# Making earth science data more accessible: experience with chunking and compression

Russ Rew

January 2013

93rd Annual AMS Meeting

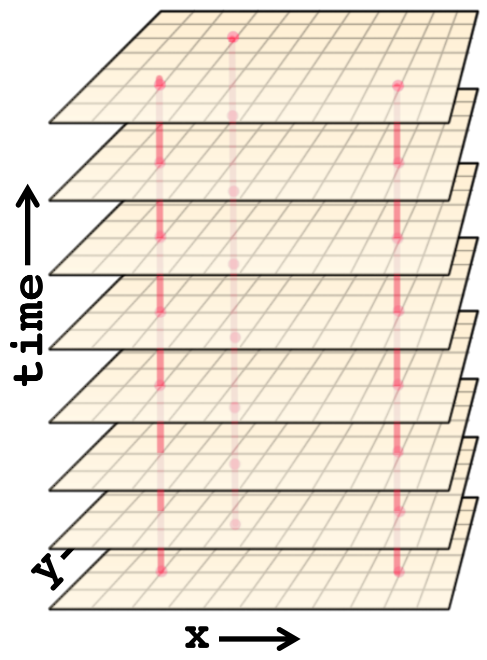
Austin, Texas

# What’s the Problem?

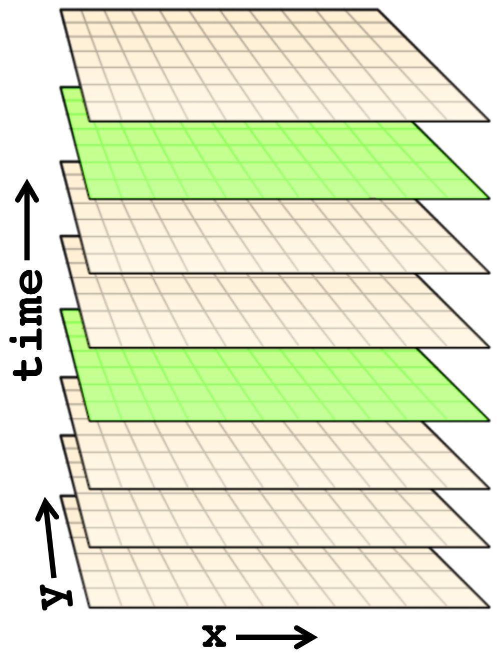
Can large multidimensional datasets be organized for fast *and* flexible access?

***Without multiple versions of the data***, **what’s the best you can do?**

Time range access



Spatial access



| Conventional storage layout | Time range access | Spatial access |
|---|---|---|
| **Time varying fastest** | Fast | Slow |
| **Time varying slowest** | Slow | Fast |

unidata

Goal is to solve what looks like a little problem that becomes more serious with …

