GROMACS - Task #2831

Bump required version numbers of infrastructure for 2020

01/14/2019 10:08 AM - Mark Abraham

Status:	New	
Priority:	Normal	
Assignee:	Mark Abraham	
Category:	build system	
Target version:	2020-infrastructure-update-post-beta1	
Difficulty:	uncategorized	

Description

For GROMACS 2020, we should choose some new minimum supported versions to target, remembering that they start being relevant to users when we release in 2020. Remember, there is a trade-off between what new things we can do (because we support a smaller range of things) vs what things we might require some users to do (because GROMACS no longer builds out of the box for them). Your workload as a GROMACS developer goes up for each extra version we support, even if that only shows up in what you have to do to keep Jenkins happy, or how hard it is for you to get access to others' feedback because they are thinking about something that is mysteriously broken in some Jenkins combination. Also, versions that may or may not be the bases for forks that support hardware that GROMACS users might have that isn't already shipping in early 2020 really shouldn't factor highly. The statements below are true in July 2016.

CMake

Discussion at https://redmine.gromacs.org/issues/2505. The latest cmake 3.9.x looks like a good compromise (supports native CUDA on Linux and Windows; has supported C++14 for a long time).

DONE - cmake 3.9.6 adopted

c++

We will move to c++14 for the main code base. If we would do that also for .cu files, then we would either need CUDA 9.0, or to complicate the build system for ensuring that they compile with c++11 in an ABI-compatible way (which means that any std::vector exposed across the internal linking boundary would have to be compiled in the same way in both sides; I don't know if it's technically feasible to link two libstdc++ for the different use cases - it would be risky even if possible). That might or might not be simpler if we already used native CUDA support in cmake.

WIP at https://gerrit.gromacs.org/#/c/8699/

Done - c++14 adopted, but stdlibc++ questions remain

CUDA

Ignoring corner cases, we adopted 7.0 (released March 2015) for GROMACS 2019 (released January 2019). We adopted 6.5 (released August 2014) for GROMACS 2018 (eventually released January 2018). So from a user perspective, the CUDA minimum requirement has been for one that is at least 3 years old.

That suggests CUDA 8.0 (released September 2016) would be a consistent bump for GROMACS 2020 (released January 2020). It adds support for clang 3.7 and clang 3.8 on Linux, but otherwise doesn't look compelling.

CUDA 9.0 would be the minimum required for device-side c++14 compilation. We probably don't want or need to write any c++14 code there, but not having an internal language barrier would be convenient for developers. But the version bump is more aggressive for users, and if we did that then we should consider agreeing that GROMACS 2021 should avoid a version bump.

WIP at gerrit 8963/3

DONE Decided on CUDA 9.0, patch available for review. Undecided on using native cmake cuda support

Python

Python 3 is required and is the only one supported by the core code. releng, physicalvalidation, check-source and documentation code will have to be ported (e.g. via 2to3). Some patches are already in gerrit

gcc

We should adopt -4.9-5, as it supports c++14 and avoids some rough edges in 4.8 (e.g. missing regex support). DONE gcc 5 required

OpenCL

We support 1.2 and that's probably all that's feasible.

icc

Probably worth a bump to get rid of any remaining workarounds for its codegen and/or bugs.

clang

3.4 Edit: DONE required 3.6

replace uncrustify with clang-format

Need to pick a version of clang-format and update the existing config file on a Gerrit patch for any reasonable format that doesn't introduce a massive change while not requiring something ugly, install that on Jenkins, update the releng stuff during the transitional period, make the transitional patches. After a few years, remove uncrustify support in releng and Jenkins.

Subtasks:

Feature # 2126: implement native CUDA support in CMake	**New**
Task # 2505: increase cmake reqirement for GROMACS 2020	**Closed**
Task # 2842: Add libstdc++ check	**Closed**

Related issues:

Related to GROMACS - Task #2012: Bump required version numbers of infrastruct...	**Closed**
Related to GROMACS - Task #2899: Update testing matrix versions for GROMACS 2...	**In Progress**
Related to GROMACS - Feature #2615: Switch to Python3	**Accepted**

Associated revisions

Revision f0947c5a - 01/16/2019 06:27 AM - Mark Abraham

Remove several unnecessary things

More modern cmake requirements mean we don't need to
include some files.

