York, USA, 43–52.

[17] Anthony T Chronopoulos and Gang Wang. 1996. Traffic flow simulation through parallel processing. *Transportation Research Record* 1566, 1566 (1996), 31–38.

[18] Yifeng Cui, Kim B. Olsen, Thomas H. Jordan, Kwangyoon Lee, Jun Zhou, Patrick Small, Daniel Roten, Geoffrey Ely, Dhabaleswar K. Panda, Amit Chourasia, and et al. 2010. Scalable Earthquake Simulation on Petascale Supercomputers. In *Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis (SC '10)*. IEEE Computer Society, USA, 1–20.

[19] Junhua Ding and Xin Hua Hu. 2017. Application of metamorphic testing monitored by test adequacy in a Monte Carlo simulation program. *Software Quality Journal* 25, 3 (2017), 841–869.

[20] J. Ding, X. Li, and X. Hu. 2019. Testing Scientific Software with Invariant Relations: A Case Study. In *2019 IEEE 19th International Conference on Software Quality, Reliability and Security (QRS)*. 406–417.

[21] Aaron Y. Dunn, Laurent Capolungo, Enrique Martinez, and Mohammed Cherkaoui. 2013. Spatially resolved stochastic cluster dynamics for radiation damage evolution in nanostructured metals. *Journal of Nuclear Materials* 443, 1 (2013), 128 – 139.

[22] E E Lewis, M A Smith, N Tsoulfanidis, G Palmiotti, T A Taiwo, R N Blomquist. 2001. Benchmark specification for Deterministic 2-D/3-D MOX fuel assembly transport calculations without spatial homogenisation (C5G7 MOX).

[23] L Eca and M Hoekstra. 2013. Verification and validation for marine applications of CFD. *International shipbuilding progress* 60 (2013), 107–141.

[24] Robert B. Evans and Alberto Savoia. 2007. Differential testing: A new approach to change detection, In ESEC/FSE 2007. *6th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering, ESEC/FSE 2007*, 549–552.

[25] S. M. Yiu F. T. Chan, T. Y. Chen, S. C. Cheung, M. F. Lau. 1998. Application of Metamorphic Testing in Numerical Analysis. In *Proceedings of the IASTED International Conference on Software Engineering*. 191–197.

[26] Jun-Xuan Fan, Shu-Zhong Shen, Douglas H Erwin, Peter M Sadler, Norman MacLeod, Qiu-Ming Cheng, Xu-Dong Hou, Jiao Yang, Xiang-Dong Wang, Yue Wang, Hua Zhang, Xu Chen, Guo-Xiang Li, Yi-Chun Zhang, Yu-Kun Shi, Dong-Xun Yuan, Qing Chen, Lin-Na Zhang, Chao Li, and Ying-Ying Zhao. 2020. A high-resolution summary of Cambrian to Early Triassic marine invertebrate biodiversity. *Science* 367, 6475 (Jan 2020), 272–277.

[27] Anthony Di Franco, Hui Guo, and Cindy Rubio-gonzález. 2017. A Comprehensive Study of Real-World Numerical Bug. *ASE'17* (2017), 509–519.

[28] Haohuan Fu, Conghui He, Bingwei Chen, Zekun Yin, Zhenguo Zhang, Wenqiang Zhang, Tingjian Zhang, Wei Xue, Weiguo Liu, Wanwang Yin, and et al. 2017. 18.9-Pflops Nonlinear Earthquake Simulation on Sunway TaihuLight: Enabling Depiction of 18-Hz and 8-Meter Scenarios. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis* (Denver, Colorado) *(SC '17)*. Association for Computing Machinery, New York, NY, USA, Article 2, 12 pages.

