Incorrectly getting "Enabling single compilation unit for the CUDA non-bonded module." when using Cuda 9.0 and gromacs 2018.1

The Cmake code in cmake/gmxManageGPU.cmake incorrectly ends up in the message(STATUS "Enabling single compilation unit for the CUDA non-bonded module...." part of the if-stmt when building with CUDA 9.9. It does this despite correctly identifying CUDA 9.0 and using only >= 3.0 cuda targets.

Since the code in cmake/gmxManageNvccConfig.cmake correctly sets GMX_CUDA_NVCC_FLAGS regardless of doing autodetection or using user specified SM/COMPUTE targets, the code in cmake/gmxManageGPU.cmake macro(gmx_gpu_setup) can be simplified to:

```cmake
if (GMX_GPU AND NOT GMX_CLANG_CUDA)
    gmx_check_if_changed(_gmx_cuda_target_changed GMX_CUDA_NVCC_FLAGS)
    if(_gmx_cuda_target_changed OR NOT GMX_GPU_DETECTION_DONE)
        if(GMX_CUDA_NVCC_FLAGS MATCHES "._2[01]"
            message(STATUS "Enabling single compilation unit...
        else()
            message(STATUS "Enabling multiple compilation
        ...
```

(This is the problem I incorrectly targeted in issue 2560)

**Related issues:**
Related to GROMACS - Bug #2390: GROMACS build system should check for valid n...

**Feedback wanted**

**Associated revisions**
Revision 1982bad5 - 08/21/2018 01:18 AM - Szilárd Páll
Disable single compilation unit with CUDA 9.0

CUDA 9.0 does not support CC 2.x, hence even with the default list of targeted arch, multiple compilation units are enabled.

Fixes #2561
Change-Id: i741081da39539f211fad32bda5c1f1dcccc437f19

**History**

#1 - 07/17/2018 10:37 AM - Szilárd Páll
I can't reproduce the issue. Can you please provide full command line invocation?

#2 - 07/17/2018 11:02 AM - Åke Sandgren

```
cmd line used:
cmake ..gromacs-2018.2 -DCMAKE_INSTALL_PREFIX=/scratch/ake/gromacs/inst -DCMAKE_VERBOSE_MAKEFILE=ON
-DCMAKE_BUILD_TYPE=Release -DGMX_EXTERNAL_BLAS=ON -DGMX_EXTERNAL_LAPACK=ON -DGMX_X11=OFF -DGMX_OPENMP=ON
-DGMX_MPI=OFF -DGML_CUDA_GPU=ON -DCUDA_TOOLKIT_ROOT_DIR=/hpc2n/eb/software/Compiler/GCC/7.3.0-2.30/CUDA/9.2.88
-DGMX_BLAS_USER="/hpc2n/eb/software/Compiler/GCC/7.3.0-2.30/OpenBLAS/0.3.1/lib/libopenblas.a"
-DGMX_LAPACK_USER="/hpc2n/eb/software/Compiler/GCC/7.3.0-2.30/OpenBLAS/0.3.1/lib/libopenblas.a" > cmake.out 2>&1
```
grep 'Enabling single compilation' cmake.out
-- Enabling single compilation unit for the CUDA non-bonded module. Multiple compilation units are not compatible with CC 2.x devices, to enable the feature specify only CC >=3.0 target architectures in GMX_CUDA_TARGET_SM/GMX_CUDA_TARGET_COMPUTE.

#3 - 07/17/2018 01:46 PM - Szilárd Páll
Åke Sandgren wrote:

```
make ... -DCMAKE_INSTALL_PREFIX=/scratch/ake/gromacs/inst -DCMAKE_VERBOSE_MAKEFILE=ON
-DCMAKE_BUILD_TYPE=Release -DGMX_EXTERNAL_BLAS=ON -DGMX_EXTERNAL_LAPACK=ON -DGMX_X11=OFF
-DCMAKE_BUILD_TYPE=Release -DGMX_EXTERNAL_BLAS=ON -DGMX_EXTERNAL_LAPACK=ON -DGMX_X11=OFF
-DCUDA_TOOLKIT_ROOT_DIR=/hpc2n/eb/software/Compiler/GCC/7.3.0-2.30/CUDA/9.2.88
-DGMX_BLAS_USER="/hpc2n/eb/software/Compiler/GCC/7.3.0-2.30/OpenBLAS/0.3.1/lib/libopenblas.a"
-DGMX_LAPACK_USER="/hpc2n/eb/software/Compiler/GCC/7.3.0-2.30/OpenBLAS/0.3.1/lib/libopenblas.a" > cmake.out 2>&1
```

This does not contain either of the GMX_CUDA_TARGER_SM or GMX_CUDA_TARGER_COMPUTE passed to cmake to restrict the targeted architectures.

#4 - 07/17/2018 01:47 PM - Szilárd Páll
- Status changed from New to Feedback wanted

#5 - 07/17/2018 11:23 PM - Åke Sandgren
Exactly, it doesn't contain them, thus the code should take the default of using what the CUDA supports. And there is code in there to figure this out based on CUDA version, but it isn't used. The result is just ignored.

