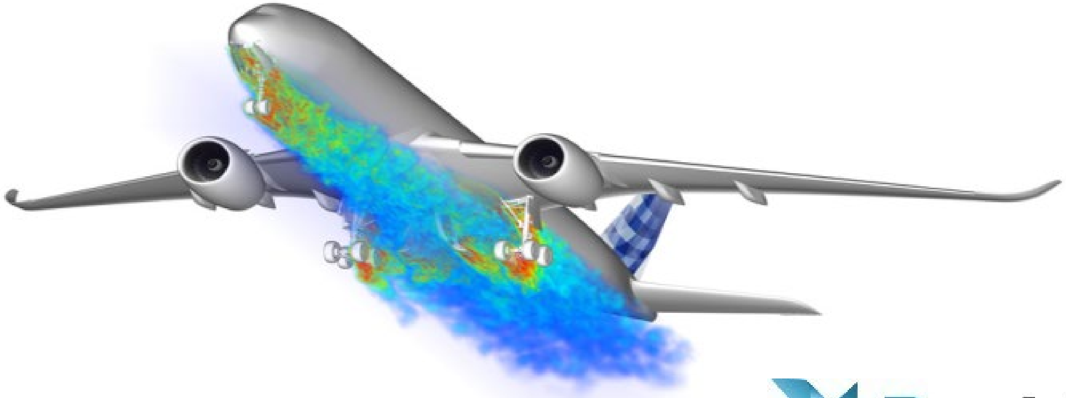
WORKING BY YOUR SIDE
TO TACKLE YOUR CRITICAL CHALLENGES



Collaborative development of physical models in ProLB using dataflow programming

Denis Ricot

Teratec, HPC & simulation workshop
15 June 2022

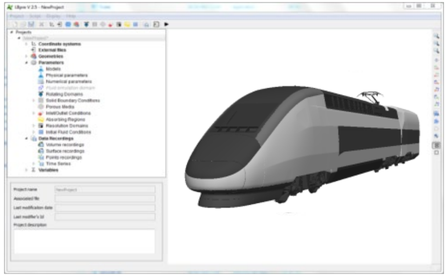
Agenda

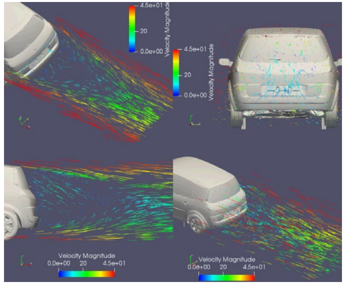
LABS TO PROLB

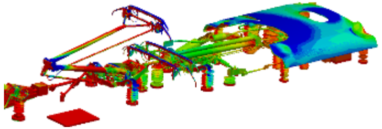
- LaBS software for R&D :
  - Developed collectively in the framework of collaborative projects since the “LaBS” Project 2009-2014
  - Commercial version since 2017 : ProLB

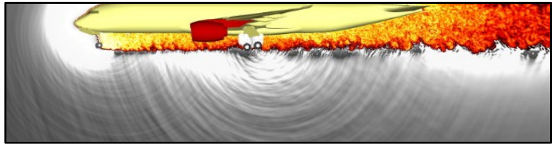
...

- Commercial CFD software : ProLB
  - Developed and distributed by CS GROUP on the LaBS software base
  - Advanced industrial features: local mesh refinement, rotating mesh, porous media, code coupling, advanced output functions
  - State of the art of parallel performance on standard HPC servers
  - Used for projects at Renault, Airbus, Nissan, Framatome...









PROLB : A LATTICE BOLTZMANN SOLVER

- Lattice Boltzmann Method :

$$f_\alpha(\vec{x}, t+\Delta t) = f_\alpha^{collision}(\vec{x} - \vec{c}_\alpha \Delta t, t)$$

Propagate the distribution functions

$$\rho(\vec{x}, t) = \sum_\alpha f_\alpha(\vec{x}, t)$$

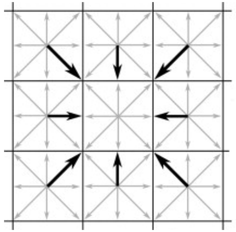
$$\rho(\vec{x}, t) u(\vec{x}, t) = \sum_\alpha \vec{c}_\alpha f_\alpha(\vec{x}, t)$$

Compute the macroscopic variables

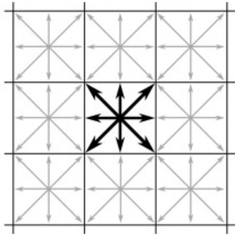
$$f_\alpha^{collision}(\vec{x}, t) = \left(1 - \frac{1}{\tau}\right) f_\alpha(\vec{x}, t) + \frac{1}{\tau} f_\alpha^{eq}\left(\rho(\vec{x}, t), u(\vec{x}, t)\right)$$