Some TODO comments cannot be addressed in the forecast manner

Simplified some code

Refs #2505, #2831

Change-Id: I3593a072da0ece9b993c7f531ae43fd9fff4afc2

Revision 3f1ea269 - 01/25/2019 06:53 AM - Mark Abraham

Use native cmake C++11 support

Removed custom CMake settings for compiler flags and standard library
settings.

Updated template to use the same build approach as the library
it needs

Refs #2831

Change-Id: Id9461df8695f8a0eadaec8e844e974a32cc7485f

Revision 471518d4 - 01/25/2019 06:53 AM - Mark Abraham

Bump required gcc compiler to 5.1

This prepares for the C++14 switch. Bumped CUDA version to 8.0 (which
we anyway will bump to 9.0 for C++14 support shortly). Bumped
regressiontests version requirement to 5.1 (and the cmake version to
3.9.6, which we forgot recently).

Removed references to minimum versions of gcc and clang that are now
always satisfied by our minimum requirements for those.

Refs #2831

Change-Id: Ifa062b361af08848fd92c3906fd2af04cfa1b8d6

Revision ff34f009 - 01/25/2019 12:17 PM - Mark Abraham

Unify regex implementation

Now that we require compiler versions that all implement std::regex,
we can always have regex support and only use one implementation.

Refs #2831

Change-Id: I0118fc1aac127a5e2fcebd55a6bbef13b7c47762

Revision b0b38c74 - 01/25/2019 06:52 PM - Mark Abraham

Require clang 3.6

This version is the lowest that supports a smooth c++14 debug experience.
The clang in XCode 6.3 is based off this branch, and that is also the
earliest that supported c++14.

Adjusted build matrices because clang 3.6.2 is still affected by AVX
codegen issues in release and debug mode. clang 3.7 is fine. Thus
moved some minor coverage to the post-submit matrix.

Updated install guide text also for some unrelated compiler-support
things.

Also removed an overlooked gcc version check

Refs #2831

Change-Id: Id780ff06147e54c7a4516cc33c9897a4a4d2d3fa

Revision 0dc40abc - 01/25/2019 06:57 PM - Roland Schulz

Require C++14

The current minimum compiler versions tested in Jenkins are already
sufficient. Leaves for a future change whether for CUDA C++11 or C++14
is used.

Addresses C++14 TODO in compat/pointers.h and stophandler.cpp.

Update developer guide

Refs #2831

Change-Id: Ib4329b96327d15c22328715172a9930092ecb64f

Revision d938ae05 - 09/27/2019 09:02 AM - Mark Abraham

Remove leftover support for pre-9.0 CUDA

Refs #2831

Change-Id: I7ec33bb3582006123e745d06da27c9eed12fbfc2

History

#1 - 01/14/2019 10:09 AM - Mark Abraham

- Related to Task #2012: Bump required version numbers of infrastructure for 2018 added

#2 - 01/14/2019 10:09 AM - Mark Abraham

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

#3 - 01/14/2019 10:13 AM - Mark Abraham

A reasonable path would be

1. use c++11 native compilation in cmake
2. bump cmake to 3.9
3. use cuda native compilation in cmake
4. explore whether it is feasible to compile .cpp and .cu files differently

Radical alternative: the CUDA driver API would mean we don't have .cu files, no use of nvcc, and can compile all host-side code the same way. The usage would now look much more like OpenCL, which is bad for developer convenience in writing NVIDIA-focused GPU code, but convenient for keeping OpenCL dev in tight sync.

#4 - 01/14/2019 10:19 AM - Berk Hess

Change set https://gerrit.gromacs.org/#/c/8963/ bump the CUDA minimum to 9.0. But isn't CUDA 9.0 too new for the minimum requirement in releaes-2020? Although it would be nice to be able to use C++14 everywhere, we have always had lower requirements on CUDA device side code. As long as we don't use C++14 features in the CUDA kernels, we don't need CUDA 9.0, or am I missing something?

