GROMACS - Bug #2578

with PME on GPU, EM setup does not propagate state->x pinning settings

07/18/2018 04:49 PM - Mark Abraham

Status: Closed
Priority: Normal
Assignee: Mark Abraham
Category: mdrun
Target version: 2018.3
Affected version - extra info:
Affected version: 2018.2

Difficulty: uncategorized

Description

`init_em()` does a dumb deep copy of `state_global` into `ems->s` which is probably where we are losing pinning settings.

Related issues:
Related to GROMACS - Bug #2554: write lowest energy coordinates fail with CG Closed
Related to GROMACS - Bug #2612: HostAllocationPolicy shares pointers and pinn... Closed

Associated revisions

Revision e69cbf27 - 08/22/2018 11:07 AM - Mark Abraham
Permit PME on GPUs only for dynamical integrators

We did not intend to support anything else (and at least energy minimization does not work, because copying of HostVector does not work properly).

Fixes #2578

Change-id: l260e578b718614e868ea264a1f487b903704f94f

History

#1 - 07/18/2018 04:56 PM - Mark Abraham
- Related to Bug #2554: write lowest energy coordinates fail with CG added

#2 - 08/14/2018 03:13 PM - Berk Hess
- Status changed from New to Feedback wanted

Actually EM creating new `t_state`s on the stack, which are then By default unpinned. But I tried using assignment instead.

The issue is the other way around. Rather than the deep copy being "dumb", it is "smart" an unpins the source of the assignment. I understand that this is to minimize use of limited resources, but it is extremely counterintuitive that an assignment modifies the source. I think the issue of source modification is much worse than the risk of creating many pinned buffer copies where the original is no longer used for pinning. Furthermore, even if resource limitation is extremely important, when you do a copy assignment, why should the source be unpinned? You could equally well argue that the target should not be pinned.

Also, the unit test does not checking for the unpinning of the source.

#3 - 08/14/2018 03:18 PM - Gerrit Code Review Bot
Gerrit received a related patchset `1` for Issue #2578.
Uploader: Szilárd Páll (pall.szilard@gmail.com)
Change-id: gromacs~master~Idf2f12c1fcd39a918b7b287630f309b65e5dea25
Gerrit URL: https://gerrit.gromacs.org/8186

#4 - 08/14/2018 03:38 PM - Szilárd Páll
Berk Hess wrote:

Also, the unit test does not checking for the unpinning of the source.

Added a change to cover that.
To the issue: IMO, with a simple assignment we should not transparently double the amount of pinned pages we consume. It is a known risk that this resource easily gets exhausted. Just as before, I still think the memory allocation API should avoid making it easy and transparent for this to be triggered -- not based on the current limited use-cases we have. I agree with Berk that the current unpin the source is somewhat counter-intuitive, and perhaps not clear enough in the API docs -- at least I could not find it (nor explicitly expressed in the unit tests until now).

To keep things safe I see two options:
- leaving either source or target unpinned -- the latter is perhaps more intuitive?
- disable assignment and force a more verbose / obvious way of constructing/initialization that requires explicitly requesting that pinning is preserved.

#5 - 08/14/2018 03:40 PM - Gerrit Code Review Bot
Gerrit received a related patchset '1' for Issue #2578.
Uploader: Berk Hess (hess@kth.se)
Change-Id: gromacs~release-2018~I97a1e82f328a379fe57773d2e22b3d48cf2d2d7b
Gerrit URL: https://gerrit.gromacs.org/8187

#6 - 08/14/2018 03:40 PM - Gerrit Code Review Bot
Gerrit received a related patchset '1' for Issue #2578.
Uploader: Berk Hess (hess@kth.se)
Change-Id: gromacs~release-2018~I96096fc98b06719248c5528e983bc6673af78215
Gerrit URL: https://gerrit.gromacs.org/8188

#7 - 08/14/2018 05:50 PM - Mark Abraham
If the copy assignment is indeed changing the source, then we should have a new unit test to show that this used to be the behavior and now is not. And test both copy initialization that looks like copy assignment, and true copy assignment.