Collide

Propagate functions along the lattice directions



Collide

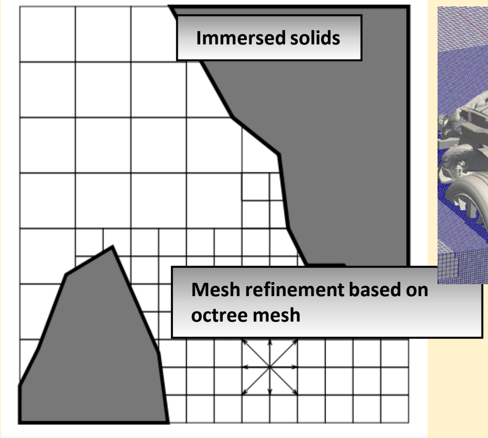


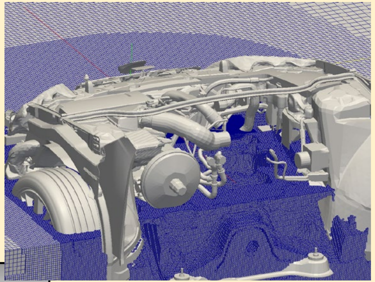
- Simple arithmetic calculations, first order neighbors, explicit in time → recover fluid flows with the same physics than Navier-Stokes equations with very low numerical dissipation
- Very simple to implement on uniform mesh with simple boundary conditions

PROLB : A LATTICE BOLTZMANN SOLVER

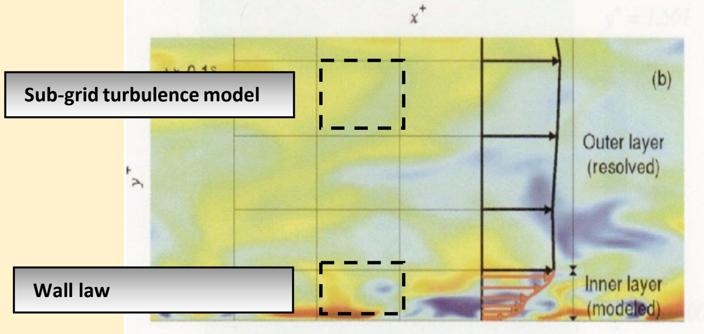
- … but the simplicity and strength of LBM are put to the test by many complementary ingredients that are mandatory to get an industrial CFD solver :

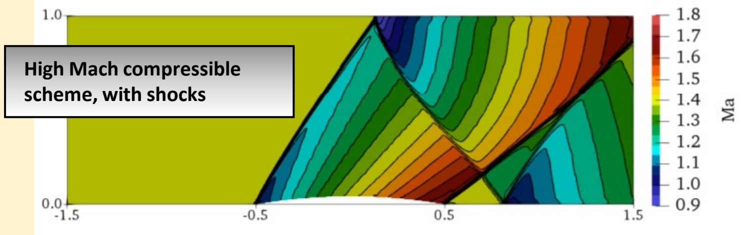
Complex data structure and features





Advanced physical models



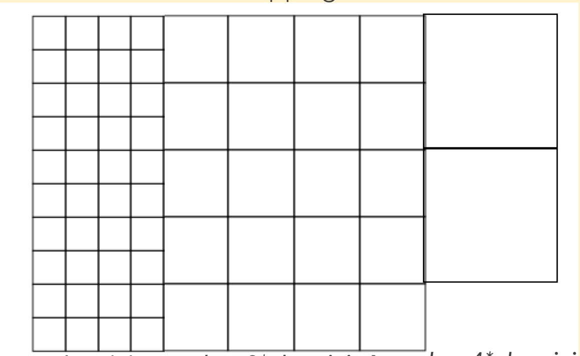


PROLB : A LATTICE BOLTZMANN SOLVER

- … but the simplicity and strength of LBM are put to the test by many complementary ingredients that are mandatory to get an industrial CFD solver :

Complex data structure and features

- LBM : exact propagation of the variables
  - → dx/dt = constant for all the mesh levels
  - → Local time-stepping



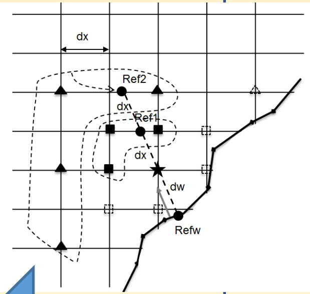
dx_mini : compute all the time iterations

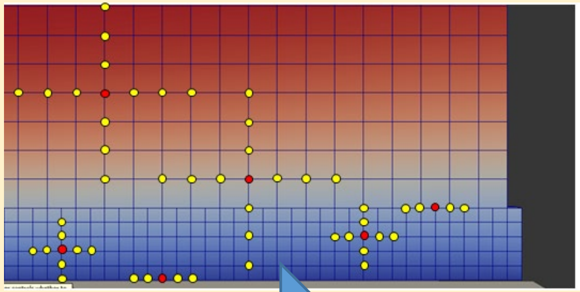
dx = 2 dx_mini → dt = 2*dt_mini compute one out of two times*

dx = 4 dx_mini : compute one out of for times*

Advanced physical models

- For advanced physics algorithms : need for extended stencils which can eventually cross two levels of refinement





