INCOMPACT3D USER GUIDE

Version 1.1

Sylvain Laizet & Eric Lamballais

sylvain.laizet@gmail.com and eric.lamballais@univ-poitiers.fr

1 Overview of Incompact3d

Incompact3d is a powerful numerical tool for academic research. It combines the versatility of industrial codes with the accuracy of spectral codes. Thank to a successful software development project with NAG and HECToR (UK's national supercomputing facility), Incompact3d can be used on up to hundreds of thousands computational cores to solve the incompressible Navier-Stokes equations. This high level of parallelisation is achieved thanks to a highly scalable 2D decomposition library and a distributed Fast Fourier Transform (FFT) interface [5, 2]. This library is available at http://www.2decomp.org and can be freely used for your own code.

Incompact3d is based on a Cartesian mesh. The use of such a simplified mesh offers the opportunity to implement high-order compact schemes in the code for the spatial discretization whilst an Immersed Boundary Method (IBM) allows the implementation of any solid wall/bluff body geometry inside the computational domain. The main originality of the code is that the Poisson equation (to ensure the incompressibility) is fully solved in the spectral space via the modified wave number formalism, no matter what the boundary conditions are (periodic, free-slip, no-slip, inflow/outflow, etc.). Note finally that the pressure mesh is staggered from the velocity one by half a mesh to avoid spurious pressure oscillations that can be introduced by the IBM [1].

A priori, the combination of high-order schemes with the IBM might be problematic because of the discontinuity in velocity derivatives imposed locally by the artificial forcing term. However, even though the formal order of the solution can be reduced as a result of the IBM, the code has been demonstrated to be far more accurate with a sixth-order scheme than with a second order scheme both in statistics and instantaneous field realisations[7, 6]. Note that there is an ongoing research project in Poitiers (France) on this topic. The idea is to use an innovative 1D approach to reduce the discontinuity in velocity derivatives at the wall of the solid body. This new strategy is not yet implemented in the released version of the code.

More information about the numerical methods can be found in:

- Laizet S.& Lamballais E., **High-order compact schemes for incompressible flows: a simple and efficient method with the quasi-spectral accuracy**, J. Comp. Phys., vol 228-15, pp 5989-6015, 2009
- Lamballais E., Fortune V. & Laizet S., **Straightforward high-order numerical dissipation via the viscous term for Direct and Large Eddy Simulation**, J. Comp. Phys., Vol 230-9, pp 3270-3275, 2011

More information about the parallel strategy of the code can be found in:

- Laizet S.& Li N., **Incompact3d, a powerful tool to tackle turbulence problems with up to** $0(10^5)$ **computational cores**, Int. J. of Numerical Methods in Fluids, Vol 67-11, pp 1735-1757, 2011
- Li N. & Laizet S., **2DECOMP&FFT - a highly scalable 2D decomposition library and FFT interface**, Cray User Group meeting: Simulation comes of age Edinburgh, Scotland – 24/05-27/05, 2010
- Laizet S., Lamballais E. & Vassilicos J.C., **A numerical strategy to combine high-order schemes, complex geometry and parallel computing for high resolution DNS of fractal generated turbulence**, Computers & Fluids, vol 39-3, pp 471-484, 2010

IMPORTANT:

- **It is strongly recommended to read references [1] and [2] before starting using the code.**

- **We kindly ask you to cite the previous references (when suitable) in your work based on Incompact3d.**
- **This version of the code is not working in 2D (when nz=1). We are considering realeasing in the near future a 2D version without any domain decomposition.**

2 Getting started

The code should be easy to compile with the Makefile provided. You only need to customise the commands and flags depending on your machine/compiler. To start with, there is no need of external FFT libraries as a generic FFT subroutine is supplied. However, you are advised to eventually use a production quality FFT library, such as FFTW. Please kindly report bugs/problems, if any, at the compilation stage. The executable file is called incompact3d. On a local system, it is recommended to use the combination `gfortran` along with the `openmpi` library. `gfortran` is installed by default in most of the Linux distributions. Instruction to compile the `openmpi` library with `gfortran` can be found on www.openmpi.org

3 Detailed description of the code

In this section, only the files related to the solving of the incompressible Navier-Stokes equation are described. The 2D domain decomposition files are derived from the open source 2DECOMP&FFT library and are not explained in this document (see section 4 for more details). The input/output are described in the section 4.

