GROMACS - Bug #1616

configuration should check that the compiler will work with nvcc

10/04/2014 10:21 AM - Mark Abraham

Status: Closed
Priority: Normal
Assignee: Mark Abraham
Category: build system
Target version: 2018
Affected version - extra info: all versions since 4.6
Affected version: 5.0.2

Difficulty: uncategorized

Description

We've had a couple of gmx-users posts complain of a build-time error from nvcc when someone uses a compiler that their CUDA version does not support. One of them thought that it meant that we don't support icc 14, for example.

We can check for this with CMake at configure time, simply by compiling a test program. We do this for lots of other functionality, so I think we should do it for CUDA also.

I would suggest that a failure is fatal (and descriptive), since the decision of which compiler and which CUDA version to use needs user involvement. Also suggest release-5-0 branch. Should we suggest the work-around of hacking off the check in the CUDA headers?

Related issues:

- Related to GROMACS - Bug #2583: CUDA host compiler check is not retriggered
- Closed

Associated revisions

Revision 5e9bcbcf - 01/05/2018 04:58 PM - Erik Lindahl

Test that nvcc/host compiler combo works

Compile a trivial CUDA program during CMake time to catch both unsupported nvcc/host compiler version combinations and other unknown errors.

Fixes #1616.

Change-Id: I3cc55e4d0db9d6eb01e8a7cd8916cc7a7a1e21fd

Revision 029e1e95 - 01/28/2019 05:49 PM - Mark Abraham

Stop trying to check nvcc on Windows

The execute_process() call used in this check is not constructed to work on Windows, so we should not run it there.

Refs #1616

Change-Id: I2103b78203f71d3f68b54898dd03c8fe0eb0fa4c

History

#1 - 06/20/2015 10:25 AM - Gerrit Code Review Bot

Gerrit received a related patchset '1' for Issue #1616.
Uploader: Erik Lindahl (erik.lindahl@gmail.com)
Change-Id: I38a4a7f59b61ae6628d9d181ab52de27f5d35927
Gerrit URL: https://gerrit.gromacs.org/4749

#2 - 06/20/2015 10:26 AM - Erik Lindahl

- Status changed from New to Fix uploaded

#3 - 04/03/2016 10:58 PM - Erik Lindahl

Duplicate/related to #1248.
Since this issue is now more than a year old, I think it's time we decide whether somebody wants to invest significant time into testing it properly, or whether we should forget about it.

No feedback from anybody for a week. We'll wait one more week for people to volunteer, but if nobody steps out we'll simply decide that we cannot debug the entire build landscape for nvcc and arbitrary host compilers in the Gromacs configuration files, but leave that to the user and close this issue.

I'll try to have a go at this

- Assignee set to Mark Abraham

Gerrit received a related patchset '1' for Issue #1616.
Uploader: Erik Lindahl (erik.lindahl@gmail.com)
Change-Id: gromacs~release-2018~I3cc55e4d0db9d6eb01e8a7cd8916cc7a7a1e21fd
Gerrit URL: https://gerrit.gromacs.org/7357

Target version set to 2018

- Status changed from Accepted to Fix uploaded

I think that would be the right solution. As noted on gerrit, this would be nice and user-friendly, but it is not in the scope of the GROMACS build system to shield users from any toolchain incompatibility at the cost of fragile code.

Can't find where, but it's been discussed in the past that without a CUDA-enabled try_compile, this would be hard to test reliably and I believe that is why this issue has stalled. If we'd adopt native CUDA support in CMake we should be able to just use try_compile, I think.

Erik's proposed code is not fragile, ie likely to fall apart under use. The worst it can do is test nvcc and conclude it is broken and tell the user to use different things when it would actually work for compiling GROMACS. That weighs against the users I have seen conclude that compiler x version y is not supported by GROMACS, because they can't understand the message from nvcc.

That would require adopting cmake 3.8 for at least CUDA builds (which we could potentially consider for GROMACS 2019).
Szilárd Páll wrote:

Erik Lindahl wrote:

> we'll simply decide that we cannot debug the entire build landscape for nvcc and arbitrary host compilers in the Gromacs configuration files, but leave that to the user and close this issue.