[29] Haohuan Fu, Junfeng Liao, Jinzhe Yang, Lanning Wang, Zhenya Song, Xiaomeng Huang, Chao Yang, Wei Xue, Fangfang Liu, Fangli Qiao, Wei Zhao, Xunqiang Yin, Chaofeng Hou, Chenglong Zhang, Wei Ge, Jian Zhang, Yangang Wang, Chunbo Zhou, and Guangwen Yang. 2016. The Sunway TaihuLight supercomputer: system and applications. *Science China Information Sciences* 59, 7 (2016), 072001. https://doi.org/10.1007/s11432-016-5588-7

[30] Zhoulai Fu and Zhendong Su. 2017. Achieving high coverage for floating-point code via unconstrained programming. In *Proceedings of the 38th ACM SIGPLAN Conference on Programming Language Design and Implementation - PLDI 2017*. ACM Press, New York, New York, USA, 306–319. arXiv:1704.03394

[31] S Sara Gilani, H Montazeri, and Bje Bert Blocken. 2016. CFD simulation of stratified indoor environment in displacement ventilation : validation and sensitivity analysis. *Building and Environment* 95 (2016), 299–313.

[32] Geoffrey Gunow, Benoit Forget, and Kord Smith. 2019. Full core 3D simulation of the BEAVRS benchmark with OpenMOC. *Annals of Nuclear Energy* 134 (2019), 299–304.

[33] Hui Guo and Cindy Rubio-González. 2018. Exploiting community structure for floating-point precision tuning. *ISSTA 2018 - Proceedings of the 27th ACM SIGSOFT International Symposium on Software Testing and Analysis* 333 (2018), 333–343.

[34] Jon C Helton, Jay D Johnson, Cedric J Sallaberry, and Curtis B Storlie. 2006. Survey of sampling-based methods for uncertainty and sensitivity analysis. *Reliability Engineering & System Safety* 91, 10 (2006), 1175–1209.

[35] Upulee Kanewala, James M. Bieman, and Asa Ben-Hur. 2016. Predicting Metamorphic Relations for Testing Scientific Software: A Machine Learning Approach Using Graph Kernels. *Softw. Test. Verif. Reliab.* 26, 3 (May 2016), 245–269.

[36] Jinhan Kim, Robert Feldt, and Shin Yoo. 2019. Guiding Deep Learning System Testing Using Surprise Adequacy. In *Proceedings of the 41st International Conference on Software Engineering* (Montreal, Quebec, Canada) *(ICSE '19)*. IEEE Press, 1039–1049.

[37] Vu Le, Chengnian Sun, and Zhendong Su. 2015. Finding Deep Compiler Bugs via Guided Stochastic Program Mutation. In *Proceedings of the 2015 ACM SIGPLAN International Conference on Object-Oriented Programming, Systems, Languages, and Applications* (Pittsburgh, PA, USA) *(OOPSLA 2015)*. Association for Computing Machinery, New York, NY, USA, 386–399.

[38] Wonyeol Lee, Rahul Sharma, and Alex Aiken. 2016. Verifying bit-manipulations of floating-point. In *Proceedings of the 37th ACM SIGPLAN Conference on Programming Language Design and Implementation - PLDI 2016*. ACM Press, New York, New York, USA, 70–84.

[39] Mckeeman W M. 1998. Differential Testing for Software. *Digital Technical Journal* 10, 1 (1998), 100–107.

[40] Jinqiu Yang Mahdi Nejadgholi. 2019. A Study of Oracle Approximations in Testing Deep Learning Libraries. In *ASE'19*. IEEE/ACM, ACM.

[41] Ke Mao, Mark Harman, and Yue Jia. 2016. Sapienz: Multi-Objective Automated Testing for Android Applications. In *ISSTA'16* (Saarbrücken, Germany) *(ISSTA 2016)*. Association for Computing Machinery, New York, NY, USA, 94–105.

[42] Harshitha Menon, Michael O. Lam, Daniel Osei-Kuffuor, Markus Schordan, Scott Lloyd, Kathryn Mohror, and Jeffrey Hittinger. 2019. ADAPT: Algorithmic differentiation applied to floating-point precision tuning. *SC'19* (2019), 614–626.

