GROMACS - Bug #2164

SIMD sqrt in double-precision build does not work correctly

04/21/2017 11:13 PM - Mark Abraham

Status:	Closed		
Priority:	Normal		
Assignee:			
Category:	core library		
Target version:	2018		
Affected version - extra info:	likely all versions since the new SIMD layer came about	**Difficulty:**	uncategorized
Affected version:	git master		

Description

For a double precision build, SIMD sqrt(x) for a double x is implemented by x * rsqrt(x). rsqrt(x) is implemented via truncation to single, which produces inf at some point if x is suitably small (presumably x < smallest double that is representable in float).

code in simd_math.h is

```
static inline SimdDouble gmx_simdcall
sqrt(SimdDouble x)
{
    return x * maskzInvsqrt(x, setZero() < x);
}
```

If we're happy with the SIMD library producing the value zero for the sqrt of such small values of x, then we can fix it by instead clamping the range to "less than some small positive value of x" rather than zero (not sure offhand what value is best).

Found by Carsten's essentialdynamics linfix regressiontest in double on avx_256. This has a single dihedral that coincidentally produces a nearly zero angle on step 5.

Related issues:

Related to GROMACS - Task #2134: assess whether Jenkins is testing multi-rank...	**Closed**
Related to GROMACS - Feature #2163: Use better input/output value choices for...	**Closed**

Associated revisions

Revision e34ead15 - 09/04/2017 11:41 PM - Erik Lindahl

More SIMD math argument checking, added unsafe options

This change adds more argument checking and safeguards
for sqrt, exp2, and exp-related SIMD math functions, and
properly documents allowed values. These functions now
have an (optional) template parameter that makes it possible
to avoid the checks where it is important to save every cycle,
and the developer is certain that this usage is fine. For
now we only use the unsafe versions in the nonbonded kernels.
The SIMD function test code has also been extended with options
to allow denormals to be considered zero.

Fixes #2164.
Refs #2163.

Change-Id: I93ddadf74dd0fa013f61cf27fd1993f11cde28bc

Revision 6d32275c - 09/09/2017 09:24 AM - Berk Hess

Avoid inf in SIMD double sqrt()

Arguments >0 and <float_min to double precision SIMD sqrt()
would produce inf on many SIMD architectures. Now sqrt() will
return 0 for arguments in this range, which is not fully correct,
but should be unproblematic.
Updated the tests to check for this range and to produce output
that checks all double precision mantissa bits.

Fixes #2164.
Refs #2163.

Change-Id: Ic6d2c6d4102d602703b40e7e8bcc1974a7283f7c

History

#1 - 04/21/2017 11:13 PM - Mark Abraham

- Related to Task #2134: assess whether Jenkins is testing multi-rank runs appropriately added

#2 - 04/21/2017 11:14 PM - Mark Abraham

- Related to Feature #2163: Use better input/output value choices for SIMD tests added

#3 - 04/22/2017 10:17 AM - Berk Hess

The clamping you suggest is not nice, as you often take the sqrt of a square, you get issues at values of float_min (1e-38) instead of double_min (1e-308). Still one could wonder if there are cases where it matters that one gets 0 instead of 1e-38.
A somewhat better solution would be to clamp the value passed to (inv)sqrt to float_min or to have a different maskRsqrt function that return sqrt(float_max) when masked. I don't know how many extra instructions that would take.

An even simpler solution is to use double rsqrt. I have done timings, but from the Intel SIMD intrinsics guide it would seem that the two conversions + single rsqrt already take more time than a double rsqrt on Haswell. On Skylake double rsqrt is as fast as single, so there we should surely use the double rsqrt.

#4 - 04/22/2017 10:22 AM - Berk Hess

Ignore the last paragraph in my previous comment. There is no rsqrt in double on x86.

#5 - 04/22/2017 10:34 AM - Berk Hess

Also ignore my other suggestions (I need more coffee, I think).

