CUDA-enabled mdrun reports error in CPU runs on non-GPU host

When running a CUDA-enabled mdrun on a host without a GPU runs seem to always issue an error during cleanup:

```
$ gmx mdrun -quiet -v -ntmpi 1 -nsteps 0 -nb cpu -pme cpu
Back Off! I just backed up md.log to ./#md.log.3#
NOTE: Error occurred during GPU detection:
    CUDA driver version is insufficient for CUDA runtime version
    Can not use GPU acceleration, will fall back to CPU kernels.

 [...]  
```

Program:  gmx mdrun, version 2018-beta1-dev-20171129-2154a4f.dirty
Source file:  src/gromacs/gpu_utils/gpu_utils.cu (line 159)

Fatal error:
Unexpected CUDA error: CUDA driver version is insufficient for CUDA runtime version

For more information and tips for troubleshooting, please check the GROMACS website at http://www.gromacs.org/Documentation/Errors

Stack trace at exit:

```
#0  0x00007ffff535c510 in gmx_fatal () from /home/pszilard/projects/gromacs/gromacs-master/build_gcc5.4/bin/../lib/libgromacs.so.3
#1  0x00007ffff6098471 in isHostMemoryPinned(void*) () from /home/pszilard/projects/gromacs/gromacs-master/build_gcc5.4/bin/../lib/libgromacs.so.3
#2  0x00007fffff5fe5699 in fft5d_destroy () from /home/pszilard/projects/gromacs/gromacs-master/build_gcc5.4/bin/../lib/libgromacs.so.3
#3  0x00007fffff5fe645a in gmx_parallel_3dfft_destroy ()

cudaPointerGetAttributes() and friends are unfriendly in that they fail when no compatible driver is found. We should treat such errors as non-fatal (or even swallow) them; additionally, it would be better if we'd query and pass to the fft cleanup whether this was a GPU run (info that should be query-able form the task manager?).

On a side-note, given that CPU mode has been explicitly requested we should not be doing GPU detection anyway (or at least we should treat detection error in a less crude fashion).

Associated revisions
Revision 9d4f0df6 - 12/03/2017 06:52 PM - Mark Abraham
Fix fft5d pinning

A CUDA build on a node with no driver installed can never have selected a CUDA pinning policy, and erroneously unpinning leads to a fatal error. Instead, FFT5D now remembers whether it made pinning...

02/23/2020 1/5
possible, which can only occur when there was a driver and a valid device, so that it can unpin only when appropriate.

Removed some C++ guards and named a variable more precisely.

Noted the a TODO to make a Jenkins configuration to test this code path.

Fixes #2322

Change-Id: i50ae9cdeeb26ac0d0bd5ecf48b28b44cf0716745

Revision 315f6f83 - 12/04/2017 01:37 PM - Mark Abraham
Fix free_gpu

If a device context was not used, CUDA gives an error if we attempt to clear it, so we must avoid clearing it.

Refs #2322

Change-Id: i67b8b2d263eae9c7489a6de6f612b27496cc6c2

Revision 9ee128dc - 12/11/2017 11:08 AM - Mark Abraham
Check for GPU detection support before detecting

When a CUDA-enabled binary was run on a node with no CUDA driver available, a note was issued that the version of the CUDA driver is insufficient, which was wrong.

Fixed this by separating detection of a valid CUDA driver (or OpenCL platform) from enumerating the compatible devices. This permits a GPU-enabled build configuration to gracefully degrade to the same behaviour as a CPU-only build configuration.

Also suppressed more warnings about use of OpenCL API elements that have been deprecated but which we intend to continue to use regardless.

Also fixed confusing name of rank_local, and replaced it with a boolean that cleanly describes the required functionality.

Also fixed and simplified logic of printing the GPU report. The implementation only prints details about the node of the master rank, so there is no value in checking a variable that reflects the number of GPUs detected across all nodes.

Fixes #2322

Change-Id: i831d3c0017daf00f7bb8e3f71be5b122657d1e

Revision 6803c658 - 12/29/2017 12:08 PM - Mark Abraham
Separate canDetectGpus and findGpus further, and fix tests

Renamed detect_gpu to findGpus so that no code can silently call detect_gpu while forgetting to call the required canDetectGpus first. Some test code is updated accordingly, which should have happened earlier. The function with the new name now needs no return value, so the formerly confusing return value of zero for success is no longer present.

Shifted some more responsibilities from findGpus to canDetectGpus, so that the latter now has responsibility for ensuring that when it returns true, the former will always succeed.

Fixed tests that compile with CUDA, but cannot run unless there are visible compatible devices and a valid context.

Refs #2347, #2322, #2321

Change-Id: i34ac8b8e4c0f0dccc29e931d83c970ba945865ca7

History
Confirmed. Regression tests with -nb cpu will not have caught this, because those hosts have GPUs. We should have a post-submit build to build with CUDA on a slave with no GPUs.

The note and error happen also with -nb auto where it is clearly appropriate to run the detection, so the problem is less about that we always run the detection than it is about that the error handling needs improvement. With a CUDA build and either -nb cpu or -nb auto, failing to detect a GPU is something worthy of a one-line note in the log file that no GPUs were detected.

The fft cleanup looks like it needs improvement. I have a solution for this aspect, but it still fails later in free_gpu() for other reasons.

Further aspects need attention. I will upload a fix to free_gpu() shortly.

Note that the note/error in 2016.3 is the same if one runs a CUDA build on a machine with no CUDA devices:

