GROMACS - Bug #1919

static linking issues cause by hwloc support

03/13/2016 08:36 PM - Szilárd Páll

Status:	Accepted		
Priority:	Normal		
Assignee:			
Category:	build system		
Target version:	future		
Affected version - extra info:		**Difficulty:**	uncategorized
Affected version:	git master		

Description

The hwloc support is causing a number of minor issues:
~~* CMake output even when GMX_HWLOC=OFF~~
* static linking breaks

```
/usr/bin/g++-4.8   -march=core-avx2    -std=c++0x -Wundef -Wextra -Wno-missing-field-initializers
-Wpointer-arith -Wall -Wno-unused-function  -O3 -DNDEBUG -funroll-all-loops -fexcess-precision=fas
t  -Wno-array-bounds   src/programs/CMakeFiles/gmx.dir/gmx.cpp.o src/programs/CMakeFiles/gmx.dir/l
egacymodules.cpp.o src/programs/CMakeFiles/mdrun_objlib.dir/mdrun/md.cpp.o src/programs/CMakeFiles
/mdrun_objlib.dir/mdrun/membed.cpp.o src/programs/CMakeFiles/mdrun_objlib.dir/mdrun/runner.cpp.o s
rc/programs/CMakeFiles/mdrun_objlib.dir/mdrun/mdrun.cpp.o src/programs/CMakeFiles/mdrun_objlib.dir
/mdrun/repl_ex.cpp.o src/programs/CMakeFiles/mdrun_objlib.dir/mdrun/resource-division.cpp.o src/pr
ograms/CMakeFiles/view_objlib.dir/view/view.cpp.o  -o bin/gmx  -rdynamic lib/libgromacs.a -fopenmp
 /opt/tcbsys/cuda/7.5/lib64/libcudart.so -Wl,-Bstatic -lhwloc -Wl,-Bdynamic -lrt -lm -Wl,-Bstatic
-lz -Wl,-Bdynamic /opt/tcbsys/fftw/3.3.4-sse2-avx/lib/libfftw3f.a -lpthread -Wl,-rpath,/opt/tcbsys
/cuda/7.5/lib64:::::::::::::::::::::::::::::::: && :
/usr/lib/gcc/x86_64-linux-gnu/4.8/../../../x86_64-linux-gnu/libhwloc.a(components.o): In function
`hwloc__dlforeach_cb':
(.text+0x77): undefined reference to `lt_dlopenext'
/usr/lib/gcc/x86_64-linux-gnu/4.8/../../../x86_64-linux-gnu/libhwloc.a(components.o): In function
`hwloc__dlforeach_cb':
(.text+0x91): undefined reference to `lt_dlerror'
/usr/lib/gcc/x86_64-linux-gnu/4.8/../../../x86_64-linux-gnu/libhwloc.a(components.o): In function
`hwloc__dlforeach_cb':
(.text+0x107): undefined reference to `lt_dlsym'
/usr/lib/gcc/x86_64-linux-gnu/4.8/../../../x86_64-linux-gnu/libhwloc.a(components.o): In function
`hwloc__dlforeach_cb':
(.text+0x3d5): undefined reference to `lt_dlclose'
/usr/lib/gcc/x86_64-linux-gnu/4.8/../../../x86_64-linux-gnu/libhwloc.a(components.o): In function
`hwloc_plugins_exit':
(.text+0x446): undefined reference to `lt_dlclose'
/usr/lib/gcc/x86_64-linux-gnu/4.8/../../../x86_64-linux-gnu/libhwloc.a(components.o): In function
`hwloc_plugins_exit':
(.text+0x490): undefined reference to `lt_dlexit'
/usr/lib/gcc/x86_64-linux-gnu/4.8/../../../x86_64-linux-gnu/libhwloc.a(components.o): In function
`hwloc_plugins_init':
(.text+0x4da): undefined reference to `lt_dlinit'
/usr/lib/gcc/x86_64-linux-gnu/4.8/../../../x86_64-linux-gnu/libhwloc.a(components.o): In function
`hwloc_plugins_init':
(.text+0x556): undefined reference to `lt_dlforeachfile'
/usr/lib/gcc/x86_64-linux-gnu/4.8/../../../x86_64-linux-gnu/libhwloc.a(topology-linux.o): In funct
ion `hwloc_linux_set_area_membind':
(.text+0x1ce2): undefined reference to `mbind'
/usr/lib/gcc/x86_64-linux-gnu/4.8/../../../x86_64-linux-gnu/libhwloc.a(topology-linux.o): In funct
ion `hwloc_linux_set_area_membind':
(.text+0x1d52): undefined reference to `mbind'
/usr/lib/gcc/x86_64-linux-gnu/4.8/../../../x86_64-linux-gnu/libhwloc.a(topology-linux.o): In funct
ion `hwloc_linux_set_thisthread_membind':
```

```
(.text+0x1e72): undefined reference to `set_mempolicy'
/usr/lib/gcc/x86_64-linux-gnu/4.8/../../../x86_64-linux-gnu/libhwloc.a(topology-linux.o): In funct
ion `hwloc_linux_set_thisthread_membind':
(.text+0x1f08): undefined reference to `migrate_pages'
/usr/lib/gcc/x86_64-linux-gnu/4.8/../../../x86_64-linux-gnu/libhwloc.a(topology-linux.o): In funct
ion `hwloc_linux_set_thisthread_membind':
(.text+0x1f4b): undefined reference to `set_mempolicy'
/usr/lib/gcc/x86_64-linux-gnu/4.8/../../../x86_64-linux-gnu/libhwloc.a(topology-linux.o): In funct
ion `hwloc_linux_find_kernel_max_numnodes':
(.text+0x1fef): undefined reference to `get_mempolicy'
/usr/lib/gcc/x86_64-linux-gnu/4.8/../../../x86_64-linux-gnu/libhwloc.a(topology-linux.o): In funct
ion `hwloc_linux_get_thisthread_membind':
(.text+0x210b): undefined reference to `get_mempolicy'
/usr/lib/gcc/x86_64-linux-gnu/4.8/../../../x86_64-linux-gnu/libhwloc.a(topology-linux.o): In funct
ion `hwloc_linux_get_area_membind':
(.text+0x2282): undefined reference to `get_mempolicy'
collect2: error: ld returned 1 exit status
```