List of files (alphabetic order)

◇ **convdiff.f90**: This file contains two subroutines: `convdiff` for the evaluation of the convective-diffusive terms and `scalar` for the entire evaluation of a passive scalar equation. Depending on your flow configuration and Reynolds number, it it recommended to use the extra-dissipation procedure introduced artificially via the viscous term. This opportunity is offered by the use of the compact schemes to compute second derivatives. This family of finite difference scheme can be easily adjusted to be over-dissipative on a narrow range of scales in the neighbourhood of the cutoff wave number associated with the mesh. It also allows an efficient control of the aliasing errors (non-negligible when high-order schemes are used) by comparison with a compact filtering of the non-linear terms. The numerical viscosity introduced by the present discretization is only concentrated at the highest wave numbers so that it cannot replace a subgrid-scale model in the context of large eddy simulation (LES).More details about this numerical procedure can be found in [3]. To control this extra-dissipation procedure, see the description of **schemes.f90**.

◇ **derive.f90**: This file contains all the derivatives and interpolations (derivatives from velocity mesh to velocity mesh, derivatives/interpolations from velocity mesh to pressure mesh, derivatives/interpolations from pressure mesh to velocity mesh). It is possible to switch from sixth order to fourth or second order by modifying the coefficients of the finite difference schemes in the file **schemes.f90**.

◇ **incompact3d.f90**: It is the main file of the code, see dedicated sub-section.

◇ **module_param.f90**: File with the modules for the code. Any variables/1D arrays should be declared in this file. The following four lines are very important to define the number of mesh nodes and the 2D mapping.

```
integer,parameter :: nx=129,ny=64,nz=32
integer,parameter :: nstat=4,nvisu=2
integer,parameter :: p\_row=4,p\_col=8
integer,parameter :: nxm=nx-1,nym=ny,nzm=nz
```

$n*$ (where $* = x,y,z$) correspond to the number of mesh nodes for the velocity mesh. $nm*$ correspond to the number of mesh nodes for the staggered pressure mesh and have to be changed depending on the set of boundary conditions. $p_{row} \times p_{col}$ defines the 2D mapping (size of the pencils, see [2] for more details). *nstat* defines

the size of the 3D array for the collection of the statistics. The statistics are saved every *nstat* mesh nodes. It is recommended to set $nstat = 2,4$ or 8. *nvisu* defines the size of the 3D arrays for the collection of 3D snapshots. The 3D snapshots are saved every *nvisu* mesh nodes. It is recommended to set $nvisu = 1$ or 2 for nice visualizations. The generic Fast Fourier Transformations (FFTs) implemented in Incompact3d allow the use of almost any number for $n*$. However the choice for $n*$ is very important in terms of performances. It is very important to know the architecture of the supercomputer where a simulation will be undertaken. For better performances, it is advised to take a multiple of the number of cores per processor. In a similar way, it is recommended to have n_* divisible by p_{row} and p_{col} in order to have an equal load balance among the computational cores and to avoid potential bugs (although the underlying communication library can in general handle unevenly distributed data). Finally, if $ncl* = 0$ then $n*$ needs to be even. If $ncl* = 1,2$ then $n*$ needs to be odd.

◇ **navier.f90**: This file contains various subroutines for the evaluation of the Navier-Stokes equations:

