**Description**

Two tests in the basic 'make check' suite fail when compiling Debian packages from the 2016-beta2 tarball on the i386 (32-bit) architecture. These are from the single precision / non-MPI build; the test failures abort the build so I haven't yet checked the double precision or MPI variants for the same issue. It fails on three different build systems with three different kernels (Linux, FreeBSD, and Hurd), so it's unlikely to be a flaky machine. From the displayed output, though, it's not immediately obvious to me what's failing.

```c
[----------] 4 tests from NormalDistributionTest
[ RUN      ] NormalDistributionTest.Output
[    OK    ] NormalDistributionTest.Output (0 ms)
[ RUN      ] NormalDistributionTest.Logical
[    OK    ] NormalDistributionTest.Logical (0 ms)
[ RUN      ] NormalDistributionTest.Reset
  /«PKGBUILDDIR»/src/gromacs/random/tests/normaldistribution.cpp:104: Failure
  Value of: valB
    Actual: -1.19571
  Expected: valA
    Which is: -1.19571
[ FAILED   ] NormalDistributionTest.Reset (0 ms)

[ RUN      ] NormalDistributionTest.AltParam
  /«PKGBUILDDIR»/src/gromacs/random/tests/normaldistribution.cpp:120: Failure
  Value of: distB(rngB, paramA)
    Actual: -1.19571
  Expected: distA(rngA)
    Which is: -1.19571
[ FAILED   ] NormalDistributionTest.AltParam (0 ms)
[----------] 4 tests from NormalDistributionTest (0 ms total)
```

**Build logs**

- [https://buildd.debian.org/status/fetch.php?pkg=gromacs&arch=i386&ver=2016%7Ebeta2-1&stamp=1465406125](https://buildd.debian.org/status/fetch.php?pkg=gromacs&arch=i386&ver=2016%7Ebeta2-1&stamp=1465406125)

**Associated revisions**

Revision fa1d62c6 - 07/08/2016 11:08 AM - Erik Lindahl

Work around compiler issue with random test

```
gcc-4.8.4 running on 32-bit Linux fails a few tests for random distributions. This seems to be caused by the compiler doing something strange (that can lead to differences in the lsb) when we do not use the result as floating-point values, but rather do exact binary comparisons. This is valid C++, and bad behaviour of the compiler (IMHO), but technically it is not required to produce bitwise identical results at high optimization. However, by using floating-point tests with zero ULP tolerance the problem appears to go away.
```
Fixes #1986.

Change-Id: i252f37b46605424c02435af0f7a4f81b493eb8

History

#1 - 06/13/2016 11:24 AM - Gerrit Code Review Bot
Gerrit received a related patchset '1' for Issue #1986.
Uploader: Berk Hess (hess@kth.se)
Change-Id: f3461ae29584e8afba6d1cb646b7aeb16de
Gerrit URL: https://gerrit.gromacs.org/5951

#2 - 06/13/2016 11:26 AM - Berk Hess
- Category set to mdrun
- Status changed from New to Feedback wanted

There was already a loosening of the precision of this check committed to the master branch. I don't know why this was not committed to release-2016. I assume this fixes your issue as well. Could you try: https://gerrit.gromacs.org/#/c/5951

#3 - 06/18/2016 01:05 AM - Nicholas Breen
I see that 5951 has already been abandoned, but just to confirm: no, that patch does not affect the issue.

#4 - 07/01/2016 01:13 AM - Nicholas Breen
In the two affected tests, replacing EXPECT_EQ with EXPECT_FLOAT_EQ allows the test to pass again. This seems like an intuitively correct solution - EXPECT_FLOAT_EQ is already used extensively in other tests.

Patch: https://paste.debian.net/plain/772649

As a side note, although single-precision passes all tests with this patch, four different tests fail on the double-precision build. Needs a little more investigation before submitting new issues. However, two of those tests are found in random/tests/uniformrealdistribution.cpp and also use EXPECT_EQ on floating point values, which fail in the same way; I expect these need to be converted to EXPECT_FLOAT_EQ as well.

#5 - 07/01/2016 02:07 AM - Gerrit Code Review Bot
Gerrit received a related patchset '1' for Issue #1986.
Uploader: Mark Abraham (mark.j.abraham@gmail.com)
Change-Id: id597f9f8952703186f2363c0f9f2d25cf790190ce
Gerrit URL: https://gerrit.gromacs.org/6000

#6 - 07/01/2016 02:09 AM - Mark Abraham
I've been through the implementation and didn't find anything. The underlying UniformRealDistribution and Random engine both pass reset tests. My only lead is that http://redmine.gromacs.org/projects/gromacs/repository/revisions/master/entry/src/gromacs/random/threefry.h#L525 uses internalCounterBitsBits and subsequent methods use internalCounterBits. I think the former is more likely to be wrong, based on the docs, but the alternative of using BitsBits in all three places made the Discard test fail. We need Erik to clarify whether part of this is e.g. a hangover from refactoring his implementation.

