GROMACS - Bug #1468

boost/assert.hpp:102:47: error: ‘noinline’ was not declared in this scope

03/23/2014 08:56 PM - Christoph Junghans

<table>
<thead>
<tr>
<th>Status:</th>
<th>Closed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Priority:</td>
<td>Normal</td>
</tr>
<tr>
<td>Assignee:</td>
<td>Teemu Murtola</td>
</tr>
<tr>
<td>Category:</td>
<td>build system</td>
</tr>
<tr>
<td>Target version:</td>
<td>5.0</td>
</tr>
<tr>
<td>Affected version - extra info:</td>
<td>5.0-beta1</td>
</tr>
</tbody>
</table>

Description
/usr/include/boost/assert.hpp:102:47: error: ‘noinline’ was not declared in this scope

BOOST_NOINLINE void assertion_failed_msg(CharT const * expr, char const * msg, char const * function,

Details:
<https://bugs.gentoo.org/show_bug.cgi?id=505480>

Related issues:
Related to GROMACS - Bug #1402: Error compiling 5.0-beta1 with CUDA and gcc >...

History
#1 - 03/23/2014 11:10 PM - Mark Abraham
- Status changed from New to Blocked, need info

There's no GROMACS file I can see that even #includes assert.hpp, and the failing command is pretty opaque. Thus it is probably originating in something generated by CUDA. The build is picking up a system Boost 1.55. I've no idea what version of Boost may or may not be compatible with what CUDA, but setting cmake -DGMX_EXTERNAL_BOOST=off should work around it. Whether that is good for any upstream library linking to GROMACS is another problem.

So far, I don't see a GROMACS problem to fix.

#2 - 03/23/2014 11:14 PM - Roland Schulz
Did you check whether this is still an issue with latest master. It might be fixed because the propagation of compiler flags to nvcc has been improved and this in turn might fix it. If not we could fix it by making sure we don't indirectly include boost from cuda code. Because we shouldn't need to.

#3 - 03/23/2014 11:24 PM - Mark Abraham
gmx_fatal.h is being #included, but I can see no reason for that to end up #including assert.hpp.

#4 - 03/24/2014 05:31 AM - Teemu Murtola
Roland Schulz wrote:

If not we could fix it by making sure we don't indirectly include boost from cuda code. Because we shouldn't need to.

In principle, yes. But that may just create other problems for us in the future (this was also briefly discussed in #1402), unless we think that C++ code will never propagate deeper than it currently does. We just got rid of the insanity that CUDA code couldn't call any function declared in a header that declared a function that took a t_commrec argument (duplicate functionality that this has prompted in CUDA code should still be removed, though); this would just create a similar mess, except that it would be even more difficult to split headers sensibly. Would we, for example, implement a separate exception hierarchy for CUDA that doesn't depend on boost::exception? If we need to do this, we should make Jenkins fail if someone violates the constraint. And should we make Jenkins fail artificially, even if we can't find a compiler/CUDA/boost combination that would trigger the error?

boost is currently propagating to CUDA likely fromlegacyheaders/smalloc.h -> utility/common.h.

#5 - 03/31/2014 02:36 AM - Roland Schulz
https://gerrit.gromacs.org/#/c/3306/ probably works around the problem which isn't a Gromacs problem. I suggest closing the issue.

04/03/2020
#6 - 04/03/2014 07:31 PM - Roland Schulz
  - Status changed from Blocked, need info to Closed

#7 - 07/15/2014 06:41 AM - Teemu Murtola
  - Related to Bug #1402: Error compiling 5.0-beta1 with CUDA and gcc >= 4.6 added

#8 - 07/15/2014 10:37 AM - Teemu Murtola
  - Assignee changed from Mark Abraham to Teemu Murtola
  - Target version changed from 5.x to 5.0