I do not yet see why changing the implementation of the malloc function is the correct point for a fix.

#8 - 08/14/2018 06:24 PM - Mark Abraham
Specifically, malloc should not be called on the allocator of the vector on the rhs of either copy initialization or copy assignment. So I don't yet see why Berk's fix has the intended effect of avoiding unpinning the rhs HostVector.

I can't see the code in EM right now, but reasonable options could include moving the contents from one HostVector to another. If deep copy is the right semantics for the operation, then the rhs should be unchanged, and IMO the LHS should have the pinning state of the RHS - the coder asked for a deep copy and copying all the state is the least surprising result.

Just as with a normal vector, if the coder wanted to minimize resource usage then they need to not ask for a deep copy, ie use a move, or a manual unpinning, or a std::copy depending on the required semantics.

#9 - 08/14/2018 07:48 PM - Gerrit Code Review Bot
Gerrit received a related patchset '1' for Issue #2578.
Uploader: Szilárd Páll (pall.szilard@gmail.com)
Change-Id: gromacs~release-2018~Idf2f12c1fcd39a918b7b287630f309b65e5dea25
Gerrit URL: https://gerrit.gromacs.org/8189

#10 - 08/14/2018 08:59 PM - Berk Hess
To answer Mark's first question:
In the current implementation, the LHS get a copy of RHS pointer. It then unpins that pointer before overwriting it with the new one obtained from malloc.

I think we all agree that we should not modify the RHS in any way.
It seems we also agree that not propagating the pinning settings to the LHS is not ideal.
So then there are two solutions left:
1) Propagate the pinning settings
2) Disallow copy completely

Option 2) is only workable when we have different objects for the global and local state, since we often need to copy global state objects. Sebastian already has such a separation. That would still require a solution for generating extra local state objects in the energy minimization.

Is solution 1) acceptable for the moment?

#11 - 08/14/2018 10:27 PM - Gerrit Code Review Bot
Gerrit received a related patchset '1' for Issue #2578.
Uploader: Berk Hess (hess@kth.se)
Change-Id: gromacs~release-2018~Ic6d87045c97e96cb0cbad81032e2bc0a428f5c84
Gerrit URL: https://gerrit.gromacs.org/8191

#12 - 08/15/2018 12:29 AM - Gerrit Code Review Bot
Gerrit received a related patchset '2' for Issue #2576.
Uploader: Roland Schulz (roland.schulz@intel.com)
Change-id: gromacs~master~I15034e4db2434ea46eda84e70e3f8e6072660d20
Gerrit URL: https://gerrit.gromacs.org/8192

#13 - 08/15/2018 09:13 AM - Berk Hess
- Related to Bug #2612: HostAllocationPolicy shares pointers and pinning setting through shared_ptr added

#14 - 08/15/2018 10:46 AM - Roland Schulz
Copy assignment and copy construction are different for allocators. Their behavior is selected with propagate_on_container_copy_assignment and select_on_container_copy_construction in very different ways for a good reason. For copy assignment there are two choices where the allocator comes from: the vector copied into or copied from. I believe we should choose the first. This prevents accidentally pinning. This mean that if someone does:
HostVector<T> v;
HostVector<T> o;
v = o;
Whether v is pinned depends only on v and can be explicitly chosen by doing:
HostVector<T> v(policy);
v = o;
This mean that code such as in minimize can create a copy and can choose how it wants to pin independent of whether the source is pinned. And it prevents accidentally pinning. Because if you create a new empty HostVector by default it isn't pinned. But if you choose to have it pinned than it is efficient (doesn't require reallocation in changePinningPolicy) because you set pinning prior to copy.
This option doesn't exist for copy construction. For copy construction no previous container exists and no way to choose the policy exists. Thus I believe that copy construction (but not copy assignment) should be disallowed.
My patch does that and also gets git of the Impl class to prevent the other problems.

