Wavelet-Based Compression of CFD Big Data

◯ Dmitry Kolomenskiy, JAMSTEC, Yokohama, Japan, E-mail: dkolomenskiy@jamstec.go.jp
Ryo Onishi, JAMSTEC, Yokohama, Japan, E-mail: onishi.ryo@jamstec.go.jp
Hitoshi Uehara, JAMSTEC, Yokohama, Japan, E-mail: uehara@jamstec.go.jp

A new approach for compression of CFD data is presented, inspired by image processing. It uses wavelet decomposition and subsequent range coding with quantization suitable for floating-point data. The effectiveness of this method is demonstrated by applying it to the velocity field of a turbulent vortex flow past a cylinder.

1. Introduction

High-performance computing (HPC) can produce large volumes of output data. A computational fluid dynamics (CFD) simulation using several hundred or thousands of processor cores would allocate three-dimensional fields of many Gigabytes per hydrodynamic variable. As the solution evolves in time, a new three-dimensional data set is produced at every time step. This high 'volume' and 'velocity' of data flow is characteristic of big data applications (1), and it is not surprising that high-performance CFD hits the limitations of its contemporary data handling technologies.

In particular, data storage capacity is finite. To alleviate this constraint in practice, data reduction is routinely performed during the simulation and only the quantities of interest such as hydrodynamic forces, etc., are stored as time-resolved sequences. Nevertheless, it is often required to store the complete hydrodynamic fields for purposes such as flow visualization, restart of the simulation or additional post-processing. These large datasets quickly saturate the available disk space if stored as floating-point arrays without compression.

Unfortunately, lossless data compression tools, such as LZMA compression, reduce the file size by only about 25%. Common practice is to accept some data loss, store the CFD output fields in single precision and to downsample the data, e.g., save every second point value in each direction. Such reduced datasets, while being of insufficient information capacity for the simulation, are often suitable for postprocessing. Given that the differences between neighboring point values can be interpreted as wavelet coefficients, a more refined version of the mentioned approaches is to apply wavelet transform to the CFD field and encode the wavelet coefficients using entropy coding. This technique is currently widely in use for image compression, being part of the JPEG2000 standard. In the present study, we adapt this approach for CFD data storage.

2. Description of the Method

We only consider those datasets that consist of structured Cartesian blocks, each block is treated independently by the algorithm, but the method can be generalized to unstructured grids as well. Let $\{f_{i,j,k}\}$ be a three-dimensional scalar field sampled on a grid $\{x_i, y_j, z_k\}$, where $i = 1, n_x$; $j = 1, n_y$; $k = 1, n_z$, and $\max |f_{i,j,k}| = 1$.

On the highest level, the compression method consists of the following steps: (I) wavelet transform; (II) quantization; (III) entropy coding. Reconstruction is the inverse of the above operations in the reverse order.

The wavelet transform produces an array of real values F_m with the same number of elements as in the original array f, i.e., $m = 1, n_x \cdot n_y \cdot n_z$. We use a three-dimensional multiresolution transform based on bi-orthogonal Cohen–Daubechies–Feauveau 9/7 (CDF9/7) wavelet, expressed in terms of lifting steps. Numerical experiments with sample CFD velocity data confirmed this transform leading to the highest compression ratio among all wavelet transforms that we tested.

Quantization reduces the real values F_m to a set of 1-byte numbers (i.e., integer numbers ranging from 0 to 255), as required for the subsequent entropy coding. Note that, in general, entropy coders are not limited to 256 size alphabet, but this size is convenient as it corresponds to *char* type native in C. A double precision variable F can be losslessly quantized using eight 1-byte numbers. Sets of less than eight 1-byte numbers are used in a lossy compression. The preset tolerance parameter ε determines the accuracy in that case. Bit planes of the quantized data are determined as follows.

Startup:

- Begin with $F^1 = F$ (for all m - index m is omitted for brevity);
- Set the tolerance ε;
- Assign $q = 256$, $\ell = 1$.

Iterations:

- $\delta^\ell = (\max F^\ell - \min F^\ell)/(q-1)$;
- If $\delta^\ell < \varepsilon$ then assign $\delta^\ell = \varepsilon$;
- $Q^\ell = \left[(F^\ell - \min F^\ell)/\delta^\ell\right]$;
- $F^{\ell+1} = F^\ell - (\delta^\ell Q^\ell + \min F^\ell)$;
- Increment ℓ by 1.

Iterations end when δ^ℓ reaches the desired tolerance ε.

At every iteration, coefficients Q^ℓ of each bit plane ℓ are encoded using a range coder, which is one of the existing entropy coding techniques. Our current implementation uses the *rngcod13* coder (`http://www.compressconsult.com/rangecoder/`). The data array is divided in blocks of 60,000 elements. The frequencies of each value in a block are counted and copied to the output stream of the coder. The data block is then encoded using the calculated frequencies. Given a stream of 256-bit symbols Q_m^ℓ and their frequencies, the coder produces a shorter stream of bits to represent these symbols. The output stream is written in a file, appended with metadata necessary for reconstruction.

3. Numerical Example

This section shows an example application of our data compression tool. We consider a three-dimensional velocity field obtained from a numerical simulation of viscous incompressible flow past a circular cylinder. The fluid domain is a rectangular box with 10 m length, 8 m width and 4 m height. The cylinder has the diameter of 1.894 m. Its axis is positioned vertically at 1.5 m downstream from the inflow boundary of the domain. The fluid has the kinematic viscosity of 0.001 m^2/s and density of 1 kg/m^3. Mean inflow velocity of 1.246 m/s is imposed. The boundary conditions at the exterior faces of the domain are periodic. In addition to that, a vorticity sponge condition is applied over the 48 grid slabs adjacent to the outflow boundary, to avoid the wake re-entering the domain. A salient detail was appended on one side of the cylinder to quickly break the bilateral symmetry of the flow, and random noise was added to provoke three-dimensional instabilities at some point in the simulation. The computational domain is discretized using a uniform Cartesian grid consisting of $960 \times 768 \times 384$ points. The no-slip boundary condition at the surface of the cylinder is modeled using the volume penalization method with the penalization parameter $C_\eta = 5 \cdot 10^{-4}$ s^{-1}. For more information about the numerical method, see (2).

