GROMACS - Task #1587

improve the configurability of regression tests

08/26/2014 05:08 PM - Szilárd Páll

Status:	New
Priority:	Normal
Assignee:	
Category:	testing
Target version:	2021
Difficulty:	

Description

Currently there is no mechanism to pass some run configuration/hardware support setting to the regression tests. A variety of different ways have been proposed/implemented so far (e.g. "max-mpi-ranks" file, checking the name of the test directory) but these are more of the workaround/quick and dirty solutions for a problem that could use a more elegant, extensible, and future-proof solution.

An easy to implement and quite extensible solution would be to replace the "max-mpi-ranks" file with general rc/config file containing key-value pairs. The file could be optional and it could contain runtime, hardware, or software/configuration requirements of the tests e.g. whether GPU or MIC can be used, max DD, max ranks, max threads, etc.

Subtasks:

Feature # 2034: Unit tests for bonded forces	**New**
Task # 2178: Move checks for specific warnings to source repo	**New**
Task # 2795: Incorporate regressiontests into core gromacs	**New**

Related issues:

Related to GROMACS - Task #1455: fix regressiontests to correctly rerun and r...	**Closed**	
Related to GROMACS - Bug #1730: gmx compare does not compare all fields of a ...	**New**	**05/12/2015**
Related to GROMACS - Bug #1840: Testing mpi version of Gromacs 5.1	**Rejected**	
Related to GROMACS - Bug #1680: Multiple problems with Jenkins+GPU regression...	**Rejected**	
Related to GROMACS - Task #2566: Refactor pdb2gmx into c++ framework	**Closed**	
Related to GROMACS - Bug #1661: Bug with free-energy + GPU + 2/3D domain deco...	**Closed**	**12/17/2014**

Associated revisions

Revision d2d27247 - 03/02/2015 09:53 AM - Mark Abraham

Add infrastructure for running MPI integration tests

Some algorithms need a minimum number of ranks to run meaningfully.
With this patch, ctest acquires the necessary logic that "make tests"
can do the right thing if the MPIEXEC_* variables are set correctly
for the excecution environment.

Added machinery so that integration tests can be written to use
the new logic, but no tests actually do so yet.

Updated install-guide section for the new (and some old)
functionality.

Minor fix to the ordering of mdrun integration test files so that they
match the comments.

Refs #1587

Change-Id: I01b419c10db42e98be708d1d4deffeba34a6d22d

Revision 180b2270 - 03/02/2015 09:56 AM - Mark Abraham

Run replica-exchange tests with MPI enabled

Replica exchange only works with real MPI and at least two ranks. This
can now be handled by ctest, so we can run such tests automatically.
The other integration tests run in MPI mode, but do not place any

requirements upon the number of ranks, so they continue to run in the usual way.

Fixed the replica-exchange integration tests so that they actually do run. Changed tests to use mdrun -multi because it is easier to write the tests that way. (Using mdrun -multidir is problematic, because grompp needs to write the .tpr file in the subdirectories. When using mdrun -multi, the only issue is that the name of the .tpr file for grompp and mdrun is different, because the latter will append the MPI rank.)

Fixed member variables to conform to naming convention.

Refs #1587

Change-Id: I11dc06f3aac81a80d679b036aef24762e9eec819

Revision efaf51cc - 03/20/2017 04:17 PM - Mark Abraham

Permit gmxtest.pl to test PME on GPUs

Introduce some temporary functionality to check whether mdrun is capable of running PME on GPUs, which will avoid needing to do a lot of Jenkins cross-verification (and humans triggering them). (We can remove the check at a later time if we think we need to.) If the code can run PME on GPUs, that can be tested in the same way as the GPU code.

Following the example of the way we re-run tests so that we also test CPU non-bonded kernels on hardware with GPUs, we arrange to test PME on both CPU and GPU when testing of GPUs is required. A better harness could perhaps detect whether GPUs are present (or were used by default for PME), but the current implementation works for Jenkins, at least.

The continuation tests such as tip4p_continue now support re-running.

Accordingly, extracted reused functionality into prepare_run_with_different_task_decomposition.

Refs #2124, #1587

Change-Id: I64c45c62f96d06f057a6f886727fc5852b84b9ba

Revision 27a4d57c - 07/16/2018 05:35 PM - Mark Abraham

Replacement for pdb2gmx tests

This test directly asserts upon the .top and .gro files that are written out, using fragments of the regressiontests/complex/aminoacids/conf.gro because these cover all basic amino acid types. It also adds testing for hydrogen vsites for amber and charmm.

We now omit doing an energy minimization after the string checks, which was always a doubtful way to test pdb2gmx. These tests are still too slow to run with other pre- and post-submit testing, so a new CTest category has been made for them, and that category is excluded from Jenkins builds by default. Developers will still run these by default with "make check" or "ctest" but that should be fast enough on a workstation. Later we can probably refactor them to use in-memory buffers and be fast enough to put with the other tests.

Modified pdb2gmx to avoid writing fractional charges for every atom in the [atoms] output, which isn't very useful for users and makes writing tests more difficult.

Fixed unstable sorting of dihedrals whose parameters are strings that identify macros.

Added new capability for refdata tests to filter out lines that vary at run time by supplying a regex that matches the lines to skip. That's not ideal, but useful for now. Better would be to refactor tools so that e.g. header output can go to a different stream, but first we should have basic tests in place.

Added tests for Regex. Fixed minor bug in c++ stdlib regex implementation of Regex. Noted the remaining reason why we have Regex supported by two different implementations.

Minor updates to use compat::make_unique

Extended functionality of CommandLine for convenient use.

Refs #1587, #2566

Change-Id: I6a4aeb1a4c460621ca89a0dc6177567fa09d9200

Revision 9ec14ef9 - 01/10/2019 09:47 AM - Mark Abraham

Remove specific group kernel testing

These are going to be removed before 2020, so it's time to stop wasting time testing them.

Refs #1587

Change-Id: Ie4be0b64279c0671b2595e952d52845c10cd2e23

Revision 50236013 - 01/10/2019 02:58 PM - Mark Abraham

Add more checks for division by zero

The rotation code was still vulnerable to numerical coincidences of atom positions and slab centers.

Refs #1587
Refs #1431

Change-Id: If7928825d74226a5b25ed3441e6b6dbdf004115a

Revision 00201fe2 - 01/10/2019 03:50 PM - Mark Abraham

Add more checks for division by zero

The rotation code was still vulnerable to numerical coincidences of atom positions and slab centers.

Refs #1587
Refs #1431

Change-Id: If7928825d74226a5b25ed3441e6b6dbdf004115a
(cherry picked from commit 5023601307ea4a905e19d83113c2ac43aaf0d71e)

Revision fe310472 - 01/11/2019 09:32 AM - Roland Schulz

Remove all testing of group scheme

All tests have converted to Verlet scheme. Some tests have been deleted where Verlet scheme doesn't support features.