High interlacing between the HPC “backend” (data-structure, parallelization features..) and the numerical schemes

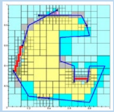
EFFICIENT COLLABORATIVE DEVELOPMENT OF THE PHYSICAL MODELS

- Challenge for an efficient collaborative development of physical models :
  - Quick prototype of complex physics algorithms, with minimal HPC/programming skills
  - Validate it on simple academic cases but also on complex geometries
  - Simpler/quicker steps between the prototype version and the industrial version
- Solution found by CS GROUP, based on three pillars :
  - Clear separation between the physics team and the HPC kernel team : not the same persons, not same objectives, not same skills, …
  - Physics developers have no direct access to low-level data structure : access only though API (i.e. just some access functions)
  - No management of the parallelism by the physics developers and simplified management of the "complexity" of the space and time dependencies of data : dataflow programming

Development and maintenance of the HPC kernel

CS GROUP

HPC kernel development team



- Access to kernel "services" and data structure via an API

- Dataflow programming & domain-specific language

R&D on physical models

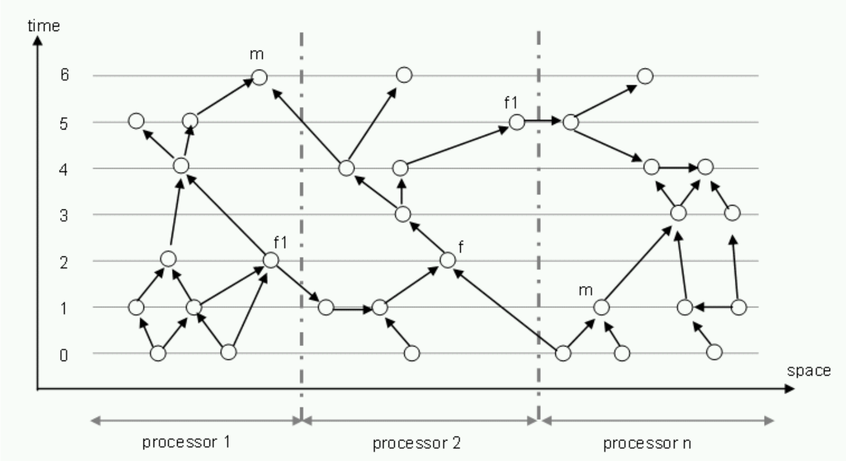
CS GROUP

R&D and physical model integration team

Aix Marseille université · ÉCOLE CENTRALE LYON · ONERA · CERFACS · MATELYS Research Lab · AIRBUS · RENAULT · CETHIL UMR 5008

DATAFLOW PROGRAMMING

- Dataflow programming is a programming paradigm that models a program as a directed graph of the data flowing between operations
- For a simulation software :
  - “operations” means the base computations and data manipulations (gradient, time integration, multiply/add variables, invert a matrix…)
  - “data” are the physical variables to be calculated
  - data “flowing” means the time and spatial dependencies of data between each other.
    - In a framework of parallel computation, the “flow” also includes the dependency of data thought different processors



DATAFLOW PROGRAMMING + DECLARATIVE PROGRAMMING

- Dataflow programming, some examples :
  - TensorFlow
  - Lustre (langage)

- Dataflow programming is well suited to be used with declarative programming
  - Declarative programming is a non-imperative style of programming in which programs describe their desired results without explicitly listing commands or steps that must be performed.
    - describe *what* the program must accomplish, rather than describe *how* to accomplish it as a sequence of the programming language primitive
    - this is in contrast with imperative programming, which implements algorithms in explicit steps.
  - Declarative programming are quite rare, and are mainly used for Domain-Specific Language (DSL) :
    - SQL, Prolog

DSL FOR PROLB PHYSICAL MODELS

- Such a Domain-Specific Language with dataflow programming approach has been created by CS GROUP for the development of ProLB
  - Main software architect : Jean-Pierre Lahargue
  - Very simplified language, only focused on the development of physics schemes in ProLB
  - Very few rules, with a simple syntax (few keywords, …)
  - The language parser/interpreter and the associated dataflow computation (task schedule calculation) is implemented as a dedicated module in the ProLB kernel (C++ / OO programming)

- Not so “Specific”
  - Not specific to the octree mesh, not specific to any physical/mathematical computation
  - Adapted to compute explicit time-marching scheme on generic data
  - Only the local time-step management is very specific to LBM algorithms