- `subroutine intt`: subroutine for the time advancement.
- `subroutine corgp`: subroutine for the velocity correction by the pressure gradients.
- `subroutine inflow/outflow`: By convention, for a spatially evolving configuration, the streamwise direction of the flow is in the $x-$direction. Therefore, when open boundaries conditions ($nclx = 2$) are used these subroutines are necessary to impose the inflow condition (most of the time a uniform profile with any perturbations) and the outflow condition (based on a basic 1D convection equation).
- `subroutine ecoule`: subroutine to define the mean flow configuration for the initial/input conditions (channel flow, wake, mixing-layer, etc.).
- `subroutine init`: subroutine to initialise the flow configuration when not starting with a restart file.
- `subroutine divergence`: subroutine to compute the divergence of a vector from velocity mesh to pressure mesh.
- `subroutine gradp`: subroutine to compute the gradient of the pressure (from pressure mesh to velocity mesh).
- `subroutine corgp_IBM`: subroutine for a pre/post correction by the pressure gradient on the intermediate velocity field. See [1] for more details.
- `subroutine body`: subroutine to define a solid body (such as a circular cylinder) inside the computational domain. The idea is to freeze the velocity to zero inside the computational domain. It is possible to improve this technique by using a mirror flow inside the solid body in order to avoid the discontinuity on the velocity field for a better estimation of the gradients. More details can be found in [7, 6].
- `subroutine pre_correc`: subroutine to impose the boundary conditions on the velocity when open boundary conditions are used on the velocity field ($ncl* = 2$). For instance, for a channel flow with walls in the y-direction ($ncly = 2$), $u_* = 0$ at the wall will be imposed in this subroutine. Boundary conditions for the scalar should be set up in the scalar subroutine.

We recommend not to modify these subroutines, except the subroutines `ecoule` and `init` to add more flow configurations. When a modification is needed, the user should add new subroutines rather than altering the existing ones.

◇ **parameters.f90**: Subroutine which reads the input file `incompact3d.prm` and then defines various parameters for the simulation.

◇ **poisson.f90**: This file contains the subroutines related to the evaluation of the Poisson equation in the spectral space (initialisation of modified wave numbers and transfer functions; shifted Fast Fourier Transformations (FFTs); matrix for stretched mesh in the lateral direction, see [1] for more details).

◇ **schemes.f90**: This file contains the subroutine with the initialisation of the variables needed for the derivative and interpolations. The coefficients should not be modified by the users, except to reduce the order of the schemes or to adjust the extra-dissipation procedure [3].

◇ **tools.f90**: This file contains various subroutines such as the computation of the min/max for the velocity/scalar, inversion of a pentadiagonal matrix (if a stretched mesh is used) and the restart procedure.

◇ **variables.f90**: All the 2D/3D arrays should be declared in this file. **By convention, only 3D arrays should appear in the argument of a subroutine**. It is important to declare 2D/3D arrays in the correct pencil (X, Y or Z-pencil, corresponding to 1,2,3 respectively), with the correct size: $*start(1,2,3) : *end(1,2,3)$ OR $*size(1,2,3)$ where $* = 1,2,3$. The $*size$ is used if there is no need to use the coefficients i, j, k in a subroutine. For instance, to store the velocity *uy* at the previous time step, the array *gy* is defined with $allocate(gy(xsize(1), xsize(2), xsize(3)$ whereas *uy*1 is defined with $allocate(uy1(xstart(1) : xend(1), xstart(2) : xend(2), xstart(3) : xend(3)))$. In pratice, both arrays have the same size but the coefficients i, j, k are different.

EXAMPLE:
A simulation is based on $n_x \times n_y \times n_z = 129 \times 64 \times 32$ mesh nodes using a 2D mapping $p_{row} \times p_{col} = 4 \times 8$=32 computational cores, ranking from 0 to 31. For this configuration, $xsize(1) = n_x$, $xsize(2) = n_y/p_{row} = 16$, $xsize(3) = n_z/p_{col} = 4$.
The size of *gy* is $gy(129,16,4)$ with $i = 1,..,129$, $j = 1,..,16$ and $k = 1,..,4$ for all the ranks.
The size of *uy* is $uy(129,16,4)$ with $i = 1,..,129$ for all the ranks; $j = 1,..,16$ for ranks 0 to 7; $j = 17,..,32$ for ranks 8 to 15; $j = 33,..,48$ for ranks 16 to 23 and $j = 49,..,64$ for ranks 24 to 31. $k = 1,..,4$ for ranks 0/8/16 and 24; $k = 5,..,8$ for ranks 1/9/17 and 25; $k = 9,..,12$ for ranks 2/10/18 and 26; $k = 13,..,16$ for ranks 3/11/19 and 27; $k = 17,..,20$ for ranks 4/12/20 and 28; $k = 21,..,24$ for rank 5/13/21 and 29; $k = 25,..,28$ for ranks 6/14/22 and 30 and finally $k = 29,..,32$ for ranks 7/15/23 and 31.