The change to verlet made other changes necessary:
complex/{field,dec+water,urea}
reduced steps and nstlist because of instability
complex/urea:
integrator=md (race in do_steep)
dt=0.0005 (large step size unstable w/o constraints)
most of freenergy
rvdw=rcoulomb (other not supported)
complex/{cbt,reb}
coulombtype shift->cutoff (now in coulomb-modifier)
complex/aminoacids:
pbc=xyz comm-mode=None
complex/sw
rvdw=rcoulomb=0.8 (box size too small)
gen-vel=no (unstable)

Deleted changes because of unsupported features
complex/water: coulombtype switch
simple/{bham|morse}: buckingham
simple/simple_search_group: group search

Deleted changes because of in-progress transition
to main repo:
simple/angles1
simple/angles125
simple/bonds1
simple/bonds125
simple/dih1
simple/dih125
simple/g96angles1
simple/g96angles125
simple/g96bonds1
simple/g96bonds125
simple/imp1
simple/imp36
simple/morese
simple/rb1
simple/rb125

Refs #1587

Change-Id: I3ebfbe4360a273766551ae66b8fe2c817f4c963c

Revision 17ad792a - 01/14/2019 04:32 PM - Mark Abraham

Remove ctest section called kernel

Now that the regressiontests no longer has such a category, we should
not have a ctest section for it.

Refs #1587
Change-Id: I9871e32dfbde329ca4eceb718a94a7c8a418282a

Revision 2b63f190 - 01/17/2019 04:12 PM - Mark Abraham

Finish removing simple tests

This coverage has been moved to the source repo and GoogleTest.

Refs #1587

Change-Id: I4333535186e1c038ec6671aaaf721be6e5472d16

Revision ff29f713 - 01/18/2019 06:53 AM - Mark Abraham

Remove ctest section called simple

Now that the regressiontests no longer has such a category, we should
not have a ctest section for it.

Refs #1587

Change-Id: I6ea4fea10e01079bf9779b4127faa3f23a8d81d5

Revision 4e132b55 - 07/30/2019 04:39 PM - Mark Abraham

Remove duplicate regression tests

These were originally intended to test the group scheme implementation
of LJ-PME, which is now removed. We already duplicated them for the
Verlet scheme as nbnxn-ljpme*, and forgot to remove the duplicates
when we converted the remaining group-scheme tests over.

Refs #1587

Change-Id: I6dc1be921af446ff2316ac5de69b163790ee3f1c

History

#1 - 08/26/2014 09:25 PM - Mark Abraham

I think the only thing that is elegant, future-proof and extensible is a standalone test binary. Take a look at the history of regressiontests suite on BG (where gmx and mdrun are built for different platforms) and Windows (where POSIX is a fantasy), and think about things hardware might do in future!

I've spent the last two days discovering that if we'd had a Verlet-scheme energy-group test case all along, then it's never been able to pass on BlueGene/Q, because the intermediate precision of everything is double and so the accumulation of the potential in the j loop rounds differently from the reference code for each energy group. Even hacking in round-to-single at the j-loop accumulation point is not enough to fix the issue. So we need

another hack to relax the tolerance of just the energy part (not forces!) of the new test for Verlet-scheme energy-group and the old nbnxn-free-energy, which requires we run gmx check with different tolerances in those specific cases, or risk having a bug in forces exposed by changes in that code path be less likely to be detected. Fun fun fun. The only elegant way to add such flexibility is right next to the test case definition - in the same test source file, in the same repo as the code change.

This is part of the reason for my work on the integration test machinery - coding this stuff in C++ is not pretty either, but at least it's a language devs already have to be comfortable hacking! Accordingly, I'll continue to have limited interest in anything that looks like adding features to the regressiontests machinery.

The retry-mdrun-until-it works feature is not great, but it achieves the purpose of getting the code to run and produce a testable result. Permitting an empty .rc to fall back to old behaviour is not actually an improvement; it also can allow a test that might run in either CPU or GPU mode to silently run in just one mode unless someone does the necessary change to the regressiontests repo. So we would need an .rc to have real content so there's a chance someone will think about it. Curating the .rc contents once per test case is all very well, but someone has to write the new machinery to use it, perhaps ripping out the old rerunning machinery, then later someone else will want the values to change because we've relaxed the DD limitations somehow and there could be a bug exposed by regression test case xqczq that we're currently not testing on old_nmax+1 ranks, and ...

I think there is much to be said for having all the regression tests run single-threaded - they are about preserving behaviour and MD behaviour in parallel is barely preservable now and is going to become less so. In addition, we should have a small number of test cases, perhaps in some new harness, that are capable of demonstrating that different parallelism combinations in fact have the model-physics invariance that we expect - but I think that should be limited to observing that the total energy and force from a step is acceptably identical, because that is where we have a good modularity boundary. Testing DD, and neighbour search, and grompp warnings and parallelism-soundness of the update+constraint scheme should be done separately - because in each case there is an independent correct behaviour to verify. Then, the whole thing can be tested for producing correct ensembles, which none of our current tests go close to doing.

#2 - 08/26/2014 09:54 PM - Roland Schulz

Two points to Mark's comment (sorry a bit off-topic):

The point of integration tests is to find problems which only occur in the combination of feature A and B. If you you test parallelization separate with a few tests then for any feature not utilized by those few (e.g. ener-grps or vsite), we won't notice if its parallelization doesn't work.

I think we should try to improve reproducibility and not accept non-reproducible parallelization. In those cases where the fastest algorithm isn't reproducible we should have a reproducible fallback. Currently most algorithms are reproducible. The flop option of dlb is a good example of a option to make it testable under reproducible conditions. And the fact that the GPU doesn't have a reproducible aggregation makes it impossible to find a problem if the problem only occurs at large scale. I think this is an unsustainable situation.

#3 - 08/27/2014 10:57 AM - Mark Abraham

Roland Schulz wrote:

> Two points to Mark's comment (sorry a bit off-topic):
>
> The point of integration tests is to find problems which only occur in the combination of feature A and B. If you you test parallelization separate with a few tests then for any feature not utilized by those few (e.g. ener-grps or vsite), we won't notice if its parallelization doesn't work.

Correct, and we should test energies and thus energy groups component-wise, not total I like I said above. But I do not think that "testing parallelization" is the right way to think.

Testing that a component works in parallel, by reproducing the result of other implementations, is good. The vsite implementation has nothing directly to do with parallelism - that the various projections work is unit-testable, that DD correctly recognizes that vsite-related data needs communication is testable, that the thread-parallel implementation works the same as the serial implementation is testable. Wanting reproducible parallel hardware-agnostic end-to-end regression test machinery is the poor man's version. With those tests of vsite functionality in place, and machinery to observe that (say) protein in water with vsites produces the right ensemble across various parallelization schemes, what's missing?

> I think we should try to improve reproducibility and not accept non-reproducible parallelization.

That's a topic in its own right, but to do that properly we'd have to have a fixed precision implementation. There's a lot to be said for that (convenience in developing & thus confidence in the correct result). But for an opening gambit, to do that on BlueGene/Q (and probably K), where there is only double-precision SIMD, we'd have to do at least the j-loop accumulation for q, C6 and C12 in fixed-point on the non-SIMD part of the CPU, and still we'd have issues with (lack of) intermediate rounding to single precision while actually computing stuff on the FPU before that.

Fixed-point update phase is somewhat more achievable (caveat: I don't understand the detailed arithmetic needs of LINCS and SHAKE).

> In those cases where the fastest algorithm isn't reproducible we should have a reproducible fallback.

Sure, but that's only a partial solution - very useful, but mostly of use to devs. We also need tests that are valid on the code that we expect people to use in production simulations, and those are going to be using parallelism, load balance, redundant computation, etc.

> Currently most algorithms are reproducible. The flop option of dlb is a good example of a option to make it testable under reproducible conditions.

