Better Compression for UFS with Support from the NetCDF Community

Unifying Innovations in Forecasting Capabilities Workshop, July, 2023

Edward Hartnett, CIRES/NOAA

This research was supported by NOAA cooperative agreements NA17OAR4320101 and NA22OAR4320151.

UIFCW 2023
A UFS Collaboration Powered by EPIC

New Compression Features are Available in NetCDF

Three new compression features are now available in netCDF.

- Parallel I/O + compression.
- Zstandard - new lossless compression.
- Quantize - enables lossy compression.

Using the new features will result in faster I/O, and smaller data files.

- Use compression with parallel I/O (since netcdf-c-4.7.4).
- Use Zstandard compression instead of zlib, for better compression and faster I/O.
- Use the new quantize feature with floating point data to enable lossy compression.
- Using both zstandard and quantize may result in 5x improvement in compression and I/O.
- Data are backward compatible with existing codes (once they are re-linked with new version of netCDF).

A History of Community Collaboration

UIFCW 2023
A UFS Collaboration Powered by EPIC

Date	Feature	Impact	Collaborators
2016	NCO tools support new compression, bit-grooming.	New compression filters in post-processed data, demonstrates value of bit-grooming. (Geosci. Model Dev)	U. of California, Irvine
2019	Bit-shaving in GFS atmospheric history data.	Reduces size of ATM history file from 36 GB to 6 GB.	NOAA
2019	Creation of CCR project.	Allows testing of new compression filters in netCDF. (AMS)	U. of California, Irvine, Unidata/UCAR, NOAA
2020	NetCDF parallel I/O writing using compression.	Order of magnitude improvement in write time for GFSv16. (WMO WGNE Blue Book)	NOAA, Unidata/UCAR
2021	Test with UFS; Add quantize/zstandard support.	Demonstrated value to NOAA and the wider community. (AGU)	NOAA, Unidata/UCAR, NetCDF power users.
2022	netcdf-c-4.9.0.	Quantization/zstandard available to all. (EGU)	NOAA, U. of California, Irvine, Unidata/UCAR
2023	netcdf-c-4.9.2/ netcdf-fortran-4.6.1.	Increased ease-of-use for Fortran users.	NOAA, Unidata/UCAR, U. of California, Irvine,

Parallel IO with Compression

- Added to netcdf-c-4.7.4 to support GFS 16.
- All compression features now work with parallel I/O.
- Special thanks to Unidata/UCAR for quickly doing a release in support of GFS-16.

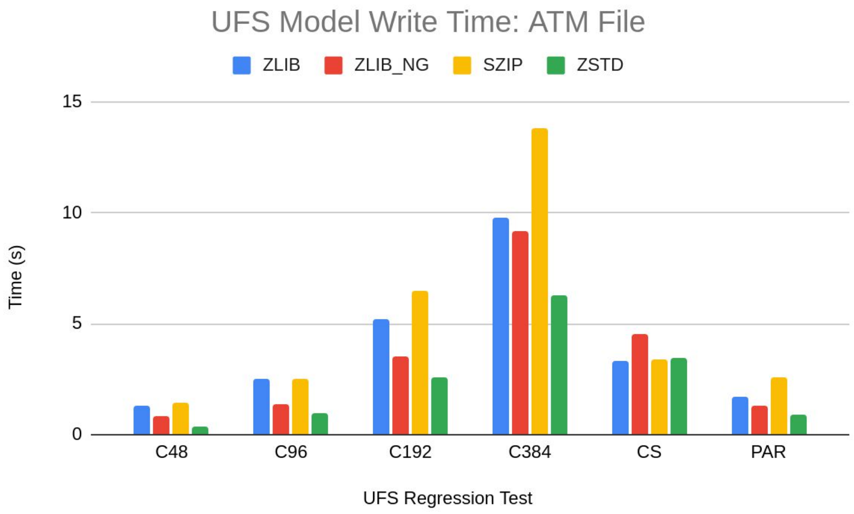
C768L127 fcst output	Nemsio No compression	Netcdf No compression	Netcdf Lossless (deflate=1,nbit=0)	Netcdf Lossy (deflate =1, nbit=20)	Netcdf Lossy(deflate=1,nbit=14)	Netcdf Lossy (deflate=1, nbits=14),parallel writing, default decomposition chunksize	Netcdf Lossy (deflate=1, nbits=14),parallel writing Layer chunkcsize
A 3D file size, (total fcst)	33.6GB (7TB)	33.6GB (7TB)	23.6GB (5TB)	13.5GB (2.8TB)	6.3GB (1.3TB)	6.3GB (1.3TB)	6.3GB (1.3TB)
Write Time	79s	300s	960s	680s	400s	43s	34s

