CFD code Notus (0.2.0) :
environment, architecture,
verification and validation, performances

Stéphane Glockner,
Mathieu Coquerelle, Antoine Lemoine, Joris Picot

I2M
Université de Bordeaux, Bordeaux-INP, CNRS UMR 52 95

glockner@bordeaux-inp.fr, https://notus-cfd.org

September 11th 2017
1. Notus code
2. User interface
3. Development environment
4. Installation, compilation
5. Architecture, some development keys
6. Documentation
7. Third party libraries: solvers and IO
8. Notus Verification & Validation tools
9. Notus Porting & Performance tools
What is (not) Notus

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

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

Intended users

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

What is not Notus

- A concurrent of, a commercial tool, a click button code
Notus - some objectives

Objectives

- Rationalize research efforts
- Benchmark methods on identified physical test cases
- Numerical toolbox
- Towards numerical experiments

Means

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

Interfaces

- Fluid / fluid interfaces (advection, surface tension)
- Fluid / solid boundaries (with or without wetting)
- Fluid / porous media interface
- Fluid / solid phase change

2nd order “everywhere” ? Efficiency ?

- 2nd order advection scheme, one-fluid model ?
- 2nd order immersed boundaries, but scalable ?
- 2nd order interface reconstruction, even if immersed boundaries ?
- 2nd order interface reconstruction, and curvature ?
- ...

Glockner (I2M / TREFLE)
Journée RNS COSIN
September 11th 2017
### Notus: models

#### Domain
- 2D/3D Cartesian, axisymmetric
- 2nd order immersed boundary

#### Incompressible Navier-Stokes equations
- Buoyancy force (Boussinesq approximation)
- Surface tension force (CSF model)

#### Energy equation
- liquid/solid phase change

#### Multiphase immiscible flows
- N advected phases

#### Species transport equations
- N passive scalars

#### Turbulence
- Large Eddy Simulation model (mixed scale)
Notus: numerical methods

Discretisation

- 2D/3D Cartesian Finite Volume on staggered grids, automatic partitioning
- Time discretisation: implicit, up to 2nd order
- Spatial discretisation: up to 2nd order implicit schemes (advection and diffusion)
- Spatial discretisation: 3rd / 5th order WENO schemes (advection)
- 2nd order immersed boundary method

Navier-Stokes

- Velocity/pressure coupling: time splitting methods (Goda, Timmermans)
- 2nd order open and traction boundary condition
- Surface tension: Closest-Point method to compute curvature (→ Level-set only)
- Wetting: macroscopic/microscopic approach

Fluid / fluid interface representation and transport

- Volume-of-Fluid method / PLIC
- Moment-of-Fluid method 2D / 3D
- Level-set / WENO
- MOF + Level-set
User Interface

Concept

- ASCII .nts files
- Self-explanatory keywords, precise grammar
- Efficient parser that supports:
  - variable declaration
  - formula
  - 'include'
  - if condition and loop

Organisation

- Physical fluid properties data base
- One .nts file per test case
  - domain{}
  - mesh{}
  - modelisation{}
  - 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 "one";}
    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 mumpsmetis;
    }
}

post_processing {
    output_library adios;
    output_frequency 1;
    output_fields temperature;
}
Contents

1. Notus code
2. User interface
3. Development environment
4. Installation, compilation
5. Architecture, some development keys
6. Documentation
7. Third party libraries: solvers and IO
8. Notus Verification & Validation tools
9. Notus Porting & Performance tools
## 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

### Compilers and MPI libraries

- GNU compilers (> 5.2) and Open MPI (2.10)
- Intel compilers (> 14) and SGI MPT (2.11) and BullxMPI (1.2.8.3)
- IBM XL compilers (14.1) and MPI libraries (2.21.1)

### Supercomputers

- Curie at TGCC
- Occigen at CINES
- Turing at IDRIS
- Condor at I2M
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

Notus Git repository server

https://git.notus-cfd.org

git clone https://git.notus-cfd.org/notus/notus.git notus
Installation: third party libraries

Third party libraries

- ADIOS (MXML), HYPRE, MUMPS (METIS, Scalapack), LIS
- BLAS & LAPACK → system
- Simplify the installations of these libraries
- Be sure of the version installed
- Git repository with tarballs
  
  git clone https://git.notus-cfd.org/notus/notus_third_party.git notus_third_party

- Installation script (default in $HOME/usr)
  
  ./build_notus_third_party_lib.sh -a

Options:

Compiler name: --cc --fc ...