The clamping to zero for float_min will switch to zero for return values smaller than 1e-19, which is relatively early, but I don't see cases where it would really matter. But masking invsqrt with sqrt(float_max) doesn't help accuracy much.
So clamping for input to sqrt <float_min is certainly better than what we do now (and not much more expensive). I don't think there are better solutions that are not at least double the cost.

#6 - 04/22/2017 11:43 AM - Gerrit Code Review Bot

Gerrit received a related patchset '1' for Issue #2164.
Uploader: Berk Hess (hess@kth.se)
Change-Id: gromacs~release-2016~Ic6d2c6d4102d602703b40e7e8bcc1974a7283f7c
Gerrit URL: https://gerrit.gromacs.org/6599

#7 - 04/22/2017 11:53 AM - Erik Lindahl

Actually, there is a double version of it, but not until Skylake with AVX512 - in that case we already use it.

However, the problem isn't at all in sqrt(). That routine only works with double, and all comparisons are correct. The main issue occurs with the double-to-float-to-double conversions inside the rsqrt() call of invsqrt() - and this routine is in the critical path.

Over the last few years I think we've changed attitude to say that performance is irrelevant unless we can guarantee that routines always provide correct results.

The proper way to fix it is to clamp values to single precision before calling either rsqrt or rcp. Since the result from that call will still be valid to within the bits specified in the lookup table, the N-R iterations will provide a correct result.

#8 - 04/22/2017 12:03 PM - Erik Lindahl

Actually, even that won't help, since the clamping means we don't get enough valid bits in the lookup. I need to think a bit about the math here, but this won't be a "free lunch" fix.

#9 - 04/22/2017 12:23 PM - Erik Lindahl

Actually, as far as I can see, all SIMD architectures except x86 have the lookup work right (i.e., it will provide correct results with double precision), and even on x86 it works from AVX512.

Based on that, the way to fix it is that we will need to add code to the rsqrt() and rcp() SIMD calls for SSE & AVX so the double precision lookups provide correct results.

#10 - 04/22/2017 04:51 PM - Erik Lindahl

I've had a look to see what compilers generate internally, and unfortunately there doesn't appear to be any easy fix. We might have to solve it my providing both normal and "fast" versions of sqrt/invsqrt.

#11 - 04/22/2017 06:54 PM - Mark Abraham

It's quite reasonable to say we clamp either sqrt or rsqrt to zero when the value is super small. We could choose not to use that rsqrt implementation in normal kernels. We just can't return inf when a dihedral has nearly zero angle, or someone produced coordinates that almost overlap. Presumably the inf arises from UB when the truncation to single precision cannot succeed?

#12 - 05/25/2017 03:09 PM - Gerrit Code Review Bot

Gerrit received a related patchset '1' for Issue #2164.
Uploader: Erik Lindahl (erik.lindahl@gmail.com)
Change-Id: gromacs~master~I93ddadf74dd0fa013f61cf27fd1993f11cde28bc
Gerrit URL: https://gerrit.gromacs.org/6661

#13 - 05/31/2017 08:29 PM - Szilárd Páll

- Status changed from New to In Progress

#14 - 06/05/2017 04:03 PM - Erik Lindahl

- Status changed from In Progress to Fix uploaded

- Target version changed from 2016.4 to 2018

Changing the target to 2017, since this requires fairly large changes both to code and the specifications for what SIMD values we allow in double precision builds.

Since this doesn't seem to have hurt any actual simulations, it seems like a reasonable balance to focus on future versions instead.

#15 - 09/05/2017 06:29 PM - Erik Lindahl

- Status changed from Fix uploaded to Resolved

Applied in changeset e34ead15b327872ba9362de9416178cce53158b3.

#16 - 09/11/2017 03:51 PM - Berk Hess

Applied in changeset 6d32275cd2948aa93949d819b09d3957a2ef48dd.

#17 - 10/25/2017 04:57 PM - Aleksei Iupinov

- Status changed from Resolved to Closed