CS341: Video Compression for Grib2 Files

A Project Report by Asaad Alghamdi

Introduction

Video compression can be used to encode multiple images with a spatio-temporal relationship. The advantage and impact of using Video Compression on Grib messages is unexplored in literature. Furthermore, there is no clear documentation that explains well what Grib is. In this report, I present the Grib architecture, the goals I have set, the experiments I have performed, and the conclusions I have reached. For the purpose of this report, the numbers that will be shown and used are related to the Red Sea Climate Dataset.

I define the following objectives:

- [**Analysis**] Understand the standards and architecture of Grib2 and how the methods are connected
- [**Implementation**] Reconstruct Grib2 JP2000 encoder and decoder methods so that
  - The encoder saves encoded image files on hard disk instead of the normal behavior where the encoder saves the encoded image within the Grib file
  - The decoder uses external encoded image files that are saved on the hard disk instead of the normal behavior where the decoder accesses the memory address of the grib buffer
- [**Video Codec Encoding**] Encode the results of the modified Encoder image files using FFMPEG libx264 video codec
- [**Video Decoding**] Decode the video frames back into individual uncompressed images
- [**Analysis**] Compare between the original uncompressed images and the images that were decoded from the video using PSNR as a metric

- [**Analysis**] Compute and Compare the MSE between the original floating point values and the values decoded from the Video Compression method
- [**Discussion**] Reason and reach conclusion about the potential approaches that could be advantageous to our current setup.

Grib Architecture

What is GRIB? And why does it offer more compressability than other file formats? I have sampled 12 timesteps (7 variables) from the **2.6TB .nc file.** The resulting file is a .nc file **341 MB** in size. The variables are the following:

- **Time**: Minutes since 1980-01-01 12:00:00
- **Longitude**
- **Latitude**
- **QV at 2M** : Specific Humidity at 2M ?
- **Temperature** at 2 M
- **PSFC**
- **U10**: U-component of the wind at 10M
- **V10**: V-component of the wind vector at 10M

The variable naming, as well as the grid format of the nc file, does not conform with GRIB2 standards. This can be fixed by applying the following command:

```
cdo -chname,U10,u,V10,v,Q2,q,T2,t,PSFC,sp,GSW,nswrs,GLW,dlwrf asaad.nc
asaad_modified.nc
```

However, it is still possible to convert such a file to GRIB2 using the following command:

```
cdo -f grb2 -z jpg -b 8 -copy asaad_modified.nc output_asaad.grb2
```

This command will create a 8-bit quantized jpeg-2000 compressed file. Now we have a GRIB and a NetCDF file.

NetCDF4 is **bad** when it comes to **compressibility**. Even when applying the post-processing step of **Gzip**, there is not much improvement on GRIB as much as NetCDF4. The table below summarize the results.

File \ Attribute	asaad.nc	asaad.grb
Size	**341 MB**	**165 MB** (2.06)
JPEG	-	**96 MB** (1.72)
GZIP	**269 MB** (1.26)	**97 MB**** (0.99)
Compression Ratio (Original NC -> Best GRB)	-	**3.55** using **JPEG**

In order to understand the compressability of GRIB, we need to understand the architecture. The architecture of GRIB is as follows:

- Every GRIB file consists of messages and each message has its own date and timestamp
- Every message consists of a variable, or multiple variables if multi-field feature is enabled, that is mapped to two dimensions Longtitude and Latitude. Ultimately, there are Longtitude * Latitude values in a message.
- When performing encoding and decoding, these operations apply over only the floating point values
- GRIB offers different bit-packing techniques and compression methods such as JP-2000. Different encoding\decoding methods are called depending on the type of encoding and decoding needed for a GRIB file

It is important to note that GRIB messages are of a sequential nature. This concludes the architecture part.

Experiments

In this section, I will describe the experiments I have performed and their results in chronological order.

File-based Encoder and Decoder

The default JPEG-2000 Encoder and Decoder provided by Eccodes GRIB library uses memory addresses to store the encoded images and the decoded data. However, I am interested in obtaining these images as external files so I can apply video codecs on them. Therefore, I have implemented a file-based encoder and decoder of the JP-2000 standard. The file-based Encoder will create image files instead of storing the encoded image within the grib file. The file-based decoder will go through a directory of encoded images and decode them.

