GROMACS - Task #2819

Task # 2899 (Closed): Update testing matrix versions for GROMACS 2020 release

figure out latest clang + native CUDA that works on our hardware

12/21/2018 08:09 PM - Szilárd Páll

Status: Closed
Priority: Low
Assignee: Szilárd Páll
Category: testing
Target version: 2020-infrastructure-stable
Difficulty: uncategorized

Description
Tried to enable clang 7 native CUDA with CUDA 9.1/9.2 builds in Jenkins and kept running into failing tests (see https://gerrit.gromacs.org/#/c/8663/). The failures suggest that it's a PME compilation issue.
I could not reproduce on CC 3.5, 5.2, and 6.1 hardware outside of Jenkins and given that the build slaves have rather old 3.0 hardware, I suspect this may be an issue specific to that arch.

TODO:
- Confirm the issue and flag in cmake.
- Figure out which clang version can we use without hardware upgrade (consider harwdare upgrade, CC 3.0 is really dated anyway)

Related issues:
Related to GROMACS - Task #3011: misc upgrades of testing matrices Closed

Associated revisions
Revision b3c35087 - 08/19/2019 03:42 PM - Szilárd Páll
Work around clang CUDA device code codegen bug

Some of the PME kernels (namely PME spread and solve) get miscompiled when clang native device code compilation is used with assertions and optimization on, causing errors in RelWithAssert builds.
As a workaround, this change disables optimization for these kernels when assertions are enabled (and only if the device compiler is clang).

Refs #2819
Change-Id: I815fc86116ffb57c4d5803ce7fa4d260909ae7ba

Revision 922e24d5 - 08/19/2019 09:45 PM - Szilárd Páll
Bump clang-cuda post-submit compiler/CUDA version
clang 8 + CUDA 10 is the latest working setup

Fixes #2819
Refs #3006
Change-Id: l3dd0bb5c667d2295593178d445260beaf0509277

History
#1 - 12/21/2018 08:09 PM - Szilárd Páll
- Description updated

Note the low prio, just bump to .1 if not addressed before release.

#2 - 12/21/2018 08:25 PM - Szilárd Páll
- Tracker changed from Bug to Task
- Status changed from New to In Progress
Looks like clang 6 + CUDA 9 doesn’t work either on CC 3.0.

Target version changed from 2019 to 2019.1

bumped

Update: I’ve been making the mistake of compiling in Release mode which has all tests passing. However, the jenkins config is RelWithAssert and with that I can also confirm that PME-GPU tests fail.

Ok. What’s the next move?

Mark Abraham wrote:

Identifying what is different between RelWithAssert and other build types that can/does make unit tests fail. Suggestions would be welcome.

Alternatively we can flag/ignore broken RelWithAssert if cuda-clang is not considered important enough.

Experiment 1. Added NDEBUG around PME solve code to see if this eliminates the failure.

```bash
$ git diff | tail -n1000
diff --git a/src/gromacs/ewald/pme-gpu-internal.cpp b/src/gromacs/ewald/pme-gpu-internal.cpp
index 2a88a02..8271b27 100644
--- a/src/gromacs/ewald/pme-gpu-internal.cpp
+++ b/src/gromacs/ewald/pme-gpu-internal.cpp
@@ -1098,6 +1098,7 @@ void pme_gpu_solve(const PmeGpu *pmeGpu,
     GridOrdering gridOrdering, bool computeEnergyAndVirial)
 { +
     #define NDEBUG
     const bool copyInputAndOutputGrid = pme_gpu_is_testing(pmeGpu) || !pme_gpu_performs_FFT(pmeGpu);
     auto *kernelParamsPtr = pmeGpu->kernelParams.get();
@@ -1199,6 +1200,7 @@ void pme_gpu_solve(const PmeGpu *pmeGpu, t_complex *h_grid,
     GridOrdering gridOrdering, bool computeEnergyAndVirial)
 } +
 #undef NDEBUG
 
 void pme_gpu_gather(PmeGpu *pmeGpu,
```
This still produces errors:

```
$ bin/ewald-test --gtest_filter="SaneInput/PmeSolveTest.ReproducesOutputs/0*"
Note: Google Test filter = SaneInput/PmeSolveTest.ReproducesOutputs/0*
[==========] Running 1 test from 1 test case.
[----------] Global test environment set-up.
[ RUN ] SaneInput/PmeSolveTest.ReproducesOutputs/0
/home/pszilard/projects/gromacs/gromacs-19/src/testutils/refdata.cpp:929: Failure
  In item: /Virial/Cell 0 0
  Actual: -0.11273560672998428
  Reference: 8.2591867446899414
  Difference: 8.37192 (2129332110 single-prec. ULPs, rel. 1.01), signs differ
  Tolerance: abs. 0.00286102, 24 ULPs
  Google Test trace:
  /home/pszilard/projects/gromacs/gromacs-19/src/gromacs/ewald/tests/pmesolvetest.cpp:143: Testing solving (Coulomb, YZX, with energy/virial) with GPU (GPU #0: NVIDIA GeForce GTX 1080, compute cap.: 6.1, ECC: no, stat: compatible) for PME grid size 16 12 28, Ewald coefficients 2 0.7
/home/pszilard/projects/gromacs/gromacs-19/src/testutils/refdata.cpp:929: Failure
  In item: /Virial/Cell 0 0
  Actual: -0.11273560672998428
  Reference: 8.2591867446899414
  Difference: 8.37192 (2129332110 single-prec. ULPs, rel. 1.01), signs differ
  Tolerance: abs. 0.00286102, 24 ULPs
  Google Test trace:
  /home/pszilard/projects/gromacs/gromacs-19/src/gromacs/ewald/tests/pmesolvetest.cpp:143: Testing solving (Coulomb, YZX, with energy/virial) with GPU (GPU #1: NVIDIA GeForce GTX 960, compute cap.: 5.2, ECC: no, stat: compatible) for PME grid size 16 12 28, Ewald coefficients 2 0.7
```