~~The former should be an easy fix~~; the latter likely requires figuring out which libraries need to be added to the link line, here at least libnuma and libltdtl is needed.

Associated revisions

Revision ccbd250a - 12/09/2016 01:43 AM - Szilárd Páll

Turn off hwloc support when static lib found

Hwloc dependencies are not resolved at CMake time when static
libwloc.a is detected and in most of these cases link-time
errors will prevent building GROMACS. As it is hard for a user to know
how to solve such cryptic errors and hwloc is not a required dependency,
we turn off hwloc support when a static lib is detected. The user can
override this on the cmake command line.

Refs #1919

Change-Id: Ic3917e2e59ee4c291b78ee47150e513ae77ced85

History

#1 - 03/13/2016 08:38 PM - Szilárd Páll

Edit: should have pulled before filing the issue, the verbosity has apparently been fixed. Will update the description.

#2 - 03/13/2016 08:39 PM - Szilárd Páll

- Subject changed from minor issues with the new hwloc support to static linking issues cause by hwloc support

- Description updated

#3 - 03/18/2016 06:20 PM - Mark Abraham

It would be nice to understand how to fix this kind of thing properly. But I never see these kinds of problems, else they'd be fixed already. :-)

So:

- On what machine?
- Calling cmake how?
- Trying to do what kind of static linking?
- To get what done?
- Does -DCMAKE_BUILD_SHARED_EXE=off do the job it is intended to do?

#4 - 03/18/2016 08:30 PM - Szilárd Páll

Mark Abraham wrote:

> It would be nice to understand how to fix this kind of thing properly. But I never see these kinds of problems, else they'd be fixed already. :-)

I guess you don't see it because you don't configure with static libs. One try would've been enough to repro the above issue.

