Notus first steps (0.3.0)

Stéphane Glockner, Antoine Lemoine, Mathieu Coquerelle, and students, PhDs, ...

Institut de Mécanique et d'Ingénierie de Bordeaux
Université de Bordeaux, Bordeaux-INP, CNRS UMR 52 95

https://notus-cfd.org

June 5th 2019
<table>
<thead>
<tr>
<th></th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Notus code purposes</td>
</tr>
<tr>
<td>2</td>
<td>Development environment</td>
</tr>
<tr>
<td>3</td>
<td>Installation, compilation</td>
</tr>
<tr>
<td>4</td>
<td>Run notus</td>
</tr>
<tr>
<td>5</td>
<td>User interface</td>
</tr>
<tr>
<td>6</td>
<td>IO / Visualisation</td>
</tr>
<tr>
<td>7</td>
<td>Architecture, some development keys, user mode</td>
</tr>
<tr>
<td>8</td>
<td>Documentation</td>
</tr>
<tr>
<td>9</td>
<td>Notus Verification &amp; Validation tools</td>
</tr>
<tr>
<td>10</td>
<td>Notus Porting &amp; Performance tools</td>
</tr>
<tr>
<td>11</td>
<td>Development tools</td>
</tr>
<tr>
<td>12</td>
<td>Git usage</td>
</tr>
</tbody>
</table>
Notus code purposes

Open-source project started from scratch in 2015 (CeCILL Licence)

- Modelisation and simulation of **incompressible fluid flows**, multiphysics
- 2D/3D Finite Volume methods on staggered grids, **Massively parallel**

Intended users

- **Mechanical community**: easy to use and adapt, proven state-of-the-art numerical methods, towards numerical experiments
- **Mathematical community**: develop new numerical schemes, fast and efficient framework for comparative and qualitative tests
- Researchers, students, industrials

Some key points

- Take advantage of synergies between Research / Teaching / Industry / HPC
- A clear and complete development environment
- **Mask parallelism** complexities for easy programming
- **Porting** on GENCI, PRACE, mesocentres
- A thoroughly **validated and documented code**, **non-regression** approach

What is not Notus

- A concurrent of, a commercial tool, a click button code
Notus - some examples
Several user types

Notus first step: focus on “Notus user”, Simulation & Advanced
Contents

1. Notus code purposes
2. Development environment
3. Installation, compilation
4. Run notus
5. User interface
6. IO / Visualisation
7. Architecture, some development keys, user mode
8. Documentation
9. Notus Verification & Validation tools
10. Notus Porting & Performance tools
11. Developement tools
12. Git usage
## Development environment

### Development framework

- **Fortran 2008**
  - Allocatable arrays, structured and derived type
  - Module-oriented programming (private or public internal subprograms)
  - Optional arguments & intent attribute
  - Generic subroutine
  - Preprocessor
  - Interoperability with C (binding)

- **MPI** parallel coding library
- **Git** distributed version control system
- **CMake** cross-platform build system
- **Doxygen** documentation generator from source code
- **Linux only!**
- **Web site**
  - [https://notus-cfd.org](https://notus-cfd.org)
  - [https://doc.notus-cfd.org](https://doc.notus-cfd.org)
  - [https://git.notus-cfd.org](https://git.notus-cfd.org)

### Compilers and MPI libraries

- GNU compilers (> 7.3) and Open MPI (2.10)
- Intel compilers (> 18) and Intel MPI

### Supercomputers

- Irene at TGCC, Occigen at CINES
- Curta at MCIA
- Condor at I2M
Installation of Notus

Two steps

- Third part libraries
  - BLAS & LAPACK → system
  - Other dependencies: ADIOS (MXML), HYPRE, MUMPS (METIS, Scalapack), LIS
  - Be sure of the version installed → Git repository with tarballs
  - https://git.notus-cfd.org/notus/notus.third_party/

- Notus code
  - https://git.notus-cfd.org/notus/notus

Get and build third part libraries

Clone third part lib repository
$ git clone https://git.notus-cfd.org/notus/notus.third_party.git notus.third_party

Build libraries
Help:
$ ./build_notus_third_party_lib.sh -h

Compilation and installation on Manjaro/Arch Linux/Gentoo:
$ ./build_notus_third_party_lib.sh -a

Compilation and installation on Ubuntu 18.04:
$ ./build_notus_third_party_lib.sh -a --with-MPI-include /usr/include/mpi

Compilation and installation on CINES Occigen supercomputer:
$ ./build_notus_third_party_lib.sh -a --use-mkl --cc icc --cxx icpc --fc ifort --mpicxx mpiicpc

→ Readme page: https://git.notus-cfd.org/notus/notus.third_party
Installation of Notus

Get Notus

$ git clone https://git.notus-cfd.org/notus/notus.git notus

or, if you have a git account:

$ git clone git@git.notus-cfd.org:user/notus.git notus
$ cd notus
$ git remote add official git@git.notus-cfd.org:notus/notus.git
$ git remote update

→ to create a gitlab account: https://doc.notus-cfd.org/d3/d64/install.getnotus.html
Installation of Notus

**Build Notus with Cmake (Open-source software for managing build process)**

- Compiler independant
- Supports directory hierarchies
- Automatically generates file dependencies, supports library dependencies
- Builds a directory tree outside the source tree

**CMake and Notus**

- CMakeLists.txt
  - several development environments: GNU, Intel, etc.
  - find third party libraries
  - MPI (only), release or debug (default) builds
  - always debug for development; release for production

- Build scripts available for specific computers: *linux workstation, condor, occigen, curta, curie, etc.*
  
  $ cp build_scripts/build_notus_linux.sh .

  $ ./build_notus_linux.sh -h
  Usage: buildcmakecondor.sh [OPTIONS]
  -c clean the build directory
  -r release build (default: debug)
  -m use MUMPS solver (default: false)
  -l use LIS solvers (default: false)
  -j NUMBER number of compilation jobs (default: 1)
  -h print usage

  $ ./build_notus_linux.sh -mj 4
Run Notus

Basic way

- **Parallel execution** → `mpirun` command
  
  $ mpirun -np 8 notus test_case_validation_free_convection/square.nts

- **Test case data base** in `test_case` directory
  
  verification: laplacian, navier, phase_advection, phase_change, etc.
  validation: laminar_flow, free_convection, multiphase, etc.

- **Use your own directory** to store your `.nts` files

- **Complete list of command line options:** $ ./notus -h

Advanced ways

- `notus.py`
- `notus_grid_convergence`
Run Notus

Job submission on a supercomputer

- Share resources managed thanks to a job scheduler
- You have to submit your job and wait
- Limit amount of processors and CPU time
- `tools/submission_scripts`
- Dependency

Choose the amount of processors you need

3D: 100,000 cells / core
2D: 10,000 cells per core
User Interface: .nts file

Concept

- ASCII .nts files
- Self-explanatory keywords, precise grammar
- Efficient parser that supports:
  - variable declaration
  - formula
  - 'include'
  - if condition and loop
- Associated documentation → test_cases/doc directory

Organisation

- Physical fluid properties data base: std/physical_properties.nts file
- One .nts file per test case, block structure:
  - include and variable declarations
  - system{}
  - domain{}
  - mesh{}
  - modeling{}
  - numerical_methods{}
  - post_processing{]
include std "physical_properties.nts";

system {measure_cpu_time;}

domain {
    spatial_dimension 2;
    corner_1_coordinates (0.0, 0.0);
    corner_2_coordinates (1.0, 2.0);
}

grid {
    grid_type regular;
    number_of_cells (32, 32);
}

modeling {
    fluids {fluid "water";}
    equations {
        energy {
            boundary_condition{
                left dirichlet 0.0;
                right dirichlet 1.0;
                top neumann 0.0;
                bottom neumann 0.0;
            }
            source_term {constant -2.0;}
            disable_advection_term;
            disable_temporal_term;
        }
    }
}

numerical_parameters {
    time_iterations 1;
    energy {
        solver mumps_metis;
    }
}

post_processing {
    output_library adios;
    output_frequency 1;
    output_fields temperature;
}
string s = "Notus";
integer i = 1;
double a = 10.0;
boolean l = true;

a = 3.0d2; # 'a' contains 300.0
a = 2.0e1; # 'a' contains 20.0
a = b/c + c + sqrt(a) + cos(b) + pow(b,3);
s = "I" + " love " + "Notus";

integer h2g2 = 42;
export h2g2;

while (i < 10) {
    # Code to execute while i < 10
    if (i > n) {exit;} # Break the loop if i > n
    i = i + 1;
}

integer no_redefine scale = 2;
system {
  # [OPTIONAL] Overwrite default output directory (default: "output")
  output_directory STRING_EXPRESSION;

  # [OPTIONAL] Checkpoint (write alternatively, in 2 sets of files,
  information to restart the job at the same state).
  checkpoint_frequency INTEGER_EXPRESSION;

  # [OPTIONAL] Restart with file set 1 or 2
  restart_with_file_set_1;
  restart_with_file_set_2;

  # [OPTIONAL] Measure CPU time in several parts of the code
  measure_cpu_time;
}
domain {
    spatial_dimension 2; # or 3

    # The coordinates of 2 opposite corners of the physical domain
    corner_1_coordinates DOUBLE_ARRAY;
    corner_2_coordinates DOUBLE_ARRAY;