DLB is reproducible, and thus a great thing to have while developing. But the flop counts are unmaintained, and not a great proxy for CPU cycles any more. Aside, should we stop using -notunepme in regressiontests, in favour of this mode?

> And the fact that the GPU doesn't have a reproducible aggregation makes it impossible to find a problem if the problem only occurs at large scale. I think this is an unsustainable situation.

I think we should be thinking in terms of fixed point here, also. Accumulation and reduction starts at the end of the j loop, and we don't really have a mixed-precision mode yet.

#4 - 10/08/2014 11:49 PM - Szilárd Páll

If the test binary approach provides high-level regression testing functionality that can replace what gmxtest does now, I can fully agree with your suggested approach - perhaps even with the unwillingness of maintaining gmxtest **if**:

- We at least tried to avoid hiding our heads in the sand. It is dangerous to ignore the major shortcomings of the current testing - some of which wouldn't be hard to fix - and there have been several bugs that confirm this. It's easy, but not productive to just point at some of the flaws of the regressiontesting and use that as an argument for keeping it unmaintained and abandoning right now.
- The framework was ready to replace gmxtest including most test cases (or functionalities) that gmxtests supports ATM ported to the test binary.
- Extending on the above points, regressiontesting has become an essential part of our workflow, developers rely on it. Hence, creating the false sense of security by leaving regressiontesting issues not addressed and instead focusing on a different solution is dangerous and overall not a resonable approach, I think.

#5 - 12/26/2014 12:48 PM - Mark Abraham

Szilárd Páll wrote:

> If the test binary approach provides high-level regression testing functionality that can replace what gmxtest does now, I can fully agree with your suggested approach - perhaps even with the unwillingness of maintaining gmxtest **if**:
>
> - We at least tried to avoid hiding our heads in the sand. It is dangerous to ignore the major shortcomings of the current testing - some of which wouldn't be hard to fix - and there have been several bugs that confirm this. It's easy, but not productive to just point at some of the flaws of the regressiontesting and use that as an argument for keeping it unmaintained and abandoning right now.

Agreed - it should be kept in place while being replaced, but I think implementing new functionality is a good thing to do in a new harness (whatever form it takes). In 2014, we extended gmxtest.pl for doing CPU-only reruns and per-test-case limiting of parallelism. We did the refactorings in a way that should permit similar behaviour extensions. I'm not opposed to people using that, but I'll be much more interested in moves towards using better infrastructure - and will choose to spend my time creating new infrastructure to address the kinds of concerns you guys raise, rather than extending the old.

> - The framework was ready to replace gmxtest including most test cases (or functionalities) that gmxtests supports ATM ported to the test binary.

Regarding the existing regressiontests modules:

- simple should pretty much go and be unit tests on bonded functions - I don't think they do anything much else (but we should check)
- kernel will die with the group scheme (these take about half the run time at the moment)
- pdb2gmx is already "run pdb2gmx... grompp... mdrun and check the EM output energy." This is a few hours of work to do in the integration test machinery right now, but would be better done with code that intercepts writing the .edr file and does the checking with reference data in memory. Note that these tests spent years silently doing nothing until someone noticed that the implementation that grepped md.log for the final energy was silently not working...
- complex and freeenergy test coverage needs to be largely preserved (and in some places converted to Verlet and in many places extended). Some specific considerations:

1. Already several test cases are documented in a README to be "this other test case run under these different .mdp conditions," which is a bit of a waste from redundant checking for some stuff. It's also hard to get perspective on the test coverage with similar-but-different things at the same level as totally different things. This would be much better with a loop in a single place over matching sets of inputs + expected outputs + tolerances (in old or new harness).
2. It is perfectly reasonable to test that some set of .mdp inputs produces a given set of grompp warnings. But such a test ought to be focused only on that. If the text of the warning is checked, it should be checked against the same format string that produces it, so that there is not manual work for us regenerating reference data for a separate repo requiring coordinated review and merging when someone just wants to fix something trivial. I think the format string should be defined where it is used, though - having to add a format string for a warning to some database of strings somewhere else will lead to people being lazy about adding warnings. Better would be build-time (and tarball-time) parsing of the handful of source files so that grompp_warning(timeStepTooShort, "Shortest oscillation period is %f, but time step is %f", period, delta_t) generates a build-tree header file of format strings in some container indexed by timeStepTooShort, so that we can have grompp integration tests parameterized on the .mdp settings and matching expected format strings. Same for mdrun warnings (currently untested). If so, then when someone changes the code so that a warning is no longer issued, then they can address that in the same commit, and reviewers can prompt them to add a test to the set with just adding a couple of lines of .mdp fragment. (One could also argue that grompp_warning when run by the test binary should have a mock implementation that just records the event, so that no fragile parsing of anything is ever needed. That's probably a lot less work. There should also be a separate test that output from grompp_warning() normally does in fact end up on stderr.)
3. I'm not sure what value exists comparing that the .tpr file generated by the new code matches that of the old code. David probably added it long

ago just because it was possible. It's pretty hard to inadvertently change the .tpr output. When we change the .tpr format deliberately there's some value in being able to see via .tpr-diff exactly what we changed, but a Jenkins implementation that accepts an .mdp fragment, builds/gets the old code, builds the new code and runs the new gmx check on the .tpr files is a good way to give developers and reviewers that feedback. I don't see the value in requiring a manual update, review and simultaneous merge of some number of archived .tpr files for every commit that is some kind of breaking change to old .tpr behaviour, in particular because the amount of work tends to increase as we improve test coverage and having to do that work stops us doing other things. (But we should keep tests that show that gmx check correctly observes equivalence and difference between all of its input file types.)
4. That leaves the checks on .edr and .trr output. These are valuable and I have a number of proposals here, but I will speak more of these in a later post.

Note also that the integration test framework is no longer completely new - it's been testing things including mdrun trajectory writing, IMD and Carsten's swap code in Gromacs 5 and hasn't fallen over (but doesn't assert very much about the code it runs...). I'm aware of problems getting ctest to run MPI binaries in parallel, which needs work, e.g the replica-exchange cases are currently neither run automatically (nor useful if they are run manually).

- Extending on the above points, regressiontesting has become an essential part of our workflow, developers rely on it. Hence, creating the false sense of security by leaving regressiontesting issues not addressed and instead focusing on a different solution is dangerous and overall not a resonable approach, I think.

I am prepared to help fix existing problems, but even coordinating fixes needs timely help from other people if we want two people to sign off on all fixes... I could email qualified individuals or spam gmx-developers every time such things need to happen, but now I sound like your mother nagging. I'd like to think the qualified individuals browse gerrit approximately daily and see such things happen.

I think there is greater current and future value if we focus some time on getting the above kinds of things done (plus perhaps being able to run grompp and mdrun without writing a .tpr to disk). In particular, I think we can make the end-to-end testing much more useful through making it run in seconds rather than minutes (through fewer binaries to load, less disk access overall, less parsing of files, less checking of useless things), and not needing so many stored binary blobs or a separate repo. Then, adding more code paths under automatic tests is a much more palatable proposition, for users, devs and Jenkins.