I most of the time use a command line similar to this (the options irrelevant to the topic I moved to the second line):

```
CC=gcc-XX CXX=g++-XX cmake ../ -DBUILD_SHARED_LIBS=OFF -DGMX_PREFER_STATIC_LIBS=ON \
-DGMX_EXTERNAL_BLAS=OFF -DGMX_EXTERNAL_LAPACK=OFF -DCMAKE_PREFIX_PATH="${FFTW_HOME}" -DCUDA_TOOLKIT_ROOT_DIR=$
{CUDA_HOME} -DGMX_CYCLE_SUBCOUNTERS=ON
```

> So:
>
> - On what machine?

In my experience any machine where hwloc.a is present, I ran into this issue on my desktop, tcbsXX, beskow, etc.

> - Calling cmake how?

See above.

> - Trying to do what kind of static linking?

See above.

> - To get what done?

Static linking as the title states. As static as possible, to be more precise.

BUILD_SHARED_LIBS=OFF so that I can keep multiple versions of the gmx or mdrun binary + GMX_PREFER_STATIC_LIBS=ON so to eliminate external dependencies whenever possible.

> - Does -DCMAKE_BUILD_SHARED_EXE=off do the job it is intended to do?

I don't understand the question. This report is not about CMAKE_BUILD_SHARED_EXE behavior, and in any as far as I can tell, it doesn't do anything other than what my command line invocation achieves with BUILD_SHARED_LIBS=OFF + GMX_PREFER_STATIC_LIBS=ON.

#5 - 03/18/2016 08:42 PM - Szilárd Páll

Szilárd Páll wrote:

> > - Does -DCMAKE_BUILD_SHARED_EXE=off do the job it is intended to do?
>
> I don't understand the question. This report is not about CMAKE_BUILD_SHARED_EXE behavior, and in any as far as I can tell, it doesn't do anything other than what my command line invocation achieves with BUILD_SHARED_LIBS=OFF + GMX_PREFER_STATIC_LIBS=ON.

Actually, it's GMX_BUILD_SHARED_EXE=off what I'm talking about, is that what you meant?

#6 - 04/07/2016 04:19 PM - Mark Abraham

Szilárd Páll wrote:

> Szilárd Páll wrote:
>
> > - Does -DCMAKE_BUILD_SHARED_EXE=off do the job it is intended to do?
> >
> > I don't understand the question. This report is not about CMAKE_BUILD_SHARED_EXE behavior, and in any as far as I can tell, it doesn't do anything other than what my command line invocation achieves with BUILD_SHARED_LIBS=OFF + GMX_PREFER_STATIC_LIBS=ON.
>
> Actually, it's GMX_BUILD_SHARED_EXE=off what I'm talking about, is that what you meant?

Yes, that's what I meant, sorry.

#7 - 04/07/2016 04:21 PM - Mark Abraham

Probably unrelated, but e.g. building on bs_nix-amd_gpu I get configure time warnings:

```
CMake Warning at src/testutils/TestMacros.cmake:47 (add_executable):
  Cannot generate a safe runtime search path for target selection-test
  because files in some directories may conflict with libraries in implicit
  directories:
```

```
    runtime library [libhwloc.so.5] in /usr/lib/x86_64-linux-gnu may be hidden by files in:
      /usr/local/lib

  Some of these libraries may not be found correctly.
```

There's probably a way to be more specific about which hwloc to find, but this is an issue we need to address if we want to make general use of hwloc in future.

#8 - 07/09/2016 03:36 AM - Szilárd Páll

Given that all builds with hwloc found as static lib typically fail, I think is is a blocker for a final 2016 release (if not for RC), but as we discussed offline with Mark a temporary solution could be to preventively disable hwloc in those cases (and pkg-config would be also a partial solution to generate working link command with hwloc's dependencies included).

