



Challenges in Fluid Flow Simulations Using Exascale Computing

Mahendra K. Verma¹ · Roshan Samuel² · Soumyadeep Chatterjee¹ · Shashwat Bhattacharya² · Ali Asad¹

Received: 27 May 2019 / Accepted: 2 May 2020
© Springer Nature Singapore Pte Ltd 2020

Abstract

In this paper, we briefly discuss the challenges in porting hydrodynamic codes to futuristic exascale HPC systems. In particular, we sketch the computational complexities of finite difference (FD) method, pseudo-spectral method, and fast Fourier transform (FFT). The global data communication among the compute cores brings down the efficiency of pseudo-spectral codes and FFT. A FD solver involves relatively lower data communication. However, an incompressible FD flow solver has a pressure Poisson equation, whose computation in multigrid scheme is quite expensive. Hence, a comparative study between the two sets of solvers on exascale system would be valuable. In this paper, we report a comparative performance analysis between a FD code and a spectral code on a relatively smaller grid using 1024 compute cores of Shaheen II; here, the FD code yields comparable accuracy to the spectral code, but it is relatively slower. The above features need to be retested on much larger grids with many more processors.

Keywords Exascale computing · Computational fluid dynamics · Fast Fourier transform · Finite difference method · Pseudo-spectral method

Introduction

High-performance computing (HPC) or supercomputing is an interdisciplinary area of research. In addition to strong proficiency in the scientific domain and numerical algorithms, scientists and engineers working in HPC need strong programming skills, as well as good knowledge of state-of-the-art computing hardware and software. What makes it even more challenging is the rapid evolution of computer hardware and software technologies in a race towards exascale HPC systems. In this article, we will present the challenges faced by computational fluid dynamicists while using state-of-the-art supercomputers. Here, we explore how some applications could possibly be scaled to

exascale systems. This paper is based on the talk given by one of us, Verma, at the conference “Software Challenges to Exascale Computing (SCEC)” held in Delhi on 13–14 December 2018.

Computational fluid dynamics, CFD in short, is a major field of science and engineering with wide applications in weather predictions and climate modelling; in modelling interiors and atmospheres of stars and planets; in modelling flows in rivers, oceans, and astrophysical objects (in galaxies, black holes); in designing and optimising automobiles and airplanes; in space technology; in petrochemical industry; in engineering appliances such as turbines, engines, etc. [2, 14]. Also, hydrodynamic simulations are used for understanding and modelling turbulence, which remains an unsolved problem till date. These CFD simulations consume a large fraction of computing resources in major HPC systems of the world. Given this, it is important to design large-scale CFD applications that can be deployed on futuristic exascale supercomputers.

The leading methods of CFD are finite difference (FD), finite volume, finite elements, spectral or pseudo-spectral, spectral elements, vortex method, etc., each of which has its advantages and disadvantages [2, 14]. For example, a spectral method is very accurate, but it is useful for simulating flows only in idealised geometries such as cubes, cylinder,

This article is part of the topical collection “Software Challenges to Exascale Computing” guest edited by Amit Majumdar and Ritu Arora.

✉ Mahendra K. Verma
mkv@iitk.ac.in
<http://turbulencehub.org>

¹ Department of Physics, Indian Institute of Technology Kanpur, Kanpur 208016, India

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

and spheres [8, 10]. In addition, its parallel version is inefficient due to `MPI_Alltoall` communications of data. In comparison, finite-difference and finite-volume schemes are less accurate, but they can simulate flows in complex geometries. In addition, finite-difference and finite-volume schemes are more amenable to parallelisation compared to spectral method [14].

Simulation of complex flows, specially in turbulent regime, involves a large number of mesh points, going up to trillions. For example, a spectral simulation on a 8192^3 grid has approximately trillion variables (velocity and pressure fields at the mesh points). A major challenge in HPC is how to design CFD codes for exascale systems that will have millions of processors connected via a network of interconnects. In this paper, we will illustrate several parallelisation strategies for FD and spectral methods, and contrast their performance and limitations.

We illustrate the above methods for incompressible Navier–Stokes equations, which are

$$\frac{\partial \mathbf{u}}{\partial t} + \mathbf{u} \cdot \nabla \mathbf{u} = -\nabla p + \nu \nabla^2 \mathbf{u}, \quad (1)$$

$$\nabla \cdot \mathbf{u} = 0, \quad (2)$$

where $\mathbf{u}(\mathbf{r}, t)$ is the velocity field, $p(\mathbf{r}, t)$ is the pressure field, and ν is the kinematic viscosity. We assume incompressible limit for which the density is constant. In the following two sections, we will describe the parallel complexity of FD and spectral techniques, as well as that of fast Fourier transform (FFT) which takes 70% to 80% of the total time of a spectral run. In FD codes, the pressure Poisson solver is a complex function that takes a significant fraction of the total time. In this paper, we skip many algorithmic details of these methods. The reader is referred to Anderson [2] and Ferziger et al. [14] for details.

Let us review some of the key CFD works connected to spectral and FD methods. There are several FFT libraries available at present. Multicore-based FFTs are FFTW [16], P3DFFT [29], PFFT [30], FFTK [11], hybrid (MPI + OpenMP) FFT [27], and FFT based on Charm++ [23]. There are several GPU-based FFTs as well [12, 26]. McClanahan et al. [26], Czechowski et al. [12], and Ravikumar et al. [31] have studied the communication complexity of 3D FFT and its implications to exascale computing. These libraries have been scaled up to several hundred thousand processors. For examples, FFTK scales reasonably well up to 196,608 cores of Cray XC40 [11], and Ravikumar et al. [31]’s FFT scales up to 18,432 NVIDIA Volta GPUs of *Summit*. Refer to Aseeri et al. [4] for a summary of parallel scaling of the above FFT libraries and some others.

There have been many large-scale pseudo-spectral simulations. Here we list only some of them. In 2002 itself,

Yokokawa et al. [44] performed a turbulence simulation on 4096^3 grid using *Earth Simulator*. Chatterjee et al. [11] and Verma et al. [40] performed simulations of hydrodynamic turbulence and turbulent thermal convection on 4096^3 grid. Yeung et al. [43] performed 8192^3 grid simulation using 262,144 cores of Blue Waters, a Cray XE/XK machine. Ishihara et al. [22] executed a pseudo-spectral simulation on a $12,299^3$ grid. Another notable high-resolution spectral simulation is by Rosenberg et al. [32]. Recently, Ravikumar et al. [31] simulated incompressible Navier–Stokes equations on maximum of $18,432^3$ grid using 3072 nodes (or 18,432 GPUs) of *Summit*.

There are many more numerical simulations using FD and finite-volume methods than those using pseudo-spectral method. A popular FD-based astrophysical code is ZEUS, which is detailed in Stone and Norman [35]. ENZO [9] and SpECTRE [24] are advanced versions of ZEUS, and they employ adaptive mesh refinement (AMR), etc. Some other major codes in this category are by Balsara [6] and Samtaney et al. [33]. Recently, Yang et al. [41] received the 2016 Gordon Bell prize for a numerical simulation of atmospheric dynamics using 10 million cores. Due to lack of space, we apologetically skip many other works that merit mention here.

Many past codes were written in traditional computer languages, such as Fortran and C. However, a significant number of new codes are written in object-oriented languages, such as C++ and Java, and they exploit modern language features—objects, classes, templates, etc. Kale [23] designed a new parallel and efficient object-oriented language called Charm++ using asynchronous many-task (AMT) designs. Several petascale numerical applications—ChaNGa, NAMD, OpenAtom, XPACC, Enzo-P/Cello, SpECTRE—have been written using Charm++. Some of these applications employ migratable-objects programming model and actor execution model. In addition, for fast development and efficiency, it is beneficial to employ efficient libraries, such as Blitz++, HDF5, cmake, etc. We have developed a spectral code TARANG and a FD code SARAS in C++ using many of the above features and libraries. We have also developed a FFT library, FFT Kanpur (FFTK), in C++. In this paper, we present several features and results of these codes.

In this paper, we present a bird’s-eye view of pseudo-spectral and FD methods, and report results of a comparative study of two codes based on these methods. Our preliminary studies, performed on relatively smaller grids using 1024 compute cores, are only illustrative; it needs to be extended to much larger grids using millions of processors. This paper also covers various issues faced by a computational scientist working in the area of CFD.