CS
GROUP

DSL FOR PROLB PHYSICAL MODELS

- All the « source » code is written into a single text file :
  - Data to be computed are represented by datatype :

```
DATA_TYPES
    0 DATA_TYPE fdist        19
    1 DATA_TYPE fcoll        19
…
END DATA_TYPES
```

a data structure with 19 fields (double or float), and 1 to 3 time slots (depending on the following declarations)

  - A computation on a DATA_TYPE is defined as :

Langage keyword

DataType to be computed at the current time

Node filter function

Predefined node catagory

Langage keyword

Callback function to be used (→ C++ function)

Langage keyword : at which local time step we want to compute ? (INIT,ALWAYS, ODD, EVEN)

Dependency filter function

DataType requested at the input of the computation

List of the neighbor indexes to be used

Time slot to be used

```
COMPUTE  fdist  ON_nearBoundary  MIDDLE  BY  PropagateMiddle  ALWAYS  FROM  fcoll[0..18,1] macro[0,0]
COMPUTE  fcoll  ON_awayBoundary  MIDDLE  BY  Collide          ALWAYS  FROM  fdist[0,0] grad[0,0]
[…]
```

  - Other declarations in the DSL source file : list of the callback functions (computation functions), list of the node filter functions, …

DSL FOR PROLB PHYSICAL MODELS

Langage keyword	DataType to be computed at the current time	Node filter function	Predefined node catagory	Langage keyword	Callback function to be used (→ C++ function)	Langage keyword : at which local time step we want to compute ? (INIT,ALWAYS, ODD, EVEN)	Dependency filter function	DataType requested at the input of the computation	List of the neighbor indexes to be used	Time slot to be used
COMPUTE	fdist	ON_nearBoundary	MIDDLE	BY	PropagateMiddle	ALWAYS	FROM	fcoll	[0..18,	1]

```
COMPUTE  fdist  ON_nearBoundary  MIDDLE  BY  PropagateMiddle  ALWAYS  FROM  fcoll[0..18,1] macro[0,0]
```

- Using the node type keywords (`MIDDLE`,…) and the node filter functions (e.g. `ON_nearBoundary`), all the mesh nodes must be treated
  - The node filter functions are C++ functions written by the physics developer that analyze the properties of a node and return a Boolean
- Main rules for the DSL coding:
  - Single assignment rule : a DataType is calcuted only once. Overwriting a DataType is impossible.
    - Algorithms such as predictor/corrector (not used in standard LBM) should use two different DataTypes for the calculation : one for the predicted value, one other for the corrected one
  - A DataType is calculated/modified only for the current time step (current time step for its mesh height/size).
    - No modification at time t of its value at time t-1
- The order of writing the scheme is free (declarative programming)
- Only ~200 lines of command for the full industrial LBM scheme

GROUP

PROLB STEPS

Steps

Pre-process

- Detect
- Input
- Sewing
- Octal
- Populate

[…]

- Scheduler
- Balance
- Migrate
- ReduceStatistics
- FieldMapping

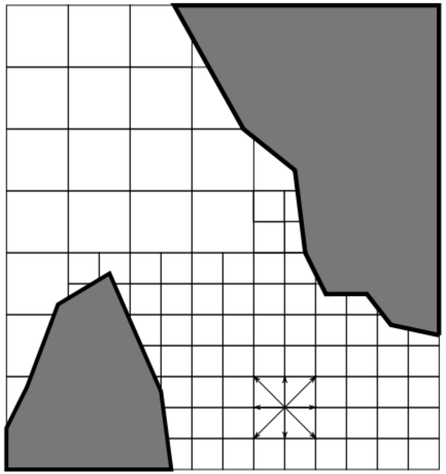
Solve

- Solver

Convert files

- Merger
- Poster

Parallel octal mesher, with a first domain decomposition



PROLB STEPS

Steps

Pre-process

- Detect
- Input
- Sewing
- Octal
- Populate

[…]

- Scheduler
- Balance
- Migrate
- ReduceStatistics
- FieldMapping

Solve

- Solver

Convert files

- Merger
- Poster

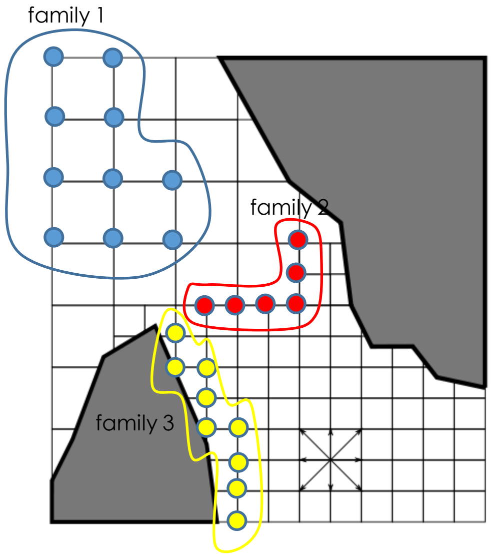
1/ Read the scheme file (text file)

