

ESSI 2021 Leptoukh Lecture

What Geoscientists Want: Short and Sweet Commands with Eco-friendly Data

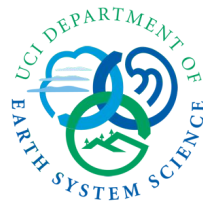


Dr. Greg Leptoukh

Data management and analysis, large-scale computation and modeling, and hardware and software infrastructure profoundly affect the research capabilities of all AGU disciplines. The Earth and Space Science Informatics Focus Group established the **Leptoukh Lecture** in 2012 to recognize advances in these fields and their contributions to Earth and space science. Named in honor of the late Dr. Greg Leptoukh, a pioneer in satellite data quality and provenance, the Leptoukh Lecture aims to identify and support achievements in the computational and data sciences.

Charlie Zender

Departments of Earth System Science and Computer Science
University of California, Irvine



Three Acts with the Theme of Interoperability

- I. Role of Self-Describing Data Formats
- II. Promise of Domain Specific Languages
- III. Viability of Greener Computing and Compression





Act I:



Self-Describing Data Formats

Once Upon a Time in Boulder...

Climate data was stored in idiosyncratic model/dataset-specific formats understood by few mortals, accessible through monolithic tools...

```
1  #!/bin/csh
2  #QSUB -q econ -lt 30 -lT 33 -lm 2Mw -lM 2Mw -eo
3  cd $TMPDIR
4
5  cat >! parms << 'END'
6  C
7  C plot.sh read temperature from a history tape and plot
8  C
9  TITLEA = 'Plot.sh: 850,500,200mb T from the CCM2 Control Run'
10  TAPESA = '/CSM/ccm2/414/hist/h0002'
11  DAYSA = 11.,12.,13.,14.,15.
12  FIELD1 = 'T'
13  PRESSLE = 850.,500.,200.
14  HPROJ = 'RECT'
15  HPCINT = 'T',850.,5., 'T',500.,10., 'T',200.,0.
16  DPLTMF = 'MP'
17  ENDOFDATA -----
18  'END'
19
20  /ccm/proc/Processor
21  exit
```

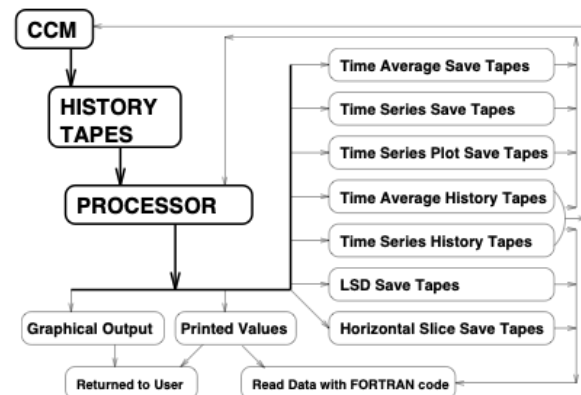


NCAR TECHNICAL NOTE

December 1992

Introduction to the UNICOS CCM Processor

LAWRENCE E. BUJA



CLIMATE AND GLOBAL DYNAMICS DIVISION

NATIONAL CENTER FOR ATMOSPHERIC RESEARCH
BOULDER, COLORADO

...Analysis tools were powerful, but...

Climate data were stored in a proprietary model/dataset-specific history tape format understood by few mortals, accessible through the Fortran-based CCM Processor...

```
1  #!/bin/csh
2  #QSUB    -q econ    -lt 30    -lT 33    -lm 2Mw    -lM 2Mw    -eo
3  cd $TMPDIR
4
5  cat >! parms << 'END'
6  C
7  C plot.sh    read temperature from a his
8  C
9  TITLEA = 'Plot.sh: 850,500,200mb T from CCM Run'
10 TAPESA = '/CSM/ccm2/414/hist/h0002'
11 DAYS = 11.,12.,13.,14.,15.
12 FIELDS = 'T'
13 PRESSLE = 850.,500.,200.
14 HPROJ = 'RECT'
15 HPCINT = 'T',850.,5., 'T',500.,10., 'T',200.,0.
16 DPLTMF = 'MP'
17 ENDOFDATA -----
18 'END'
19
20 /ccm/proc/Processor
21 exit
```

Extensible file
lists, coordinate
subsetting

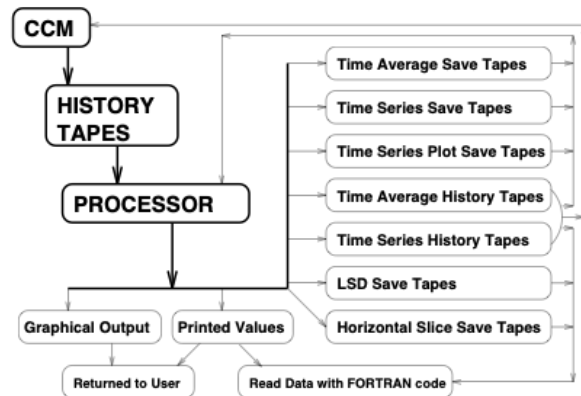
Random access to variables



NCAR/TN-383+IA
NCAR TECHNICAL NOTE

December 1992

OS-specific
Introduction to the UNICOS
CCM Processor
Dataset-specific
LAWRENCE E. BUJA

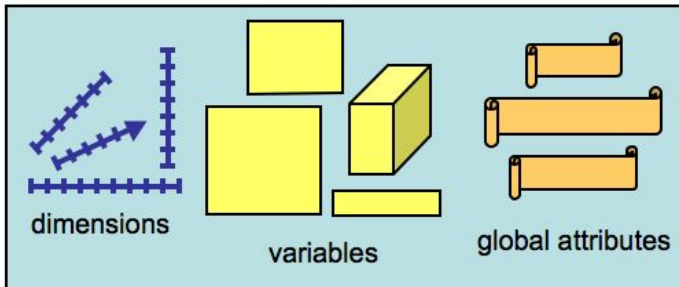


CLIMATE AND GLOBAL DYNAMICS DIVISION

NATIONAL CENTER FOR ATMOSPHERIC RESEARCH
BOULDER, COLORADO

...When a legendary (well, to ESSi) team...

Unidata designed netCDF
API/format to store
datasets that fit the
Common Data Model:

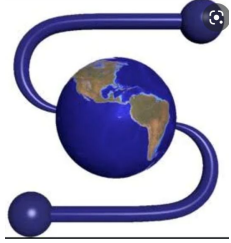


netCDF API enables
clients to interrogate
these "self-describing"
files.



Harnett and Raphael, *The School of netCDF*, 1509-2021

...Other teams had similar aims...

