GROMACS - Feature #1715

improve cycle counting GPU sharing and multi-sim

04/10/2015 05:37 PM - Szilárd Páll

Status:	New	
Priority:	Low	
Assignee:		
Category:	core library	
Target version:	future	
Difficulty:	hard	

Description

While DLB does take into account GPU sharing within a simulation by redistributing the GPU wait times among ranks sharing a GPU, this does not happen with multi-sim. This can throw off the PP-PME load balancing, so it would be useful to account for GPU sharing among individual simulation in a multi run.

Related issues:

Related to GROMACS - Bug #1880: PP-PME load balancing issue	**Feedback wanted**

History

#1 - 04/14/2015 02:21 PM - Mark Abraham

Indeed, thanks. Some more detailed questions have occurred to me since our discussion, which we should probably think about and then discuss.

Assume R is the number of simulations, N is the number of nodes, S is the number of sockets per node, C the number of CPU cores per socket, and G the number of GPUs per node.

- Which counters are important to share? Does that change if there is more than one domain per sim?
- Is the current code OK for R < N? Even when G > 1? (I think yes, because that's the target use case for non-multi simulations)
- Is the current code OK in the degenerate case of R = N*G? Even if each sim has more than one domain (for some weird reason)?
- R * k = N * S * C for positive integers k is the natural way to use multi-sim, and (e.g.) we can currently run multi-sim on nodes with 2 6-core CPUs and 3 sims per node. (Probably not optimal throughput, but maybe you might do that to improve time-to-solution.) With GPUs, should we constrain that further so that k must be 1 or a multiple of G? For example, on those same nodes with 2 6-core CPUs and 2 GPUs, prohibit running 3 sims per node, where we would now need two ranks per sim, with one sim split over two GPUs, and thus complicated GMX_GPU_ID setup? If so, then on such a node, k in {1,2,4,6,12} would be permitted, but not k=3. That seems to me likely to simplify the proposed load-balancing code, because you can now assume that each simulation has all its domains on the same GPU, or is the only simulation on its GPU (of which there might be more than one).

Any other issues I haven't thought of?

#2 - 06/18/2015 02:57 PM - Erik Lindahl

- Target version changed from 5.x to future

#3 - 04/03/2016 10:20 PM - Erik Lindahl

- Tracker changed from Bug to Feature

Since this does not produce any incorrect results, and it's not important enough for anybody to actually look into during the last year, I'm changing this to a future feature.

#4 - 04/03/2016 10:20 PM - Erik Lindahl

- Priority changed from Normal to Low

#5 - 04/03/2016 11:29 PM - Szilárd Páll

Sorry for the long silence, this slipped off of my radar.

Mark Abraham wrote:

> Assume R is the number of simulations, N is the number of nodes, S is the number of sockets per node, C the number of CPU cores per socket, and G the number of GPUs per node.
>
> - Which counters are important to share? Does that change if there is more than one domain per sim?

The force-counter, input of the DD load balancer. What changes with more than one domain per sim is that GPU wait times may need to be averaged both across ranks within a sim and across sims.

That does not seem to be a problem if had a communicator per physical node (or even per ranks sharing an accelerator).

- Is the current code OK for R < N? Even when G > 1? (I think yes, because that's the target use case for non-multi simulations)

Not necessarily. One feature that I know is useful (and I know somebody whom I advised did use it) is to interleave ranks of a smulti-sim to achieve better GPU utilization.

So in general, just because R < N, it does not mean that simulations won't share nodes.

- Is the current code OK in the degenerate case of R = N*G? Even if each sim has more than one domain (for some weird reason)?

R = N*G <==> there is one simulation per GPU, so yes, that's correct now because there is no GPU sharing.

- R * k = N * S * C for positive integers k is the natural way to use multi-sim, and (e.g.) we can currently run multi-sim on nodes with 2 6-core CPUs and 3 sims per node. (Probably not optimal throughput, but maybe you might do that to improve time-to-solution.) With GPUs, should we constrain that further so that k must be 1 or a multiple of G? For example, on those same nodes with 2 6-core CPUs and 2 GPUs, prohibit running 3 sims per node, where we would now need two ranks per sim, with one sim split over two GPUs, and thus complicated GMX_GPU_ID setup? If so, then on such a node, k in {1,2,4,6,12} would be permitted, but not k=3. That seems to me likely to simplify the proposed load-balancing code, because you can now assume that each simulation has all its domains on the same GPU, or is the only simulation on its GPU (of which there might be more than one).

I don't think such simplifications are necessary or that they would allow more simple implementation.

As we hope to integrate some/more ensemble features into mdrun, I think it is best to keep it reasonably flexible. It is not unreasonable to assume slight inhomogeneity in the hardware used for ensemble runs. Also, sometimes you may end up with R * k != N * S * C if e.g. you need 10 replicas, but you have 12-core nodes. Another quite resonable case is that one wants to combine slightly inhomogenous hardware into a multi-run; e.g some nodes with G=2 while other with G=1, but faster GPUs (e.g. like tcbs20/tcbs21.theophys).

Generally, as long as the GPU is not on the critical path (which can easily be the case e.g. if k is small, or free energy CPU kernels are on the critical path), having an uneven split is not an issue.

However, such generalization does not seems to require complicating load balancing code. To me it seems that the following steps would solve the issue: use a non simulation-structure related communicator setup, first split with physical host as key (we already have that AFAIK), than split these further with GPU ID as key to get groups that should collaborate in correcting their force timings.

#6 - 04/03/2016 11:30 PM - Szilárd Páll

Erik Lindahl wrote:

> Since this does not produce any incorrect results, and it's not important enough for anybody to actually look into during the last year, I'm changing this to a future feature.

Buggy load balancing is not a feature. Incorrect software behavior does not turn into a feature just because we have not worked on the issue. So, IMHO this is still a **bug** no matter how we spin it. There would certainly be room for re-considering the issue, agreeing on relevance, priority, target release, whether/when anyone intends to work on it (if any), but the recent posts don't give the impression of leaving room for anything like that. I would prefer to discuss it first before essentially getting a seemingly a useful report downgraded to junk.

#7 - 08/03/2016 12:32 PM - Mark Abraham

- Difficulty hard added

Szilárd Páll wrote:

> Erik Lindahl wrote:
>
> > Since this does not produce any incorrect results, and it's not important enough for anybody to actually look into during the last year, I'm changing this to a future feature.
>
> Buggy load balancing is not a feature. Incorrect software behavior does not turn into a feature just because we have not worked on the issue. So, IMHO this is still a **bug** no matter how we spin it. There would certainly be room for re-considering the issue, agreeing on relevance, priority, target release, whether/when anyone intends to work on it (if any), but the recent posts don't give the impression of leaving room for anything like that. I would prefer to discuss it first before essentially getting a seemingly a useful report downgraded to junk.

Nobody called the wrong behaviour a feature. Nobody downgraded anything here to junk. Please keep discussions technical and focussed.

It would be useful to use a Redmine category like "bug" as a proxy for "we should consider fixing it in a release branch", and if so, then it isn't useful to

regard "bug" as synonymous with "existing feature isn't perfectly designed/implemented." By extension, that approach also makes every feature request also a very high-level bug...

#8 - 08/04/2016 12:58 AM - Mark Abraham

- Related to Bug #1880: PP-PME load balancing issue added