GROMACS - Bug #1846

NVML management resets GPU application clocks rather than restoring

10/23/2015 01:55 AM - Szilárd Páll

<table>
<thead>
<tr>
<th>Status:</th>
<th>Closed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Priority:</td>
<td>Low</td>
</tr>
<tr>
<td>Assignee:</td>
<td>Szilárd Páll</td>
</tr>
<tr>
<td>Category:</td>
<td>mdrun</td>
</tr>
<tr>
<td>Target version:</td>
<td>5.1.3</td>
</tr>
<tr>
<td>Affected version - extra info:</td>
<td>5.1</td>
</tr>
</tbody>
</table>

Description
If mdrun is able to change the GPU application clocks, at cleanup it calls nvmlDeviceResetApplicationsClocks() which will reset the clocks possibly leaving the card with different clocks than the ones set at mdrun startup.

Related issues:
Related to GROMACS - Bug #1970: GPU clock resetting at cleanup can interfere ...

Closed

Associated revisions
Revision ddc42de5 - 06/28/2016 01:56 PM - Berk Hess
Properly reset CUDA application clocks
We now store the application clock values we read when starting mdrun and reset to these values, but only when clocks have not been changed (by another process) in the meantime.
Fixes #1846.
Change-Id: i722d715320e8f4c6a5330948dcbef06bb6acf28

History
#1 - 05/23/2016 01:10 AM - Erik Lindahl
- Priority changed from Normal to Low

I would guess this is going to be virtually impossible to fix in our current setup, so I would nominate this for rejection if we're never going to fix it. When we use multiple MPI processes several of those processes might access the same GPU, and while one process might get the original value prior-to-Gromacs, the others will just see the value we already set.

Even if we add a lot of code to save the state and try to set the original value at exit, we would then depend on all MPI processes doing that setting in reverse order, or somehow detecting what value was the "true" original one with additional communication.

While that is certainly technically possible, in my opinion it simply isn't worth the work. Any other program that also relies on application clocks will always have to do the same type of detection at runtime (since the card might have been reset/rebooted), so I think doing a reset is a perfectly reasonable alternative, unless somebody has a really quick & simple fix for it.

#2 - 05/23/2016 11:59 AM - Szilárd Páll

Erik Lindahl wrote:

When we use multiple MPI processes several of those processes might access the same GPU, and while one process might get the original value prior-to-Gromacs, the others will just see the value we already set.

No, unless I'm mistaken, a single rank does the detection.

Even if we add a lot of code to save the state and try to set the original value at exit, we would then depend on all MPI processes doing that setting in reverse order, or somehow detecting what value was the "true" original one with additional communication.

I don't think "a lot of code" is needed; to me it seems all it takes is saving two integers that are anyway queried (and we already have a boolean variable to keep track of whether the clocks have been changed) and setting it back before exit.

02/23/2020
I'll have a go at it later today.

#3 - 05/23/2016 06:13 PM - Szilárd Páll
- Assignee set to Szilárd Páll

On a second thought, we may have to save the clocks prior to changing them as well as the values we change them to to be able to check whether the clocks have been changed during the run. Still rather simple, I think.

#4 - 05/25/2016 01:37 AM - Erik Lindahl
Simple case of two independent mdrun jobs using the same GPU. Each job takes 4h.

The first one starts at noon, and then second at 2pm. What will happen at 4pm when the first job finishes?

#5 - 05/25/2016 04:54 PM - Szilárd Páll

Erik Lindahl wrote:

Simple case of two independent mdrun jobs using the same GPU. Each job takes 4h.

The first one starts at noon, and then second at 2pm. What will happen at 4pm when the first job finishes?

Good point, I have not thought about node sharing. What will happen is that the simulation that finishes earlier will set back/reset clocks and risk slowing down the still other simulation. As both mdrun jobs would set the clocks to the same value (and AFAIK there is no way to detect that the cocks have been set since start), there is not much that can be done. In fact, the same issue is present in the current code which resets the clocks at cleanup (instead of preserving the state the card was in at startup).

Good that you pointed out this concurrency issue; not sure how to address it, but I think it's best if I file a separate redmine for that.

#6 - 05/25/2016 05:02 PM - Szilárd Páll
- Related to Bug #1970: GPU clock resetting at cleanup can interfere with other processes added

#7 - 05/25/2016 11:10 PM - Erik Lindahl
I think the conclusion is that we should simply increase the application clocks when we can in GROMACS, but avoid trying to reduce them. If the user is passionate about not having high clocks enabled, they can either disable our NVML support or run a script to reset it - at least that is safe and won't case impossible-to-predict side effects.

#8 - 05/26/2016 06:25 PM - Szilárd Páll

Not sure if we're talking about this issue. This issue is about considering to restore the state mdrun found the GPU in at startup.

mdrun is either assumed to be running in sanitized HPC environments, case in which this all does not matter, otherwise, if mixed desktop/workstation/shared node environments are valid usage scenarios, than we have to include in the design considerations/requirements the interference with other processes. (That's why we don't pin if we're not running on all hw threads.) It's not a matter of the user's passion whether "cleaning up" is the right thing or now, it's a matter of not being rogue -- the same way as no pinning when other processes may be present (with 0 evidence of this!) has been accepted as the right balance.

