GROMACS - Bug #2390

GROMACS build system should check for valid nvcc flags before use

01/23/2018 08:58 AM - Mark Abraham

Status:	Feedback wanted		
Priority:	Normal		
Assignee:			
Category:	build system		
Target version:	2021		
Affected version - extra info:	5.x master	**Difficulty:**	uncategorized
Affected version:	2016.4		

Description

We specify many cross-compilation targets in our default build, including for CC 2.0, for which NVIDIA removed support in CUDA 9.0. Thus our users might see

```
"nvcc fatal   : Unsupported gpu architecture 'compute_20'
nvcc fatal   : Unsupported gpu architecture 'compute_20'
CMake Error at
libgromacs_generated_nbnxn_cuda_data_mgmt.cu.o.Release.cmake:218 (message):"
```

which I have seen reported for 5.1.5 and 2016.3.

The minimal fix (present in 2016.4) is to remove sm_20 (and compute_20?) from our nvcc invocations for CUDA 9.0.

However, we should probably invest in better CUDA support in our CMake. Rather than checking CUDA version numbers, we should check for feature support, as we do for other compiler flags, so that we will not require the use of compiler flags that we have not checked will work.

For a further example, the CUDA 9.1.85 release notes mention that CC 7.0 support still does not work on MacOS, so there is no version of GROMACS that works with CUDA 9.0 on that platform - in all released versions, either we require sm_20 or require sm_70, and compilation fails for different reasons.

Fortunately, the recent addition to 2018 that checks that nvcc works will be a useful template for fixing it in that branch. Maybe we could consider back-porting that to 2016 branch.

Related issues:

Related to GROMACS - Feature #2126: implement native CUDA support in CMake	**New**
Related to GROMACS - Task #2505: increase cmake reqirement for GROMACS 2020	**Closed**
Related to GROMACS - Bug #2561: Incorrectly getting "Enabling single compilat...	**Closed**

History

#1 - 01/23/2018 01:59 PM - Szilárd Páll

Mark Abraham wrote:

> We specify many cross-compilation targets in our default build, including for CC 2.0, for which NVIDIA removed support in CUDA 9.0. Thus our users might see
>
> [...]
>
> which I have seen reported for 5.1.5 and 2016.3.
>
> The minimal fix (present in 2016.4) is to remove sm_20 (and compute_20?) from our nvcc invocations for CUDA 9.0.

It should work, it was fixed by Jiri in the initial CUDA 9 support change in 97f9f399.

> However, we should probably invest in better CUDA support in our CMake. Rather than checking CUDA version numbers, we should check for feature support, as we do for other compiler flags, so that we will not require the use of compiler flags that we have not checked will work.
>
> For a further example, the CUDA 9.1.85 release notes mention that CC 7.0 support still does not work on MacOS, so there is no version of

> GROMACS that works with CUDA 9.0 on that platform - in all released versions, either we require sm_20 or require sm_70, and compilation fails for different reasons.

OK, that's annoying but hardly an important target (unless I"m mistaken Apple hasn't shipped any NVIDIA GPU in the last few years).

> Fortunately, the recent addition to 2018 that checks that nvcc works will be a useful template for fixing it in that branch. Maybe we could consider back-porting that to 2016 branch.

Sure, but it's ugly and fairly high LOC for for something that we'd ideally have better CMake support for via try_compile (and the native CUDA support might actually be able to do that).

#2 - 01/23/2018 02:47 PM - Mark Abraham

Szilárd Páll wrote:

> Mark Abraham wrote:
>
> > We specify many cross-compilation targets in our default build, including for CC 2.0, for which NVIDIA removed support in CUDA 9.0. Thus our users might see
> >
> > [...]
> >
> > which I have seen reported for 5.1.5 and 2016.3.
> >
> > The minimal fix (present in 2016.4) is to remove sm_20 (and compute_20?) from our nvcc invocations for CUDA 9.0.
>
> It should work, it was fixed by Jiri in the initial CUDA 9 support change in 97f9f399.
>
> > However, we should probably invest in better CUDA support in our CMake. Rather than checking CUDA version numbers, we should check for feature support, as we do for other compiler flags, so that we will not require the use of compiler flags that we have not checked will work.
> >
> > For a further example, the CUDA 9.1.85 release notes mention that CC 7.0 support still does not work on MacOS, so there is no version of GROMACS that works with CUDA 9.0 on that platform - in all released versions, either we require sm_20 or require sm_70, and compilation fails for different reasons.
>
> OK, that's annoying but hardly an important target (unless I"m mistaken Apple hasn't shipped any NVIDIA GPU in the last few years).

Sure, but it shows that our approach risks being broken anywhere that we don't have Jenkins or devs testing - and we don't have CUDA 9 in Jenkins at all, yet.

