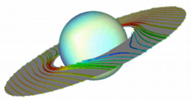
Code_Saturne: an HPC point of view

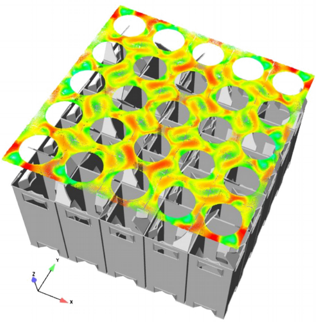
Yvan Fournier, EDF

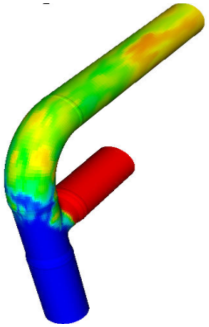
June 18, 2015

Code development at EDF R&D (1/2)

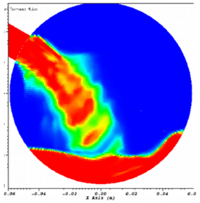
- *Code_Saturne*
  - general usage single phase CFD, plus specific physics
  - property of EDF, open source (GPL)
  - http://www.code-saturne.org

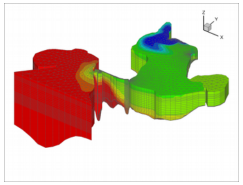




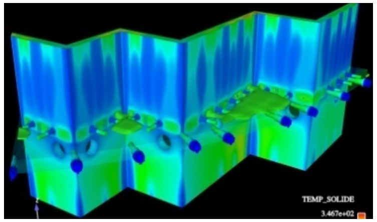


- NEPTUNE_CFD
  - multiphase CFD, esp. water/steam
  - property of EDF/CEA/AREVA/IRSN

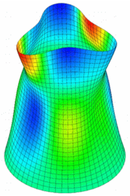




- SYRTHES
  - thermal diffusion in solid and radiative transfer
  - property of EDF, open source (GPL)
  - http://rd.edf.com/syrthes

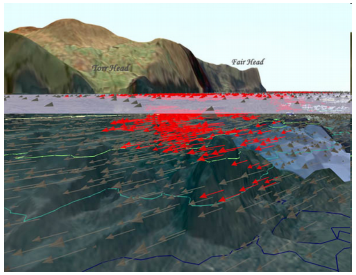


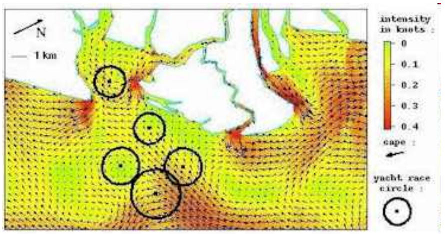
- *Code_Aster*
  - general usage structure mechanics
  - property of EDF, open source (GPL)
  - http://www.code-aster.org



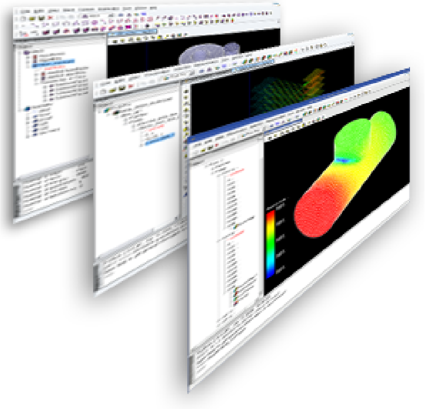
Code development at EDF R&D (2/2)

- TELEMAC system
  - free surface flows
  - Many partners, mostly open source (GPL, LGPL)
  - http://www.opentelemac.org

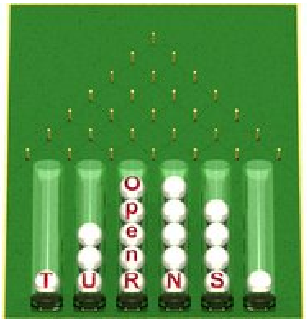




- SALOME platform
  - integration platform (CAD, meshing, post-processing, code coupling)
  - property of EDF/CEA/OpenCascade, open source (LGPL)
  - http://www.salome-platform.org



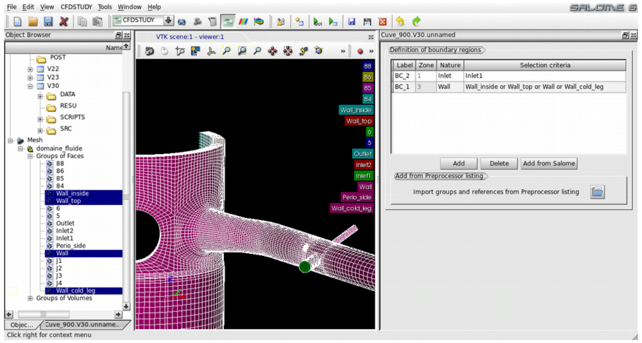
- Open TURNS
  - tool for uncertainty treatment and reliability analysis
  - property of EDF/CEA/Phimeca, open source (LGPL)
  - http://trac.openturns.org



- and many others
  - Neutronics, electromagnetism
  - component codes, system codes
  - ...

Code_Saturne
EDF’s general purpose CFD tool

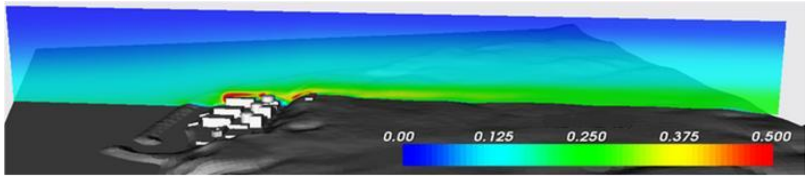
- Technology
  - Co-located unstructured finite volume, predictor-corrector
  - 420 000 lines of code, 35% Fortran, 50% C, 13% Python

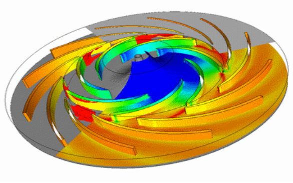


Label	Zone	Nature	Selection criteria
BC_2	1	Inlet	Inlet1
BC_1	3	Wall	Wall_inside or Wall_top or Wall or Wall_cold_leg

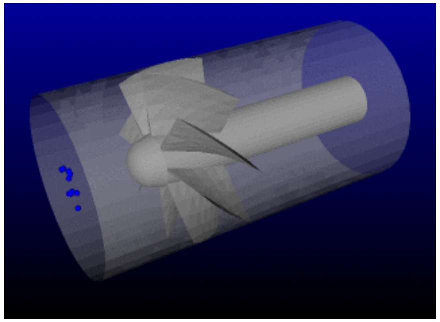
Import groups and references from Preprocessor listing

- Physical modeling
  - Laminar and turbulent flows: k-ε, k-ω SST, v2f, RSM, LES
  - Radiative transfer (DOM, P-1)
  - Coal, heavy-fuel and gas combustion
  - Electric arcs and Joule effect
  - Lagrangian module for particles tracking
  - Atmospheric modeling
  - ALE method for deformable meshes
  - Rotor / stator interaction for pumps modeling
  - Conjugate heat transfer (SYRTHES & 1D)
  - Common structure with NEPTUNE_CFD for Eulerian multiphase flows



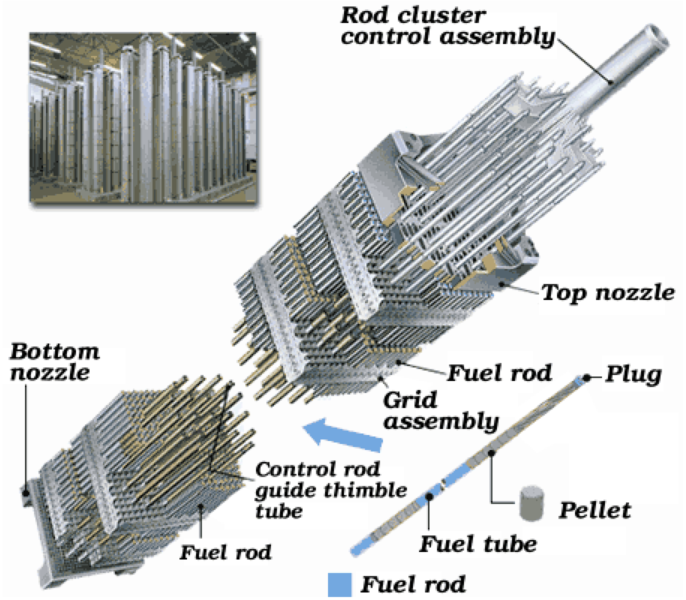


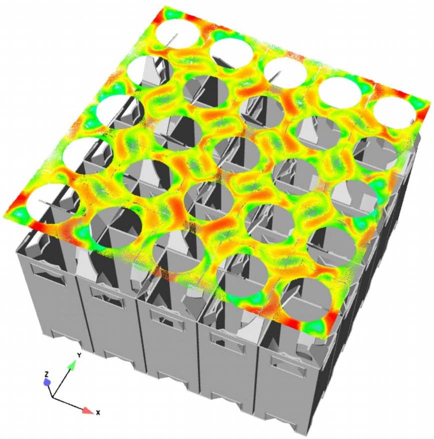
- Ecosystem
  - Portable (Linux, MacOS, Unix X; beta-version of Windows port)
  - GUI with possible integration within the SALOME platform
  - Parallel using MPI + OpenMP - periodic boundaries with arbitrary interfaces
  - Accepts large range of unstructured meshes with arbitrary interfaces
    - generated with SALOME, STAR-CCM+, ANSYS Meshing, GMSH, ...
  - Code coupling (*Code_Saturne*/*Code_Saturne*, *Code_Saturne*/Code_Aster, ...)

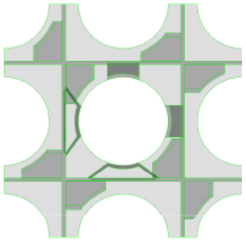


EDF Applications: Fuel Assemblies

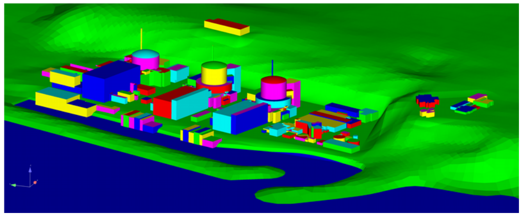
- 157 - 241 fuel assemblies in a PWR
- Each fuel assembly has 17*17 fuel rods or guide tubes and 8 to 10 grids
  - complex, non symmetrical geometry
  - Different vendors and models
- Many constraints / stakes
  - If head loss/lift too high, stronger springs needed to keep FA down, leading to possible bowing and deformation
  - Good heat exchange: need mixing grids to generate turbulence, avoid DNB (de-nucleate boiling)
  - Low vibration: loss of cladding integrity may result from vibration_induced fretting



