GROMACS - Task #1781
re-design benchmarking functionality
07/16/2015 12:19 AM - Szilárd Páll

Status: Accepted
Priority: Normal
Assignee: Mark Abraham
Category: core library
Target version: future
Difficulty: uncategorized

Description
mdrun counter reset does not interact well with PME tuning. We should probably delay counter reset until after tuning completes, and have a better interface, e.g. mdrun -benchmarksteps 1000 and/or mdrun -benchmarktime 0.05

See discussion beginning at comment 6.

Original issue follows:

The following PME LB error can be triggered by some runs:

NOTE: DLB will not turn on during the first phase of PME tuning

starting mdrun 'Water'
1000 steps, 2.0 ps.
step 80: timed with pme grid 200 100 100, coulomb cutoff 1.000: 1989.6 M-cycles
step 160: timed with pme grid 168 84 84, coulomb cutoff 1.187: 2006.8 M-cycles
step 240: timed with pme grid 144 72 72, coulomb cutoff 1.384: 2664.3 M-cycles
step 320: timed with pme grid 160 80 80, coulomb cutoff 1.246: 2205.9 M-cycles
step 400: timed with pme grid 168 84 84, coulomb cutoff 1.187: 2016.8 M-cycles
step 480: timed with pme grid 192 96 96, coulomb cutoff 1.038: 2097.2 M-cycles
optimal pme grid 200 100 100, coulomb cutoff 1.000
step 500: resetting all time and cycle counters

Program gmx mdrun, VERSION 5.1-rc1
Source code file: /var/data0/sandbox/gromacs/bmdir/data/source/src/gromacs/ewald/pme-load-balancing.cpp, line: 927

Software inconsistency error:
pme_loadbal_do called at an interval != nstlist
For more information and tips for troubleshooting, please check the GROMACS website at http://www.gromacs.org/Documentation/Errors

Program gmx mdrun, VERSION 5.1-rc1
Source code file: /var/data0/sandbox/gromacs/bmdir/data/source/src/gromacs/ewald/pme-load-balancing.cpp, line: 927

Software inconsistency error:
pme_loadbal_do called at an interval != nstlist
For more information and tips for troubleshooting, please check the GROMACS website at http://www.gromacs.org/Documentation/Errors

Detected by the NVIDIA Perflab team, they note that:

I've also noticed that unless this line pops up
NOTE: DLB can now turn on, when beneficial
Before
step 500: resetting all time and cycle counters
Then I get the pme_loadbal_do error.

Original log and standard output attached, input is 384k water box (but that is likely not relevant).

### Subtasks:

<table>
<thead>
<tr>
<th>Task #</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>2495</td>
<td>replace -noconfout with mdp option</td>
</tr>
</tbody>
</table>

### Related Issues:

<table>
<thead>
<tr>
<th>Related to</th>
<th>Issue Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GROMACS</td>
<td>Bug #1870: pme_loadbal_do called at an interval != nstlist</td>
</tr>
<tr>
<td>GROMACS</td>
<td>Bug #2131: mdrun hangs upon &quot;-nsteps=&quot; or &quot;-maxh&quot; triggering</td>
</tr>
<tr>
<td>GROMACS</td>
<td>Bug #2041: mdrun -resetstep can finish too early</td>
</tr>
<tr>
<td>GROMACS</td>
<td>Task #1971: Removing buggy features vs. keeping workflows</td>
</tr>
<tr>
<td>GROMACS</td>
<td>Bug #1777: Teach mdrun about explicit -append</td>
</tr>
<tr>
<td>GROMACS</td>
<td>Bug #2136: mdrun -noconfout checkpointing issue</td>
</tr>
<tr>
<td>GROMACS</td>
<td>Feature #2224: Proposed feature: conditional stop</td>
</tr>
<tr>
<td>GROMACS</td>
<td>Bug #2360: error at counter reset with PME-only rank</td>
</tr>
<tr>
<td>GROMACS</td>
<td>Task #2569: announce deprecations in GROMACS 2019</td>
</tr>
<tr>
<td>GROMACS</td>
<td>Bug #2717: mdrun runs infinitely when checkpoint file is...</td>
</tr>
<tr>
<td>GROMACS</td>
<td>Bug #2757: mdrun refuses to start with .cpt if nsteps is...</td>
</tr>
</tbody>
</table>

### Associated revisions

**Revision 785aad1a - 08/11/2015 01:51 PM - Mark Abraham**

Abort if PME tuning is active and counters reset

Triggering counter reset (in various ways) could happen at a non-nstlist step, which provokes a software inconsistency error in 5.1. This is reveals that all recent releases have permitted reset while tuning was active, which is useless and potentially wrong.

Introduced a getter pme_loadbal_is_active, so that the fatal error can be issued when conditions for counter reset are satisfied and PME load balancing is still active.

Noted a TODO to have the load-balancing module use its own getter in future; such a refactoring is probably fine, but worth avoiding in a bugfix branch. Noted a TODO to make a counter-reset module, consider alternative solutions to #1781, and other clean-up. Documented some stuff.

Fixes #1781

Change-Id: I912e3da837bd32280f295ad98cc6b88170f4d2d81

**Revision 7252de1a - 11/16/2016 05:59 PM - Carsten Kutzner**

Increased the default reset step to 1500 in gmx tune_pme

Commit 785aad1a introduced a gmx_fatal() in mdrun for cases where cycle counters are reset when PME tuning is still active. In almost all cases, tuning takes longer than 100 steps (which was the default at which gmx tune_pme would request mdrun to reset its counters). This leads to gmx tune_pme reporting that all the runs failed. Note that the small default of 100 steps was from times where there was only DLB to account for, but not PME tuning.

With the increased default, this should happen only very rarely.

For future versions it would be nicer to implement a "-benchmarksteps" command line parameter for mdrun which resets counters exactly after PME tuning is finished and then performs a requested number of benchmark MD steps. Refs #1781

Change-Id: lcbce1eccc8a23d35302c04c9a6be13c06b1be8c8

**Revision f5cb6c13 - 01/11/2018 01:41 AM - Mark Abraham**

Announce in user log files that features are deprecated.
These are merely informational notes, not warnings or errors.

Refs #1781, #1971, #2136
Change-Id: I96e19acb0e15d3f42b0929f555b451299a2882e4

Revision 1aa4fa40 - 08/22/2018 03:59 PM - Mark Abraham
Fix handling of counter resets

There is no reason for or need to change max_hours, ir->nsteps or step_rel when doing a counter reset.

This makes clear that the behaviour for the combination

mdrun -maxh t -resetstep n

matches the documentation of -maxh.

Updated the API for walltime_acounting and its usage, because elapsed time is an insufficiently clear context. Changed the names of the start and stop functions so that no callers can silently rely on semantics that have changed.

Avoided variables such as elapsed_time and max_hours, which were insufficiently precisely worded.

Refs #1781
Change-Id: I16c96985f43a7b4ac75b94f378da3d05914d6986

Revision cf2d8336 - 10/13/2018 09:53 PM - Mark Abraham
Deprecate various functionality in GROMACS 2019

Published a deprecation policy.

Updated the release notes to refer also to previously deprecated features.

Announced intent to change some functionality:

- gmx mdrun -membed options (but not feature)
- gmx mdrun -rerun option (but not feature)
- integrator .mdp field will contain only integrators
- gmx do_dssp to be replaced by gmx dssp
- gmx trjconv and friends to be split and rewritten

List of newly deprecated functionality:

- conversion of aromatic rings to virtual sites
- gmx mdrun -table options (but not feature)
- gmx mdrun -gcom option and feature
- gmx mdrun -nsteps option and feature
- gmx mdrun -nsteps -resetstep -resetstepway moved to a gmx benchmark tool
- gmx mdrun -confout removed

Also updated release notes for functionality removed in GROMACS 2019.