The three components of the velocity field (u_x, u_y, u_z) at time $t = 330$ s were saved, respectively, in three separate files in double precision. Each file occupied 2.2 Gb of hard disk space. The flow configuration and the result of the simulation are visualized in Fig. 1. Grey color shows the cylinder and blue color shows an iso-surface of the Q-criterion. The wake is apparently turbulent with a variety of scales and heterogeneity reminiscent of industrial flows.

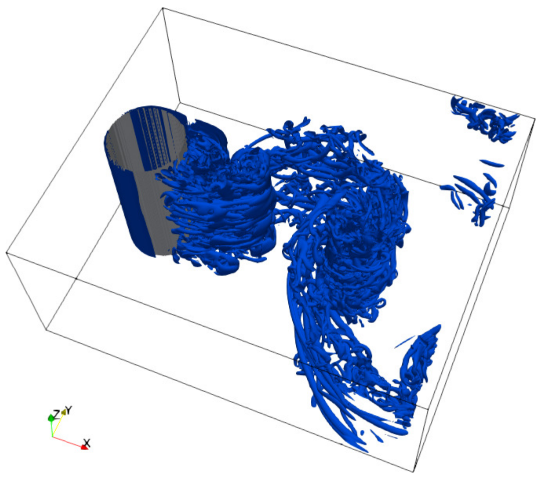

Fig. 1: Visualization of the flow corresponding to the analyzed data set.

To evaluate the performance of the data compression method, each velocity component was first compressed with some given tolerance ε, then reconstructed from the compressed data, and the L^∞ error was calculated between the original and the reconstructed fields. Fig. 2 shows the compressed file size as a function of the error norm, component-wise. The compressed file size is normalized with the original file size. The error norm is divided by the maximum absolute value among all three velocity components. The compression method is equally effective for all velocity components. Perfect reconstruction cannot be reached because of round-off errors, but reconstruction with 10^{-14} error is achieved from a compressed file slightly larger than one half of the original size. It can also be noticed that, if the velocity field were stored in a single precision file, the error would be 100,000 times larger than when storing the same field in a compressed format in an equally large file.

Purple diagonal line shows the gain in compression achieved by discarding the least significant digits, i.e., by degrading the precision of each point value. The difference between this upper bound and the actually measured file size is the combined effect of wavelet transform and entropy coding.

Greater compression ratios can be achieved when precision requirements are less stringent. For instance, in this example one can achieve tenfold reduction in the volume of data if it is stored with 10^{-5} accuracy, which is largely sufficient for flow visualization.

Fig. 3 confirms that the measured L^∞ error is indeed of the same order of magnitude as the tolerance ε requested during the compression. The error control only fails for the smallest values of ε because of round-off errors. Otherwise, the L^∞ error generally falls within 20% of the tolerance ε.

We conclude that the proposed data compression method appears promising for high-performance CFD applications. Our current work in progress is to test it with different kinds of hydrodynamic and meteorological data.

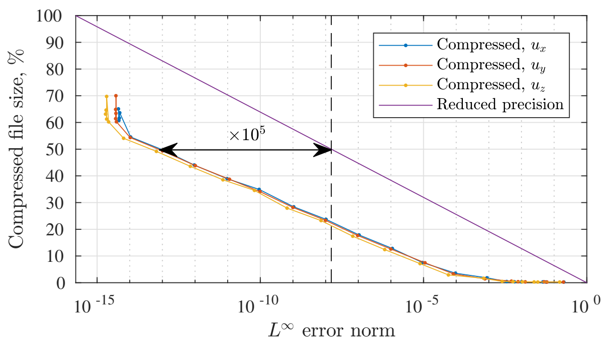

Fig. 2: Compressed file size in per cent of the original file size, versus the L^∞ error norm normalized to $\max(|u_x|, |u_z|, |u_z|)$.

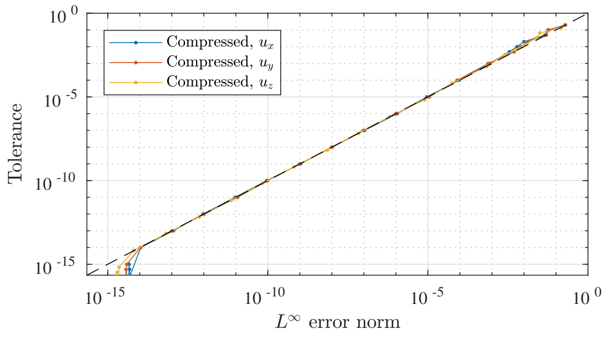

Fig. 3: Tolerance versus the L^∞ error norm, both normalized to $\max(|u_x|, |u_z|, |u_z|)$.

This work is supported by the FLAGSHIP2020, MEXT within the priority study4 (Advancement of meteorological and global environmental predictions utilizing observational "Big Data").

References

(1) Hilbert, M., "Big Data for development: a review of promises and challenges," Dev. Pol. Rev., 34 (2016), pp. 135-174.

(2) Engels, T., Kolomenskiy, D., Schneider, K. and Sesterhenn, J., "FluSI: a novel parallel simulation tool for flapping insect flight using a Fourier method with volume penalization," SIAM J. Sci. Comput., 38 (2016), pp. S3-S24.