The outline of the paper is as follows: “[Flow Solvers Based on FD Scheme](#)” section contains a brief discussion on the FD method, while “[Scaling Analysis of a Parallel](#)

“FD Code” section briefs scaling analysis of the FD code SARAS. “Flow Solvers Based on Pseudo-Spectral Scheme” and “Parallel Computation of FFT” sections describe similar issues for the spectral code TARANG and FTK. “Contrasting the Performance of Pseudo-Spectral and FD Methods” section describes results of a comparative study between spectral and FD codes simulating decaying hydrodynamic turbulence. In “Challenges in Implementation of CFD Codes in Exascale Systems” and “Challenges Faced by an Application Scientist for Using Large HPC Systems” sections, we discuss some of the challenges faced in CFD and in developing applications for large-scale HPC systems. We conclude in “Conclusions” section.

In the next two sections, we will describe the broad features of a FD code, and its scaling for moderate grids.

Flow Solvers Based on FD Scheme

In a FD scheme, the real space domain is discretised; the grid points are labelled as (i, j, k) , where i, j , and k are integers. The grid spacing is denoted by $(\Delta x, \Delta y, \Delta z)$, hence, the real space coordinates for the grid point (i, j, k) is $(i\Delta x, j\Delta y, k\Delta z)$. We discretise the field variables at N^3 grid points.¹

The grid points are divided evenly among p compute cores² using pencil decomposition, as shown in Fig. 1. The compute cores themselves are divided equally along the x and y directions. Hence, each compute core has approximately $(N/\sqrt{p}) \times (N/\sqrt{p}) \times N$ points, as shown in Fig. 1. In this figure, $p = p_x p_y$, with $p_x = 4$ and $p_y = 4$, and the core indices vary from 0 to 15.

In Fig. 2, we illustrate how the data are shared among the compute cores. Each compute core shares data, such as the grid point (i', j', k') of Fig. 2b. As a result of the shared data points, each compute core handles slightly more data than $(N/\sqrt{p}) \times (N/\sqrt{p}) \times N$. Sharing of data is important for derivative computation, as we describe below.

The derivatives are approximate in a FD scheme. For example, a formula for $(\partial p/\partial x)_{(i,j,k)}$ in central difference scheme is

$$\left(\frac{\partial p}{\partial x}\right)_{(i,j,k)} = \frac{P_{(i+1,j,k)} - P_{(i-1,j,k)}}{2\Delta x}. \tag{3}$$

¹ This simple arrangement is called *collocated grid*, in contrast to more complex one called *staggered grid* in which the velocity fields are represented at the face centres, and pressure at the centre of the cube. In this section, we assume collocated grid for simplicity.

² In some CFD literature, *compute cores* are referred to as processors. In this paper, we reserve the word *processor* for a CPU that contains many compute cores.

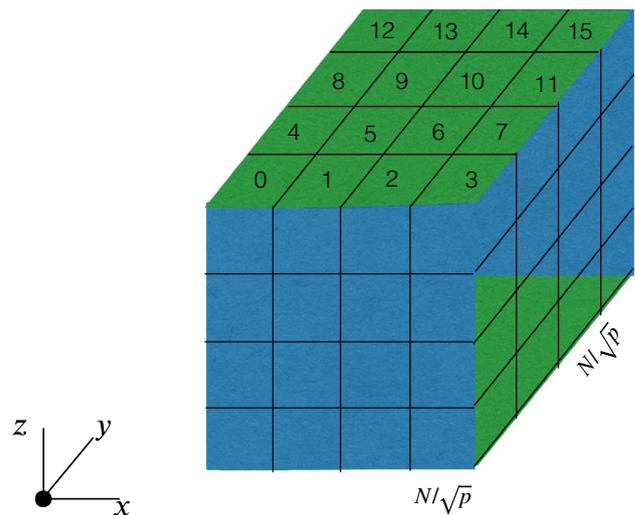


Fig. 1 Pencil decomposition of N^3 grid points among p compute cores. The compute cores, numbered as 0 to 15, are divided equally among x and y axis. Each compute core has $N/\sqrt{p} \times N/\sqrt{p} \times N$ grid points (apart from shared points of Fig. 2)

The above derivatives can be computed by a compute core if both the points, $P_{(i+1,j,k)}$ and $P_{(i-1,j,k)}$, are present in the compute core. However, the derivatives cannot be computed near the edges unless the data are shared among the neighbouring compute cores. This is the reason why some data near the edges need to be shared among the compute cores. The shared grid points are inside the orange-coloured regions of Figs. 2a, b; one of the shared points is illustrated as (i', j', k') in Fig. 2b. After the computation of the derivatives, the velocity field is time advanced. For example, in Euler’s scheme, Eq. (1) is time advanced as

$$\mathbf{u}_{(i,j,k)}(t + \Delta t) = \mathbf{u}_{(i,j,k)}(t) + (\Delta t)\mathbf{R}_{(i,j,k)}(t), \tag{4}$$

where the right-hand-side (RHS) term \mathbf{R} is

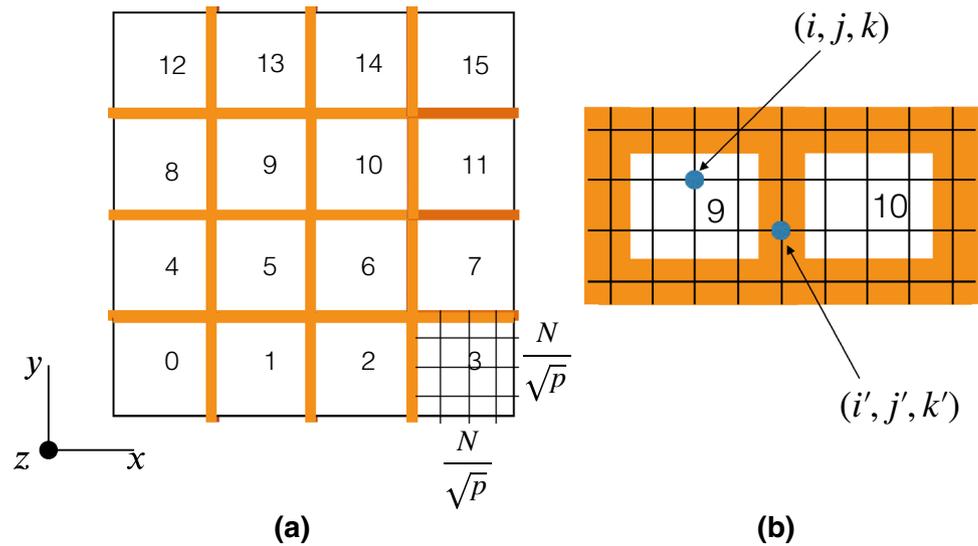
$$\mathbf{R} = -\mathbf{u} \cdot \nabla \mathbf{u} - \nabla p + \nu \nabla^2 \mathbf{u}. \tag{5}$$

After computation of $\mathbf{u}_{(i,j,k)}(t + \Delta t)$, as in Eq. (4), each compute core shares the updated field variables at the four interfaces with four of its neighbouring compute cores. The amount of data to be shared is $O(N^2/\sqrt{p})$, where O stands for “of the order of”. It is easy to see that in the above scheme, the total amount of data to be transmitted at every timestep is

$$D_{\text{FD}} \approx 4 \times 4 \times \frac{N^2}{\sqrt{p}} \times p \approx 16N^2 \sqrt{p}. \tag{6}$$

In the above formula, the factors 4×4 are for the 4 field variables (u_x, u_y, u_z, p) and for the 4 interfaces, respectively. Note however that some of these data exchanges occur within a

Fig. 2 **a** The orange-coloured regions exhibit the shared data among the compute cores (labelled as 0 to 15). **b** A zoomed view of the data decomposition in compute cores 9 and 10. The grid point (i, j, k) belongs to compute core 9 alone, while the grid point (i', j', k') belongs to compute cores 9 and 10



processor (that contains many cores), while some involving across the processors. Therefore, it is best to implement a hybrid version—OpenMP for the cores within a processor, and MPI for the communication across the processors; many FD codes have such arrangements.

For the pencil decomposition shown in Fig. 2, Torus-2 (T2) is the most efficient interconnect for inter-processor communications due to connections among the nearest neighbours. If sufficient number of ports (4 incoming and 4 outgoing) are available at compute nodes, direct connections among the neighbouring processors will minimise the communication time; this arrangement will be optimum for a small HPC cluster.

An implementation of a FD scheme involves many more steps. For example, the pressure for an incompressible flow is solved using a Poisson solver. We refer the reader to Anderson [2] and Ferziger [14] for further details.

In addition to the aforementioned data transmission cost, we also have significant computational cost. The computational cost of the multigrid Poisson solver is $O(N^3)$; the prefactor to N^3 is quite large due to a series of restriction and prolongation operations [36]. Gholami et al. [17] showed that the computation cost of a parallel multigrid solver is comparable to or more than that of a parallel FFT. Thus, the total time taken by a FD-based flow solver depends on many factors, e.g. CPU speed, network bandwidth and topology, etc.