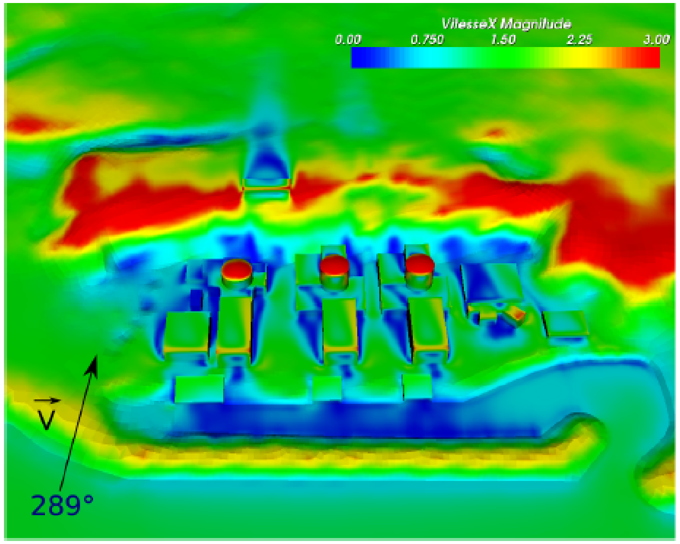


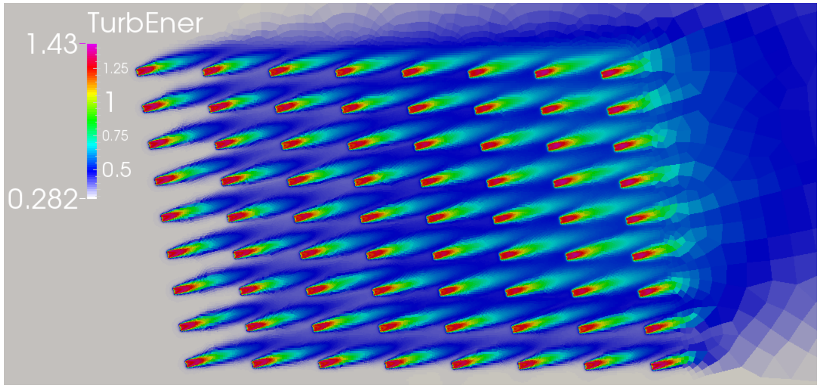


EDF Applications: meteo



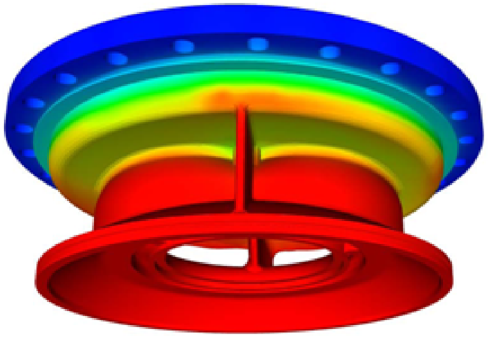
- Simulations for the atmospheric environment at the local and micro-scale (a few tens of meters to ~20km)
  - account for terrain, buildings
    - environmental impact of industrial sources, of road traffic, …
- all pollutant types: radionuclides, chemical or biological pollutants, heavy gases
- environmental impact on energy production with renewables:
  - estimation of wind production, wake effects
- model energy exchanges and pollution in urban areas
- environment impact on plants
  - wind and turbulence on buildings, HT lines
- impact of external aggressions
  - rupture of gas tank near a power plant



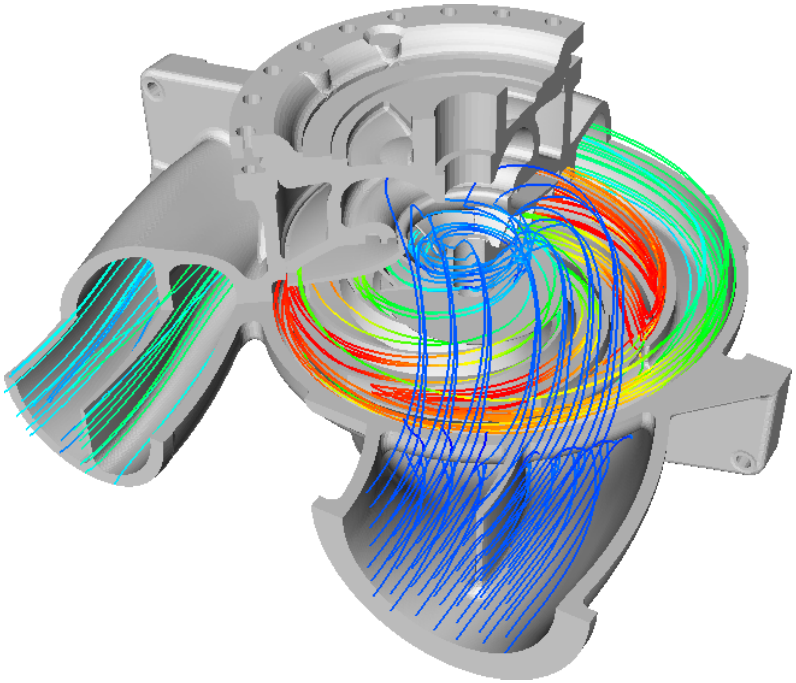


EDF Applications: CSS PUMP

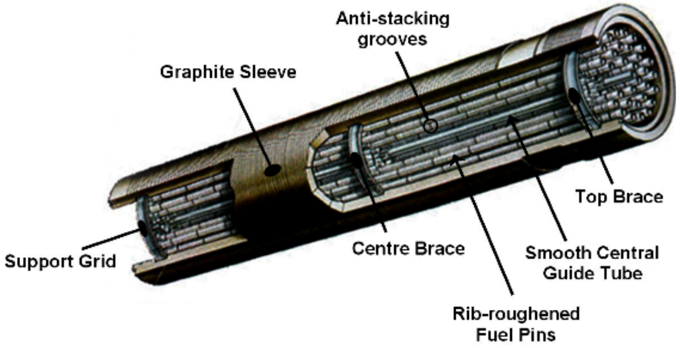
- Thermal shock
  - Large temperature gradient in the upper part of the lid
- Particle nocivity
  - Prediction of the particle distribution at the inlet of the lubrication system (in progress…)
- Cavitation model



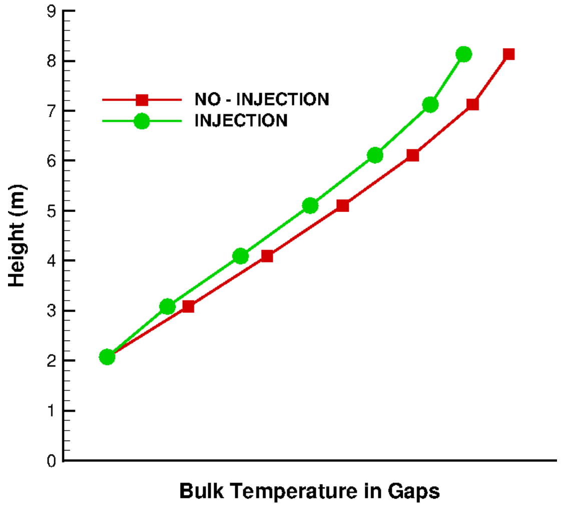
Temperature field in the lid

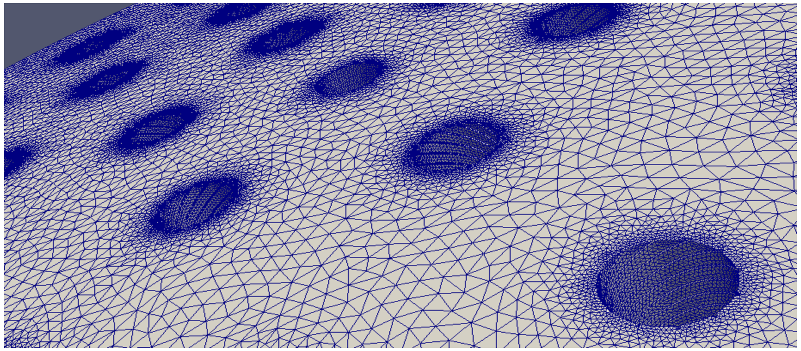


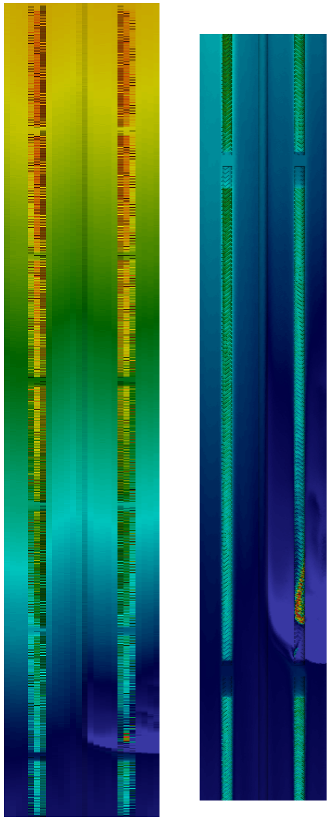
Example large scale application: Advanced Gas Reactor



- Advanced Gas Reactor (EDF Energy) simulation
  - collaboration between EDF Energy R&D UK Centre and STFC Daresbury
  - fuel assembly: 36 pins, 1 control rod
    - 1 m height, 8 assemblies stacked
    - gap between each assembly
  - 200 elemental meshes, replicated and joined using parallel mesh joining
    - detailed representation (see riblets in zoom below)
- 1.06 billion cells
  - runs on STFC's Blue Joule (Blue Gene/Q)
  - 512 nodes, 32 ranks/node, 16384 cores total