#6 - 12/26/2014 10:04 PM - Mark Abraham

Roland Schulz wrote:

> I think we should try to improve reproducibility and not accept non-reproducible parallelization. In those cases where the fastest algorithm isn't reproducible we should have a reproducible fallback. Currently most algorithms are reproducible. The flop option of dlb is a good example of a option to make it testable under reproducible conditions. And the fact that the GPU doesn't have a reproducible aggregation makes it impossible to find a problem if the problem only occurs at large scale. I think this is an unsustainable situation.

And Mark then wrote:

> ... Wanting reproducible parallel hardware-agnostic end-to-end regression test machinery is the poor man's version...

I've since learned that there is a fairly cheap way of implementing a reproducible floating-point reduction, which is the key thing we'd like to have to make it possible to demonstrate the equivalence of different implementations or under different degrees of parallelism. I got the idea from http://htor.inf.ethz.ch/publications/img/arteaga-fuhrer-hoefler-reproducible-apps-ipdps14.pdf: you measure the largest force component, and use that to set a power of two called M that is greater than the largest partial sum that will be correctly accumulated. Of course, this is the same thing you need to do for implementing fixed-point reduction, so there might be some guidance available from Amber's choice. You then add and subtract M from each component before accumulating it in the reduction. This does a pre-rounding that makes the accumulation binary reproducible. Since we'll have about a few hundred force contributions for each atom (counting long-ranged as one contribution), then our M from their equation 1 will probably be about 1e7 kJ/mol/nm. That's approximately 2e23, so using such M for forces means the total force on an atom will have around zero digits of precision after the decimal point. However, their choice of M caters to theoretical worst-case maximal partial sums, which we are unlikely to see in practice, so M one or two orders of magnitude smaller is probably useful. There is the option to measure the remainder and do a second pass of accumulation if we think we need to (see their paper), but that would mean implementing some force data structure whose implementation we can vary in a way that doesn't slow down the normal code. We can play with this in a plain-C kernel to see what our actual numbers look like.

The good thing I think we can do is add a special testing build mode where the accumulation at the end of all force component calculations is modified to use a pre-computed M, flagging if any overflow actually happens from choosing M too small. Because the accumulation is now binary reproducible, I think we can run integration tests using any implementation of our algorithms on any platform, and we do not need to choose any tolerance for accumulation or integration losses. Because the only difference in the code is the few flops with M before reduction, then we can be confident that the normal build has no issues arising from parallelism+algorithm+implementation choices that have passed in reproducible mode, and we can store a single set of reference data that many combinations can compare against. Testing in normal mode should also occur (and be available to normal users), but for those we need some old-school floating-point tolerances. The point is to try to stop us missing bugs (e.g #1647, #1661, and doubtless others) because the problem of choosing old-school tolerances that are tight enough to detect subtle real bugs but loose enough for no false hits is perhaps too hard to solve and maintain. We'd perhaps also have caught #1603 if we'd been able to feel confident to test with PME tuning on (but perhaps no existing test case ran long enough to see the effect).

Such testing does not assert anything directly about the quality of the normal reduction wrt floating-point precision or operand ordering; those issues should be addressed differently e.g. through conservation and ensemble-correctness tests. Once we've got such a modified implementation, it could also be good for probing the limits of the normal accumulation. An alternative could be to pick a scaling factor to convert force components to an integer (whose byte width matches real) and accumulate in that type. The use of M above restricts the practical range of the resulting force, so we waste some of the bits in the significand that we can get back if we use an integer. Then we convert back to real before the update. This could even be cheap enough to use in production code if we find there's something real to gain. (Hur hur, "real!") Note that our inter-rank reduction is in double

precision, but the j-loop accumulates in single, so I don't think we have something we should really call mixed-precision mode at present.

#7 - 12/26/2014 10:17 PM - Mark Abraham

Discussion continued from #1661:

Mark said

> Replacing [current FEP regression test approach], sure...

Szilard said

> Getting off-topic here, but let me answer briefly this time because this is getting old. As far as I can tell, that's your personal opinion that the developer community never agreed on.

Right, but what I think counts a lot, because I've made more than half the commits to that repo in the last two years, and if it takes me announcing that I'm not going to add more functionality to it in order to force some change, then that will cost me no sleep. I had been the sole maintainer for several years when we adopted it for CI testing. I said at that time that I thought it was unsuitable, but as IIRC Roland said at the time there was no reasonable alternative in the short term. That was true, but about three years have now passed.

> For instance, both Roland and I were on the opinion that currently there is nothing on the horizon that can take up the role that regression-testing has (that is integration tests).

Not true. We already have other machinery for running integration tests for grompp and mdrun. It needs more work, but it can already do as much as the regression tests can do (run gmx grompp, mdrun, and gmx check on the output files, dumping issues to XML/terminal).

The fact that it took me 6 hours to discover that Windows does not have the same behaviour for flushing stderr and stdout, so the test harness can't parse it properly, so that we can't reliably observe that the grompp warnings haven't changed, is itself a bug (https://gerrit.gromacs.org/4332). This is not an isolated incident, and I'm unwilling to commit more time than absolutely necessary to work on machinery that I do not think serves our needs well enough any more. See above posts for my proposals for new forms to cover the range of testing behaviours that I think are good to preserve.

#8 - 01/04/2015 02:15 PM - Gerrit Code Review Bot

Gerrit received a related patchset '1' for Issue #1587.
Uploader: Mark Abraham (mark.j.abraham@gmail.com)
Change-Id: I01b419c10db42e98be708d1d4deffeba34a6d22d
Gerrit URL: https://gerrit.gromacs.org/4344

#9 - 01/04/2015 03:05 PM - Gerrit Code Review Bot

Gerrit received a related patchset '1' for Issue #1587.
Uploader: Mark Abraham (mark.j.abraham@gmail.com)
Change-Id: I11dc06f3aac81a80d679b036aef24762e9eec819
Gerrit URL: https://gerrit.gromacs.org/4345

#10 - 04/09/2015 11:23 AM - Mark Abraham

- Related to Task #1455: fix regressiontests to correctly rerun and report the mdrun usage added

#11 - 05/12/2015 02:46 PM - Mark Abraham

- Related to Bug #1730: gmx compare does not compare all fields of a .tpr added

#12 - 06/15/2015 02:14 PM - Teemu Murtola

Let me just paste an e-mail from gmx-developers (from Oct 4, 2014) that never got any reply; maybe it gets considered in this context at some point. There is actually some discussion in this Redmine issue on things that were lacking in that e-mail thread, but it does not hurt to reiterate things. And that said, I'm still a bit sceptical about the possibility to creating large-scale integration tests that are binary-reproducible across different compilers, platforms, and different optimization levels, even if the force reduction could be improved in this respect.

> The discussion here focuses mostly on technical implementation details of how to organize the testing, but I think an even more important discussion would be to investigate what we should be testing. A major reason why the existing regression tests do not test things like -tunepme is that they try test for exact reproducibility of trajectories in an inherently chaotic algorithm, and that simply cannot support testing longer runs that would be required for testing the actual mdrun functionality.
>
> This is where unit tests could help: if there were separate unit tests that would test the individual forces and energies, the regression tests could concentrate on testable global properties of longer runs that are still sensitive to correct implementation, such as energy conservation, instead of trying to test every fragile property of the trajectory.
>
> Like Roland said, I think there is nothing inherently bad with an external script to run the tests (although, if we rewrite the script from scratch, I'd propose Python). The main benefit of a C++ testing framework comes from the ability to access internal data structures, which in turn allows

> more lightweight tests and assertions on the internal state. But in order to have such access, substantial refactoring of the code under test may be required, so this is not a magic bullet. For integration-style tests for whole executables, an external driver written in a scripting language is probably a much easier solution.
>
> Finally, if the blocker is the availability of resources for writing such scripts (and not deciding what they should test), I can volunteer to help with the script infrastructure, if people think that the most valuable thing I could be doing. I can't promise to deliver such scripts in O(days) time, but if the alternative is that it never happens, waiting a few weeks or months should not be a big deal.