2/ Evaluate all the node filter functions, for all the mesh nodes

3/ Gather mesh nodes into node families : nodes that have the same filter node properties and that are summitted to the same computation functions

4/ Build the dependency (dataflow) graph

5/ Calculate the task schedule : define the order of the computation and processor exchange tasks



The size of each family strongly depends on the mesh configuration **AND** on the granularity of the developer node filters

→ From few nodes to millions of nodes

PROLB STEPS

Pre-process

Steps

- Detect
- Input
- Sewing
- Octal
- Populate

[…]

- Scheduler
- Balance
- Migrate
- ReduceStatistics
- FieldMapping

Solve

- Solver

Convert files

- Merger
- Poster

The solver reads and executes the task schedule for the N requested time iterations

```
SCHEDULE

  ELEMENTS

   0 COMPUTE1 39 macro scalars (MIDDLE) COMP 5 (Initm) BACK 1 SEND 2
   1 COMPUTE1 41 macro scalars (MIDDLE) COMP 5 (Initm) BACK 1 SEND 2
   [...]
   303 COMPUTE  43 macro macroNeq (MIDDLE) COMP 79 (Macroscopic) BACK 0 SEND 2
   304 OUT      43 outmacro (MIDDLE) COMP 155 (Dimensionality) BACK 0
   305 SYNCHRONIZE
   306 COMPUTE  39 macro macroWall (MIDDLE) COMP 97 (MacroscopicNoFullyFluidStd) BACK 0 SEND 2
   307 COMPUTE  39 scalars (MIDDLE) COMP 69 (Reconstructionscalars) BACK 0 SEND 2
   [...]
   425 COMPUTE   9 grad (INTERSPACE_FACE) COMP 114 (GradientInterpFaceODD) BACK 0 SEND 2
   426 COMPUTE   9 macroTurb (INTERSPACE_FACE) COMP 121 (NuTurb) BACK 0 SEND 6
   427 COMPUTE   9 fcoll (INTERSPACE_FACE) COMP 141 (Collide) BACK 0 SEND 5
   428 SYNCHRONIZE
   429 NEXT_TIME
   430 COMPUTE   1 triangleInfos (TRANSITION_FINE) COMP 57 (InitNormalAndCenterOrDoNothing) BACK 0 SEND 0
   [...]
   597 COMPUTE   9 fcoll (INTERSPACE_FACE) COMP 141 (Collide) BACK 0 SEND 5

   598 REPEAT 222

  END ELEMENTS

END SCHEDULE
```

TASK SCHEDULE

- A human-readable schedule file can be created for checking/debbuging purpose :

```
SCHEDULE

  ELEMENTS

   0 COMPUTE1 39 macro scalars (MIDDLE) COMP 5 (Initm) BACK 1 SEND 2
   1 COMPUTE1 41 macro scalars (MIDDLE) COMP 5 (Initm) BACK 1 SEND 2
   [...]
   303 COMPUTE  43 macro macroNeq (MIDDLE) COMP 79 (Macroscopic) BACK 0 SEND 2
   304 OUT      43 outmacro (MIDDLE) COMP 155 (Dimensionality) BACK 0
   305 SYNCHRONIZE
   306 COMPUTE  39 macro macroWall (MIDDLE) COMP 97 (MacroscopicNoFullyFluidStd) BACK 0 SEND 2
   307 COMPUTE  39 scalars (MIDDLE) COMP 69 (Reconstructionscalars) BACK 0 SEND 2
   [...]
   425 COMPUTE   9 grad (INTERSPACE_FACE) COMP 114 (GradientInterpFaceODD) BACK 0 SEND 2
   426 COMPUTE   9 macroTurb (INTERSPACE_FACE) COMP 121 (NuTurb) BACK 0 SEND 6
   427 COMPUTE   9 fcoll (INTERSPACE_FACE) COMP 141 (Collide) BACK 0 SEND 5
   428 SYNCHRONIZE
   429 NEXT_TIME
   430 COMPUTE   1 triangleInfos (TRANSITION_FINE) COMP 57 (InitNormalAndCenterOrDoNothing) BACK 0 SEND 0
   [...]
   597 COMPUTE   9 fcoll (INTERSPACE_FACE) COMP 141 (Collide) BACK 0 SEND 5

   598 REPEAT 222

  END ELEMENTS

END SCHEDULE
```

- Each line = one task
- A COMPUTE line is applied on a given node family
- Some information are given on that family computation (which DataType with which callback function, …)

COMPUTATION FUNCTIONS

- A human-readable schedule file can be created for checking/debbuging purpose :