- **GFSv16 could NOT be implemented** without this feature!
- **Collaborated with Unidata and PSL**, testing, release and deployment in operations in under two months

Computational performance improvements in GFSv16, Jun Wang, Jeffrey Whitaker, Edward Hartnett, James Abeles, Gerhard Theurich, Wen Meng, Cory Martin,Jose-Henrique Alves, Fanglin Yang, Arun Chawla

Zstandard is Faster than Zlib

- Zstandard is lossless compression.
- Zstandard library is available on all platforms (Unix/Windows/Mac).
- Significantly faster and more compressive than zlib.
- Provides greater control (than zlib) of speed/compression trade-off.
- Very fast decompression.



Quantization and Next-Generation Zlib Compression for Fully Backward-Compatible, Faster, and More Effective Data Compression in NetCDF Files - Edward Hartnett, Charles S. Zender, Ward Fisher, Dennis Heimbigner, Hang Lei, Brian Curtis, Kyle Gerheiser (see also extended abstract)

UIFCW 2023
A UFS Collaboration Powered by EPIC

New Zstandard Parameter in nf90_def_var

```
nf90_def_var(ncid, VAR1_NAME, NF90_FLOAT, dimids, varid1, &
          zstandard_level = 4, shuffle = shuffle)
```

Do not try to use **both** zstandard_level and deflate_level.

Quantization Enables Lossy Compression

- Will only reduce data size if zlib/zstandard compression is turned on.
- Only for NC_FLOAT, NC_DOUBLE types.
- Fill values are not quantized.
- Quantize works and is tested with parallel I/O.
- Quantized data are fully backward-compatible, and can be read correctly by all versions of netCDF and netCDF-Java.
- An attribute is added to the data variable, recording the algorithm and the number of significant digits.

Sign	Exponent	Fraction (significand)	Decimal	Notes
0	10000000	10010010000111111011011	3.14159265	Exact
0	10000000	10010010000111111011011	3.14159265	NSD = 8
0	10000000	10010010000111111011010	3.14159262	NSD = 7
0	10000000	10010010000111111011000	3.14159203	NSD = 6
0	10000000	10010010000111111000000	3.14158630	NSD = 5
0	10000000	10010010000111100000000	3.14154053	NSD = 4
0	10000000	10010010000000000000000	3.14062500	NSD = 3
0	10000000	10010010000000000000000	3.14062500	NSD = 2
0	10000000	10010000000000000000000	3.12500000	NSD = 1

Figure 1: The value of Pi expressed as a 32-bit floating point number, with different levels of quantization applied, from Number of Significant Digits (NSD) equal to 8 (no quantization), to 1 (maximum quantization). The least significant bits of the significand are replaced with zeros, to the extent possible, while preserving the desired number of significant digits. In this example the Bit Grooming quantization algorithm is used.

Hartnett, Zender, EGU22-13259, Adding Quantization to the NetCDF C and Fortran Libraries to Enable Lossy Compression

Quantize Algorithms

BitGroom

Determines rounding bitmask for all data which preserves NSD (decimal digits). Alternate values have extra bits set to 0/1. Fast, conservative algorithm.

Granular BitRound

Determines rounding bitmask for each value which preserves NSD (decimal digits). IEEE rounding. Slightly slower, more aggressive algorithm.

BitRound

Allows user to specify number of bits to be retained. IEEE rounding. Allows user to specify bits instead of decimal digits.