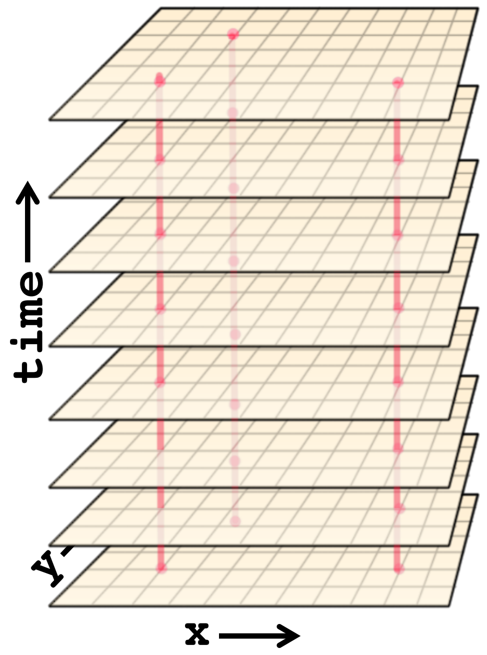
# Big Data

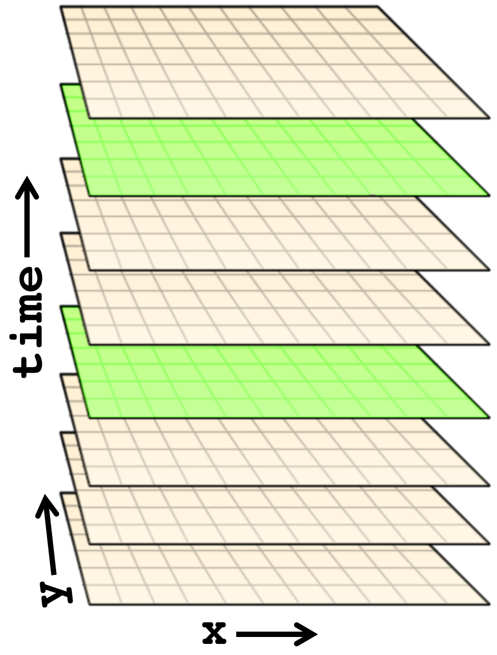
# Real data, conventional storage

**`NCEP North American Regional Reanalysis`**
**`float 200mb_TMP(time=98128, y=277, x=349)`**

9.5 billion values
38 GB of data
8.5 GB compressed

Contiguous storage with
- ❑ time varying fastest
- ❑ time varying slowest

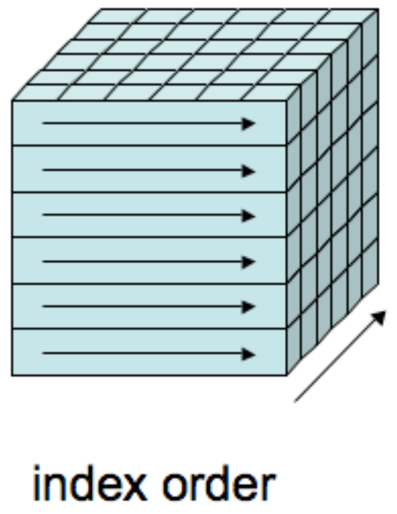


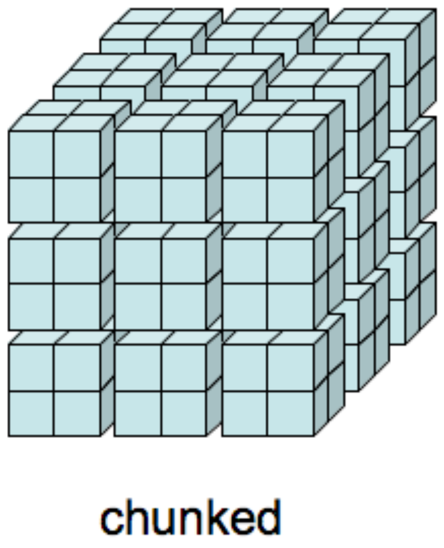


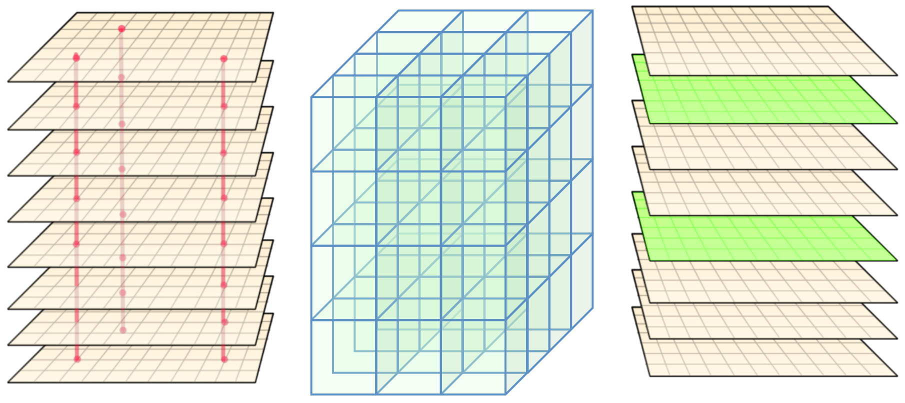
| | **Time range access** | **Spatial access** |
|---|---|---|
| **Time varying fastest** | 0.013 sec | 180 sec |
| **Time varying slowest** | 200 sec | 0.012 sec |

*Single file access time averaged over 100 independent queries, after clearing disk caches. 7200 rpm disk.

# Chunking







- Storing data in "chunks" along each dimension in a multidimensional variable makes access along any dimension similar
- Slows down fast accesses, but speeds up slow accesses
- How to choose shape and size of chunks for acceptable performance?