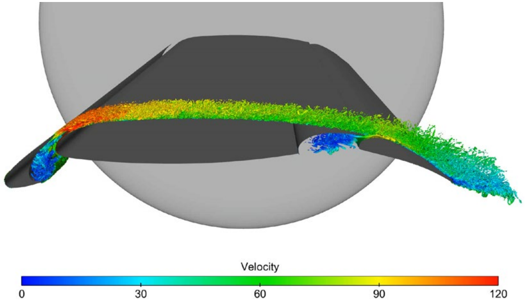
```
SCHEDULE

  ELEMENTS

   0 COMPUTE1 39 macro scalars (MIDDLE) COMP 5 (Initm) BACK 1 SEND 2
   1 COMPUTE1 41 macro scalars (MIDDLE) COMP 5 (Initm) BACK 1 SEND 2
   [...]
   303 COMPUTE  43 macro macroNeq (MIDDLE) COMP 79 (Macroscopic) BACK 0 SEND 2
   304 OUT      43 outmacro (MIDDLE) COMP 155 (Dimensionality) BACK 0
   305 SYNCHRONIZE
   306 COMPUTE  39 macro macroWall (MIDDLE) COMP 97 (MacroscopicNoFullyFluidStd) BACK 0 SEND 2
   307 COMPUTE  39 scalars (MIDDLE) COMP 69 (Reconstructionscalars) BACK 0 SEND 2
   [...]
   425 COMPUTE   9 grad (INTERSPACE_FACE) COMP 114 (GradientInterpFaceODD) BACK 0 SEND 2
   426 COMPUTE   9 macroTurb (INTERSPACE_FACE) COMP 121 (NuTurb) BACK 0 SEND 6
   427 COMPUTE   9 fcoll (INTERSPACE_FACE) COMP 141 (Collide) BACK 0 SEND 5
   428 SYNCHRONIZE
   429 NEXT_TIME
   430 COMPUTE   1 triangleInfos (TRANSITION_FINE) COMP 57 (InitNormalAndCenterOrDoNothing) BACK 0 SEND 0
   [...]
   597 COMPUTE   9 fcoll (INTERSPACE_FACE) COMP 141 (Collide) BACK 0 SEND 5

   598 REPEAT 222

  END ELEMENTS

END SCHEDULE
```

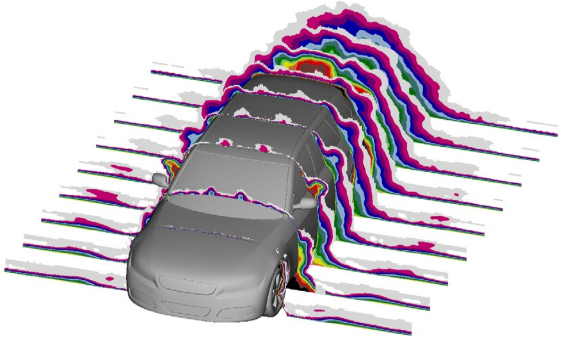
- The call-back functions are standard C++ functions :

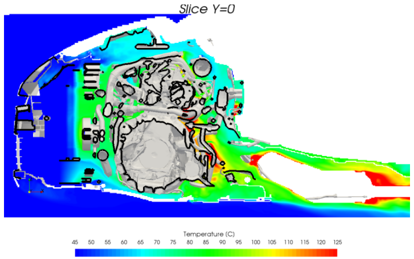
```cpp
void D3Q19DRT::Macroscopic(int inodeBegin, int inodeEnd) {

  // read access to the input data
  for (int k = 0; k < NBDIR; k++) {
    fcolC[k]      = solver->getLinkTraverser<mandatory>(fcoll, k, 1) ;
  }

  // write access to computed data
  Float* fdis[NBDIR];
  for (int index = 0; index < NBDIR; index++) {
    fdis[index] = solver->putDataArray(fdist, index);
  }


  // loop over the nodes of the family
  for (int inode = inodeBegin; inode < inodeEnd; inode++) {

     // loop over LBM index directions
    for (int index = 0 ; index < NBDIR ; index++) {

      int jnode = extremity[linkREL];
      int jsize = piece->getSize(jnode);

      if (isize == jsize) {
        fdis[index][inode] = fcolC[index][linkREL] ;
      } else {
        …
      }
    }
    …
  }
```

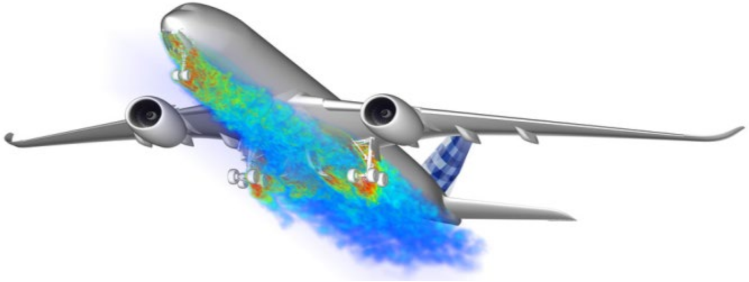
APPLICATIONS WITH THE COMMERCIAL VERSION