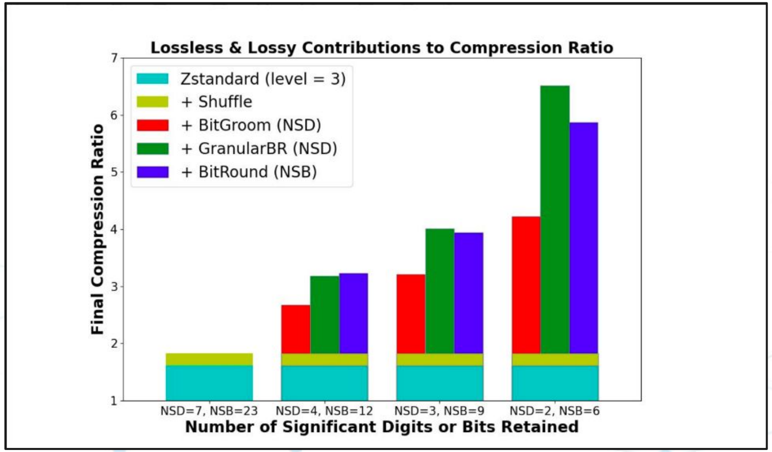
Most users will want Granular BitRound.

New Quantize Parameters in nf90_def_var

```
nf90_def_var(ncid, VAR1_NAME, NF90_FLOAT, dimids, varid1, &
     deflate_level = DEFLATE_LEVEL, &
     quantize_mode = nf90_quantize_bitgroom, nsd = 3)
```

Use Zstandard + Quantize for Best Results

- Zstandard can be tried quickly, no change in data results.
- Quantization will change output and require decisions about NSD.
- Downstream code does not need to change for either compression filter or use of quantization.
- Downstream netCDF installs must be updated to read zstandard compressed data.
- Quantized data can be read on all downstream installations.



Charlie Zender - Lossy compression: The netCDF implementation and towards encoding precision

Installing NetCDF with Compression Features

Some build settings are required to get the netCDF C and Fortran to work with zstandard.

Quantization requires no special settings.

netcdf-c and netcdf-fortran must be upgraded to use the new features.

Once netCDF is correctly installed, and programs are re-built with the new version of netCDF, no code changes are required in reading code for zstandard or quantize.

Only the nf90_def_var() call in the writing program needs to change.

Installing NetCDF - The C Library

- netcdf-c-4.9.2
- Use –enable-parallel-tests to turn on parallel I/O tests with mpiexec.
- Use –with-plugin-dir to get zstandard HDF5 plugin correctly installed.
- For FISMA disable DAP, byterange, and ncZarr. (Though these are great features!)

```
t CC=mpicc
t CPPFLAGS=-I/usr/local/hdf5-1.14.1_mpich/include
t LDFLAGS=-L/usr/local/hdf5-1.14.1_mpich/lib
figure --enable-parallel-tests --with-plugin-dir --disable-dap --disable-byterange --disable-nczarr –prefix=.
-j check
-j install
```