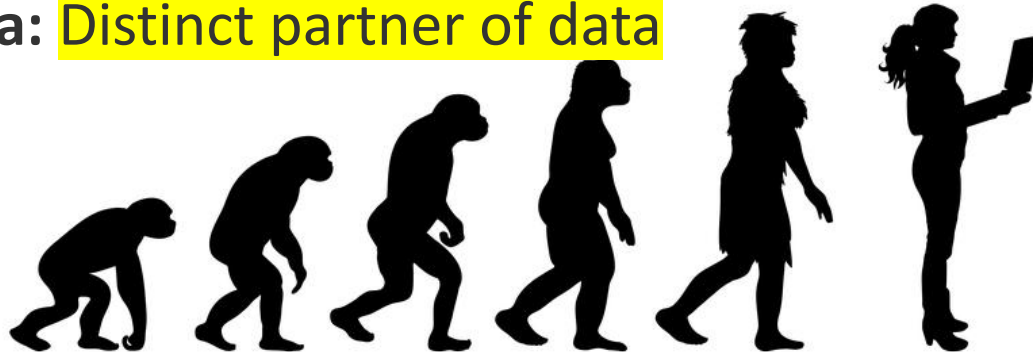


NCSA developed the [Hierarchical Data Format](#) (HDF) and API, and a netCDF API for HDF. HDF is also self-describing. NASA chose HDF over netCDF for EOSDIS. In ~2007 the netCDF API married the HDF5 format, became netCDF4.



netCDF/HDF Make Data SPASSAM

- **Self-Describing:** *Queries reveal file contents*
- **Portable:** Allow different integer, character, FP formats
- **Archivable:** Guaranteed backwards compatibility
- **Scalable:** Efficiently access **subsets/hyperslabs** of datasets
- **Sharable:** SWMR access to the same file
- **Appendable:** No longer copy/redefine entire dataset
- **Metadata:** **Distinct partner of data**



Dataset Intercomparison Became Easier!



Ease of use, power of netCDF/HDF APIs led to rapid adoption:

Tools: GrADS, IDL, NCL, **NCO**, ...

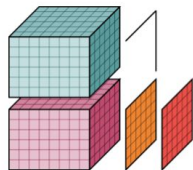
Producers: GISS, MPI, NASA, NCAR, UKMO

Middleware: DODS-DAP

Standards: COARDS, **CF**



CMIP6



xarray



Iris



What are the netCDF Operators (NCO)?

- A) A modular set of tools for scriptable data analysis
- B) A means to reduce future graduate suffering
- C) A standard comparator for your newer, faster, better method
- D) A way to treat files like atomic data
- E) All of the above

With the time netCDF saves us,
we can juggle more science!



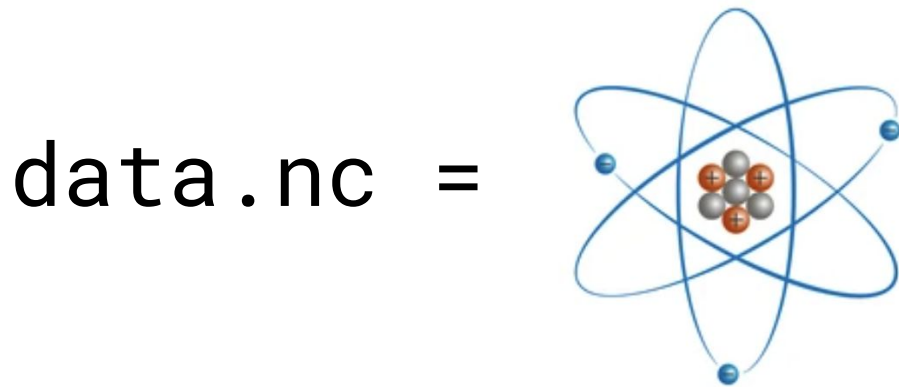
What are the netCDF Operators (NCO)?

- [ncap2](#) Arithmetic Processor
- [ncatted](#) ATtribute EDitor
- [ncbo](#) Binary Operator (differencer)
- [ncclimo](#) CLIMatOlogy Generator
- [nces](#) Ensemble Statistics
- [ncecat](#) Ensemble conCATenator
- [ncflint](#) FiLe INTerpolator
- [ncks](#) Kitchen Sink
- [ncpdq](#) Permute Dimensions Quickly
- [ncra](#) Record Averager
- [ncrcat](#) Record conCATenator
- [ncremap](#) REMAPer
- [ncrename](#) RENAMEer
- [ncwa](#) Weighted Averager



Treat Datasets as Fundamental Units of Data

Every NCO operator can treat a dataset as a unit of data, independent of the numbers, types, and dimensionality of its variables



$C = A - B$: ncbo a.nc b.nc c.nc

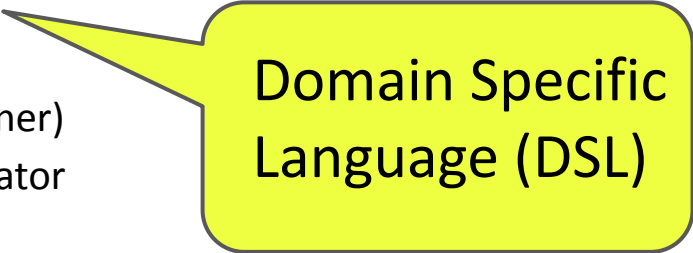
Act II:

Domain-Specific

Languages

What are the netCDF Operators (NCO)?

- [ncap2](#) Arithmetic Processor
- [ncatted](#) ATtribute EDitor
- [ncbo](#) Binary Operator (differencer)
- [ncclimo](#) CLIMatOlogy Generator
- [nces](#) Ensemble Statistics
- [ncecat](#) Ensemble conCATenator
- [ncflint](#) FiLe INTerpolator
- [ncks](#) Kitchen Sink
- [ncpdq](#) Permute Dimensions Quickly
- [ncra](#) Record Averager
- [ncrcat](#) Record conCATenator
- [ncremap](#) REMAPer
- [ncrename](#) RENAMER
- [ncwa](#) Weighted Averager

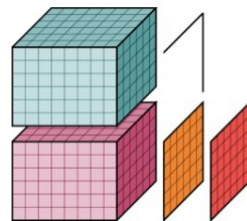
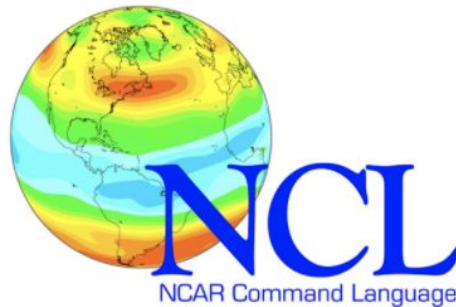


Domain Specific
Language (DSL)

ncap2 is/as a Domain Specific Language

NCO's `ncap2` is ANTLR-based lexer/parser, uses netCDF API for I/O:

- C-like array-oriented language: `d=a+b(i,j)+c(:, :, 3)`
- Loops, conditionals: `if(idx==sz) print("Finish")`
- Broadcast arrays to conform: `d3D=a1D+b2D+c3D`
- Common methods: `foo=bar.avg($time, $lon)`
- Special functions: `gsl_sf_legendre_P1()`
- Autoload input, save output: `wind=U^2+V^2`
- Avoid disk, use RAM: `for(*idx=0;idx<sz;idx++)`
- Propagate metadata



xarray

Short, Sweet Commands

Construct pressure field **p** in hybrid vertical coordinate system from **P0**, **PS(time, lat, lon)**, **hyam(lev)**, and **hybm(lev)**:

```
ncap2 -s 'p[time,lat,lon,lev]=P0*hyam+PS*hybm'
```

Broadcast RHS to shape of LHS, propagate attributes of first RHS variable.
Output **p** has same attributes as **P0**!