#9 - 07/10/2016 04:31 PM - Mark Abraham

- Status changed from New to Accepted

This is the same kind of transitive-linking problem as we had with e.g. libxml2 sometimes needing zlib. Even if one builds a static libhwloc, by default that links to libnuma, which requires hwloc built like https://github.com/open-mpi/hwloc/wiki/StaticBuild. Worse, even when you have a libnuma.a and use their recipe, libnuma.a calls getaddrinfo(), which requires glibc to be available statically at link time (or dynamically at run time). That sounds pretty familiar, from attempting static builds on Crays.

Our detection is currently happy because if hwloc needs libnuma when the system has any shared or static libnuma, but then the build fails (as Szilard reported, confirmed by me) because libhwloc.a needs libnuma.a and our FindHwloc.cmake doesn't try to grapple with that aspect.

pkg-config --libs --static hwloc is a possible route to happiness, but it merely reports some subset of -Lwhatever -lhwloc -lm -lnuma -lpthread -lcudart -lxml2 -lz -lX11 -lcairo, so most of the problem remains. http://stackoverflow.com/questions/27586503/how-to-use-pkg-config-to-link-a-library-statically suggests that hwloc might be able to improve on that, which I'll explore. We could parse that anyway, and try to find the necessary stuff (which might mean a bit of chasing of hwloc configure-time options across versions). Or we could make some educated guesses and try to find_package(hwloc) a few times. Or find hwloc, then subject it to a series of link-time tests.

CMake's find_package() doesn't have a way to force that only static libraries are found. That's quite sound - the most reliable way to restrict what artifacts can be built is to configure the toolchain so that it can only link statically, which is what the hwloc link above is doing.

However, even if using such a toolchain when compiling GROMACS and thus only being able to find libhwloc.a, if a similar toolchain (or suitably dependency-free configuration) wasn't used for compiling libhwloc.a, then find_package will simply fail because libhwloc.a still has unsatisfied link-time requirements.

Vendors/HPC sites that encourage/require static builds for back ends should be providing appropriate and useful fully-static libhwloc.a and its dependencies (without e.g. X11 support configured), of course. But they won't in a lot of cases. If we want to depend on hwloc in a future version of GROMACS, then we need a reliable solution. We also need one for Jenkins, because setting up our MemorySanitizer build (if not others) will probably need some active work.

In the future, we could either bundle hwloc, or do a download-and-build like we have for FFTW3. It's 3.8MB for hwloc-1.11.3.tbz, which expands to 17M. However, 4.5M is tests, 1M is utils, and 7M is doc. We could perhaps strip down to about 1M configure, 1M src and 0.5M include and bundle that. Either way, we'd also provide a GMX_EXTERNAL_HWLOC=on to suit distro maintainers. And we'll probably get some useful things, so we might be able to drop a bunch of our own code for GPU detection and/or fallback CPU detection. We can also keep the version of hwloc fairly current, too, which is probably useful with an eye to a hairy future with hardware heterogeneity. Bundling offers the advantage that GROMACS developers won't have to build their own static hwloc every time the work somewhere weird. But if we bundle, then we will probably want to bump our CMake version up, to pick up better implementations of CMake's external_project gear (and probably deploy that for the other stuff in src/extern also).

Given that hwloc is an optional dependency of GROMACS, I don't think that awkwardness about a static build is a blocker for 2016. I attempted hwloc's fully-static build, but it doesn't bundle libnuma.a, so the link-time problem remains. I don't know whether we need the functionality from libnuma.a now (or will need it later), but I suspect we do, and we need to know that before we can decide on a recommended work-around.

#10 - 07/12/2016 06:39 PM - Mark Abraham

Further, if libnuma is important for functionality that we want to use in GROMACS, in all cases we have to react to the presence of libnuma.a on the system. That's true whether we stick with using a system hwloc, or (in future) configure our own hwloc build (whether downloaded or bundled), because such things are configure-time options of hwloc and we're still going to need to tell CMake how to link gmx to libgromacs.a AND to libhwloc.a AND any .a dependencies. Same goes for libcudart.a (which a system hwloc might well have configured, though whether we'd use it for a build we manage is also unclear at this stage). So we might be looking at needing FindNUMA.cmake, at least.