UIFCW 2023
A UFS Collaboration Powered by EPIC

Installing NetCDF - The Fortran Libraries

- netcdf-fortran-4.6.1
- Must set env var HDF5_PLUGIN_PATH

```
export HDF5_PLUGIN_PATH=/usr/local/hdf5/lib/plugin
export FC=mpifort
export FCFLAGS=-I/usr/local/netcdf-c-4.9.2_hdf5-1.14.1_mpich/include
export CPPFLAGS=-I/usr/local/netcdf-c-4.9.2_hdf5-1.14.1_mpich/include
export LDFLAGS=-L/usr/local/netcdf-c-4.9.2_hdf5-1.14.1_mpich/lib
./configure --prefix=/usr/local/netcdf-fortran-4.6.1_mpich --enable-parallel-tests
make -j check
make -j install
```

```
# NetCDF C Configuration Summary
==============================

# General
-------
NetCDF Version:             4.9.2
Dispatch Version:           5
Configured On:              Thu Jun  1 12:06:34 MDT 2023
Host System:                x86_64-pc-linux-gnu
Build Directory:         /home/ed/Downloads/netcdf-c-4.9.2
Install Prefix:          /usr/local/netcdf-c-4.9.2_hdf5-1.14.1_mpich
Plugin Install Prefix:   /usr/local/hdf5/lib/plugin

# Compiling Options
-----------------
C Compiler:                 /usr/bin/mpicc
CFLAGS:                      -fno-strict-aliasing
CPPFLAGS:               -I/usr/local/hdf5-1.14.1_mpich/include
LDFLAGS:                -L/usr/local/hdf5-1.14.1_mpich/lib
AM_CFLAGS:
AM_CPPFLAGS:
AM_LDFLAGS:
Shared Library:             yes
Static Library:             yes
Extra libraries:         -lhdf5_hl -lhdf5 -lm -lz -ldl -lzstd -lxml2
XML Parser:               libxml2

# Features
--------
Benchmarks:                 no
NetCDF-2 API:               yes
HDF4 Support:               no
HDF5 Support:               yes
NetCDF-4 API:               yes
CDF5 Support:               yes
NC-4 Parallel Support:      yes
PnetCDF Support:            no

DAP2 Support:               no
DAP4 Support:               no
Byte-Range Support:         no

S3 Support:                 no

NCZarr Support:             no
NCZarr Zip Support:         no

Diskless Support:           yes
MMap Support:               no
JNA Support:                no
ERANGE Fill Support:        no
Relaxed Boundary Check:     yes

Multi-Filter Support:       yes
Quantization:               yes
Logging:                    no
SZIP Write Support:         no
Standard Filters:           deflate bz2 zstd
ZSTD Support:               yes
Parallel Filters:           yes
```

```
# NetCDF Fortran Configuration Summary
==============================

# General
-------
Library Version:            4.6.1
Configured On:              Thu Jun  1 12:42:55 MDT 2023
Host System:                x86_64-pc-linux-gnu
Build Directory:            /home/ed/Downloads/netcdf-fortran-4.6.1
Install Prefix:             /usr/local/netcdf-fortran-4.6.1_mpich

# Compiling Options
-----------------
Fortran Compiler:           /usr/bin/mpifort
FFLAGS:                     -g -O2
LDFLAGS:                    -L/usr/local/netcdf-c-4.9.2_hdf5-1.14.1_mpich/lib
C Compiler:                 gcc
CPPFLAGS:                   -I/usr/local/netcdf-c-4.9.2_hdf5-1.14.1_mpich/include
CFLAGS:                     -g -O2 -DLONGLONG_IS_LONG
Shared Library:             yes
Static Library:             yes
Extra libraries:            -lnetcdf -ldl -lm

# Features
--------
F03:                        yes
Dap Support:                no
Logging Support:            yes
NetCDF-2 API:               yes
NetCDF-4 API:               yes
CDF5 Support:               yes
Parallel IO:                yes
NetCDF4 Parallel IO:        yes
PnetCDF Parallel IO:        no
SZIP Write Support:         no
Zstandard Support:          yes (HDF5_PLUGIN_PATH: /usr/local/hdf5/lib/plugin)
Quantize:                   yes
```

UIFCW 2023
A UFS Collaboration Powered by EPIC

All netCDF compression work is fully unit tested.

As with the rest of the netCDF code base, the compression features are fully documented and tested, for C, F77, and F90.

They would not have been accepted otherwise.

```
C     Turn on zstandard compression if available, zlib otherwise.
#ifdef ENABLE_ZSTD
         retval = nf_def_var_zstandard(ncid, varid(x), ZSTD_LEVEL)
         if (retval .ne. nf_noerr) then
            if (retval .eq. nf_enofilter) then
               print *, 'Zstandard filter not found.'
               print *, 'Set HDF5_PLUGIN_PATH and try again.'
            else
               print *, nf_strerror(retval)
            endif
            stop 5
         endif
#else
         retval = nf_def_var_deflate(ncid, varid(x), 0, 1, 1)
         if (retval .ne. nf_noerr) stop 6
#endif

      end do

C     Write some data (which automatically calls nf_enddef).
      start(1) = 1
      count(1) = DIM_LEN_5
      retval = nf_put_vara_real(ncid, varid(1), start, count,
     $     real_data)
      if (retval .ne. 0) stop 7
      retval = nf_put_vara_double(ncid, varid(2), start, count,
     $     double_data)
      if (retval .ne. 0) stop 8

C     Check it out.
      retval = check_file(ncid, var_name, var_type, dim_name)
      if (retval .ne. 0) stop 9
```