When developing your own subroutine in the code or before starting a simulation, it is recommended to check/print the $*size, *start$ and $*end$ for a better understanding of the 2D mapping decomposition.

◇ **visu.f90**: Input/Output file. See section 3 for more details.

Input parameters file: incompact3d.prm

The size of the computational domain is $xlx \times yly \times zlz$. It should be normalize with the reference length of the flow configuration. This could be the diameter D of a cylinder or half the size h of a channel flow. The input Reynolds number *re* is also based on the same characteristic length. The time step has to be chosen carefully with regards to the stability conditions of the simulation [4]. There is no procedure in the code to check if a time step is valid or not. It is recommended to undertake a preliminary study in order to find the optimum time step before running production simulations. The time step must be not too small for a reasonable computational cost but not too big for obvious stability issues. Note that at the moment, it is possible to use four different temporal schemes (parameter *nscheme*). It is recommended to use an AB scheme ($nscheme = 1,4$) in the presence of a solid body inside the computational domain ($ivirt = 1$). A random noise can be use for the inflow and/or for the initial condition ($noise/noise1$).

Different boundary conditions can be used in the three spatial direction:

- Periodic conditions corresponding to $ncl = 0$
- Free-slip conditions corresponding to $ncl = 1$
- Open Boundary conditions (Dirichlet conditions for the velocity for no-slip or inflow/outflow conditions), corresponding to $ncl = 2$

So far, 10 different combinations can be used in the code, covering a wide range of flow configuration: $(0-0-0)$, $(1-0-0)$, $(2-0-0)$, $(0-2-0)$, $(1-1-0)$, $(2-1-0)$, $(1-1-1)$, $(2-2-1)$, $(1-2-1)$ and $(2-2-2)$. For instance the combination for a temporal channel flow is $(0-2-0)$. For a spatially evolving flow, with periodic boundary conditions in the spanwise direction and free-slip boundary conditions in the lateral direction, the combination is $(2-1-0)$. It is important to know that all the combinations have not been tested so you may eventually experience some problems.

It is possible to use a stretched mesh in the lateral $y-$direction with the parameter *istret*. If $istret = 0$, no stretched mesh is used. If $istret = 1$ then a stretching is used with a mesh refinement in the centre of the computational domain. If $istret = 2$ then a stretching is used with a mesh refinement at the boundaries of the computational domain. Finally, if

istret $= 3$ then a stretching is used with a mesh refinement only at the bottom boundary of the computational domain. *istret* $= 2$ can be used for a turbulent channel flow, *istret* $= 3$ can be used for a turbulent boundary layer.

It is possible to use a checkpoint or restart procedure in the code with the parameter *ilit*. If *ilit* $= 1$ then the code will use the restart file `sauve.dat` that is generated every *isave* time step. It contains all the arrays to restart a simulation. The parameters *ifirst* and *ilast* must be modified accordingly. If *ilit* $= 0$ then the simulation will start with the initial conditions (subroutine *init*) with the initial configuration defined through the parameter *itype*.

Two different formulations can be used for the convective terms of the Navier-Stokes equation:

- the rotational formulation $H_i = u_j \left(\frac{\partial u_i}{\partial x_j} - \frac{\partial u_j}{\partial x_i} \right)$ corresponding to *iskew* $= 0$,
- the skew-symmetric formulation $H_i = \frac{1}{2} \left(\frac{\partial u_i u_j}{\partial x_j} + u_j \frac{\partial u_i}{\partial x_j} \right)$ corresponding to *iskew* $= 1$.

We recommend to use the skew-symmetric formulation for a better modelisation of the small scales.

Main file of the code: incompact3d.f90

`incompact3d.f90` is the main file of the code, containing the time loop relative to the evaluation of the incompressible Navier-Stokes equations.