## Benefits of chunking

- Large performance gains are possible with good choice of chunk shapes and sizes
- Benefits of chunking for compression are under-appreciated
- Chunking supports efficiently extending data along multiple dimensions
- *So why is use of chunking not widespread?*
  - Software to rechunk big datasets is available, but defaults work poorly for some common use cases
  - Specific guidance for how to set chunk shapes is lacking

# Importance of chunk shapes

**Example:** `float 200mb_TMP(time=98128, y=277, x=349)`

| Storage layout, chunk shapes | Read time range (seconds) | Read horizontal slice (seconds) | Performance bias: (slowest / fastest) |
|---|---|---|---|
| **Contiguous, for time range** | 0.013 | 180 | 14,000 |
| **Contiguous, for spatial slices** | 200 | 0.012 | 17,000 |
| **Record chunks, 1 x 277 x 349** | 690 | 0.007 | 98,000 |
| **Default chunks, 4673 x 12 x 16** | 1.4 | 34 | 24 |
| **36 KB chunks, 92 x 9 x 11** | 2.4 | 1.7 | 1.4 |
| **8 KB chunks, 46 x 6 x 8** | 1.4 | 1.1 | 1.2 |

## Chunk shapes

- 2-dimensional analog of chunking is inadequate for common use case of 1D and nD access in an (n+1)-dimensional dataset
- In 2D, want chunks to be same shape as data domain to get same number of chunks in each direction of access
- In 1D and nD access, need to divide chunks read per access equally between 1D and nD domains
- For 3D use case example, balancing 1D and 2D accesses:
  - Let $n_{time}$ = number of *time*, $n_y$ = number of *y*, $n_x$ = number of *x*,
    $N$ = number of chunks = $(n_{time}\ n_y\ n_x)$ / valuesPerChunk
  - *time* by *y* by *x* chunk shape should be integral, near

    $n_{time} / N^{1/2}$ by $c\ n_y / N^{1/4}$ by $1/c\ n_x / N^{1/4}$ *( for any $c > 0$ )*
- More detailed guidance will appear soon in Unidata's Developer's Blog and netCDF documentation

## Chunking transparency

- Only creators of a dataset need to be concerned with chunk shapes and sizes
- Chunking and compression are ***invisible*** to reading software, except for performance
- Per-variable rechunking and compression implemented in calls to access libraries
- Per-dimension rechunking and compression supported by **nccopy** or **h5repack** utilities

## Chunking and compression

- In using netCDF or HDF5 libraries, a chunk is an indivisible unit of disk access, compression, and caching
- In general, smaller chunks mean worse compression
- Smaller chunks improve access times for compressed data, due to less computation for uncompression
- Benchmark times reported above are for uncompressed data
- Including compression doesn't change

