Object-oriented Pseudo-spectral code TARANG for turbulence simulation

Mahendra K. Verma¹

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

(Dated: January 14, 2013)

In this paper we describe the design and implementation of TARANG, a pseudospectral code to simulate turbulent flows in fluids, magnetohydrodynamics (MHD), convection, passive scalar, etc. We use the object-oriented features of C++ to abstract operations involved in the simulation. TARANG has been validated and used for solving problems in convection and MHD.

INTRODUCTION

Turbulence is one of the most challenging and unsolved problems of physics. Theoretical or exact analysis of turbulence has been rare due to the complex nonlinearities present in the equations. Therefore, major attempts to understand turbulence has been through experiments and numerical simulations. Over time very powerful supercomputers and software tools have emerged, that have propelled the computational capabilities of turbulent flows to unimaginable heights.

Some of the popular schemes to solve fluid flows are finite difference, finite element, finite volume, spectral elements, pseudo-spectral, vortex method, etc. The pseudo-spectral method [1] is most accurate among them, and it employed for studying small-scale turbulence. In the present paper we will describe the design and capability of a pseudo-spectral code named “TARANG”, which has been developed by our group. TARANG is a Sanskrit word that means “waves”.

TARANG is an object-oriented parallel pseudo-spectral code that can simulate flows in fluids, magnetofluids, convection, magnetohconvection etc. The convective flow module can also be used to solve Rayleigh-Taylor instability and turbulence, stratified flows, non-Boussinesq convection, and passive scalars, etc. We designed TARANG in an object-oriented fashion for generality and easy adoptability to solve varied problems. We make use of fast libraries, FFTW (Fastest Fourier Transform in the West) and blitz++ for efficiency. The code is neatly segregated into libraries and src directory. The fluid, magnetohydrodynamics (MHD), convection solvers etc. make use of the libraries, and they are arranged in the src directory.

We will describe the design issues and various features of TARANG in the following sections.

DESIGN AND IMPLEMENTATION ISSUES OF TARANG

In TARANG we use the object-oriented features of C++ to design general purpose libraries to create solvers for the incompressible fluid flows. For example, a library function Compute_nlin computes \((\mathbf{u} \cdot \nabla)\mathbf{u}\) for all basis functions. A solver containing many complex features and boundary conditions, and more field variables are easily constructed using these functions. Main features including the class structures of TARANG are described below.

External libraries

The handling of arrays and mathematical functions is computationally slow in C++. Fortunately, several efficient C++ libraries are available to perform the above tasks. We chose blitz++ for TARANG since it handles multidimensional arrays in a nice and succinct manner. The other libraries with similar functions are boost, ndarrays, and eigen, but presently we continue our development with blitz++.

Pseudospectral codes use FFT (Fast Fourier Transforms) heavily. In a typical code, approximately 80% of the computational time is spent of FFT. FFTW is the most popular and the most efficient parallel FFT library available today, therefore we use FFTW in our code.

Basis functions

A pseudo-spectral code uses basis functions to expand the real-space functions. The choice of the basis functions depend critically on the boundary conditions. \(\exp(ik \cdot x)\), where \(k, x\) are the wavenumber and real-space coordinates respectively, is the natural choice for periodic boundary conditions. Convection, channel flows etc. however involves walls. All the components of the velocity fields at the wall must vanish for the no-slip boundary condition. Chebyshev and Legendre polynomials are used to expand such functions. For the free-slip boundary condition, the velocity field perpendicular to the wall, and the perpendicular gradient of the horizontal velocity components are zero. Sine and cosine functions are obvious and simple choices for such simulations. Spherical harmonics are natural choice for spherical simulations.

TARANG focusses on basis-independent libraries, so we have designed the basis functions in a modular fashion. At present TARANG has FOUR and SCFT (sin/cos-Fourier) basis functions that can simulate flows...
in a box geometry under periodic boundary conditions (all directions) and free-slip boundary conditions along \(x\), and periodic along \(y\) and \(z\) directions. The SCFT basis functions with appropriate modifications has been also used to simulate flows with free-slip boundary conditions along all the directions. The no-slip boundary conditions has been successfully tested for channel flow, but this module will be integrated with TARANG soon. The spherical geometry and cylindrical geometry will be implemented in future.