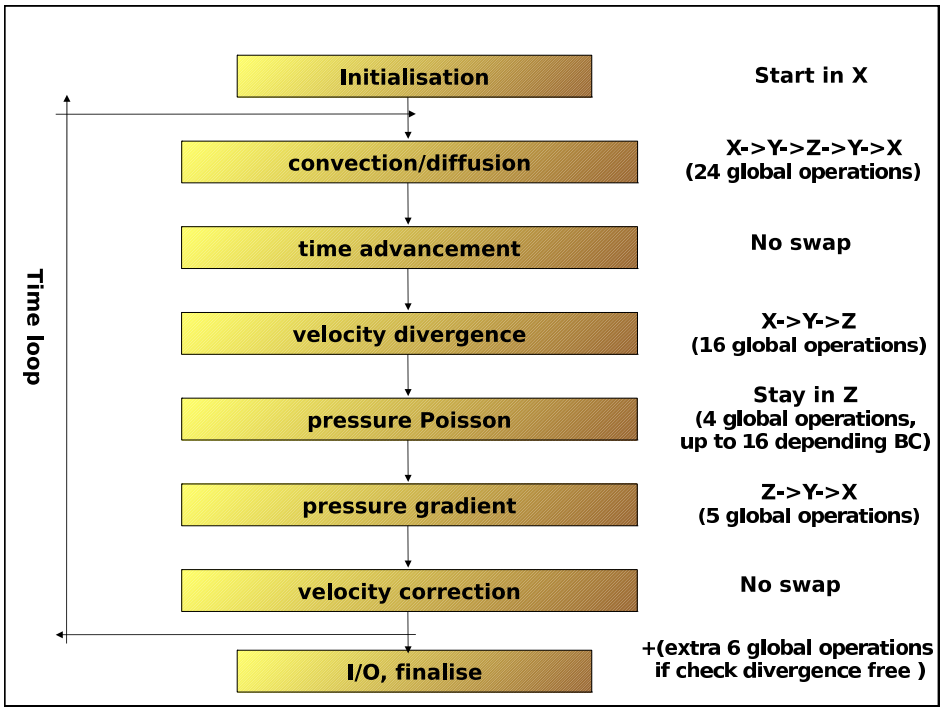

Figure 1: *Stucture of* **Incompact3d** *with the 2D domain decomposition.*

The structure of the code is presented in figure 1. This figure also shows the management of the pencils swaps. For the Poisson solver in the spectral space, a single division is required and the modified wave numbers combined with the transfer functions are all independent with each other. Note that, when performing the 3D FFT forward, we are in Z-pencil in the physical space, then in X-pencil in the spectral space and finally, in Z-pencil again after the 3D FFT backward, in order to reduce the global transpose operations. For the code using tri-periodic boundary conditions, 55 global transpose operations need to be performed at each time step. This number can be up to 69 depending on the boundary conditions.

The subroutine `STATISTIC` is collecting first and second order moments statistics. The idea is to collect data over time in space (every *nstat* mesh nodes). In the subroutine `VISU_INSTA`, it is possible to collect 2D/3D snapshots of the

flow (similar to a virtual cameras), to record various quantities at each time step (similar to a virtual probes). These subroutines are in the file `visu.f90` and the Input/Output tools are described in the following section.

4 Input/Output

Several tools have been developed to manage efficiently Input/Output in the code. As previously explained there is a restart procedure in the code (file `tools.f90`, subroutine `restart`). A restart file `sauve.dat` is generated every *isave* time step. Note that it is possible to restart the simulation with different number of computational cores. In terms of 2D/3D snapshots, several subroutines can be used:

- **2D snapshots (full resolution):**
  subroutine $decomp\_2d\_write\_plane(pencil, var, iplane, n, filename)$:
  $ipencil$ is equal to 1,2,3 for X-pencil, Y-pencil, Z-pencil respectively. *var* is the name of the 3D array. $iplane$ is equal to 1,2,3 to save a X-plane, Y-plane, Z-plane respectively. *n* corresponds to the location of the plane. $filename$ is the name of the output file.
  EXAMPLE:
  call decomp_2d_write_plane(1,ux1,1,112,'ux2d') is going to write in *ux2d* a 2D (y-z)-plane of the 3D array ux1 (defined in X-pencil), for i=112.

