GROMACS - Bug #1880

PP-PME load balancing issue

12/16/2015 03:24 PM - Szilárd Páll

Status:	Feedback wanted		
Priority:	Normal		
Assignee:	Berk Hess		
Category:	mdrun		
Target version:	future		
Affected version - extra info:	5.1	**Difficulty:**	hard
Affected version:	git master		

Description

In some cases the first measurement proves to be very slow and the consecutive setting much faster than the it can happen that the first setting is pruned and never tried again which will lead to suboptimal performance due to the CPU waiting for the GPU. Example below, repro case (rnase cubic) + log attached.

Perhaps we should (always?) repeat measurements with the first setting?

```
step   80: timed with pme grid 56 56 56, coulomb cutoff 0.900: 167.6 M-cycles
step  160: timed with pme grid 48 48 48, coulomb cutoff 1.046: 149.1 M-cycles
step  240: timed with pme grid 44 44 44, coulomb cutoff 1.141: 169.0 M-cycles
step  320: timed with pme grid 48 48 48, coulomb cutoff 1.046: 145.6 M-cycles
step  400: timed with pme grid 48 48 48, coulomb cutoff 1.046: 143.2 M-cycles
              optimal pme grid 48 48 48, coulomb cutoff 1.046
```

Related issues:

Related to GROMACS - Feature #1715: improve cycle counting GPU sharing and mu...	**New**

Associated revisions

Revision b33a991d - 01/08/2016 05:55 PM - Berk Hess

Fix PME load balancing skipping a setup

During stage 0 of the PME tuning we only time every second setup.
In stage 1 only a reduced range of setups is considered. If the tpr
setup (setup index 0) was not part of this reduced range, the first
setup in the range would never be timed. This could lead to choosing
a non-bonded cut-off that was longer than optimal.

Fixes #1880

Also fixed two conditionals (but this had no effect on the settings).

Change-Id: Id9f361d23e3294e41553b39ae8e8f37ceb80bbe5

History

#1 - 12/16/2015 03:29 PM - Szilárd Páll

- File _test3.log added

After a few tries I got the idea that this may be caused by the sluggish frequency scaling of the GPUs. I ran multiple mdruns back-to-back to test. The results are mixed and, while the number of cases where the tuning did find the correct cut-off (see attached) has increased, this is not always the case.

Any ideas how to eliminate this first, very much off measurement?

#2 - 01/04/2016 03:10 PM - Gerrit Code Review Bot

Gerrit received a related patchset '1' for Issue #1880.
Uploader: Berk Hess (hess@kth.se)
Change-Id: Id9f361d23e3294e41553b39ae8e8f37ceb80bbe5
Gerrit URL: https://gerrit.gromacs.org/5523

#3 - 01/04/2016 03:13 PM - Berk Hess

- Status changed from New to Fix uploaded

- Assignee set to Berk Hess

The main issue here seems to be that the first potentially useful setup is skipped in the second stage (grid 52 in this case), when the tpr setup is not included in the final range. I uploaded a fix for this.
I see some variation in the timing of the first (tpr) setup, but not so much that it seems to be problematic.

#4 - 01/08/2016 06:00 PM - Berk Hess

- Status changed from Fix uploaded to Resolved

Applied in changeset b33a991d7e95ced3e106597129e51a8229dc252e.

#5 - 01/12/2016 04:22 PM - Mark Abraham

- Status changed from Resolved to Closed

#6 - 01/13/2016 06:13 PM - Szilárd Páll

- File test_1x16_notune_5.log added

- File test_1x16_tune_5.log added

- File test_1x16_tune_1.log added

I have more data that still shows more or less the same as the initial report - even with the fix included. This seems to be caused by timing issues. On the one hand, as I guessed initially, the load balancing can erroneously discard the base cut-off due to the first measurement being consistently very much off (see test_1x16_tune_1.log), but even when it re-tries the shorter cut-offs (including the base) it will very often measures higher timings with shorter cut-offs that otherwise obviously give better overall performance.

The attached files clearly show that the picked 1.046 nm cut-off leads to worse performance (and large GPU wait time).

I'm considering reopening this but I can also file a new report if that's preferred.

#7 - 01/13/2016 06:13 PM - Szilárd Páll

- Status changed from Closed to Feedback wanted

#8 - 01/13/2016 11:46 PM - Mark Abraham

Szilárd Páll wrote:

> I have more data that still shows more or less the same as the initial report - even with the fix included. This seems to be caused by timing issues. On the one hand, as I guessed initially, the load balancing can erroneously discard the base cut-off due to the first measurement being consistently very much off (see test_1x16_tune_1.log)

It looks like only the first 56x56x56 of test_1x16_tune_1.log has a higher time. I assume you did several replicates of this run - what was the baseline variability of this first segment, vs later segments. Is this an artefact of a cold GPU?