    # [OPTIONAL] Domain periodicity
    periodicity_x;
    periodicity_y;
    periodicity_z;

    # [OPTIONAL] Define a subdomain
    subdomain STRING_EXPRESSION {
        SHAPE # See shapes.nts
    }

    # [OPTIONAL] Add an immersed boundary
    immersed_boundary {
        SHAPE # See shapes.nts
    }
}
User Interface: `grid block`

```plaintext
grid {
    number_of_cells (32, 32);

    grid_type regular;

    grid_type chebyshev;

    grid_type exponential;
    expansion_ratio DOUBLE_EXPRESSION;  # Last step/first step
    first_step DOUBLE_EXPRESSION;        # Impose first step
    last_step  DOUBLE_EXPRESSION;        # Impose last step

    grid_type composite;

    grid_x {
        grid_type regular;

        grid_type chebyshev;

        grid_type exponential;
        expansion_ratio DOUBLE_EXPRESSION;  # Last step/first step
        first_step DOUBLE_EXPRESSION;        # Impose first step
        last_step  DOUBLE_EXPRESSION;        # Impose last step

        next_bound DOUBLE_EXPRESSION;
        length DOUBLE_EXPRESSION;

        number_of_cells INTEGER_EXPRESSION;
    }
    grid_x {
        ...
    }
    ...

    number_of_ghost_cells INTEGER_EXPRESSION;
}
```

### Composite grids

Generate a grid by parts.
For example:

```
regular  exponential
|-----------------| → direction x
```

### Number of ghost cells

Recommended values:
1. default value
2. if MOF
2. if VOF-PLIC
2. if Immersed boundary and second order Neumann b.c.
4. if WENO
4. if level-set and curvature
modeling {
  fluids {
    fluid "air"; => std/physical_properties.nts
      # Generic definition of a fluid
    fluid STRING_EXPRESSION [ { FLUID_PROPERTIES } ];
  }

  # Definition of FLUID_PROPERTIES
  fluid STRING_EXPRESSION {
    density DENSITY_TYPE DOUBLE_EXPRESSION;

      # DENSITY_TYPE can be either 'constant' or 'linear_temperature'
    density constant DOUBLE_EXPRESSION; # Constant value
    density linear_temperature DOUBLE_EXPRESSION; # Boussinesq

    viscosity DOUBLE_EXPRESSION;  
    conductivity DOUBLE_EXPRESSION; 
    specific_heat DOUBLE_EXPRESSION;  
    thermal_expansion_coefficient DOUBLE_EXPRESSION; 
    reference_temperature DOUBLE_EXPRESSION;
  } 
}

species {
  species "species_1" {
    reference_concentration 1.0;
    fluid "air" {
      diffusion_coefficient 2.0; 
      solutal_expansion_coefficient 3.0;
    }
    fluid "water" {
      diffusion_coefficient 4.0; 
      solutal_expansion_coefficient 5.0;
    }
  }
}

equations {
    navier_stokes {
        boundary_condition {
            # See boundary_conditions.nts
        }

        initial_condition {
            VECTOR_INITIALIZER # See initializer.nts
        }

        # [OPTIONAL]
        gravity_term (0, -9.81);
        source_term {
            VECTOR_INITIALIZER # See initializer.nts
        }
        linear_term {
            VECTOR_INITIALIZER # See initializer.nts
        }
        grad_div_term;
        brinkman_term;
        capilarity_term {
            surface_tension DOUBLE_EXPRESSION;
        }
        pressure_initial_condition {
            SCALAR_INITIALIZER # See initializer.nts
        }
        immersed_boundary_condition {
            # Immersed boundaries implements wall and inlet boundary conditions.
            wall;
            inlet DOUBLE_ARRAY | VECTOR_INITIALIZER;
        }
    }
}
energy {
  boundary_condition {
    # See boundary_conditions.nts
  } 

  initial_condition {
    SCALAR_INITIALIZER # See initializer.nts
  } 

  # [OPTIONAL]
  disable_advection_term;
  disable_diffusion_term;

  phase_change {
    liquid_phase STRING_EXPRESSION; # Fluid name
    solid_phase STRING_EXPRESSION; # Fluid name
    latent_heat DOUBLE_EXPRESSION;
    melting_temperature DOUBLE_EXPRESSION;
  } 

  source_term {
    SCALAR_INITIALIZER # See initializer.nts
  } 

  linear_term {
    SCALAR_INITIALIZER # See initializer.nts
  } 

  immersed_boundary_condition {
    dirichlet DOUBLE_EXPRESSION | SCALAR_INITIALIZER;
    neumann DOUBLE_EXPRESSION | SCALAR_INITIALIZER;
  } 

  species_transport {
    ...
  } 
}
phase_advection {
    # Select the fluid to advect and associate initial and boundary conditions
    fluid STRING_EXPRESSION {
        boundary_condition {
            # See boundary_conditions.nts