- **3D snapshots (full resolution):**
  subroutine $decomp\_2d\_write\_one(nx, ny, nz, ipencil, var, filename)$:
  $ipencil$ is equal to 1,2,3 for X-pencil, Y-pencil, Z-pencil respectively. *var* is the name of the 3D array. $filename$ is the name of the output file.
  EXAMPLE:
  call decomp_2d_write_one(nx,ny,nz,2,uy2,'uy2.dat') is going to write in *uy*2.*dat* the 3D array *uy*2 (defined in Y-pencil).

- **3D snapshots (coarse resolution):**
  subroutine $decomp\_2d\_write\_one(ipencil, var, filename, icoarse)$:
  $ipencil$ is equal to 1,2,3 for X-pencil, Y-pencil, Z-pencil respectively.*var* is the name of the 3D array. $filename$ is the name of the output file. *icoarse* is equal to *nstat* or *nvisu* (see file $module\_param.f90$). The array *var* is defined on a coarse mesh (every *nstat* or *nvisu* mesh node). Before calling the subroutine it is necessary to call the subroutine
  $fine\_to\_coarseV(ipencil, var\_full, var\_coarseV)$ or
  $fine\_to\_coarseS(ipencil, var\_full, var\_coarseS)$.
  The size of *var_coarseV* is $(xszV(1), xszV(2), xszV(3))$.
  The size of *var_coarseS* is $(xszS(1), xszS(2), xszS(3))$.
  EXAMPLE:
  call fine_to_coarseV(1,uz1,uvisu) will write in the coarse array *uvisu* the 3D arrays (full resolution) *uz*1 (defined in Z-pencil).
  Then *call decomp_2d_write_one(1,uvisu,'uz_coarse.dat',2)* will write in *uz_coarse.dat* the 3D array *uvisu* (defined every *nvisu* mesh nodes in X-pencil).

- **X-pencils (various resolutions):**
  It is obviously possible to save data for only a few numbers of X-pencils.
  EXAMPLE:
  In this example, a simulation with $n_x \times n_y \times n_z = 2881 \times 360 \times 360$ is running on 3600 computational cores with a 2D mapping $p_{raw} \times p_{col} = 60 \times 60$. I want to save the three components of the velocity at each time step for $j = k = 181$ and every 8 mesh nodes in the $x-$direction. I will simply use the following lines:

```
if (nrank==4095) then
 if (itime==ifirst) then
   write (filename, 923) nrank
   open (nrank,file=filename,form='unformatted')
 endif
```

```
write (nrank) (((ux1(i,j,k),i=1,xsize(1),8),j=1,xsize(2),2),k=1,xsize(3),2),&
              (((uy1(i,j,k),i=1,xsize(1),8),j=1,xsize(2),2),k=1,xsize(3),2),&
              (((uz1(i,j,k),i=1,xsize(1),8),j=1,xsize(2),2),k=1,xsize(3),2)
 if (itime==ilast) close (nrank)
endif
```

To identify which computational core corresponds to $j = k = 181$, it is recommended to use the arrays *xstart* and *xend*. A similar procedure can be used to save data in Y-pencils and Z-pencils.

5 2D Decomp&FFT

In order to make the best use of supercomputers, the code is using the 2DECOMP&FFT library. The files related to this library are not explained in this document. 2DECOMP&FFT is a software framework to facilitate the creation of large-scale parallel scientific applications on supercomputers. As a communication library, it implements so-called 2D pencil decomposition for partitioning three-dimensional data sets on distributed systems and performing transpose-based communication. It also provides a highly scalable and efficient interface to perform three-dimensional Fast Fourier Transforms (FFT). The library is written in Fortran and built on top of MPI.

Applications that are based on three-dimensional Cartesian-topology structured mesh and use some of the following numerical algorithms may benefit from 2DECOMP&FFT:

- spectral method
- compact finite difference (or similar spatially implicit schemes)
- elliptic PDE solver (Poisson, Helmholtz, etc.)
- other algorithms involving 3D FFT.