High-level Workflows can be Short and Sweet

Create climatologies, or split and regrid CMIP6 timeseries:

```
ncclimo --case=foo --start=2001 --end=2020  
--in_drc=raw --out_drc=clm --map=map.nc # Climo
```

```
ls *cam*h1*.nc | ncclimo -v T,Q,FSNT,FLNT  
--start=1850 --end=2014 --map=map.nc # Splitter
```

```
ls *.nc | ncremap --map=map.nc # Regrid en-masse
```

```
ls *.nc | ncremap --dst=180x360 # Regrid swaths
```

NCO Workflow Scripts
(ncremap, ncclimo)

ESMF_RegridWeightGen,
TempestRemap

MPI

PyNCO

NCO Ecosystem

NCO Binary Executables
(ncks, ncra, ncatted, nces, ...)

ncap2 Binary (DSL)

CCR Libraries (BitGroom,
Bzip2, Zstd, ...)

NCO C Library

(OpenMP)

NCO C++ Library & ANTLR

netCDF C Library

Rx

UDUnits

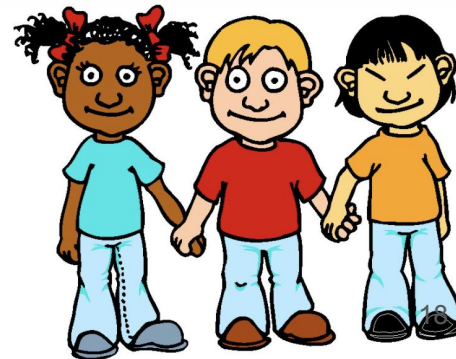
GSL

HDF5

HDF4

OPeNDAP

BLAS



Greg Leptoukh



On designing Giovanni v4
c.2008: “...if we used NetCDF
as our internal format, the
entire library of NCO (NetCDF
command operators) would
be available to us, as well as
the usual GrADS and IDL...”



Issues **Solved** by Conventions + DSLs

- Units conversion **UDUnits**
- Attach geophysical meaning **CF standard_name**
- Stitch datasets together **CFA, mfdataset(), DAP/Hyrax**
- Associate variables with formulae **CF formula_terms**
- Grid/projection **ESMF, UGRID, CF grid_mapping, WKT-CRS**

Powered By 



unidata UDUnits

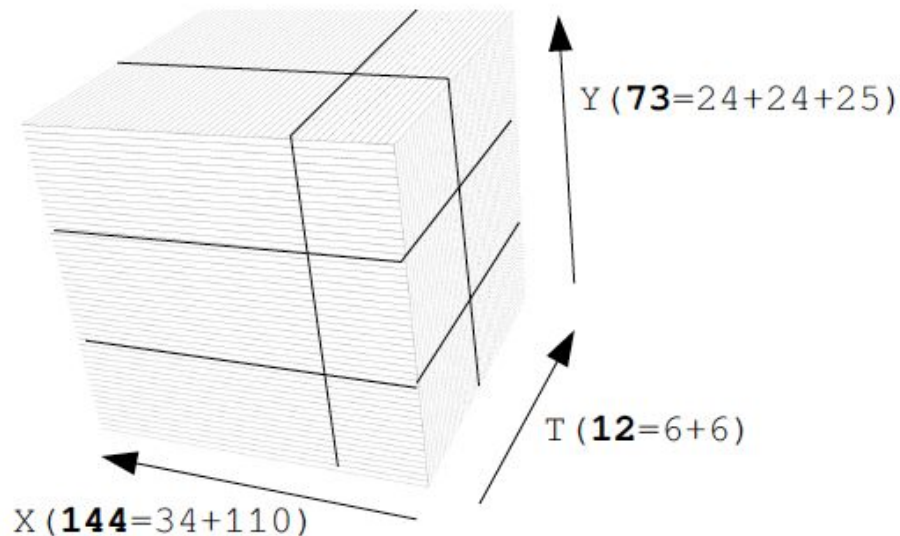
ESMF



Open
Geospatial
Consortium

Climate and Forecast Aggregation (CFA) Conventions

```
aggregation_location = 6, 6, -,  
                      24, 24, 25,  
                      34, 110, - ;
```



```
dimensions:  
  time = 12 ;  
  latitude = 73 ;  
  longitude = 144 ;  
  // Fragment dimensions  
  f_time = 2 ;  
  f_latitude = 1 ;  
  f_longitude = 1 ;  
  i = 3 ;  
  j = 2 ;  
variables:  
  double temp ;  
  temp:standard_name = "surface_temperature" ;  
  temp:units = "K" ;  
  temp:cell_methods = "time: mean" ;  
  temp:aggregated_dimensions = "time latitude longitude" ;  
  temp:aggregated_data = "location: aggregation_location  
                          file: aggregation_file  
                          format: aggregation_format  
                          address: aggregation_address" ;  
  
  double time(time) ;  
  time:units = "days since 2001-01-01" ;  
  double latitude(latitude) ;  
  latitude:units = "degrees_north" ;  
  double longitude(longitude) ;  
  longitude:units = "degrees_east" ;  
  // Aggregation definition variables  
  int aggregation_location(i, j) ;  
  string aggregation_file(f_time, f_latitude, f_longitude) ;  
  string aggregation_format ;  
  string aggregation_address(f_time, f_latitude, f_longitude) ;  
  
// global attributes:  
:Conventions = "CF-1.9 CFA-0.6" ;  
data:  
  time = 0, 31, 59, 90, 120, 151, 181, 212, 243, 273, 304, 334 ;  
  temp = - ;  
  aggregation_location = 6, 6,  
                        73, -,  
                        144, - ;  
  aggregation_file = "January-June.nc", "July-December.nc" 21  
  aggregation_format = "nc" ;  
  aggregation_address = "temp1", "tas" ;
```

Issues **Solvable** by Conventions + DSLs + AI/NL

"We should be able to treat climate datasets as atomic objects as well — namely, divorce the climate data from the number of time steps per file and which files contain which variables." — Paul Ullrich

Interpret intent: `this(500hPa, JJA), MOC(2000-2010, NAT1),`
"plot monthly sensitivity of surface albedo to surface
air temperature in all gridcells with snow and ice"



Natural Language
Processing with
Python

PANGEO

Act III:

Green Computing and Storage

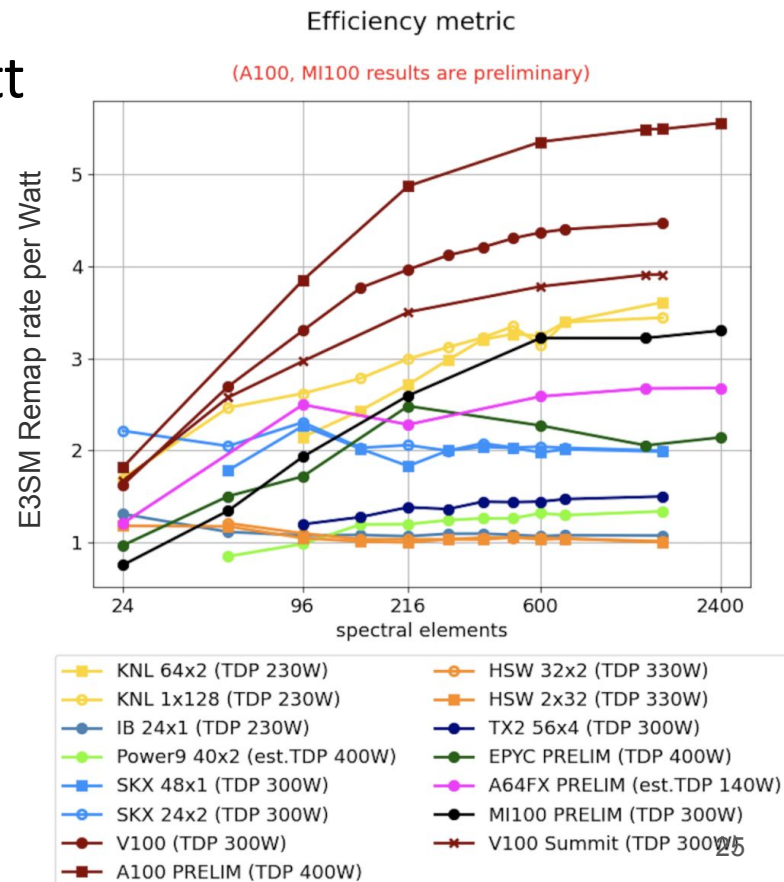
Why Reduce Computing Demands, Dataset Size?

1. Datacenters consume ~1-3% of global electrical power (~1% of global CO₂ emissions)
2. Storage accounts for ~40% of datacenter power, **emissions**
3. *Geoscience* computing is power-hungry, archives mostly false precision



ESM Energy Efficiency Tradeoffs: CPU vs. GPU

- Target Simulated Years per Day per Watt
- Optimize with hardware-agnostic APIs



Energy Efficiency of Programming Languages

- Dominated by execution time
- Total memory (DRAM) accounts for ~10%
- Compiled languages most efficient, C wins!

SLE'17, October 23–24, 2017, Vancouver, Canada

R. Pereira et. al.

Table 5. Pareto optimal sets for different combination of objectives.

Time & Memory	Energy & Time	Energy & Memory	Energy & Time & Memory
C • Pascal • Go Rust • C++ • Fortran Ada Java • Chapel • Lisp • Ocaml Haskell • C# Swift • PHP F# • Racket • Hack • Python JavaScript • Ruby Dart • TypeScript • Erlang	C Rust C++ Ada Java Pascal • Chapel Lisp • Ocaml • Go Fortran • Haskell • C# Swift	C • Pascal Rust • C++ • Fortran • Go Ada Java • Chapel • Lisp OCaml • Swift • Haskell C# • PHP Dart • F# • Racket • Hack • Python JavaScript • Ruby TypeScript	C • Pascal • Go Rust • C++ • Fortran Ada Java • Chapel • Lisp • Ocaml Swift • Haskell • C# Dart • F# • Racket • Hack • PHP JavaScript • Ruby • Python TypeScript • Erlang Lua • JRuby • Perl

Table 4: Normalized global results for Energy, Time, and Memory

Energy (J)		Time (ms)		Mb	
(c) C	1.00	(c) C	1.00	(c) Pascal	1.00
(c) Rust	1.03	(c) Rust	1.04	(c) Go	1.05
(c) C++	1.34	(c) C++	1.56	(c) C	1.17
(c) Ada	1.70	(c) Ada	1.85	(c) Fortran	1.24
(v) Java	1.98	(v) Java	1.89	(c) C++	1.34
(c) Pascal	2.14	(c) Chapel	2.14	(c) Ada	1.47
(c) Chapel	2.18	(c) Go	2.83	(c) Rust	1.54
(v) Lisp	2.27	(c) Pascal	3.02	(v) Lisp	1.92
(c) Ocaml	2.40	(c) Ocaml	3.09	(c) Haskell	2.45
(c) Fortran	2.52	(v) C#	3.14	(i) PHP	2.57
(c) Swift	2.79	(v) Lisp	3.40	(c) Swift	2.71
(c) Haskell	3.10	(c) Haskell	3.55	(i) Python	2.80
(v) C#	3.14	(c) Swift	4.20	(c) Ocaml	2.82
(c) Go	3.23	(c) Fortran	4.20	(v) C#	2.85
(i) Dart	3.83	(v) F#	6.30	(i) Hack	3.34
(v) F#	4.13	(i) JavaScript	6.52	(v) Racket	3.52
(i) JavaScript	4.45	(i) Dart	6.67	(i) Ruby	3.97
(v) Racket	7.91	(v) Racket	11.27	(c) Chapel	4.00
(i) TypeScript	21.50	(i) Hack	26.99	(v) F#	4.25
(i) Hack	24.02	(i) PHP	27.64	(i) JavaScript	4.59
(i) PHP	29.30	(v) Erlang	36.71	(i) TypeScript	4.69
(v) Erlang	42.23	(i) Jruby	43.44	(v) Java	6.01
(i) Lua	45.98	(i) TypeScript	46.20	(i) Perl	6.62
(i) Jruby	46.54	(i) Ruby	59.34	(i) Lua	6.72
(i) Ruby	69.91	(i) Perl	65.79	(v) Erlang	7.20
(i) Python	75.88	(i) Python	71.90	(i) Dart	8.64
(i) Perl	79.58	(i) Lua	82.91	(i) Jruby	19.84

Energy Efficiency of Programming Languages

$$\begin{bmatrix} Op \\ BL \end{bmatrix}^\top \times \begin{bmatrix} en \\ AS \end{bmatrix}$$



How much
data do
geoscientists
store?

