The Journal of Open Source Software

fastSF: A parallel code for computing the structure functions of turbulence

Shubhadeep Sadhukhan[1], **Shashwat Bhattacharya**[2], **and Mahendra K. Verma**[1]

1 Department of Physics, Indian Institute of Technology Kanpur, Kanpur 208016, India **2** Department of Mechanical Engineering, Indian Institute of Technology Kanpur 208016, India

DOI: 10.21105/joss.02185

Software

- Review
- Repository
- Archive

Editor: Jed Brown

Reviewers:

- @cpgr
- @iljah

Submitted: 06 February 2020
Published: 10 July 2020

Summary

Turbulence is a complex phenomenon in fluid dynamics involving nonlinear interactions between multiple scales. Structure function is a popular diagnostics tool to study the statistical properties of turbulent flows (Frisch, 1995; Kolmogorov, 1941a, 1941b). Some of the earlier works comprising of such analysis are those of Gotoh, Fukayama, & Nakano (2002), Ishihara, Yokokawa, Itakura, & Uno (2003), and Ishihara & Gotoh (2009) for three-dimensional (3D) hydrodynamic turbulence; Yeung, Donzis, & Sreenivasan (2005) and Ray, Mitra, & Pandit (2008) for passive scalar turbulence; Biferale, Cencini, Lanotte, Sbragaglia, & Toschi (2004) for two-dimensional (2D) hydrodynamic turbulence; and Kunnen et al. (2008), Kaczorowski & Xia (2013), and Bhattacharya, Sadhukhan, Guha, & Verma (2019) for turbulent thermal convection. Structure functions are two-point statistical quantities; thus, an accurate computation of these quantities requires averaging over many points. However, incorporation of a large number of points makes the computations very expensive and challenging. Therefore, we require an efficient parallel code for accurate computation of structure functions. In this paper, we describe the design and validation of the results of `fastSF`, a parallel code to compute the structure functions for a given velocity or scalar field.

`fastSF`, written in C++, is a fast and efficient code that uses vectorization for computing the structure functions. The code employs MPI (Message Passing Interface) parallelization with equal load distribution. The user has a choice on the type (scalar or vector) and the dimensions of the fields to be read by the code, and the range of the orders of the structure functions to be computed. The code writes the computed structure functions to `hdf5` files that can be further processed by the user.

To the best of our knowledge and belief, no other commercial or open-source packages for computing structure functions exist or are available at the moment. Currently, such codes are developed in-house and are not open to the general public. On the other hand, `fastSF` is an open-source package available to all, and thus will be a useful tool for computing and analysing structure functions.

The code uses the following libraries:

1. blitz++ (version 1.0.2)
2. hdf5 (version 1.8.20)
3. h5si (version 1.1.1)
4. yaml-cpp (version 0.3.0)
5. mpich (version 3.3.2)
6. cmake (version 3.16.4)

In the next section, we will briefly discuss the performance and scaling of `fastSF` in a Cray XC40 system.

Velocity and scalar structure functions

We denote the velocity and scalar fields using $\mathbf{u}$ and θ respectively. The velocity difference between any two points $\mathbf{r}$ and $\mathbf{r}+\mathbf{l}$ is $\delta\mathbf{u} = \mathbf{u}(\mathbf{r}+\mathbf{l}) - \mathbf{u}(\mathbf{r})$. The difference in the parallel components of the velocity field along $\mathbf{l}$ is $\delta u_{\parallel} = \delta\mathbf{u}\cdot\hat{\mathbf{l}}$. The corresponding difference in the perpendicular component is $\delta u_{\perp} = |\delta\mathbf{u} - \delta u_{\parallel}\hat{\mathbf{l}}|$. Assuming statistical homogeneity, we define the longitudinal velocity structure functions of order q as

$$S_q^{u_{\parallel}}(\mathbf{l}) = \langle(\delta u_{\parallel})^q\rangle = \langle[\{\mathbf{u}(\mathbf{r}+\mathbf{l}) - \mathbf{u}(\mathbf{r})\}\cdot\hat{\mathbf{l}}]^q\rangle, \qquad (1)$$

and the transverse velocity structure functions of order q as

$$S_q^{u_{\perp}}(\mathbf{l}) = \langle(\delta u_{\perp})^q\rangle = \langle|\delta\mathbf{u} - \delta u_{\parallel}\hat{\mathbf{l}}|^q\rangle. \qquad (2)$$

Here, $\langle\cdot\rangle$ denotes ensemble averaging. Similarly, we can define the scalar structure functions for the scalar field as

$$S_q^{\theta}(\mathbf{l}) = \langle(\delta\theta)^q\rangle = \langle[\theta(\mathbf{r}+\mathbf{l}) - \theta(\mathbf{r})]^q\rangle. \qquad (3)$$

For isotropic turbulence (in addition to being homogeneous), the structure functions become functions of l, where $l = |\mathbf{l}|$. The second-order velocity structure function $S_q^{u_{\parallel}}(l)$ provides an estimate for the energy in the eddies of size l or less (Davidson, 2004).