```
GROMACS: gmx mdrun, version 2016.3
Executable: /opt/tcbsys/gromacs/2016.3/SSE2/bin/gmx
Data prefix: /opt/tcbsys/gromacs/2016.3/SSE2
Command line:
gmx mdrun -s reference_s -deffnm test
NOTE: Error occurred during GPU detection:
CUDA driver version is insufficient for CUDA runtime version
Can not use GPU acceleration, will fall back to CPU kernels.
```

The string about CUDA driver version is from the CUDA API. I think we should give a fatal error if the user used -nb gpu (or equivalent), be silent if -nb cpu is used, and give at most a one-liner in the log file if -nb auto is used (but then we could distinguish between "can't detect devices, driver version is insufficient" from "can detect devices, and none were found", by reacting to the CUDA error that is producing the string here).

The above patch resolves the issues for release-2018, apart from the output to the user about failure to detect GPUs, which is common with at least release-2016. I'm not yet sure if the same fix is applicable to both. But if so, then I will re-target the remaining items for this issue for release-2016.
The note and error happen also with -nb auto where it is clearly appropriate to run the detection, so the problem is less about that we always run the detection than it is about that the error handling needs improvement.

Indeed, but my comment was originally more referring to the fact that the command line explicitly maps all tasks to CPUs, there is little point it detecting GPUs at all -- which used to be the behavior that for some reason got changed relatively recently.

With a CUDA build and either -nb cpu or -nb auto, failing to detect a GPU is something worthy of a one-line note in the log file that no GPUs were detected.

-nb auto yes but with explicit -nb cpu -pme cpu (or "-nb cpu" with DD) not sure...

#9 - 12/10/2017 09:14 AM - Mark Abraham
Szilárd Páll wrote:

Mark Abraham wrote:

The note and error happen also with -nb auto where it is clearly appropriate to run the detection, so the problem is less about that we always run the detection than it is about that the error handling needs improvement.

Indeed, but my comment was originally more referring to the fact that the command line explicitly maps all tasks to CPUs, there is little point it detecting GPUs at all -- which used to be the behavior that for some reason got changed relatively recently.

My reasoning was that if we are going to print a report of the hardware detected on the node, then why should that report change for the same binary and node depending on the user's choice of -nb? That's a surprising behaviour.

With a CUDA build and either -nb cpu or -nb auto, failing to detect a GPU is something worthy of a one-line note in the log file that no GPUs were detected.

-nb auto yes but with explicit -nb cpu -pme cpu (or "-nb cpu" with DD) not sure...

Assuming the implementation of hardware detection is robust to run-time errors, not slow, and doesn't issue unwarranted errors and warnings, then there's very little difference whether -nb cpu runs attempt GPU device detection. If the hardware report merely reports the detected hardware (rather than also mixing in aspects of the user's input), then the user has more of a chance to realise that they're running with mdrun flags they did not intend, or on a node they did not intend. It's also slightly more complex to have to write and maintain the code to look through all possible ways the user might have required a run to use a GPU (now -nb, -pme, -gpu_id, -tasks, and in future maybe other possibilities), which means that the code handling task assignment is constrained to have done some input parsing and evaluate some logic before hardware detection can take place. That complicates the code that must run at setup time because it imposes ordering requirements.

I am not aware of an advantage for trying to skip GPU detection. If there is, why stop at -nb and -pme? Why not skip detection if we have a group-scheme tpr, or a rerun with energy groups? IMO the answer in all cases is that the simplest code that should be implemented and tested well enough to always work is to always call the detection code (which does nothing if it's not a GPU build).

#10 - 12/10/2017 09:40 AM - Gerrit Code Review Bot
Gerrit received a related DRAFT patchset '1' for Issue #2322.
Uploader: Mark Abraham (mark.j.abraham@gmail.com)
Change-Id: gromacs~release-2018~I831d3c0017dafc00f7bb82e3f71be5b122657d1e
Gerrit URL: https://gerrit.gromacs.org/7315

#11 - 12/10/2017 12:17 PM - Mark Abraham
- Status changed from In Progress to Fix uploaded

non-draft now also up

#12 - 12/11/2017 12:06 PM - Erik Lindahl
- Status changed from Fix uploaded to Resolved

#13 - 12/11/2017 12:06 PM - Erik Lindahl
- Status changed from Resolved to Closed

#14 - 12/14/2017 01:36 PM - Gerrit Code Review Bot
Gerrit received a related patchset '1' for Issue #2322.
Uploader: Mark Abraham (mark.j.abraham@gmail.com)
Change-Id: gromacs~release-2018~I34acf8be4c0f0dcc29e931d83c970ba945865ca7
There may be a remaining issue here - for a CUDA build, many things can stop a GPU being usable. We're now OK if there's no driver (cudaGetDriverVersion() returns 0 in that case), but one of the errors we're seeing is "driver version insufficient for runtime" e.g. on login.biophysics.kth.se. One can call cudaGetRuntimeVersion(), but what would we do with it?

Once we can detect that we are in this case, I think mdrun should swallow the error and proceed to doing a CPU-only run.

At some point we need to accept that it's not necessarily Gromacs' responsibility to take over cluster management. It's important that we can run in auto mode when no GPU is available, and that works on e.g. login.biophysics.kth.se now. However, if there is a GPU available, and a driver installed, but something is so wrong that we can't use it, I can accept that we simply stop execution.

Alternatively, somebody should take it on and start working on it, but given that we don't have any assignee, I'd like to avoid this becoming another one of those issues that stays open for a year without anybody planning to work on it - that contributes to other bugs drowning!

Thus, if nobody takes it on the next week, I'd vote for closing this.

Agree - I think there's no known problems in 2016, and they're not scientific correctness problems. 2018 is fine.