Video Compression

Encoding

After obtaining image files by using my file-based Encoder, I attempted to apply different video codecs over the encoded images in the Encoded_Images directory and have run into a lot of trial and error. However, I have found that the following command works with no errors:

```
ffmpeg -i EncodedJPGs/%d.j2k -c:v libx264 -crf 18 -pix_fmt gray8
Temp100_libx264_grib.mov
```

This will apply the video codec libx264 over the encoded images in the directory. Any video codec could be applied as long as it respects the assumptions GRIB2 is making about the image (8 bits, greyscale). The constant rate factor of 18 enables us to perform lossy compression, the higher it is the more lossy our compression is. The pixel format is very important and it has to be an 8 bits per pixel gray. Once we apply the aforementioned command, we will have a video of the format .mov.

Decoding

Now we do the reverse and decode the video we just created into individual images using the following command:

```
ffmpeg -i Temp100_libx264_grib.mov -c:v jpeg2000 -pix_fmt PIX_FMT_GRAY8
Decoded_VideoImages/%d.jp2
```

This will produce individual images that can be used to reconstruct the grib file using the memory-encoder and file-based decoder. Again, it is very important to specify the pixel format.

I have applied the Video Compression Experiment in two instances:

1. 100 timesteps temprature grib file and the goal is to compute MSE
2. 10K timesteps temperature grib file and the goal is to compute the images PSNR

MSE was computed by extracting the values as a json file using grib_dump and performing the computation using Python. PSNR was computed using the original images and the reconstructed ones. In the table below you can find a summary of the result of the experiments done using the described method..

GRIB File	Compressed Video Size	PSNR	MSE
100 Timesteps Temp (Original: 21MBs)	10 MBs	N/A	0.33
10K Timesteps Temp (2.4GB)	283 MBs	40.953	N/A

Compression ratio for 100 Timesteps: 2.1, and for 10K Timesteps: 12

For the 100 Timesteps, even though the video size is half that of the original but the MSE is low. There was no need to perform PSNR metric analysis on the 100 timesteps as I'm performing PSNR analysis on the 10K timesteps. Getting a grib_dump json file from 10K timesteps Temp takes a very long time and I was faced with a dilemma of having sudo access but low storage (my personal laptop) vs no sudo access and huge storage (ibex). I look forward to expanding this experimental section by varying the video codec parameters such as CRF and possibly the codec itself.

Conclusion

In this paper, I have introduced the goals of this project and how I have implemented them as well as the results. While the experimental results are limited, it still shows a lot of promise. By combining file-based encoders\decoders with video codecs compression, I was able to reconstruct the original values with good accuracy. However some challenges still remain such as:

1. Integrating the new encoder and decoder methods with the commandline interface
2. Investigating the potential of multi-field messages and RGB images (a color for every attribute)
3. Investigating the potential of applying SZ on GRIB floating point data instead of using JPEG-2000
4. Investigating the potential of AVIF Image File Format
5. Investigating and optimizing the parameters for Video encoding and decoding

Development on Ibex was tough because the limited sudo access available. While development on personal machines was troubling because the filesize can go up to 60GB+ after trying to dump a file.

Resources and Appendix

Shell Commands

One-liner for compressing\decompressing images so you can run the PSNR script

```
for((i=1;i<=10000;i++)); do opj_compress -i
~/github/panos/build/EncodedJPGs_new/$i.j2k -o orign/$i.pgm; done
```

Links

- https://confluence.ecmwf.int/display/ECC/
- https://github.com/AZed/cdo
- https://github.com/ecmwf/eccodes-python
- https://link.springer.com/content/pdf/10.1007%2F978-3-642-38750-0.pdf (On compression)
- https://code.mpimet.mpg.de/attachments/download/13557/CDO_Seminar_20161206.pdf
- https://code.mpimet.mpg.de/projects/cdo/wiki/Tutorial
- https://code.mpimet.mpg.de/attachments/download/13557/CDO_Seminar_20161206.pdf
- https://www.unidata.ucar.edu/software/netcdf/workshops/2008/utilities/Cdo.html

- [http://xarray.pydata.org/en/stable/index.html#:~:text=xarray%20%28formerly%20xray%29%20is%20an%20open%20source%20project,with%20labelled%20multi-dimensional%20arrays%20simple%2C%20efficient%2C%20and%20fun%21]