Further details are available at http://www.2decomp.org

6 Recommendations/Remarks

- COMPILERS and OPTIONS: The code has been used on several supercomputers (more than 15 so far, with different architectures, such as Cray XT/XE, Blue Gene, etc.), with different Fortran compilers and different options. The code is (or at least should be!) machine/compiler independent. However, it is possible to experience some difficulties at the compilation stage or when running the simulations. It is important to report any suspicious behaviour so we can investigate any potential problem. In terms of optimisation, it is crucial to test the options available for a compiler. The code can be very sensitive to optimization options and finding the good options can save you a lot of time!
- NUMBER OF CORES and 2D MAPPING: As previously explained, it is very important to find the optimum number of computational cores and the more efficient 2D mapping ($p\_row \times p\_col$) for a given production simulation in order to save computational time. The underlying communication library can be instructed to run in an auto-tuning mode (by setting p_{row} and p_{col} to 0). It will then suggest a reasonable 2D processor mapping to be used. With such suggestion in hand, users are advised to try other combinations of p_{row} and p_{col} to decide the best mapping. Based on the authors experience, it is recommended to use more than 100,000 mesh nodes per computational cores. Below this limit, the performance of the code will be very poor due to an unfair balance between communication and computation. It is very important to undertake your own performance testing as supercomputers can be very different from one to another.
- BUGS/PROBLEMS: This is the first release of the code so please be nice with the code and not too demanding! Opening the code is an opportunity for every user to benefit from contributions from others, contributions that will be in future versions, widening the range of applications of the code and correcting all the minor/major bugs. The possibilities of the code are huge and we did not have the chance to validate all the different configurations. **Please be kind to report any bugs/problems through the user group forum (you need to register). Other users may be able to help you!**

- TUTORIALS and TRAINING: We are planning to release two or three tutorials: T1 (Flow past a cylinder at $Re = 300$) and T2 (turbulent channel flow at $Re_\tau = 180$). Those tutorials should be very easy to follow for new users.

  Training courses will be offered very soon up to twice a year in various institutions in Europe to present and promote Incompact3d. Any user who wants to use the code for a research project should attend the training course. The authors will be very happy to interact with potential users and to discuss for any kind of collaborations.

7 How to open a file generated by Incompact3d

In the subroutine **visu.f90**, you have the possibility of generated 2D/3D snapshots. Those snapshots, saved in the direct access format, can be easily open on your linux computer and then can re-written in order to use the visualization software of your choice. Below is a programme to open the file "ux00008" generated by Incompact3d and then re-write it in a vtr format for Paraview.

```
program visu

  implicit none

  integer, parameter :: nx=1441, ny=360, nz=360
  real(4),dimension(nx,ny,nz) :: ux
  integer :: i,j,k,count,nfil
  real(4),dimension(nx) :: y1
  real(4),dimension(ny) :: y2
  real(4),dimension(nz) :: y3

  OPEN(10,FILE='ux00008',FORM='UNFORMATTED',ACCESS='DIRECT', RECL=4, STATUS='OLD')
  COUNT = 1
  DO K=1,nz
     DO J=1,ny
        DO I=1,nx
           READ(10,REC=COUNT) ux(I,J,K)
           COUNT = COUNT + 1
        ENDDO
     ENDDO
  ENDDO
  CLOSE(10)

!generation of the mesh
do i=1,nx
   y1(i)=(i-1)*0.8 !0.8 is DX
enddo
do j=1,ny
   y2(j)=(j-1)*0.8 !0.8 is DY
enddo
do k=1,nz
   y3(k)=(k-1)*0.8 !0.8 is DZ
enddo

  nfil=41
  open(nfil,file='tampon33.vtr')
  write(nfil,*)'<VTKFile type="RectilinearGrid" version="0.1"',' byte_order="LittleEndian">'
  write(nfil,*)'<RectilinearGrid WholeExtent=','"1 ',nx,' 1 ',ny,' 1 ',nz,'">'
```

```
  write(nfil,*)'<Piece Extent=','"1 ',nx,' 1 ',ny,' 1 ',nz,'">'
  write(nfil,*)'<Coordinates>'
  write(nfil,*)'<DataArray type="Float32"',' Name="X_COORDINATES"',' NumberOfComponents="1">'
  write(nfil,*) (y1(i),i=1,nx)
  write(nfil,*)'</DataArray>'
  write(nfil,*)'<DataArray type="Float32"',' Name="Y_COORDINATES"',' NumberOfComponents="1">'
  write(nfil,*) (y2(j),j=1,ny)
  write(nfil,*)'</DataArray>'
  write(nfil,*)'<DataArray type="Float32"',' Name="Z_COORDINATES"',' NumberOfComponents="1">'
  write(nfil,*) (y3(k),k=1,nz)
  write(nfil,*)'</DataArray>'
  write(nfil,*)'</Coordinates>'
  write(nfil,*)'<PointData Scalars="scalar">'
  write(nfil,*)'<DataArray Name="test"',' type="Float32"',' NumberOfComponents="1"',' format="ascii">'
  write(nfil,*) (((ux(i,j,k),i=1,nx),j=1,ny),k=1,nz)
  write(nfil,*)'</DataArray>'
  write(nfil,*)'</PointData>'
  write(nfil,*)'</Piece>'
  write(nfil,*)'</RectilinearGrid>'
  write(nfil,*)'</VTKFile>'
  close(nfil)

end program visu
```