#13 - 06/15/2015 02:22 PM - Teemu Murtola

And about the configurability (which is the main topic here), I think we should restructure the test cases: instead of creating a set of tests and a script that tries to run all the tests with the same parallelization options. Instead, each test should specify (a set of) parallelization options that it **needs** to use, selected based on what we think that test should be testing and what different parallelization options make sense in this context. The user/caller of the test script can then tell what resources are available, and the test either runs or it doesn't run (and if it doesn't, it clearly reports why it didn't), using at most the resources the user provided.

Running tests on a user-provided number of ranks etc. is probably only useful for performance testing, and possibly for ensemble validation and such.

For this, it does not matter what is the mechanism to run the tests; both the regression tests and the new stuff that uses CTest have the same problem to solve. The issue is particularly bad with the regression tests, leading to the current complicated machinery.

#14 - 06/29/2015 02:33 PM - Mark Abraham

Concern and discussion over the state of pdb2gmx and its tests took place here https://gerrit.gromacs.org/#/c/4772/

#15 - 07/19/2016 07:26 PM - Mark Abraham

- Related to Bug #1840: Testing mpi version of Gromacs 5.1 added

#16 - 07/19/2016 07:35 PM - Mark Abraham

- Related to Bug #1680: Multiple problems with Jenkins+GPU regressiontests added

#17 - 08/01/2016 09:38 PM - Mark Abraham

https://gerrit.gromacs.org/#/c/5522/ should have referenced this issue, but did not. We now have infrastructure callable from C++ code that is able to compare energies and forces computed by mdrun simulations, which makes it possible to implement tests that could replace the coverage available from much of the regressiontests repo. One (new) example application is at https://gerrit.gromacs.org/#/c/5435.

Using this infrastructure, I plan to propose

- a set of tests on inputs with small numbers of particles for which it is reasonable to store per-step energies and forces even in XML format
- a different set of inputs, closely related to those already in the regressiontests repo, for which the focus is to test the high-level aspects of the implementation, and where e.g. the final energy with suitable tolerance is sufficient as a measure of success
- that both can be run so that they target different hardware implementations, preferably in a way that can be controlled from the command line (but already bin/mdrun-test -nt x and mpirun -np x bin/mdrun-mpi-test do what one expects; also see http://redmine.gromacs.org/projects/gromacs/repository/revisions/master/entry/docs/install-guide/index.rst#L920 for how to control the testing behaviour at CMake time)

There are some other aspects I note above that are worth preserving from the current behaviours tested by regressiontests repo (and some new tests driven by gmxtest.pl) but I expect that together this removes the need to store large binary blobs, and with it the need to have an external repo to contain reference data against which we test.

#18 - 08/03/2016 01:38 PM - Mark Abraham

Further thoughts arising from Monday's meeting

Consider exposing any useful command-line options for mdrun tests (at least -gpu_id and -ntomp; number of ranks can already be controlled for both kinds of MPI), but also consider varying such things internally for some kinds of testing

When an end-to-end mdrun test fails in GoogleTest, work out how to capture the (intermediate and?) final output and present it from Jenkins (but this is hard while we just dump stuff to stdout/stderr, because there's no seam through which we can capture that on a per-test-case basis, and also hard if the failure was a segfault or gmx_fatal call)

#19 - 08/07/2016 11:38 AM - Teemu Murtola

Mark Abraham wrote:

> When an end-to-end mdrun test fails in GoogleTest, work out how to capture the (intermediate and?) final output and present it from Jenkins (but this is hard while we just dump stuff to stdout/stderr, because there's no seam through which we can capture that on a per-test-case basis, and also hard if the failure was a segfault or gmx_fatal call)

For files, this should certainly be possible with a combination of releng scripting (and possibly junit-attachments Jenkins plugin), some CTest/GTest features, and TestFileManager changes.

stdout/stderr can't work very well before we can redirect the output from the code to a separate file (or a memory buffer).

#20 - 03/08/2017 06:09 PM - Mark Abraham

Note that the fact that regressiontests is still separate from the tests in the source repo is adding extra work because we have to add feature support to gmxtest.pl to work out whether the gmx binary it uses supports PME on GPUs, or not, so that we don't have lots of work to do triggering cross-verification.

If the regression tests had already been implemented in the source repo using something based on the above machinery, then that would be a simpler job because the same patch in Gerrit could turn on its own testing.

#21 - 03/10/2017 05:46 PM - Gerrit Code Review Bot

Gerrit received a related patchset '3' for Issue #1587.
Uploader: Aleksei Iupinov (a.yupinov@gmail.com)
Change-Id: regressiontests~master~Ie6f327d139b09b61eeb969655c18d9750c2bdb1d
Gerrit URL: https://gerrit.gromacs.org/6510

#22 - 03/10/2017 07:49 PM - Szilárd Páll

Mark Abraham wrote:

> Note that the fact that regressiontests is still separate from the tests in the source repo is adding extra work because we have to add feature support to gmxtest.pl to work out whether the gmx binary it uses supports PME on GPUs, or not, so that we don't have lots of work to do triggering cross-verification.
>
> If the regression tests had already been implemented in the source repo using something based on the above machinery, then that would be a simpler job because the same patch in Gerrit could turn on its own testing.

IF. It has not and as I noted on gerrit, I think it's not a robust approach to try to implement integration/regression tests for one kind of offload/acceleration testing in GTest at this time (especially considering time limitations) and leave others in the old framework.

#23 - 03/20/2017 03:03 PM - Mark Abraham

Szilárd Páll wrote:

> Mark Abraham wrote:
>
> > Note that the fact that regressiontests is still separate from the tests in the source repo is adding extra work because we have to add feature support to gmxtest.pl to work out whether the gmx binary it uses supports PME on GPUs, or not, so that we don't have lots of work to do triggering cross-verification.
> >
> > If the regression tests had already been implemented in the source repo using something based on the above machinery, then that would be a simpler job because the same patch in Gerrit could turn on its own testing.
>
> **IF**. It has not and as I noted on gerrit, I think it's not a robust approach to try to implement integration/regression tests for one kind of offload/acceleration testing in GTest at this time (especially considering time limitations) and leave others in the old framework.

I'm not sure what you mean. In src/programs/mdrun/tests/moduletest.cpp, callMdrun has exactly the leverage needed to let a test case specify its parallelism limitations and integrate that with what the build can do. (But like gmxtest.pl, it does not yet have a way to find out what the execution hardware is, so what it puts into gpu_id is just a hack.) I wrote https://gerrit.gromacs.org/#/c/5522/ infrastructure long ago, and follow-up work on https://gerrit.gromacs.org/#/c/5435/30. The argument that "we're out of time so someone should hack gmxtest.pl again" does not apply - particularly not to me. It is straightforward to use that reading infrastructure to implement a regressiontest vs e.g. a handful of final energies kept in XML style, but I haven't done it yet.