        }

        # [OPTIONAL]
        initial_condition {
            SHAPE # See shapes.nts
        }
    }
}

turbulence {
    # Select an LES model
    les_model mixed_scale;
}
boundary_condition {
    left wall;
    left inlet{
        shaped_instructions {
            shape {
                rectangle {
                    corner_1_coordinates (-0.01, 1.);
                    corner_2_coordinates (0.01, 2.);
                }
            }
            instructions {
                @return (mean_velocity*6.0*(y - 1.0)*(1.0 - (y - 1.0))/(1.0*1.0), 0);
            }
        }
        right neumann;
    }
    top wall;
    bottom wall;
}
numerical_parameters {
  time_iterations 1000;
  time_step 1.0; # By default this value is assigned to all solved equations
  cfl 0.5; # [OPTIONAL]
  max_time_step 10.0; # [OPTIONAL], default value: infinity

  time_order_discretization INTEGER_EXPRESSION; # Can be 1 or 2, 1 by default

  # [OPTIONAL] Stop the simulation before the max time iteration number is all the selected test are satisfied.
  stop_tests {
    # [OPTIONAL] Stop if the elapsed time exceed 10.0 s
    elapsed_time 10.0;

    # [OPTIONAL] Stop the simulation if the incompressibility criterion is small enough
    incompressibility 1e-10;
    stationarity_temperature 1e-10; # [OPTIONAL]
    stationarity_velocity 1e-10; # [OPTIONAL]
    stationarity_species 1e-10; # [OPTIONAL]
  }
}
navier_stokes {
    time_step 1.0; # [OPTIONAL]

    # [OPTIONAL], Automatically chosen
    disable_velocity_pressure_method;
    velocity_pressure goda; # goda or timmermans

    # Define the advection scheme (select one)
    advection_scheme implicit o2_centered;
    advection_scheme implicit o1_upwind;
    advection_scheme implicit o2_upwind;
    advection_scheme implicit hybrid_o2_centered_o2_upwind;
    advection_scheme implicit hybrid_o2_centered_o1_upwind;
    advection_scheme explicit o2_centered;
    advection_scheme explicit o2_upwind;

    solver_momentum # See basic_solvers.nts
    solver_pressure # See basic_solvers.nts

    immersed_boundary {
        method direct;
        method direct, linear;
        order 2;
        order 2, 1;
        # Value to assign at outer cells
        outer_value velocity (0.0, 0.0);
    }
}
energy {
    time_step 1.d0; # [OPTIONAL]

    # Select advection scheme (pick one)
    advection_scheme implicit o2_centered;
    advection_scheme implicit o1_upwind;
    advection_scheme implicit o2_upwind;
    advection_scheme explicit o1_upwind;
    advection_scheme explicit weno5_upwind;

    solver # See basic_solvers.nts

    # Phase change numerical parameters [OPTIONAL]
    phase_change {
        method iterative; # Iterative algorithm (Voller)
        method apparent_heat_capacity; # Use Apparent heat capacity method
        method phase_field; # Use Phase Field method
        ...
    }

    immersed_boundary {
        method direct;
        order 2;
        outer_value 4.0;
    }
}
User Interface: numerical block

define phase_advection {
  time_step 1.0;  # [OPTIONAL] replace main time step defined above

  # [OPTIONAL] sampling level to initialize VOF and MOF (default: 10)
  initial_condition_samples 50;

  vof_plic {
    smooth_volume_fraction INTEGER_EXPRESSION;
  }

  mof {
    use_analytic_reconstruction true;  # [OPTIONAL]
    use_filaments BOOLEAN_EXPRESSION;  # [OPTIONAL]
    max_filaments INTEGER_EXPRESSION;  # [OPTIONAL]
    smooth_volume_fraction 2;  # [OPTIONAL]
    ...
  }

  level_set {
    curvature_method normal_divergence;  # [OPTIONAL]
    curvature_method closest_points;  # [OPTIONAL], implies compute_closest_point
    compute_closest_point;  # [OPTIONAL]

    time_order_discretization 0;  # Euler
    time_order_discretization 1;  # RK2 simple
    ...
  }
}
post_processing {
  output_library adios;
  output_frequency 100;

  # Fluid properties
  output_fields conductivity, density, specific_heat, viscosity;

  # Navier-Stokes related variables
  output_fields velocity, divergence, navier_stokes_source_term, permeability, pressure, etc.