Primary functions related to the basis functions are forward transform (transformation from the real space to Fourier space), inverse transform (transformation from the Fourier space to the real space), computation of energy spectrum etc. We use FFTW for majority of transform operations.

**Objects of TARANG**

The class structure of TARANG is illustrated in Fig. 1. The class IncFluid contains the Incompressible velocity field and the associate solvers. This is the final class that inherits more classes. We describe the main features of the classes below.

**Class CVF: Complex Vector Field**

The class CVF stands for Complex Vector Field. It contains three dynamic arrays associated with the three components of the velocity or magnetic fields in the Fourier space. As mandated by FFTW, the size of each array is \(N_1, N_2, N_3/2 + 1\) that spans wavenumbers \((k_1, k_2, k_3) = [-N_1/2 : N_1/2, -N_2/2 : N_2/2, 0 : N_3/2]\) in FOUR basis, and \((k_1, k_2, k_3) = [0 : N_1, -N_2/2 : N_2/2, 0 : N_3/2]\) in SCFT basis. These arrays contains complex numbers that represent the Fourier amplitude of the vector field. The arrays are created dynamically at the runtime.

Forward and inverse transforms, and input/output of the vector fields are some of the main functions of the class CVF.

**Class RVF: Real Vector Field**

The class RVF stands for Real Vector Field, and it contains three dynamic arrays to represent the vector fields in the real space. We still create complex arrays of the size \(N_1, N_2, N_3/2 + 1\) for ease of FFTW and blitz++ operations; here the real and imaginary parts of a complex number represent two adjacent points in the real space.

**Class CSF: Complex Scalar Field**

The class CSF stands for Complex Scalar Field. It contains a dynamic array associated with a scalar field, e.g., temperature in the Fourier space. The indexing of the array is similar to that of CVF.

**Class RSF: Real Scalar Field**

The class RSF stands for Real Scalar Field, and it contains a dynamic array associate with a scalar. The array features are same as that for RVF.

The above four classes reside in directory named fields.

**Class IncVF: Incompressible Vector Field**

The class IncVF, acronym for Incompressible Vector Field, contains most crucial functions of the solver. IncVF inherits classes CVF, RVF, NLIN, EnergyTr. The classes CVF, RVF contain the velocity field in the Fourier and real space respectively. The class NLIN contains three arrays for storing the nonlinear term \(\hat{\partial_j u_j u_i}\), where the symbol \(\hat{\ }\) represents the Fourier transform. In addition, NLIN inherits CSF, whose array is used for storing the pressure field. The class EnergyTr contains function

---

FIG. 1. The class structure of IncFluid (Incompressible Fluids), which is the final class of TARANG.

FIG. 2. The class structure of IncSF (Incompressible scalar field).
for computing energy flux, shell-to-shell energy transfer etc. [2, 3].

The class IncVF also contains three arrays Force\_i to store the force fields, and two array *VF\_temp, *VF\_temp2 to save temporary fields. It also has an array *VF\_temp\_r that is used for storing temporary arrays in real space.

**Class IncSF: Incompressible Scalar Field**

The class IncSF is used for a scalar field accompanying incompressible velocity field. For example, in Rayleigh Bénard convection this class is used to represent the temperature field. IncSF inherits a CSF and a RSF to store the scalar field in the Fourier and real space respectively (see Fig. 2). It also contains arrays nlin, Force, and SF\_temp to store nonlinear term $u \cdot \nabla T$, forcing, and temporary array.