Vision for testing management with separated roles (building on others' ideas also):

1. The matrix specification can specify high-level algorithms/implementation styles to test (e.g. MPMD, style of DD, NB/PME on GPU, run tests with -nb auto/gpu/cpu rather than relying on re-running cleverly chosen stuff)
2. releng can be extended to make test runners (any of them) aware of what the hardware on the actual build slave is, because we specify it in slaves.py,
3. test runners need to accept such information (e.g. in Jenkins, gmxtest.pl can read a config file about what the matrix specification requires, and another for what the hardware on this slave to use is; in the hands of users we have to make the defaults always work and always test something in a useful way)
4. individual tests state their own limitations (e.g. as max-mpi-ranks does, but for more things, so that an RF test doesn't blindly specify mdrun -ntmpi 3 -npme 1 -gpu_id 01 leading to an mdrun error)
5. test runner synthesizes all that data so that mdrun is called in ways that if it issues an error, there's something somewhere to fix (test specification, synthesis code, mdrun logic implementing auto, or actual mdrun simulation functionality)

But as discussed at length many times, for many reasons, I have zero enthusiasm for doing any of that for gmxtest.pl. If people want to keep pouring effort into it, I'm not going to stop you, but I'm also not going to spend my time on it.

One advantage of a GTest approach is that point 3 might be able to re-use the GROMACS hardware detection code without having to do some/all of maintain a hardware description in releng, write python code to serialize that, and write perl or C++ code to unserialize that. And of course there is lower friction when tests are not maintained in a separate repo.

#24 - 03/23/2017 03:16 PM - Szilárd Páll

Mark Abraham wrote:

> Szilárd Páll wrote:
>
> > Mark Abraham wrote:
> >
> > > Note that the fact that regressiontests is still separate from the tests in the source repo is adding extra work because we have to add feature support to gmxtest.pl to work out whether the gmx binary it uses supports PME on GPUs, or not, so that we don't have lots of work to do triggering cross-verification.
> > >
> > > If the regression tests had already been implemented in the source repo using something based on the above machinery, then that would be a simpler job because the same patch in Gerrit could turn on its own testing.
> >
> > **IF**. It has not and as I noted on gerrit, I think it's not a robust approach to try to implement integration/regression tests for one kind of offload/acceleration testing in GTest at this time (especially considering time limitations) and leave others in the old framework.
>
> I'm not sure what you mean.

I mean that the tests have not been ported. The existence of code that in theory can do it it won't encourage porting of the old regressiontests. The enforced deprecation of gmxtest.pl contributes equally little to a collective effort. It does however contribute to continued frustration where new features often can't be tested with reasonable effort from the contributor of the feature.

> Vision for testing management with separated roles (building on others' ideas also):

Thanks for communicating it. Note however that this alone won't make the devs align with the vision nor will it encourage broader effort to getting things done. Without any of those, this sounds like a private todo list made public that may or may not happen until the next feature needs testing infrastructure.

> But as discussed at length many times, for many reasons, I have zero enthusiasm for doing any of that for gmxtest.pl. If people want to keep pouring effort into it, I'm not going to stop you, but I'm also not going to spend my time on it.

Point taken. But that does still not help the project converge toward better testing and ultimately I don't think it's relevant what individual preferences are, but that testing is improved and contribution can be made with lower effort.

#25 - 05/10/2017 01:05 AM - Mark Abraham

Szilárd Páll wrote:

> Mark Abraham wrote:
>
> > Szilárd Páll wrote:
> >
> > > Mark Abraham wrote:
> > >
> > > > Note that the fact that regressiontests is still separate from the tests in the source repo is adding extra work because we have to add feature support to gmxtest.pl to work out whether the gmx binary it uses supports PME on GPUs, or not, so that we don't have lots of work to do triggering cross-verification.
> > > >
> > > > If the regression tests had already been implemented in the source repo using something based on the above machinery, then that would be a simpler job because the same patch in Gerrit could turn on its own testing.
> > >
> > > **IF**. It has not and as I noted on gerrit, I think it's not a robust approach to try to implement integration/regression tests for one kind of offload/acceleration testing in GTest at this time (especially considering time limitations) and leave others in the old framework.
> >
> > I'm not sure what you mean.
>
> I mean that the tests have not been ported. The existence of code that in theory can do it it won't encourage porting of the old regressiontests. The enforced deprecation of gmxtest.pl contributes equally little to a collective effort. It does however contribute to continued frustration where

> new features often can't be tested with reasonable effort from the contributor of the feature.

I wrote https://gerrit.gromacs.org/#/c/5522/ and https://gerrit.gromacs.org/#/c/5435/ and nobody much cared. I'm happy to do an illustrative port of a regression test case, but it seemed like other people were happy with the status quo. :-)

> > Vision for testing management with separated roles (building on others' ideas also):
>
> Thanks for communicating it. Note however that this alone won't make the devs align with the vision nor will it encourage broader effort to getting things done. Without any of those, this sounds like a private todo list made public that may or may not happen until the next feature needs testing infrastructure.

Right. So one of the things that is needed is buy-in from multiple devs to work on testing infrastructure so that we can have something that works well and can be adapted and extended. :-) I've said multiple times that I'm not going to the primary work to move gmxtest.pl into some future state, and nobody else has stepped up to do that, either. So we've voted with our feet against gmxtest.pl.

I think the approach of using GoogleTest to implement high-level tests permits points 2-5 above to be implemented in straightforward ways that combine well with other objectives (acquire more C++ experience in devs, encourage error handling that doesn't terminate the binary, evolve enough modularity that we don't need to write actual output files from tools just to assert that it was correct, doesn't require dependencies that rate to be awkward on weird new hardware platforms).

I'm open to alternatives, but there haven't been any proposed in the last 2+ years.

I could find myself agreeing with a proposition that some python runner script in the source repo could be a good test harness, ie would call gmx replacing gmxtest.pl, but it's still limited to testing outputs by what goes into files, which is a) limited, and b) slow, so c) we're artificially limited in what and how much we test.

For example, grompp issues some warnings, and we should want to test that certain inputs do and do not trigger them. Such tests need to be in the same repo, so that someone correcting a spelling error in the warning text does not have to manage commits in two repos. So something in the source repo needs to be able to construct a range of tpr files and run them and parse the output to find the appropriate strings, and perhaps keep reference copies of those strings up to date. We could write a new e.g. python runner to do that, or instead write the warnings using some form of Teemu's new logging functionality (which we anyway need to do at some point). Then, in test code we can adapt the implementation so that e.g. we compare the messages directly to those stored in refdata XML, or use a GoogleMock expectation (easiest if we are content to just assert on the number of warnings issued).

> > But as discussed at length many times, for many reasons, I have zero enthusiasm for doing any of that for gmxtest.pl. If people want to keep pouring effort into it, I'm not going to stop you, but I'm also not going to spend my time on it.
>
> Point taken. But that does still not help the project converge toward better testing and ultimately I don't think it's relevant what individual preferences are, but that testing is improved and contribution can be made with lower effort.

OK, since it's the only proposition with any example code written, what's good and bad about https://gerrit.gromacs.org/#/c/5435/30/src/programs/mdrun/tests/rerun.cpp?