#15 - 08/15/2018 10:58 AM - Berk Hess
Roland's solution sounds reasonable to me. The question is what we want to fix and how in release-2018. Note that I added https://redmine.gromacs.org/issues/2612 on this. HostAllocationPolicy is seriously broken, but this currently doesn't affect functionality in release-2018. What should we do? options:
1) Leave it broken as is in release-2018
2) Apply Roland's change (+ Szilard's test extension) in release-2018
3) Disallow all copy and move in release-2018

And we still need to fix the actual EM bug. My hacky fix https://gerrit.gromacs.org/#/c/8191/ works, while avoiding any of the allocator issues. But this is not a sustainable solution, since if we if we change more contents of t_state to use HostAllocationPolicy, we have to manually update the pinning settings copy function. Should we use my hacky solution for the moment, or does anyone know a better solution?

#16 - 08/15/2018 06:18 PM - Szilárd Páll
Berk Hess wrote:
Roland's solution sounds reasonable to me.

The question is what we want to fix and how in release-2018. Note that I added https://redmine.gromacs.org/issues/2612 on this. HostAllocationPolicy is seriously broken, but this currently doesn't affect functionality in release-2018. What should we do? options:
1) Leave it broken as is in release-2018
2) Apply Roland's change (+ Szilard's test extension) in release-2018
3) Disallow all copy and move in release-2018

And we still need to fix the actual EM bug. My hacky fix https://gerrit.gromacs.org/#/c/8191/ works, while avoiding any of the allocator issues. But this is not a sustainable solution, since if we if we change more contents of t_state to use HostAllocationPolicy, we have to manually update the pinning settings copy function. Should we use my hacky solution for the moment, or does anyone know a better solution?

#17 - 08/16/2018 05:36 PM - Mark Abraham
We didn't target supporting EM with PME on GPUs, and I don't think the code for EM, t_state, nor HostAllocationPolicy is in good enough shape to claim that we should do anything risky now in order to support it. In particular, the copying of a HostVector was never really part of my intended target functionality (though it was an error that I neither tested it nor prohibited it). So I think the simplest thing to do in release-2018 is to modify pme_gpu_supports_input so that only dynamical integrators are supported (ie no NM, no EM, no TPI).

There's a fixme in runner.cpp that already notes that the pinning of state_global->xx is a hack that happens to work because only the two cases (single-domain and single-rank-for-PME-only) are targeted for supporting PME on GPUs. Managing the different kinds of state and their requirements is not at all simple design task, and worse if you want to minimize resource usage and copying. A better design would probably have

- different types for global and local state,
- local state has a few containers of suitable type(s) that model vector
- modules register their interest with the local state (or states, for EM)
local state queries all the modules that registered interest and handles the requirements they then describe
local state does not support deep or shallow copy

We can't keep adding features and widening support of the current mess. Nor can we keep on not having proper test coverage of EM - the reason pme_gpu_supports_input() does not support TPI is probably related to the fact we have a test of TPI.

I am open to minor changes in release-2018 that prevent the misuses that EM was using, in case others might do the same.

#18 - 08/21/2018 07:05 PM - Gerrit Code Review Bot
Gerrit received a related patchset '1' for Issue #2578.
Uploader: Mark Abraham (mark.j.abraham@gmail.com)
Change-id: gromacs~release-2018~I260e578b718614e868ea264a1f487b903704f94f
Gerrit URL: https://gerrit.gromacs.org/8212

#19 - 08/22/2018 11:15 AM - Mark Abraham
- Status changed from Feedback wanted to Resolved

Applied in changeset e69cbf27b359504859fd961ea6852d2a9e5787596.

#20 - 08/23/2018 10:47 AM - Paul Bauer
- Status changed from Resolved to Closed