Comments and remarks about this example:

- Note that this example is for single precision and little endian data.
- For double precision, `RECL=4` should be `RECL=8` (along with a correct declaration of variables).
- For big endian data, `convert='big_endian'` needs to be added in the list of argument of the first `open` command.
- When using the Fortran compiler, you need to add the `-assume byterecl` option.
- For large file, you need to add the `-shared-intel -mcmodel=large` option with the Fortran compiler and the `-mcmodel=large` option with the `gfortran` compiler.
- This program may not work for very large file with the `gfortran` compiler.

Acknowledgements

Sylvain Laizet and Eric Lamballais would like to thank Imperial College London and The University of Poitiers for agreeing to make the code Incompact3d available for the scientific community. The authors would also like to thank Dr Ning Li from NAG for the implementation of the 2D decomposition library and distributed Fast Fourier Transform (FFT) interface in the code. They also thank Dr Sylvain Lardeau for the very early development of the code (about ten years ago...!). Sylvain Laizet would like to thank Prof. Christos Vassilicos for the freedom in his research in the last six years. EPSRC (grant EP/H030875/1), the UK turbulence consortium (grant EP/G069581/1), NAG and HECToR through the dCSE initiative are also acknowledged for financial support and computational time.

References

[1] S. Laizet and E. Lamballais. High-order compact schemes for incompressible flows: a simple and efficient method with the quasi-spectral accuracy. *J. Comp. Phys.*, **228(16)**:5989–6015, 2009.

[2] S. Laizet and N. Li. Incompact3d, a powerful tool to tackle turbulence problems with up to $o(10^5)$ computational cores. *Int. J. Numer. Methods Fluids*, **67(11)**:1735–1757, 2011.

[3] E. Lamballais, V. Fortune, and S. Laizet. Straightforward high-order numerical dissipation via the viscous term for direct and large eddy simulation. *J. Comp. Phys.*, **230(9)**:3270–3275, 2011.

[4] S. K. Lele. Compact finite difference schemes with spectral-like resolution. *J. Comp. Phys.*, **103**:16–42, 1992.

[5] N. Li and S. Laizet. 2DECOMPFFT a highly scalable 2d decomposition library and FFT interface. In *Cray User Group 2010*, Edinburgh, 2010.

[6] P. Parnaudeau, J. Carlier, D. Heitz, and E. Lamballais. Experimental and numerical studies of the flow over a circular cylinder at Reynolds number 3900. *Phys. Fluids*, **20**:085101, 2008.

[7] P. Parnaudeau, E. Lamballais, D. Heitz, and J. H. Silvestrini. Combination of the immersed boundary method with compact schemes for DNS of flows in complex geometry. In *Proc. DLES-5*, Munich, 2003.