GROMACS - Task #2168

Design for multiple comparisons against same test reference data

05/05/2017 03:51 PM - Aleksei Iupinov

Status:	Feedback wanted
Priority:	Normal
Assignee:	
Category:	testing
Target version:	future
Difficulty:	uncategorized

Description

In PME CPU/GPU unit tests I want to run CPU and GPU functions and compare their outputs with the same TestReferenceData.
Example: https://gerrit.gromacs.org/#/c/6357/24/src/gromacs/ewald/tests/pmesplinespreadtest.cpp

What I do in pseudo-code is:

```
TestReferenceData refdata;
for (mode: {CPU, GPU})
{
    outputs = runPmeFunction(mode);
    TestReferenceChecker checker(refdata.rootChecker());
    checker.check(outputs, ...).
}
```

This seemed to work fine and all, until I discovered that one of the PME GPU test outputs was almost always empty, and the tests never caught on to it.
The reason is that when refdata.rootChecker() is called, internally refdata is marked as checked (hasBeenChecked_ = true).
Now if at the end of each iteration I could call something like refdata.resetCheckedStatus(), that would work. Probably easy to implement, too.
If I could just create multiple refdata's within a test, that would also be a (not very nice) solution, but there is an assert explicitly against that.
Thoughts?
Meanwhile I can hack around the problem for this specific output.

History

#1 - 05/07/2017 10:08 AM - Teemu Murtola

Now, what is it that you want to test with this particular structure? That the CPU and GPU paths produce the same results, or that they both produce some specific, relatively small set of predefined data?

If you want to test that they produce the same results, you most likely need a feature set that is nearly orthogonal to the current reference data implementation; you most likely do not want all the data written to a file, or converted to strings for intermediate representation, and/or you may want to test a large amount of data where both of the previous properties are just unnecessary overhead and may limit what you decide to test. You would probably be better off by storing the CPU computation results in an internal data structure, and then comparing them directly with the GPU output, or writing a very different generic framework for this that does everything in-memory with floating-point values...

If, on the other hand, you want to test the CPU and GPU paths separately, why not put them into different tests? That way, one test is testing just one thing, is doing the same independent of the compilation options, and any failure is easier to interpret. You can even name the tests so that it is clear what they are testing. Then the problem reduces to having two different tests use the same reference data file, which is not that hard, the main decision being how to construct the file name. If a string hard-coded in each test is sufficient, that should also be quite easy to implement. Some minor complications will come up if it needs to be possible to set the file name after creation of the TestReferenceData object (or even after rootChecker() has been called), which would require some more lazy initialization within the implementation, but even that should be relatively straightforward.

If you want to have the ability to check for unused data, I think the best solution would be to generalize the current TestReferenceChecker::checkUnusedEntries(), so that it is not global, but within the scope of that checker instance. That requires separating the "checked" state from the internal representation, and keeping that in the checkers (and in the root data object) hierarchically, and updating them also for the parents when a checker checks something.

#2 - 05/08/2017 02:45 PM - Aleksei Iupinov

The second option is what I had in mind in the very beginning (mode was one of the input parameters for the test instantiation). It's just not nice having duplicate reference data files. By iterating over mode manually within each test, I hoped to achieve ref data de-duplication (and GPU/CPU comparison, accordingly). So yes, if there was a way to provide the custom (relative?) filename to the TestReferenceData on construction, it would also solve the problem. I would only have to assert its uniqueness!