In particular, it'd be good to get some insight from Cray about their plans for catering for such things.

#11 - 07/14/2016 07:51 PM - Szilárd Páll

Indeed, it is similar to the libxml issue, with the difference that hwloc detection is on by default rather than in optionally when test suite is requested to be built. Therefore, it won't only fail if one enables something explicitly (e.g. tests), but it will result in builds throwing unfriendly link time errors whenever libhwloc.a is picked up. This will happen both when the user wanted dependencies linked statically or if the .so was simply not found (e.g. hwloc built static-only). How common this use-case is hard to judge, but I don't think it's an HPC vendor toolchain specific issue.

I suggested preemptively switching off hwloc and (or?) issuing a note to the user exactly because the feature is optional and the link failure will offer

little hint to what the workaround is.

It is not clear why do you think it is in GROMACS' interest to use libnuma directly? If you meant using it through hwloc's interface, the answer is likely yes, I don't think we can rely solely on first-touch allocation
policies, on platforms like KNL in some more the near memory can be configured such that it's presented as a separate NUMA region.

#12 - 07/19/2016 07:50 PM - Mark Abraham

Szilárd Páll wrote:

> Indeed, it is similar to the libxml issue, with the difference that hwloc detection is on by default rather than in optionally when test suite is requested to be built. Therefore, it won't only fail if one enables something explicitly (e.g. tests), but it will result in builds throwing unfriendly link time errors whenever libhwloc.a is picked up. This will happen both when the user wanted dependencies linked statically or if the .so was simply not found (e.g. hwloc built static-only). How common this use-case is hard to judge, but I don't think it's an HPC vendor toolchain specific issue.

Yeah, but what action do you suggest? Anywhere with only libhwloc.a needs to document how to use it, and one can append to the link flags readily enough.

> I suggested preemptively switching off hwloc and (or?) issuing a note to the user exactly because the feature is optional and the link failure will offer little hint to what the workaround is.

I no longer recall what "preemptively switching off hwloc" means. Default to not use hwloc for GMX_BUILD_SHARED_EXE=off?

> It is not clear why do you think it is in GROMACS' interest to use libnuma directly?

I didn't say that. "Even if one builds a static libhwloc, by default that links to libnuma, ... I don't know whether we need the functionality from libnuma.a now (or will need it later), but I suspect we do, and we need to know that before we can decide on a recommended work-around."

> If you meant using it through hwloc's interface, the answer is likely yes, I don't think we can rely solely on first-touch allocation
> policies, on platforms like KNL in some more the near memory can be configured such that it's presented as a separate NUMA region.

So we seem to agree that we probably want the functionality in hwloc that libnuma supports (in the long term, at least).

For 2016, we can recommend a build-time link-flags-based workaround for the few users who will have hwloc and only have a static library found, or who try to require a static build. And we can note that the hwloc dependency is optional, so people can turn it off. But I've investigated and found that libhwloc.a can have many dependencies, and pkgconfig isn't likely to help unless we write a whole bunch of find_package() modules, so I am not going to try to write cmake-ry to handle that.

Making the hwloc dependency optional and off by default doesn't meet our needs - if we did, then we'll still have the eleventh-hour argument next year that we don't know whether we can rely on it.

For the future, we need something more robust (and I have discussed the complex issues above).

#13 - 07/28/2016 05:03 PM - Szilárd Páll

Mark Abraham wrote:

> Szilárd Páll wrote:
>
> > Indeed, it is similar to the libxml issue, with the difference that hwloc detection is on by default rather than in optionally when test suite is requested to be built. Therefore, it won't only fail if one enables something explicitly (e.g. tests), but it will result in builds throwing unfriendly link time errors whenever libhwloc.a is picked up. This will happen both when the user wanted dependencies linked statically or if the .so was simply not found (e.g. hwloc built static-only). How common this use-case is hard to judge, but I don't think it's an HPC vendor toolchain specific issue.
>
> Yeah, but what action do you suggest? Anywhere with only libhwloc.a needs to document how to use it, and one can append to the link flags readily enough.

