Supporting Sparse Data in HDF5

Elena Pourmal elena.pourmal@lifeboat.llc
John Mainzer john.mainzer@lifeboat.llc

Lifeboat

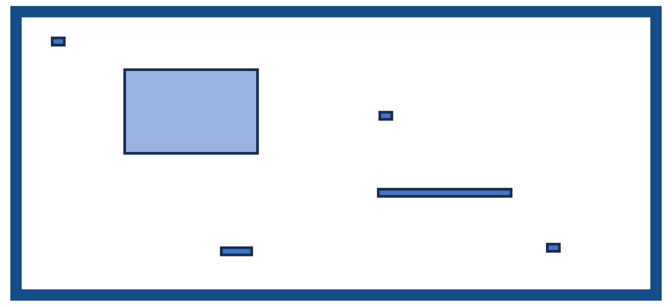
Outline

- Motivation for new type of storage
- Notion of structured chunk and its metadata
- Programming model and new APIs

What is Sparse Data?

- Sparse data is ubiquitous; examples come from the experimental sciences and computer modeling:
  - High Energy Physics (HEP); Neutron and X-Ray scattering; Mass Spectrometry experiments
  - Transmission electron microscopy
  - Genomics
  - AMR
  - Machine learning applications
- There is no ”standard” definition of “sparse data”.
  - **Linear algebra** – data is considered sparse if less than 30% of matrix elements are non-zeros.
  - **Experimental sciences** - only 0.1% to 10% of gathered data is of interest, but it may contain a bigger percentage.

Motivation for Sparse Storage: LCLS-II Use Case



- Experiments produce a stream of two-dimensional images.
- For each image it is possible to automatically identify either:
  - A rectangular **R**egion **o**f **I**nterest (ROI) in each image which will typically comprise about 10% of the image, or
  - 50 – 100 small subsections in each image (typically 5 to 10 contiguous points or pixels).
  - The number, size, configurations, and locations of ROI or the small subsections change over time.
- For each image in the stream it is desired to store
  - Only the ROI or the point list in a three-dimensional HDF5 dataset
    - One must be able to recover both the location and contents of the ROI and/or the elements of the point list.
  - Every N^{th} two-dimensional image in full, where N is constant over any given experiment. Note that the ROI or point list of each “full” two-dimensional image must be recoverable as well.

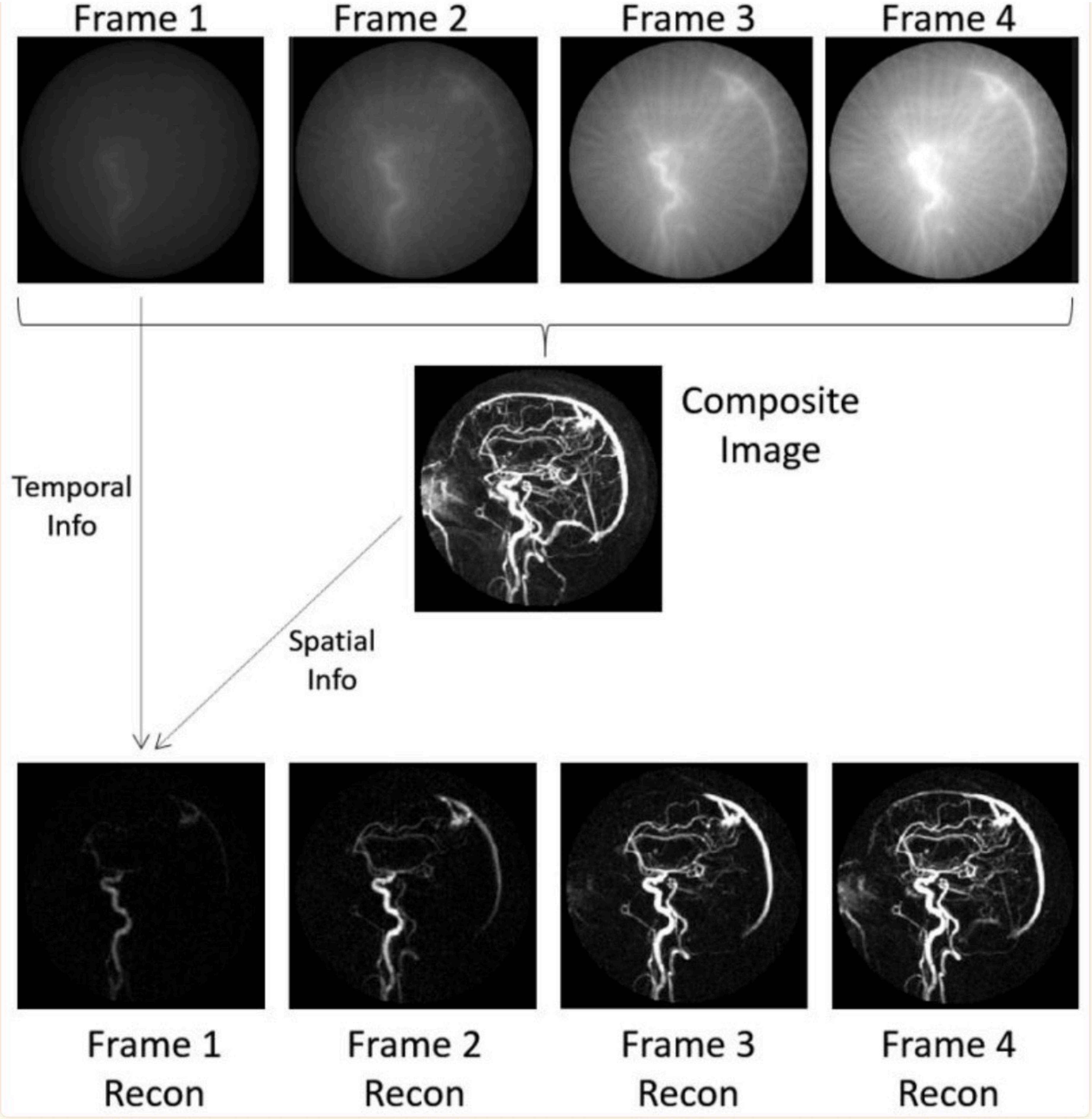
LCLS-II Use Case (cont'd)