In the following section, we present the scaling analysis of a FD solver on a relatively smaller grid.

Scaling Analysis of a Parallel FD Code

In this section we present a scaling analysis of our FD code SARAS. This code has been developed using the object-oriented features of C++. Our code is a general-purpose

PDE (partial differential equation) solver for various kinds of boundary conditions (no-slip, free-slip, periodic, and mixture of these). The general functions offered in the code make it easy for a new user to add custom boundary and initial conditions.

At present, we employ central difference scheme on a staggered grid [21] for the derivative calculation. The staggered grid implementation, which is more complex than the collocated grid, is not detailed here. The reader is referred to Anderson [2] and Ferziger et al. [14] for details. Also, there is a plan to incorporate higher order schemes and Fourier continuation (see “Challenges in Implementation of CFD Codes in Exascale Systems” section) for the derivative computations. The solver uses semi-implicit method for pressure-linked equations (SIMPLE) algorithm [28] along with Marker and Cell (MAC) method [21] to solve the Navier–Stokes equations. In this scheme, an intermediate (guessed) velocity field, u^* , is computed iteratively using the velocity and pressure field at time t . The pressure correction is computed using this guessed velocity field by solving a Poisson equation. Finally, the velocity correction is calculated from the updated pressure field and added to u^* to obtain $u^{t+\Delta t}$. A multigrid method is used for solving the pressure Poisson equation. We employ Blitz++ library [37] for efficient array operations.

For the scaling analysis, we simulate lid-driven cavity using the above FD solver on Shaheen II with a maximum of 1024 compute cores. Shaheen II of KAUST is a Cray XC40 system with 6174 compute nodes, each containing two Intel Haswell processors with 16 cores each. In total, the system has a total of 197,568 cores and 790 TB of memory. The compute nodes of Shaheen II are connected via Cray’s Aries high-speed network, which is based on a dragonfly topology [20]. The dragonfly is a hierarchical

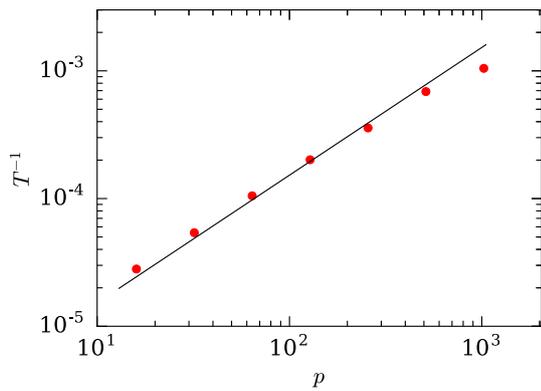


Fig. 3 Scaling of FD solver for simulation of a lid-driven cavity using 16, 32, 64, 128, 256, 512, and 1024 cores of Shaheen II. All the simulations were performed on a 512^3 grid

topology that helps in reduction in number of long links and network diameter [25].

We performed seven simulations on a 512^3 grid using 16, 32, 64, 128, 256, 512, and 1024 cores. Keeping the number of OpenMP processes to be 16 for all the simulations, we varied the number of MPI processes from 1 to 64 with one MPI process per node. The Reynolds number based on lid velocity is 800. We ran the simulations till $t = 0.1$ nondimensional time using a time-step of 0.005.

In Fig. 3, we plot the inverse of total time taken (in seconds) for each run versus the number of cores p used for the simulation. The data points follow the $T^{-1} \sim p$ curve to a good approximation. Thus, we claim that our FD solver exhibits strong scaling. We remark however that the present version of our code is not highly optimised, an exercise that is planned for the near future. We plan to study the scaling of multigrid Poisson solver and SARAS on all the cores of Shaheen II for simulating flows on as large as 8192^3 grid.

There are many FD and finite-volume codes for studying turbulence. Here we list several advanced codes. ZEUS [35] is a second-order accurate FD flow solver. ENZO [9] is an advanced code that employs block-structured adaptive mesh refinement for high spatial and temporal resolution. SpECTRE [24] uses Charm++ for parallel implementation of discontinuation Galerkin method with a task-based parallelism model. SpECTRE shows excellent scaling up to 671,400 cores of Blue Waters. Yang et al.’s atmospheric code [41] too scales up to 10 million cores. Most of the past codes employ second-order space and time discretisation.

In the next section, we discuss flow solvers based on pseudo-spectral scheme.

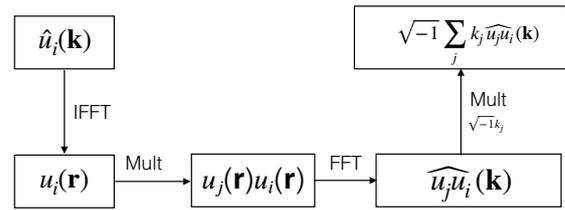


Fig. 4 The computation of the nonlinear term $\sqrt{-1} \sum_j k_j \widehat{u}_j \widehat{u}_i(\mathbf{k})$ in a pseudo-spectral method

Flow Solvers Based on Pseudo-Spectral Scheme

In Fourier space, Eqs. (1, 2) get transformed to

$$\frac{d}{dt} \widehat{u}_i(\mathbf{k}, t) = -\sqrt{-1} k_i \widehat{p}(\mathbf{k}, t) - \widehat{N}_{u,i}(\mathbf{k}) - \nu k^2 \widehat{u}_i(\mathbf{k}), \tag{7}$$

$$k_i \widehat{u}_i(\mathbf{k}, t) = 0, \tag{8}$$

where \mathbf{k} is the wavenumber, \widehat{f} is the Fourier transform of field f , and $\widehat{N}_{u,i}$ is i th component of the nonlinear term:

$$\widehat{N}_{u,i} = \sqrt{-1} \sum_j k_j \widehat{u}_j \widehat{u}_i. \tag{9}$$

The nonlinear term is computed using fast Fourier transform (FFT) to avoid convolution, whose computational complexity is $O(N^6)$ for an N^3 grid. Comparatively, the computation complexity of a FFT is $O(N^3 \log N^3)$. The steps involved in the computation process are as follows (see Fig. 4) [8, 10, 11, 39]:

1. Compute $\mathbf{u}(\mathbf{r})$ from $\widehat{\mathbf{u}}(\mathbf{k})$ using inverse FFT.
2. Compute $u_i(\mathbf{r})u_j(\mathbf{r})$ in real space by multiplying the field variables at the space points.
3. Compute Fourier transform of $u_i(\mathbf{r})u_j(\mathbf{r})$ using forward FFT that yields $(\widehat{u}_i \widehat{u}_j)(\mathbf{k})$.
4. Compute $\sqrt{-1} \sum_j k_j (\widehat{u}_i \widehat{u}_j)(\mathbf{k})$, which is the desired $\widehat{N}_{u,i}(\mathbf{k})$.

Given $\widehat{N}_{u,i}(\mathbf{k})$, the pressure field is easily computed using

$$\widehat{p}(\mathbf{k}) = \sqrt{-1} \frac{1}{k^2} \sum_j k_j \widehat{N}_{u,j}(\mathbf{k}). \tag{10}$$

The Fourier modes are time advanced using Euler or Runge–Kutta schemes. In Euler scheme,

$$\mathbf{u}(\mathbf{k}, t + \Delta t) = \mathbf{u}(\mathbf{k}, t) + (\Delta t)\mathbf{R}(\mathbf{k}, t), \tag{11}$$

where

$$\mathbf{R}(\mathbf{k}, t) = -\mathbf{N}(\mathbf{k}, t) - \sqrt{-1} \mathbf{k} p(\mathbf{k}, t) - \nu k^2 \mathbf{u}(\mathbf{k}, t). \tag{12}$$