> 3700 PB

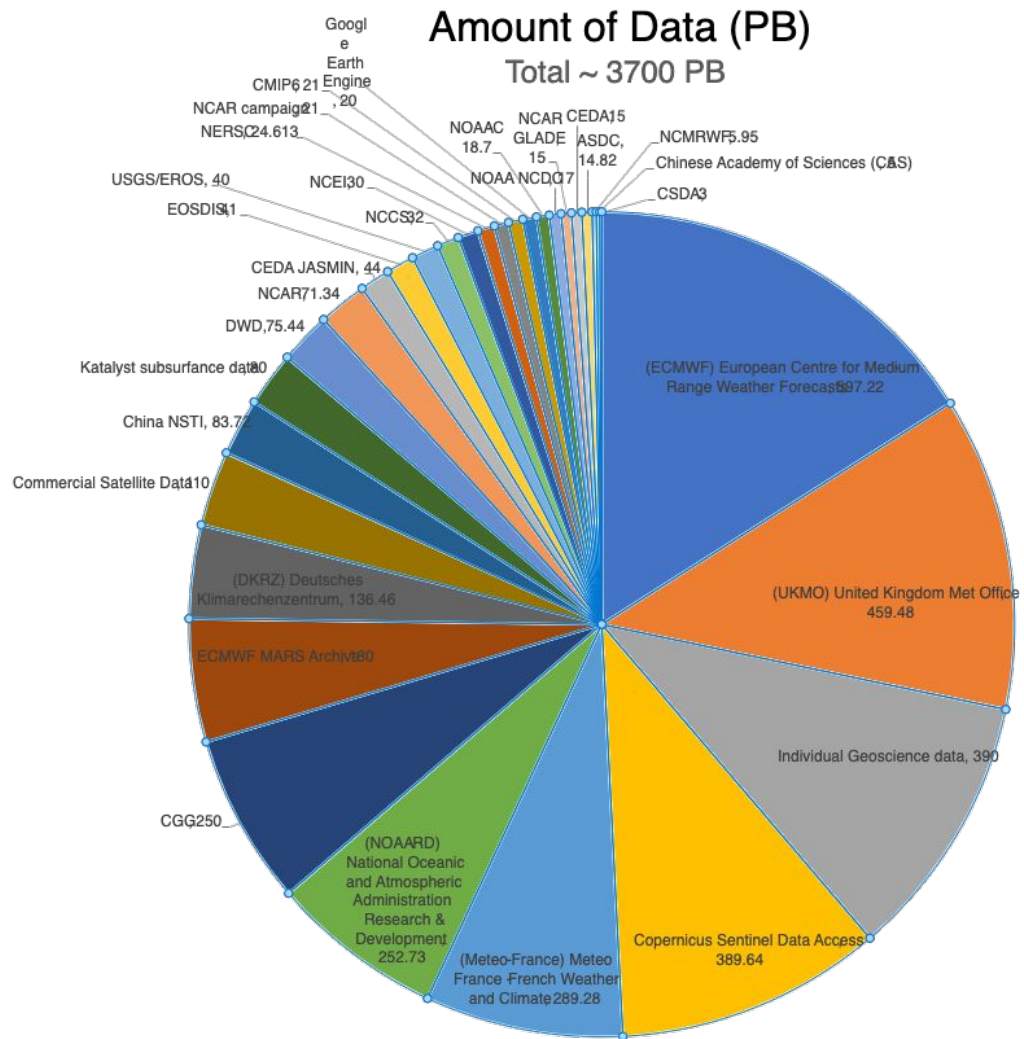
Assumes:

Data Centers ~3300 PB

130k AGU members @

3 TB on PCs ~400 PB

Trampush and Zender (2021), unpublished



CO2 Emissions from Geoscience Data Storage

> 10000 tCO₂e/yr

Assumes:

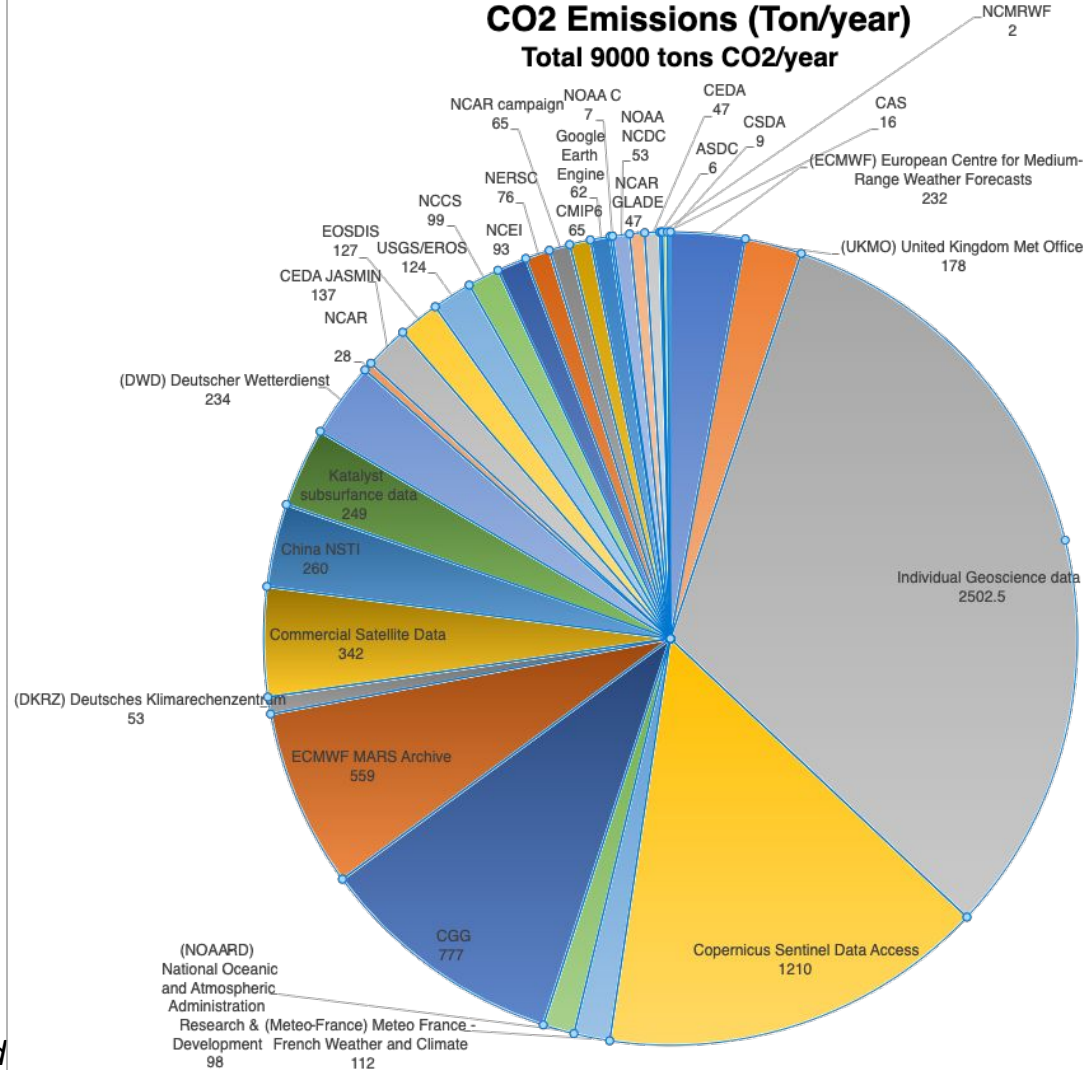
PC ~ 200 kWhr/TB/yr,

HDD ~ 7 kWhr/TB/yr,

Tape ~ 1 kWhr/TB/yr,

0.45 tCO₂eq/MWhr

Trampush and Zender (2021), unpublished



Barriers to Green Computing & Storage

- Accessing modern compressors (codecs) difficult
- Compression requires extra post-processing steps
- Traditional compression too slow
- Lossy compression discernibly distorts data
- How to identify false precision in model data?
- IEEE 64-bit, 32-bit formats are inflexible
- Culture of science encourages keeping false precision



Compression Capabilities of HDF5/netCDF4/Python

- HDF5 data chunks pass through "filters" (codecs)
- HDF5/netCDF4 APIs natively support [checksum](#), [shuffle](#), [DEFLATE](#) (aka [zlib](#)) filters, and [gzip](#) (if built-in to HDF5)
- Python numcodecs: [Zlib](#), [BZ2](#), [LZMA](#), [ZFPY](#), [Blosc](#)
- *Interoperable, modern codec support has lagged in netCDF...*

Until Recently! netCDF 4.8.2-develop supports new compression features: [BitGroom](#), [GranularBG PR](#), [Zarr/numcodecs](#), [CCR](#)