In the next section, we provide a brief description of the code.

Design of the Code

First we present a sketch of the structure function computation for the velocity structure functions. Typical structure function computations [Eqs (1-3)] in literature involve calculation of the velocity difference using loops over $\mathbf{r}$ and $\mathbf{l}$. These computations require six nested `for` loops for 3D fields that makes the computations very expensive for large grids. In our code, we employ vectorization and loops over only $\mathbf{l}$, thus requiring three loops instead of six for 3D fields. The new algorithm enhances the performance approximately 20 times over the earlier schemes due to vectorization. In the following, we provide the algorithm for structure function computation for a 2D velocity field.

Pseudo-code

Data: Velocity field $\mathbf{u}$ in domain (L_x, L_z); number of processors P.

Procedure:

- Divide $\mathbf{l}$'s among various processors. The process of data division among the processors has been described later in this section.
- For every processor:
  - for $\mathbf{l} = (l_x, l_z)$ assigned to the processor:
    - Compute $\delta\mathbf{u}(l_x, l_z)$ by taking the difference between two points with the same indices in pink and green subdomains as shown in Fig. 1. This feature enables vectorized subtraction operation.
    - $\delta u_{\parallel}(l_x, l_z) = \delta\mathbf{u}\cdot\hat{\mathbf{l}}$ (Vectorized).
    - $\delta u_{\perp}(l_x, l_z) = |\delta\mathbf{u} - \delta u_{\parallel}\hat{\mathbf{l}}|$ (Vectorized).
    - for order q:

- $S_q^{u_\parallel}(l_x, l_z)$ = Average of $\delta u_\parallel^q$ (Vectorized).
- $S_q^{u_\perp}(l_x, l_z)$ = Average of $\delta u_\perp^q$ (Vectorized).
- Send the values of $S_q^{u_\parallel}(l_x, l_z)$, $S_q^{u_\perp}(l_x, l_z)$, q, l_x, and l_z to the master processor.

- The master processor stores $S_q^{u_\parallel}(l_x, l_z)$ and $S_q^{u_\perp}(l_x, l_z)$.
- Stop

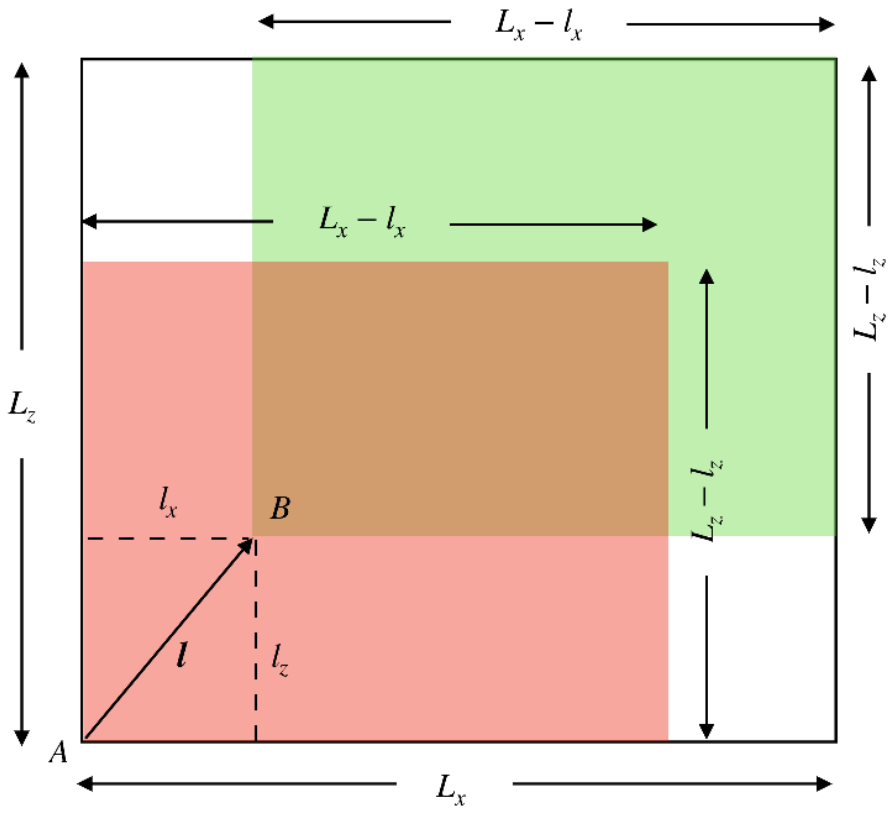

Figure 1: The velocity difference $\delta\mathbf{u}(\mathbf{l})$ is computed by taking the difference between two points with the same indices in the pink and the green subdomains. For example, $\mathbf{u}(\mathbf{l}) - \mathbf{u}(0,0) = \mathbf{u}_B - \mathbf{u}_A$, where B and A are the origins of the green and the pink subdomains. This feature enables vecotrization of the computation.

