Exercise 10 — MPI I/O, Weak Scaling

Question 1: Data Compressor

Concepts:

- MPI I/O: Collective/Non-Collective operations
- File offsets: Determine correct byte offsets to write a file in parallel

Question 2: Weak Scaling of Data Compressor

Concepts:

- Scaling analysis on paper
- Understand scaling behavior of parallel code

Question 1 — MPI I/O for Data Compressor

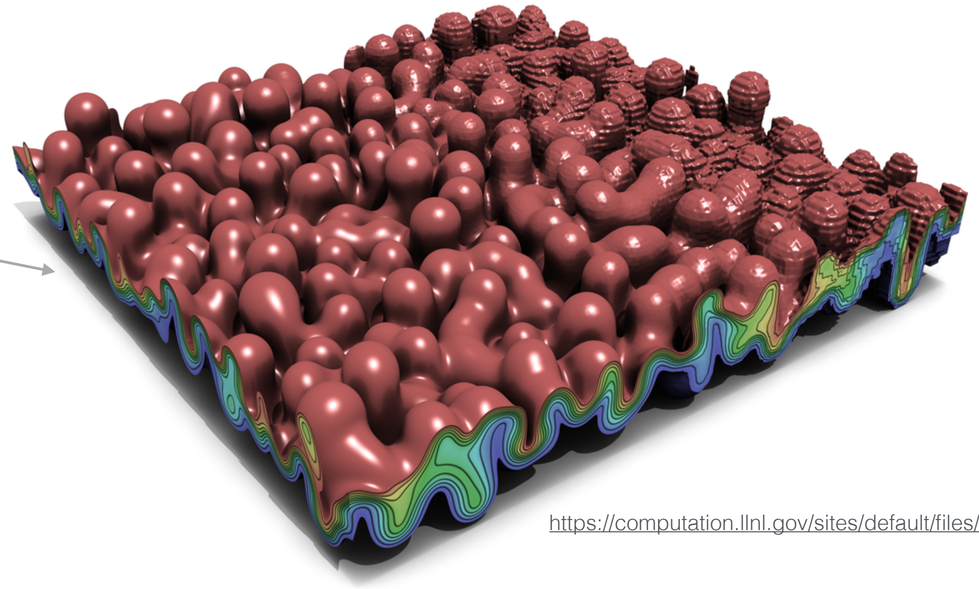
- Code performs floating point data compression for 2D input data
- The compressor is already implemented
- You focus on implementing a parallel file format with the compressed data
- Your code must be able to read and write a file format that you define in parallel using MPI file operations
- We will work with a 2D image file as input to the compressor

Question 1 — MPI I/O for Data Compressor

Data compression:

- Important for HPC computing
- Examples: Reduce disk footprint; Reduce communication overhead
- Special interest in scientific computing: Floating point compression
- Near-lossless or lossy
- We will use the zfp (lossy) floating point compressor

Near-lossless Compression: Information is preserved



Lossy Compression: Information is lost during compression/ decompression cycle

https://computation.llnl.gov/sites/default/files/public/zfp-adaptivity.png

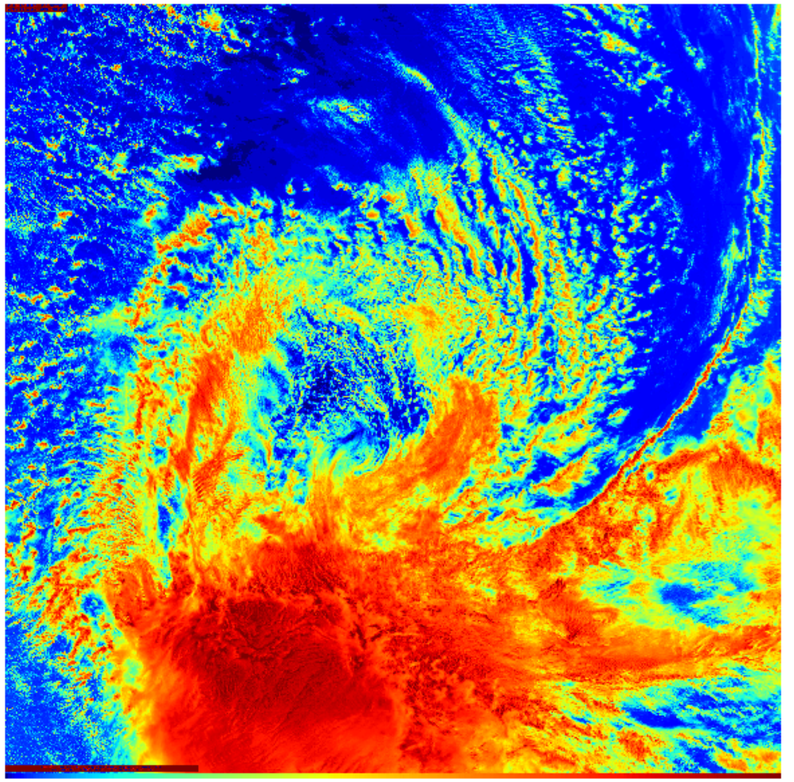
Question 1 — MPI I/O for Data Compressor