#26 - 05/10/2017 11:41 AM - Mark Abraham

Mark Abraham wrote:

> For example, grompp issues some warnings, and we should want to test that certain inputs do and do not trigger them. Such tests need to be in the same repo, so that someone correcting a spelling error in the warning text does not have to manage commits in two repos.

And, case in point, Berk's https://gerrit.gromacs.org/#/c/6613 means every commit in the source master branch will have to rebase in order to pass Jenkins. I have opened #2178 to suggest and discuss ways forward.

#27 - 03/02/2018 01:52 PM - Mark Abraham

Feedback https://gerrit.gromacs.org/c/7478/ makes clear that the regressiontests suite is broken, not maintained, and most developers hope that someone else will maintain it.

Rather than let another three years go past, I suggest we stop working on any new functionality until its coverage is replaced by something else. I don't mind what or how, but so far I'm only aware of my own proposals.

#28 - 03/02/2018 02:48 PM - Szilárd Páll

Mark Abraham wrote:

> Feedback https://gerrit.gromacs.org/c/7478/ makes clear that the regressiontests suite is broken, not maintained, and most developers hope that someone else will maintain it.

BTW, it was you who declared it unmaintained, clearly stated that you do not endorse nor support any effort to maintain, extend it to support anything (that's not a last-minute need), so frankly it's not unreasonable that developers have waited for the new replacement to actually materialize.

(PS: and that some useful changes to evolve new machinery got lost in history can't always be blamed on "others"; noise and lack of concerted effort can equally be at fault)

#29 - 07/06/2018 06:46 PM - Mark Abraham

- Related to Task #2566: Refactor pdb2gmx into c++ framework added

#30 - 07/06/2018 06:47 PM - Gerrit Code Review Bot

Gerrit received a related patchset '1' for Issue #1587.
Uploader: Mark Abraham (mark.j.abraham@gmail.com)
Change-Id: gromacs~master~I6a4aeb1a4c460621ca89a0dc6177567fa09d9200
Gerrit URL: https://gerrit.gromacs.org/8067

#31 - 07/19/2018 12:38 AM - Mark Abraham

- Related to Bug #1661: Bug with free-energy + GPU + 2/3D domain decomposition added

#32 - 07/19/2018 12:51 AM - Mark Abraham

Szilárd Páll wrote:

> Mark Abraham wrote:
>
> > Feedback https://gerrit.gromacs.org/c/7478/ makes clear that the regressiontests suite is broken, not maintained, and most developers hope that someone else will maintain it.
>
> BTW, it was you who declared it unmaintained, clearly stated that you do not endorse nor support any effort to maintain, extend it to support anything (that's not a last-minute need)

Indeed I did. And nobody else has stepped up, thus proving me right.

> so frankly it's not unreasonable that developers have waited for the new replacement to actually materialize.

Maybe. You opened this issue, so I deduce that you care about the effectiveness of our testing. In e.g. https://gerrit.gromacs.org/#/c/5522/, https://gerrit.gromacs.org/#/c/5435/https://gerrit.gromacs.org/#/c/7736/, https://gerrit.gromacs.org/#/c/7735/ and other commits that hash-referenced this commit, I have been working on the necessary infrastructure before a replacement could "materialize." But over those several years, you wrote few-to-zero relevant patches, and didn't review any of mine. It would be nice to see you contribute constructively.

> (PS: and that some useful changes to evolve new machinery got lost in history can't always be blamed on "others"; noise and lack of concerted effort can equally be at fault)

You've been specifically requested to review some of them multiple times, here and elsewhere.

#33 - 07/19/2018 12:53 AM - Gerrit Code Review Bot

Gerrit received a related patchset '2' for Issue #1587.
Uploader: Mark Abraham (mark.j.abraham@gmail.com)
Change-Id: gromacs~master~Ib2e6cd26dbc58e29997ce408cee27917ede95ee9
Gerrit URL: https://gerrit.gromacs.org/8104

#34 - 10/04/2018 04:08 PM - Mark Abraham

- Target version set to 2020

I did a bunch of work here, but it won't land for 2019.

We need to move away from testing based on reproducing .tpr files stored in a separate repo, because it means we can't change the .tpr format gradually to implement key-value tree mdp options for multiple modules, because Jenkins would become unusable.

#35 - 01/09/2019 04:56 PM - Gerrit Code Review Bot

Gerrit received a related patchset '6' for Issue #1587.
Uploader: Mark Abraham (mark.j.abraham@gmail.com)
Change-Id: regressiontests~master~I3ebfbe4360a273766551ae66b8fe2c817f4c963c
Gerrit URL: https://gerrit.gromacs.org/7921

#36 - 01/10/2019 09:16 AM - Gerrit Code Review Bot

Gerrit received a related patchset '1' for Issue #1587.
Uploader: Mark Abraham (mark.j.abraham@gmail.com)
Change-Id: regressiontests~master~Ie4be0b64279c0671b2595e952d52845c10cd2e23
Gerrit URL: https://gerrit.gromacs.org/8947

#37 - 01/10/2019 11:04 AM - Gerrit Code Review Bot

Gerrit received a related patchset '1' for Issue #1587.
Uploader: Mark Abraham (mark.j.abraham@gmail.com)
Change-Id: gromacs~master~If7928825d74226a5b25ed3441e6b6dbdf004115a
Gerrit URL: https://gerrit.gromacs.org/8951

#38 - 01/10/2019 03:50 PM - Gerrit Code Review Bot

Gerrit received a related patchset '1' for Issue #1587.
Uploader: Mark Abraham (mark.j.abraham@gmail.com)
Change-Id: gromacs~release-2019~If7928825d74226a5b25ed3441e6b6dbdf004115a
Gerrit URL: https://gerrit.gromacs.org/8954

#39 - 01/14/2019 09:24 AM - Gerrit Code Review Bot

Gerrit received a related patchset '1' for Issue #1587.
Uploader: Mark Abraham (mark.j.abraham@gmail.com)
Change-Id: gromacs~master~I9871e32dfbde329ca4eceb718a94a7c8a418282a
Gerrit URL: https://gerrit.gromacs.org/8968

#40 - 01/17/2019 04:15 PM - Gerrit Code Review Bot

Gerrit received a related patchset '1' for Issue #1587.
Uploader: Mark Abraham (mark.j.abraham@gmail.com)
Change-Id: regressiontests~master~I4333535186e1c038ec6671aaaf721be6e5472d16
Gerrit URL: https://gerrit.gromacs.org/8999

#41 - 01/18/2019 06:17 AM - Gerrit Code Review Bot

Gerrit received a related patchset '1' for Issue #1587.
Uploader: Mark Abraham (mark.j.abraham@gmail.com)
Change-Id: gromacs~master~I6ea4fea10e01079bf9779b4127faa3f23a8d81d5
Gerrit URL: https://gerrit.gromacs.org/9001

#42 - 12/12/2019 01:54 PM - Mark Abraham

Another limitation of the regressiontests suite is the quite limited coverage of the update-groups path (needed for GPU update). Only 12 run them when there are 2 PP ranks, and most of those have "special" things running that are the kinds of things that don't combine well with other features. 19 run them with 1 PP rank. We should have some core test cases that are capable of running GPU update with and without DD on plain inputs (e.g. many of the complex/nbnxn_* tests, which currently use all-bonds constraints).