- Production versions are based on the dataflow programming framework
- Multi-physics schemes are already in production with ProLB (Renault, Airbus, Nissan,…)
    - Low Mach aerodynamics, aeroacoustics, aerothermal flows
    - Very competitive turn-around time



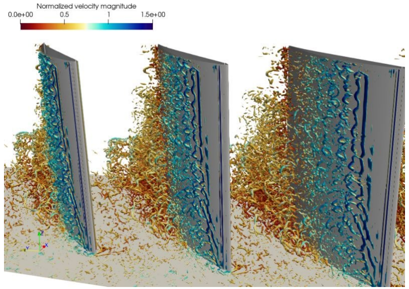




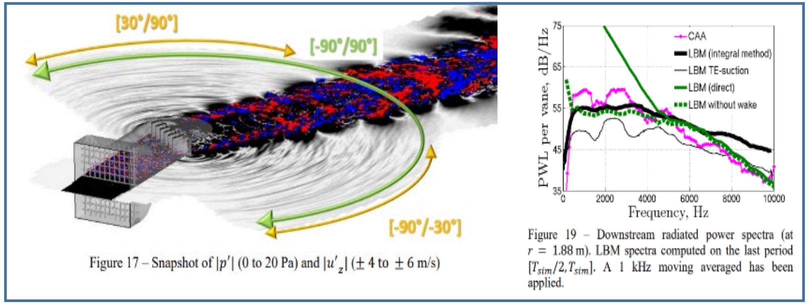


PROTOTYPE PHYSICS DEVELOPED BY OUR PARTNERS

- Partners have access to the source code of the DSL and the associated C++ computation functions, while the HPC kernel is provided as a binary executable which is dynamically linked to the physics module
- It allows real-world simulations, with the same complexity and size than the simulations performed with the commercial version
- More than 40 physics developers (Researchers, post-docs, PhD students,…)



Development of turbulence models
Unsteady Lattice Boltzmann Simulations of Corner Separation in a Compressor Cascade
J. Boudet, E. Lévêque, H. Touil, 2022



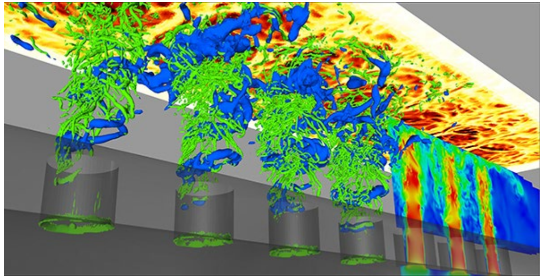
Noise generated by serrated vanes.
More than 2 billions mesh points, 8000 CPU cores, 2 days of calculation
Martin Buszyk *at al.*, 2022

ONERA
THE FRENCH AEROSPACE LAB

CS GROUP

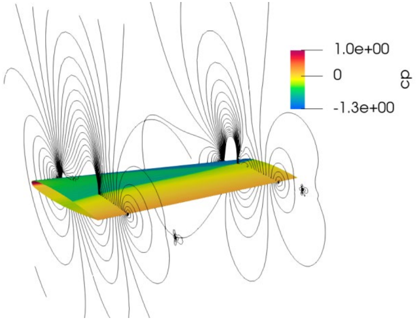
PROTOTYPE PHYSICS DEVELOPED BY OUR PARTNERS

- Partners have access to the source code of the DSL and the associated C++ computation functions, while the HPC kernel is provided as a binary executable which is dynamically linked to the physics module
- It allows real-world simulations, with the same complexity and size than the simulations performed with the commercial version
- More than 40 physics developers (Researchers, post-docs, PhD students,…)



Compressible flow / thermal management of a low pressure turbine
Minh NGUYEN, PhD student Safran @ Cerfacs

CERFACS SAFRAN



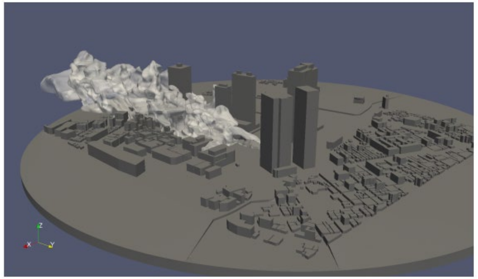
High Mach flow with shocks on the ONERA M6 wing
Thomas Coratger *at al.*, 2021