Get zfp library and image data before you start:

```
make setup
```

This will do:

1. Download zfp library and compile it
2. Download the input image `cyclone.bin.gz`



You can compile the code with:

```
make
mpirun -n <p> ./mpi_float_compression <tol> 4096 cyclone.bin.gz
```

Number of MPI processes. The image dimension 4096 must be evenly divisible by p

Compression tolerance. 0 results in near-lossless compression (within machine precision)

Input image dimension (square image) and image filename

Question 1 — MPI I/O for Data Compressor

Part a.) Implement file protocol for data compressor

You have to complete the function `write_data`:

- The function will write your custom file format for the compressed data (similar to a `.zip` file for example)
- You need to store meta data in the file which you need when you read the file at a later time
- Each MPI rank writes the compressed buffer obtained from zfp into the file
- You have to use `MPI_File...` routines. You can use collective or non-collective calls.

Question 1 — MPI I/O for Data Compressor

Part a.) Implement file protocol for data compressor

You can find the meta data that you need to store in the file in the `FileHeader` and `BlockHeader` structures:

Required once

```
1 struct FileHeader
2 {
3     double tol;
4     size_t Nx, Ny, Nb;
5 };
```

Compression tolerance

Dimensions of 2D data

Number of compressed blocks stored in the file

Required once for each compressed block

```
1 struct BlockHeader
2 {
3     size_t start, compressed_bytes, bufsize;
4 };
```

Number of bytes in compressed block

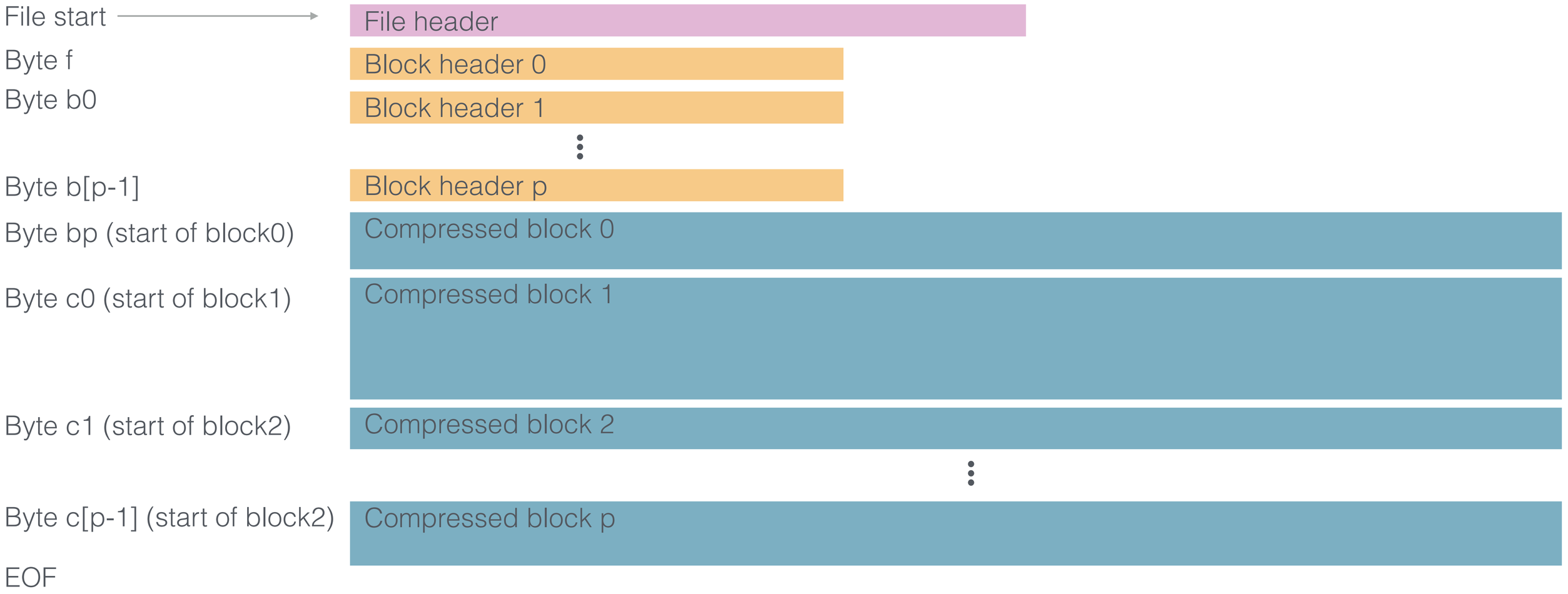
Size of work buffer needed to decompress the data