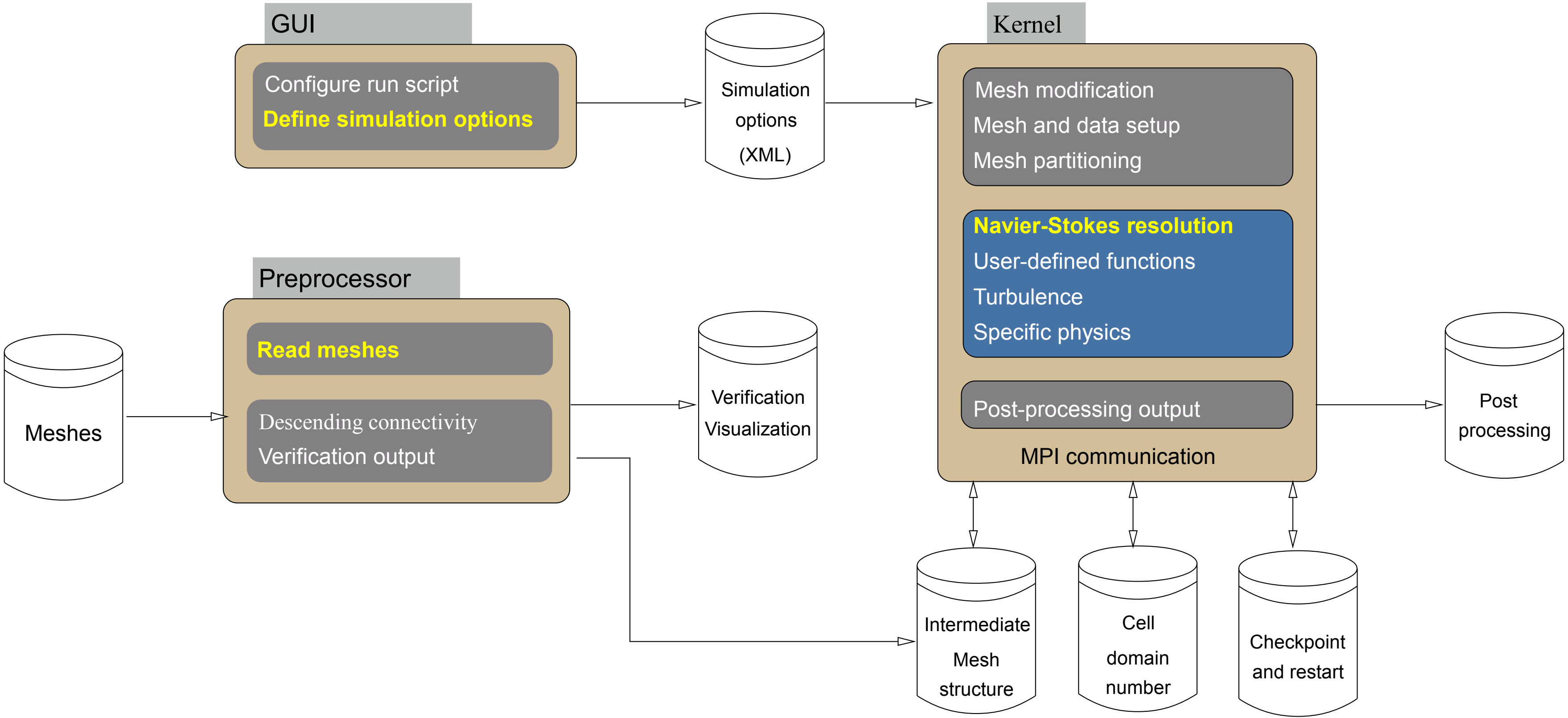






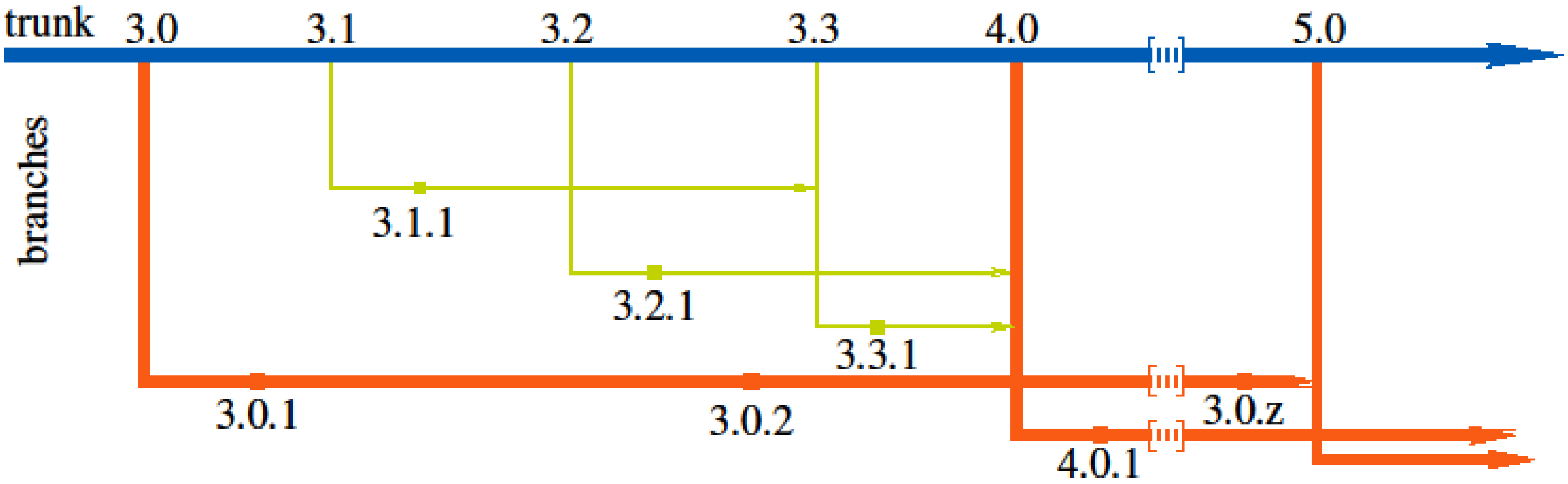
Code_Saturne toolchain

- Reduced number of tools, each with rich functionality
  - Natural separation between interactive and long-running parts
  - Try to minimize IO and partitioning dependency
    - most preprocessing and partitioning is done within the main executable, and partition-independent IO is preferred

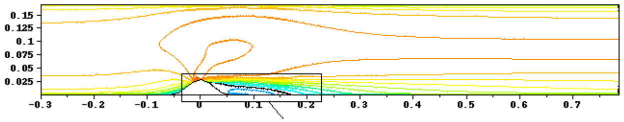


Code_Saturne versioning

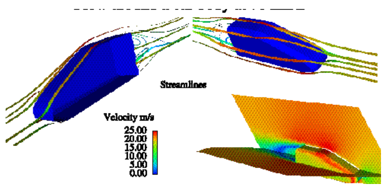
- From version 2.0 on, different kinds of versions “*x.y.z*” are released
  - Production version every two years (*x* increment)
    - With the release of a Verification & Validation summary report
  - An intermediate version every six months (*y* increment)
    - With non-regression tests to ensure the code quality
  - Corrective versions when needed (*z* increment)
    - To make sure the users are provided with bug fixes and ports

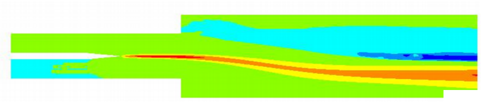


Quality Assurance and VVUQ

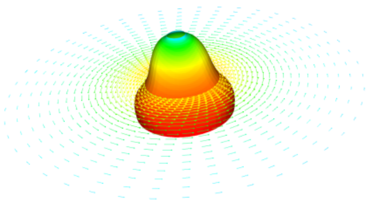


- *Code_Saturne* widely used at EDF and outside
  - 150 EDF users (R&D and engineering)
  - 400 ? “users” contacts outside EDF
  - Used for all general CFD calculations at EDF
  - Used in calculations for Nuclear Authorities
    - Code developed under Quality Assurance

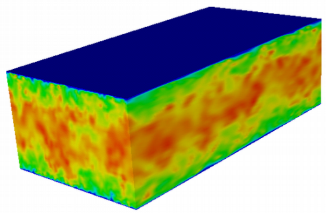


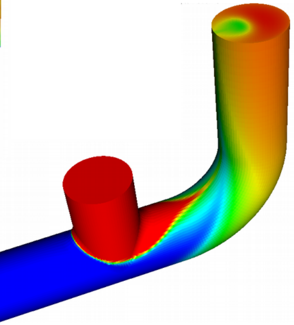


- Validation and verification of *Code_Saturne*
  - Before each release is declared “fit for industrial use”
  - Around 50 test cases, covering all capabilities of the code
    - 1 to 15 calculations per test case, 650 total
  - Academic test cases to industrial configurations
  - Major part of the project resources



- Qualification of *Code_Saturne*
  - Further qualification for specific industrial domains single phase thermal-hydraulics
  - Automatic validation suite run tool





Required Environment: **Linux**, **Unix**, **OS-X**, Window (from laptop to supercomputer)

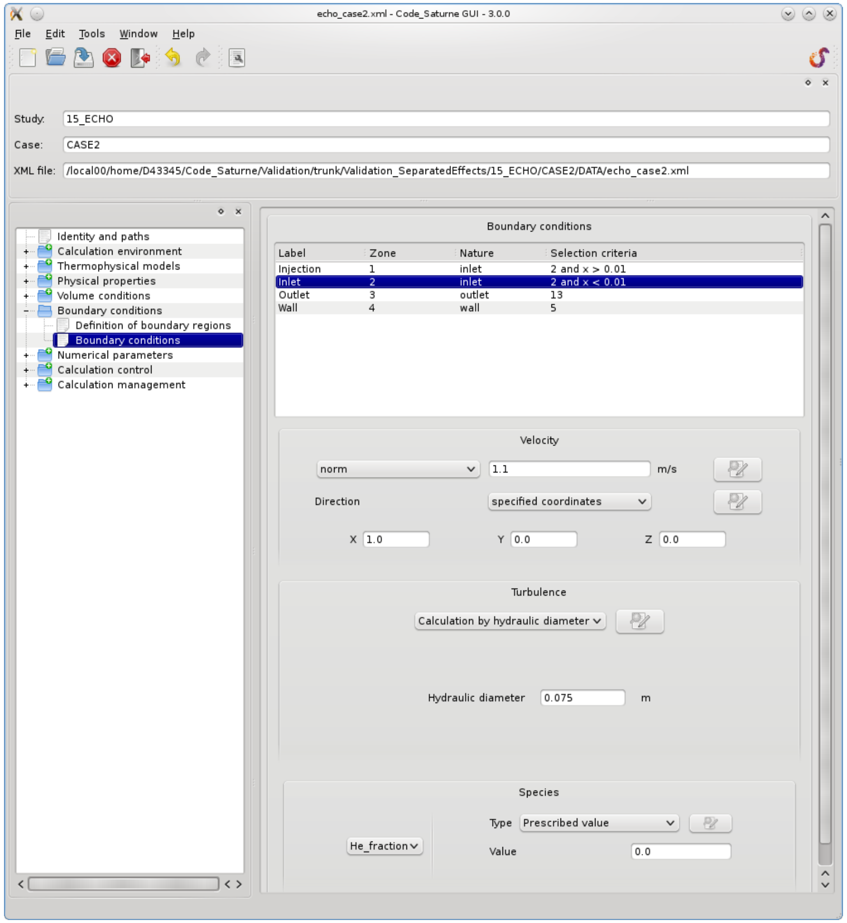
- Pre-requisite: compilers
  - C (gcc, xlc, icc, …)
  - Fortran (gfortran, xlf, ifort, ...)

- Pre-requisites for parallel computing:
  - MPI library : Open MPI, MPICH, ...
  - PT-SCOTCH or ParMETIS partitioner (optional)

- Pre-requisites for GUI:
  - Python, Qt4, SIP, libxml2

- Optional data exchange libraries:
  - MED, HDF5
  - CGNS
  - libCCMIO

- Optional integration under SALOME
  - GUI extensions
  - mouse selection of boundary zones
  - advanced user files management
  - from CAD to post-processing in one tool



Study: 15_ECHO

Case: CASE2

XML file: /local00/home/D43345/Code_Saturne/Validation/trunk/Validation_SeparatedEffects/15_ECHO/CASE2/DATA/echo_case2.xml

Boundary conditions

Label	Zone	Nature	Selection criteria
Injection	1	inlet	2 and x > 0.01
Inlet	2	inlet	2 and x < 0.01
Outlet	3	outlet	13
Wall	4	wall	5