Refs #2495, #1781
Fixes #2569, #1925
Change-Id: I1d00859d0f15409a472984f5a65347a50c71ad17

History
#1 - 07/16/2015 03:05 AM - Szilárd Páll
- Status changed from New to Accepted
- Priority changed from Normal to Low
- Affected version - extra info set to 4.6.x, 5.0.x

This seems to be an old issue that gets caught due to the newly introduced sanity check. If the cycle counter resetting does not happen at an NS
step, it can reset the counters while the PP-PME LB is collecting measurements. This is problematic because it could result in measurements of << nstlist step used in the tuning. One solution is delaying resetting so it does not happen during PP-PME LB timing collection. This would result in not resetting exactly halfway, but that's I guess better than trashing the load balancer's data.

#2 - 08/08/2015 01:11 AM - Gerrit Code Review Bot
Gerrit received a related patchset '1' for Issue #1781.
Uploader: Mark Abraham (mark.j.abraham@gmail.com)
Change-id: I30e21749d0b5af376de1aa91e43a0c2e8f2af6f
Gerrit URL: https://gerrit.gromacs.org/4964

#3 - 08/08/2015 01:28 AM - Mark Abraham
Szilárd Páll wrote:

This seems to be an old issue that gets caught due to the newly introduced sanity check. If the cycle counter resetting does not happen at an NS step, it can reset the counters while the PP-PME LB is collecting measurements. This is problematic because it could result in measurements of << nstlist step used in the tuning. One solution is delaying resetting so it does not happen during PP-PME LB timing collection. This would result in not resetting exactly halfway, but that's I guess better than trashing the load balancer's data.

My uploaded fix prevents counter reset until after tuning completes. In theory, with my patch, tuning might then delay counter reset so much that it doesn't happen, or that there are so few steps the subsequent timing measurement isn't accurate, but in those cases the performance data was already unreliable (because tuning time was included).

In future, I think we should replace the several reset methods with gmx mdrun -benchmark 1000, which implements doing 1000 steps after tuning completes, and perhaps after DBL stabilizes also. This will be simpler and seems like it will produce more reliable numbers more efficiently, particularly in non-expert hands.

I could also see gmx mdrun -benchmarktime 100 to run 100 post-stabilization seconds being useful for strong-scaling comparisons (because otherwise you have to choose in advance to run a number of steps that is meaningful at the strong scaling limit, which takes a uselessly long time when using small amounts of that hardware). Is that useful enough to warrant the second flag?

#4 - 08/08/2015 06:31 PM - Szilárd Páll
Mark Abraham wrote:

My uploaded fix prevents counter reset until after tuning completes. In theory, with my patch, tuning might then delay counter reset so much that it doesn't happen, or that there are so few steps the subsequent timing measurement isn't accurate, but in those cases the performance data was already unreliable (because tuning time was included).

Sounds good, I'll have a look alter.

In future, I think we should replace the several reset methods with gmx mdrun -benchmark 1000, which implements doing 1000 steps after tuning completes, and perhaps after DBL stabilizes also. This will be simpler and seems like it will produce more reliable numbers more efficiently, particularly in non-expert hands.

I could also see gmx mdrun -benchmarktime 100 to run 100 post-stabilization seconds being useful for strong-scaling comparisons (because otherwise you have to choose in advance to run a number of steps that is meaningful at the strong scaling limit, which takes a uselessly long time when using small amounts of that hardware). Is that useful enough to warrant the second flag?

Some kind of built-in benchmarking options could be useful, but the assumption of static PP-PME load balancing is something that does allpy for current versions, but it will likely not apply in the future. Additionally, I often want more fine-grained control over the resetting because it's quite easy to get fake performance on hardware that's prone of heat up and throttle if resetting to early.

#5 - 08/10/2015 09:30 AM - Berk Hess
Note that this is a new issue. In 5.0 and before pme_load_balance was only called inside a conditional (step % ir->nstlist == 0).

I would suggest to replace the gmx_incons call by a return (this was in my original code, but Szilard suggested to replace it by an error since the condition should never occur in correct code).

#6 - 08/10/2015 04:40 PM - Mark Abraham

- Subject changed from inconsistency error in PME load balancing to inconsistency error in PME load balancing
- Assignee set to Mark Abraham
- Target version set to 5.1
- Affected version - extra info deleted (4.6.x, 5.0.x)

I need to do some testing still, but https://gerrit.gromacs.org/#/c/4964/3 is likely sufficient to remove the problem here. The situation still isn't good (you shouldn't be able to do a counter reset with tuning active) but it's not vital to fix.
Note to self https://gerrit.gromacs.org/#/c/4964/2 has the kind of content to suggest for master branch, once things are stable and merged up.

#7 - 08/10/2015 07:11 PM - Szilárd Páll

Berk Hess wrote:

Note that this is a new issue. In 5.0 and before pme_load_balance was only called inside a conditional (step % ir->nstlist == 0).

Good point.

I would suggest to replace the gmx_incons call by a return (this was in my original code, but Szilard suggested to replace it by an error since the condition should never occur in correct code).

But that will result in possibly incorrect load balancing. Exactly because we often want to run as short as possible, but the PP-PME balancing may take a long time due to jitter, clock scaling/throttling, etc. I think it's best to prevent incorrect reset.

#8 - 08/10/2015 07:58 PM - Mark Abraham

Szilárd Páll wrote:

Berk Hess wrote:

I would suggest to replace the gmx_incons call by a return (this was in my original code, but Szilard suggested to replace it by an error since the condition should never occur in correct code).

But that will result in possibly incorrect load balancing.

Yes, it might lead to possibly incorrect load balancing. My understanding is that it is no worse than in 5.0. Am I wrong?

Exactly because we often want to run as short as possible, but the PP-PME balancing may take a long time due to jitter, clock scaling/throttling, etc. I think it's best to prevent incorrect reset.

In all recent releases, it was possible to do a premature reset, so that aspect is not a new bug to fix for 5.1. As reflected in update to the gerrit patch set 3, we can change that mdrun behaviour in master branch.

#9 - 08/10/2015 08:29 PM - Szilárd Páll

Mark Abraham wrote:

Szilárd Páll wrote:

Berk Hess wrote:

I would suggest to replace the gmx_incons call by a return (this was in my original code, but Szilard suggested to replace it by an error since the condition should never occur in correct code).

But that will result in possibly incorrect load balancing.

Yes, it might lead to possibly incorrect load balancing. My understanding is that it is no worse than in 5.0. Am I wrong?

Does that justify leaving a bug in 5.1?

Exactly because we often want to run as short as possible, but the PP-PME balancing may take a long time due to jitter, clock scaling/throttling, etc. I think it's best to prevent incorrect reset.

In all recent releases, it was possible to do a premature reset, so that aspect is not a new bug to fix for 5.1. As reflected in update to the gerrit patch set 3, we can change that mdrun behaviour in master branch.

Sorry, I'm confused here. I'm not sure how does this relate to the point I made above. Performance benchmarks can easily produce invalid results, and regardless of whether this affects v3.3 or 4.5 or whatever, it's quite helpfule to prevent that from happening in the latest release which will likely be the most benchmarked version in the near future.

#10 - 08/10/2015 08:58 PM - Mark Abraham

Szilárd Páll wrote:
Berk Hess wrote:  
I would suggest to replace the gmx_incons call by a return (this was in my original code, but Szilard suggested to replace it by an error since the condition should never occur in correct code).

But that will result in possibly incorrect load balancing.