MPI wrapper name: --mpicc --mpifc ...

Install libraries separately: --adios --hype ...

Download a new version and install it: --hype-version 2.12.0 ...

Change installation directory: --install-dir
Development environment - CMake - Build Notus

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 done for several development environnement: GNU, Intel, etc.
- Find third party libraries
- Build scripts available for specific computers: linux workstation, condor, occigen, avakas, curie, etc.
- MPI (only) release or debug (default) builds

  $ ./build_notus_curie.sh -h
  Usage: buildcmakecondor.sh [OPTIONS]
  -c clean the build directory
  -s sequential build (default: MPI)
  -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

- Easily adaptable
Development environment - Architecture

**Project tree**

```
src
std
test_cases
tools
```

**Source tree**

- `src/lib`
  - 1st level:
    - Core
    - Geometry
    - I/O
    - Modeling
    - Discretization
    - Lin. Sys. Solver
    - User
    - Post Process

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

- Explicit routine name

```plaintext
solve_navier
compute_mean_velocity
add_div.diffusive_flux.to.matrix
```

→ nearly “guessable” variables → Auto-documentation → Use “git grep” to locate variables, routines, etc.
Numerical domain and process ghost cells

- The global domain is partitioned subdomain
- Addition of a few layers of cells surrounding the local domain: $nx.ny.nz$ cells

MPI generic routines to exchange data

- 2D/3D, whatever overlapping zone size
- Integer, double
- Cell array, or vector defined on staggered grid

```fortran
  call mpi_exchange(pressure)
  call mpi_exchange(velocity)
```

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

- Void routine by default
- Uncomment, modify, compile
- Specific initial condition
- Variable boundary conditions
- Source terms
- Computation of physical properties
- Schemes
- → User directory
- → Avoid a user to known very well the code

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
endo
```
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
- Upper level doc: installation, git, architecture, howtos, etc. (markdown format)
- One documentation group per src/lib subdirectories (physics, numerical_methods, io, etc.)

```plaintext
cat /src/lib/mesh/grid_generation/doc.f90
>> @defgroup grid_generation Grid Generation
!! @ingroup mesh
!! @brief Compute grid coordinates and spatial steps
```