Velocity

norm 1.1 m/s

Direction specified coordinates

X 1.0 Y 0.0 Z 0.0

Turbulence

Calculation by hydraulic diameter

Hydraulic diameter 0.075 m

Species

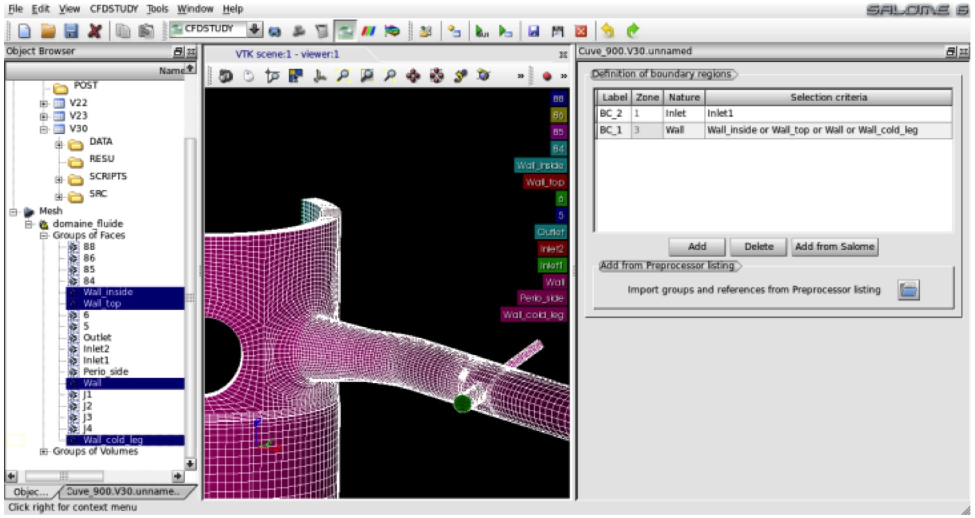
He_fraction

Type Prescribed value

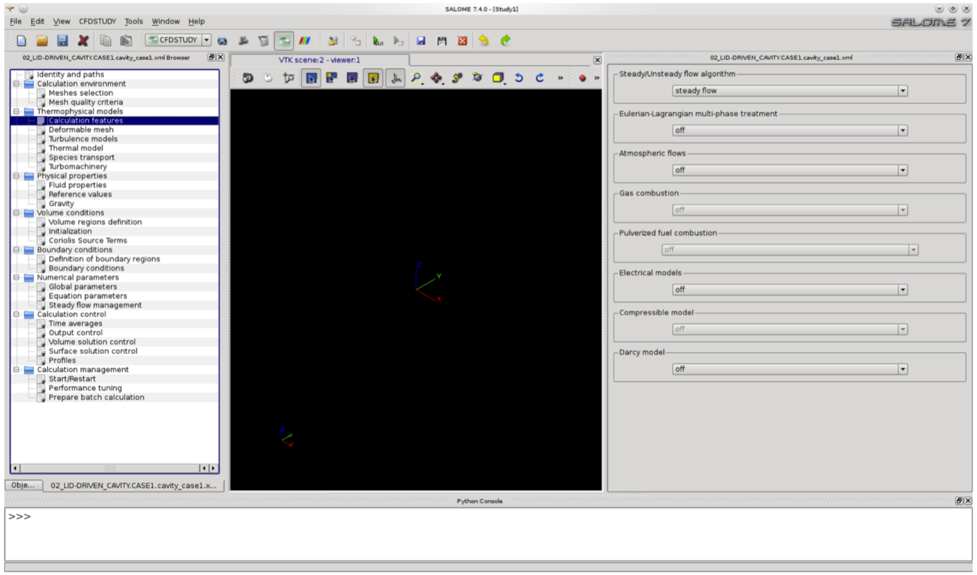
Value 0.0

Increasing use of the SALOME platform

- Provides a complete (mesh to visualization) environment
  - www.salome-platform.org
- Some parts of SALOME are better at HPC than others
  - Integration does not get in the way of HPC aspects



Label	Zone	Nature	Selection criteria
BC_2	1	Inlet	Inlet1
BC_1	3	Wall	Wall_inside or Wall_top or Wall or Wall_cold_leg



Steady/Unsteady flow algorithm: steady flow

Eulerian-Lagrangian multi-phase treatment: off

Atmospheric flows: off

Gas combustion: off

Pulverized fuel combustion: off

Electrical models: off

Compressible model: off

Darcy model: off

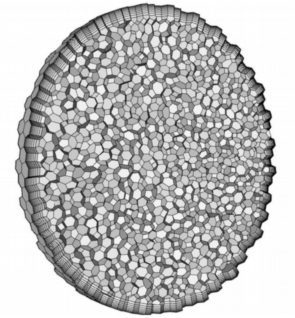
```
>>>
```

Installation on HPC resources

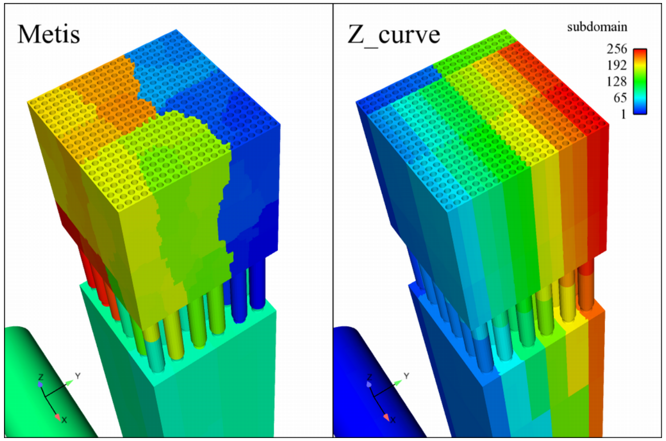
- The code can be installed quite easily on most clusters, as most external libraries are optional
  - GNU autotools (autoconf/automake/libtool) based install
    - out of source builds recommended
    - about half of the installation manual is dedicated towards examples of builds on clusters
  - Use whatever MPI library is recommended
  - An optional post-install configuration file can help with fine-tuning
    - paths to coupled codes
    - specific MPI environment commands
    - path of optional environment file to source before execution
  - The build system recognizes environment modules, and saves the configuration so as to use the same one upon executions
    - allows side-by-side builds with different tools
    - if this fails or when building an environment module for the code itself, configure using “”--with-modules=no” and handle modules by sourcing a file, or in user configuration
  - The GUI is not absolutely required on a cluster
    - an XML file can be built on a workstation, run on a cluster
    - completely user C and Fortran source file-based setups are possible

Supported Meshes

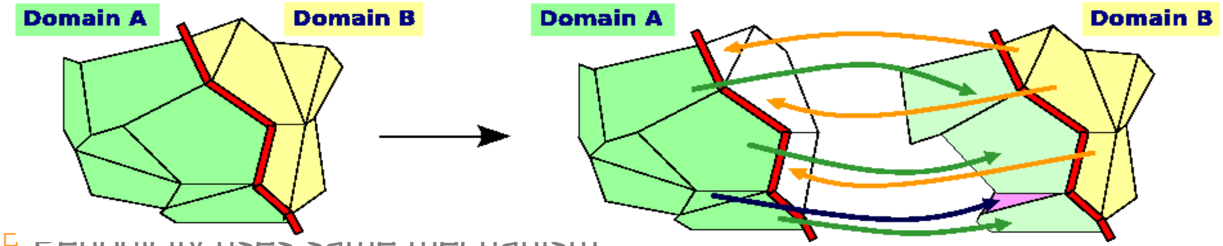
- Mesh generators
  - SALOME SMESH (http://salome-platform.org)
  - GAMBIT (Fluent), ICEM-CFD, ANSYS meshing
  - Star-CCM+
  - Simail: easy-to-use, with command file, but no CAD
  - I-deas NX
  - Gmsh
- Formats
  - MED, CGNS, Star-CCM+, Simail, I-deas universal,
  - GAMBIT neutral, EnSight Gold
- Cells: arbitrary arrangement of polyhedra
  - For example: tetrahedra, hexahedra, prisms, pyramids, general n-faced polyhedra,…



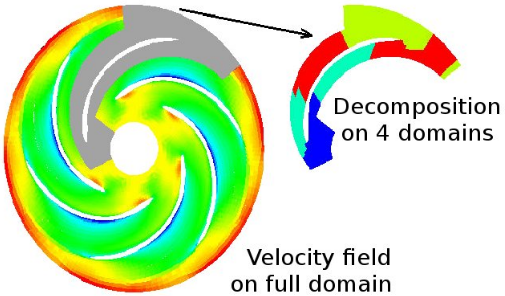
Parallelism and periodicity (1)



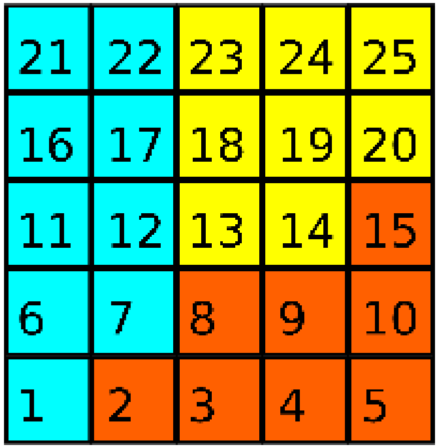
- Classical domain partitioning using MPI
  - Partitioning using METIS, SCOTCH or internal Morton or Hilbert space-filling curve
  - Classical « ghost cell » method
    - Most operations require only ghost cells sharing faces
    - Extended neighborhoods for gradients also require ghost cells sharing vertices



  - Periodicity uses same mechanism
    - True geometric periodicity (not a BC)
    - Vector and tensor rotation may be required (semi-explicit tensor component coupling in rotation)
  - Input output is partition independent
- Recently added OpenMP for compute-intensive sections

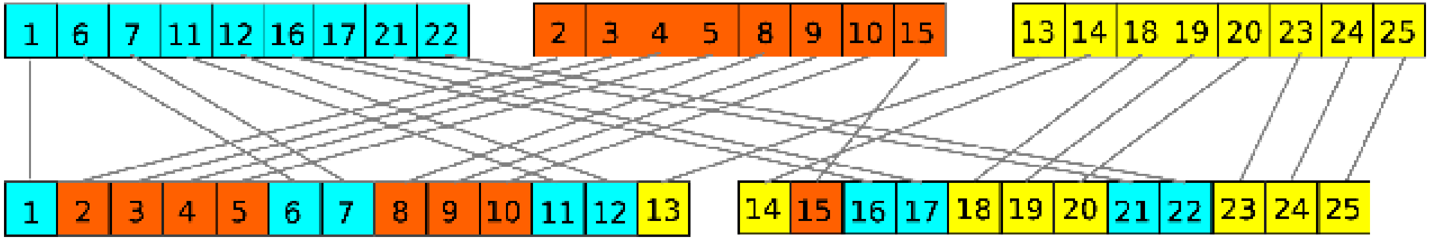


Parallelism and periodicity (2)



21	22	23	24	25
16	17	18	19	20
11	12	13	14	15
6	7	8	9	10
1	2	3	4	5

- Use of global numbering
  - We associate a global number to each mesh entity
    - A specific C type (cs_gnum_t) is used for this. An unsigned long integer (64-bit) is necessary for larger meshes
    - Currently equal to the initial (pre-partitioning) number
- Allows for partition-independent single-image files
  - Essential for restart files, also used for postprocessing output
  - Shared file MPI-IO possible does not require indexed datatypes
- Allows automatic identification of “Interfaces”
  - Matching between vertices on parallel boundaries
  - Allow summing contributions from multiple processes in a robust manner and in linear time
- Redistribution on n blocks
  - n blocks ≤ n cores
  - Minimum block size and ranks step may be adjusted, for performance, or to force 1 block (for I/O with non-parallel libraries)

