RELEASE NOTES

Fix numerical divergence on x86 and arm64. For best performance for WRF on arm64 please use armclang.

The numerical divergence on x86 and arm64 are due to

- difference between the two architectures for some transcendental functions due to round up error at last position (details see Appendix below)
- certain compiler optimizations (FMA, reciprocal math, ...)

For CONUS 12-km and 2.5-km models, exact (bit-by-bit matching) output models can be produced on x86 and arm64 using gcc compiler (gcc 10.2) and math library (glibc 2.27) with this patch. The performance degradation is negligible. To repeat this experiment on the latest WRF or WRF 4.2.2, follow the instructions below.
Please note that this work is to evaluate the severity of the numerical divergence between x86 and arm64, understand what's causing the divergence, and how to close the difference between the two. This type of numerical divergence are likely to show up again when using a different math library, version of math library, or compiler (Arm, Intel, ...).
This work also shows the original output models are both correct despite the numerical differences. It will be helpful if a criteria for acceptable differences can be established and published for arm64 and other architectures, similar to x86 (https://www2.mmm.ucar.edu/wrf/users/benchmark/v42/benchdata_v422.html).

PROCEDURES FOR THIS EXPERIMENT

1. create two parallel clusters (Intel and Graviton) with Ubuntu 18.04; the default math library is glibc 2.27
2. install WRF dependencies, including gcc-10.2, openmpi-4.1.0, zlib-1.2.13, hdf5-1.12.0, pnetcdf-1.12.2, netcdf-c-4.8.1, and netcdf-fortran-4.5.4
3. get the latest WRF code and this patch (PR 1773)

```
git clone https://github.com/wrf-model/WRF.git
cd WRF
git pull origin pull/1773/head
```

on Graviton (c6g or c7g), the default setting is sufficient.
on Intel (c5n or c6i), uncomment line 793,
`#ARCH_LOCAL = -DNONSTANDARD_SYSTEM_SUBR CONFIGURE_D_CTSM -DAARCH64_X86_CORRECTNESS_FIX`

or WRF v4.2.2 and this patch

```
curl -o WRF-v4.2.2.zip -J -L https://github.com/wrf-model/WRF/archive/v4.2.2.zip
unzip WRF-v4.2.2.zip
```

update phys/module_cu_tiedtke.F, phys/module_sf_myjsfc.F and arch/configure.defaults

4. configure WRF and install

on Intel, choose 35 (gcc, DM+SM) and 1 (default nesting); on Graviton, choose 12 (gcc, DM+SM) and 1

5. run WRF for CONUS 2.5-km and 12-km models

```
ulimit -s unlimited

export OMP_STACKSIZE=12G
export OMP_NUM_THREADS=8
export KMP_AFFINITY=compact,verbose
```

```
mpirun -report-bindings -n 8 -map-by socket:PE=8 ./wrf.exe
```

6. compare output models using either ncl tools or diffwrf.py

APPENDIX - ROUND UP ERROR AT LAST POSITION (GCC 10.2, GLIBC 2.27)

First divergence happens at the following positions and later magnified by the time integral and Stencil kernels.

1. module_sf_myjsfc.F line 1216

```
          PSIM1(K)=-2.*LOG((X+1.)/2.)-LOG((X*X+1.)/2.)+2.*ATAN(X)-PIHF
```

on arm64,

```
(gdb) p x
$387 = 2.75693512
(gdb) p pihf
$388 = 1.57079625
(gdb) p psim1(k)
$389 = -1.84473705
```

on intel,

```
(gdb) p x
$21 = 2.75693512
(gdb) p pihf
$22 = 1.57079625
(gdb) p psim1(k)
$23 = -1.84473729
```

check inside registers
on arm64
input to atanf

```
(gdb) p $s0
$385 = { f = 2.75693512, u = 1076916640, s = 1076916640 }
```

output from atanf

```
(gdb) p $s0
$382 = { f = 1.22283351, u = 1067222479, s = 1067222479 }
```

on intel
input to atanf

```
(gdb) p $xmm0
$18 = { v8_bfloat16 = {1.585e+30, 2.75, 0, 0, 0, 0, 0, 0}, v8_half = {11520, 2.0938, 0,
0, 0, 0, 0, 0}, v4_float = {2.75693512, 0, 0, 0}, v2_double = {5.320675152587852e-315,
0}, v16_int8 = {-96, 113, 48, 64, 0, <repeats 12 times>}, v8_int16 = {29088, 16432, 0,
0, 0, 0, 0, 0}, v4_int32 = {1076916640, 0, 0, 0}, v2_int64 = {1076916640, 0}, uint128 =
1076916640 }
```

output from atanf

```
(gdb) p $xmm0
$15 = { v8_bfloat16 = {-1.937e-35, 1.219, 0, 0, 0, 0, 0, 0}, v8_half = {-8.8573e-05,
1.9023, 0, 0, 0, 0, 0, 0}, v4_float = {1.2228334, 0, 0, 0}, v2_double =
{5.2727796284936553e-315, 0}, v16_int8 = {-50, -123, -100, 63, 0, <repeats 12 times>},
```

```
v8_int16 = (-31282, 16284, 0, 0, 0, 0, 0, 0), v4_int32 = (1067222478, 0, 0, 0), v2_int64
= (1067222478, 0), uint128 = 1067222478 )
```

2. module_cu_tiedtke.F line 2939

```
         zorgde = tan(arg)*3.1415*0.5/ztmzk
```

on arm64

```
(gdb) p arg
$17 = 0.529116511
(gdb) p ztmzk
$18 = 2903.62988
 (gdb) p zorgde
$16 = 0.000316316495
```

on x86

```
(gdb) p arg
$17 = 0.529116511
(gdb) p ztmzk
$18 = 2903.62988
(gdb) p zorgde
$16 = 0.000316316466
```

registers on arm64

```
(input to tan())
(gdb) p $s0
$14 = ( f = 0.529116511, u = 1057453102, s = 1057453102 )
(gdb) ni
0x0000000001e8ee54      2938                  zorgde = tan(arg)*3.1415*0.5/ztmzk
(output from tan())
(gdb) p $s0
$15 = ( f = 0.584730864, u = 1058386156, s = 1058386156 )
```

registers on x86

```
(input)
(gdb) p $xmm0
$14 = ( v8_bfloat16 = (5.514e+31, 0.5273, 0, 0, 0, 0, 0, 0), v8_half = (17120, 1.7568,
0, 0, 0, 0, 0, 0), v4_float = (0.529116511, 0, 0, 0), v2_double = (5.2245124978645956e-
315, 0), v16_int8 = (46, 116, 7, 63, 0, <repeats 12 times>), v8_int16 = (29742, 16135,
0, 0, 0, 0, 0, 0), v4_int32 = (1057453102, 0, 0, 0), v2_int64 = (1057453102, 0), uint128
= 1057453102 )
(gdb)  ni
0x0000000003ceee0f      2938                  zorgde = tan(arg)*3.1415*0.5/ztmzk
(output)
(gdb) p $xmm0
$15 = ( v8_bfloat16 = (-1.71e-09, 0.582, 0, 0, 0, 0, 0, 0), v8_half = (-0.15369, 1.7705,
0, 0, 0, 0, 0, 0), v4_float = (0.584730804, 0, 0, 0), v2_double = (5.2291223921950867e-
315, 0), v16_int8 = (-21, -80, 21, 63, 0, <repeats 12 times>), v8_int16 = (-20245,
16149, 0, 0, 0, 0, 0, 0), v4_int32 = (1058386155, 0, 0, 0), v2_int64 = (1058386155, 0),
uint128 = 1058386155 )
```