> but even when it re-tries the shorter cut-offs (including the base) it will very often measures higher timings with shorter cut-offs that otherwise obviously give better overall performance.

Is there data here to show that? The grid repeats in both test_1x16_tune_[15].log look OK to me.

> The attached files clearly show that the picked 1.046 nm cut-off leads to worse performance (and large GPU wait time).
>
> I'm considering reopening this but I can also file a new report if that's preferred.

There's a case for this kind of tuning moving to the next grid setup only when performance has stabilized, which might mitigate the effect of init-time transients. Thus, rather than (as here) 2*nstlist segments, run multiple nstlist segments and accept a performance number only when there's been 2+ segments with less than a couple of percent variation. Here, 56x56x56 ran at 132.4 and 123.8 which is 7% variation.

Is that also workable in the presence of DLB?

#9 - 01/14/2016 10:00 PM - Szilárd Páll

- File rnase-GPU-timing.pdf added

Mark Abraham wrote:

> It looks like only the first 56x56x56 of test_1x16_tune_1.log has a higher time. I assume you did several replicates of this run - what was the baseline variability of this first segment, vs later segments. Is this an artefact of a cold GPU?

See the data attached. It shows load-balancer reported timings from runs with tweaked limits to trigger more passes. This illustrates quite well that the first two cut-off settings (0.9 nm and 0.966 nm) both show 10-20% lower timings after the 2nd/3rd measurements while the third cut-off, 1.046 nm, is affected much less (even though it is the second setting during the 1st pass). The fourth cut-off shows similarly small 2-4% deviation.

> but even when it re-tries the shorter cut-offs (including the base) it will very often measures higher timings with shorter cut-offs that otherwise obviously give better overall performance.

> Is there data here to show that? The grid repeats in both test_1x16_tune_[15].log look OK to me.

The attached plots show how can this happen, essentially it seems that performance shorter cut-offs/finer grids take longer to stabilize. During the first pass 0.966 nm is not tested, hence the first data point on the curve here belongs to the second pass (I realize now that a different representation or lat least shifting 0.966 could've been better). At the second pass 0.900 is still slower than 1.046 and that's when 0.966 nm is first timed.

Based on what I've seen so far this does not *seem* to be caused by a "cold" GPU. Back-to-back runs without pause between them show the same pattern and the fact that the 1.046 nm run is less affected even though it's the 2nd setup the load balancer tried, seems to indicate the same.

it would be good to know what's the root cause of this behavior, so unless we have another idea, I should get a trace an see if it reveals where the performance change originates from.

> There's a case for this kind of tuning moving to the next grid setup only when performance has stabilized, which might mitigate the effect of init-time transients. Thus, rather than (as here) 2*nstlist segments, run multiple nstlist segments and accept a performance number only when there's been 2+ segments with less than a couple of percent variation. Here, 56x56x56 ran at 132.4 and 123.8 which is 7% variation.

Yes, that would make sense. It would also make sense to allow re-balancing later (or periodically).

> Is that also workable in the presence of DLB?

We delay turning on DLB until the balancer picks grit+cutoff and turn it on again only after that, so DLB it can not interfere.

#10 - 01/14/2016 11:36 PM - Mark Abraham

What do the colours show, please?

#11 - 01/14/2016 11:50 PM - Szilárd Páll

The different colored curves represent 10 repeats with 10s pause in between.

#12 - 01/15/2016 01:05 AM - Szilárd Páll

Skipping 15-20 nstlist cycles (that is 600-800 steps here) in the beginning *seems* to eliminate the effect. I have so far only done a quick test that increases the number of cycles for each configuration (i.e. set->count), but it looks like it may be enough to simply start later. Your suggestion of monitoring variation between consecutive segments still seems like a good idea.

#13 - 01/27/2016 01:14 PM - Mark Abraham

- Target version changed from 5.1.2 to 5.1.3

#14 - 06/01/2016 11:29 AM - Mark Abraham

- Target version changed from 5.1.3 to 2016

Release-5-1 branch is shortly moving into correctness-only fixing, so if we can/do fix anything about this in a release branch, it will only be in release-2016

#15 - 07/09/2016 03:30 AM - Szilárd Páll

I had a core spin-up solution in testing but as it was thought to be an undesirable hack (though I still thought it was better than nothing), I postponed it. Funnily enough there is now code to spin up cores that will likely be merged into 2016 which could solve this issue too (1-2 second of spinning will bring x86 cores clock up to allow reliable measurements). So if that spin-up code goes in, we could consider enabling it when -tunepme later on.