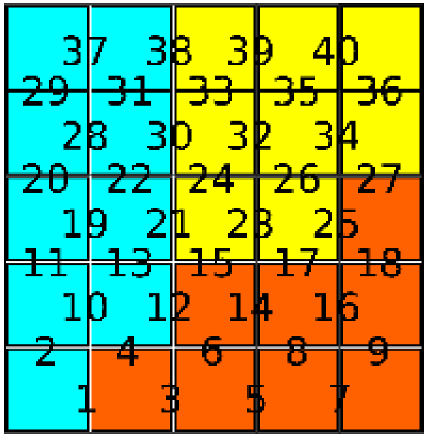


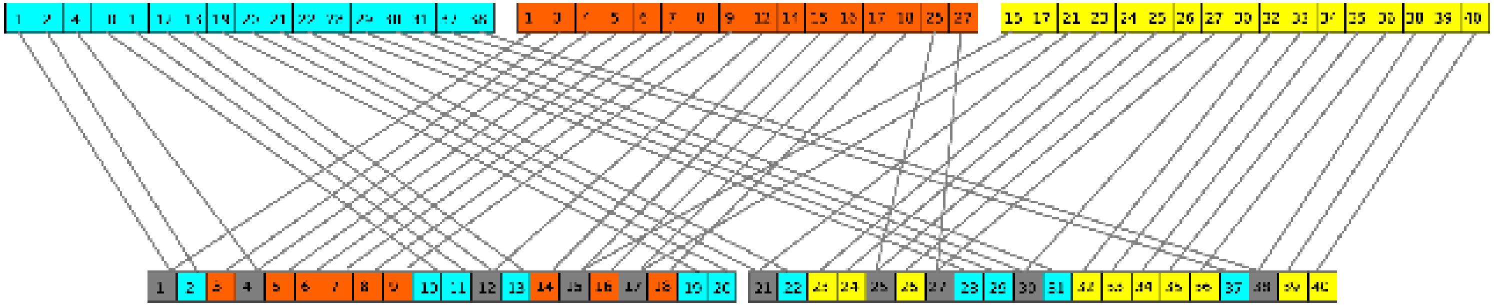
Parallelism and periodicity (3)

- Conversely, simply using global numbers allows reconstructing neighbor partition entity equivalents mapping
  - Used for parallel ghost cell construction from initially partitioned mesh with no ghost data

- Arbitrary distribution, inefficient for halo exchange, but allow for simpler data structure related algorithms with deterministic performance bounds
  - Owning processor determined simply by global number, messages are aggregated

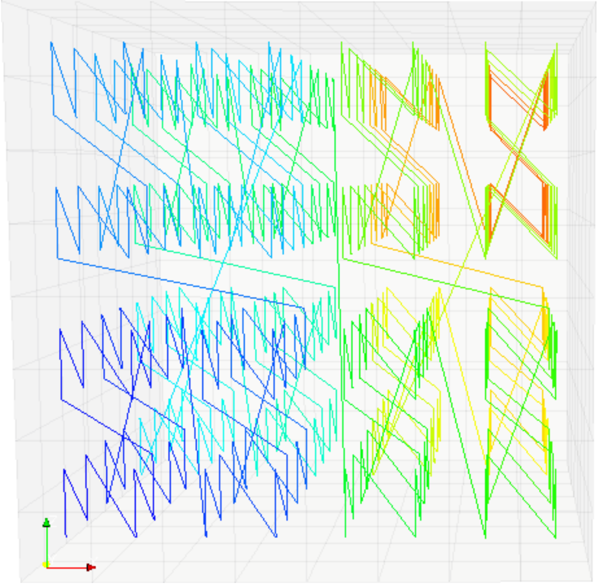
- Switch from one representation to the other currently uses `MPI_Alltoall` and `MPI_Alltoallv`, but we may switch to a more “sparse” algorithm
  - Not an issue under 16000 cores, not critical at 64000



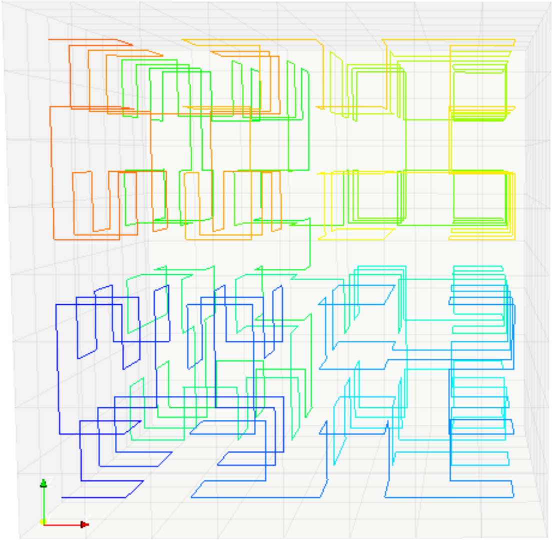


Mesh partitioning (1)

- Multiple partitioning options are possible
  - Serial Scotch or METIS
  - Parallel PT-Scotch or ParMETIS
    - Possibly run on a subset of the active ranks (allows for quality/memory optimization checks)
  - Morton or Hilbert space-filling curve (in bounding box or bounding cuve)
    - Built-in, requires no external library
    - Safe to have this as a back-up (and reference)
    - Advantage: deterministic



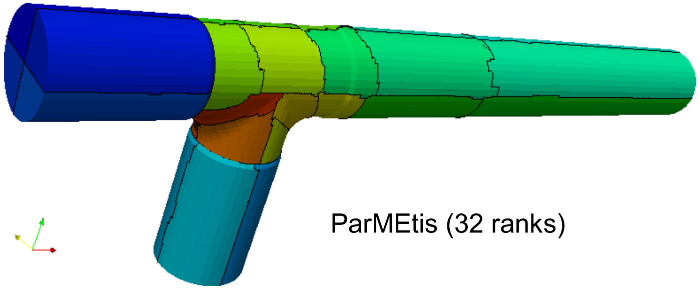
Morton curve



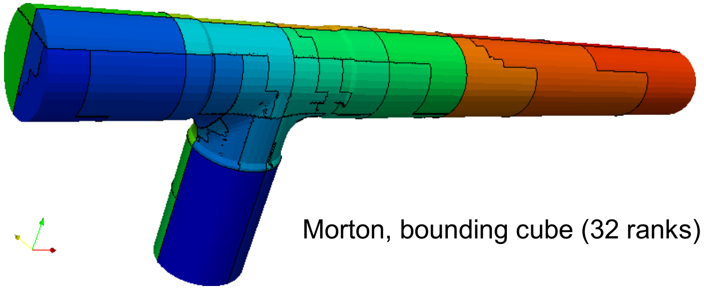
Hilbert curve

Mesh partitioning (2)

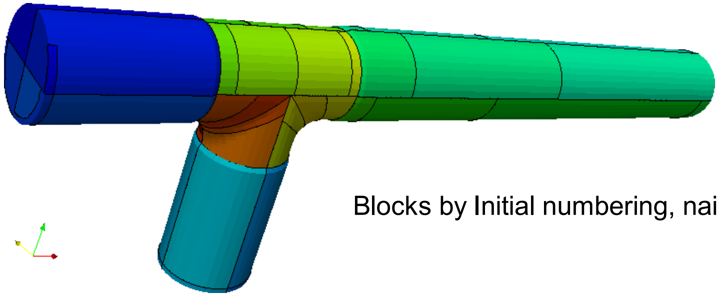
- Depending on mesh, results may vary
  - In some rare cases, even naive renumbering may be OK



ParMEtis (32 ranks)



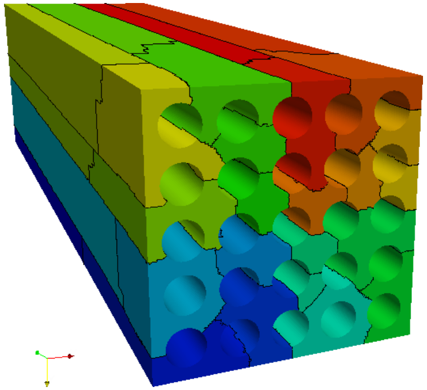
Morton, bounding cube (32 ranks)



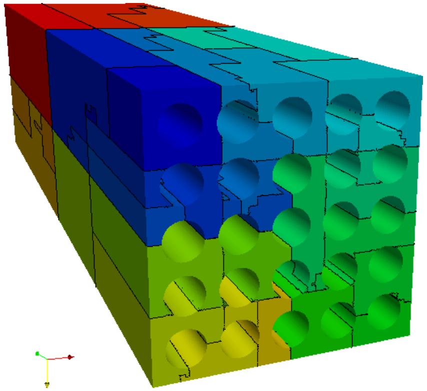
Blocks by Initial numbering, naive (32 ranks)

Mesh partitioning (3)

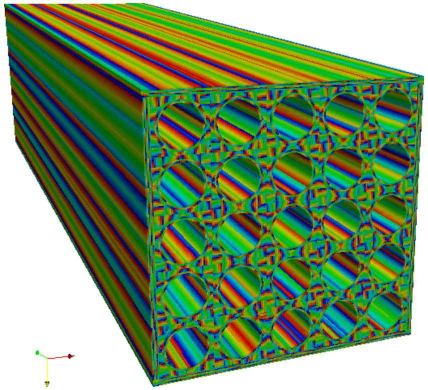
- In most cases, both are acceptable, while naïve partitioning is not
  - Always safe, 20 to 50% performance impact nonetheless



PT-Scotch (32 ranks)



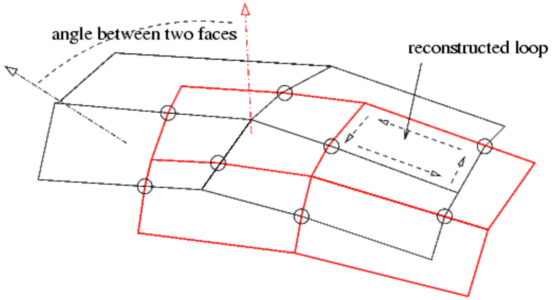
Hilbert, bounding box (32 ranks)



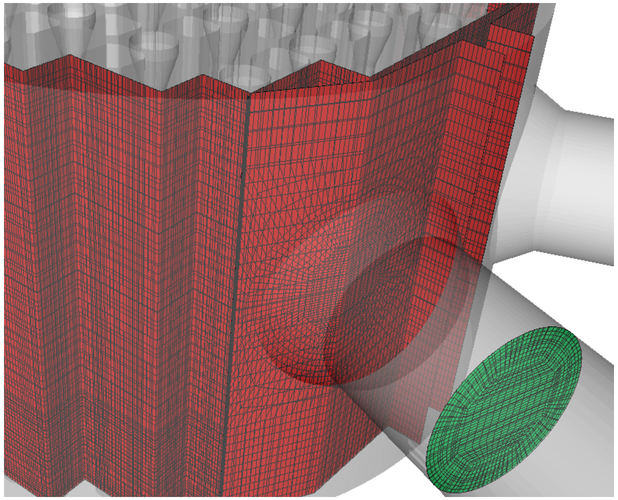
Blocks by Initial numbering, naive (32 ranks)

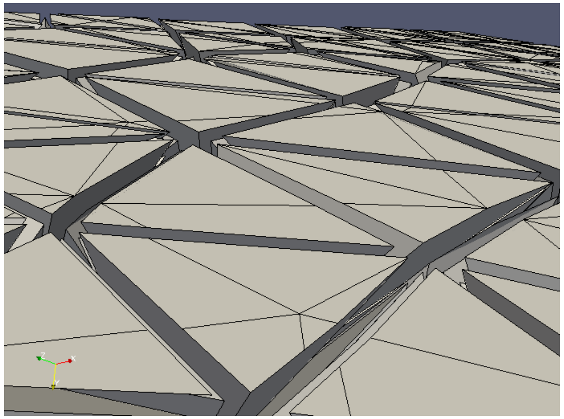
Mesh Joining (1)

- Arbitrary interfaces: « any type of mesh / format » + « any type of mesh / format »
  - Meshes may be contained in one single file or in several separate files
  - Expertise may be required if arbitrary interfaces are used:
    - in critical regions
    - with LES
    - with very different mesh refinements
    - on curved CAD interfaces