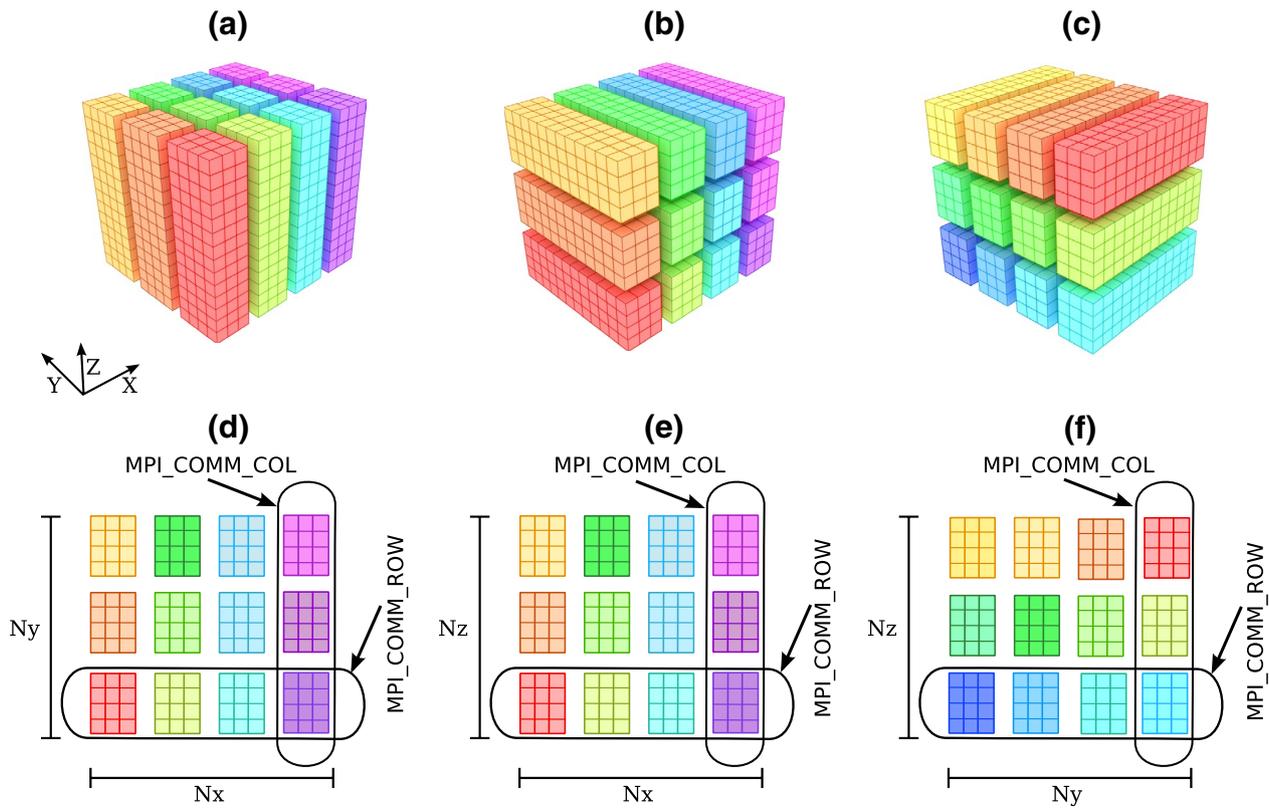


Fig. 5 Illustration of data and operations involved during a Forward FFT transform: **a** real space data, **b** intermediate configuration, **c** data in Fourier space. **d, e, f** Division of cores into p_{row} and p_{col} with $p = p_{row} \times p_{col}$ as in XY , XZ , and YZ projections, respectively. Here

$N_x = N_y = N_z = 12$. In the subfigures **a, d**, $p_{row} = 3$, $p_{col} = 4$, thus each core contains $N_x/p_{col} \times N_y/p_{row} \times N_z = 3 \times 4 \times 12$ data points. From Chatterjee et al. [11]. Reproduced with the permission from Springer Nature

The FFT computations are also used for energy transfer computations [38].

The most complex computation in a spectral method is the FFT, whose parallel implementation is described in the next section.

Parallel Computation of FFT

As in P3DFFT library [29], for large N , we divide the data among p processors using pencil decomposition, as shown in Fig. 1. The forward and inverse FFT of the above data are defined as

$$\hat{f}(k_x, k_y, k_z) = \frac{1}{N^3} \sum_{x,y,z} f(x, y, z) \exp[-\sqrt{-1}(k_x x + k_y y + k_z z)], \tag{13}$$

$$f(x, y, z) = \sum_{k_x, k_y, k_z} \hat{f}(k_x, k_y, k_z) \exp[\sqrt{-1}(k_x x + k_y y + k_z z)]. \tag{14}$$

These operations involve sums along the three directions. Note that a FFT computation involves all the data, and hence it requires global communication among many cores. This is contrary to the FD scheme that involves data transfers among the neighbouring cores.

In Fig. 5, we illustrate the steps involved in forward transform (real space to Fourier space). In this figure, the cores with same X or Y proc-coordinates form a set of communicators—`MPI_COMM_ROW` and `MPI_COMM_COL`. Note that the division of cores in Fig. 5 is slightly different from that of Fig. 1. Now, the steps involved in a FFT operation are [10, 11]:

1. We perform one-dimensional (1D) forward FFT, `r2c` real-to-complex, along the Z -axis for each data column.
2. We perform `MPI_Alltoall` operation among the cores in a `MPI_COMM_COL` communicator to transform the data of Fig. 5a to the intermediate configuration of Fig. 5b. This process is repeated for all `MPI_COMM_COL` communicators.

3. After interprocess communication, we perform forward $c2c$ (complex-to-complex) transform along the Y -axis for each pencil of the array.
4. We perform `MPI_Alltoall` operation among the cores in a `MPI_COMM_ROW` communicator to transform the data of Fig. 5b to the Fourier configuration of Fig. 5c. This process is repeated for all `MPI_COMM_ROW` communicators.
5. In the last step, we perform forward $c2c$ transform along the X -axis for each pencil [see Fig. 5c].

The `MPI_Alltoall` communications are the most expensive operations in the above. Let us estimate the amount of data being communicated in a FFT operation.

Assuming equal division of cores along the X and Y directions, each core has N^3/p data. In `MPI_Alltoall` communication within a communicator, each core sends and receives $\sqrt{p} - 1$ packets of $N^3/(p\sqrt{p})$ data. Hence, within each communicator, the amount of data exchanged is

$$D_0 = \frac{N^3}{p\sqrt{p}} \frac{\sqrt{p}(\sqrt{p} - 1)}{2} \tag{15}$$

Therefore, the total amount of data communicated across \sqrt{p} communicator is

$$D_{PS} \approx \sqrt{p}D_0 \approx N^3 \tag{16}$$

Hence, using Eqs. (6, 16) we deduce that the ratio of data communicated in FFT scheme and FD scheme is

$$\frac{D_{PS}}{D_{FD}} \sim \frac{N}{\sqrt{p}}, \tag{17}$$

which is large when $N \gg p$. Since the communication time for the two methods is proportional to the amount of transmitted data, the ratio of the corresponding communication times is expected to be

$$\frac{T_{PS}^{comm}}{D_{FD}^{comm}} \sim \frac{N}{\sqrt{p}}. \tag{18}$$

Equation (18) is only an estimate. The net communication cost of a FFT operation also includes latency. In addition, the `MPI_Alltoall` communications across distant processors may require expensive multi-hops within an interconnect [12, 26]. Note that Eq. (18) is expected to work for large-grid size. McClanahan et al. [26], Czechowski et al. [12], and Ravikumar et al. [31] performed detailed analysis of the communication performance in a parallel FFT.

The total time is a sum of communication and computation costs. As we show below, communication cost dominates the computation cost in a FFT. In comparison,

FD-based flow solvers have multigrid Poisson equation, which is quite expensive computationally.

FFT computations by various researchers show that data communication among the processors takes much longer than the computation time [12, 29]. Here, we report the performance of FFKT written by Chatterjee et al. [11]. Chatterjee et al. performed FFT on two parallel systems: Blue Gene/P (Shaheen I of KAUST), and Cray XC40 (Shaheen II of KAUST). The specification of Shaheen II is described in “Scaling Analysis of a Parallel FD Code” section. The Blue Gene/P supercomputer, an older system than Cray XC40, had 16 racks with each rack containing 1024 compute nodes. Each node contained a 32-bit 850-MHz quad-core PowerPC processor. Hence, the total number of cores in the system was 65,536. The Blue Gene/P nodes were connected via a three-dimensional Torus interconnect. Note that the above Blue Gene/P system has now been decommissioned.