[43] Nariman Mirzaei, Joshua Garcia, Hamid Bagheri, Alireza Sadeghi, and Sam Malek. 2016. Reducing Combinatorics in GUI Testing of Android Applications. In *Proceedings of the 38th International Conference on Software Engineering* (Austin, Texas) *(ICSE '16)*. Association for Computing Machinery, New York, NY, USA, 559–570.

[44] David Monniaux. 2008. The pitfalls of verifying floating-point computations. *ACM Transactions on Programming Languages and Systems* 30, 3 (may 2008), 1–41. arXiv:0701192 [cs]

[45] George Mozdzynski, Mats Hamrud, and Nils Wedi. 2015. A Partitioned Global Address Space implementation of the European Centre for Medium Range Weather Forecasts Integrated Forecasting System. *The International Journal of High Performance Computing Applications* 29, 3 (2015), 261–273. arXiv:https://doi.org/10.1177/1094342015576773

[46] Pavel Panchekha, Alex Sanchez-Stern, James R. Wilcox, and Zachary Tatlock. 2015. Automatically improving accuracy for floating point expressions. In *Proceedings of the 36th ACM SIGPLAN Conference on Programming Language Design and Implementation - PLDI 2015*. ACM Press, New York, New York, USA, 1–11. arXiv:arXiv:1603.09436

[47] Kexin Pei, Yinzhi Cao, Junfeng Yang, and Suman Jana. 2019. DeepXplore: Automated Whitebox Testing of Deep Learning Systems. *GetMobile: Mobile Comp. and Comm.* 22, 3 (Jan. 2019), 36–38.

[48] Qing Peng, Fanjiang Meng, Yizhong Yang, Chenyang Lu, Huiqiu Deng, Lumin Wang, Suvranu De, and Fei Gao. 2018. Shockwave generates 100 dislocation loops in bcc iron. *Nature Communications* 9, 1 (2018), 4880.

[49] Stephanie A. Pitts, Xianming Bai, and Yongfeng Zhang. 2017. *Light Water Reactor Sustainability Program Modeling of Cu Precipitate Contributions to Reactor Pressure Vessel Steel Microstructure Evolution and Embrittlement.*

[50] Steve Plimpton. 1995. Fast Parallel Algorithms for Short-range Molecular Dynamics. *J. Comput. Phys.* 117, 1 (March 1995), 1–19.

[51] Patrick J Roache. 1998. *Verification and Validation in Computational Science and Engineering.*

[52] Christopher J. Roy and William L. Oberkampf. 2011. A comprehensive framework for verification, validation, and uncertainty quantification in scientific computing. *Computer Methods in Applied Mechanics and Engineering* 200, 25-28 (2011), 2131–2144.

[53] Cindy Rubio-gonz, Cuong Nguyen, Hong Diep Nguyen, James Demmel, William Kahan, Koushik Sen, David H Bailey, Costin Iancu, and David Hough. 2013. Precimonious : Tuning Assistant for Floating-Point Precision Categories and

Subject Descriptors. *SC'13* (2013).

[54] Cindy Rubio-Gonzalez, Cuong Nguyen, Benjamin Mehne, Koushik Sen, James Demmel, William Kahan, Costin Iancu, Wim Lavrijsen, David H. Bailey, and David Hough. 2016. Floating-point precision tuning using blame analysis. In *ICSE'16*. 1074–1085.

[55] Alex Sanchez-Stern, Pavel Panchekha, Sorin Lerner, and Zachary Tatlock. 2018. Finding root causes of floating point error. In *Proceedings of the 39th ACM SIGPLAN Conference on Programming Language Design and Implementation - PLDI 2018*. ACM Press, New York, New York, USA, 256–269.

[56] Felix Schyboll, Uwe Jaekel, Francesco Petruccione, and Heiko Neeb. 2019. Dipolar induced spin-lattice relaxation in the myelin sheath: A molecular dynamics study. *Scientific reports* 9, 1 (2019), 14813.

[57] Sergio Segura, Gordon Fraser, Ana B. Sanchez, and Antonio Ruiz-Cortes. 2016. A Survey on Metamorphic Testing. *IEEE Transactions on Software Engineering* 42, 9 (2016), 805–824.