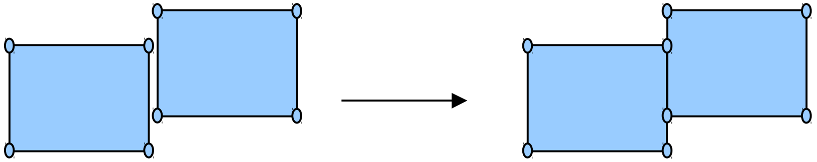


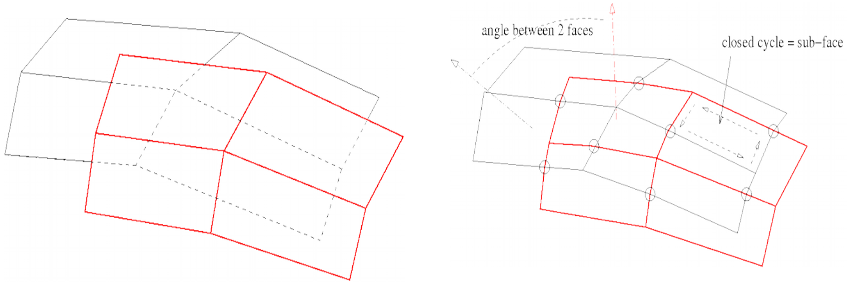
  - Done in parallel
    - allows assembling very large meshes built with serial meshing tools





Mesh joining (2)



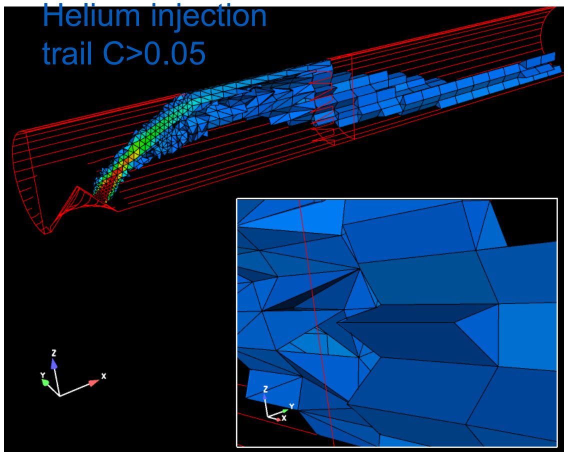


- Build a global face visibility map
  - □ Build a distributed octree-like structure of face bounding boxes
    - Built bottom-up
    - In parallel, a coarser partial octree is built first, so as to determine load balancing
  - □ Faces whose bounding boxes overlap (within a tolerance) may intersect
- Redistribute faces based on their global numbers
  - □ Copies of faces visible to a given face are also sent to its owning rank
- Determine intersections of face edges
  - □ Subdivide edges along those intersections, adding new vertices if necessary
- Merge vertices
  - □ Pre-merge vertices within a very small distance, then build “chains” of vertices within merging distance, reduce local merging distance if this leads to excessive merging (subdiving chains), then merge all vertices in a same chain
- Reconstruct sub-faces
  - □ Close shortest loops of edges on approximate face surfaces
  - □ Merge identical sub-faces: when 2 sub-faces with one adjacent each becomes 1 face with 2 adjacent cells, it becomes an interior face.

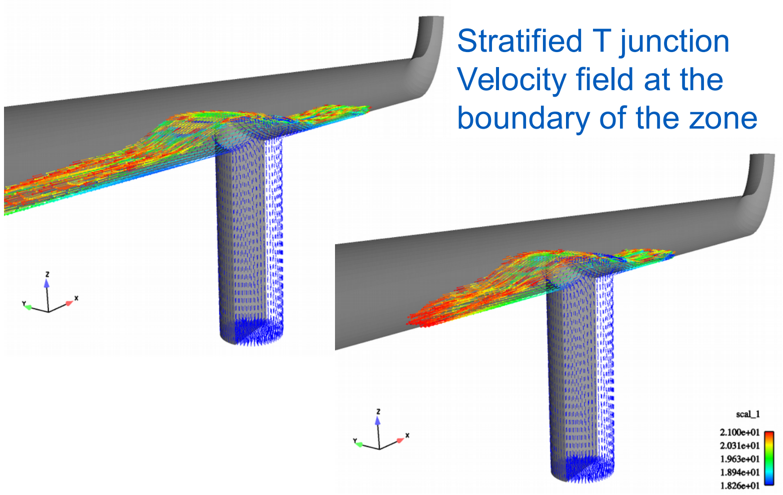
Postprocessing output

- Users may define an arbitrary number of post-processing:
  - Writers
    - Choice of an output format (EnSight, MED, CGNS, CCMIO, ParaView Catalyst), format options, and output frequency
  - Meshes
    - Subsets of the computational mesh
    - Each associated to a list of writers

- This allows fine control of user output
  - Frequently output fields may be associated to partial data, on a possibly time-dependent mesh subset
    - Useful for example to output partial data at higher frequency so as to generate movies, with “reasonable” output volumes



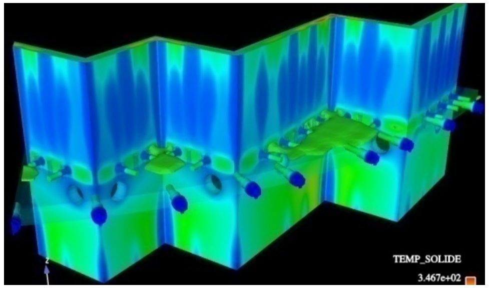
Helium injection trail C>0.05



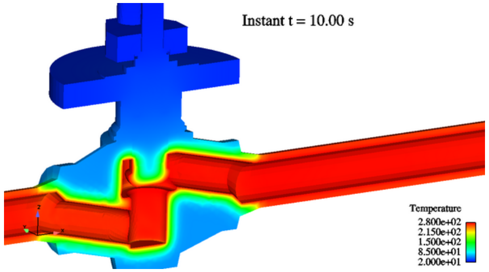
Stratified T junction Velocity field at the boundary of the zone

Parallel Code coupling (1)

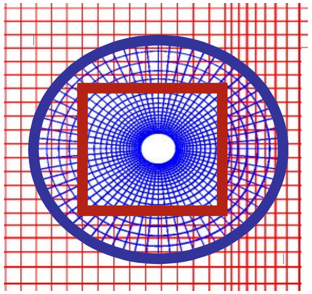
- Parallel n to p coupling using “**Parallel Location and Exchange**” sub-library
  - Uses MPI
  - Fully distributed (no master node requiring global data)
  - Successor to /refactoring of FVM (also used in Cwipi)
    - Core communication in PLE, rest moved to code
  - Also now used/tested by ALYA

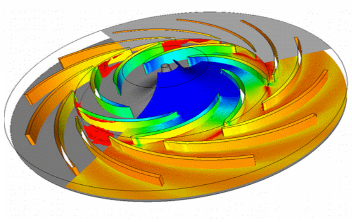


- SYRTHES (conjugate heat transfer)
  - Coupling with (parallel) SYRTHES 4 rshows good scaling using 128+ processors
  - Scaling of initial location not so good with current algorithm on 100's of ranks on BG/Q
    - need to upgrade with neighbor search from mesh joining



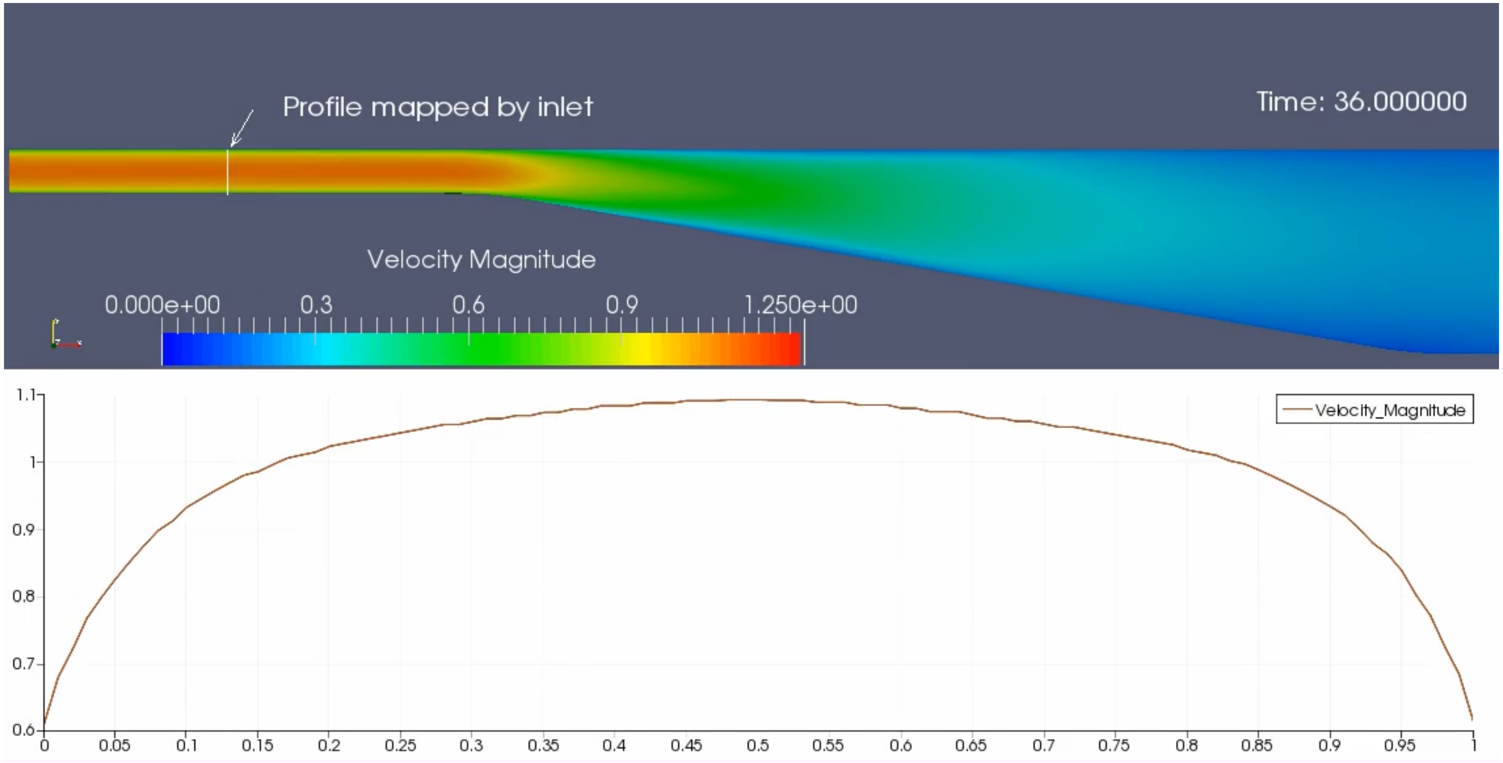
- *Code_Saturne*
  - RANS/LES
    - Different turbulence models and time steps in fixed overlapping domains
  - Turbomachinery (non-joining variant)
    - Same turbulence model time step, moving subdomain





Parallel Code coupling (2)

- Coupling with self also allows “mapped inlet boundary conditions”
  - point values at inlet mapped to cells inside domain
    - allows good profiles even with short inlets
    - several rescaling options available
  - works in parallel
    - partition-independent

