II International Conference on Particle-based Methods - Fundamentals and Applications
PARTICLES 2011
E. Oñate and D.R.J. Owen (Eds)

STRATEGIES FOR SIMULATION SOFTWARE QUALITY ASSURANCE APPLIED TO OPEN SOURCE DEM - PARTICLES 2011

Stefan AMBERGER*, **Christoph GONIVA***, **Alice HAGER***, **Christoph KLOSS***

*Christian Doppler Laboratory on Particulate Flow Modelling
Johannes Kepler University
Altenbergerstr. 69, 4020 Linz, Austria
e-mail: stefan.amberger@cfdem.com, www.cfdem.com

Key words: Granular Materials, Open Source DEM, Quality Assurance, Test Harness

Abstract. We present a strategy to improve the software quality for scientific simulation software, applied to the open source DEM code LIGGGHTS [1] [2]. We aim to improve the quality of the LIGGGHTS DEM code by two measures:

Firstly, making the simulation code open source gives the whole user community the possibility to detect bugs in the source code and make suggestions to improve the code quality.

Secondly, we apply a test harness, which is an important part of the work-flow for quality assurance in software engineering [5]. In the case of scientific simulation software, it consists of a set of simulation examples that should span the range of applicability of the software as good as possible. Technically, in our case it consists of a set of 10-50 LIGGGHTS simulations and is being run automatically on our cluster, where the number of processors, the code features and the numerical models are varied. Qualitative results are automatically extracted and are plotted for comparison, so thus a huge parameter space of flow regimes, numerical models, code features and parallelization situations can be governed.

A test harness can aid in (a) finding bugs in the software, (b) checking parallel efficiency and consistency, (c) comparing different numerical models, and, most importantly, (d) experimental validation. Parallel consistency means that within a parallel framework, we need to have the possibility to compare the answers that a run with a different number of processors gives and the time that it takes to compute them. Experimental validation is especially important for scientific simulations. If experimental data is available for a test case, the experimental data is automatically compared to the numerical results, by means of global quantities such number of particles in the simulation, translational and rotational kinetic energy, thermal energy etc.

The LIGGGHTS test harness aims to be a transparent and open community effort that everybody can contribute to in order to improve the quality of the LIGGGHTS code. We illustrate the usefulness of the test harness with several examples, where we especially focus on experimental validation.

1 INTRODUCTION

This paper presents one instance of *testing* as a possible strategy for software quality assurance, using the example of the test harness, that is used for LIGGGHTS development.

Reading instructions for this paper are the following: The chapter *Classification* shows the main reason why we used testing as the strategy of our choice. Following that the chapter *Functional Range* presents the functionality of our test harness. The structure of our test harness and terminology (names of modules) we use later on are defined in section *Basic Structure of the Program*. At last we present several examples in order to emphasize the benefits of highly modular software in section *Examples* and afterwards conclude the paper.

2 CLASSIFICATION

One can distinguish two types of software verification: *static verification* and *dynamic verification*.

For large systems like LIGGGHTS static verification, especially non-heuristic methods, i.e. methods capable of actually proving the correctness of software by means of formal methods of mathematics via automated theorem provers, are topic of current research and not yet applicable [4]. A more applicable approach used excessively by software developers is dynamic verification alias *testing*. One main purpose of this test harness is to do large scale system tests on user defined sets of inputs, automatically verifying the correctness of the output via comparison with the output of previous versions of LIGGGHTS as well as data obtained via experiments, thus guaranteeing version consistency.