[58] Sergio Segura, José A. Parejo, Javier Troya, and Antonio Ruiz-Cortés. 2018. Metamorphic Testing of RESTful Web APIs. In *Proceedings of the 40th International Conference on Software Engineering* (Gothenburg, Sweden) *(ICSE '18)*. Association for Computing Machinery, New York, NY, USA, 882.

[59] Alexey Solovyev, Marek S. Baranowski, Ian Briggs, Charles Jacobsen, Zvonimir Rakamariundefined, and Ganesh Gopalakrishnan. 2018. Rigorous Estimation of Floating-Point Round-Off Errors with Symbolic Taylor Expansions. *ACM Trans. Program. Lang. Syst.* 41, 1, Article 2 (Dec. 2018), 39 pages. https://doi.org/10.1145/3230733

[60] A Stamou and Ioannis Katsiris. 2006. Verification of a CFD model for indoor airflow and heat transfer. *Building and Environment* 41, 9 (2006), 1171–1181.

[61] Ting Su, Guozhu Meng, Yuting Chen, Ke Wu, Weiming Yang, Yao Yao, Geguang Pu, Yang Liu, and Zhendong Su. 2017. Guided, Stochastic Model-Based GUI Testing of Android Apps. In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering* (Paderborn, Germany) *(ESEC/FSE 2017)*. Association for Computing Machinery, New York, NY, USA, 245–256.

[62] S M Yiu T. Y. Chen S. C. Cheung. 1998. *Metamorphic testing: a new approach for generating next test cases*. Technical Report HKUST-CS98-01. Hong Kong University of Science and Technology.

[63] Toshikazu Takeda and Hideaki Ikeda. 1991. 3-D Neutron Transport Benchmarks. *Journal of Nuclear Science and Technology* 28, 7 (1991), 656–669.

[64] Yuchi Tian, Kexin Pei, Suman Jana, and Baishakhi Ray. 2018. DeepTest: Automated Testing of Deep-Neural-Network-driven Autonomous Cars. In *ICSE'18*. arXiv:1708.08559

[65] Xiaoyuan Xie, Joshua W.K. Ho, Christian Murphy, Gail Kaiser, Baowen Xu, and Tsong Yueh Chen. 2011. Testing and validating machine learning classifiers by metamorphic testing. *Journal of Systems and Software* 84, 4 (2011), 544–558.

[66] Xin Yi, Liqian Chen, Xiaoguang Mao, and Tao Ji. 2017. Automated Repair of High Inaccuracies in Numerical Programs. In *ICSME'17*. IEEE, 514–518.

[67] Xin Yi, Liqian Chen, Xiaoguang Mao, and Tao Ji. 2018. Efficient Global Search for Inputs Triggering High Floating-Point Inaccuracies. *Proceedings - Asia-Pacific Software Engineering Conference, APSEC* 2017-Decem (2018), 11–20.

[68] Xin Yi, Liqian Chen, Xiaoguang Mao, and Tao Ji. 2019. Efficient automated repair of high floating-point errors in numerical libraries. *Proceedings of the ACM on Programming Languages* 3, POPL (jan 2019), 1–29.

[69] Jie Zhang, Junjie Chen, Dan Hao, Yingfei Xiong, Bing Xie, Lu Zhang, and Hong Mei. 2014. Search-Based Inference of Polynomial Metamorphic Relations. In *ASE'14* (Vasteras, Sweden). Association for Computing Machinery, New York, NY, USA, 701–712.

[70] Daming Zou, Ran Wang, Yingfei Xiong, Lu Zhang, Zhendong Su, and Hong Mei. 2015. A Genetic Algorithm for Detecting Significant Floating-Point Inaccuracies. In *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*. IEEE, 529–539.

[71] Daming Zou, Muhan Zeng, Yingfei Xiong, Zhoulai Fu, L U Zhang, and Zhendong Su. 2020. Detecting Floating-Point Errors via Atomic Conditions. In *Proceedings of the ACM on Programming Languages*, Vol. 4. 1–27.