Yes, it might lead to possibly incorrect load balancing. My understanding is that it is no worse than in 5.0. Am I wrong?

Does that justify leaving a bug in 5.1?

A mis-design present in both 4.6 and 5.0 that AFAIK hasn't been discussed ever is not a last-minute blocker for 5.1. The fix I proposed at patch set 2 touches a couple of points of the main MD loop. I'm confident in it (or I would not have pushed it), but it needs more testing and review than patch set 3 does, I think. The fatal error possibilities suggested at the gerrit patch seem better, though.

Exactly because we often want to run as short as possible, but the PP-PME balancing may take a long time due to jitter, clock scaling/throttling, etc. I think it's best to prevent incorrect reset.

In all recent releases, it was possible to do a premature reset, so that aspect is not a new bug to fix for 5.1. As reflected in update to the gerrit patch set 3, we can change that mdrun behaviour in master branch.

Sorry, I'm confused here. I'm not sure how does this relate to the point I made above.

It doesn't relate. There are two points. mdrun -reset gear and PME load balancing have always interacted unfavorably. That doesn't have to be fixed now. The other point is the the new sanity check you suggested has been useful to observe both the old problem and one introduced in the refactoring. We need a fix for the latter, which I think might be patch set 3. The plus side of fixing the former is that we don't have to fix the latter.

Performance benchmarks can easily produce invalid results, and regardless of whether this affects v3.3 or 4.5 or whatever, it's quite helpful to prevent that from happening in the latest release which will likely be the most benchmarked version in the near future.

Yes, we agree that fixing this is helpful. The question is when and how much work that adds for testing right now.

#11 - 08/10/2015 10:47 PM - Szilárd Páll
Mark Abraham wrote:

A mis-design present in both 4.6 and 5.0 that AFAIK hasn't been discussed ever is not a last-minute blocker for 5.1.

OK. I never said it was blocker. The bugfix can be delayed for 5.1.1 which I would find quite reasonable. Not so much trying to erase the issue from the books.

Should we do that? Is this really the last thing blocking the release?

Performance benchmarks can easily produce invalid results, and regardless of whether this affects v3.3 or 4.5 or whatever, it's quite helpful to prevent that from happening in the latest release which will likely be the most benchmarked version in the near future.

Yes, we agree that fixing this is helpful. The question is when and how much work that adds for testing right now.

Good that we agree.

#12 - 08/11/2015 01:00 AM - Mark Abraham
Szilárd Páll wrote:

Mark Abraham wrote:

A mis-design present in both 4.6 and 5.0 that AFAIK hasn't been discussed ever is not a last-minute blocker for 5.1.

OK. I never said it was blocker.
In comment 9 you implied it would be a bad idea to release 5.1 with the potential for improper load balancing if reset occurs. If you don't want the release made, that sounds to me like you think it's a blocker...

The bugfix can be delayed for 5.1.1 which I would find quite reasonable. Not so much trying to erase the issue from the books.

Szilard, I have commented both here and on gerrit that I have no intention of "erasing the issue from the books." Please read and think before you write, and re-read before you post - particularly when your choice of words might imply things about others' motives. :-) "I'm concerned we might forget to fix this later" would have been a much less antagonistic way to raise your concerns.

Should we do that? Is this really the last thing blocking the release?

The time before the release is not infinitely extensible. Only one part of this issue is a high priority to fix. The rest should be resolved in favour of stability and minimizing any required testing. I've done pretty much nothing but release testing and management since March, and I am drawing the line.

#13 - 08/11/2015 01:02 AM - Gerrit Code Review Bot
Gerrit received a related patchset '1' for Issue #1781. uploader: Mark Abraham (mark.j.abraham@gmail.com)
change-id: I912e3da837bd32280f295ad98cc6b8170f4d2d81
Gerrit URL: https://gerrit.gromacs.org/4974

#14 - 08/11/2015 05:03 AM - Szilárd Páll
Mark Abraham wrote:

Szilárd Páll wrote:

Mark Abraham wrote:

A mis-design present in both 4.6 and 5.0 that AFAIK hasn't been discussed ever is not a last-minute blocker for 5.1.

OK. I never said it was blocker.

In comment 9 you implied it would be a bad idea to release 5.1 with the potential for improper load balancing if reset occurs. If you don't want the release made, that sounds to me like you think it's a blocker...

The bugfix can be delayed for 5.1.1 which I would find quite reasonable. Not so much trying to erase the issue from the books.

Szilard, I have commented both here and on gerrit that I have no intention of "erasing the issue from the books." Please read and think before you write, and re-read before you post - particularly when your choice of words might imply things about others' motives. :-) "I'm concerned we might forget to fix this later" would have been a much less antagonistic way to raise your concerns.

OK. Sorry for the assumptions, I have admittedly overreacted due to the recent experience of bugs getting converted to feature requests and fixes being questioned without much discussion. The wording ("that aspect is not a new bug to fix for 5.1") seemed to suggest that there is no room for such a fix in the 5.1.x series.

Should we do that? Is this really the last thing blocking the release?

The time before the release is not infinitely extensible. Only one part of this issue is a high priority to fix. The rest should be resolved in favour of stability and minimizing any required testing.

I changed the priority to "low" at the very beginning indicating that even a fix for the inconsistency error was something I did not consider urgent. So I thought it was obvious that I did not want to pose this report as a blocker for the 5.1.0 release.

I've done pretty much nothing but release testing and management since March, and I am drawing the line.

I am aware, you may have noticed that I too have spent much more time with testing and fixing things than I should have.

#15 - 08/11/2015 01:59 PM - Mark Abraham
Szilárd Páll wrote:

Mark Abraham wrote:
Szilárd Páll wrote:

Mark Abraham wrote:

A mis-design present in both 4.6 and 5.0 that AFAIK hasn't been discussed ever is not a last-minute blocker for 5.1.

OK. I never said it was blocker.

In comment 9 you implied it would be a bad idea to release 5.1 with the potential for improper load balancing if reset occurs. If you don't want the release made, that sounds to me like you think it's a blocker...

The bugfix can be delayed for 5.1.1 which I would find quite reasonable. Not so much trying to erase the issue from the books.

Szilard, I have commented both here and on gerrit that I have no intention of "erasing the issue from the books." Please read and think before you write, and re-read before you post - particularly when your choice of words might imply things about others' motives. ;-) "I'm concerned we might forget to fix this later" would have been a much less antagonistic way to raise your concerns.

OK. Sorry for the assumptions, I have admittedly overreacted due to the recent experience of bugs getting converted to feature requests and fixes being questioned without much discussion. The wording ("that aspect is not a new bug to fix for 5.1") seemed to suggest that there is no room for such a fix in the 5.1.x series.

No problem. The existence of not-perfect behaviour is not itself evidence of the need for an immediate fix. Stability and usability are also important. A change such as https://gerrit.gromacs.org/#/c/4974/4 we might have done in 5.1.1, but my earlier effort at https://gerrit.gromacs.org/#/c/4964/2 was too risky. Such discussions get confused when there are multiple problems and people don't always think and write precisely as they would like to! ;-) Should we do that? Is this really the last thing blocking the release?

The time before the release is not infinitely extensible. Only one part of this issue is a high priority to fix. The rest should be resolved in favour of stability and minimizing any required testing.

I changed the priority to "low" at the very beginning indicating that even a fix for the inconsistency error was something I did not consider urgent. So I though it was obvious that I did not meat to pose this report as a blocker for the 5.1.0 release.

OK. Independent reports of problems from Carsten (in tune_pme in the final-days thread on gmx-dev) and Nvidia seemed to me to make it worth fixing something. Changes affecting the main MD loop are risky and the solution soon to be merged is a good compromise, since normal MD is provably unaffected.