The radical alternative mentioned by Mark might require 9.0, but even that I'm not sure about.

#5 - 01/14/2019 11:10 AM - Joe Jordan

From a user perspective, I don't think there will be much difference between CUDA 8 or 9. CUDA 9 supports GPUs going back to Keppler and is in the Ubuntu and Debian testing repos and can be installed with macports. I think users with different operating systems will probably know how to install CUDA for themselves, or they would likely be using Ubuntu/Debian or on a mac. So it seems like a big consideration is what will lead to the cleanest code and fewest bugs. On this front, I do not have the expertise to speak, but from the user end I don't think people will see much difference.

#6 - 01/14/2019 11:50 AM - Mark Abraham

Berk Hess wrote:

> Change set https://gerrit.gromacs.org/#/c/8963/ bump the CUDA minimum to 9.0. But isn't CUDA 9.0 too new for the minimum requirement in releaes-2020? Although it would be nice to be able to use C++14 everywhere, we have always had lower requirements on CUDA device side code. As long as we don't use C++14 features in the CUDA kernels, we don't need CUDA 9.0, or am I missing something?

I already discussed this above:

> That suggests CUDA 8.0 (released September 2016) would be a consistent bump for GROMACS 2020 (released January 2020). It adds support for clang 3.7 and clang 3.8 on Linux, but otherwise doesn't look compelling.
>
> CUDA 9.0 would be the minimum required for device-side c++14 compilation. We probably don't want or need to write any c++14 code there, but not having an internal language barrier would be convenient for developers. But the version bump is more aggressive for users, and if we did that then we should consider agreeing that GROMACS 2021 should avoid a version bump.

The question is whether the cost to developers of having an internal boundary is worth the cost to users.

> The radical alternative mentioned by Mark might require 9.0, but even that I'm not sure about.

The CUDA driver API has been available and stable for years AFAIK. nvcc might even translate the CUDA runtime API into such calls (but don't quote me on that). Using it means that we don't care what c++ standards nvcc understands, because nvcc is not in the game.

#7 - 01/14/2019 11:55 AM - Mark Abraham

Our clang cuda build is useful for code checking. It's not supported by cmake-native-cuda. So if we went native cuda, we would probably lose clang-cuda in its current form. But as above, CUDA 8.0 allows nvcc to target clang for host code, so presumably cmake-native-cuda can target clang for host code.

#8 - 01/14/2019 12:24 PM - Berk Hess

To answers Joe's comment. It's not about what the user can install. It's about what versions HPC centers and local cluster managers will have. They are often far behind. So we need to find a good compromise here.

The radical solution is clearly the best, but requires more coding work, at least in the short term. In the long term we could win a lot if we can unify more of our CUDA and OpenCL code paths.

#9 - 01/14/2019 03:56 PM - Mark Abraham

- Assignee set to Mark Abraham

Per meeting today, I'm the lead on this task. Szilard, Paul, Joe will support. Roland's input and patches are of course most welcome.

#10 - 01/14/2019 03:57 PM - Mark Abraham

- Category set to build system

- Target version set to 2020

#11 - 01/14/2019 05:40 PM - Roland Schulz

Mark Abraham wrote:

> C++: I don't know if it's technically feasible to link two libstdc++ for the different use cases - it would be risky even if possible

Why would you want to link two libstdc++? The same libstdc++ supports different standard versions. See: https://stackoverflow.com/questions/46746878/is-it-safe-to-link-c17-c14-and-c11-objects. Should be fine for MSVC too: https://blogs.msdn.microsoft.com/vcblog/2016/06/07/standards-version-switches-in-the-compiler/ (answer by Andrew in comments). Probably fine too for libc++ (comment on stackoverflow).

> CUDA: We probably don't want or need to write any c++14 code there, but not having an internal language barrier would be convenient for developers.

I think this is much more than just inconvenient. We should be moving to making everything constexpr. In particularly for utilities used from everywhere (e.g. not_null). But those exact utilities would be included from many headers which in turn would likely be included (indirectly by CUDA files). Using C++11 for CUDA would severely in which headers we could use constexpr (and auto return type deduction).