3 FUNCTIONAL RANGE

The design of this test harness allows it to be used to

1. assure version consistency
2. detect bugs in newly added functionality
3. detect and pinpoint bugs accidentially introduced while updating
4. detect unwanted behaviour of newly implemented models
5. check parallel efficiency and consistency

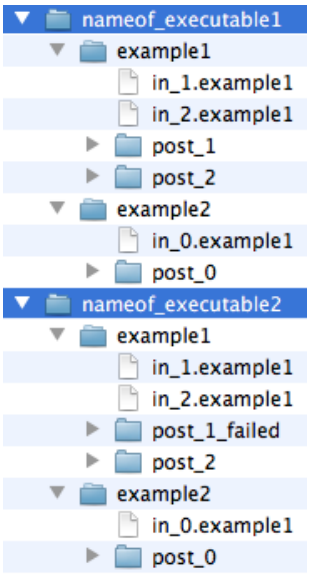


Figure 1: Typical folder structure created by step1

6. compare different numerical models (w.r.t. cpu time, total translational / rotational kinetic energy of the simulated system)

7. provide a platform for generating sets of data for experimental verification

8. automated and repeated model verification via experimental data

4 BASIC STRUCTURE OF THE PROGRAM

The test harness consists of four main parts that can be combined in different ways to yield different results, thus supporting a wide range of application (see section 3).

In step one a folder structure is generated: for each executable (one or two) a folder containing all examples and all input-scripts is created. These input-scripts are then sequentially simulated with the respective executables. All this happens on the cluster, where multiple nodes can be accessed by LIGGGHTS via MPI, according to what the input-scripts of the respective examples require. The output of each simulation is stored in a folder which unique name indicates success, failure or abortion of the simulation. Figure 1 shows a typical folder-structure created by step1, where "in_n.ex" is the n-th input-script of example ex and nameof_executable1 / nameof_executable2 are the (file)names of the executables used. The name of this step is *step1*.

In step two the test harness branches: depending on which data was produced in step one (dump-data: coordinates, velocities, and radii of all particles for each timestep OR thermo-data: timestep, number of particles, translational and rotational kinetic energy of the system, cpu-time) one chooses the branch accordingly. In this phase two data sets

in the form of folder structures of step one are compared. This happens in 5 (6) steps in *step2_dump* (and *step2_thermo*):

1. For all examples it is checked whether the simulation finished successfully.
2. For all examples which successfully finished the number of timesteps is checked (or difference thereof).
3. For all examples that passed tests one and two positively and without differences the normalized (regarding total number of particles) root mean square of the number of particles per timestep is calculated.
4. and 5. For all examples that passed tests one and two positively and without differences the normalized (regarding total number of particles and total translational / kinetic energies) root mean square of translational and rotational kinetic energies are calculated.
6. the difference in CPU time is calculated. This only happens with thermo output.

This step outputs a table that shows the differences between the two simulations of corresponding examples on different executables and basic properties of the simulation. If the folder-structure, that is analyzed during execution of the step, contains only the output of one executable the program automatically tries to extract basic properties of the simulation instead of comparing to data of another simulation. The name of this steps are *step2_dump* and *step2_thermo*.

The third part of the test harness is software, that parses input-scripts of LIGGGHTS and detects differences therein, thus allowing to locate bugs if some scripts fail to execute, and some do not. The output of this part is a table showing which LIGGGHTS functionality is called during execution, allowing easy comparison of input-scripts. The name of this part of the program is *compareIS*.

The fourth part of the test harness allows to compare experimental data e.g. in the sense of hopper mass-flow rate [3] to data obtained by simulations, and prints a visualization of the accuracy of the models used. The name of this part is *mDischarge*.

These four parts can be combined in different ways to yield different results. Some of the possible combinations are listed in section 3 and described in section 5.