I've done pretty much nothing but release testing and management since March, and I am drawing the line.

I am aware, you may have noticed that I too have spent much more time with testing and fixing things than I should have.

Indeed, I have noticed. This is part of why we plan some changes in process for the next version, so that we do not have such large problems with people stressed over multiple issues for long periods.

#16 - 08/11/2015 03:30 PM - Mark Abraham
- Status changed from Accepted to Resolved
- % Done changed from 0 to 100

Applied in changeset 785aad1a1d548261b9a32cfbd1468ae271a8a3c3.

#17 - 08/13/2015 11:32 AM - Mark Abraham
- Status changed from Resolved to Closed

#18 - 08/13/2015 11:37 AM - Mark Abraham
- Tracker changed from Bug to Task
- Subject changed from inconsistency error in PME load balancing to re-design benchmarking functionality
- Description updated
- Status changed from Closed to Accepted
- Assignee deleted (Mark Abraham)
We've been encountering this problem frequently on 4-GPU runs (serial failures with multiple jobs); rerunning on CPU sometimes fixes, running on a different architecture with dual-GPU seems always to work. It's turning into a major PITA. Any suggestions for a workaround hack?

Also bug #1870 appears to be another user encountering this.

Thanks!

PS does just removing the check as in https://gerrit.gromacs.org/#/c/4964/3 take care of this? If so, we'll put that into our local version ASAP. This is blocking for us.

Are you doing very short runs? Could you delay the reset until the load balancing has most likely been finished?

I tried to reproduce with the tpr at #1870 but so far have not succeeded, and am not sure whether we even have access to any 4-gpu nodes. Sharing a tpr and both successful and failing log files would be useful, of course.

The issue is that the bNSList = (ir->nstlist > 0 && step % ir->nstlist == 0);, but after a restart with a different nstlist, this will lead to the load balancer called after nstlist_old-nstlist_new steps. Unless I'm mistaken, the aforementioned conditional should use the relative step counter rather than the absolute one.

As noted on the gerrit issue, there are two possible trigger conditions (that I can see) for #1870 and Peter's issues. In general, the error can occur whenever step_start % nstlist != 0 which can happen either when nstlist changes between runs and therefore the first trigger could happen e.g. when step_rel nstlist_new-nstlist_old, but also when the continued run did not stop at a search step.

The latter could actually be the intended behavior - although it does not seem to make much sense because search anyway needs to be done at step_rel 0, so it is not always possible to retain the periodicity of the search across continuations.

Regarding the particular crashes observed by Peter and the OP of #1870, these most likely hit the second, restart from non-search step case (verified in Peter's log files I got off-redmine).

(Side-note: I was trying to reproduce the restart from non-search step and was actually unable to do that, it seems that the signal-based non-search step interruption does not for me.)

Ah, that makes sense if we're hitting this with non-search-step restarts so things get out of sync. It is an interesting conflict of the desire to have binary-reproducible runs with continuation vs. consistency. My $.02 would be that continuations could sacrifice exactly keeping the search step periodicity if that maintains consistency.
My opinion (and what I think the current code is doing) is that we should always do search at absolute step % nstlist == 0. The rest of the code will have to adapt to this. If you quit a run at a non-nstlist step or change nstlist, you simply get an extra search.

I was not comfortable with adding the assertion in pme_loadbal (upon Szilard's suggestion). We can simply remove it.

Berk Hess wrote:

My opinion (and what I think the current code is doing) is that we should always do search at absolute step % nstlist == 0. The rest of the code will have to adapt to this.

Can you explain why is this better than doing it at step_rel % nstlist?

If you quit a run at a non-nstlist step or change nstlist, you simply get an extra search.

And right now you also get a possibly quite incorrect first PP-PME load balancer timing. That's the issue we should solve.

I was not comfortable with adding the assertion in pme_loadbal (upon Szilard's suggestion). We can simply remove it.

...and simply removing the assert does not solve it, right?

Berk Hess wrote:

My opinion (and what I think the current code is doing) is that we should always do search at absolute step % nstlist == 0. The rest of the code will have to adapt to this.

Can you explain why is this better than doing it at step_rel % nstlist?

If you quit a run at a non-nstlist step or change nstlist, you simply get an extra search.

And right now you also get a possibly quite incorrect first PP-PME load balancer timing. That's the issue we should solve.

I was not comfortable with adding the assertion in pme_loadbal (upon Szilard's suggestion). We can simply remove it.

...and simply removing the assert does not solve it, right?

#30 - 09/27/2016 06:05 PM - Gerrit Code Review Bot

Gerrit received a related patchset '2' for Issue #1781.
Uploader: Carsten Kutzner (ckutzne@gwdg.de)
Change-Id: Icbcce1ecc8a23d35302c04c9e13c06b1be8c8
Gerrit URL: https://gerrit.gromacs.org/6214

Mark Abraham wrote:

Further, I think that implementing something like mdrun -benchmarksteps n will remove most of the developer-based interest in mdrun -nsteps n

Mark Abraham wrote:

Further, I think that implementing something like mdrun -benchmarksteps n will remove most of the developer-based interest in mdrun -nsteps n

Sure, most benchmarking needs will be met by such a feature (but certainly not all, we use -nsteps for profiling, and other cases too). This could also allow us to make sure that re-balancing or DLB re-triggering plays well with benchmarking.

One of my objectives here is to return to the .mdp/.tpr settings as the source of how long the simulation will try to run. tpbconv -extend and mdrun -maxh are well designed ways to manage that. Already at least #1942 has users using mdrun -nsteps because they can, and it is infeasible for us to maintain multiple ways of doing things.

My vision for mdrun -benchmarksteps -n is that we do n steps after performance stabilizes (e.g. as measured by drift over variation) and report for those n, so that we have something comparable for reporting in benchmarking studies. Of course, the only user who'd use that is one doing benchmarking.

I take your point, from a programmer's pov perhaps it is an unnecessary complexity and bug-prone behavior to allow setting the length of a production simulation in two ways. However, I would argue that the simulation length is by design a free variable that should be easy to change without having to regenerate a binary input file -- which is especially irritating on a cluster where all you will often have build only a compute-node compatible build. I've no strong feelings, but one bug report does not seem to be a solid argument that this is a dangerous flexibility.

However, more importantly, what I'm certain about is that there is no single way to do performance measurement. In a subset of cases it may suffice to have some magic benchmark mode decide from when and how long to measure in a benchmark run, but in general that's not the case. That one recipe will likely not work well for profiling or other dev-oriented performance measurement.

So please let's talk about how the devs want to be able to control the run length and measurement/performance counter accumulation and resetting before making (or planning) changes.

Szilárd Páll wrote:

I take your point, from a programmer's pov perhaps it is an unnecessary complexity and bug-prone behavior to allow setting the length of a
Indeed. The entire structure of do_md is broken, and trying hack solutions for all things for all cases is part of how we got there. It has to stop and has to be reversed.

However, I would argue that the simulation length is by design a free variable that should be easy to change without having to regenerate a binary input file.

Choosing the length of the simulation is a significant part of MD study design, and making nsteps easy to change reduces the extent to which the input files document what will happen, and how to reproduce what did happen. Already we've seen on gmx-users a post asking why mdrun -nstxout isn't implemented to enable them to suppress what they thought of as minimization per-step terminal spam, given that mdrun -nsteps is implemented. And I do not want to deal with bug reports where they give you a .tpr but do not tell you the set of things they changed on the command line.

-- which is especially irritating on a cluster where all you will often have build only a compute-node compatible build.