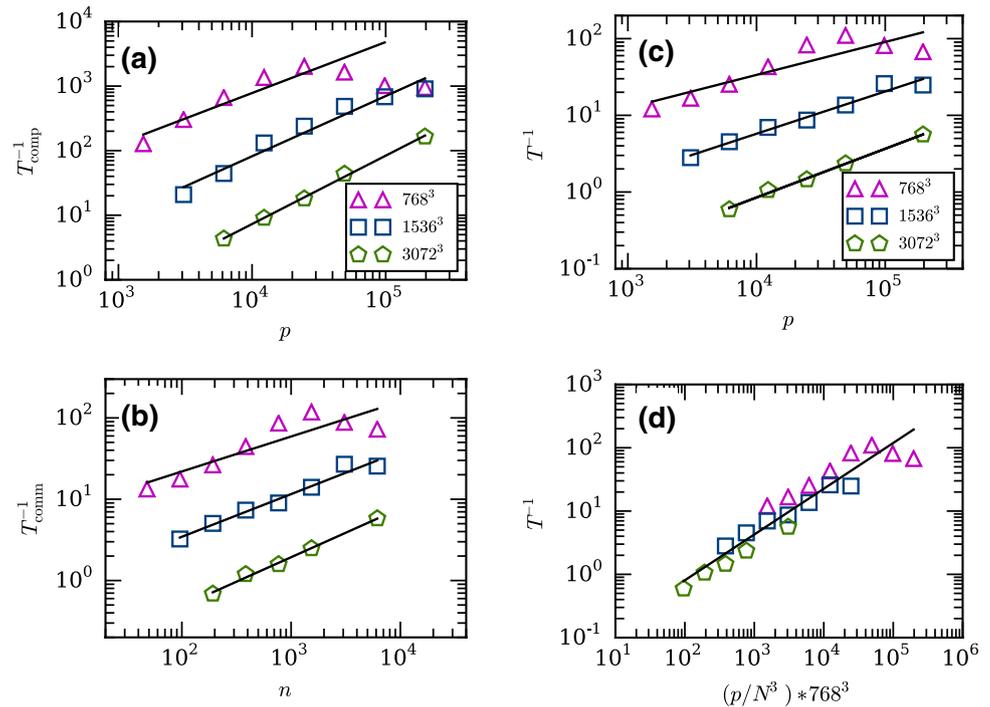
For the runs on Cray XC40 for 768^3 , 1536^3 , and 3072^3 grids, Chatterjee et al. reported the computation time, communication time, and total time taken for a pair of FFT computations (forward and inverse). They employed a maximum of 196,608 cores, which are all the compute cores of Shaheen II. The computation time decreases linearly with number of cores, i.e. $T_{comp}^{-1} \sim p$. Chatterjee et al. characterized the communication time using an exponent γ_2 : $T_{comm}^{-1} \sim n^{\gamma_2}$, where n is the number of nodes. They found the exponent γ_2 for the three grids to be 0.43 ± 0.09 , 0.52 ± 0.04 , and 0.60 ± 0.02 , respectively. Since the communication time dominates the computation time, the exponents for the total time are close to γ_2 . The above scaling is illustrated in Fig. 6. Figure 6c, d show, respectively, the strong and weak scaling of FFT.

In addition to the above scaling, Chatterjee et al. [11] estimated the efficiency of a FFT operation as the ratio of the effective FLOP (floating point operations) rating and the peak FLOP rating.³ The effective FLOP rating was estimated as the ratio of the total number of floating point operations and the total time taken. For the three grids employed, Chatterjee et al. reported the efficiencies to be 0.013, 0.015, and 0.018, respectively. Such a low efficiency is due to the extreme data communication involved in FFT.

Chatterjee et al. [11] carried out similar analysis on maximum of 65,536 cores of Blue Gene/P, which is an older machine compared to Cray XC40. They tested FFT scaling for 2048^3 , 4096^3 , and 8192^3 grids using 1, 2, and 4 processors per node. Surprisingly, Blue Gene/P yields better scaling—higher γ_2 and efficiency—than that on

³ The usual definition of efficiency, $T_{serial}/(pT_{parallel})$, is not suitable for large grids. This is because such large data cannot be accommodated within a single processor, hence, a sequential run for a large grid is impossible.

Fig. 6 Scalings of FFTK on Cray XC40: **a** Plots of T_{comp}^{-1} versus p (number of cores) for 768^3 , 1536^3 , and 3072^3 grids. **b** Plots of T_{comm}^{-1} versus n (number of nodes) using the above convention. The curves follow $T_{\text{comm}}^{-1} \sim n^{\gamma_2}$ with $\gamma_2 = 0.43 \pm 0.09$, 0.52 ± 0.04 , and 0.60 ± 0.02 for the 768^3 , 1536^3 , and 3072^3 grids. **c** Plots of T^{-1} versus p for 768^3 , 1536^3 , and 3072^3 grids exhibit strong scaling. **d** Plots of T^{-1} versus p/N^3 exhibits weak scaling with an exponent of $\gamma = 0.72 \pm 0.03$. Adopted from the figures of Chatterjee et al. [11]



Cray XC40. The peak efficiency of Blue Gene/P is 0.11, which is approximately 6 times the peak efficiency of Cray XC40. See Chatterjee et al. [11] for further details. Note that each core of Cray XC40 is around 100 times faster than that of Blue Gene/P; however, the interconnect speed of Cray XC40 has not improved in a similar proportion. Also, the Torus architecture of Blue Gene/P is better suited for MPI_Alltoall communications than the Dragonfly architecture of Cray XC40. These are the prime factors for the lower efficiency of FFT on Cray XC40 compared to Blue Gene/P. There could be other factors involving cache, memory access, etc., that needs to be examined carefully. A lesson to be learnt from this exercise is that efficiency of a code crucially depends on the speeds and architecture of processors, interconnects, memory, and cache. Hence, these issues are as important as the algorithms designed to solve the problems.

Let us contrast the above two results with those of Earth Simulator that operated in early 2000's. Earth Simulator had 640 vector processors that were connected to each other via a 640×640 crossbar switch [18, 19]. The crossbar interconnect offers an efficient implementation of MPI_Alltoall communication with a single hop. This architecture led to a remarkably efficient implementation of a spectral code on the Earth Simulator. For example, the Earth Simulator achieved an efficiency of 64.9% for a global atmospheric circulation model, which is based on spectral method. Note that on Earth Simulator, the N^3 data were divided into slabs because its total number of processors, 640, was much smaller than $N = 4096$.

The enhanced efficiency of spectral codes on the Earth Simulator indicates a need for specially-designed and novel hardware for FFT. We may generalise the efficient design of Earth Simulator to pencil decomposition, for which the optimum communication requires a fully connected network within each communicator. Such schemes are not available on Torus architecture or on Dragonfly architecture. We plan an approximate implementation of the above on Shaheen II.

The aforementioned discussion indicates that on massively parallel supercomputers, the most expensive operation is data communication, specially for communication-intensive applications like FFT. It is often quoted that the "FLOPS are free, but data communications are expensive". Hence, even though the algorithm of parallel FFT is well known, its efficient implementation requires optimisation on communications (often involving network topology). Such issues need to be explored for exascale computing that consists of millions of processor and complex communication networks.

Before we close our discussion on FFT, we also report the comparative performance of FFTK and P3DFFT libraries. Chatterjee et al. [11] ran both these libraries on Blue Gene/P (Shaheen I) and observed their performance to be nearly the same. This is consistent with the fact that both these libraries employ the same algorithm. PFFT library too has comparable efficiency.

FFTs take 60% to 80% of the total compute time in a spectral code. Hence, the performance of a spectral code is close to that of FFT. This is in addition to input/output of large data, which is typically implemented using parallel

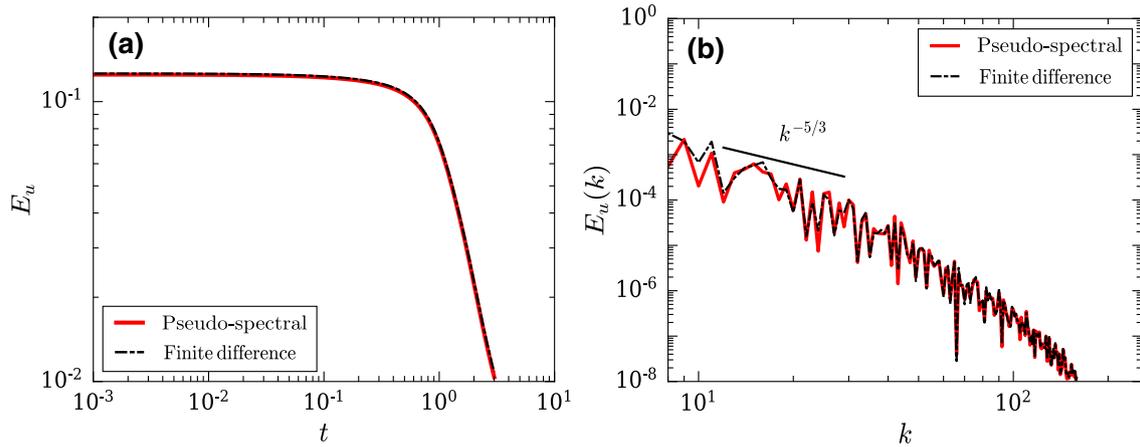


Fig. 7 For the flow simulation of decaying turbulence on a 512^3 grid using 1024 processors with pseudo-spectral code TARANG (red lines) and FD code SARAS (black-dashed lines): **a** plot of the total energy $E_u = \int \mathbf{dr}u^2/2$ versus t , **b** plot of $E_u(k)$ versus k at $t = 1$

I/O, e.g. HDF5 library. Refer to Chatterjee et al. [11] for the details on the scaling analysis of TARANG.