Below is the output from running gmxtest.pl with GMX_FORCE_UPDATE_DEFAULT_GPU (when that was needed for DD to run with GPU update) illustrating the problem:

```
update_groups_with_ntmpi_1.txt:complex/awh_multidim/md.log:Updating coordinates and applying constraints on th
e GPU.
update_groups_with_ntmpi_1.txt:complex/distance_restraints/md.log:Updating coordinates and applying constraint
s on the GPU.
update_groups_with_ntmpi_1.txt:complex/nbnxn-ljpme-geometric/md.log:Updating coordinates and applying constrai
nts on the GPU.
update_groups_with_ntmpi_1.txt:complex/nbnxn_pme/md.log:Updating coordinates and applying constraints on the G
PU.
update_groups_with_ntmpi_1.txt:complex/nbnxn_pme_order5/md.log:Updating coordinates and applying constraints o
n the GPU.
update_groups_with_ntmpi_1.txt:complex/nbnxn_pme_order6/md.log:Updating coordinates and applying constraints o
n the GPU.
update_groups_with_ntmpi_1.txt:complex/nbnxn_rf/md.log:Updating coordinates and applying constraints on the GP
U.
update_groups_with_ntmpi_1.txt:complex/nbnxn_rzero/md.log:Updating coordinates on the GPU.
update_groups_with_ntmpi_1.txt:complex/nbnxn-vdw-force-switch/md.log:Updating coordinates and applying constra
ints on the GPU.
update_groups_with_ntmpi_1.txt:complex/nbnxn-vdw-potential-switch-argon/md.log:Updating coordinates on the GPU
.
update_groups_with_ntmpi_1.txt:complex/nbnxn-vdw-potential-switch/md.log:Updating coordinates and applying con
straints on the GPU.
update_groups_with_ntmpi_1.txt:complex/nst_mismatch/md.log:Updating coordinates on the GPU.
update_groups_with_ntmpi_1.txt:complex/octahedron/md.log:Updating coordinates and applying constraints on the
GPU.
```

```
update_groups_with_ntmpi_1.txt:complex/position-restraints/md.log:Updating coordinates and applying constraint
s on the GPU.
update_groups_with_ntmpi_1.txt:complex/pr-vrescale/md.log:Updating coordinates and applying constraints on the
 GPU.
update_groups_with_ntmpi_1.txt:complex/pull_cylinder/md.log:Updating coordinates and applying constraints on t
he GPU.
update_groups_with_ntmpi_1.txt:complex/pull_geometry_angle-axis/md.log:Updating coordinates and applying const
raints on the GPU.
update_groups_with_ntmpi_1.txt:complex/pull_geometry_angle/md.log:Updating coordinates and applying constraint
s on the GPU.
update_groups_with_ntmpi_1.txt:complex/pull_geometry_dihedral/md.log:Updating coordinates and applying constra
ints on the GPU.
update_groups_with_ntmpi_2_npme_1.txt:complex/aminoacids/md.log:Using update groups, nr 523, average size 1.9
 atoms, max. radius 0.139 nm
update_groups_with_ntmpi_2_npme_1.txt:complex/awh_multidim/md.log:Using update groups, nr 10, average size 2.2
 atoms, max. radius 0.104 nm
update_groups_with_ntmpi_2_npme_1.txt:complex/nbnxn-free-energy/md.log:Using update groups, nr 516, average si
ze 2.9 atoms, max. radius 0.084 nm
update_groups_with_ntmpi_2_npme_1.txt:complex/nbnxn-free-energy-vv/md.log:Using update groups, nr 516, average
 size 2.9 atoms, max. radius 0.084 nm
update_groups_with_ntmpi_2_npme_1.txt:complex/octahedron/md.log:Using update groups, nr 1728, average size 3.0
 atoms, max. radius 0.124 nm
update_groups_with_ntmpi_2_npme_1.txt:complex/position-restraints/md.log:Using update groups, nr 523, average
 size 1.9 atoms, max. radius 0.139 nm
update_groups_with_ntmpi_2_npme_1.txt:complex/pull_geometry_angle-axis/md.log:Using update groups, nr 10, aver
age size 2.2 atoms, max. radius 0.104 nm
update_groups_with_ntmpi_2_npme_1.txt:complex/pull_geometry_angle/md.log:Using update groups, nr 10, average s
ize 2.2 atoms, max. radius 0.104 nm
update_groups_with_ntmpi_2_npme_1.txt:complex/pull_geometry_dihedral/md.log:Using update groups, nr 10, averag
e size 2.2 atoms, max. radius 0.104 nm
update_groups_with_ntmpi_2_npme_1.txt:freeenergy/expanded/md.log:Using update groups, nr 901, average size 3.0
 atoms, max. radius 0.103 nm
update_groups_with_ntmpi_2_npme_1.txt:freeenergy/simtemp/md.log:Using update groups, nr 897, average size 3.0
 atoms, max. radius 0.153 nm
update_groups_with_ntmpi_2_npme_1.txt:freeenergy/transformAtoB/md.log:Using update groups, nr 901, average siz
e 3.0 atoms, max. radius 0.103 nm
update_groups_with_ntmpi_2.txt:complex/aminoacids/md.log:Using update groups, nr 523, average size 1.9 atoms,
 max. radius 0.139 nm
update_groups_with_ntmpi_2.txt:complex/awh_multidim/md.log:Using update groups, nr 10, average size 2.2 atoms,
 max. radius 0.104 nm
update_groups_with_ntmpi_2.txt:complex/nbnxn-free-energy/md.log:Using update groups, nr 516, average size 2.9
 atoms, max. radius 0.084 nm
update_groups_with_ntmpi_2.txt:complex/nbnxn-free-energy-vv/md.log:Using update groups, nr 516, average size 2
.9 atoms, max. radius 0.084 nm
update_groups_with_ntmpi_2.txt:complex/octahedron/md.log:Using update groups, nr 1728, average size 3.0 atoms,
 max. radius 0.124 nm
update_groups_with_ntmpi_2.txt:complex/position-restraints/md.log:Using update groups, nr 523, average size 1.
9 atoms, max. radius 0.139 nm
update_groups_with_ntmpi_2.txt:complex/pull_geometry_angle-axis/md.log:Using update groups, nr 10, average siz
e 2.2 atoms, max. radius 0.104 nm
update_groups_with_ntmpi_2.txt:complex/pull_geometry_angle/md.log:Using update groups, nr 10, average size 2.2
 atoms, max. radius 0.104 nm
update_groups_with_ntmpi_2.txt:complex/pull_geometry_dihedral/md.log:Using update groups, nr 10, average size
 2.2 atoms, max. radius 0.104 nm
update_groups_with_ntmpi_2.txt:freeenergy/expanded/md.log:Using update groups, nr 901, average size 3.0 atoms,
 max. radius 0.103 nm
update_groups_with_ntmpi_2.txt:freeenergy/simtemp/md.log:Using update groups, nr 897, average size 3.0 atoms,
 max. radius 0.153 nm
update_groups_with_ntmpi_2.txt:freeenergy/transformAtoB/md.log:Using update groups, nr 901, average size 3.0 a
toms, max. radius 0.103 nm
```

Most cases use FEP, or enhanced sampling, or restraints, and some of those don't yet run in combination with newer implementations like GPU update.

We should at least convert nbnxn_[pme|rf]* to not use all-bonds constraints.

#43 - 12/20/2019 12:02 PM - Paul Bauer

- Target version changed from 2020 to 2021

I think that might be easier to be done in Gitlab in the future