That's simply a broken installation. The latest install guide mentions explicitly how cluster installs should work, and we can only design for convenience things if it doesn't compromise function and usability of important things.

I've no strong feelings, but one bug report does not seem to be a solid argument that this is a dangerous flexibility.

There are multiple dangers. This just one of several ways the end of the run is organized, and that has led to years of bug reports that termination, checkpointing, appending, restarting or extending doesn't work reliably in this or that corner case - #1942 is just the start. Someone implementing mdrun -nsteps needs to design for the behaviour that replica-exchange should never miss nor repeat an exchange attempt (and vice versa).

Someone implementing mdrun -resetstep needs to design for PME tuning not to get broken (and vice versa).

However, more importantly, what I'm certain about is that there is no single way to do performance measurement. In a subset of cases it may suffice to have some magic benchmark mode decide from when and how long to measure in a benchmark run, but in general that's not the case.

That one recipe will likely not work well for profiling or other dev-oriented performance measurement.

Please start thinking about what is :-) I'm pretty sure that mdrun -nsteps -resetstep is a fairly poor substitute for anything that is useful (especially when you have to scale them with particle or node count in order to get reportable quantities, while also imposing minima for things like fixed PME tuning phases), and rather inferior to mdrun -benchmarksteps n for the kind of performance reporting the majority of vendors, sysadmins and GROMACS paper/deliverable writers need to do. Even mdrun -benchmark that returns the average performance once the relative standard error is below some quantity is lots better than we have now.

I'm happy to consider support for anything that's useful for important subsets of people, but we can't have different subsets compromising usability for the other. Whether we need something fancy for profiling (e.g. environment variable GMX_PROFILING_START_STEP) is not a question we have to resolve before we do something useful to address the first-order problems.

So please let's talk about how the devs want to be able to control the run length and measurement/performance counter accumulation and resetting before making (or planning) changes.

Sure - but so far all I'm proposing is the same kind of thing as you proposed when you opened this issue. Things like waiting for hardware to warm up before measuring performance affect production simulation users too, which is part of why we need to redesign these things on new principles.

Waiting for performance to stabilize before starting any tuning or load balancing sounds like a useful thing - and can even serve as an initial hint about how conservative the tuning needs to be.

We need to actually do something here. We have now introduced an additional bug for normal users doing normal runs (#2131) while fixing issues with implementing features only developers would use (#2041). This is in addition to #1870, so one of Szilard's observations in comment 34 is eroding (even though strictly -nsteps and -reset* are different options, both exist to suit kinds of performance testing).

History shows that it is too complicated for us to implement and test multiple ways of setting simulation length, ending simulations and resetting counters for performance testing, while handling the concerns of PME autotuning and separate PME ranks. This will get even worse over time.

For GROMACS 2017:
- I will implement that environment variables GMX_BENCHMARKDELAY=m and GMX_BENCHMARKLENGTH=n will wait for any PME tuning to complete, run m steps, reset counters, run n steps, report timings, and exit. This seems simple to implement, and at least as useful as any of the combinations of existing options, and sometimes more useful in practice. Using environment variables makes clear that we are not intending users to manage their simulations with it. Using it in benchmarking scripts is no harder than using a command-line argument.
- I will remove mdrun -resetstep and mdrun -resethway, which will be superseded by the above
- mdrun -maxh will be retained (it probably works in all combinations that are sensible, and in most that aren't)
- I will remove mdrun -nsteps. This was added in the big Verlet-scheme commit (I assume, to suit performance testing), apparently without enough consideration then, or enough re-consideration as we added more functionality like PME auto-tuning. We have gmx convert-tpr -nsteps if developers need to do some hacking while fixing bugs and testing fixes. We only documented it in mdrun -h (GROMACS 5.1 did give it one mention in the user guide), so I'm comfortable with saying that any user who is currently using it needs to change their scripts to use a properly supported path (which we should document in the user guide if we haven't already).

Noting Szillard's request for further discussion, in the future, we can reconsider more complex benchmarking behaviours. The names of the above variables are hopefully stable and useful even if we change the implementation, e.g. to interpret GMX_BENCHMARKDELAY as a minimum delay period.

### #39 - 03/07/2017 06:17 PM - Roland Schulz

Overall this is a great proposal but I disagree on a few points.

I disagree with using environment variables. If the command line argument is called benchmarkdelay/benchmarklength (or benchmark_delay) you achieve the same thing that it is clear that it should be only used for benchmarking. You can even make it a hidden option if you don't want to confuse normal users. Unless you plan to also update the environment variable handling code, we should not use environment variables meant for users. And sysadmins doing benchmarking aren't developers either. The problems with environment variables is that we don't have proper error handling. If you misspell the option or use an invalid argument you don't get a proper error message.

There is one case the new suggested options don't cover. The number of steps of PME tuning aren't reproducible. If you need a reproducible long run for benchmarking (e.g. to measure power consumption), you won't get that with the new options (you could use convert-tpr for length but you would still need resetstep) because of the first part being non-reproducible. It was suggested in some other redmine issue (I can't find it) that we should have an option to set the tuned fourier-spacing (or rlist) if one wants to use the previous autotuned result in subsequent runs. This would solve this and this option would anyhow be very useful (prevents normal users from getting bad performance for long runs if auto-tuning did poorly because of some hardware fluctuation - or that multiple independent simulations use different tuned values). I'm of the opinion we shouldn't drop the old option before this option is added. Note that telling people to simply change the tpr is not an option because for benchmarks the inputs to the tpr is often not available. The option to set the tuned spacing/rlist could be either a convert-tpr or mdrun option.

I don't understand the argument to remove nsteps. Why does it add any complexity? It can simply overwrite the value read from the tpr and can be treated exactly the same way (the other options are different in that they trigger different behavior which needs to be implemented separately). I see the options used all over the place both by normal users and in benchmarks. Thus I don't think it should be removed given that it is popular and isn't one of the options which causes the complexity issue.

### #40 - 03/07/2017 06:47 PM - Mark Abraham

Roland Schulz wrote:

Overall this is a great proposal but I disagree on a few points.

I disagree with using environment variables. If the command line argument is called benchmarkdelay/benchmarklength (or benchmark_delay) you achieve the same thing that it is clear that it should be only used for benchmarking. You can even make it a hidden option if you don't want to confuse normal users. Unless you plan to also update the environment variable handling code, we should not use environment variables meant for users. And sysadmins doing benchmarking aren't developers either. The problems with environment variables is that we don't have proper error handling. If you misspell the option or use an invalid argument you don't get a proper error message.

I think we should generally have only one way of specifying things (for general sanity and convenience). Because users are used to using command-line options, they will expect that such options they see in documentation are to be used. I think we should discourage the use of environment variable options by users, because different shells make it awkward for us to document, and it reduces the value of the command lines that we record in user output (.log, .xvg). Hidden options are a reasonable alternative for the benchmarking options, however.

There is one case the new suggested options don't cover. The number of steps of PME tuning aren't reproducible. If you need a reproducible long run for benchmarking (e.g. to measure power consumption), you won't get that with the new options (you could use convert-tpr for length but you would still need resetstep) because of the first part being non-reproducible. It was suggested in some other redmine issue (I can't find it) that we should have an option to set the tuned fourier-spacing (or rlist) if one wants to use the previous autotuned result in subsequent runs. This would solve this and this option would anyhow be very useful (prevents normal users from getting bad performance for long runs if auto-tuning did poorly because of some hardware fluctuation - or that multiple independent simulations use different tuned values). I'm of the opinion we shouldn't drop the old option before this option is added. Note that telling people to simply change the tpr is not an option because for benchmarks the inputs to the tpr is often not available. The option to set the tuned spacing/rlist could be either a convert-tpr or mdrun option.

