DNS TurISMi 1.7

This program is based upon DNS TurISMi 1.4 of Michele Iovieno. See http://areeweb.polito.it/ricerca/philofluid/ for information and documentation.

1 Requirements

Fortran 2008 compiler (GNU, Intel, IBM, ...) with Fortran 2008 standard support (see https://gcc.gnu.org for documentation of an open-source fortran compiler)

MPI 3.0 library or any other which support the MPI 3.0 standard, like OpenMPI 1.7.3, MPICH 3.0.4, ... (see http://www.mpi-forum.org for documentation)

FFTW 3.3.4 or higher (see http://www.fftw.org/ for installation and documentation)

2 Installation

A sample Makefile is included in the release. Installation can be performed via `make install` command. `make clean` is used to delete intermediate compilation files and `make veryclean` to uninstall the package. Installation can be customized as follows:

FC executable of the mpi Fortran compiler

FCF custom flags for the fortran compiler (like -g, -O3, etc.)

FFT fftw flags for the fortran compiler (already set-up)

IPATH path or system variable for of the fftw include files

LPATH path or system variable for the fftw library files

The following executable are available:

dns main code exec which evaluates the time integration of a periodic, parallelepipedic flowfied via Direct Numerical Simulation of NS equations (RK4 in time, Fourie-Galerkin Pseudo-spectral in space)

prog_trasf_dir_fftw transforms a parallelepipedic dataset from the physical space to its pseudo-fourier 3-D transform

prog_trasf_inv_fftw transforms a parallelepipedic dataset from a pseudo-fourier 3-D transform to physical space

3 Usage

Trasformation Programs

```
./prog_trasf_dir_fftw [alias] [file(s)]
./prog_trasf_inv_fftw [alias] [file(s)]
```

Perform the direct/inverse transform of the `[file(s)]` indicated in the arguments. If the argument `[alias]` is T, then a domain contraction/reduction is performed – following the 2/3 law. In other cases, the dimensions of the output are the same of the input.

DNS Program

```
./dns
```

No argument are needed.
The main program is responsible to load the data (depending on the `old_step` value of the parameter file. If `old_step = 0` velocity data are taken from the files indicated in the parameter file, and the scalar fields are computed in the `initialize_scalars` method. The integration is iterated then basing on the parameters indication for total number of steps and store frequency. Those configuration can be set in configuration file `param.dat`, for which an example is here represented in the following.

Section Grid for grid dimension

- `N = N1, N2, N3` – grid with dimension $N_1 \times N_2 \times N3$
- `N = 2*N , N3` – grid with dimension $N^2 \times N_3$
- `N = 3*N` – grid with dimension N^3

Section Params for physical parameters

- `Re = xxx` – Reynolds number
- `Sc = x1, x2, ...` – Schmidt(s) numbers
- `DT = xxx` – Integration time step
- `NSCAL = xxx` – Number of Passive Scalars

Section Simul for simulation parameters

- `TOTAL_STEPS` – Total number of steps
- `SAVE_FREQ` – Step save frequency
- `OLD_STEP` – Initial count for output files

Section Names for input file names

- `FNAME =xxx1,xxx2,xx3` – name of the files which contains the initial conditions for velocity field (used only if `OLD_STEP = 0`)

Example of param.dat.

```
&GRID
N        = 2*128,          512
/
&PARAM
RE       =   330.0000     ,
SC       =   1.000000     , 5*0.0000000E+00   ,
DT       =  4E-03,
NSCAL    =             1
/
&SIMUL
TOTAL_STEPS      =            2000,
SAVE_FREQ        =            40,
OLD_STEP         =            0
/
&NAMES
FNAME    = "f_r_u1.bin", "f_r_u2.bin", "f_r_u3.bin"
/
```

4 Domain Dimensions

N_1, N_2 and N_3 are the dimensions in wave-number space. All these values must be even.

M_1, M_2 and M_3 are the dimensions in phisical space. For the 2/3 anti-aliasing rule, $M_i/3 = N_i/2$

In wavenumber space, data are ordered like follows:

A(0) A(N+1) A(1) B(1) A(2) B(2) A(3) B(3) .... A(N-1) B(N-1) A(N) B(N)

where A(x) is the real part relative at the wavenumber x, and B(x) is the immaginary one. The advantage of this kind of ordering is that to derive a variable, the exchange of real and immaginary part can be done between contiguous vales.

n_process Total number of MPI instances

n_process_s Number of process along the first distributed direction (both for slab and stencil parallelization)

n_process_p Number of process along the second distributed direction (only for stencil parallelization)

Slab In that case `n_process_s = n_process` and `n_process_p = 1`. There is only one distributed direction at time. The requirement of such kind of distribution is that `n_process` is a divisor of $N_i/2$ for all directions.