Byte offset in the file for compressed block

Question 1 — MPI I/O for Data Compressor

Part a.) Implement file protocol for data compressor

You are free to implement the file protocol yourself. One way to do it:



Question 1 — MPI I/O for Data Compressor

Part b.) Implement the inverse operations of part a.)

You have to complete the function `read_data`:

- The function will read your custom file format in order to decompress the stored blocks
- You need to read the meta data in the file and pass it to the decompressor (pass by reference, see function signature)
- For simplicity, the same number of MPI ranks is used for reading and each rank again reads one compressed block and decompresses that block.
- The decompressed data is written to a `.bin.gz` file and can be compared to the original image with the `print_png.py` Python script.

Question 1 — MPI I/O for Data Compressor

Part b.) Implement the inverse operations of part a.)

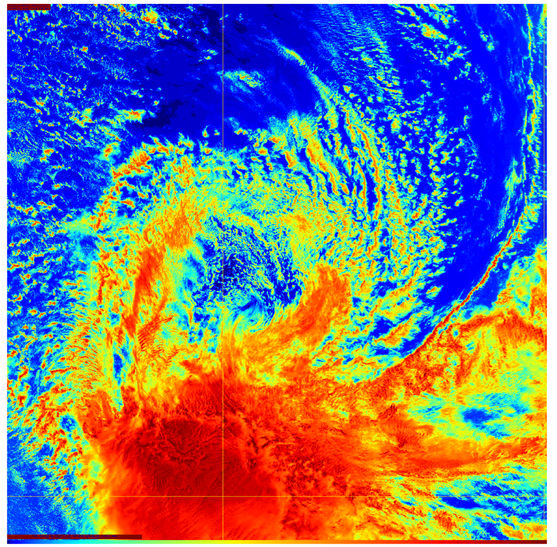
To simplify the post-processing with Python, you can use

```
./run.sh <tol>
```

Script does the following:

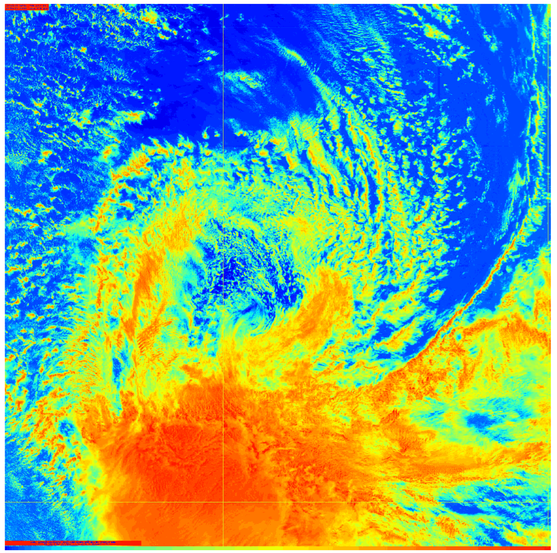
1. Source the build environment
2. Compile the code
3. Run it with MPI
4. Run the Python post-processing

tol=0



128 MB

tol=200



3.4 MB

Compression rate (zfp): 37.7

Compression rate (zip): 7.5

Question 2 — Weak Scaling

- Pen and Paper
- You are given a table with execution time of the compressor in Question 1
- Measurements correspond to different number of MPI processes and different (square) image sizes N
- Some of the measurement data is not suitable for this problem
- For the weak scaling the problem size *per process* remains constant
- Identify the correct processor/image size pairs and compute the weak efficiency and draw these points in the blank diagram

$$E_w = \frac{t_1}{t_p}$$