In the next section we compare the performance of the spectral and FD codes.

Contrasting the Performance of Pseudo-Spectral and FD Methods

In this section we simulate decaying turbulence using spectral and FD codes, and compare their results. We employ Taylor–Green vortex [34] as the initial condition for both the runs. That is,

$$\mathbf{u}(x, y, z, t = 0) = u_0 \begin{bmatrix} \sin(2\pi k_0 x) \cos(2\pi k_0 y) \cos(2\pi k_0 z) \\ -\cos(2\pi k_0 x) \sin(2\pi k_0 y) \cos(2\pi k_0 z) \\ 0 \end{bmatrix}, \tag{19}$$

where $u_0 = 1$ and $k_0 = 1$. We perform our simulation in a periodic box of size $1 \times 1 \times 1$ with a grid resolution of 512^3 . These simulations were performed on Shaheen II, which is a Cray XC40 system. We perform both the simulations up to three nondimensional time units; here time is nondimensionalised using L/u_0 , with $L = 1$, as the time scale. The initial Reynolds number of the flow was $Re = 1000$. The pseudo-spectral solver employs fourth-order Runge–Kutta scheme (RK4) for time integration. In “Scaling Analysis of a Parallel FD Code” section we described the time evolution scheme for the FD scheme. We choose constant $dt = 0.001$ for time-stepping both the solvers. Also, we employ 1024 compute cores for both the runs.

The two methods exhibit similar results. The temporal evolution of total kinetic energy ($\int \mathbf{dr}(u^2/2)$), plotted in Fig. 7a, are nearly equal for both the runs, with the maximum difference between the two energies being

approximately 3.4%. The flow profiles of the two runs are quite similar, as is evident from the density plots of Fig. 8 that illustrates the vertical vorticities (ω_z) of the two runs at the horizontal mid-plane at $t = 0$ and $t = 1$. We also compute the energy spectra for the two runs at different times and find them to be approximately equal, thus illustrating similar multiscale evolution of the flow fields for the two runs. See Fig. 7b for an illustration of the energy spectra of the two runs at $t = 1$. Interestingly, for both the runs, the energy spectra in the inertial range are closer to Kolmogorov’s $k^{-5/3}$ prediction.

The above results indicate that the spectral and FD schemes yield similar results even though the FD method employs lower-order schemes for space and time discretisation. This is an encouraging result considering that the derivative calculation by spectral method is much more accurate than the FD method [2, 14]. It may be possible that the spectral and FD flow solvers converge to the solution, at least for smooth flows. Possibly, the flow with Reynolds number of 1000 is resolved quite well by the FD scheme. We need to investigate this issue in detail using simulations on larger grids.

On Shaheen II, the spectral and FD codes took approximately 2803 and 11,657 s, respectively. Thus, the FD scheme is approximately four times slower than the spectral method. One would expect the FD scheme to be faster than the spectral method due to communication issues. However, in the FD scheme, the multigrid pressure Poisson solver involves a series of restriction and prolongation operations. In contrast, FFT provides the pressure directly. The larger time taken by the FD solver may be due to these reasons. Resolution of these issues requires detailed analysis (communication and computation) of the two flow solvers, as well as accurate time complexity computation of Poisson solver. We plan to perform these computations in future. Note that Eq. (18)

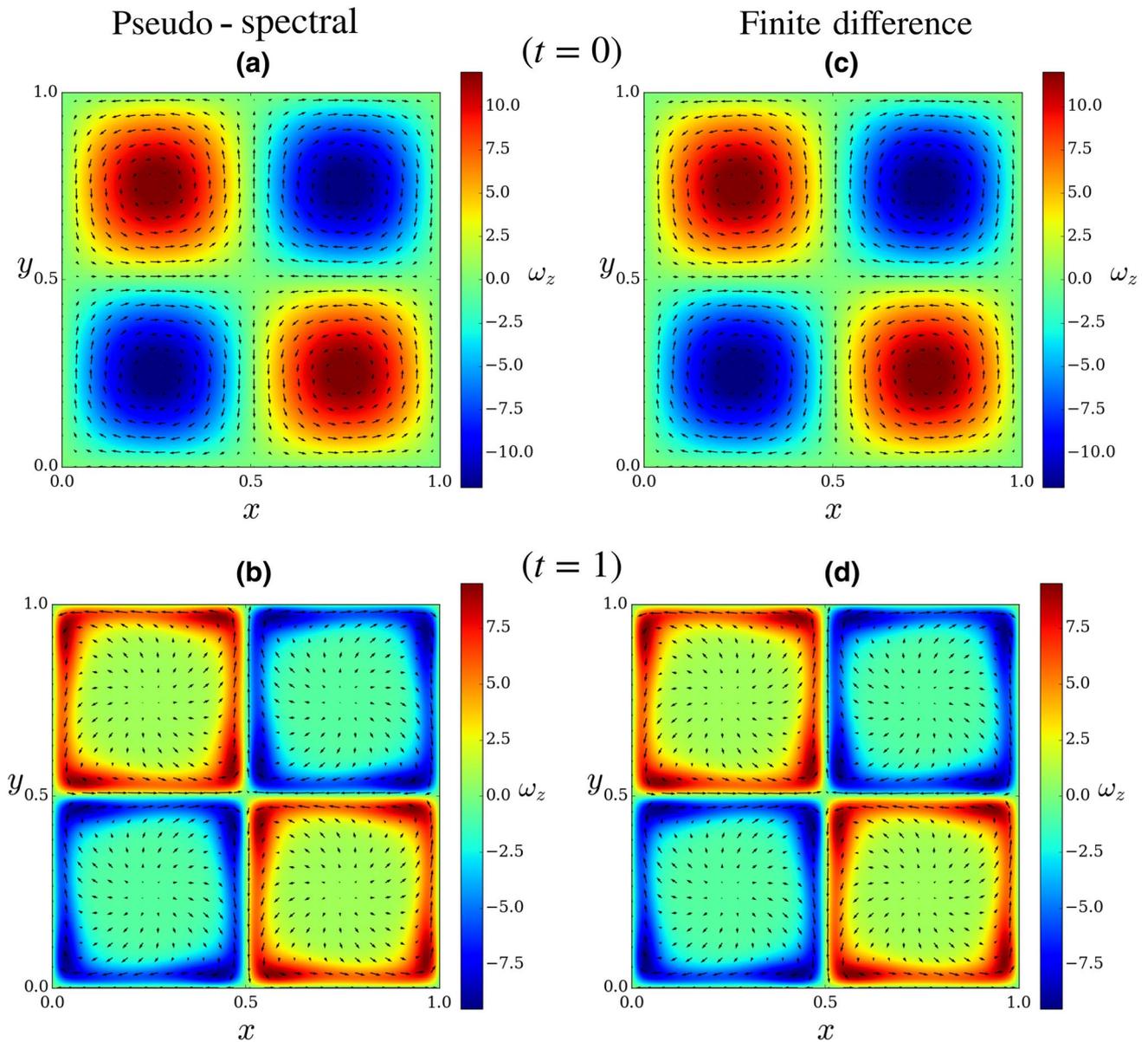


Fig. 8 For the flow simulation of decaying turbulence on a 512^3 grid using 1024 processors, vector plots of the velocity field and density plots of the vertical vorticity fields (ω_z) for the horizontal mid plane at $z = 1/2$: for the data from **a, b** pseudo-spectral code TARANG,

and **c, d** FD code SARAS at $t = 0$ and $t = 1$. These plots and Fig. 7 show the good agreement between the pseudo-spectral and FD methods

only reflects the data transfer complexity, that too, for large grids.

There are some past works that compare the above two methods. For example, Fornberg [15] simulated elastic wave equation using FD and spectral methods. He showed that for the same grid, spectral method provides much more accurate solution than the FD scheme, but the spectral method takes longer time than the FD method for the same grid. Note however that incompressible flow solvers take longer time than the elastic wave simulations due to the pressure Poisson solver. We need to analyze the time and space complexity

of incompressible and compressible flow solvers, including the Poisson solver. For the same, we will compare the performance of SARAS with other flow solvers, such as ENZO, OpenFOAM, etc.

In the next two sections, we present the challenges in CFD and in writing software for exascale systems.

Challenges in Implementation of CFD Codes in Exascale Systems

In previous sections, we summarised the complexities of FD and spectral codes, as well as that of FFT. Using these examples we can conclude that in modern supercomputers, communication across interconnect is the one of the leading bottlenecks for the efficiency of CFD codes. For flow simulations on a smaller grid with smaller number of cores, both spectral and FD codes yield somewhat similar results. Here, the FD code is slower than the spectral code. However, for large grids with larger number of cores, a FD implementation may become more efficient than a spectral one. As described earlier, a FD code requires communication of much smaller dataset, that too among neighbouring processors. In comparison, the `MPI_Alltoall` communications in a spectral code requires transfer of much larger datasets among distant processors. These communications may involve multi-hops within an interconnect. However, the pressure Poisson solver could be a bottleneck for the FD codes. Due to these competing performance bottlenecks for the two methods, we need to perform comparative studies of these schemes on large grids with many processors.