- To meet this requirement, we propose to implement sparse datasets:
  - Only the entries that have been written explicitly are defined.
  - The defined entries can be readily identified, and read. To the above minimal requirement, we also add:
    - *Compatibility with dense datasets* – thus code designed for the existing dense datasets will still work, reading defined values if available, and the fill value (default 0) where not.
    - *Ability to erase defined values* – that is to remove them from the set of defined values.
    - *Ability to use filtering* (compression).

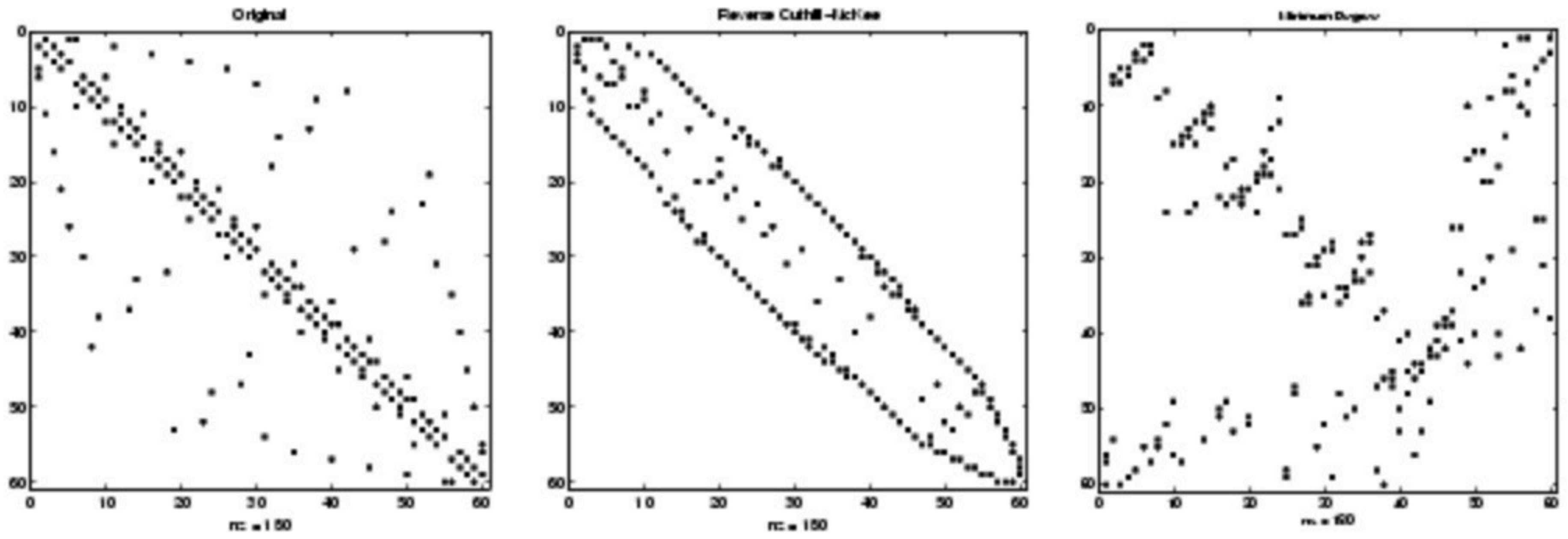
Other Use Cases

See notes for references

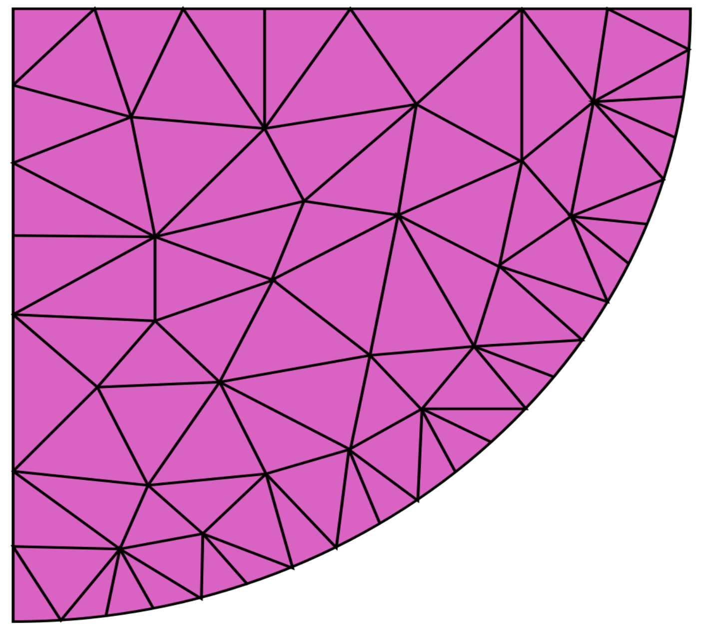
Sparse Reconstruction in MRI