> gcc: We should adopt 4.9, as it supports c++14 and avoids some rough edges in 4.8 (e.g. missing regex support).

GCC 4.9 doesn't support a lot of important C++14 features (e.g. constexpr). We should require 5.

#12 - 01/14/2019 06:01 PM - Roland Schulz

Mark Abraham wrote:

> Radical alternative: the CUDA driver API would mean we don't have .cu files, no use of nvcc, and can compile all host-side code the same way. The usage would now look much more like OpenCL, which is bad for developer convenience in writing NVIDIA-focused GPU code, but convenient for keeping OpenCL dev in tight sync.

Currently this would be awesome. But I'm wondering whether this would be the right direction long term. If GROMACS switches from OpenCL to SYCL (or C++2b with built-in offload features) in the future than this would have been wasted effort. See http://lists.llvm.org/pipermail/cfe-dev/2019-January/060811.html for Intel's effort in this space.

Berk Hess wrote:

> It's about what versions HPC centers and local cluster managers will have. They are often far behind.

Most HPC centers should update CUDA if it is required by GROMACS, no? GROMACS is used so frequently on most of the systems, that not supporting it properly would be foolish. I think we should value our time more than some lazy HPC center admin who should have done the update a long time ago anyhow.

#13 - 01/15/2019 05:52 AM - Mark Abraham

Roland Schulz wrote:

> Mark Abraham wrote:
>
> > C++: I don't know if it's technically feasible to link two libstdc++ for the different use cases - it would be risky even if possible
>
> Why would you want to link two libstdc++? The same libstdc++ supports different standard versions. See: https://stackoverflow.com/questions/46746878/is-it-safe-to-link-c17-c14-and-c11-objects. Should be fine for MSVC too: https://blogs.msdn.microsoft.com/vcblog/2016/06/07/standards-version-switches-in-the-compiler/ (answer by Andrew in comments). Probably fine too for libc++ (comment on stackoverflow).

OK good to know. However, the issue I misdescribed is discussed at one of those pages you linked: https://stackoverflow.com/a/49118876/334142

That is, if we have a header (whether from the c++ std library, a third party, or ourselves) that compiles differently under different language standards, then it is on us to use only the compatible subset. Looking at the examples there, that's probably not a big deal.

> > CUDA: We probably don't want or need to write any c++14 code there, but not having an internal language barrier would be convenient for developers.
>
> I think this is much more than just inconvenient. We should be moving to making everything constexpr. In particularly for utilities used from everywhere (e.g. not_null). But those exact utilities would be included from many headers which in turn would likely be included (indirectly by CUDA files). Using C++11 for CUDA would severely in which headers we could use constexpr (and auto return type deduction).

Indeed, the effects of the barrier would be felt transitively.

> > gcc: We should adopt 4.9, as it supports c++14 and avoids some rough edges in 4.8 (e.g. missing regex support).
>
> GCC 4.9 doesn't support a lot of important C++14 features (e.g. constexpr). We should require 5.

Sorry, yes I meant to say 5!

#14 - 01/15/2019 06:04 AM - Mark Abraham

Roland Schulz wrote:

> Mark Abraham wrote:
>
> > Radical alternative: the CUDA driver API would mean we don't have .cu files, no use of nvcc, and can compile all host-side code the same way. The usage would now look much more like OpenCL, which is bad for developer convenience in writing NVIDIA-focused GPU code, but convenient for keeping OpenCL dev in tight sync.
>
> Currently this would be awesome. But I'm wondering whether this would be the right direction long term. If GROMACS switches from OpenCL to SYCL (or C++2b with built-in offload features) in the future than this would have been wasted effort. See http://lists.llvm.org/pipermail/cfe-dev/2019-January/060811.html for Intel's effort in this space.

Indeed, the momentum behind SYCL is pretty encouraging. But we'd have to see that there was at least as much enthusiasm for SYCL in the HPC space as OpenCL, and our limited resources means we'd have to replace the OpenCL port with SYCL and probably keep the CUDA port.