#7 - 07/01/2016 02:22 AM - Mark Abraham
Nicholas Breen wrote:

In the two affected tests, replacing EXPECT_EQ with EXPECT_FLOAT_EQ allows the test to pass again. This seems like an intuitively correct solution - EXPECT_FLOAT_EQ is already used extensively in other tests.

Right, that logical change is what we thought we were doing with https://gerrit.gromacs.org/#/c/5951, but the tests TabulatedNormalDistribution and NormalDistribution are different things. We have our own things like EXPECT_FLOAT_EQ that allow us to choose ULP tolerances that suit particular contexts, but they share the default of 4 that GoogleTest uses.

Patch: https://paste.debian.net/plain/772649

As a side note, although single-precision passes all tests with this patch, four different tests fail on the double-precision build. Needs a little more investigation before submitting new issues. However, two of those tests are found in random/tests/uniformrealdistribution.cpp and also use EXPECT_EQ on floating point values, which fail in the same way; I expect these need to be converted to EXPECT_FLOAT_EQ as well.

OK, that would make sense - the fine detail of whether and how any extended precision was used will indeed mean binary reproducibility is unreliable, and particularly unlikely for a machine with a different word size.
Hmm, no, something is unsatisfying about needing a tolerance. We're running a fully reproducible algorithm twice on the same hardware, so we should get a binary-identical result. It mightn't be the same as on e.g. 64-bit machine but that's not the reason for the failure.

Which test cases fail in double precision, please?

Mark Abraham wrote:

Which test cases fail in double precision, please?

- TabulatedNormalDistributionTableTest.HasValidProperties (see #1930)
- UniformRealDistributionTest.Reset
- UniformRealDistributionTest.AltParam
- SimdIntegerTest.cvtR2I (with -DGMX_SIMD=SSE2, other configurations not yet tested)
- SimdIntegerTest.cvttR2I (likewise)

This is after my modification to NormalDistributionTest.Reset and .AltParam, which I have not separately tested in double precision yet. Possible that those would continue to be an issue otherwise.

Changing the tests to use our own EXPECT_REAL_EQ_TOL with a zero tolerance would produce a better failure message.

I agree with Mark that in principle, the test should produce binary identical result from doing the same thing twice. Is this an optimized build? It is conceivable that if the compiler inlines all the involved functions and then optimises that as a whole that it would produce a different arithmetic order for some operations, but that's just a wild guess.

Mark Abraham wrote:

Hmm, those last two are also failing at #1997, which I had so far attributed to rounding modes being used the wrong way (but maybe the word unpacking is also an issue on that arch?).

Perhaps we should be using strict IEEE for compiling our test binaries?

Erik Lindahl wrote:

- Status changed from Feedback wanted to In Progress
- Assignee set to Erik Lindahl

Update: All these failures are benign. One of the double precision tests was caused by a too tight tolerance, and the reason we see it on i386 is likely that the compiler cannot issue FMA instructions for that case (both FMA and non-FMA are valid IEEE) - the result is still within 3ULP in double precision after a large summation.

The other failures appear to be fragile in the sense that they disappear the second I try to print the result, and then it suddenly matches perfectly bit-by-bit. I need to look into Google test to check if there is any special reason we need to avoid calling EXPECT_EQ() for floating-point numbers, or if it's a compiler thing.

Gerrit received a related patchset '1' for Issue #1986.
Uploader: Erik Lindahl (erik.lindahl@gmail.com)
Change-Id: I252f37b46605424c02435af0fb7a4f81b493eb8
Gerrit URL: https://gerrit.gromacs.org/6023
Erik Lindahl wrote:

The other failures appear to be fragile in the sense that they disappear the second I try to print the result, and then it suddenly matches perfectly bit-by-bit. I need to look into Google test to check if there is any special reason we need to avoid calling EXPECT_EQ() for floating-point numbers, or if it's a compiler thing.

I doubt there's a special reason, but if there was, one could cast to a suitable integer and compare that.

I do both in the new patch (FP comparison provides much more readable output when the test fails completely, but the integer test is great to see if there's a 1-bit-difference).

Gerrit received a related patchset '1' for Issue #1986.
Uploader: Erik Lindahl (erik.lindahl@gmail.com)
Change-IID: I5518eeccc7dcdbb020d83c6beaa82b99e3691c28
Gerrit URL: https://gerrit.gromacs.org/6024

- Status changed from In Progress to Fix uploaded

- Status changed from Fix uploaded to Resolved

Applied in changeset fa1d62c66cfd845fcd733783a7881bca7aa8c1d.

- Status changed from Resolved to Closed