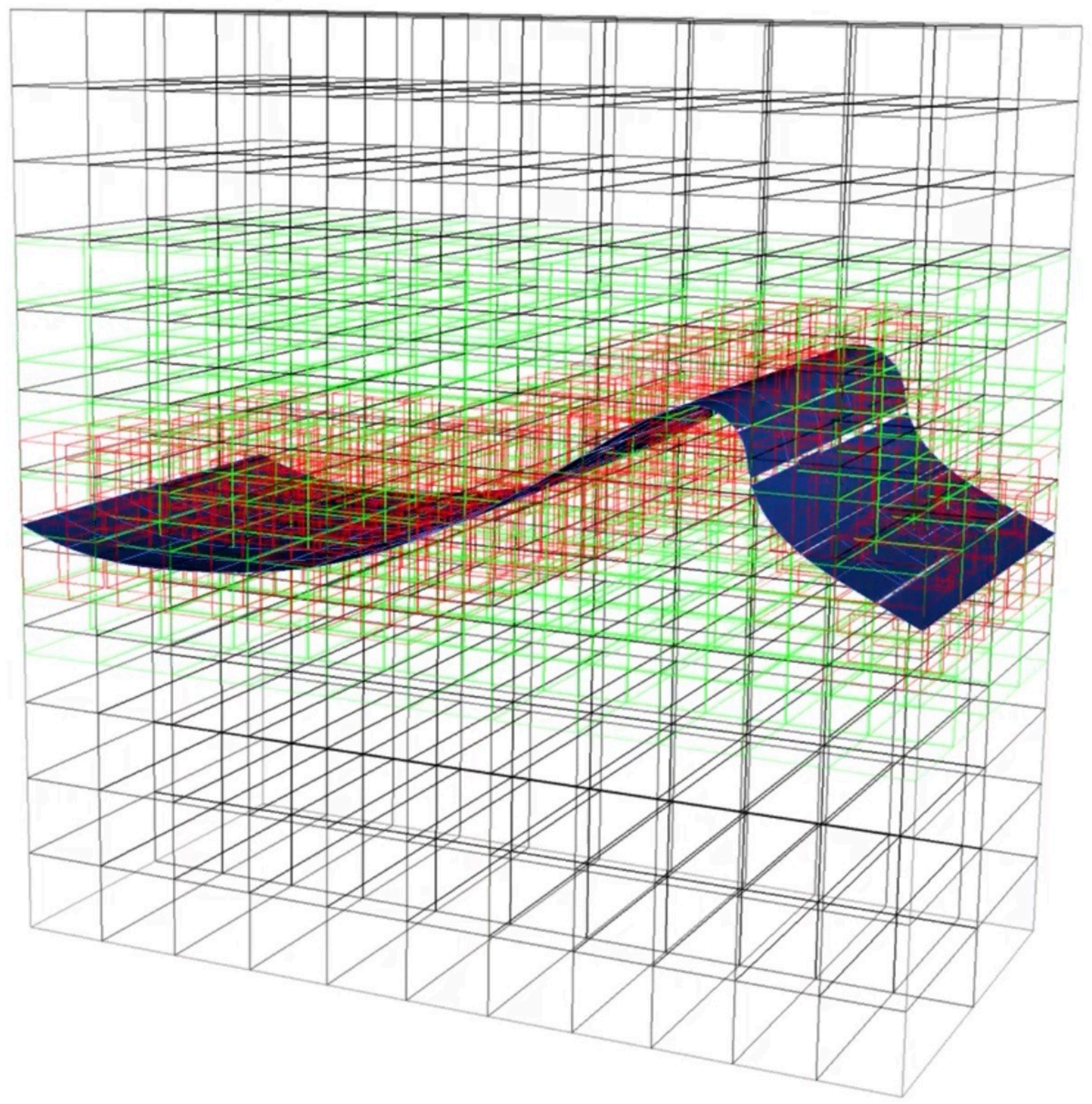
Linear algebra



Computer modeling



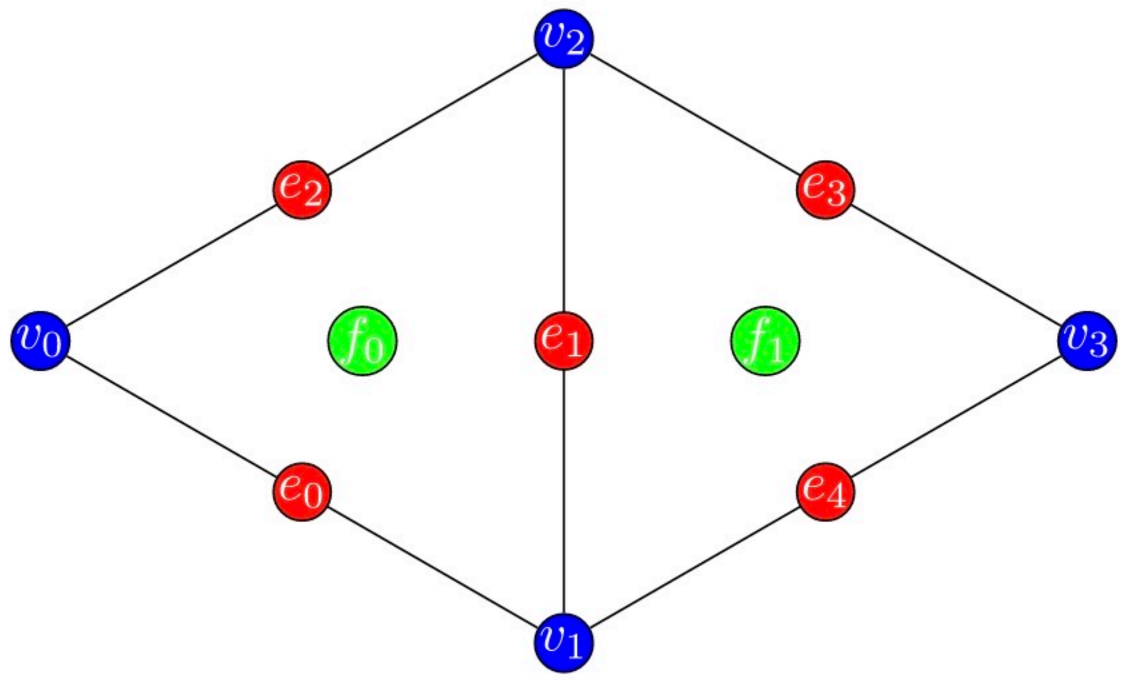
Unstructured meshes



AMR

Unstructured Meshes and Sparse Data

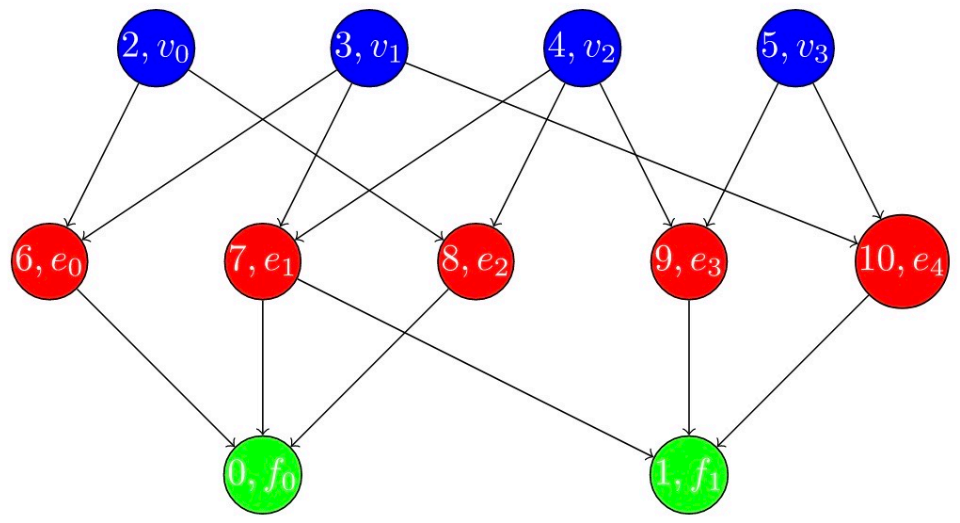
Mesh



Dataset
(connectivity matrix)

	2	3	4	5
0	1	1	1	0
1	0	1	1	1

Mesh represented as Directed Acyclic Graph



	0	1	2	3	4	5	6	7	8	9	10
0	0	0	0	0	0	0	**1**	**1**	**1**	0	0
1	0	0	0	0	0	0	0	**1**	0	**1**	**1**
2	0	0	0	0	0	0	**1**	0	**1**	0	0
3	0	0	0	0	0	0	**1**	**1**	0	0	**1**
4	0	0	0	0	0	0	0	**1**	**1**	**1**	0
5	0	0	0	0	0	0	0	0	0	**1**	**1**
6	**1**	0	**1**	**1**	0	0	0	0	0	0	0
7	**1**	**1**	0	**1**	**1**	0	0	0	0	0	0
8	**1**	0	**1**	0	**1**	0	0	0	0	0	0
9	0	**1**	0	0	**1**	**1**	0	0	0	0	0
10	0	**1**	0	**1**	0	**1**	0	0	0	0	0