> Berk Hess wrote:
>
> > It's about what versions HPC centers and local cluster managers will have. They are often far behind.
>
> Most HPC centers should update CUDA if it is required by GROMACS, no? GROMACS is used so frequently on most of the systems, that not supporting it properly would be foolish. I think we should value our time more than some lazy HPC center admin who should have done the update a long time ago anyhow.

I suspect that our historical experience is not very informative here... e.g. TITAN couldn't have CUDA 6.5 for a while, but that was a technical problem and got fixed. I agree that in general HPC centers will have multiple reasons to want to get the latest drivers and CUDA toolkits, and that in general we should expect that a version more then two years old is installed or able to be installed when a GROMACS user wants to upgrade. And often a GROMACS user won't want to install it the moment we release it, so in practice the window is often larger. There's surely HPC software out there that only supports a particular CUDA version - we're hardly being bleeding edge with requiring stuff that's two years old.

There's definitely gains to be had from saving our time. That does cost some time for admins and users updating CUDA versions, but those versions are also in demand from other pieces of software in a general-purpose cluster. A cluster dedicated for GROMACS will already have had reasons to use the latest hardware, drivers and CUDA library.

So I think there's enough benefit for us in the simplicity of just using c++14 everywhere that we should bump to CUDA 9.0

#15 - 01/15/2019 03:27 PM - Mark Abraham

- Description updated

Added python to infrastructure to be bumped

#16 - 01/16/2019 06:30 AM - Gerrit Code Review Bot

Gerrit received a related patchset '1' for Issue #2831.
Uploader: Mark Abraham (mark.j.abraham@gmail.com)
Change-Id: gromacs~master~I3593a072da0ece9b993c7f531ae43fd9fff4afc2
Gerrit URL: https://gerrit.gromacs.org/8987

#17 - 01/16/2019 07:21 AM - Gerrit Code Review Bot

Gerrit received a related patchset '1' for Issue #2831.
Uploader: Mark Abraham (mark.j.abraham@gmail.com)
Change-Id: gromacs~master~Id9461df8695f8a0eadaec8e844e974a32cc7485f
Gerrit URL: https://gerrit.gromacs.org/8988

#18 - 01/17/2019 06:32 AM - Gerrit Code Review Bot

Gerrit received a related patchset '1' for Issue #2831.
Uploader: Mark Abraham (mark.j.abraham@gmail.com)
Change-Id: gromacs~master~I3179b1a62288226322fae83430afec1312be1fd4
Gerrit URL: https://gerrit.gromacs.org/8997

#19 - 01/18/2019 07:31 AM - Roland Schulz

Is there still ongoing discussion about the CUDA 9.0 requirement or can we proceed with enabling C++14 everywhere?

#20 - 01/18/2019 09:58 AM - Mark Abraham

Roland Schulz wrote:

> Is there still ongoing discussion about the CUDA 9.0 requirement or can we proceed with enabling C++14 everywhere?

I was planning to get a consensus decision at Monday's meeting.

#21 - 01/18/2019 10:49 AM - Berk Hess

I don't have strong opinions on the CUDA 9.0 issue. But we should gets Szilard's opinion, I added him as a watcher.

#22 - 01/21/2019 06:29 PM - Szilárd Páll

Berk Hess wrote:

> I don't have strong opinions on the CUDA 9.0 issue. But we should gets Szilard's opinion, I added him as a watcher.

The only concern is that the CUDA requirement is the most difficult to fulfill mdrun dependency to given that it generally requires device driver upgrade (and sometimes kernel too) which can be a concern on clusters.

I've no passion for old versions of proprietary toolkits with tricky compatibility concerns, so I'm fine with the suggestion as long as we are fine with the risk of an impact on adoption of the next release.

Also note that 9.1 came only a few months after 9.0 -- not a significant difference in time. If we run into any other issues with 9.0, we may as well bump to that.

#23 - 01/21/2019 06:30 PM - Szilárd Páll

Szilárd Páll wrote:

> I've no passion for old versions of proprietary toolkits with tricky compatibility concerns, so I'm fine with the suggestion as long as we are fine with the risk of an impact on adoption of the next release.

we can however recommend OpenCL on NVIDIA for those affected -- that may have additional positive side-effects too.