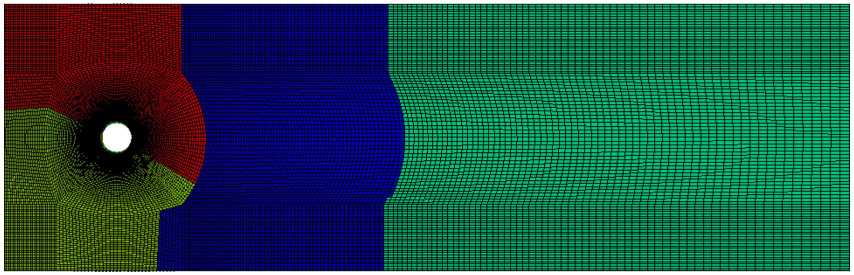


Other features of note to HPC aspects

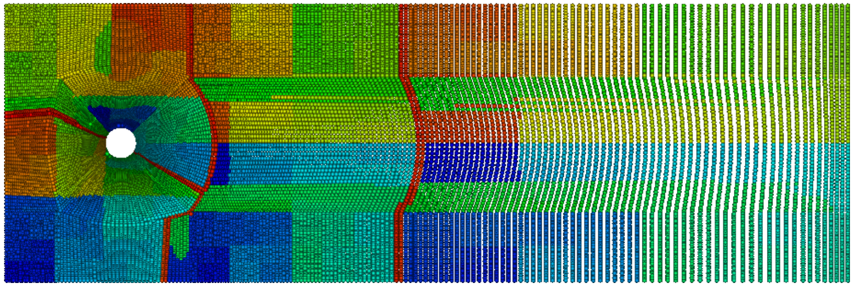
- Segregated solver
  - All variables are solved or independently, coupling terms are explicit
    - though components of the velocity are coupled, leading to a matrix with dense 3x3 blocks on the diagonal, and with a multiple of 3x3 Identity matrix off diagonal (spedific storage)
  - Solve by increments, with terms due to non-orthogonalities in the mesh mostly added at the RHS
    - *Multiple "sweeps" (solves) are possible* (and recommended for unsteady flows) to add contributions of mesh non-orthogonalities (same matrix, different increments/RHS)
  - This limits "reasonable" CFL values to 5-20
  - Algebraic multigrid solver (with PCG as smoother and solver) or diagonal-preconditioned CG used for pressure equation
  - Jacobi (or bi-Cgstab) used for other variables
    - BiCGSTab2 or GMRES or Gauss-Seidel/Jacobi hybrid also possible
- Some experiments with a fully coupled solver have been done outside EDF, but this may *not be easily generalizable* to all physical models, and would require a robust and scalable coupled solver
- More important to HPC, most matrices have no block structure, and are very sparse
  - Typically 7 non-zeroes per row for hexahedra, 5 for tetrahedra
  - Indirect adressing + no dense blocs means less opportunities for MatVec optimization, as memory bandwith is as important as peak flops.
  - Linear equation solvers often amount to 80% of CPU cost (dominated by pressure), gradient reconstruction about 20%
    - The larger the mesh, the higher the relative cost of the pressure step
  - By default, and for all operations outside linear solvers, a face→ cells based structure is used
    - analogous to FEM "edge-based" structures (the edges of the graph are the FV cell's faces)

Mesh renumbering

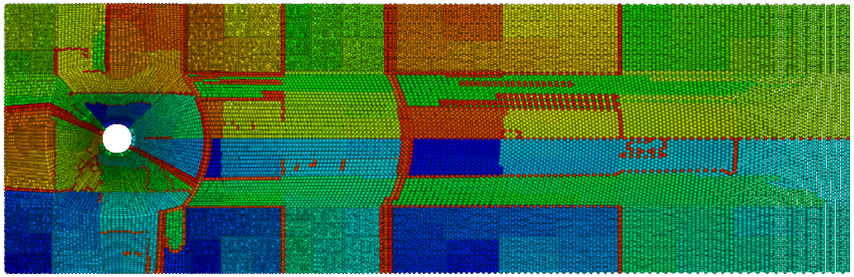
- Renumbering required to avoid thread write conflicts
  - due to the face-based nature of our loops
  - options to place halos-adjacent cells and faces last will also allow computation/communication overlap
    - MPI progress engines only recently asynchronous enough for this to be worthwhile
  - May improve cache effects, but low impact
    - 10% performance changes
    - initial numbering usually not so bad...



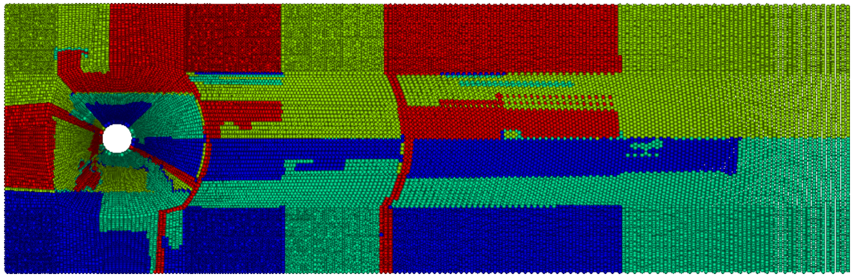
cell MPI rank id



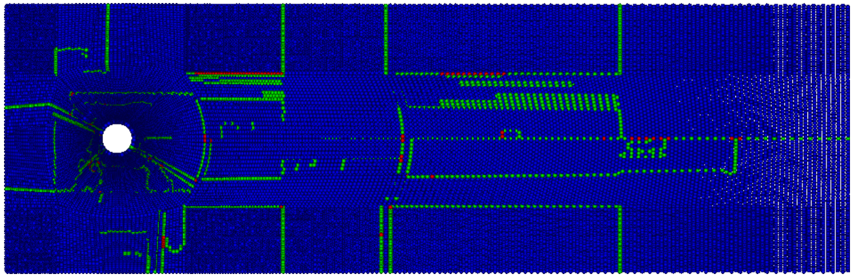
local cell id (Morton, ghost adjacent last)



local face id (Morton, ghost adjacent last)



local face thead id (Morton, ghost adjacent last)



local face thread group id (Morton, ghost adjacent last)

High Performance Computing with *Code_Saturne*

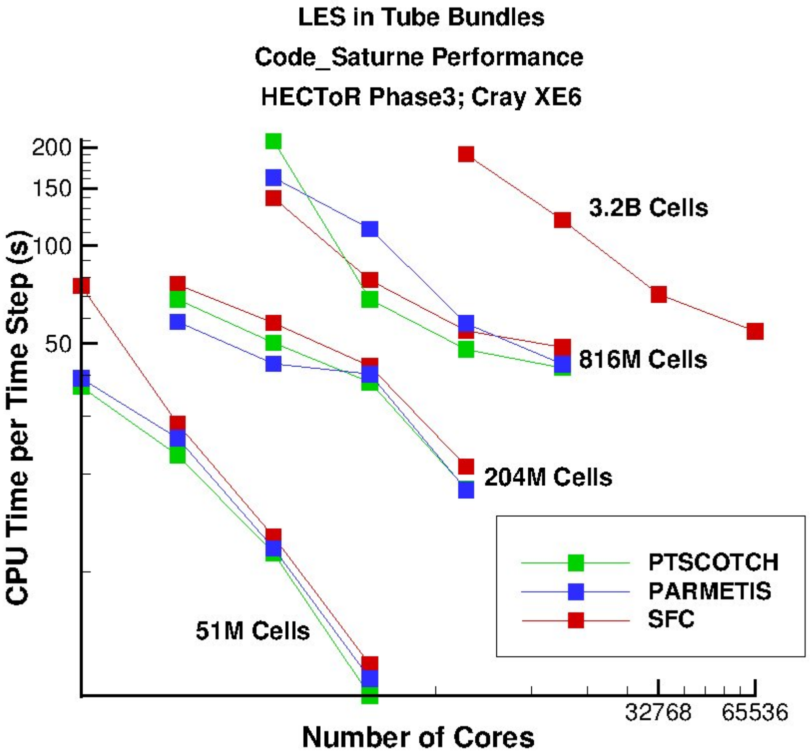
- *Code_Saturne* and NEPTUNE_CFD used extensively on HPC machines
  - EDF clusters (IBM Idataplex, Blue Gene/P; Blue Gene/Q)
  - CCRT calculation center (CEA based)
  - PRACE machines
    - Archer (EPCC), Jugene (FZJ), Curie (GENCI)
  - DOE machines (through INCITE access)
    - Jaguar (ORNL), MIRA (Argonne)

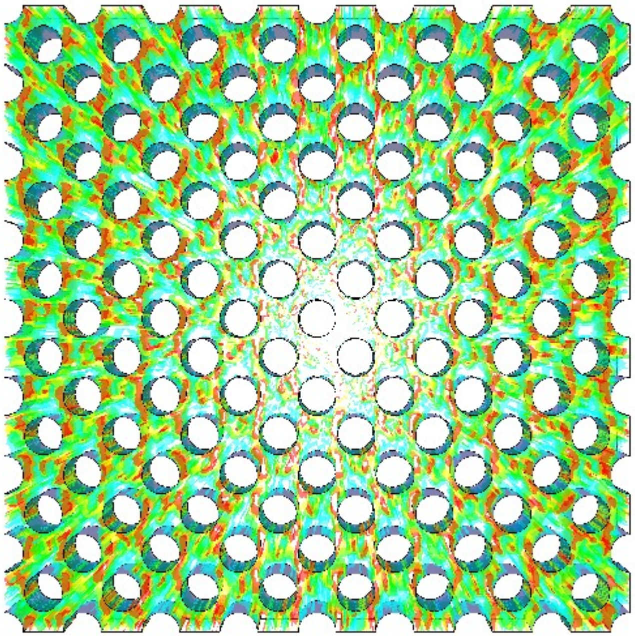
- Tests run by STFC Daresbury up to 7 billion cells on reference code
  - intensive work on parallel optimization and debugging loop
  - higher cell count reached with experimental global refinement from VSB
  - typical production studies use 10 to 50 million cells, with a few outliers in the 200-400 million cell range
    - largest production run to date: 1+ billion cells , by STFC and EDF R&D UK, on 4 racks of STFC's Blue Joule (Blue Gene/Q) machine

- *Code_Saturne* is one of the 12 codes selected for the PRACE and DEISA Unified European Application Benchmark Suite (UEABS)
  - along with QCD, NAMD, GROMACS, Quantum Espresso, CP2K, GPAW, ALYA, NEMO, SPECFEM3D, GENE, and GADGET
  - short list from 29 application codes in initial PRACE suite

Scalability of *Code_Saturne*

- Best scalability usually observed on Blue Gene and Cray machines
  - degrades faster on typical clusters
  - recent experience on X86/Inifiniband: do not always trust default MPI parameters…
- Scalability as a function of mesh size
  - At 65 000 cores and 3,2 Billion cells, about 50 000 cells / core





All to all operations (1)

- enable a family of fully distributed algorithms using global mesh element ids
  - no process needs to hold all data, as knowledge of which MPI rank holds a data element with a given if is immediate:
  - owning rank = global id / block size
    - We name this the block distribution, as opposed to the main partition distribution.
    - Data is usually read and written using the block distribution, while the code's main data structures and operations use the partition distribution
    - Conversely, simply using global numbers allows reconstructing neighbor partition entity equivalents mapping
  - Used for parallel ghost cell construction from initially partitioned mesh with no ghost data

- 2-stage algorithms are used:
  - count elements to exchange with each distant rank
    - loop on data to count elements to send to distant ranks
    - use MPI_Alltoall to exchange counts
  - exchange elements
    - using MPI_Alltoallv
    - requires looping on data again to build buffers