Stencil In that case `n_process = n_process_s*n_process_p` There are two distributed direction at time. The requirement of such kind of distribution is that both `n_process_s` and `n_process_p` are a divisor of $N_i/2$ for all directions.

Why divisor of $N_i/2$ instead N_i It is due to the first derivative algorithm, which envisages that a sigle process know at the same time both real than immaginary part of a given wavenumber (also for distributed direction). In that way, derivative can be always computed locally inside a single process, without envolving MPI communications.

5 Modules and Methods

5.1 Module MPI Utils

Module for domain distribution management and inizialization of MPI variables. A smart domain recognization allow to run-time use a slab parallelization (1D) or a stencil parallelization (2D) depending on the domain dimension and on the total number of MPI threads. It makes available the so called local dimension, and initialize the comunicators among the MPI threads.

Method and arguments	Description	Notes
mpi_initialization()	initialize MPI	Must be called once during initialization
make_slab()	produce local dimension and comunicators for slab parallelization	
make_base_stencil()	produce local dimension and comunicators for stencil parallelization	
print_grid()	print the computed grid	

5.2 Module MPI Communications

Module which performs matricial transposition in a distributed domain. It use mpi global communication routines - in particular MPI_AllToAll.

Method name	Description	Notes
Trasp13s(A,B,Nl1,Nl2,Nl3)	Transposition between the first and the third index, single precision	Input: A(Nl1, Nl2, Nl3) Output: B(Nl3, Nl2, Nl1)
Trasp13d(A,B,Nl1,Nl2,Nl3)	Transposition between the first and the third index, double precision	Input: A(Nl1, Nl2, Nl3) Output: B(Nl3, Nl2, Nl1)
Trasp12s(A,B,Nl1,Nl2,Nl3)	Transposition between the first and the second index, single precision	Input: A(Nl1, Nl2, Nl3) Output: B(Nl2, Nl1, Nl3)
Trasp12d(A,B,Nl1,Nl2,Nl3)	Transposition between the first and the second index, double precision	Input: A(Nl1, Nl2, Nl3) Output: B(Nl2, Nl1, Nl3)

5.3 Module MPI InOut

Module which manage binary input/output operation with parallelized streams of data. Interfaces allows to use the same modules both in single than in double precision.

Method name	Description	Notes
save_3d_single(A,fname)	Store the variable A (single precision) into the file fname	
save_3d_double(A,fname)	Store the variable A (single precision) into the file fname	
save_3d(A,fname)	Store the variable A into the file fname	Interface
read_3d_single(A, fname)	Read the variable A (single precision) from the file fname	
read_3d_double(A, fname)	Read the variable A (single precision) from the file fname	
read_3d_(A, fname)	Read the variable A from the file fname	Interface

5.4 Module Derivative

Module for basic differential operation management. All the operations are local (no MPI communication is required.)

Method name	Description	Notes
Make_Wavenumbers()	Initialize the first and second order wave-numbers in each direction	Must called once after mpi inizialization
Derive1(A,B)	Compute the first derivative along the first index of the input A. Results are stored in B	Input: A Output: B
Derive2(A,B)	Compute the first derivative along the second index of the input A. Results are stored in B	Input: A Output: B
Derive3(A,B)	Compute the first derivative along the third index	Input: A Output: B

5.5 Module FFTW Utils

Module which manage the Fourier transform via FFTW. FFTW works computing the best execution schemas (called plans) during the initialization, based upon the domain dimensions. The plans are used run-time for quickly evaluate the direct/inverse transform of data with coherent dimension.

Method name	Description	Notes
Make_Fftplans()	Manage the plans needed to perform the transform according to domain dimension	Must be called once after initialization
Plan_dir(N, plan)	Evaluate the plan for a direct transform of a data with dimension N	Private usage
Plan_inv(N, plan)	Evaluate the plan for am inverse transform of a data with dimension N	Private usage
fft_1d_single_dir (var,Nl1,Nl2,Nl3,plan)	Execute a direct transform along the first index	input/output: var Nl1, Nl2, Nl3 – dimension of the domain (explicit definiton is needed) plan – the plan for the transform of segment with dimension equal to Nl1
fft_1d_single_inv (var,Nl1,Nl2,Nl3,plan)	Execute an inverse transform along the first index	input/output: var Nl1, Nl2, Nl3 – dimension of the domain (explicit definiton is needed) plan – the plan for the transform of segment with dimension equal to Nl1
fft_3d_single_dir_aa(A,B)	Exec a 3D direct transform of A into B, with domain contraction for Anti-aliasing	During the operation data are transposed
fft_3d_single_inv_aa(A,B)	Exec a 3D inverse transform of A into B, with domain expansion for Anti-aliasing	During the operation data are transposed
fft_3d_single_dir_na(A,B)	Exec a 3D direct transform of A into B, without domain contraction	During the operation data are transposed
fft_3d_single_inv_na(A,B)	Exec a 3D direct transform of A into B, without domain expansion	During the operation data are transposed