  # Multiphase variables
  output_fields volume_fraction;
  output_fields mof_phases;  # Requires mof
  output_fields interface_curvature, level_set_function;  # Requires level_set

  # Species variables
  output_fields species_concentration, species_diffusion_coefficient;

  # Energy variables
  output_fields energy_source_term, temperature;

  # Post-processing variables
  output_fields grid_volume, q_criterion, strain_rate_magnitude, vorticity;

  # Validation/verification variables
  output_fields error, reference_solution, reference_solution_face;

  # Diagnostic quantities computation
  diagnostic_quantities mean_kinetic_energy, mean_pressure, mean_temperature, nusselt_number, wall_shear_stress

  # [OPTIONAL] Print debug information
  debug_io matrix, fields;

  # [OPTIONAL] statistics (compute mean time fields, fluctuation, etc.)
  statistics {
    start_time 1.0;
    compute_mean_time_fields velocity, pressure, temperature
  }
}
# Available basic solver list:
# - hypre_bicgstab or hypre_gmres
# - mumps_metis
# - lis_bi* or lis_*gmres

# Hypre BiCGStab => scalar equation
solver hypre_bicgstab {
  max_iteration 50;
  tolerance 1.0d-10;
  initial_preconditioner left_jacobi; # [Optional]

  preconditioner smg { => more robust
    max_iteration 1;
  }
  preconditioner pfmg { => less robust
    max_iteration 1;
  }
}

# Hypre BiCGStab => momentum equation multiphase flow
solver hypre_bicgstab {
  max_iteration 50;
  tolerance 1.0d-10;
  initial_preconditioner left_jacobi; # [Optional]
}

# Hypre BiCGStab => momentum equation / scalar with stencil of size 2
solver hypre_parcsr_bicgstab {
  max_iteration 50;
  tolerance 1.0d-10;
  initial_preconditioner left_jacobi; # [Optional]

  preconditioner boomeramg {
    max_iteration 1;
    tolerance 1.0d-14;
    strong_threshold 0.25;
    coarsen_type 6;
    aggressive_coarsening_level 0;
    interpolation_type 0;
    post_interpolation_type 0;
    relaxation_type 6;
  }
}
User Interface: solver block

# MUMPS Metis
solver mumps_metis {}  

# LIS solvers
solver lis_bicgstab{
  max_iteration 400;
  tolerance 1.0d-14;
  initial_preconditioner left_jacobi; # [Optional]
  preconditioner iluk{
    fill_level 1; # default 0
  }
  preconditioner iluc{
    drop_tolerance 0.001; # default 0.05
    rate 5.; # default 5
  }
  preconditioner ilut{
    drop_tolerance 0.001; # default 0.05
    rate 5.; # default 5
  }
}
$ mpirun -np 8 notus test.cases/validation/free_convection/square_cavity.nts

Notus - build: release
commit: 08a8cf8
branch: ibd-ane
Compiled by ifort
on Tue Feb 13 09:19:09 CET 2018

Initialization
Grid information
Number of ghost cells: 02
Partitioning: 0004 x 0002 x 0001 = 000000008
Global size: 0032 x 0032 x 0001 = 00000001024

Momentum stencil type: 1_STAR
Pressure stencil type: 1_STAR
Energy stencil type: 1_STAR
Write grids and fields into ‘test_cases/validation/free_convection/output/square_cavity.000000.bp’

Time iteration n°1 time 0.5000E+00

Momentum solver: iterations and residual: 34 0.5108E-15
Pressure solver: iterations and residual: 100 0.2804E-13
Divergence (predicted & corrected): 0.2920E+02 0.8290E-11
Energy solver: residual: 0.8817E-14
Nusselt number, left boundary: 1.627072605124241E+001
Nusselt number, right boundary: 1.627072605341143E+001
Mean velocity magnitude: 1.748763516276342E-001
Stationarity temperature error linf: 4.3704802876646909E-001
Stationarity.velocity.u error linf: 5.2485424141074921E-001
Stationarity.velocity.v error linf: 1.3221265398959479E+000
Stationarity_velocity_error_linf: 1.32212653989595
Divergence (Linf & L2 norms): 2.6182E-09 8.2898E-12

Time iteration n°2 time 1.0000E+00

...
Output

... Time iteration n°287 time 0.1435E+03

Momentum solver: iterations and residual: 21 0.8742E-15
Pressure solver: iterations and residual: 12 0.4509E-14
Divergence (predicted & corrected): 0.8112E-12 0.1013E-16
Energy solver: residual: 0.2073E-15
Nusselt number, left boundary: 1.049093321926628E+001
Nusselt number, right boundary: 1.049093321927709E+001
Mean velocity magnitude: 3.792175505471097E-003
Stationarity temperature: error.linf: 9.4928509497549385E-012
Stationarity.velocity.u: error.linf: 7.5430200280335313E-013
Stationarity.velocity.v: error.linf: 3.9763027939732076E-013
Stationarity.velocity_error.linf: 7.543020028033531E-013
Divergence (Linf & L2 norms): 4.9960E-16 1.0133E-17

Satisfied convergence

Residual stationarity temperature (L2 norm): 9.492850949754938E-12
Residual stationarity velocity (L2 norm): 7.543020028033531E-13

Write grids and fields into 'test_cases/validation/free_convection/output/square_cavity.000287.bp'
IO / Visualisation

Domain is partitioned, data are distributed

→ How to write and plot data efficiently on thousands of processors?