Turning off hwloc if GMX_PREFER_STATIC_LIBS=ON or even better, when "HWLOC_LIBRARIES=*/libwloc.a".

Who will append the link flags, users? How will they relate the presence of link-time errors with the need for this workaround? Sure, they could just read the docs, but that's not what people do when a linker error shows up, I guess.

> > I suggested preemptively switching off hwloc and (or?) issuing a note to the user exactly because the feature is optional and the link failure will offer little hint to what the workaround is.
>
> I no longer recall what "preemptively switching off hwloc" means. Default to not use hwloc for GMX_BUILD_SHARED_EXE=off?

See above.

> If you meant using it through hwloc's interface, the answer is likely yes, I don't think we can rely solely on first-touch allocation policies, on platforms like KNL in some more the near memory can be configured such that it's presented as a separate NUMA region.

> So we seem to agree that we probably want the functionality in hwloc that libnuma supports (in the long term, at least).

But the libnuma dependency does not seem to be optional, so I'm not sure how does it help knowing whether libnuma is useful or not?

> For 2016, we can recommend a build-time link-flags-based workaround for the few users who will have hwloc and only have a static library found, or who try to require a static build. And we can note that the hwloc dependency is optional, so people can turn it off.

That's of course useful, but it may not help in many cases as the suggestions will be context insensitive.

> Making the hwloc dependency optional and off by default doesn't meet our needs - if we did, then we'll still have the eleventh-hour argument next year that we don't know whether we can rely on it.

Regardless of when you require it, it needs testing. If you think a stable a release is the right place to test and explore issues, I won't stand in its way.

#14 - 09/07/2016 03:23 PM - Mark Abraham

- Target version changed from 2016 to 2016.1

We should consider Szilard's suggestion for "Turning off hwloc if GMX_PREFER_STATIC_LIBS=ON" and documenting when and how to hack on linker flags if people want hwloc support (but nobody really needs it at the moment).

Addressing bundling hwloc per my comment 9 is now more palatable with the recently merged cmake requirements bump.

Szilard, I'd love not to "do testing in stable releases," but part of the solution has to be people being proactive about reviewing and testing in the ~1 year that the hwloc-support patches were in gerrit.

#15 - 09/08/2016 02:18 AM - Szilárd Páll

Mark Abraham wrote:

> Szilard, I'd love not to "do testing in stable releases," but part of the solution has to be people being proactive about reviewing and testing in the ~1 year that the hwloc-support patches were in gerrit.

I understand your point, but I'm not sure I agree with it. To be consistent we could apply the following logic often used: if it was not important enough for the authors (or the community) to bring up the matter, get the change reviewed, and the test feature thoroughly, it could not have been that important (and could have waited another year).

#16 - 10/31/2016 11:50 AM - Mark Abraham

- Target version changed from 2016.1 to 2018

Szilárd Páll wrote:

> Mark Abraham wrote:
>
> > Szilard, I'd love not to "do testing in stable releases," but part of the solution has to be people being proactive about reviewing and testing in the ~1 year that the hwloc-support patches were in gerrit.
>
> I understand your point, but I'm not sure I agree with it. To be consistent we could apply the following logic often used: if it was not important enough for the authors (or the community) to bring up the matter, get the change reviewed, and the test feature thoroughly, it could not have been that important (and could have waited another year).

Erik did the work, motivated by the difficulties faced by those of us handling performance issues on heterogeneous hardware, and got it reviewed by multiple people, who all thought it was important and worth having. If you didn't have time in a year to test / comment / review / contribute, then clearly it wasn't important for you. But please respect the work others did do, even though you feel it was incomplete in the above aspect.