Example: PETSc Tutorial https://petsc.org/release/manual/dmplex/

Developing a Concept of Structured Chunk

Chunked dataset 13 x 10
Chunk size 4 x 5

Data to store from the upper-left chunk

Encoded Selection for [2,2] - [3,4] hyperslab
Data 66 69 72 96 99 102

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	66	69	72	75	78	81	0	0
0	0	96	99	102	105	108	111	0	0
0	0	126	129	132	135	138	141	0	0
0	0	0	0	0	0	0	0	0	2
100	0	-100	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	3	0

Generalizing for fixed-size and VL datatypes…

Fixed-size HDF5 datatype

byte	**byte**	**byte**	**byte**
Section 0: Encoded Selection of Defined Elements			
Section 0 Checksum			
Section 1: Fixed Length Data Section			

Variable-length HDF5 datatype

byte	**byte**	**byte**	**byte**
Section 0: Encoded Selection of Defined Elements			
Section 0 Checksum			
Section 1: Fixed Length Data Section			
Section 1 Checksum			
Section 2: Variable-size data heap			
Section 2 Checksum			

Structured Chunk Layout and its Metadata

Structured Chunk Layout

byte	byte	byte	byte
Section 0 (variable size – may be empty)			
Section 0 Checksum (may not exist)			
...			
Section N (variable size - may be empty)			
Section N Checksum (may not exist)			

Structured Chunk Metadata

byte	byte	byte	byte
Offset of section 1			
...			
Offset of section N			

Filtered Structured Chunk

- Filtered Structured Chunk is a Structured Chunk with one or more of its sections passed through filter pipelines.