**Compute\_nlin(): a class function**

The class IncVF has many functions. However, Compute\_nlin is one of the most important functions of this class. We will describe this function as an illustration of TARANG function:

```cpp
void IncVF::Compute\_nlin()
{
  *V1r = *V1;
  *V2r = *V2;
  *V3r = *V3;
  // Inverse transform of *Vir using
  // *VF\_temp\_r as temporary array
  RV\_Inverse\_transform(*VF\_temp\_r);
  // Vr[i] -> Vr[i]^2 stored in nlin[i]
  Compute\_RSprod\_diag();
  // nlin[i]= Di T[Vr[i]^2];
  // T = Forward transform
  // Di=derivative along i-th dirn
  NLIN\_diag\_Forward\_transform\_derivative(*VF\_temp\_r);
  // Vr[i] = Vr[i]*Vr[j]
  Compute\_RSprod\_offdiag();
  // Vr[i] = T(Vr[i]*Vr[j])
  RV\_Forward\_transform\_RSprod(*VF\_temp\_r);
  // nlin[i] = Dj[T(Uj * Ui)]
  Derivative\_RSprod\_VV();
}
```

Listing 1. Compute\_nlin

The comments above the C++ statements explain the logic of the functions. Related functions compute the nonlinear term in the presence of scalar field and another vector field.

- void Compute\_nlin\_scalar(IncSF& T): $nlin\_i = \partial_j u_j u_i$ and $T.nlin\_i = \partial_j u_j F$, where $T$ is the scalar field.
- void Compute\_nlin\_RB(IncSF& T): same as Compute\_nlin\_scalar(IncSF& T).
- void Compute\_nlin(IncVF& W, IncSF& T): $nlin\_i = (\partial_j u_j w_i - \partial_j w_j w_i)$, and $W.nlin\_i = (\partial_j u_j w_i - \partial_j w_j u_i)$, where $w_i$ is the vector field associated with the IncVF class $W$.

**Class IncFluid: Incompressible Fluid**

The class IncFluid inherits IncVF and Time. Major functions of this class deal with time advancement of solver, forcing function, and input/output. The files and their associated functions are defined within this class. At present, the code includes Euler, Runge-Kutta second order (RK2), and Runge-Kutta fourth order (RK4) for the time advance function. The forcing function is used to include the buoyancy term in Rayleigh-Bénard convection (RBC), Coriolis force in the rotating turbulence etc.

For input/output, we have the option of reading/writing the data either the ASCII format or in High Density Format (HDF5) format. The HDF5 part of the code is being integrated with the main code. Also note that the classes IncVF and IncFluid have multiple inheritance.

**Solvers of TARANG**

We invoke the library functions discussed above to create solvers for fluid, magnetohydrodynamics, passive scalar, RBC flows etc. We illustrate a code segment containing the time-loop of fluid solver for an illustration.

```cpp
// A code segment of the fluid solver
// Read initial condition
U.Read\_init\_cond();
int iter = 0; // iterations
U.Tnow = U.Tinit;
do {
  U.Compute\_force();
  U.Compute\_nlin();
  U.Add\_force();
  U.Compute\_pressure();
  U.Tdt = U.Get\_dt();
```

```cpp
// A code segment of the fluid solver
// Read initial condition
U.Read\_init\_cond();
int iter = 0; // iterations
U.Tnow = U.Tinit;
do {
  U.Compute\_force();
  U.Compute\_nlin();
  U.Add\_force();
  U.Compute\_pressure();
  U.Tdt = U.Get\_dt();
```
In the above code segment, \( U \) is an instantiation of the class \( \text{IncFluid} \) which contains the incompressible velocity field. Most of the functions are obvious. The function \( U.\text{Get\_dt}() \) computes \( dt \) using CFL condition.