Meanwhile, having had it merged has found an issue that we can work on solving, which if it was still in gerrit waiting for perfect testing we might not have yet discovered. I see a few aspects in progressing towards a stable solution:

1. explore the use of CMake toolchains that can only do static linking, so that we can't find libraries that have dependencies that will need resolution later, but which got satisfied with dynamic linking when we tested at CMake time

2. bundle and build a fallback hwloc, because the set of possible dependencies of hwloc is large, and the CMake code (and user documentation) to find them out and see if they can be satisfied is probably comparable with that for building hwloc and re-using its configuration machinery (which lets us defer a lot of things to their developers and documentation)
3. if an all-static CUDA(MPI+OpenMP+libstdc+) build is desirable (e.g. for big iron, even though dynamic libraries are routinely available in recent CUDA releases), then someone needs to come up with recipe(s) for one of our GPU slaves to get it done, so we can have it in CI testing (at some level) so it would have been something in the mind of people who did the design and review

#17 - 11/03/2016 12:58 AM - Gerrit Code Review Bot

Gerrit received a related patchset '1' for Issue #1919.
Uploader: Szilárd Páll (pall.szilard@gmail.com)
Change-Id: Ic3917e2e59ee4c291b78ee47150e513ae77ced85
Gerrit URL: https://gerrit.gromacs.org/6318

#18 - 11/04/2016 09:30 PM - Szilárd Páll

It's besides the point whether I had time to test it. So is whether the bug is "important" or urgent for me personally (from the pov of the GROMACS project and its users). Nor does gerrit-aging a change alone warrant default-on live-testing a non-essential feature, I simply thought, but I'm not the one to decide.

But more to the point, the bug was accepted and nominated for 2016.1, so I'm not sure if you meant to bump to 2017 and not 2016.2? To the simple workaround I suggested your reply sounded like "the suggestion may be acceptable", so I was expecting feedback. In any case, I've uploaded the suggested mid-term fix; long-term as we want to rely more on hwloc, we should consider a more robust solution, but given the number of hwloc dependencies, I'm not sure what is the best approach.

1. explore the use of CMake toolchains that can only do static linking, so that we can't find libraries that have dependencies that will need resolution later, but which got satisfied with dynamic linking when we tested at CMake time

If CMake toolchains exist that can deal with dependency resolution/early testing to ensure that the build will not fail at link-time, that would be great. Not sure if the issue you mention is a hypothetical one or one that exists in the code? If we'd try to link against libhwloc.a at CMake-time, we'd immediately notice a link-time error, the linked won't link against the .so instead.

1. bundle and build a fallback hwloc, because the set of possible dependencies of hwloc is large, and the CMake code (and user documentation) to find them out and see if they can be satisfied is probably comparable with that for building hwloc and re-using its configuration machinery (which lets us defer a lot of things to their developers and documentation)

I can't comment how straightforward is it to strip hwloc down and bundle a simplified version, but unless it can be done with only/mostly configure-tweaks, it may not be worth the effort.

1. if an all-static CUDA(MPI+OpenMP+libstdc+) build is desirable (e.g. for big iron, even though dynamic libraries are routinely available in recent CUDA releases), then someone needs to come up with recipe(s) for one of our GPU slaves to get it done, so we can have it in CI testing (at some level) so it would have been something in the mind of people who did the design and review

All-static builds are IMO not a very important target on all but HPC machines; in those cases the native toolchains will hopefully help with avoiding dependency issues. BTW CUDA has always been shipped with dynamic libs, it's the static version of the the runtime (libcudart) that's only been available for a few releases. Testing of static linking against *some* libs is likely relevant as all a user has to do is ./configure --enable-static --prefix=... when building fftw or hwloc to have cmake detect the static archive and not the dynamic lib.

#19 - 06/27/2017 11:30 PM - Mark Abraham

- Target version changed from 2018 to 2019

Nobody's decided on a suitable approach yet.

#20 - 10/12/2018 10:49 PM - Mark Abraham

- Target version changed from 2019 to future

UNtil we have a reliable all static build, I don't see this making any progress