Advanced Concepts and Issues with H5Z-ZFP

Mark C. Miller, LLNL
Outline

• The ZFP compression library
• Endianness portability and targeting
• Handling >4D datasets even though ZFP’s max is 4D
• Understanding interplay between ZFP chunklets and HDF5 chunking
• Partial writes and ZFP chunklets
• Writes over main “time” loop of application and ZFP chunklets
• Reading and writing ZFP compressed arrays
• Letting ZFP compression parameters vary over single dataset
H5Z-ZFP uses ZFP Library

- [https://github.com/LLNL/zfp](https://github.com/LLNL/zfp)
- Development lead by Peter Lindstrom
- Lossy (and mostly lossless) compression
- 1, 2, 3 and 4 dimensional data
- 32 and 64 bit integer and floating point data
- Rate, accuracy, precision and expert modes
- GPU and OpenMP kernels available
- Creates a data stream that is endian-agnostic
Endianness portability and targeting

- ZFP operates in an endian-agnostic way
- Uncompression produces the correct endianness result for consumer
- What if data is written on big-endian but read on little-endian?
  - HDF5's internal pipeline processing does not expect this
  - HDF5 expects to have to byte-swap whatever compressor returns itself
  - Correct result is returned from H5Z-ZFP filter and HDF5 will ruin it
  - Have logic to detect this situation and pre-ruin the result so HDF5 will un-ruin it
- We also disallow endian targeting (it is meaningless)
Handling >4D Data

• ZFP Library supports a maximum of 4 dimensions
• How to handle datasets with more than 4 dimensions?
• Ensure that at most 4 dimensions of the HDF5 chunking are non-unity
• Magic of HDF5 is that ZFP is compressing individual chunks and as long as those are \( \leq 4D \), everything works
• When you have a choice, select smoothest dimensions for non-unity
ZFP Chunklets and HDF5 Chunks

• ZFP operates in quanta of $4^d$ chunklets where $d$ is the dimensionality

• Example: For 2D, ZFP chunklets are 4x4

• What about data that has dimensions that are not multiple of 4?
  • This leads to partial chunklets
  • ZFP uses its own notion of a “fill value” (which I think varies with chunklet)

• For a 2D array, 27 x 101, ZFP will treat as 28 x 104 (potential 6.4% waste)

• For a 3D array 1024 x 1024 x 2, ZFP will treat as 1024 x 1024 x 4 (50% waste)

• If writing 2D slices in memory to 3D array in file AND want ZFP compressed over all 3 dimensions...
Partial Writes and ZFP Chunklets

• Chunk size and shape in relation to partial write impacts performance

• Writing scenario 1
  • I/O request might partially overlap chunks already present in file (maybe from a previous write)
  • HDF5 must engage in read/modify write for those chunks (if lucky they are cached)

• Write scenario 2
  • I/O request might partially overlap chunks NOT already present in file
  • HDF5 will assume “fill value” (which defaults to zero) for those regions
  • May interfere with ZFP’s compression performance and own notion of fill value
Writes over main “time” loop and ZFP chunklets

• Maybe iterating overtime computing 2D slices of some ultimately 3D dataset (2D+time) in the file and want ZFP compression over all 3 dimensions of the data in the file.
• Remember, ZFP wants to treat every dimension as a multiple of 4, even in the time dimension.
• Choice is to buffer 4 timesteps up before calling H5Dwrite or
• Suffer performance issues associated with ZFP’s “padding” to 4 and however that plays out with HDF5 chunk
ZFP Compressed Arrays

• Works only with rate mode of ZFP compression (guarantees size)
• Use case 1: Read compressed data from file instantiating compressed array in memory
• Use case 2: Write compressed array from memory creating compressed dataset in file such that any downstream reader is completely normal
• Use the H5Dread_chunk() and H5Dwrite_chunk() routines
  • Slightly problematic because it changes how consumer or producer use API
Letting ZFP compression params vary over HDF5 chunks

• Currently, H5Z-ZFP encodes filter params in “cd_values”
  • Actually somewhat problematic due to double precision ZFP params and unsigned int type for cd_values

• HDF5 delivers to filter individual chunks

• Could just decide to vary ZFP compression params on chunk-by-chunk basis and instead store those params as part of each chunk

• For reasonably sized chunks, overhead would be negligible