Aix*Marseille université

CS GROUP

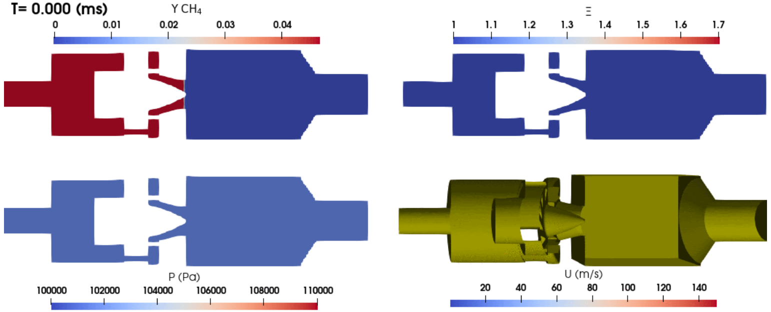
PROTOTYPE PHYSICS DEVELOPED BY OUR PARTNERS

- Partners have access to the source code of the DSL and the associated C++ computation functions, while the HPC kernel is provided as a binary executable which is dynamically linked to the physics module
- It allows real-world simulations, with the same complexity and size than the simulations performed with the commercial version
- More than 40 physics developers (Researchers, post-docs, PhD students,…)



Pollutant atmospheric dispersion in Tokyo
J. Jacob *at al.*, 2019

Aix*Marseille université



Combustion inside the Preccinsta burner
Pierre Boivin *at al., 2021*

Aix*Marseille université

PROS & CONS

PHYSICS DEVELOPER POINT OF VIEW

- Pros :
  - Very easy to develop complex scheme
  - No memory management (no allocate/deallocate), <u>no parallelism management : he does not see any MPI command</u>
  - High security in term of “bugs” of data dependency : unexpected data overwriting is not possible, not possible call/use a data before it has been updated …

- **No miracle**: the physics developer must create a scheme for which the task graph can be solved, i.e. without cross dependencies between data
  - Readability of the scheme is however largely improved. Trial&error is much easier.

- Cons :
  - Some simple tricks used in CFD cannot be coded directly by the physics developer : for example, no simple access to MPI reduce command → each MPI feature must be ported/developed in the HPC kernel by the dedicated CS GROUP team
  - The challenges of the data management in parallel context have not disappeared, they have just been hidden into the DSL/dataflow programming module : some bugs may remain when integrating new functionalities → maturity and robustness of the DSL is of primary importance for wide adoption.

PROS & CONS — GLOBAL SOFTWARE PERFORMANCE POINT OF VIEW

- Pros :
  - Robustness/generality of the software features regarding the variety of numerical schemes
  - No lost of HPC performance :
    - The overhead of the calculation of the task schedule is about dozens of minutes (small compared to the total calculation time)
    - The overhead of “reading” the schedule lines is negligible
  - Strong separation between physics and HPC kernel. For example, the DataType object used in the DSL does not depend on actual data structure : Structure of Arrays, Arrays of Structures, direct/indirect addressing :
    - HPC kernel developers can test/change the data layout without impacting the physics scheme
  - The explicit knowledge of the task schedule for the computations and data exchanges allows some optimizations :
    - Gather computations, merge and reorganize (delay) the MPI exchanges,…
  - The pre-calculated task schedule with full knowledge of data dependencies facilitates new HPC architecture porting : GPU porting has been launch (EuroHPC project “SCALABLE”)

PROS & CONS

GLOBAL SOFTWARE PERFORMANCE POINT OF VIEW

- Cons :

  - Performance optimization with high degree of interlacing between the data-structure and the physics algorithm are more difficult to implement. *As an example : LBM implementations with very clever pointer/data swapping to reduce read and write access.*
    - Some modifications of the dataflow kernel have been recently done to introduce LBM-dedicated optimizations
    - → trade-off to be found between high specialization and versatile development framework

  - Data and parallelism management is **too simple** for the physics developers !
    - Adding a datatype is straightforward : risk of unnecessary increase of the memory footprint
    - Request to access of any current-time data neighbors during the current time-step is straightforward
      - It allows to write clean/accurate algorithms,… but it can create excessive intermediate synchronization points
      - Synchronization points at the end of each time iteration are natural, but each access to data neighbors at current time-step in the middle of the current time step can generate a sub-synchronization point → more data exchange/waiting time.

```
COMPUTE fdist ON_nearBoundary  MIDDLE BY PropagateMiddle ALWAYS  FROM macro[1..6,0]
```

Six neighbor variables are requested, at current time

    - → need for an optimization phase to reduce memory footprint and reduce the data exchanges between processors

CONCLUDING REMARKS

- Dataflow programming with a simple DSL framework has proved to be a good solution for an efficient development of complex physical models into the lattice Boltzmann solver
- It allows a large community of researchers/developers to implement physics models on a HPC framework with full industrial features
- Even with naïve usage of the DSL/Dataflow programming, the HPC performance is relatively good
    - For production phase, some re-working and optimization phases are necessary
- Dataflow pre-calculation of the dependency graph with the data movements is a clear advantage for porting the physics scheme to GPU processors.
    - Work in Progress

CS GROUP

www.csgroup.eu