With the code shown in the original post, everything works in both cases, with and without GMX_CUDA_TARGET_SM or GMX_CUDA_TARGET_COMPUTE

#6 - 07/18/2018 01:41 PM - Mark Abraham
- Description updated

#7 - 07/18/2018 03:33 PM - Mark Abraham
- Target version set to 2018.3

With

```
export CC=gcc-6
export CXX=g++-6
```

```
cmake ... -DCMAKE_INSTALL_PREFIX=/opt/gromacs/$version -G Ninja
```

I see

```
...
-- Enabling single compilation unit for the CUDA non-bonded module. Multiple compilation units are not compatible with CC 2.x devices, to enable the feature specify only CC >=3.0 target architectures in GMX_CUDA_TARGET_SM/GMX_CUDA_TARGET_COMPUTE.
...
```

which is the expected result (supporting 2.x devices with CUDA < 9 requires a single compilation unit in 2018; CC 2.x is now not supported at all in master) if perhaps a little tricky for users to understand whether they should act. (If they know about compute capabilities, it's not clear that by default GROMACS builds for all of them, and if they don't know about compute capabilities then nothing makes sense.)

```
cmake ... -DCMAKE_INSTALL_PREFIX=/opt/gromacs/$version -G Ninja -DGMX_CUDA_TARGET_SM=30 -DGMX_CUDA_TARGET_COMPUTE=30
```

enables multiple compilation units, as expected since that is now possible.

With the reform proposed in #2390, then we can avoid some aspects of such problems in future, but for now Ake's proposal looks good to me. Szilard can you prepare a patch, please?

#8 - 07/18/2018 03:33 PM - Mark Abraham
- Related to Bug #2390: GROMACS build system should check for valid nvcc flags before use added

#9 - 07/18/2018 07:30 PM - Szilárd Páll
Åke Sandgren wrote:

```
02/23/2020
```
Exactly, it doesn't contain them, thus the code should take the default of using what the CUDA supports.

Sorry, forgot that CUDA 9.0 skips generating CC 2.0 code.

IIRC the reason why the current code does not just test CUDA_NVCC_FLAGS is that the compiler can generate code for CC 2.0 despite a command line that does not contain "-201". For the same reason, the proposed change could also lead to not triggering single compilation unit for some CUDA version <9.0 for a CUDA_NVCC_FLAGS string that implicitly triggers CC 2.0 code generation. I'd rather make the code explicit and add a CUDA version check to the conditional than take the risk (or trawl through the nvcc docs).

#10 - 07/19/2018 01:06 AM - Mark Abraham

If we test for the features supported by nvcc then we will have a good way to guess whether it is at all capable of e.g. generating code for CC 2.0 (which Szilárd identifies as a risk). If the user chooses particular CC, then we check in cmake that this will work. If the user chooses multiple compilation units but CC 2.0 is supported, then cmake should fail (inconsistent input). If the user chooses nothing and CC 2.0 is supported, then we default to single compilation unit. If the user chooses nothing and CC 2.0 is not supported, then we default to multiple compilation units.

I don't see any extra value delivered by any CUDA version checks. Different OS first support newer CC in different CUDA versions, so such checks are brittle. They can't help the build work with unknown future CUDA versions. This is the same thinking behind taking advantage of compiler and glibc feature-reporting gear, and testing functionality at configure time, rather than hard coding specific version checks.

#11 - 07/19/2018 04:49 PM - Gerrit Code Review Bot

Gerrit received a related patchset ‘1’ for Issue #2561.
Uploader: Szilárd Páll (pall.szilard@gmail.com)
Change-Id: gromacs~release-2018~I741081da39539f211fad32bda5c1f1dccc4378f9
Gerrit URL: https://gerrit.gromacs.org/8106

#12 - 07/19/2018 04:52 PM - Szilárd Páll

Mark Abraham wrote:

If we test for the features supported by nvcc then we will have a good way to guess whether it is at all capable of e.g. generating code for CC 2.0 (which Szilárd identifies as a risk).

How do you suggest testing individual features of nvcc? (Writing a bunch of execute_process calls with manual regex matching is fragile and tedious IMO).

If the user chooses particular CC, then we check in cmake that this will work. If the user chooses multiple compilation units but CC 2.0 is supported, then cmake should fail (inconsistent input). If the user chooses nothing and CC 2.0 is supported, then we default to single compilation unit. If the user chooses nothing and CC 2.0 is not supported, then we default to multiple compilation units.

What we don't check for is default behavior of nvcc, e.g. when -gencode flags are cleared. I've just checked and we don't accept empty GMX_CUDA_TARGET_* to clear the list of targeted arch, so only the fully manual override would be able to trigger that case (and that can be triggered regardless of how this fix is implemented). Admittedly, this case would however be caught by a compile-time error, so there's no risk of incorrect results.

I don't see any extra value delivered by any CUDA version checks.