Not a blocker for 2016 RC anyway.

#16 - 07/11/2016 05:08 PM - Mark Abraham

Isn't it better all round to spin up by doing some MD steps before launching this tuning?

#17 - 07/12/2016 01:23 AM - Szilárd Páll

It can be reasonable, but we don't know how long is and MD step and therefore that's not a very reliable way to ensure that independent of the time/step, we always spin up cores to the same extent. This is not an issue as long as steps are much shorted than the required spin-up time, but that may not always be the case, right?

#18 - 07/12/2016 06:23 PM - Mark Abraham

Szilárd Páll wrote:

> It can be reasonable, but we don't know how long is and MD step and therefore that's not a very reliable way to ensure that independent of the time/step, we always spin up cores to the same extent. This is not an issue as long as steps are much shorted than the required spin-up time, but that may not always be the case, right?

We're never going to know how long spin-up time is, so that problem is common to the idea of writing/using some spin-up code, and doing MD steps has the benefit of being useful to the user (assuming we're not shooting ourselves in the foot with different tuning interacting suboptimally).

We do know that the assumption of constant performance characteristics per step is wrong unless both the simulation contents and the hardware conform to idealized behaviour, so as above I think we need to move in the direction of being more sophisticated about when we accept a performance estimate. If there might be a starting transient that takes hundreds of steps to die away, and mdrun will never know in advance when that might be true, and also mdrun can never know when the network performance, or node sharing, or CPU clocks, or GPU clocks might change (where applicable, and we can probably hard-wire a few things), we have to address this with better high-level management code. It might be quite useful to try plausible alternatives every few hundred or so nstlist steps, where "plausible" depends on how confident we were that the current choice was better than the alternatives, and whether the current performance characteristics are still consistent with the data that led to that choice.

#19 - 07/28/2016 05:34 PM - Mark Abraham

I assume nobody has plans to try to fix anything here for 2016

#20 - 07/28/2016 05:40 PM - Mark Abraham

#2007 is possibly related

#21 - 08/03/2016 03:29 PM - Erik Lindahl

Given that the load balancing iterations take at least several hundred steps in itself, I think it makes a lot of sense to wait e.g. 500 steps before we start doing it.

#22 - 08/03/2016 10:36 PM - Szilárd Páll

- Difficulty hard added

Mark Abraham wrote:

> #2007 is possibly related

I don't think it is as the present issue is only related to measurements on a possibly "cold" CPU which can only result in suboptimal cut-off picked, but not in corruption.

Erik Lindahl wrote:

> Given that the load balancing iterations take at least several hundred steps in itself, I think it makes a lot of sense to wait e.g. 500 steps before we start doing it.

That's what I/we considered initially too and it would be an acceptable workaround for in the majority of the cases. However, if the time/step is in the ~1 ms or lower regime, that will give <1s delay which, at least on the i7 6960X, is a bit too short.

#23 - 08/04/2016 12:58 AM - Mark Abraham

- Related to Feature #1715: improve cycle counting GPU sharing and multi-sim added

#24 - 08/04/2016 01:00 AM - Mark Abraham

Indeed, which is why we have to plan to move to a style where we observe actual performance rather than hard-code assumptions about whether there is a steady state. Doing it in a more responsive way might also help resolve #1715 "naturally," where GPU sharing between simulations can increase jitter in the per-simulation performance measurements.

#25 - 09/07/2016 03:23 PM - Mark Abraham

- Target version changed from 2016 to 2018

#26 - 12/21/2017 03:20 PM - Mark Abraham

- Target version changed from 2018 to 2019

We need to rework all of this code at some point

#27 - 10/03/2018 09:45 PM - Mark Abraham

- Target version changed from 2019 to future

Files

topol.tpr	1.02 MB	12/16/2015	Szilárd Páll
test_1x12_1task_notune_noT.log	22.6 KB	12/16/2015	Szilárd Páll
test_1x12_1task_noT.log	23.5 KB	12/16/2015	Szilárd Páll
_test3.log	25.4 KB	12/16/2015	Szilárd Páll
test_1x16_notune_5.log	23.8 KB	01/13/2016	Szilárd Páll
test_1x16_tune_5.log	25 KB	01/13/2016	Szilárd Páll
test_1x16_tune_1.log	24.9 KB	01/13/2016	Szilárd Páll
rnase-GPU-timing.pdf	48.9 KB	01/14/2016	Szilárd Páll