Since $S_q^u(\mathbf{l})$ is important for intermediate scales (inertial range) only, we vary $\mathbf{l}$ upto half the domain size, that is, upto $(L_x/2, L_z/2)$, to save computational cost. The $\mathbf{l}$'s are divided among MPI processors along x and z directions. Each MPI processor computes the structure functions for the points assigned to it and has access to the entire input data. Thus, we save considerable time that would otherwise be spent on communication between the processors during the calculation of the velocity difference. After computing the structure function for a given $\mathbf{l}$, each processor communicates the result to the master processor, which stores the $S_q^{u_\parallel}(\mathbf{l})$ and $S_q^{u_\perp}(\mathbf{l})$ arrays.

It is clear from Fig. 1 that the sizes of the pink or green subdomains are $(L_x - l_x)(L_z - l_z)$, which are function of $\mathbf{l}$'s. This function decreases with increasing $\mathbf{l}$ leading to larger computational costs for small l and less cost of larger l. Hence, a straightforward division of the domain among the processors along x and z directions will lead to a load imbalance. Therefore, we assign both large and small $\mathbf{l}$'s to each processor to achieve equal load distribution. We illustrate the above idea using the following example.

Consider a one-dimensional domain of size $L = 15$, for which the possible l's are

$$l = \{0, 1, 2, 3...15\}.$$

We need to compute the structure functions for l ranging from 0 to 7. We divide the task among four processors, with two l's assigned to each processor. The following distribution of l's ensures equal load distribution:

$$\begin{aligned}
&\text{Processor 0:} \quad l = \{0, 7\}, \quad \sum(L - l) = (15 - 0) + (15 - 7) = 23,\\
&\text{Processor 1:} \quad l = \{1, 6\}, \quad \sum(L - l) = (15 - 1) + (15 - 6) = 23,\\
&\text{Processor 2:} \quad l = \{2, 5\}, \quad \sum(L - l) = (15 - 2) + (15 - 5) = 23,\\
&\text{Processor 3:} \quad l = \{3, 4\}, \quad \sum(L - l) = (15 - 3) + (15 - 4) = 23.
\end{aligned}$$

Similarly, if two processors are used, then the following distribution results in load balance.

$$\begin{aligned}
&\text{Processor 0:} \quad l = \{0, 7, 2, 5\},\\
&\text{Processor 1:} \quad l = \{1, 6, 3, 4\}.
\end{aligned}$$

This idea of load distribution has been implemented in our program and has been extended to higher dimensions.

Note that for 2D, $l_x > 0$, but l_z can take both positive and negative values. However, for isotropic turbulence, the structure functions for $+l_z$ and $-l_z$ are statistically equal. Therefore, in our computations, we restrict to $l_x > 0$, $l_z > 0$. For anisotropic turbulence, not discussed here, the structure functions will depend on (l_x, l_z) rather than l; our code will be extended to such systems in future.

For 3D turbulence, the structure functions will depend on (l_x, l_y, l_z). We divide the tasks among processors over l_x and l_y as done above for 2D turbulence. The aforementioned algorithm can be easily extended to the 3D case. We employ a similar method for the computation of scalar structure functions as well.

In the next section, we discuss the scaling of our code.

Scaling of `fastSF`

`fastSF` is scalable over many processors due to vectorization and equal load distribution. We demonstrate the scaling of `fastSF` for the third-order longitudinal structure function for an idealized velocity field on a 128^3 grid. For our computation we employ a maximum of 1024 cores. We take the velocity field as

$$\mathbf{u} = \begin{bmatrix} x \\ y \\ z \end{bmatrix}.$$

We perform four runs on a Cray XC40 system (Shaheen II of KAUST) for this problem using a total of 16, 64, 256, and 1024 cores. We used 16 cores per node for each run. In Fig. 2,

we plot the inverse of time taken in seconds versus the number of cores. The best fit curve for these data points yields

$$T^{-1} \sim p^{0.986 \pm 0.002},$$

Thus, the data-points follow $T^{-1} \sim p$ curve to a good approximation. Hence, we conclude that our code exhibits good scaling.