Spectral simulations could be performed efficiently on Earth Simulator due to the crossbar architecture of its interconnect. Modern parallel computers do not allow such possibilities because the crossbar architecture requires enormous connectivity (n^2 for n nodes). Note however that modern compute nodes offer 4 to 8 processors, each having up to 64 cores. A fully connected network for limited number of such compute nodes may offer large efficiency for a FFT.

Recent spectral simulations employ a fraction of million processors. However, to the best of our knowledge, the scaling studies on FFT have been performed up to maximum of 196,608 processors [11]; in this study, the scaling curves tend to saturate near 196,608 cores. Though FFT implementation on multi-GPUs remains a major challenge due to communication issues, there are significant successes in this front [12, 26, 31]. The upcoming HPC system Fugaku that hosts 150,000 nodes with 48/52 core CPUs connected via Tofu-D 6D torus network yielding 60 Petabps injection bandwidth could be one of the ideal platforms for highly efficient FFT. Since a FFT involves communication data size to be of the order N^3 , 60 Petabps injection bandwidth can facilitate efficient communication for $(10^5)^3$ grid simulation, which is very significant. Note that Fugaku is projected to be one of the first exascale HPC systems.

FD and finite-volume schemes scale up to millions of cores. For example, Yang et al. [41] ran an atmospheric

dynamics code on 10 million cores. The efficiencies of the explicit and implicit versions of their code are approximately 100% and 52%, respectively, which are much higher than those of spectral codes. As described earlier, higher efficiency for the above FD code is due to its lower communication complexity. Hence, FD and finite-volume schemes may be good candidates for exascale systems. As described in “Flow Solvers Based on FD Scheme” section, Torus interconnect could provide an optimum data transfer for FD codes.

Spectral elements [13] and Fourier continuation [1] offer promises for exascale computing. These schemes provide flexibility of finite-element/FD methods, as well as spectral accuracy. In Fourier continuation, the real space domain (within a processor) is extended so as to make it periodic. After this, accurate derivatives are computed by the respective processor using FFT, as in “Flow Solvers Based on Pseudo-Spectral Scheme” and “Parallel Computation of FFT” sections. Since these derivatives are computed using partial data (within the processor), they are not as accurate as those of pseudo-spectral method. But, there is an enormous saving in communication costs. In a spectral-element method, the derivatives are computed using polynomials. Thus, codes based on spectral elements and Fourier continuation could scale well in exascale systems. Higher-order FD schemes, including compact schemes, yield reasonably accurate results that are comparable to those from spectral codes [7]. It is however claimed that for the same accuracy, the resolution required in a FD code is more than that in a spectral code [15]. Note however that our comparative study of turbulence simulations indicate that FD codes yield similar accuracy as the spectral code, possibly due to the role played by iterations in Poisson solver, or due to sufficient resolution of the FD code; These issues need to be tested in future.

We also remark that shared-memory HPC systems with a hybrid implementation (OpenMP+MPI) provide interesting possibilities for efficient implementation of both spectral and FD codes. Several existing codes including SARAS and that by Rosenberg et al. [32] employ such ideas.

Large-scale numerical simulations generate big data, extending up to tens and hundreds of terabytes. Exascale computing will generate even larger data. Hence, input/output and processing of big data is an enormous challenge [3]. Parallel I/O libraries, such as HDF5 or Hadoop, are employed for handling large data. There are significant efforts for in-situ data processing, for example, for generating images and videos in real time, using libraries such as Paraview and Visit.

In the next section we detail some general computing issues in creating new scientific applications for large HPC systems.

Challenges Faced by an Application Scientist for Using Large HPC Systems

In this section, we will describe some of the difficulties faced by an application scientist in HPC. The issues involved in modern HPC are quite complex requiring expertise in software, hardware, and in application domain. On top of them, one needs to keep track of rapid development in hardware and software technologies, without which it is impossible to implement the applications efficiently.

New processors have large number of cores and large caches. In addition, modern interconnects are much faster than those of previous generation. Exploitation of the above features requires hybrid codes—OpenMP implementation for the internal cores, and MPI implementation across nodes. As described in this paper, appropriate network architecture and job scripts are required for efficient implementation of FFT and FD codes.

Regarding software, large codes need to be structured and flexible. They need to be readable and easy to modify so that new users/developers can expand the codes to newer applications. For the same, an application scientist needs to learn object-oriented programming. Also, the features of parallel tools such as MPI and OpenMP are changing rapidly. In addition, new and efficient language tools and libraries, such as Charm++, are being created constantly. Porting the codes to multi-GPUs and implementation of parallel I/O and version control are quite complex.

Lastly, some computational algorithms (e.g. optimum cache usage) are sometimes too complex for an application scientist. It is difficult to find physics and engineering students who are skilled in these areas. On top of it, there is pressure to deliver results in science and engineering domain. Hence, one does not get sufficiently long time for developing efficient and robust codes.

The above difficulties could be alleviated in a cross-disciplinary group having computational and application scientists. Such groups are being formed these days, and we hope that they will become common in near future.

We conclude in the next section.

Conclusions

Futuristic exascale computers offer immense opportunities as well as challenges to application scientists. In this paper, we present computational challenges in computational fluid dynamics (CFD). We present two generic schemes, FD and pseudo-spectral; the latter involves FFT. For FFT and pseudo-spectral codes, inter-node communication is

the biggest bottleneck in a generic HPC system. Faster processors and relatively slower interconnects bring down the efficiency of a FFT library. Higher efficiency of FFT on Earth Simulator indicates that a fully connected network could yield an enhanced FFT; such network however would be very expensive. Quantum Fourier Transform may offer an alternative, but its discussion is beyond the scope of this paper [42]

In comparison, FD codes require communication of smaller datasets across neighbouring processors. Incompressible flow solvers however require pressure Poisson solver, which is quite expensive computationally. Due to less communication cost, such codes tend to scale well on large number of processors. Yang et al. [41] demonstrated how a finite-volume based atmospheric physics code could be ported to 10 million cores. For better efficiency of such codes, it is important that the processors communicate among themselves in a single hop, or in least possible hops. To test the efficacy of flow solvers, we need to perform comparative studies of the two methods on very large grids using millions of processors.

In this paper, we performed a preliminary set of runs to compare the performance of FD and spectral codes. We performed these tests on a relatively smaller grid (512^3) with smaller number of processors (up to 1024) and observed that the accuracy of the FD code is comparable to that of spectral code. For such runs, the FD code takes longer than the spectral code. However, we need to perform very large resolution runs using many processors to ascertain this observation. In addition, we need to scale the Poisson solver that takes approximately half of the total time taken by a FD code. Such simulations and scaling analysis will be performed in near future. We will also perform comparative performance analysis of SARAS, OpenFOAM, ENZO-P/Cello, SpECTRE, etc.

A complex application involves many subsystems. For example, weather prediction codes have the following components: atmosphere, oceans, land, ice, etc. [5]. For such codes, it is advisable to simulate the subsystems in different sets of processors, and then communicate the data among the subsystems. Such software architecture would be robust, as well as less communication intensive. Such codes too will be suitable for exascale systems.

Finally, there are design and documentation issues for large codes. All the above concerns need to be kept in mind while developing large-scale CFD codes for exascale systems.

Acknowledgements The authors thank all the co-developers of FFTK, TARANG, and SARAS. Some of the key contributors to the codes are Anando Chatterjee, Ravi Samtaney, Fahad Anwer, Gaurav Gautam, Abhishek Kumar, Mani Chandra, Akash Anand, and Awanish Tiwari. In addition, we thank Akash Anand, Samar Aseeri, Rooh Khurram, Bilel Hadri, V. Balaji, and Preeti Malakar for discussion and ideas; and

to Ritu Arora, Venkatesh Shenoy, and Amitava Majumdar for organizing wonderful conference “Software Challenges to Exascale Computing (SCEC)”.

Funding This study was funded by research Grants 6104-1 from Indo-French centre (CEFIPRA), and STC/PHY/2018037 from Indian Space Research Organization. Our numerical simulations were performed on Blue Gene/P (Shaheen I) and Cray XC40 (Shaheen II) of KAUST supercomputing laboratory, Saudi Arabia, through Projects k1052 and k1416.

Compliance with Ethical Standards

Conflict of interest The authors declare that they have no conflict of interest.