Use of ADIOS library (Oak Ridge National Laboratory)

- Open-source
- Adaptable IO System
- Simple and flexible way to describe the data
- Masks IO parallelism
- Different methods: POSIX, MPI-IO, aggregation
- From 1 to 100 000 processors
- .bp files

Notus IO

- A list of data is created, printed at the end of the time loop
- Add a field anywhere in the code:
  ```
  use mod_field_list
  call add_field_to_list(print_list, enstrophy, 'enstrophy')
  ```

Visualisation of the results → VisIt (Lawrence Livermore National Laboratory)

- Open-source, Sequential and Parallel, etc.
1. Notus code
2. Development environment
3. Installation, compilation
4. Run notus
5. User interface
6. IO / Visualisation
7. Architecture, some development keys, user mode
8. Documentation
9. Notus Verification & Validation tools
10. Notus Porting & Performance tools
11. Development tools
12. Git usage
# Development environment - Architecture

## Project tree

- `src`
- `std`
- `test_cases`
- `tools`

## Source tree

- `src/lib`
  - 1st level:
    - 2 or 3 sub-levels
- `src/notus`
  - `notus.f90`
  - `ui/`
- `src/doc`
Some development keys

## Naming

- Hundreds of variables
  - self explanatory variable names *(velocity, pressure, temperature, ...)*
  - as few abbreviations as possible

- Prefix
  - module starts with `mod_`
  - scalar variable module starts with `variables_`
  - field array module starts with `fields_`
  - new derived types module starts with `type_`
  - new types starts with `t_`
    - ex: `struct_face_field velocity%u %v`
  - scalar names associated to an equation suffixed *(navier.time.step, etc.)*

- Explicit routine name
  - `solve_navier`
  - `compute_mean_velocity`
  - `add_div_diffusive_flux_to_matrix`

→ nearly “guessable” variables
→ Auto-documentation
→ **Use git grep to locate variables, routines, etc.**
Some development keys

Code formating

- tab = 3 characters
- line = 132 characters max
- **Automatic formatting before committing**: formatcode.sh
  
  Usage: formatCode.sh [OPTIONS]
  -h print usage and exit
  -p format only modified files
  -f format only given files
  -c COMMIT format only the given commit
Some development keys - Masking parallelism

Numerical domain and process ghost cells

- The global domain is partitioned subdomain
- Addition of a few layers of cells surrounding the local domain: \( nx \cdot ny \cdot nz \) cells

MPI generic routines to exchange data

- 2D/3D, whatever overlapping zone size
- Integer, double
- Cell array, or vector defined on staggered grid
  
  ```
  call mpi_exchange(pressure)
  call mpi_exchange(velocity)
  ```
- → Mandatory after any spatial derivative computations

Global reduction routines

- encapsulate MPI ones
- generic routines for min, max of local arrays, sum of scalars
Some development keys - A set of user routines

Concept

- Avoid a user to known very well the code
- User directory `src/lib/user`
- Void routine by default
- Uncomment, modify, compile
- Initial condition
- Boundary conditions
- Source terms
- Computation of physical properties

Example

```fortran
do k=1,nz
   do j=1,ny
      energy_boundary_type%left(j,k)=cell_boundary_type_dirichlet
      temperature_boundary_value%left(j,k)=...
   enddo
enddo
```
Some development keys - useful modules

- use variables_domain
  → spatial_dimension, etc.
- use variables_grid
  → nx, ny, nxu, nyv, is, ie, isu, ieu, etc.
- use variables_spatial_step
  → dx(nx), dx_u(nxu), etc.
- usr variables_time_discretization
  → time, global_time_step, time_iteration, etc.
Documentation - Doxygen

For writing software reference documentation

- Documentation is written within the code
- Open-source, generates html, pdf, latex files

Doxygen and Notus

- [https://doc.notus-cfd.org](https://doc.notus-cfd.org)
- Upper level doc: installation, git, architecture, [howtos, best practises, etc.](https://doc.notus-cfd.org) (markdown format)

One documentation group per src/lib subdirectories (physics, numerical methods, io, etc.)

```
cat /src/lib/mesh/grid_generation/doc.f90

!> @defgroup grid_generation Grid Generation
!! @ingroup mesh
!! @brief Compute grid coordinates and spatial steps
```

Documentation inside each Fortran files

```
cat /src/lib/mesh/grid_generation/create_regular_mesh.f90

!> Create a regular Cartesian mesh (constant step size per direction).
!! The mesh is created in two steps:
!! 1. Provide global face coordinates
!! 2. Compute local variables (coordinates and space steps)
!! The second step is automated in complete_mesh_structure
!! Require the number of points per directions
!! ingroup grid_generation
subroutine create_regular_mesh()
```