Check. a Single one would be needed for this specific fix:

```cpp
if((CUDA_VERSION VERSION_LESS "9.0") AND
 (NOT GMX_CUDA_TARGET_SM AND NOT GMX_CUDA_TARGET_COMPUTE) OR
 (GMX_CUDA_TARGET_SM MATCHES "2[01]" OR GMX_CUDA_TARGET_COMPUTE MATCHES "2[01]"))
```

and this doesn't need to change later as CC 2.0 support won't be re-introduced later, I think.

Different OS first support newer CC in different CUDA versions, so such checks are brittle. They can't help the build work with unknown future CUDA versions. This is the same thinking behind taking advantage of compiler and glibc feature-reporting gear, and testing functionality at configure time, rather than hard coding specific version checks.

Not sure what you mean -- hardware support is purely CUDA version-dependent, operating systems / distros have no say in it. I don't think this is similar at all to glibc-related checks, CUDA has a single implementation with clear deprecation version cutoffs.

Back to the main point: I think for this specific fix the change proposed by Åke has the advantage of slight simplification, but the slight drawback that it does gmx_check_if_changed( gmx_cudea_target_changed GMX_CUDA_NVCC_FLAGS) which is a slight overkill in that any change in any of the C++ or nvcc flags will trigger a status message.

#13 - 07/30/2018 12:57 PM - Mark Abraham
Szilárd Pál wrote:

Mark Abraham wrote:

If we test for the features supported by nvcc then we will have a good way to guess whether it is at all capable of e.g. generating code for CC 2.0 (which Szilard identifies as a risk).

How do you suggest testing individual features of nvcc? (Writing a bunch of execute_process calls with manual regex matching is fragile and tedious IMO).

We wrap the execute_process() call of nvcc in a function that takes the name of a cache variable to set with the result of the call to nvcc, passing an optional extra flag. The current check for basic functionality of nvcc is just the special case of no extra flag. See e.g. https://redmine.gromacs.org/projects/gromacs/repository/revisions/master/entry/cmake/CheckCXXCompilerFlag.cmake and the way that cmake's Modules/CheckCXXSourceCompiles.cmake wraps a try_compile(). So we end up with a collection of cache variables named like NVCC_SUPPORTS_CC_2_0 etc.

If the user chooses particular CC, then we check in cmake that this will work. If the user chooses multiple compilation units but CC 2.0 is supported, then cmake should fail (inconsistent input). If the user chooses nothing and CC 2.0 is supported, then we default to single compilation unit. If the user chooses nothing and CC 2.0 is not supported, then we default to multiple compilation units.

What we don't check for is default behavior of nvcc, e.g. when -codegen flags are cleared. I've just checked and we don't accept empty GMX_CUDA_TARGET_* to clear the list of targeted arch, so only the fully manual override would be able to trigger that case (and that can be triggered regardless of how this fix is implemented). Admittedly, this case would however be caught by a compile-time error, so there's no risk of incorrect results.

I don't see any extra value delivered by any CUDA version checks.

Check. a Single one would be needed for this specific fix:

[...]

and this doesn't need to change later as CC 2.0 support won't be re-introduced later, I think.

Yeah something like that works, but it does not address the same problem that we will have in the future with CC 3.0 being deprecated and removed.

Different OS first support newer CC in different CUDA versions, so such checks are brittle. They can't help the build work with unknown future CUDA versions. This is the same thinking behind taking advantage of compiler and glibc feature-reporting gear, and testing functionality at configure time, rather than hard coding specific version checks.

Not sure what you mean -- hardware support is purely CUDA version-dependent, operating systems / distros have no say in it.

Different CUDA versions support different compute capabilities on different OS. See #2390. We should test the compiler for feature capabilities.

I don't think this is similar at all to glibc-related checks, CUDA has a single implementation with clear deprecation version cutoffs.

They're not clear in advance, so we are currently shipping code that will not compile on some future version of CUDA. AFAIK there is no published deprecation/removal plan for CC 3.0, for example. Using feature checks on the compiler is more future proof.

Back to the main point: I think for this specific fix the change proposed by Åke has the advantage of slight simplification, but the slight drawback that it does gmx_check_if_changed(gmx_cuda_target_changed GMX_CUDA_NVCC_FLAGS) which is a slight overkill in that any change in any of the C++ or nvcc flags will trigger a status message.

#14 - 08/13/2018 02:14 PM - Szilárd Pál

- Status changed from Feedback wanted to Resolved
- Assignee set to Szilárd Pál

#15 - 08/15/2018 11:56 AM - Paul Bauer

Hello, can this be closed now, or are there still issue open for discussion here? If so, I'll move the target to 2019.

#16 - 08/20/2018 11:07 AM - Mark Abraham

Szilard's patch resolves the issue in the short term, but IMO there is still a long term problem that has not been resolved. Let's close this, and I will note in #2390 that there is more consideration required, either for 2018.x or 2019.
#17 - 08/21/2018 01:30 AM - Szilárd Páll
Applied in changeset 1982bad52980f9f21ecb6fba1791eb07cb11ca0f.

#18 - 08/21/2018 10:33 AM - Paul Bauer
- Status changed from Resolved to Closed