> Fortunately, the recent addition to 2018 that checks that nvcc works will be a useful template for fixing it in that branch. Maybe we could consider back-porting that to 2016 branch.
>
> Sure, but it's ugly and fairly high LOC for for something that we'd ideally have better CMake support for via try_compile (and the native CUDA support might actually be able to do that).

https://redmine.gromacs.org/projects/gromacs/repository/revisions/master/entry/cmake/gmxManageGPU.cmake#L275 already does most of the work of calling execute_process(). All we need is to wrap that into gmx_nvcc_try_compile(RESULT_VAR SRC_FILE | SRC_STRING) that picks up the variables that FindCUDA.cmake manages.

> (and the native CUDA support might actually be able to do that).

Likely it does. But that isn't effective unless we'd choose to require cmake 3.8 for (at least) CUDA support in GROMACS 2019. We could do that, but we'd want to identify a few useful things, and check that at least some LTS/stable distro releases have that cmake version. (Note that FindCUDA.cmake and the native support are totally different things.) It would be extra work for us to support both in the same branch, but if we did support native compilation via cmake 3.8 in GROMACS 2019, then we could say to a hypothetical user of 2019 who's also got e.g. CUDA from 2020 that doesn't support CC 3.0 that they now have options: use the latest GROMACS, use a not-latest CUDA, or update cmake for better automation.

Meanwhile, 2018 and earlier versions will rot the moment NVIDIA de-supports CC 3.0, for example. (We should ask NVIDIA what their current thoughts are for that - if they have already decided that they are dropping CC 3.0 for CUDA 10, then we have more reason to attempt a proper fix in 2018 branch.)

Also, questions about whether CC=5.3 is or is not only available on Tegra or should be a compilation target get a bit easier to handle - ask the compiler and if the user left us a choice, compile for whatever is possible.

#3 - 03/05/2018 01:34 PM - Mark Abraham

- Related to Feature #2126: implement native CUDA support in CMake added

#4 - 05/18/2018 02:54 PM - Mark Abraham

- Related to Task #2505: increase cmake reqirement for GROMACS 2020 added

#5 - 07/18/2018 03:31 PM - Mark Abraham

- Subject changed from older GROMACS can't build with CUDA 9.0 to GROMACS build system should check for valid nvcc flags before use

- Target version set to 2018.3

CUDA 9.x no longer supports devices of compute capability (CC) 2.x, and we have a version-number check in our build system that prevents trying to compile code for those devices. But we need to reform the construction of the compilation targets so that current GROMACS can build and run with future CUDA versions even where e.g. CC 3.x is no longer supported. That's particularly useful if we stop maintaining the build system and making new releases after ~two years, otherwise users will have to know and use the particular GMX_CUDA_TARGET_* flags.

Like many other parts of the build system, the nvcc support should use feature checks rather version checks, because the nvcc capabilities differ also e.g. with OS. Erik already wrote the code for checking that nvcc is happy with the compiler version, so this should be straightforward. Szilard, can you please work on this for 2018.3?

#6 - 07/18/2018 03:33 PM - Mark Abraham

- Related to Bug #2561: Incorrectly getting "Enabling single compilation unit for the CUDA non-bonded module." when using Cuda 9.0 and gromacs 2018.1 added

#7 - 08/20/2018 11:07 AM - Mark Abraham

I made proposals relevant for this at #2561

#8 - 08/21/2018 10:39 AM - Paul Bauer

Any update on this one for 2018.3?

#9 - 08/22/2018 11:24 AM - Paul Bauer

- Target version changed from 2018.3 to 2018.4

bumped to next patch release

#10 - 11/08/2018 03:50 PM - Paul Bauer

- Target version changed from 2018.4 to 2018.5

Nothing has happened here, bumped once again, or can this be closed because Fermi support got removed in 2019?

#11 - 11/09/2018 06:40 PM - Szilárd Páll

- Status changed from New to Feedback wanted

In my opinion there is little benefit for us to write and maintain custom flag checker code just for nvcc, the same effort is better spent on moving to native CUDA cmake support (even if only partial for cmake >=3.8).

#12 - 01/15/2019 01:24 PM - Mark Abraham

- Target version changed from 2018.5 to 2020

#13 - 01/15/2019 01:25 PM - Mark Abraham

We aren't going decide about whether to do this work until we've decided on whether we will use native CUDA support in master branch.

#14 - 01/30/2019 06:16 PM - Mark Abraham

On gmx-users (https://mailman-1.sys.kth.se/pipermail/gromacs.org_gmx-users/2019-January/124029.html) I saw a report that with a Quadro P6000 that there is no driver available before CUDA 9.2. They were having other issues getting 2019 and 2018 to run (possibly related to #2762), and when they tried even older versions with CUDA 9.2, they got

```
5.1.5  +  9.2  =>Installation make: nvcc fatal   : Unsupported gpu architecture 'compute_20'*
2016.2  +  9.2  =>  Installation make: nvcc fatal   : Unsupported gpu architecture 'compute_20'*
```

That is, our use of nvcc means that users can't run older versions of GROMACS on newer CUDA toolkits. IMO that's a minor problem, but it has an easy fix (test that nvcc flags work before choosing to use them, just as we do with other compiler flags).

#15 - 12/20/2019 12:13 PM - Paul Bauer

- Target version changed from 2020 to 2021