---

Notus team (I2M / TREFLE)  Notus first steps (0.3.0)  June 5th 2019  43 / 58
Contents

1. Notus code purposes
2. Development environment
3. Installation, compilation
4. Run notus
5. User interface
6. IO / Visualisation
7. Architecture, some development keys, user mode
8. Documentation
9. Notus Verification & Validation tools
10. Notus Porting & Performance tools
11. Developement tools
12. Git usage
Verification

- proves that the continuous model is solved precisely by the discrete approach
  - analyses the numerical solution of equations
  - quantifies and reduces of the numerical errors
  - computes spatial and temporal convergence orders

→ mainly a mathematical and computing process, unlinked to physical problem

Validation

- analyses the capacity of a model to represent a physical phenomena
  - compares numerical solution to experimental results
  - identifies and quantifies errors and uncertainties of continuous and discrete models, and experience

→ Accumulation of evidence that the code works!
Verification

2 main steps

- no bug in the code or inconsistent solution
- quantify numerical errors
  - start from an exact (built) solution
  - compute errors, convergence order
  - compare the given order to the expected one

Error sources

- coding bug
- numerical stability condition not satisfied
- insufficient spatial or temporal convergence
- iterative methods not converged
- rounding errors

Hypothesis: smoothed solution in the asymptotic convergence zone

N discrete solutions \( f_k (1 \leq k \leq N) \)

\[
f_{h \to 0} = f_k + Ch_k^p + O(h_k^{p+1})
\]

\[
p_k = \frac{\log \left( \frac{E_k}{E_{k-1}} \right)}{\log \left( \frac{h_k}{h_{k+1}} \right)}
\]

where \( E_k = f_{\text{exact}} - f_k \)

<table>
<thead>
<tr>
<th>mesh</th>
<th>( L_\infty ) error</th>
<th>Order</th>
<th>( L_2 ) error</th>
<th>Order</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>2.53e-03</td>
<td>n/a</td>
<td>6.87e-04</td>
<td>n/a</td>
</tr>
<tr>
<td>20</td>
<td>6.49e-04</td>
<td>1.97</td>
<td>1.69e-04</td>
<td>2.02</td>
</tr>
<tr>
<td>40</td>
<td>1.63e-04</td>
<td>1.99</td>
<td>4.22e-05</td>
<td>2.00</td>
</tr>
<tr>
<td>80</td>
<td>4.08e-05</td>
<td>2.00</td>
<td>1.05e-05</td>
<td>2.00</td>
</tr>
</tbody>
</table>
Validation

Analyses the capacity of a model to represent a physical phenomena

- no exact solution
- post processing of physical parameter (velocity plot, Nusselt numbers, lift, drag, etc.)
- comparison with experience or other code
- quantify error and uncertainty
- 3 meshes $\rightarrow$ convergence order $\rightarrow$ Richardson extrapolation
1 - compute convergence order

Run the same case varying a parameter (mesh or time step)

- `json` file
  ```json
  {
  "number_of_cells": [ 100, 25], "time_step": 0.5},
  {
  "number_of_cells": [ 200, 50], "time_step": 0.25},
  {
  "number_of_cells": [ 400, 100], "time_step": 0.125},
  {
  "number_of_cells": [ 800, 200], "time_step": 0.0625}
  ```

- Python script: `./notus_grid_convergence -np 8 --doxygen test_case_name`
  - run (interactively or submission) the test case with different meshes
  - collect the results of the chosen quantities
  - compute convergence order and extrapolated values
  - output to doxygen format

2 - non regression

- list of V&V test cases files
- quick or full validation
- run the test cases with bash script
- results in `txt` file: OK, NO, FAIL, etc.
- commit the results (one per architecture) to Git repository
- `notus.py` script
To be developed

- Work in another directory than validation or verification ones
- As much as possible, use formula inside the .nts file
- Integration into notus test case list:
Check Portability and Performances

Portability

- Associated to V & V process
- Numerical solutions should be **independent of:**
  - compiler editors, compiler versions, MPI libraries, etc.
  - computer architectures and processor numbers
- Notus portable on:
  - GNU + OpenMPI; Intel + MPT; Intel + IntelMPI; Intel + BullXMPi
  - Sequential and Parallel versions
  - → “Same” results between $10^{-8}$ and $10^{-15}$

Performances

- Compare measured scalability to the expected one
- Identify and measure relevant parts of the code
  - partitioning
  - initialization
  - time loop: equation preparation, solvers (external), I/O
- Lot of functionalities: **identify the relevant test cases**
- Determine optimal use of supercomputers (nodes number per core)
Notus, performance tools

Objectives

- Verify weak and strong scalability
- Verify I/O performance
- Ensure non regression of these performances
- On several supercomputers (from local to PRACE one)