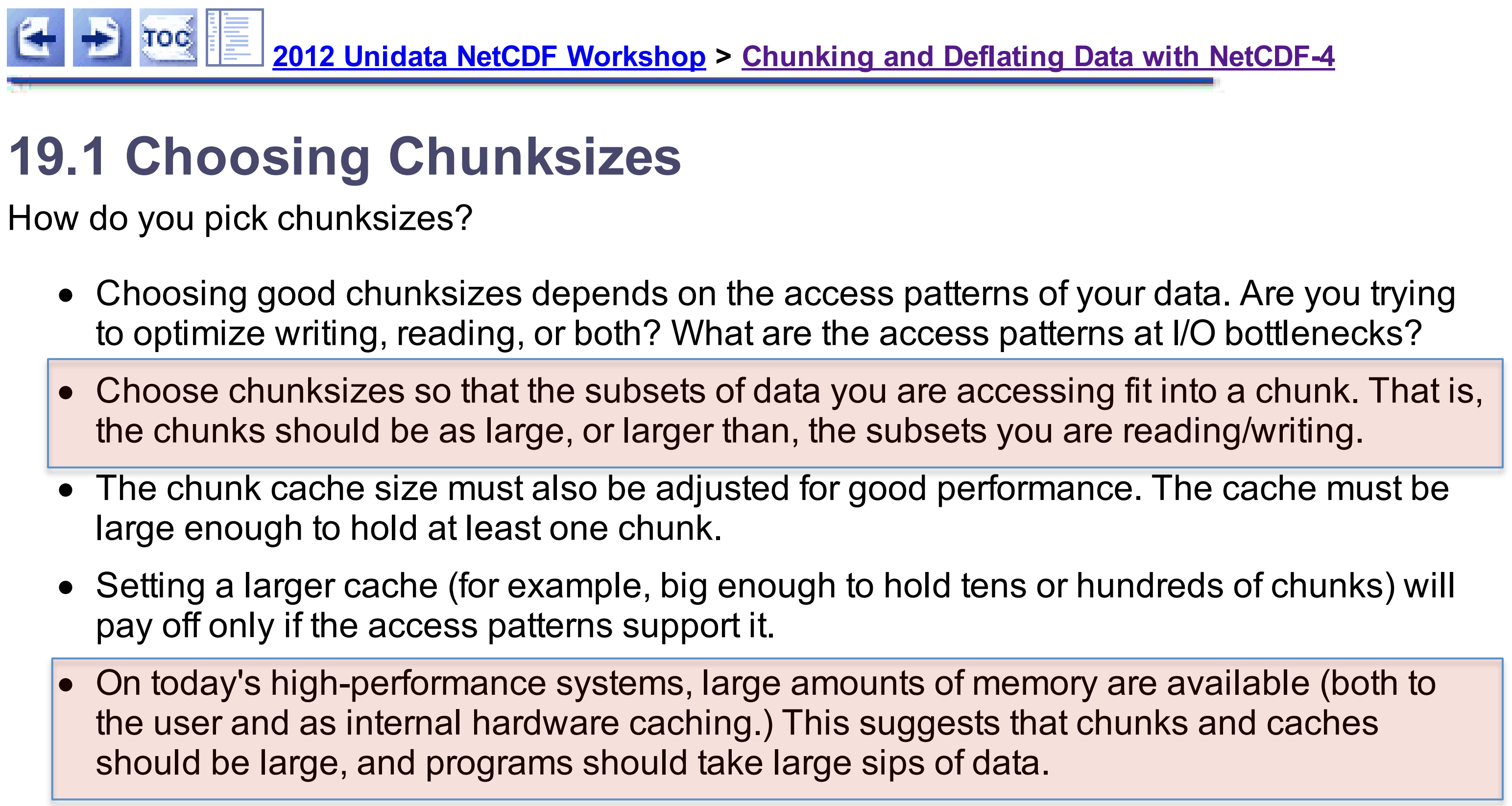
# Chunk size

small chunks | large chunks

| small chunks | large chunks |
|---|---|
| faster read access | slower read access |
| less compression | more compression |
| slower to create | faster to create |

- Chunk size should be at least size of a disk block
- Chunk shape is more important than chunk size for balanced and flexible access in multiple ways
- To re-chunk large datasets, it’s best to have lots of memory

# Some inadequate chunking advice



2012 Unidata NetCDF Workshop > Chunking and Deflating Data with NetCDF-4

## 19.1 Choosing Chunksizes

How do you pick chunksizes?

- Choosing good chunksizes depends on the access patterns of your data. Are you trying to optimize writing, reading, or both? What are the access patterns at I/O bottlenecks?
- Choose chunksizes so that the subsets of data you are accessing fit into a chunk. That is, the chunks should be as large, or larger than, the subsets you are reading/writing.
- The chunk cache size must also be adjusted for good performance. The cache must be large enough to hold at least one chunk.
- Setting a larger cache (for example, big enough to hold tens or hundreds of chunks) will pay off only if the access patterns support it.
- On today's high-performance systems, large amounts of memory are available (both to the user and as internal hardware caching.) This suggests that chunks and caches should be large, and programs should take large sips of data.

Oops, I wrote that last year, before this example …

## Summary

- By rewriting important datasets using appropriate chunking, you can tailor their access characteristics to make them more useful
- This includes structuring model outputs and remote access observations for flexible and efficient access
- Proper use of chunking can support multiple common query patterns for large datasets
- Specific guidance for how to choose optimal shapes and sizes of multidimensional chunks is currently hard to find
- We will soon be making such advice and software implementing it more widely available

## Plans

- Blog on details of what was learned, what remains a mystery
- Improve documentation with better guidance for effective use of chunking and compression
- Improve default chunking in netCDF-4 software
- Improve nccopy implementation of re-chunking to support chunking policies, as in NCO software

# For more information:

russ@unidata.ucar.edu

unidata.ucar.edu/netcdf/

hdfgroup.org/HDF5/doc/Advanced/Chunking/

Good paper on chunking details:

Optimal chunking of large multidimensional arrays for data warehousing by E. J. Otoo, Doron Rotem, and Sridhar Seshadri, 2008 unidata

Thank you!

# Benchmark details

- Disk cache in memory cleared before each run
- Measured average clock time to read 100 time ranges and 100 spatial slices
- Each of the 100 time ranges or spatial  slices used independent chunks
- Still some speedup from first read to later reads, due to disk caches not in OS memory
- Used local 7200 rpm disk for most tests (44 MB/sec)
- SSD was about 4 x faster in sample comparison runs