I think that would be the right solution. As noted on gerrit, this would be nice and user-friendly, but it is not in the scope of the GROMACS build system to shield users from any toolchain incompatibility at the cost of fragile code.

Erik’s proposed code is not fragile, ie likely to fall apart under use. The worst it can do is test nvcc and conclude it is broken and tell the user to use different things when it would actually work for compiling GROMACS.

That's the false positive that I I thought it made the code/assumptions of the code fragile.

That weighs against the users I have seen conclude that compiler x version y is not supported by GROMACS, because they can't understand the message from nvcc.

I'm not sure a few user complaints outweighs the cost of optimistic assumptions with little time to test in the wild together with the non-negligible amount of code. As I said I lean toward preferring more confidence (which would require knowing/checking what the FindCUDA wrapper does) and (/or at least) leaner code. More confidence could be gained by adding such code much earlier during a release cycle, hence the suggestion on gerrit that this might be better suited for master.

Can't find where, but it's been discussed in the past that without a CUDA-enabled try_compile, this would be hard to test reliably and I believe that is why this issue has stalled. If we'd adopt native CUDA support in CMake we should be able to just use try_compile, I think.

That would require adopting cmake 3.8 for at least CUDA builds (which we could potentially consider for GROMACS 2019).

In master we could also adopt the native CUDA-based FindCUDA fallback first that would allow not requiring cmake 3.8 right away; assuming try_compile for .cu files works, when cmake is new enough we could use that to test compatibility. Likely far less code and more robust.

Mark Abraham wrote:

Szilárd Páll wrote:

> Erik's proposed code is not fragile, ie likely to fall apart under use. The worst it can do is test nvcc and conclude it is broken and tell the user to use different things when it would actually work for compiling GROMACS.

That's the false positive that I I thought it made the code/assumptions of the code fragile.

If one is going to test nvcc, you choose the simplest thing that will work precisely to reduce the chance of false positives.

That weighs against the users I have seen conclude that compiler x version y is not supported by GROMACS, because they can't understand the message from nvcc.

I'm not sure a few user complaints outweighs the cost of optimistic assumptions with little time to test in the wild together with the non-negligible amount of code. As I said I lean toward preferring more confidence (which would require knowing/checking what the FindCUDA wrapper does)

It calls execute_process(nvcc and stuff) like Erik's code does.

and (/or at least) leaner code. More confidence could be gained by adding such code much earlier during a release cycle, hence the suggestion on gerrit that this might be better suited for master.

Yes that's reasonable.

Can't find where, but it's been discussed in the past that without a CUDA-enabled try_compile, this would be hard to test reliably and I believe that is why this issue has stalled. If we'd adopt native CUDA support in CMake we should be able to just use try_compile, I
think.

That would require adopting cmake 3.8 for at least CUDA builds (which we could potentially consider for GROMACS 2019).

In master we could also adopt the native CUDA-based FindCUDA fallback first that would allow not requiring cmake 3.8 right away; assuming try_compile for .cu files works, when cmake is new enough we could use that to test compatibility. Likely far less code and more robust.

FindCUDA is the work-around that is needed when CUDA is not a first-class language in cmake. When it is, you need the functionality in the cmake C++ code. There’s no “native CUDA-based FindCUDA fallback”. See examples at https://devblogs.nvidia.com/parallelforall/building-cuda-applications-cmake/

#16 - 01/05/2018 04:58 PM - Erik Lindahl
- Status changed from Fix uploaded to Resolved

#17 - 01/05/2018 04:58 PM - Erik Lindahl
- Status changed from Resolved to Closed

#18 - 07/23/2018 01:01 PM - Szilárd Pál
- Related to Bug #2583: CUDA host compiler check is not retriggered added

#19 - 01/10/2019 10:18 AM - Gerrit Code Review Bot
Gerrit received a related patchset ‘1’ for Issue #1616.
Uploader: Mark Abraham (mark.i.abraham@gmail.com)
Change-Id: gromacs~release-2019~I2103b78203f71d3f68b54898dd03c8fe0eb0fa4c
Gerrit URL: https://gerrit.gromacs.org/8950