5.6 Module Physica

Module which computes the physical terms of the equations. The routine method ns_operator can be customized in order to include other physical effects can be added. Note that, generally, the convective terms must be the first one computed, and the poissonian must be the last one.

$$\frac{\partial U}{\partial t} = A \qquad \text{momentum eq.}$$

$$\frac{\partial S}{\partial t} = B \qquad \text{scalar eq.}$$

Method name	Description	Notes
ns_operator(U,A,S,B)	Container for non-time-derivative equation terms	
convective_terms(U,A,S,B)	Computes the convective terms	$A_i = \partial(U_i \cdot U_j)/\partial x_j$ $B_i = U_j \partial S_i/\partial x_j$
diffusive_terms(U,A,S,B)	Compute (pseudo) diffusion of U and S and add the result to A and B	$A_i = A_i - \partial^2 U_i/\partial x_i^2/Re$ $B_i = B_i - \partial^2 S_i/\partial x_i^2/Re/Sc_i$
poiss_solver(A)	Resolve the Poissonian equation in order to take into account pressure forces in the momentum equations	$A = A + \nabla(\nabla^{-2}(\nabla \cdot A)))$

5.7 Module Time Integrator

Time integration with low storage Runge-Kutta4 algorithm.

Method name	Description	Notes
rk4(u0,s0)	Time integration with low storage Runge-Kutta4 algorithm	
skel(var)	Used to evaluate running time basic statistics (mean, variance, thrd and fourth moment) along the third direction	

Appendix A Stencil vs Slab parallelization

The basic principle of these two kind of parallelization is almost the same. A 3-dimensional domain – with regular, rectangular structured pointgrid – can be shared among a moltidute of process subdividing the grid-points along one direction (slab) or two directions (stencil).

In spectral methods, all the basic operations (sum and differentiation) are local – considering complex variables. In that way no communication between process is needed for those process.

For non-local spectral operations (AKA multiplication), the resolution cannot be performed by a process only with its own data. In particular, the widely-used approach is to anti-transform the data and perform the multiplication in physical space, and then go back to spectral space.

To perform a transformation along one direction, such direction must be not distributed: this means that, for a 3-dimensional operation, it is necessary proceed in a direction-by-direction transformation, redistributing the data between each step so that the parallelization does not affect the direction to be processed.

Considering that the domain is distributed among N process, for slab parallelization the topology is $1 \times 1 \times N_s$ – which means that the first two direction are not distributed, and the third direction is distributed among N processes. The total number of processes involved is equal to $N = N_s$

Instead, for slab parallelization, the topology is $1 \times N_p \times N_s$ – which means that the first direction is not distributed, the send direction is distributed among N_p processes, and the third one is distributed among N_s processes. The total number of processes involved is equal to $N = N_p \cdot N_s$ Considering that, Slab and Stencil parallelizations, even if they follow the same conceptual architecture, differs for two aspects:

1. Different step-by-step procedure
2. Different process communicators

Step-by-step Transform procedure

In the following we show step-by-step how 3D transformation is performed. In each step the direction situation/position are resumed using the following conventions:

x, y and z indicate the real directions in physical space

$\hat{x}$ indicates the transform along direction x

x_d indicates that x is distributed

the ordering match with the computational order – so (z, y, x) means that z is on first computational direction, y on second, x on third

Slab

- Initial situation $\rightarrow (z, y, x_d)$
- Transformation along $z \rightarrow (\widehat{z}, y, x_d)$
- Transformation along $y \rightarrow (\widehat{z}, \widehat{y}, x_d)$
- Transposition **T13** between x and $z \rightarrow (x, \widehat{y}, \widehat{z}_d)$
- Transformation along $x \rightarrow (\widehat{x}, \widehat{y}, \widehat{z}_d)$

Stencil

- Initial situation $\rightarrow (z, x_d, y_d)$
- Transformation along $z \rightarrow (\widehat{z}, x_d, y_d)$
- Transposition **T13** between z and $y \rightarrow : (y, x_d, \widehat{z}_d)$
- Transformation along $y \rightarrow (\widehat{y}, x_d, \widehat{z}_d)$
- Transposition **T12** between y and $x \rightarrow (x, \widehat{y}_d, \widehat{z}_d)$
- Transformation along $x \rightarrow (\widehat{x}, \widehat{y}_d, \widehat{z}_d)$

Process communicators

A communicator is a property of each process, indicating which other processors are involved in a data exchange during a certainly message passing interface method.

The situation is almost trivial in Slab parallelization, since during **T13** transposition a process exchange data with all the other ones. In that case there is only one slab communicator – and obviously there is none stencil communicator. In particular, in that case the only communicator present corresponds to the global one.