For the RBC, the above code segment is modified slightly. We create an instantiation \( T \) of the class \( \text{IncSF} \) to represent the temperature field.

```c
// A code segment of the RBC solver
// Read initial condition
U.\text{Read\_init\_cond}(T);
int iter=0; // iterations
U.Tnow = U.Tinit;
do {
    U.\text{Compute\_force}(T);
    U.\text{Compute\_nlin}(T);
    U.\text{Add\_force}(T);
    U.\text{Compute\_pressure}();
    U.Tdt = U.Get\_dt(T);
    U.Tnow = U.Tnow + U.Tdt;
    iter++;
    U.\text{Time\_advance}(T);
    // FIELD AT new time
    U.\text{Output\_all\_inloop}(T);
} while (U.Tnow < U.Tfinal);
```

Listing 3. RBC solver

**SAMPLE SIMULATION RESULTS AND VALIDATION**

We have performed simulations on fluids, convective, and MHD flows using TARANG on grids from \( 64^3 \) to \( 1024^3 \). The reader is referred to the published work for the scientific details. In the following discussion we detail some of the validations we performed before we launched large simulations.

**Simulations of RBC**

Rayleigh-Bénard convection (RBC) is an idealized version of the thermal convection in fluid. In the set up, a layer of incompressible fluid is confined between two thermally conducting plates separated by a distance \( d \). The bottom plate is heated and an adverse temperature gradient \( \beta \) is set across the fluid layer. The system is governed by the following equations:

\[
\begin{align*}
\partial_t u + (u \cdot \nabla) u &= -\nabla p + PR\theta \hat{z} + PV^2 u, \\
\partial_t \theta + (u \cdot \nabla) \theta &= u_3 + \nabla^2 \theta, \\
\nabla \cdot u &= 0,
\end{align*}
\]

where \( u = (u_1, u_2, u_3) \) is the velocity field, \( \theta \) is the perturbation in the temperature field from the steady conduction profile, and \( \hat{z} \) is the vertically directed unit vector. The equations are nondimensionalized by choosing length scale as \( d \), velocity scale as \( d^2/\kappa \), and temperature scale as \( \beta d \), where \( \kappa \) is the thermal diffusivity of the fluid.

Two non-dimensional parameters in the equations are the Rayleigh number, \( R = \alpha g \beta d^4/\nu \kappa \) and the Prandtl number, \( P = \nu/\kappa \), where \( \alpha \) is the coefficient of the volume expansion, \( g \) is the acceleration due to gravity, and \( \nu \) is the kinematic viscosity of the fluid. We also use another parameter, reduced Rayleigh number \( r = R/R_c \), where \( R_c \) is the critical Rayleigh number.

The top and bottom boundaries are considered to be stress free and perfectly conducting:

\[
\begin{align*}
u_3 &= \partial_z u_1 = \partial_z u_2 = \theta = 0, \quad \text{at} \quad z = 0, 1. \quad (4)
\end{align*}
\]

We assume periodic boundary conditions along the horizontal direction (\( x \) and \( y \)-direction). The boundary conditions chosen here are ideal. However they allow us to choose Fourier and 

\[
Nu = 1 + \langle u_3 \theta \rangle, \quad (5)
\]

where, \( \langle \cdot \rangle \) stands for spatial averaging. Please refer to Thual \cite{Thual}, for a detailed derivation of the expression for Nusselt number.

Thual \cite{Thual} numerically solved the Eqs. \cite{RBC} under the above boundary conditions (Eq. \cite{BC}) using a pseudospectral code. His simulations were performed in a two-dimensional (2D) box (aspect ratio \( \Gamma = 2\sqrt{2} \)) for \( r = 1.1 \) to 70 and \( P = 6.8 \). To verify TARANG we compare Nusselt numbers obtained in our simulations with those of Thual’s. The above set of Eqs. \cite{RBC} are solved numerically using TARANG under the above boundary conditions (Eq. \cite{BC}). We use same geometry (i.e. 2D box with aspect ratio \( 2\sqrt{2} \)) as used by Thual. We use Fourier basis functions for representation along the \( x \), and sin or cos functions for representation along the \( z \) direction (SCFT basis). The validation results are shown in Table \cite{Num1}. \( Nu \) values obtained with TARANG are in very good agreement with those of Thual’s until oscillation sets in the system.
TABLE I. Validation of TARANG against Thual’s [8] 2D simulations. We compare Nusselt numbers (\(Nu\)) obtained in our simulations with grid resolution 64\(^2\) against Thual’s 16\(^2\) (THU1), 32\(^2\) (THU2), and 64\(^2\) (THU3) simulations. All N\(u\) values tabulated below are for \(P = 6.8\).