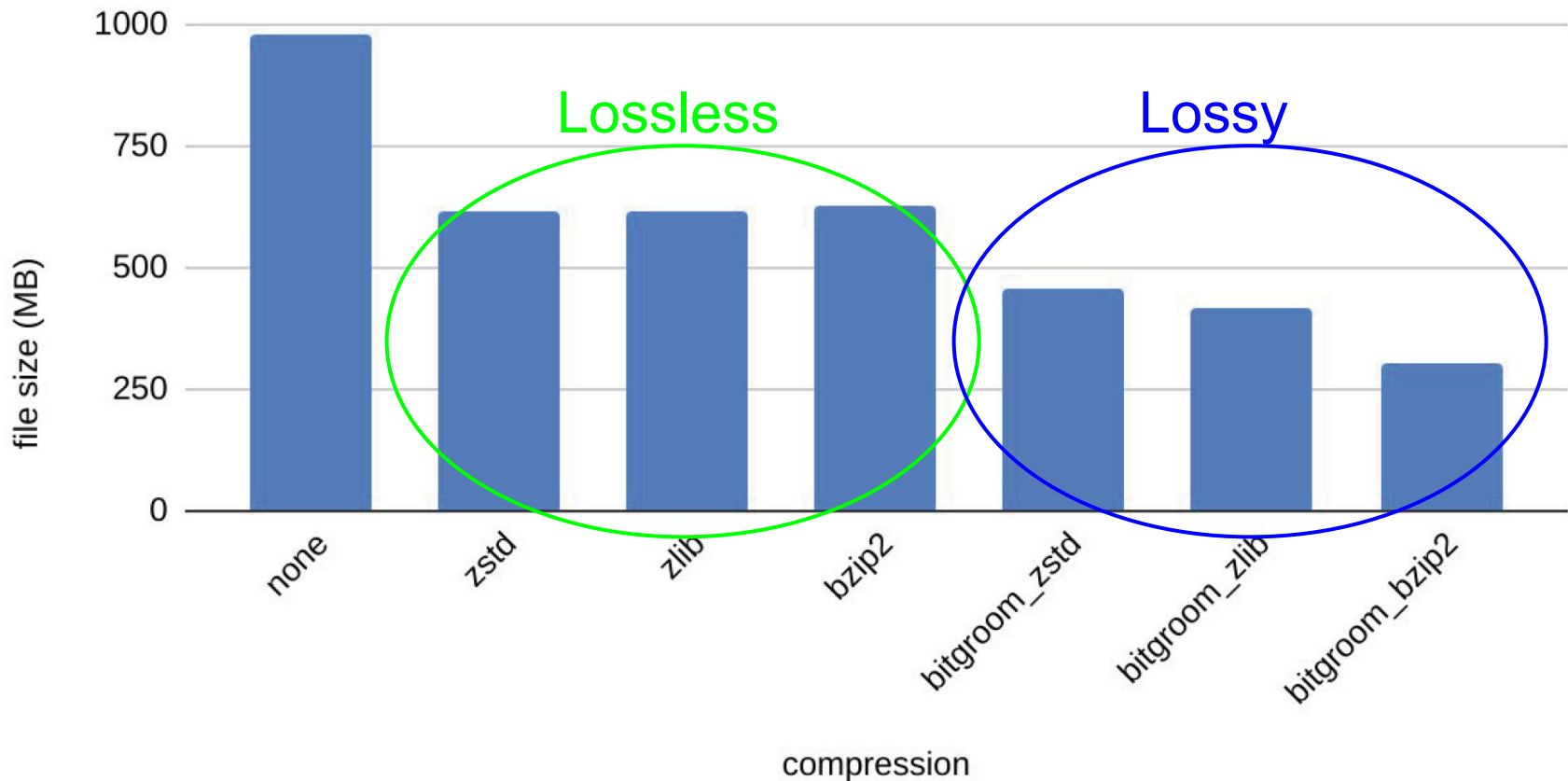
Community Codec Repository: Interoperable, End-to-End Compression for netCDF

- CCR builds filters, [ccr](#) library, netCDFish API
- Includes [BitGroom](#), [GranularBG](#), [BZ2](#), [Zstd](#)
- Lossless+lossy I/O for C/Fortran:
`nc_def_var_bitgroom(ncid,varid,level)`
`nc_def_var_zstandard(ncid,varid,level)`
- Codec I/O is transparent to users
- *Efficient compression in production phase!*



EAMv1 Climate Data 2D/3D Vars

Compression level = 1, Bitgroom nsd = 3



Lossy Compression (e.g., BitGroom) as Pre-Filter

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

*NSD = Number of Significant Digits³⁴

EAMv1 Climate 2D/3D Vars Read/Write/Re-Read Times

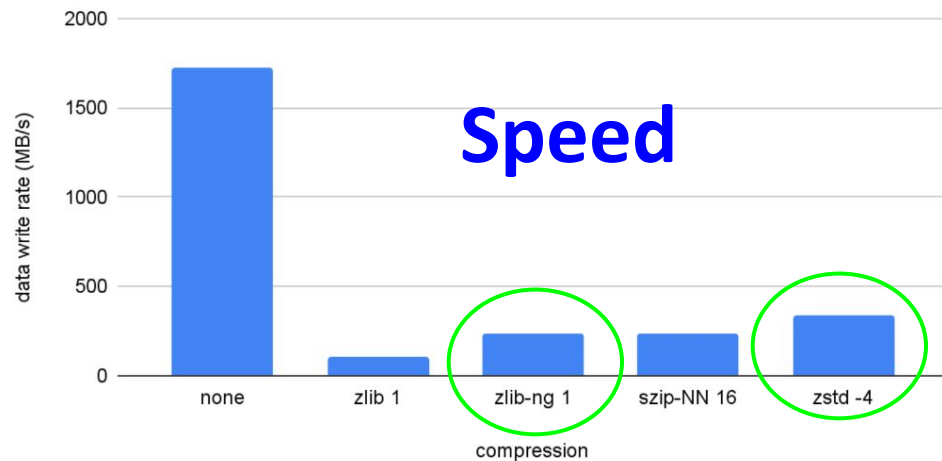
Compression level = 1, Bitgroom nsd = 3



Lossless Compression

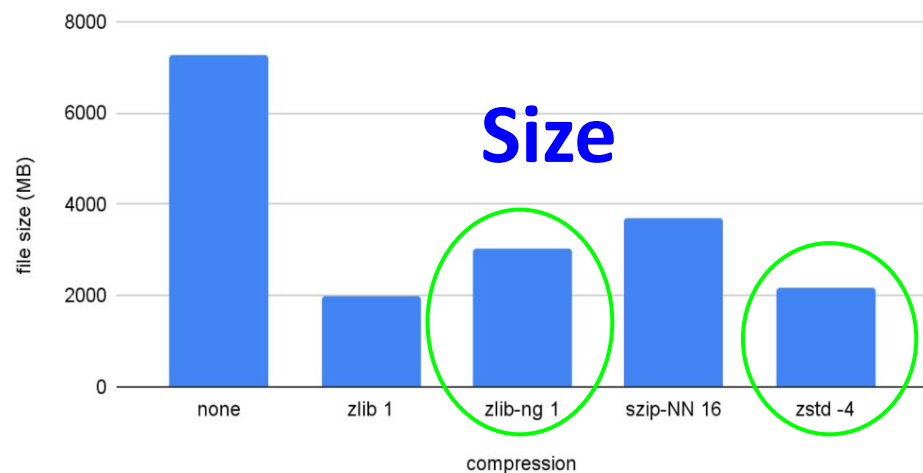
Write Rate for Different Lossless Compression Methods