For measuring power consumption, etc. it is not unreasonable to assume the person has the inputs to build a .tpr (e.g. from some benchmark set we one day create) and can set up the .mdp to reproduce the result of the tuning.

Generally I don't want to drop things without warning. But the implementations of the old functionality has made attempting to clean up the MD loop harder than it needs to be (have a look at multisim nsteps handling in 5.1 some time...), and led to multiple bugs, and the change does not affect lots of users.
I don't understand the argument to remove nsteps. Why does it add any complexity? It can simply overwrite the value read from the tpr and can be treated exactly the same way (the other options are different in that they trigger different behavior which needs to be implemented separately).

Because the interpretation of nsteps is unclear (both in the code, and in user space) in the context of a checkpoint restart. The .tpr nsteps sets the length of the whole simulation, but the mdrun -nsteps sets the length of the simulation part (which is an integer we increase in successive checkpoints). That has made problems with how to implement other things that function with step intervals, because people assume the t_inputrec is constant, and actually it is not. This is also awkward because of how we use step and step_rel (and it is unclear whether those are always correct...)

Further, why should a user expect gmx mdrun -cpi -nsteps 100000 that was interrupted somehow to revert to the .tpr nsteps when they continue with simply gmx mdrun -cpi? What is the correct/expected interpretation of gmx mdrun -cpi -nsteps n when n is greater than the number of steps in the input checkpoint? Or less than the init-step?

It seems to be a good example of something that seems fine to the developer who's only doing limited things (performance testing on the same sets of tpr files), but falls short when faced with user-world complexity. (And none of the behaviour is tested.)

I see the options used all over the place both by normal users and in benchmarks. Thus I don't think it should be removed given that it is popular and isn't one of the options which causes the complexity issue.

Disagree. There's lots of complexity associated with mdrun -nsteps.

#41 - 03/07/2017 07:20 PM - Mark Abraham
- Related to Task #1971: Removing buggy features vs. keeping workflows added

#42 - 03/07/2017 07:31 PM - Mark Abraham
I am fine with extending gmx convert-tpr to cover more niche cases for tpr hacking. Call it gmx hack-tpr to make the usage clear? :-)

Michael might find that useful for some of his simulation correctness checks, too.

#43 - 03/07/2017 07:54 PM - Roland Schulz
If we disallow the combination of nsteps and cpi does that remove all complexity? If so that seems like the better solution then removing it.

Given that we still don't provide a benchmark suite (just simply making (part of ) the WIP one public would be a step) people use benchmarks such as https://www.mpibpc.mpg.de/grubmueller/gromacs and they only have the tpr. Thus you can't assume that people have more than the tpr.

#44 - 03/07/2017 08:03 PM - Szilárd Pall
I welcome the initiative and I'll try to comment constructively on the proposal, but let me emphasize that I want it to be actively brought up for discussion among devs before implementing. It would be nice to avoid situations familiar from the past where some discussion is shut down during review as "it's too late for it" just because somebody may not have followed a redmine or gerrit conversation.

Secondly, somewhat OT, but using the recently introduced bug (#2131) to motivate removing features and claiming the erosion of IMO still valid points seem unreasonable. Blaming the bug on the complexity of run control is on the one had reasonable, but on the other hand, the poor testing and lacking coverage jenkins verification that does not run some of the common non-dev use-cases is at least equally (if not more) at fault. A single MPMD run in jenkins (which AFAIK we used to have before the simplification) would have been enough to keep such a bug out of a release. Let's focus on all lessons that can be learned from an embarrassing bug rather than cherry-picking.

Back to the topic.

For GROMACS 2017:

- I will implement that environment variables GMX_BENCHMARKDELAY=m and GMX_BENCHMARKLENGTH=n will wait for any PME tuning to complete, run m steps, reset counters, run n steps, report timings, and exit. This seems simple to implement, and at least as useful as any of the combinations of existing options, and sometimes more useful in practice. Using environment variables makes clear that we are not intending users to manage their simulations with it. Using it in benchmarking scripts is no harder than using a command-line argument.

Given that you are proposing to turn all mentioned features into explicitly dev-specific functionality, I don't think it's necessary or beneficial to be too restrictive (within reasonable limits). PME tuning is not the only factor that affects measurements. Allowing a single recipe for measurement delay which may work for some, but not for others. For occasional benchmarking the proposed behavior may makes sense, but for day-to-day work, one just wants to run N steps (total or post-reset, whichever definition we prefer) resetting at a certain step M or time T!

The naming and the intention of removing setting the fixed total step-count seems to suggest that easily picking the number of steps is only used for benchmarking. This is not the case, various development-related activities benefit greatly from setting the total step count (with or without resetting timers).

For the above two reasons I prefer keeping the means to trigger reset at an exact step rather than (or additionally to) doing so at some relative offset decided by implementation details. Additionally, GMX_BENCHMARK is not a suitable name-prefix for the broader range of dev activities involved. Hence, I suggest not calling these benchmarking-only features, and prefixing the variables with e.g. "GMX_DEV_".
As we don't report env vars in the log nor command line (nor do we have a consistent way to implement this without the previously discussed option module) having to rely on these makes reproducible work hard. It is important to be able to quickly verify parameters of a performance measurement (without having to e.g. dig through the log manually or grep for the reset message). Hidden options can serve the same purpose as env vars, but they do get printed in the log, so let's keep them as hidden options. We can start flagging dev options with a "dev:" prefix to reflect their intended purpose.

- I will remove mdrun -resetstep and mdrun -resethway, which will be superseded by the above
- mdrun -maxh will be retained (it probably works in all combinations that are sensible, and in most that aren't)

Please don't remove resetting a time-limited run without an alternative. This will break the only reasonable benchmarking methodology for scaling runs: mdrun -maxh H -resethway. Fixed length is not useful here (the same steps that take a minute at peak will take hours on a single node) and resetting at a fixed step offset is not suitable either. One can't know at which step to reset in a parallel run, especially across a set of scaling bench-runs. Testing often requires certain behavior (load balancer, device temperatures, DVFS, etc.) to stabilize and suitable point of resetting can be often estimated by time but not steps.

- I will remove mdrun -nsteps. This was added in the big Verlet-scheme commit (I assume, to suit performance testing), apparently without enough consideration then, or enough re-consideration as we added more functionality like PME auto-tuning. We have gmx convert-tpr -nsteps if developers need to do some hacking while fixing bugs and testing fixes. We only documented it in mdrun -h (GROMACS 5.1 did give it one mention in the user guide), so I'm comfortable with saying that any user who is currently using it needs to change their scripts to use a properly supported path (which we should document in the user guide if we haven't already).

Please don't. I've argued in the past for the general usefulness of the command line option, but that's admittedly debatable. However removing this option and only providing a benchmarking-oriented environment variable is detrimental for development. Command line options are better (for the reasons detailed above), so let's rename it to -dev-nstps, but setting it to whatever I want has been tremendously useful. I have been through the pain of having to generate a new tpr (in the 4.0/4.5 era) just to run during development e.g. -nsteps 0/1 first until I get a kernel ~ correct, then bump it to 40 to check that the second search is still fine.

gmx convert-tpr is just too cumbersome for common developer tasks, e.g. testing single step or a few nstlist cycles, run exactly 1000 iterations in the profiler, debugging/testing on a cluster where you'd normally build mdrun_mpi only (scp-ing tpr files is a pain). Let's not tie dev hands, please, it hampers productivity.

### #45 - 03/07/2017 08:10 PM - Szilárd Páll

Roland Schulz wrote:

> The problems with environment variables is that we don't have proper error handling. If you misspell the option or use an invalid argument you don't get a proper error message.