Step 1, scalability at test case level

Template directory

- notus template .nts file
- submission template file (depending of the workload manager)

Submission bash script

- ./submit_jobs.sh -t weak -a 9 -c 40 -m 16 -s template_sub_curie -q ccc.msub
- ./submit_jobs.sh -t strong -i 3 -a 9 -c 512 -m 16 -s template_sub_curie -q ccc.msub
- ./submit_jobs.sh -t strong.node -c 100 -m 16 -s template_sub_curie -q ccc.msub

→ copy template directory
→ adapt template files
→ submit jobs

Concatenation bash script

- ./concatenate_cpu_times.sh -t weak -a 9 -c 40 -m 16

<table>
<thead>
<tr>
<th>CPU</th>
<th>0.26000E+01</th>
<th>0.86140E+00</th>
<th>0.94443E+00</th>
<th>0.79417E+00</th>
</tr>
</thead>
<tbody>
<tr>
<td>128</td>
<td>0.26000E+01</td>
<td>0.86140E+00</td>
<td>0.94443E+00</td>
<td>0.79417E+00</td>
</tr>
<tr>
<td>256</td>
<td>0.29297E+01</td>
<td>0.10660E+01</td>
<td>0.10462E+01</td>
<td>0.81751E+00</td>
</tr>
<tr>
<td>512</td>
<td>0.30754E+01</td>
<td>0.11369E+01</td>
<td>0.11025E+01</td>
<td>0.83590E+00</td>
</tr>
<tr>
<td>1024</td>
<td>0.38859E+01</td>
<td>0.16025E+01</td>
<td>0.13959E+01</td>
<td>0.88751E+00</td>
</tr>
<tr>
<td>2048</td>
<td>0.43207E+01</td>
<td>0.18807E+01</td>
<td>0.15359E+01</td>
<td>0.90404E+00</td>
</tr>
<tr>
<td>4096</td>
<td>0.47281E+01</td>
<td>0.22302E+01</td>
<td>0.16268E+01</td>
<td>0.87108E+00</td>
</tr>
<tr>
<td>8192</td>
<td>0.65902E+01</td>
<td>0.32613E+01</td>
<td>0.23815E+01</td>
<td>0.94744E+00</td>
</tr>
</tbody>
</table>

Notus team (I2M / TREFLE) Notus first steps (0.3.0) June 5th 2019
Notus, performance tools

Weak scalability on Curie and Occigen supercomputers

→ 50³ cells per core, number of core increases, constant CPU time expected

![Weak scalability study on Curie and Occigen supercomputers](image1)

Strong scalability

→ constant number of global cells, number of core increases, linear speed-up expected

![Strong scalability](image2)

HYPRE / LIS comparison: BiCGStab + Jacobi

![HYPRE / LIS comparison](image3)
Contents

1. Notus code purposes
2. Development environment
3. Installation, compilation
4. Run notus
5. User interface
6. IO / Visualisation
7. Architecture, some development keys, user mode
8. Documentation
9. Notus Verification & Validation tools
10. Notus Porting & Performance tools
11. Developement tools
12. Git usage
Developement tools

- vim → tools/vim_syntax
- emacs → tools/emacs/.emacs
Development environment - Git

About Git VCS

- Records changes to a file(s) over time
- Allows to revert files back to a previous state
- Reverts the entire project back to a previous state
- Compares changes over time
- See who last modified something
- Recovers lost files
- Fully mirrors the repository

→ https://openclassrooms.com/fr/courses/2342361-gerez-votre-code-avec-git-et-github
Development environment - Git

### Branch model

- One directory
- One version = one branch
- Official Notus repository `master` and `dev` branches cloned to local repository

### Local branches management

create a branch, checkout a branch:

```bash
$ git branch my-branch
$ git checkout my-branch
```

merge branch:

```bash
$ git merge branch-to-merge
```

rebase from dev:

```bash
$ git rebase dev
```

branches available:

```bash
$ git branch -a
```

get differences between two branches:

```bash
$ git diff branch_name
```

### Server dialogue

get the last dev version:

```bash
$ git pull official dev
```

push a branch to your origin remote repository:

```bash
$ git push --set-upstream origin
```
The Three States, basic workflow

- File modification in the working directory
- Stage the files
- Commit

Few commands to start with Git

- Change file with text editor
  - `$ git status`
  - `$ git add file-name`
  - `$ git commit`
- → add a comment to your commit

Development environment and porting - Git
Conclusion

- Use of some standard development tools (Git, CMake, Doxygen)
- Use of specific libraries: IO, solvers
- Single Doxygen documentation: concepts, installation, modeling, subroutines
- Different users (from student to researcher, from modeling to numerical methods)
- Different computers
- A few scripts, easy to use and modify for:
  - installation
  - execution
  - V&V
  - scalability studies

→ **ongoing project, version 0.3.0 only !**