```plaintext
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()
...
```
Contents

1. Notus code
2. User interface
3. Development environment
4. Installation, compilation
5. Architecture, some development keys
6. Documentation
7. Third party libraries: solvers and IO
8. Notus Verification & Validation tools
9. Notus Porting & Performance tools
## 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

### Notus IO

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

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

- Open-source
- Sequential and Parallel
- ...
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** $f_k (1 \leq k \leq N)$

$$f_{n \to 0} = f_k + Ch_k^p + O(h_k^{p+1})$$

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

where $E_k = f_{exact} - f_k$

<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 → convergence order → Richardson extrapolation

<table>
<thead>
<tr>
<th>Mesh</th>
<th>Nusselt nb.</th>
<th>Order</th>
<th>Velocity</th>
<th>Order</th>
</tr>
</thead>
<tbody>
<tr>
<td>32</td>
<td>1.0490e+01</td>
<td>na</td>
<td>3.7921e-03</td>
<td>na</td>
</tr>
<tr>
<td>64</td>
<td>9.1842e+00</td>
<td>na</td>
<td>3.6811e-03</td>
<td>na</td>
</tr>
<tr>
<td>128</td>
<td>8.9013e+00</td>
<td>2.2070</td>
<td>3.6387e-03</td>
<td>1.3913</td>
</tr>
<tr>
<td>256</td>
<td>8.8424e+00</td>
<td>2.2635</td>
<td>3.6277e-03</td>
<td>1.9381</td>
</tr>
<tr>
<td>512</td>
<td>8.8292e+00</td>
<td>2.1622</td>
<td>3.6249e-03</td>
<td>1.9957</td>
</tr>
<tr>
<td>Ext.</td>
<td>8.8254e+00</td>
<td></td>
<td>3.6240e-03</td>
<td></td>
</tr>
<tr>
<td>Réf.</td>
<td>8.8252e+00</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
1 - compute convergence order of a test case

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
Exemple of Output of the non regression process

$ ./notus_validation.sh -h
Usage : notus_validation.sh [OPTIONS]
-s sequential validation (default: parallel)
-d 2/3 2D or 3D validation (default: 2D and 3D)
-l long validation (default: false); check for special keywords in case.nts and run the case several times
-h print usage

$ cat notus_validation.txt

Test case name     Validated Converged Time iteration Error
ibd_laplacian_dirichlet.nts  FAIL
ibd_laplacian_neumann.nts  FAIL
poiseuille.nts          OK OK 356 1.3877787807814457E-17
poiseuille_periodic.nts  OK OK 69 1.3877787807814457E-16
poiseuille_viscosity.nts OK OK 2989 0.0000000000000000E+00
ibd/ibd.poiseuille.nts  FAIL
level_set/sheared_2D.nts NO N/A 200 3.8200452689984843E-08
mof_analytic_periodic.nts OK N/A 141 2.2204460492503131E-16
mof_analytic/sheared.nts OK N/A 1000 1.9984014443252818E-15
mof_minimization/sheared.nts OK N/A 1000 2.2204460492503131E-16
vof_plic_periodic.nts   OK N/A 141 3.3306690738754696E-16
vof_plic/sheared.nts    OK N/A 1000 3.3306690738754696E-16
ball_equilibrium.nts    NO OK 1128 1.4963675386815269E-07
square_cavity.nts      OK OK 291 5.3942093847236805E-14
driven_cavity.nts      OK OK 3449 5.1625370645069779E-15
dam_break.mof.nts      OK N/A 50 2.5313084961453569E-14
dam_break_vof_plic.nts OK N/A 50 3.556490919295356E-14
solitary_wav           NO N/A 450 5.1368178637115763E-04
solitary_wav           NO N/A 500 7.6377097850394010E-03
square_cavity.nts      NO OK 235 3.0898306135895837E-10
...
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

Most of CPU time in linear system solvers
→ Third party libraries

HYPRE library (Livermore USA)
- BiCGStab, GMRES iterative solvers
- Geometric and algebraic preconditioners

LIS library (SSISC Japan)
- BiCGStab, GMRES iterative solvers
- ILU family preconditioners

MUMPS (Cerfacs / INRIA, France)
- Direct solver
- Mainly for 2D matrix
- PORD, Metis graph partitioners
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`

|   | 0.26000E+01 | 0.86140E+00 | 0.94443E+00 | 0.79417E+00 | 0.29297E+01 | 0.10660E+01 | 0.10462E+01 | 0.81751E+00 | 0.30754E+01 | 0.11369E+01 | 0.11025E+01 | 0.83590E+00 | 0.38859E+01 | 0.16025E+01 | 0.13959E+01 | 0.88751E+00 | 0.43207E+01 | 0.18807E+01 | 0.15359E+01 | 0.90404E+00 | 0.47281E+01 | 0.22302E+01 | 0.16268E+01 | 0.87108E+00 | 0.65902E+01 | 0.32613E+01 | 0.23815E+01 | 0.94744E+00 |
Notus, performance tools

Weak scalability on Curie and Occigen supercomputers

→ $50^3$ cells per core, number of cores increases, constant CPU time expected

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

Strong scalability

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

![Strong scalability](image)

HYPRE / LIS comparison:

BiCGStab + Jacobi

![HYPRE / LIS comparison](image)
Step 2 under progress, non regression list of representative test cases

- get reference times for each one and each target supercomputer
- bash script to run all the performance study
- comparison, OK, NO, FAIL
- commit the results (one per architecture) to Git repository
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 0.2.0 only!