In Stencil parallelization the situation is more complex: there are several slab and stencil communicators, depending on of domain decomposition topology. In particular, there will be N_p slab communicator, and N_s stencil communicators. During **T13** transpositions, a process exchange data only with other processes having the same slab communicator. Similarly, during **T12** transpositions, a process exchange data only with other processes having the same stencil communicator.

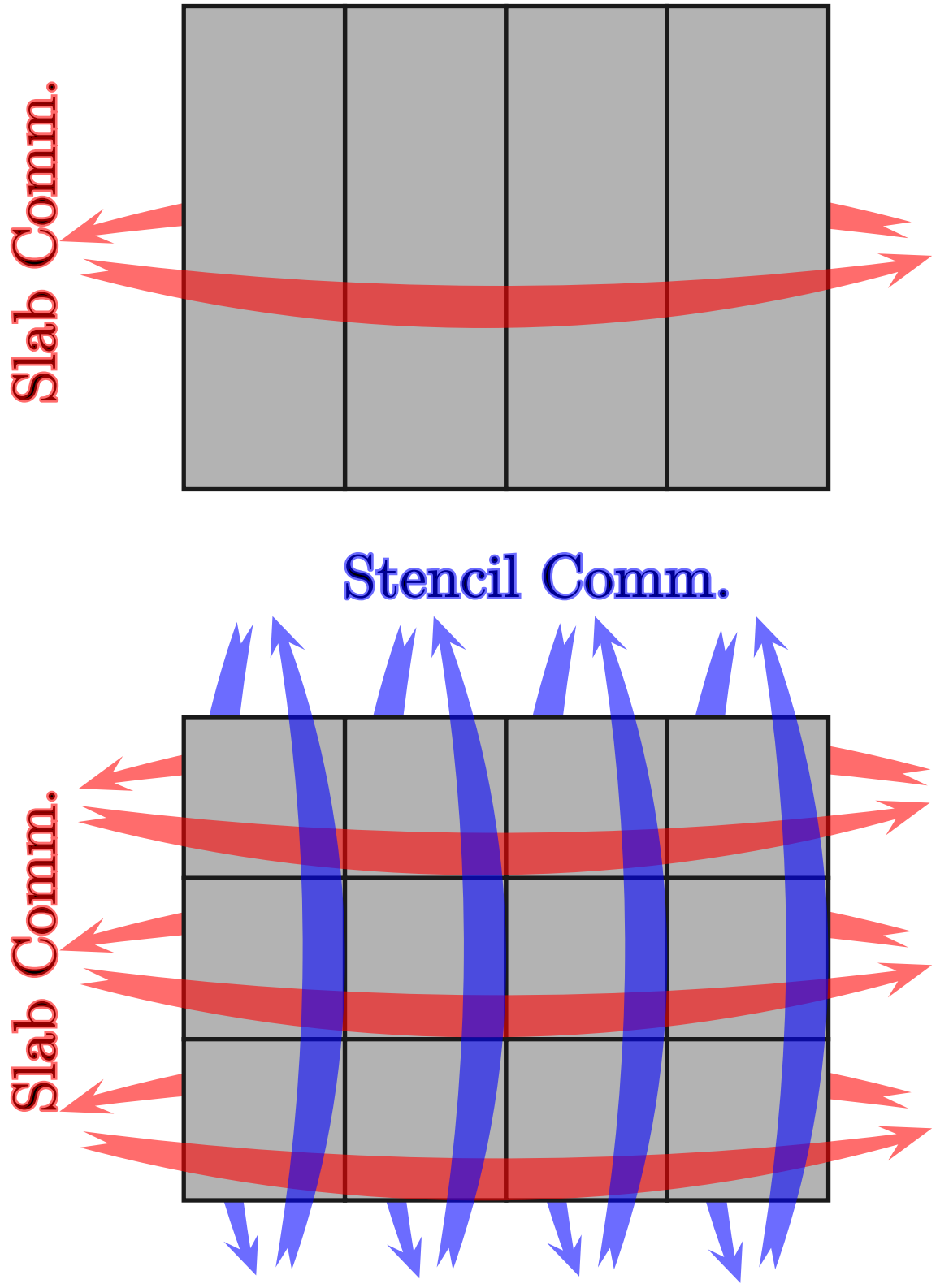

Figure 1: **Up** – Slab parallelization: there is only one slab communicator, which is the same for all process; it corresponds to the global communicator.
Down – Stencil parallelization: there are both slab and stencil communicators. A processes which belongs to a certainly slab communicator exchange data only with other processes having the same communicator during **T13** transposition. Similarly, a processes which belongs to a certainly stencil communicator can exchange data only with other processes having the same communicator during **T12** transposition.