Good points, that's an even greater issue with env vars than the one I mentioned (not reported in the log)!

- I see the options used all over the place both by normal users and in benchmarks. Thus I don't think it should be removed given that it is popular and isn't one of the options which causes the complexity issue.

I agree, we've advocated for its use for years and I can confirm that it has been adopted by many. I'm not certain how broadly is it used by users doing simulations, but a range of people from admins to vendors use it.

### #46 - 03/07/2017 09:18 PM - Szilárd Páll

Mark Abraham wrote:

> I think we should generally have only one way of specifying things (for general sanity and convenience).

Centralized option-handling that manages options (be it command line, environment variable, or any future UI/API) could give a single place of parsing/checking and make it straightforward to ensure sanity and consistency.

- Because users are used to using command-line options, they will expect that such options they see in documentation are to be used.

Better documentation, better option naming are good ways to address such issues, I think.

- I think we should discourage the use of environment variable options by users, because different shells make it awkward for us to document, and it reduces the value of the command lines that we record in user output (.log, .xvg). Hidden options are a reasonable alternative for the benchmarking options, however.

Documenting how to use an environment variable is not in the scope of the GROMACS docs, so I don't see that as an issue. That's not to say we should encourage their widespread use.

If we end up designating options for benchmarking only, it is indeed reasonable to make them hidden, even separate them in a section for easier discovery by the appropriate group of users.
Generally I don't want to drop things without warning. But the implementations of the old functionality has made attempting to clean up the MD loop harder than it needs to be (have a look at multisim nsteps handling in 5.1 some time...), and led to multiple bugs, and the change does not affect lots of users.

It will affect many vendors and compute-centers. I can't quantify the impact on pre-deployment and procurement benchmarking, but I'm sure it is not negligible. Clear docs of how to update previous workflows should help as long as relevant use-cases can still be implemented.

#47 - 03/07/2017 09:32 PM - Mark Abraham

Bottom line: Features for developer convenience must be well enough designed and tested that they do not introduce bugs in user space. We are paid to produce usable simulation software. If we've made enough past errors that we can't fix a bug in a benchmarking feature (#2041) without breaking everyday user experience (#2131) then we literally have to simplify things. Nobody will ever trust our bug fixes!

Szilárd Páll wrote:

I welcome the initiative and I'll try to comment constructively on the proposal, but let me emphasize that I want it to be actively brought up for discussion among devs before implementing. It would be nice to not have situations familiar from the past where some discussion is shut down during review as "it's too late for it" just because somebody may not have followed a redmine or gerrit conversation.

Yeah but it's a recurring pattern that people say they want something, but don't actually make it happen, then (actively or passively) block others' attempts at fixing problems that actually have happened. Discussion is now happening. Obviously, the few devs that actually do performance testing will want lots of these features and care little about the implementation, and the devs who need to transform the code into what we need in the future, or who focus on things that actually work in the hands of the user, will want as few of these features as possible, and with high quality implementations. We can't accept the kind of culture where we add features with little or no discussion (e.g. mdrun -nsteps), and thereafter can't touch anything without lots of discussion.

Secondly, somewhat OT, but using the recently introduced bug (#2131) to motivate removing features and claiming the erosion of IMO still valid points seem unreasonable. Blaming the bug on the complexity of run control is on the one had reasonable, but on the other hand, the poor testing and lacking coverage jenkin verification that does not run some of the common non-dev use-cases is at least equally (if not more) at fault. A single MPMD run in jenkins (which I think we used to have before the simplification) would have been enough to keep such a bug out of a release. Let's focus on all lessons that can be learned from an embarrassing bug rather than cherry-picking.

Exactly. I already opened #2134 so we have a reminder to reconsider what's going on in Jenkins. But if we're short-handed on implementing our testing machinery... we have to simplify our testing problem, not just tell ourselves that things will be fine in the asymptote. None of the Jenkins testing uses mdrun -tunepme, either, so it is not sufficient to blame problems on an incomplete migration to the new releng functionality.

Back to the topic.

For GROMACS 2017:

- I will implement that environment variables GMX_BENCHMARKDELAY=m and GMX_BENCHMARKLENGTH=n will wait for any PME tuning to complete, run m steps, reset counters, run n steps, report timings, and exit. This seems simple to implement, and at least as useful as any of the combinations of existing options, and sometimes more useful in practice. Using environment variables makes clear that we are not intending users to manage their simulations with it. Using it in benchmarking scripts is no harder than using a command-line argument.

Given that you are proposing to turn all mentioned features into explicitly dev-specific functionality, I don't think it's necessary or beneficial to be too restrictive (within reasonable limits).

Remember that your convenience feature can be my implementation nightmare. :-)

PME tuning is not the only factor that affects measurements. Allowing a single recipe for measurement delay which may work for some, but not for others. For occasional benchmarking the proposed behavior may makes sense, but for day-to-day work, one just wants to run N steps (total or post-reset, whichever definition we prefer) resetting at a certain step M or time T!

One only wants to run exactly N steps in the limited case where you know that tuning is inactive, irrelevant, or reproducible. Setting a benchmark length is equally effective there, as is using a .tpr with a set number of steps.

The naming and the intention of removing setting the fixed total step-count seems to suggest that easily picking the number of steps is only used for benchmarking. This is not the case, various development-related activities benefit greatly from setting the total step count (with or without resetting timers).

I can't think of any of those that can't be handled with actually setting the step count in the tpr. :-) After a short transition period, people who need it will have a library of .tpr files, and users will have more reliable production code. Nice.

For the above two reasons I prefer keeping the means to trigger reset at an exact step rather than (or additionally to) doing so at some relative offset decided by implementation details. Additionally, GMX_BENCHMARK is not a suitable name-prefix for the broader range of dev activities involved. Hence, I suggest not calling these benchmarking-only features, and prefixing the variables with e.g. "GMX_DEV_".

02/23/2020
What's an actual use case, please?

As we don't reports env vars in the log nor command line (nor do we have a consistent way to implement this without the previously discussed option module) having to rely on these makes reproducible work hard. It is important to be able to quickly verify parameters of a performance measurement (without having to e.g. dig through the log manually or grep for the reset message). Hidden options can serve the same purpose as env vars, but they do get printed in the log, so let's keep them as hidden options. We can start flagging dev options with a "dev-" prefix to reflect their intended purpose.

Good point. We can have hidden options rather than environment variables.

- I will remove mdrun -resetstep and mdrun -resethway, which will be superseded by the above
- mdrun -maxh will be retained (it probably works in all combinations that are sensible, and in most that aren't)

Please don't remove resetting a time-limited run without an alternative. This will break the only reasonable benchmarking methodology for scaling runs: mdrun -maxh H -resethway. Fixed length is not useful here (the same steps that take a minute at peak will take hours on a single node) and resetting at a fixed step offset is not suitable either. One can't know at which step to reset in a parallel run, especially across a set of scaling bench-runs. Testing often requires certain behavior (load balancer, device temperatures, DVFS, etc.) to stabilize and suitable point of resetting can be often estimated by time but not steps.

Btw -maxh is not quite reproducible in parallel runs (but probably not significant, given noisy network). I'm open to having specifying benchmark delay/duration also in time (see earlier proposals), but resethway is not a stable way to make such measurements across a range of scaling, precisely because launch transients and tuning don't have clean scaling behaviour.