However, compiling the whole binary with NDEBUG still produces passing tests:

```
$ cmake . -DCMAKE_CXX_FLAGS='-DNDEBUG' && make ewald-test
$ bin/ewald-test --gtest_filter="SaneInput/PmeSolveTest.ReproducesOutputs/0*"
Note: Google Test filter = SaneInput/PmeSolveTest.ReproducesOutputs/0*
[==========] Running 1 test from 1 test case.
[----------] Global test environment set-up.
[ RUN ] SaneInput/PmeSolveTest.ReproducesOutputs/0
/home/pszilard/projects/gromacs/gromacs-19/src/testutils/refdata.cpp:929: Failure
  In item: /Virial/Cell 0 0
  Actual: -0.11273560672998428
  Reference: 8.2591867446899414
  Difference: 8.37192 (2129332110 single-prec. ULPs, rel. 1.01), signs differ
  Tolerance: abs. 0.00286102, 24 ULPs
  Google Test trace:
  /home/pszilard/projects/gromacs/gromacs-19/src/gromacs/ewald/tests/pmesolvetest.cpp:143: Testing solving (Coulomb, YZX, with energy/virial) with GPU (GPU #0: NVIDIA GeForce GTX 1080, compute cap.: 6.1, ECC: no, stat: compatible) for PME grid size 16 12 28, Ewald coefficients 2 0.7
```

However, compiling the whole binary with NDEBUG still produces passing tests:
It seems some that in-kernel assertions are what screw up code-generation (\(?\)); if I remove all three assertions nested in the data-dependent if (notZeroPoint) condition (source:src/gromacs/ewald/pme-solve.cu#L206), the errors are gone.

Thoughts anyone?

Retargetting to 2020, as not user facing issue

The issue is eliminated by clang 8, I'll upgrade the post-submit matrix to.

Retargetting to 2020, as not user facing issue

The issue is eliminated by clang 8, I'll upgrade the post-submit matrix to.

There's one remaining issue: clang seems to emit intermediate ptx that the NVIDIA ptx compiler doesn't like and emits these warnings:

```
ptxas warning : .debug_abbrev section not found
ptxas warning : .debug_info section not found
```

We should either suppress these (e.g. in the jenkins parser) or figure out how to disable them.

I've started to implement an exception mechanism for these specific warnings in the jenkins compiler warns groovy parser, but as I'm a groovy noob I ran into hiccups and looked for an alternative. That's when I realized that we don't need debug symbols, just assertions so we can use a RelWithAssert build type. Will switch the matrix.

I've started to implement an exception mechanism for these specific warnings in the jenkins compiler warns groovy parser, but as I'm a groovy noob I ran into hiccups and looked for an alternative. That's when I realized that we don't need debug symbols, just assertions so we can use a RelWithAssert build type. Will switch the matrix.

I take it back: PME is still incorrect with optimizations on and in debug mode we get these warnings that I can't figure out how to skip in a sensible way (i.e. not a sequence of negated characters of the "\(.debug_abbrev section not found" string). Thoughts?

Would prefer to not have to debug PME code, but it might be useful to know why is clang miscompiling it.

I take it back: PME is still incorrect with optimizations on and in debug mode we get these warnings that I can't figure out how to skip in a sensible way (i.e. not a sequence of negated characters of the "\(.debug_abbrev section not found" string). Thoughts?

Would prefer to not have to debug PME code, but it might be useful to know why is clang miscompiling it.
The virtue of the clang_cuda build is clang compiling the device code. It's not essential to do so with optimizations on. Given we already have conditionality in the CMake code for clang_cuda build, we could find a way to append -O0 or something.

#18 - 08/15/2019 08:57 PM - Mark Abraham

Mark Abraham wrote:

The virtue of the clang_cuda build is clang compiling the device code. It's not essential to do so with optimizations on. Given we already have conditionality in the CMake code for clang_cuda build, we could find a way to append -O0 or something.

Or just test this config in Debug mode. We don't care about perf or codegen for this configuration.

#19 - 08/16/2019 05:57 PM - Szilárd Páll

- Category set to testing
- Status changed from In Progress to Fix uploaded

#20 - 08/19/2019 04:21 PM - Szilárd Páll

- Related to Task #3011: misc upgrades of testing matrices added

#21 - 08/19/2019 10:00 PM - Szilárd Páll

- Status changed from Fix uploaded to Resolved

Applied in changeset 922e24d552ca18045638ae759f9e739450ac1276.

#22 - 08/23/2019 11:46 AM - Szilárd Páll

- Status changed from Resolved to Closed