References

- Albin N, Bruno OP. A spectral FC solver for the compressible Navier–Stokes equations in general domains I: explicit time-stepping. *J Comput Phys*. 2011;230(16):6248–70.
- Anderson JD. *Computational Fluid Dynamics: The Basics with Applications*. New York: McGraw-Hill; 1995.
- Arora R. *Conquering Big Data with High Performance Computing*. Berlin: Springer; 2016.
- Aseeri S et al. Solving the Klein–Gordon equation using Fourier spectral methods: a benchmark test for computer performance. In: *Symposium on High Performance Computing*. Society for Computer Simulation International 2015; p. 182–91.
- Balaji V. Scientific computing in the age of complexity. *XRDS*. 2013;19(3):12–7.
- Balsara DS, Balsara D, Jongsoo K, Mac Low MM, Mathews GJ. Amplification of interstellar magnetic fields by supernova-driven turbulence. *ApJ*. 2004;617:339–49.
- Balsara DS. Higher-order accurate space-time schemes for computational astrophysics—Part I: finite volume methods. *Liv Rev Comput Astrophys*. 2017;3(1):1–138.
- Boyd JP. *Chebyshev and Fourier spectral methods*. second revised ed. New York: Dover Publications; 2003.
- Bryan GL, Norman ML, O’Shea BW, Abel T, Wise JH, Turk MJ, Reynolds DR, Collins DC, Wang P, Skillman SW, Smith B, Harkness RP, Bordner J, Kim J, Kuhlen M, Xu H, Goldbaum N, Hummels C, Kritsuk AG, Tasker E, Skory S, Simpson CM, Hahn O, Oishi JS, So GC, Zhao F, Cen R, Li Y, The Enzo Collaboration. ENZO: an adaptive mesh refinement code for astrophysics. *ApJS*. 2014;211(2):19–52.
- Canuto C, Hussaini MY, Quarteroni A, Zang TA. *Spectral methods in fluid dynamics*. Berlin: Springer; 1988.
- Chatterjee AG, Verma MK, Kumar A, Samtaney R, Hadri B, Khurram R. Scaling of a fast Fourier transform and a pseudo-spectral fluid solver up to 196608 cores. *J Parallel Distrib Comput*. 2018;113:77–91.
- Czechowski K, Battagliano C, McClanahan C, Iyer K, Yeung PK, Vuduc R. On the communication complexity of 3D FFTs and its implications for Exascale. In: *Proceedings of the 26th ACM international conference on Supercomputing*. ACM, New York, 2012; p. 205–14.
- Deville M, Fischer PF, Mund EH. *High-order methods for incompressible fluid flow*. Cambridge: Cambridge University Press; 2004.
- Ferziger JH, Peric M. *Computational methods for fluid dynamics*. 3rd ed. Berlin: Springer; 2001.
- Fornberg B. The pseudospectral method: comparisons with finite difference for the elastic wave equation. *Geophysics*. 1987;52:483–501.
- Frigo M, Johnson SG. The design and implementation of FFTW3. *Proc IEEE*. 2005;93(2):216–31.
- Gholami A, Malhotra D, Sundar H, Biros G. FFT, FMM, or multigrid? A comparative study of state-of-the-art poisson solvers for uniform and nonuniform grids in the unit cube. *SIAM J Sci Comput*. 2016;38(3):C280–306.
- Habata S, Umezawa K, Yokokawa M, Kitawaki S. Hardware system of the Earth Simulator. *Parallel Comput*. 2004;30(12):1287–313.
- Habata S, Yokokawa M, Kitawaki S. The earth simulator system. *NEC Res Dev*. 2003;44:21–6.
- Hadri B, Kortas S, Feki S, Khurram R, Newby G. Overview of the KAUST’s Cray X40 System—Shaheen II. In: *CUG2015 Proceedings 2015*.
- Harlow FH, Welch JE. Numerical calculation of time-dependent viscous incompressible flow of fluid with free surface. *Phys Fluids*. 1965;8(12):2182–9.
- Ishihara T, Morishita K, Yokokawa M, Uno A, Kaneda Y. Energy spectrum in high-resolution direct numerical simulations of turbulence. *Phys Rev Fluids*. 2016;1(8):9–299.
- Kale LV, Krishnan S. Charm++: a portable concurrent object oriented system based on c++. In: *OOPSLA*. vol 93. Citeseer, 1993; p. 91–108.
- Kidder LE, Field SE, Foucart F, Schnetter E, Teukolsky SA, Bohn A, Deppe N, Diener P, Hébert F, Lippuner J, Miller J, Ott CD, Scheel MA, Vincent T. SpECTRE: a task-based discontinuous Galerkin code for relativistic astrophysics. *J Comput Phys*. 2017;335:84–114.
- Kim J, Dally WJ, Scott S, Abts D. Technology-driven, highly-scalable dragonfly topology. In: *2008 international symposium on computer architecture*. IEEE 2008, p. 77–88.
- McClanahan C, Czechowski K, Battagliano C, Iyer K, Yeung PK, Vuduc R. Prospects for scalable 3d ffts on heterogeneous exascale systems. In: *ACM/IEEE conference on supercomputing*, SC 2011.
- Mininni PD, Rosenberg DL, Reddy R, Pouquet AG. A hybrid MPI-OpenMP scheme for scalable parallel pseudospectral computations for fluid turbulence. *Parallel Comput*. 2011;37(6–7):316–26.
- Patankar SV. *Numerical heat transfer and fluid flow*. London: Taylor and Francis; 1980.
- Pekurovsky D. P3DFFT: a framework for parallel computations of fourier transforms in three dimensions. *SIAM J Sci Comput*. 2012;34(4):C192–209.
- Pippig M, Potts D. Scaling parallel fast Fourier transform on bluegene/p. In: *Jülich BlueGeneP Scaling Workshop*. Jülich BlueGene/P Scaling Workshop; 2010.
- Ravikumar K, Appelhans D, Yeung PK. GPU acceleration of extreme scale pseudo-spectral simulations of turbulence using asynchronism. In: *SC ’19*. New York: ACM; 2019, p. 1–22.
- Rosenberg DL, Pouquet AG, Marino R, Mininni PD. Evidence for Bolgiano–Obukhov scaling in rotating stratified turbulence using high-resolution direct numerical simulations. *Phys Fluids*. 2015;27(5):055105.
- Samtaney R, Pullin DI, Kosović B. Direct numerical simulation of decaying compressible turbulence and shocklet statistics. *Phys Fluids*. 2001;13:1415–30.
- Schranner F, Hu X, Adams N. Long-time evolution of the incompressible three-dimensional Taylor–Green vortex at very high Reynolds number. In: *Proceedings of the eighth international symposium on turbulence and shear flow phenomena (TSFP-8)*, Poitiers, France; Aug 2013.
- Stone JM, Norman ML. ZEUS-2D: a radiation magnetohydrodynamics code for astrophysical flows in two space dimensions.

2. The magnetohydrodynamic algorithms and tests. *ApJS*. 1992;80:791.
36. Trottenberg U, Oosterlee CW, Schüller A. *Multigrid*. San Diego: Academic Press; 2001.
37. Veldhuizen TL. Arrays in blitz++. In: Caromel D, Oldehoeft RR, Tholburn M, editors. *Computing in object-oriented parallel environments*. Berlin: Springer; 1998. p. 223–30.
38. Verma MK. *Energy transfers in fluid flows: multiscale and spectral perspectives*. Cambridge: Cambridge University Press; 2019.
39. Verma MK, Chatterjee AG, Yadav RK, Paul S, Chandra M, Samtaney R. Benchmarking and scaling studies of pseudospectral code Tarang for turbulence simulations. *Pramana J Phys*. 2013;81(4):617–29.
40. Verma MK, Kumar A, Pandey A. Phenomenology of buoyancy-driven turbulence: recent results. *New J Phys*. 2017;19:025012.
41. Yang C, Xue W, Fu H, You H, Wang X, Ao Y, Liu F, Gan L, Xu P, Wang L, Yang G, Zheng W. 10M-core scalable fully-implicit solver for nonhydrostatic atmospheric dynamics. In: *International conference for high performance computing, networking, storage and analysis*. IEEE Press, IEEE; Sep 2016.
42. Yanofsky NS, Mannucci MA. *Quantum computing for computer scientists*. Cambridge: Cambridge University Press; 2008.
43. Yeung PK, Zhai XM, Sreenivasan KR. Extreme events in computational turbulence. *PNAS*. 2015;112(41):12633.
44. Yokokawa M, Itakura K, Uno A, Ishihara T. 16.4-Tflops direct numerical simulation of turbulence by a fourier spectral method on the earth simulator. In: *ACM/IEEE 2002 conference*. IEEE; 2002.

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.