Example of Filtered Structured Chunk with VL Data

byte	byte	byte	byte
Section 0: Filtered Encoded Selection with Checksum			
Section 1: Filtered Fixed Length Data with Checksum			
Section 2: Filtered Variable Size Data Heap with Checksum			

Filtered Structured Chunk Metadata

byte	byte	byte	byte
Filter Masks (one per section)[5]			
Section Offsets (one per section less 1)			
Section Unfiltered Sizes (one per section)			

Proposed New APIs

Function Name	Short Description
H5Dget_defined	Retrieves a dataspace object with the defined elements
H5Derase	Deletes elements from a dataset
H5Dwrite_struct_chunk	Writes structured chunk
H5Dread_struct_chunk	Reads structured chunk
H5Dget_struct_chunk_info	Gets structured chunk info
H5Dget_struct_chunk_info_by_coord	Retrieves the structured chunk information
H5Dstruct_chunk_iter	Iterates over all structured chunks in the dataset
H5Pset_filter2	Adds a filter to a filter pipeline for a specified section of sparse structured chunk
H5Pget_nfilter2	Returns the number of filters in the pipeline for a section of structured chunk
H5Pget_filter2	Returns information for a filter in the pipeline for a specified section
H5Pget_filter_by_id2	Returns information for a filter specified by its identifier in the pipeline for a specified section of structured chunk
H5Premove_filter2	Removes a filter in the filter pipeline for a specified section
H5Pmodify_filter2	Modifies a filter in the filter pipeline for a specified section of structured chunk

H5Pset_filter2

- We want to address deficiency of the current API for passing filter's data

```
herr_t H5Pset_filter2 (hid_t plist_id,
                       uint64_t section_number,   ← new parameter
                       H5Z_filter_t filter,
                       uint64_t flags,
                       size_t buf_size,
                       const void *buf)
```

new parameter

new datatype

Programming Model

```
/*
 * Create the dataset creation property list, add the gzip
 * filter to compress all sections of the sparse chunk using
 * DEFLATE filter.
 */
dcpl   = H5Pcreate (H5P_DATASET_CREATE);
status = H5Pset_layout (dcpl, H5D_SPARSE_CHUNK);
status = H5Pset_chunk (dcpl, 2, chunk_dims);
status = H5Pset_deflate (dcpl, 9);
/*
 * Create the dataset.
 */
dset = H5Dcreate (file, DATASET, H5T_STD_I32LE, space, ….);
```

Programming model (cont'd)

```
dcpl   = H5Pcreate (H5P_DATASET_CREATE);
status = H5Pset_layout (dcpl, H5D_SPARSE_CHUNK);
status = H5Pset_chunk (dcpl, 2, chunk_dims);

/* Apply compression methods to different sections of
 * a structured chunk. In this example, sparse chunk has two sections.
 * We are using gzip compression on the encoded selection section
 * and szip on the fixed-size data section.
 */
flags = H5Z_FLAG_OPTIONAL;
status = H5Pset_filter2 (dcpl, H5Z_FLAG_SPARSE_SELECTION,
                               H5Z_FILTER_DEFALTE, flags, nelem, &data);

status = H5Pset_filter2 (dcpl, H5Z_FLAG_SPARSE_FIXED_DATA,
                               H5Z_FILTER_SZIP, flags, …);
```

Acknowledgement

This work is supported by the U.S. Department of Energy, Office of Science under Award number DE-SC0023583 for SBIR project "Supporting Sparse Data in HDF5".

We thank The HDF Group developers and Quincey Koziol, Principal Engineer, AWS HPC for reviewing our technical proposal and for providing a constructive feedback.

References

1. https://github.com/HDFGroup/hdf5/discussions/3257

2. John Mainzer, Elena Pourmal, “RFC: File Format Changes for Enabling Sparse Storage in HDF5”. Available from https://github.com/LifeboatLLC/SparseHDF5/

3. John Mainzer, Elena Pourmal, “RFC: Programming Model to Support Sparse Data in HDF5” Available from https://github.com/LifeboatLLC/SparseHDF5/

4. J. Mainzer *et al*., "Sparse Data Management in HDF5," *2019 IEEE/ACM 1st Annual Workshop on Large-scale Experiment-in-the-Loop Computing (XLOOP)*, Denver, CO, USA, 2019, pp. 20-25, doi: 10.1109/XLOOP49562.2019.00009

5. The HDF Group, “Draft RFC: Sparse Chunks” https://docs.hdfgroup.org/hdf5/rfc/RFC_Sparse_Chunks180830.pdf

6. The HDF Group, Variable-Length Data in HDF5 Sketch Design, https://docs.hdfgroup.org/hdf5/rfc/var_len_data_sketch_design_190715.pdf

Thank you!

Questions?

Lifeboat

www.lifeboat.llc
info@lifeboat.llc

U.S. DEPARTMENT OF ENERGY AWARDEE™