Linux workstation, netcdf-c-4.8.1, HDF5-1.12., ccr-1.2.0



File Size for Different Lossless Compression Methods

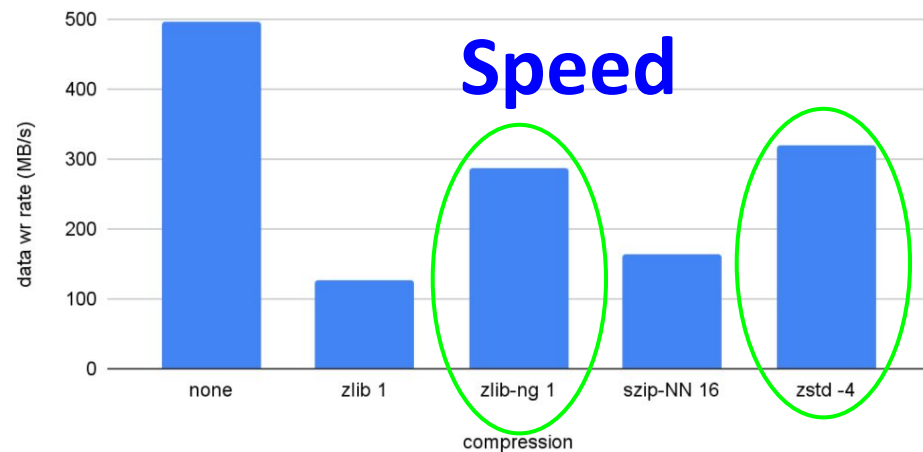
Linux workstation, netcdf-c-4.8.1, HDF5-1.12., ccr-1.2.0



Lossy Compression

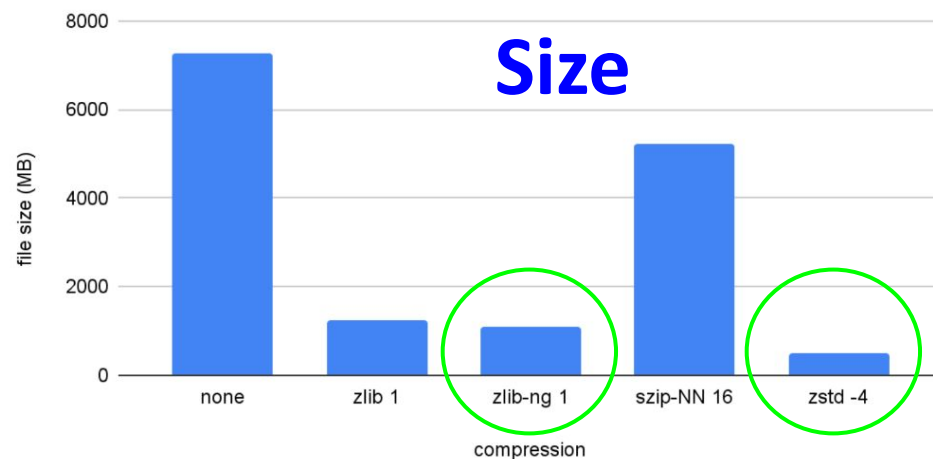
Write Rate for Different Lossy Compression Methods

4 significant digits, Linux workstation, netcdf-c-4.8.1, HDF5-1.12., ccr-1.2.0



Files Sizes for Different Lossy Compression Methods

4 significant digits, Linux workstation, netcdf-c-4.8.1, HDF5-1.12., ccr-1.2.0

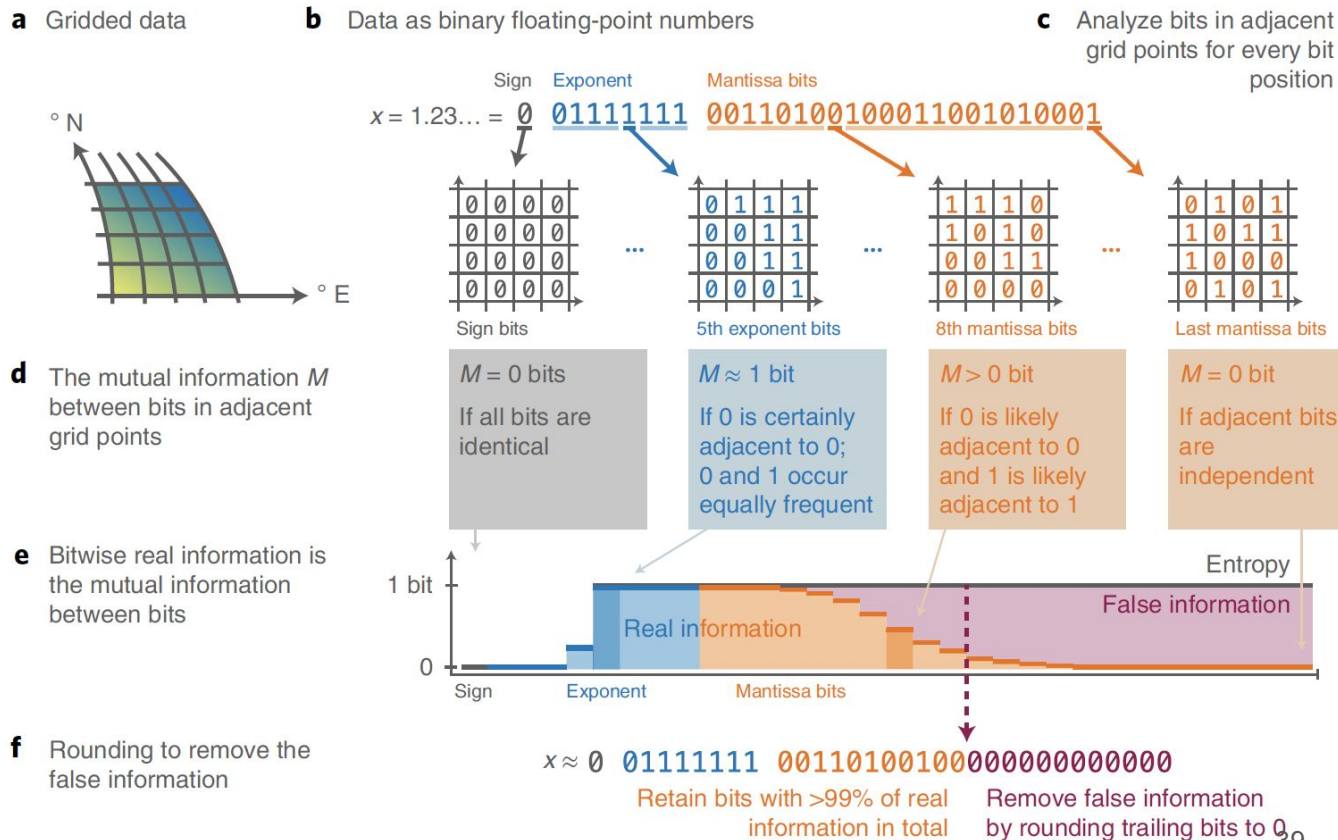


Modern Codecs More Competitive

- Zlib-NG, Zstandard I/O speed 2-5x faster than Zlib
- Zlib-NG plug-compatible with Zlib thus netCDF/HDF!
- Zstandard+Bitgroom pre-filter CR $\gtrsim 3x$ ($NSD \leq 4$)
- Zstandard+GranularBG pre-filter CR $\gtrsim 4x$ ($NSD \leq 4$)
- Inflection point for adoption of lossy compression...