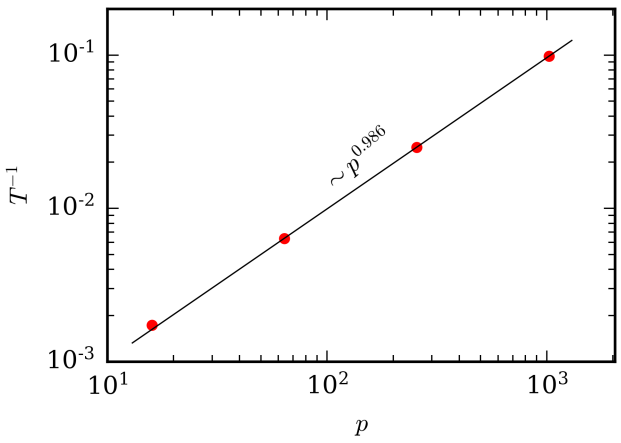

Figure 2: Scaling of `fastSF` for the computation of third-order longitudinal velocity structure function using 16, 64, 256, and 1024 processors of Shaheen II. All the runs were conducted on a 128^3 grid. We observe a linear scaling.

Conclusions

This paper provides a brief description of `fastSF`, an efficient parallel C++ code that computes structure functions for given velocity and scalar fields. This code is shown to be scalable over many processors. We are currently using this code to investigate the structure functions of two-dimensional turbulence with large-scale forcing. An earlier version of the code was used by Bhattacharya et al. (2019) for analyzing the structure functions of turbulent convection. We believe that `fastSF` will be useful to turbulence community because of its efficiency and scalability.

Acknowledgements

We thank R. Samuel, A. Chatterjee, S. Chatterjee, and M. Sharma for useful discussions during the development of `fastSF`. Our computations were performed on Shaheen II at KAUST supercomputing laboratory, Saudi Arabia, under the project k1416.

References

Bhattacharya, S., Sadhukhan, S., Guha, A., & Verma, M. K. (2019). Similarities between the structure functions of thermal convection and hydrodynamic turbulence. *Phys. Fluids*, *31*(11), 115107. doi:10.1063/1.5119905

Biferale, L., Cencini, M., Lanotte, A. S., Sbragaglia, M., & Toschi, F. (2004). Anomalous scaling and universality in hydrodynamic systems with power-law forcing. *New J. Phys.*, *6*, 37. doi:10.1088/1367-2630/6/1/037

Davidson, P. A. (2004). *Turbulence: An Introduction for Scientists and Engineers*. Oxford: Oxford University Press. doi:10.1093/acprof:oso/9780198722588.001.0001

Frisch, U. (1995). *Turbulence: The Legacy of A. N. Kolmogorov*. Cambridge: Cambridge University Press. doi:10.1017/CBO9781139170666

Gotoh, T., Fukayama, D., & Nakano, T. (2002). Velocity field statistics in homogeneous steady turbulence obtained using a high-resolution direct numerical simulation. *Phys. Fluids*, *14*(3), 1065–1081. doi:10.1063/1.1448296

Ishihara, T., & Gotoh, T. (2009). Study of high-reynolds number isotropic turbulence by direct numerical simulation. *Annu. Rev. Fluid Mech.*, *41*(1), 165–180. doi:10.1146/annurev.fluid.010908.165203

Ishihara, T., Yokokawa, M., Itakura, K., & Uno, A. (2003). Energy dissipation rate and energy spectrum in high resolution direct numerical simulations of turbulence in a periodic box. *Phys. Fluids*, *15*(2), L21. doi:10.1063/1.1539855

Kaczorowski, M., & Xia, K.-Q. (2013). Turbulent flow in the bulk of Rayleigh-Bénard convection: small-scale properties in a cubic cell. *J. Fluid Mech.*, *722*, 596–617. doi:10.1017/jfm.2013.74

Kolmogorov, A. N. (1941a). Dissipation of Energy in Locally Isotropic Turbulence. *Dokl Acad Nauk SSSR*, *32*(1), 16–18. doi:10.1098/rspa.1991.0076

Kolmogorov, A. N. (1941b). The local structure of turbulence in incompressible viscous fluid for very large Reynolds numbers. *Dokl Acad Nauk SSSR*, *30*(4), 301–305. doi:10.1098/rspa.1991.0075

Kunnen, R. P. J., Clercx, H. J. H., Geurts, B. J., Bokhoven, L. J. A. van, Akkermans, R. A. D., & Verzicco, R. (2008). Numerical and experimental investigation of structure-function scaling in turbulent Rayleigh-Bénard convection. *Phys. Rev. E*, *77*(1), 016302. doi:10.1103/PhysRevE.77.016302

Ray, S. S., Mitra, D., & Pandit, R. (2008). The universality of dynamic multiscaling in homogeneous, isotropic Navier-Stokes and passive-scalar turbulence. *New J. Phys.*, *10*(3), 033003. doi:10.1088/1367-2630/10/3/033003

Yeung, P. K., Donzis, D. A., & Sreenivasan, K. R. (2005). High-Reynolds-number simulation of turbulent mixing. *Phys. Fluids*, *17*(8), 081703. doi:10.1063/1.2001690