5 EXAMPLES

5.1 Assuring Version Consistency

After simulating three examples with *step1* the algorithm outputs a folder structure similar to Figure 1. This structure now can be the input of *step2_dump*. This yields output like illustrated in Figure 2. In this example one can easily see, that both translational and rotational kinetic energy changed slightly in one example (ex3). This - if not willingly introduced - indicates inconsistencies between the two versions of this test.

example	inputscript	finished	timesteps	weightedRmsParticleNumber	weightedRmsTransKinEnergy	weightedRmsRotKinEnergy
ex1	in_0	2	30000	0.0	0.0	0.0
ex2	in_0	2	100000	0.0	0.0	0.0
ex3	in_1	2	40000	0.0	0.1636421739	0.1135458958

Figure 2: Output of a version consistency test

example	inputscript	finished	timesteps	weightedRmsParticleNumber	weightedRmsTransKinEnergy	weightedRmsRotKinEnergy
ex1	in_0	1	30000	0.0	0.0	0.0
ex1	in_1	None	-	-	-	-
ex2	in_0	1	100000	0.0	0.0	0.0
ex3	in_0	None	-	-	-	-
ex4	in_0	1	9900	0.0	0.0	0.0
ex5	in_0	None	-	-	-	-
ex6	in_0	1	99600	0.0	0.0	0.0
ex6	in_1	None	-	-	-	-
ex7	in_0	1	100	0.0	0.0	0.0
ex8	in_0	1	7500	0.0	0.0	0.0
ex8	in_1	1	99000	0.0	0.0	0.0
ex8	in_2	None	-	-	-	-

Figure 3: Output of a test detecting bugs in newly added functionality

5.2 Detecting Bugs in Newly Added Functionality

Via running a relatively large number of examples one can check whether some new functionality has obvious errors or not. *Step1* can run the simulations (also on only one executable, namely the newly compiled one), which results can be interpreted with *step2_dump* or *step2_thermo* respectively. For now we assume the set of simulations suitable for testing the newly added functionality output dump data. Then *step2_thermo* outputs a *.csv like in Figure 3. Here the following examples did not finish, thus resulted in abortion due to a bug: example 1: input-script 1, example 3, example 5, example 6: input-script 1 and example 8: input-script 2. If one knows properties of these examples that differenciate them to the examples that finished the simulation one knows where to search for the bug. Also *compareIS* can now be used to pinpoint the function that causes the termination of the simulation.

5.3 Detecting and Pinpointing Bugs Accidentially Introduced While Updating

Detecting Bugs can be done like in Subsection 5.1. Now assume we have three input-scripts, two of them work perfectly fine and one (the third one) not functioning properly. Then *compareIS* returns a table that shows the used functionality of LIGGGHTS of the respective input-scripts and might look like the one in Figure 4, suggesting an error related to either the command *processors* or the command *pair_style*.

command	in.script1	in.script2	in.script3
atom_modify	map	map	map
atom_style	granular	granular	granular
boundary	m m m	m m m	m m m
communicate	single	single	single
...	...	...	...
newton	off	off	off
pair_coeff	+	+	+
pair_style	gran/hertz/history	gran/hertz/history	gran/hertz/history/custom
processors			2 1 1
region	block cylinder	block cylinder	block cylinder
run	upto	upto	upto
...	...	...	...

Figure 4: Output of *compareIS*

example	inputscript	finished	timesteps	weightedRmsParticle Number	weightedRmsTransKin Energy	weightedRmsRotKin Energy	CPU time of exec1	CPUTimeExec2 / CPUTimeExec1
ex1	in_1	2	80000	0.0	0.0002033644	0.0002230624	5.3619630000	1.7310126245
ex2	in_1	2	50000	0.0	0.2557722743	0.2126971570	19.1809900000	1.6038259915
ex3	in_1	2	40000	0.0	0.2827668885	0.2351455545	13.3749500000	1.6172277835