So I'd prefer to keep this open and instead solve or avoid the resetting issue. As Mark mentioned on #1970, some refactoring could make restoring even easier (or perhaps NVIDIA will provide means to detect external changes).

#9 - 05/27/2016 01:22 AM - Gerrit Code Review Bot

Gerrit received a related patchset '1' for Issue #1846.
Uploader: Erik Lindahl (erik.lindahl@gmail.com)
Change-Id: I7046082a3ddbf99a18035c24cb118f55c634d1f0
Gerrit URL: https://gerrit.gromacs.org/5901

#10 - 05/27/2016 01:29 AM - Erik Lindahl

Until NVIDIA makes the application clock a property of the process rather than the hardware, it will NOT be possible for GROMACS to solve this for the user.

If there is a good solution that you or somebody else wants, I'm perfectly happy to listen to it.

However...

1) Let's not turn this into yet another of the features where we try to outsmart the user and instead do stupid things. Note how this too was supposed to be "simple", and yet I could instantly come up with a case that where it would hurt the user instead.

2) If nobody intends to fix it (as has been the status for the last 7 months), we reject it. There is little point in keeping redmine issues open just to have a good feeling about having an entry for it, but not bothering with it. In that case we can still find it in the archives.
#11 - 05/27/2016 01:33 AM - Erik Lindahl

So, having said that: I nominate we close it with change 5901 I just uploaded. It is at least a reasonably balanced behaviour, and the code now does exactly what it claims to do.

Other suggestions are equally welcome - provided they are concrete.

#12 - 05/27/2016 04:01 PM - Berk Hess

The documentation for `nvmlDeviceSetApplicationsClocks` says:

Sets the clocks that compute and graphics applications will be running at. e.g. CUDA driver requests these clocks during context creation which means this property defines clocks at which CUDA applications will be running unless some overspec event occurs (e.g. over power, over thermal or external HW brake).

To me this sounds like the clock is only set by the driver at context creation. So a reset at the end of one mdrun process would not alter the clock for currently running applications, only for future created contexts. Since those would set application clocks themselves, if needed, we can reset application clock without issues. Either by not changing the code at all, or by querying the initial setting and restoring that.

#13 - 05/31/2016 08:25 PM - Szilárd Páll

It does not seem to be mentioned anywhere that processes can't interfere with each-other, is it?

e.g. on a K80, 562 Mhz, then mdrun sets it to 875 MHz at startup, but another process sets it to, say 810 MHz (this will likely override mdrun's 875). At exit mdrun resets which, I assumed, will place the card back into its original statem to 562 MHz. I should verify this (unless someone else has already checked?).

#14 - 06/01/2016 10:57 AM - Mark Abraham

If multiple apps have contexts and they each set the clocks, can that even be implemented by the hardware? If there's a global hardware clock, does the first context get a lock, or do subsequent contexts override?

#15 - 06/01/2016 01:27 PM - Jiri Kraus

Sorry for replying late. I will try to clarify some of the discussed questions. As I have some deadlines approaching it might take a bit till I have replies.

#16 - 06/03/2016 10:57 PM - Jiri Kraus

Application Clocks is a global setting so the setting will what ever was last set.

#17 - 06/07/2016 12:07 PM - Berk Hess

That is what I guessed.
But is it correct that they are only set at context creation, as the documentation says?
If that is the case, we have two options:

- Keep the current behavior, setting them back to default.
- Or, if possible, detect the clock before setting the mdrun application clock and set it back to that.

#18 - 06/13/2016 07:29 PM - Szilárd Páll

I don't think it's very relevant whether it happens at application startup or context creation, but given that the default is to not allow user-space process to change the application clock, I think it's even more important that we don't keep rogue behavior, e.g. resetting to the default is not particularly nice as it can affect processes launched after mdrun, not just the ones launched concurrently.

#19 - 06/14/2016 09:35 AM - Mark Abraham

If the clock is global, then we can do one of three things

- reset the clock never (which will affect any future run that doesn't set its own clock, which is probably quite a few CUDA applications)
- reset the clock always (which will affect any concurrent process, e.g. another mdrun)
- store the original clock and restore it (same problem, plus we have to write some more code)

Since none of those are perfect, and none of them can be, I suggest we go with the code we have already written at https://gerrit.gromacs.org/5901

#20 - 06/14/2016 10:46 PM - Szilárd Páll

Mark Abraham wrote:

Since none of those are perfect, and none of them can be, I suggest we go with the code we have already written at https://gerrit.gromacs.org/5901

The change you link and support is exactly the first of the possible approaches you list.

So now we have ~ as many opinions as devs who pitched in. I do agree that not resetting is the least bad, but it still falls in the rogue application category. Anyway, the easy/sloppy solution seems OK for 5.1.
Gerrit received a related patchset '1' for Issue #1846.
Uploader: Berk Hess (hess@kth.se)
Change-Id: I722d7153202e8f4c6a5330948dcbef06bb6adcf28
Gerrit URL: https://gerrit.gromacs.org/5966

Berk’s patch implements a fourth approach - reset the clock only if it hasn’t changed in the meantime, which is also quite reasonable.

- Status changed from New to Resolved

Applied in changeset ddc42de5cd85212648bbdf827688d186cbbf82843.

- Status changed from Resolved to Closed

- Target version changed from 5.x to 5.1.3