#24 - 01/22/2019 09:08 AM - Mark Abraham

Szilárd Páll wrote:

> Berk Hess wrote:
>
> > I don't have strong opinions on the CUDA 9.0 issue. But we should gets Szilard's opinion, I added him as a watcher.
>
> The only concern is that the CUDA requirement is the most difficult to fulfill mdrun dependency to given that it generally requires device driver upgrade (and sometimes kernel too) which can be a concern on clusters.

Yep. FWIW I asked the question on Twitter, only likes and a clarifying question as feedback so far.

> I've no passion for old versions of proprietary toolkits with tricky compatibility concerns, so I'm fine with the suggestion as long as we are fine with the risk of an impact on adoption of the next release.

We can't measure that, so we can only take our best guess. People can use older GROMACS, people don't always run on old clusters that can't/won't upgrade, so only a minority of users have a significant issue.

> Also note that 9.1 came only a few months after 9.0 -- not a significant difference in time. If we run into any other issues with 9.0, we may as well bump to that.

Sure.

#25 - 01/24/2019 04:15 PM - Gerrit Code Review Bot

Gerrit received a related patchset '1' for Issue #2831.
Uploader: Mark Abraham (mark.j.abraham@gmail.com)
Change-Id: gromacs~master~I0118fc1aac127a5e2fcebd55a6bbef13b7c47762
Gerrit URL: https://gerrit.gromacs.org/9032

#26 - 01/24/2019 04:20 PM - Gerrit Code Review Bot

Gerrit received a related patchset '9' for Issue #2831.
Uploader: Mark Abraham (mark.j.abraham@gmail.com)
Change-Id: gromacs~master~Ib4329b96327d15c22328715172a9930092ecb64f
Gerrit URL: https://gerrit.gromacs.org/8699

#27 - 01/24/2019 06:46 PM - Gerrit Code Review Bot

Gerrit received a related patchset '3' for Issue #2831.
Uploader: Mark Abraham (mark.j.abraham@gmail.com)
Change-Id: gromacs~master~Ifa062b361af08848fd92c3906fd2af04cfa1b8d6
Gerrit URL: https://gerrit.gromacs.org/9031

#28 - 01/25/2019 01:42 PM - Gerrit Code Review Bot

Gerrit received a related patchset '1' for Issue #2831.
Uploader: Mark Abraham (mark.j.abraham@gmail.com)
Change-Id: gromacs~master~Id780ff06147e54c7a4516cc33c9897a4a4d2d3fa
Gerrit URL: https://gerrit.gromacs.org/9040

#29 - 01/28/2019 11:54 AM - Mark Abraham

- Description updated

#30 - 04/01/2019 04:20 PM - Eric Irrgang

- Related to Task #2899: Update testing matrix versions for GROMACS 2020 release added

#31 - 04/01/2019 04:30 PM - Eric Irrgang

Is there an issue or document for cross-referencing the Linux distribution releases which are expected to be supported out-of-the-box versus with extra work? Specifically, I am trying to figure out if there is any current or future consensus that would affect which minor release of Python 3 to plan for? For instance, 'trusty' and 'jessie' imply Python 3.4, but a quick glance here makes me think that these distributions are too old to influence GROMACS 2020 plans.

Reference https://redmine.gromacs.org/issues/2615 and https://gmxapi.readthedocs.io/en/latest/layers/python.html

#32 - 04/01/2019 04:31 PM - Eric Irrgang

- Related to Feature #2615: Switch to Python3 added

#33 - 07/23/2019 09:56 AM - Mark Abraham

- Related to Task #3041: Remove workaround for gcc bug 58265 added

#34 - 08/23/2019 03:24 PM - Mark Abraham

- Target version changed from 2020 to 2020-infrastructure-update-post-beta1

#35 - 09/30/2019 08:00 PM - Eric Irrgang

- Related to deleted (Task #3041: Remove workaround for gcc bug 58265)