Compressing atmospheric data into its real information content

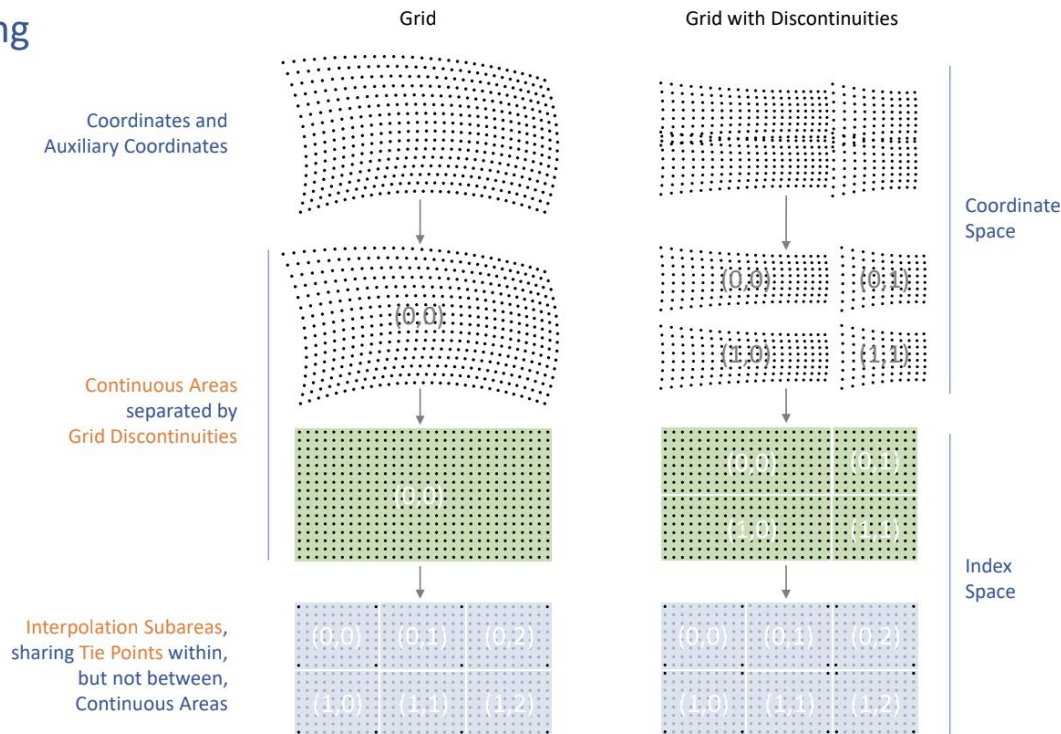
1. Determine # bits to preserve e.g. 99% *Information content* (IC) in adjacent cells
2. Round the rest (false precision)
3. $CR \approx 8x$ for 99% IC



Lossy Compression by Coordinate Subsampling

Subsampling Process

1. Store grid info. at *tie points*, reconstitute via interpolation
2. Preserves desired accuracy level.
3. Elegant for one-off grids like RS. CR ~ 40x for VIIRS



Frontiers in Green Computing & Storage

- ~~Accessing modern codecs difficult~~ numcodecs, CCR, netCDF
- ~~Compression requires extra post-processing steps~~ CCR, netCDF
- ~~Traditional compression too slow~~ Modern codecs
- ~~Lossy compression discernibly distorts data~~ Pick NSD
- How to identify false precision in model data? Info. Content
- IEEE 64-bit, 32-bit formats are inflexible 16bit Posits?
- Culture of science encourages keeping false precision ...

Ethical and Responsible Research (ER2)

PROGRAM SOLICITATION
NSF 22-526

Join our (hybrid vPICO) Session at EGU22

[ESSI2.7](#): *Lossy and Lossless Compression for Greener Geoscientific Computing and Data Storage*

Conveners: Charlie Zender, Ed Hartnett, Bryan Lawrence, V. Balaji

Confirmed Invited Speaker: Milan Klöwer

Abstract Deadline: Jan. 12, 2022

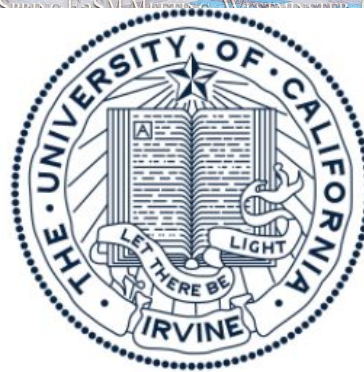
Conclusions

- Self-describing formats (netCDF/HDF) enable tools and DSLs to treat complex datasets as atomic data. Untapped potential!
- End-to-End, model-to-archive compression now possible
- Modern compression nearly speed-competitive (within 2-5x)
- Lossy+Lossless can reduce (by $\geq 3x$) scientific data storage (**cost, power, GHGs**) and retain all scientifically significant content

Green Computing has multiple benefits, few regrets!

Appreciation

- Family: Shari, Olivia, Jinx, Ashley, Joel
- Advisors+Role Models
- ESS Department at UCI, Group Members
- Collaborators, Colleagues
- OpenSource Community
- Funding and Support



Resources

- Delaunay et al. (2019), Evaluation of lossless and lossy algorithms for the compression of scientific datasets in netCDF-4 or HDF5 files, *GMD*, [DOI](#).
- Hartnett et al. (2021), Quantization and Next-Generation Zlib Compression for Fully Backward Compatible, Faster, and More Effective Data Compression in netCDF Files, *AGU ESSI*, [DOI](#).
- Hartnett and Zender (2021), Additional netCDF Compression Options with CCR, *AMS EIPT*, [DOI](#).
- Hassell et al. (2021), Climate and Forecast Aggregation (CFA) Conventions, NCAS-CMS, [URL](#).
- Klöwer et al. (2020), Number formats, error mitigation and scope for 16-bit arithmetics in weather and climate modelling analysed with a shallow water model, *JAMES*, [DOI](#).
- Klöwer et al. (2021), Compressing atmospheric data into its real information content, *NCS*, [DOI](#).
- Kouznetsov (2021), A note on precision-preserving compression of scientific data, *GMD*, [DOI](#).
- Pereira et al. (2017), Energy Efficiency across Languages: How Do Energy, Time, and Memory Relate? *ACM CSLE*, [DOI](#).
- Pereira et al. (2021), Ranking Programming Languages by Energy Efficiency, *SCP*, [DOI](#).
- Soerensen et al. (2021), Lossy Compression by Coordinate Subsampling, CF, [URL](#).
- Zender (2016), Bit Grooming: Statistically accurate precision-preserving quantization with compression, *GMD*, [DOI](#).
- Zender and Hartnett (2020), Community Codec Repository (CCR), [URL](#).



That's all Folks!