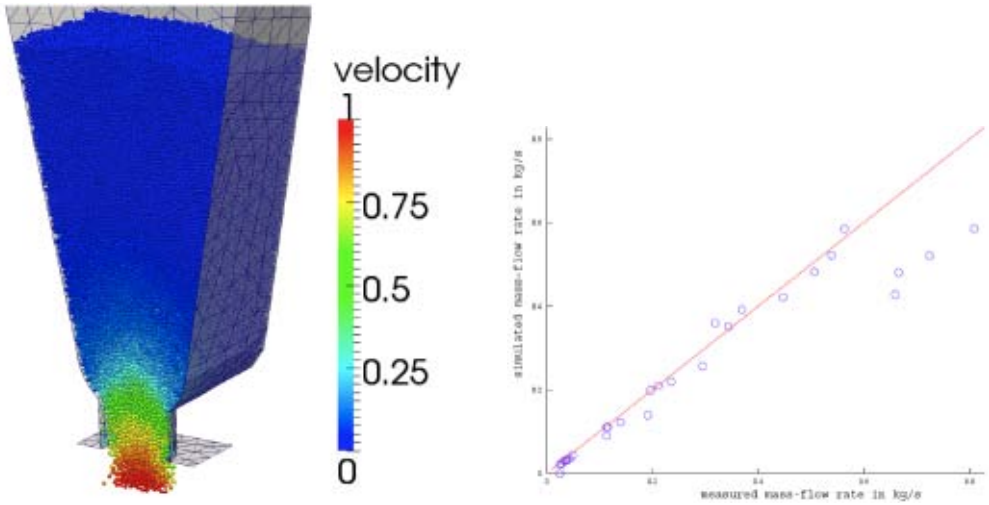
Figure 5: Output of *step2_thermo* when checking scalability

5.4 Checking Parallel Efficiency

For checking parallel efficiency a set of examples can be run two times, using a different number of processors. The number of processors is changed automatically. The output of both simulations afterwards is combined into one folder, simulating the output of *step1*. This time however this folder does not, as usual, contain the output of two different executables, but the output of one executable but using a different number of processors in each simulation. Hence *step2_thermo* compares these runs and shows the speedup in the last column of the output-table. Figure 5 shows one possible output, run for examples ex1, ex2 and ex3. In this figure all examples have about the same speedup, due to the small problem size only about 1.6.

5.5 Experimental Validation via Hopper Mass Flow Rate

One way of verifying the models used is to measure the hopper mass flow rate of e.g. glass beads and compare the results to the hopper mass flow rate of a corresponding simulation that models the experiment. A concrete example incorporated 28 experimental setups, using mono-disperse spherical glass beads of particle diameters ranging from 2-4 mm and variable hopper orifice diameters, ranging from 14-38 mm. The hopper that was used was equivalent to the hopper described by Kloss, Goniva and Pirker [2]. A simulation snapshot can be seen in Figure 6(a), the results in Figure 6(b). For most of the cases the simulated and measured mass flow rates were in good agreement.



(a) Snapshot of a hopper discharge simulation

(b) Comparison of measured and simulated hopper mass-flow rates

Figure 6: Experimental validation via hopper mass flow rates

6 CONCLUSION

It is vital to assure consistency and accurracy of models, thus the developers of LIGGGHTS put high emphasis on testing and verifying. In this context the modularity of this test harness allows it to be used in a wide area of applications and makes quality assurance easier and faster.

REFERENCES

[1] Kloss C.; Goniva C., *LIGGGHTS A New Open Source DEM Simulation Software*, Proc. 5th Intl. Conf. on Discrete Element Methods, London, August 25-26 2010

[2] Kloss C.; Goniva C.; Pirker S., *Open Source DEM and CFD-DEM with LIGGGHTS and OpenFOAM®*, Proc. Open Source CFD International Conference, Munich, November 4-5 2010

[3] Ortega-Gomes J., *Granular Experiments on Hopper Flow, Particle Charging and Bed Formation*, Master Thesis, Johannes Kepler University Linz, 2010

[4] Research Institute for Symbolic Computation, *Formal Methods in Computer Science*, Wolfgang Schreiner
http://www.risc.jku.at/research/formal/description/
April 10 2011

[5] Staknis M., *Software Quality Assurance Through Prototyping and Automated Testing*, Journal of Information and Software Technology, 32, p. 26-33, 1990