| \(r\) | THU1 | THU2 | THU3 | TARANG |
|------|------|------|------|--------|
| 2    | 2.142| –    | –    | 2.142  |
| 3    | 2.678| –    | –    | 2.678  |
| 4    | 3.040| 3.040| –    | 3.040  |
| 6    | 3.553| 3.553| –    | 3.553  |
| 10   | 4.247| 4.244| –    | 4.243  |
| 20   | 5.363| 5.333| 5.333| 5.333  |
| 30   | 6.173| 6.105| 6.105| 6.105  |
| 40   | 6.848| 6.742| 6.740| 6.740  |
| 50   | 7.441| 7.298| 7.295| 7.295  |
| 70   | –    | oscil.| oscil.| 8.267  |

**PARALLELIZATION**

TARANG has been organized in a modular manner, so parallelization of the code was quite straightforward. Another major advantage was availability of parallel FFTW. We essentially adopt FFTW’s strategy for dividing the arrays etc. If \(p\) is the number of available processors, we divide each of the arrays into \(p\) segments. For example, a complex array \(A(N_1, N_2, N_3/2+1)\) is split into \(A(N_1/p, N_2, N_3/2 + 1)\) segments, each of which is handled by a processor. The other major parallel tasks needed is the multiplication in real space, which is handled by individual processors. Input/output is presently handled by the master node that collects/distributes data from the processor nodes. We are planning to implement parallel input/output using HDF5 functions.

**PORTING TO GPU**

Graphics Processing Units (GPUs) are getting popular in high performance computing due to their larger number of cores. We have ported the FFT part of TARANG to GPUs, and have observed a reasonable speedup. We are in the process of porting the entire code to multiple GPU platform.

**FUTURE PLANS**

We have successfully performed simulations for fluid, MHD, and convective flows for grids up to 1024\(^3\) on several platforms including PARAM YUVA (Centre for Advanced Computing, Pune), EKA (Computational Research Laboratory, Pune), HPC and CHAOS (both at IITK Kanpur). Attempts are being made to run turbulence simulations on higher grids.

We have solved channel flow using Chebyshev and Fourier basis functions. We will be porting the full implementation of the above basis function to be able to solve RBC and MHD flows under no-slip or mixed boundary conditions at the walls. The other planned modules are flows for the cylindrical and spherical geometry. We are also attempting to test and operationalize the magnetohconvection module.

**CONCLUSIONS**

TARANG exploits the object-oriented programming features of C++ to build flow solvers for incompressible fluid, MHD, and convection. We adopt a modular approach where general purpose functions are assembled to create solvers for different situations. This approach is proving to be very useful for constructing a large scale parallel softwares for fluid flows.

* mkv@iitk.ac.in

[1] C. Canuto, M. Y. Hussaini, A. Quarteroni, and T. A. Zhang, *Spectral Methods in Fluid Turbulence* (Springer-Verlag, Berlin, 1988).
[2] G. Dar, M. Verma, and V. Eswaran, *Physica D* **157**, 207 (2001).
[3] M. K. Verma, *Phys. Rep.* **401**, 229 (2004).
[4] P. Pal, P. Wahi, S. Paul, M. K. Verma, K. Kumar, and P. K. Mishra, *EPL-Europhys Lett* **87**, 056316 (2010).
[5] P. K. Mishra and M. K. Verma, *Phys. Rev. E* **81**, 056316 (2010).
[6] P. K. Mishra, P. Wahi, and M. K. Verma, *EPL-Europhys Lett* **89**, 44003 (2010).
[7] R. Yadav, M. Chandra, M. K. Verma, S. Paul, and P. Wahi, *Europhysics Letters* **91**, 69001 (2010).
[8] O. Thual, *J. Fluid Mech.* **240**, 229 (1992).