- I will remove mdrun -nsteps. This was added in the big Verlet-scheme commit (I assume, to suit performance testing), apparently without enough consideration then, or enough re-consideration as we added more functionality like PME auto-tuning. We have gmx convert-tpr -nsteps if developers need to do some hacking while fixing bugs and testing fixes. We only documented it in mdrun -h (GROMACS 5.1 did give it one mention in the user guide), so I'm comfortable with saying that any user who is currently using it needs to change their scripts to use a properly supported path (which we should document in the user guide if we haven't already).

Please don't. I've argued in the past for the general usefulness of the command line option, but that's admittedly debatable. However removing this option and only providing a benchmarking-oriented environment variable is detrimental for development. Command line options are better (for the reasons detailed above), so let's rename it to -dev-nsteps, but setting it to whatever I want has been tremendously useful. I have been through the pain of having to generate a new tpr (in the 4.0/4.5 era) just to run during development e.g. -nsteps 0/1 first until I get a kernel – correct, then bump it to 40 to check that the second search is still fine.

I am sorry you didn't know about it, but tpbconv -nsteps has been around since before 4.0. Other than the cost of changing habits, mdrun -s steps1 and mdrun -s steps40 is just as usable for a developer hacking a short-ranged kernel as mdrun -nsteps 1 and mdrun -nsteps 40.

gmx convert-tpr is just too cumbersome for common developer tasks, e.g. testing single step or a few nstlist cycles, run exactly 1000 iterations in the profiler, debugging/testing on a cluster where you'd normally build mdrun_mpi only (scp-ing tpr files is a pain). Let's not tie dev hands, please, it hampers productivity.

There are many kinds of developer productivity, including being able to understand what do_md does well enough that people can fix bugs and review fixes without making new problems that require further dev time to diagnose, fix and review. Hacking on the step count, and having some things implemented in simulation steps and others in simulation-part steps and some in wall time has made for plenty of user-space problems, and I touched on several of these in my discussion here with Roland. High performance literally does not matter if the code does not work reliably, and users won't upgrade to new and faster versions because they are afraid of our history of breaking things and us not putting enough effort into our testing relative to our number of features. And I think that continuing to be able to pay devs is pretty good for productivity ;-)
It will affect many vendors and compute-centers. I can't quantify the impact on pre-deployment and procurement benchmarking, but I'm sure it is not negligible. Clear docs of how to update previous workflows should help as long as relevant use-cases can still be implemented.

Yeah, but they are not our clients - not that we want to be assholes about it. Our simulation scientists are our clients, and they care about more things than performance.

#49 - 03/08/2017 03:10 AM - Peter Kasson
I'll generally stay out of the...discussion about what should be done in this particular case, but I think that as a general design, creating special-purpose TPRs for testing is a reasonable approach if we put the functionality into convert-tpr. Making people re-generate a TPR from scratch is a PITA, but it seems like a reasonable workflow to say:
gmx convert-tpr -s foo.tpr -o bar.tpr -MOD_ARGS_FOR_TESTING
gmx mdrun -s bar

For cases where we're really modifying input options that live in the TPR.
(Of course I'll also get on my hobby horse and say this is a great use case for API functionality where we have calls to load a tpr, modify some options, and then run ;P. But that will have to wait until our intrepid API engineering team can get code together for such things.)

#50 - 03/20/2017 12:41 PM - Mark Abraham
I'm glad people find some of the mentioned functionality useful. We can consider keeping some of it, but we cannot keep it in its current form. It is not acceptable to edit the inputrec mid-simulation, and we must separate the notions of "steps in the simulation" from "steps in this simulation part" from "steps since reset", and similar notions about time. We can have benchmarking and auto-tuning features, but we cannot have them implemented without thought for the other such that it's easy to make user-space bugs. Note how using these separate notions for steps and time would have made it harder to perpetrate the problems behind #2041 and #2131 (and other similar issues further back in the past).

So, over to the people who think these are critical features - how do you propose to reform the implementations?

#51 - 03/21/2017 09:47 AM - Mark Abraham
Erik made some other suggestions about -nsteps at https://redmine.gromacs.org/issues/1777#note-6

#52 - 03/21/2017 09:48 AM - Mark Abraham
- Related to Bug #1777: Teach mdrun about explicit -append added

#53 - 04/22/2017 12:45 PM - Mark Abraham
- Related to Bug #2136: mdrun -noconfout checkpointing issue added

#54 - 08/02/2017 09:59 PM - Roland Schulz
- Related to Feature #2224: Proposed feature: conditional stop added

#55 - 10/30/2017 05:13 PM - Mark Abraham
- Target version changed from 2018 to 2019

#56 - 12/26/2017 01:17 AM - Mark Abraham
Per discussion at #2136, we will remove -confout because proper design of benchmarking features will remove the need for it. A normal user can script a call to rm just as readily as use -noconfout.

#57 - 01/03/2018 06:48 PM - Mark Abraham
- Related to Bug #2360: error at counter reset with PME-only rank added

#58 - 01/10/2018 10:32 PM - Gerrit Code Review Bot
Gerrit received a related patchset '2' for Issue #1781.
Uploader: Mark Abraham (mark.j.abraham@gmail.com)
Change-Id: gromacs~release-2018~I96e19acb0e15d3f42b0929f555b451299a2882e4
Gerrit URL: https://gerrit.gromacs.org/7451

#59 - 03/02/2018 06:29 PM - Szilárd Páll
Ref https://github.com/bio-phys/MDBenchmark

#60 - 04/27/2018 02:09 PM - Mark Abraham
Mark Abraham wrote:

So, over to the people who think these are critical features - how do you propose to reform the implementations?
So a year has passed, and nothing has happened on this while lots of other work has happened. If you leave it up to me then these features will be removed because as described above they're not fit for their actual purpose, are implemented poorly, have caused user space bugs, and are in the way for implementing APIs and generalizing the MD loop.

#61 - 04/27/2018 06:39 PM - Gerrit Code Review Bot
Gerrit received a related patchset '1' for Issue #1781.
Uploader: Mark Abraham (mark.i.abraham@gmail.com)
Change-ID: gromacs~master~I16c9685643a7b4ac76b94f378da3d05914d6986
Gerrit URL: https://gerrit.gromacs.org/7806

#62 - 09/19/2018 03:07 PM - Mark Abraham
- Related to Task #2569: announce deprecations in GROMACS 2019 added

#63 - 10/05/2018 11:49 AM - Gerrit Code Review Bot
Gerrit received a related patchset '1' for Issue #1781.
Uploader: Mark Abraham (mark.i.abraham@gmail.com)
Change-ID: gromacs~master~I1d00859d0f15409a472984f5a65347a50c71ad17
Gerrit URL: https://gerrit.gromacs.org/8488

#64 - 10/08/2018 07:04 PM - Mark Abraham
- Target version changed from 2019 to 2020

Pascal has improved aspects of triggering signals, and the move to gmx benchmark in future will also help

#65 - 11/08/2018 12:24 PM - Mark Abraham
- Related to Bug #2717: mdrun runs infinitely when checkpoint file is beyond the designated end point added

#66 - 11/15/2018 12:27 AM - Mark Abraham
- Related to Bug #2757: mdrun refuses to start with .cpt if nsteps is -1 in .tpr added

#67 - 12/20/2019 12:03 PM - Paul Bauer
- Target version changed from 2020 to future

Files

<table>
<thead>
<tr>
<th>Filename</th>
<th>Size</th>
<th>Date</th>
<th>Author</th>
</tr>
</thead>
<tbody>
<tr>
<td>automation_with_extra_info.log</td>
<td>16.6 KB</td>
<td>07/15/2015</td>
<td>Szilárd Páll</td>
</tr>
<tr>
<td>bad_run.txt</td>
<td>4.73 KB</td>
<td>07/15/2015</td>
<td>Szilárd Páll</td>
</tr>
</tbody>
</table>