- In practice, the set of ranks with which data is exchanged is sparse, but not known in advance

All to all example code (previous)

- Redistribution example
  - some parts replaced by wrappers or utility functions in real code

```
block_size = global_count / size;
if (global_count % size > 0)
  block_size += 1;
send_count = malloc(n_elts*sizeof(int));
recv_count = malloc(n_elts*sizeof(int));
send_shift = malloc(n_elts*sizeof(int));
recv_shift = malloc(n_elts*sizeof(int));

/* Count number of values to send to each process */
/*------------------------------------------------*/

for (rank = 0; rank < size; rank++)
  send_count[rank] = 0;

for (i = 0; i < n_elts; i++)
  send_count[(global_num[i] - 1) / block_size] += 1;

MPI_Alltoall(send_count, 1, MPI_INT, recv_count, 1, MPI_INT,
             comm);

send_shift[0] = 0;
recv_shift[0] = 0;

for (rank = 1; rank < size; rank++) {
  send_shift[rank] = send_shift[rank - 1] + send_count[rank -1];
  recv_shift[rank] = recv_shift[rank - 1] + recv_count[rank -1];
}

n_ent_recv = recv_shift[size - 1] + recv_count[size - 1];

recv_global_num = malloc(n_ent_recv*sizeof(cs_gnum_t));
recv_order = malloc(n_ent_recv*sizeof(cs_lnum_t));

MPI_Alltoallv(_global_num, send_count, send_shift, CS_MPI_GNUM,
              recv_global_num, recv_count, recv_shift, CS_MPI_GNUM,
              comm);
```

```
/* Do work */
…

/* Return reverse (processed) info */

MPI_Alltoallv(recv_global_num, recv_count, recv_shift, CS_MPI_GNUM,
              _global_num, send_count, send_shift, CS_MPI_GNUM,
              comm);

/* Free memory */

free(recv_order);
free(recv_global_num);
free(send_count);
free(recv_count);
free(send_shift);
free(recv_shift);
```

All to all example code (new)

- Redistribution example
  - data movement is hidden
    - allows runtime choice of algorithm
    - loses some count (previously MPI_Alltoall-based) reuse opportunities

```
block_size = global_count / size;
if (global_count % size > 0)
  block_size += 1;

dest_rank = malloc(n_elts*sizeof(int));

for (i = 0; i < n_elts; i++)
  dest_rank[i] = (global_num[i] - 1) / block_size;

/* Create distributor and associate data elements */

d = cs_all_to_all_create_s(n_part_elts,
                           data_stride,
                           datatype,
                           data,
                           dest_rank,
                           comm);

free(dest_rank);

cs_all_to_all_exchange(d);

/* Get number of elements in "receiving" distribution */

n_block_elts = cs_all_to_all_n_elts(d);
cs_all_to_all_get_data_pointer(d, &data_stride, &p);
```

```
/* Do work */
…

/* Return reverse (processed) info */

cs_all_to_all_swap_src_dest(d);
cs_all_to_all_exchange(d);

/* Sort by initial destination rank, then retrieve values */

cs_all_to_all_sort_by_source_rank(d);
cs_all_to_all_get_data_pointer(d, &data_stride, &p);

/* Do work */
…

cs_all_to_all_destroy(&d);
```

All to all algorithms (1)

- the new all to all API currently allows selecting from 2 algorithms
  - the existing MPI_Alltoall/MPI_Alltoallv based algorithm
  - a Crystal router algorithm (similar to a binomial algorithm)

- Binomial algorithm and Crystal router base idea
  - if local rank_id <= n_ranks /2
    - send all data with destination rank > n_ranks /2 to (rank_id + n_ranks/2)
    - keep all data with destination rank <= n_ranks /2
  - if rank_id > n_ranks /2
    - keep all data with destination rank > n_ranks /2
    - send all data with destination rank <= n_ranks /2 to (rank_id – n_ranks/2)
  - recurse on subsets of communicator, dividing size by 2 at each level

- Each variant has advantages/disadvantages
  - MPI_Alltoall/MPI_Alltoallv requires loops and arrays based on rank counts
    - those can become larger than the local data size at high processor counts
    - doubling rank count for a given data set doubles the size of those loops, leading to inverse scaling
    - increasing rank count significantly increases memory usage
  - Crystal router requires looping over and sending data multiple (log(p)) times before it reaches the destination rank
    - if it perfectly matches the network topology, this might match what MPI does would do and simply make routing explicit, but there is a small chance of that (and next destination determination loops are required for each level anyways)

All to all algorithms (2)

- the new all to all API is not completely deployed yet
  - strided variant is implemented, indexed one not yet
    - needs to be done before it can be applied fully
  - many places in the code where this needs to be done
    - about 80
    - specificities requiring extra caution in about 1/3 of cases
  - work in progress
    - performace data is thus limited to the few places where it is deployed

- some results on Blue Gene/Q
  - using 12.8 million cell benchmark test
  - not enough data points here
  - similar tendencies on Intel/Infiniband clusters

n_ranks	Alltoall(v) time (s)	CrystalRouter time (s)
128 (16x8)	0.03	0.45
256 (16x16)	0.02	0.26
512 (32x16)	0.012	0.20

Impact of MPI collectives (comparison with older tests)

- In previous tests, we saw inverse scaling in some stages that are believed to be due to data redistribution
  - but those issues appear at much higher rank counts

N cores	Morton		ParMetis	
	Partiton	Ghost creation	Partition	Ghost creation
8192	14.8	14.3	11.1	8.7
16384	9.7	16.1	8.9	7.8
32768	16.4	40.8	18.7	17.7
65536	57.6	94.1	81.3	63.1

Mesh partitioning and ghost cell creation times, in seconds, for 3.2 billion celll mesh (Jaguarpf)

All to all current status

- A new API is mostly in place, allowing future run-time selection of redistribution algorithms
  - performance results still favor the “classical method”
  - this is expected to change at higher rank counts
  - one advantage of the API usage is that extra instrumentation is done, specifically for those operations
    - once deployed everywhere, much more complete tests will be possible

- Currently, MPI_Alltoallv usage is believed to be where memory usage peaks
  - when running on Blue Gene/Q, for example, some cases fail due to lack of memory in those stages when trying to run with 32 or 64 MPI ranks per core
    - the same cases are OK using hybrid MPI/OpenMP with 16 ranks per core

- A 3rd algorithm variant may be possible
  - based on direct sends to the final destination ranks, matched with receives using MPI_ANY_SOURCE, and MPI-3 asynchroneous barrier
  - described in advanced MPI tutorial from SC14 conference
  - seems a perfect fit to this problem
  - no risk if done within new API

IO performance

- Parallel IO performance is hard to measure, as it may depend both on the machine and its load
- We consider only single files, read/written in collaboration by many processes (MPI_File_*_all)
  - Usually effective on GPFS
  - Effective on recent versions of LUSTRE, using striping
    - on older versions, minimal improvement, even with striping
  - Tests on our own machines (GPFS), many tests in PRACE context by STFC
    - on MIRA, about 2 to 5 Gb/s write
  - Timings include all-to-all,
    so pure IO may be better
  - Search for PRACE reports on *Code_Saturne* IO by STFC for many additional timings

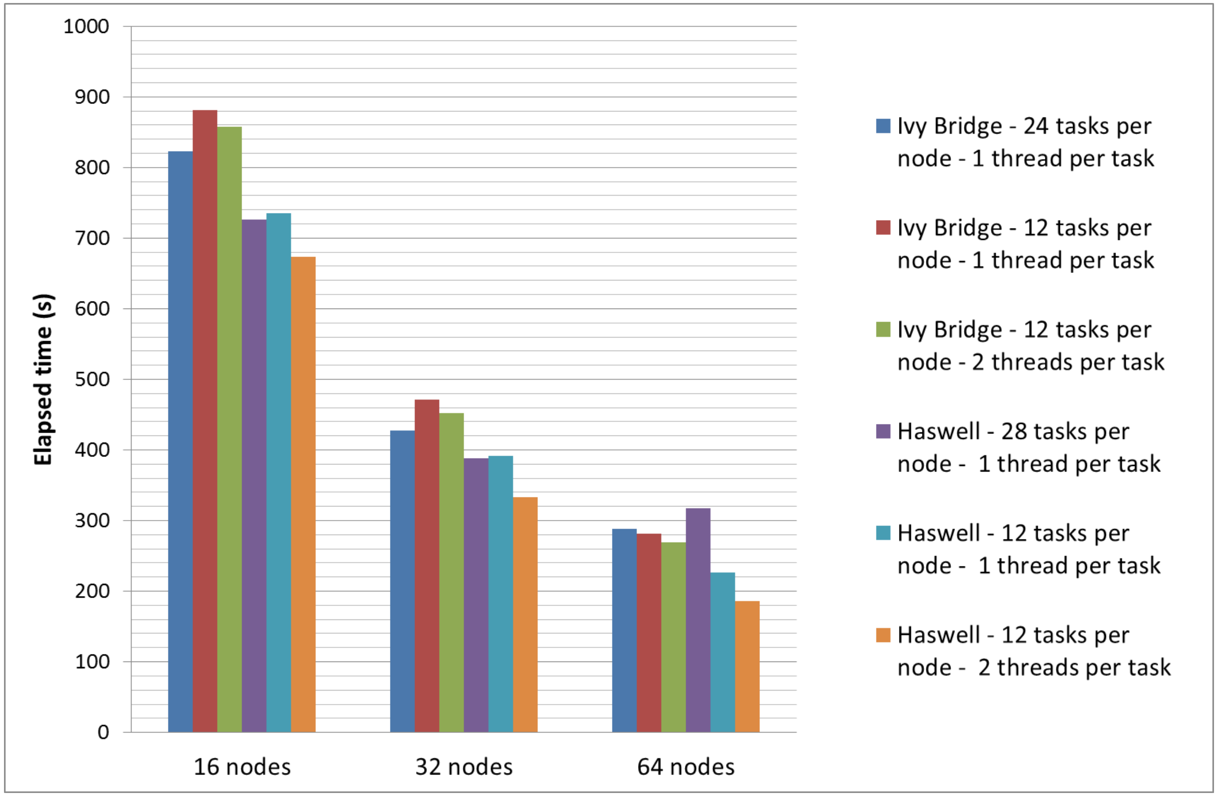
Hybrid Parallelism (1)

- Since version 4.0, hybrid parallelism using OpenMP is in a working state
  - not built by default in 4.0
  - built by default in development “trunk”
  - Requires mesh renumbering
    - Also useful for cache behavior
    - Cache effects even more important under OpenMP, to avoid false sharing, but also try not to saturate bandwidth
    - Both internal (in progress) and external (IBM library) renumberings are possible, and may be compared
  - some subsets of the code have better OpenMP scaling
  - some subsets do not use OpenMP

- OpenMP debugging still very painful
  - Valgrind DRD helps
    - very high overhead
    - requires compiling gcc with specific option
  - Writing C code with loop local variable definitions helps avoid missing “private” qualifiers
    - no equivalent in Fortran
    - alternative would be to use a tasking/data model more similar to MPI, but this would be fragile

Hybrid Parallelism (2)

- Bandwidth becomes the main limitation
- On several recent clusters, best results are observed with 2 threads per task, MPI for all the rest
  - example on 51-million cell tube bundle test case



What about accelerators ?

- Accelerators of some form in almost all future hardware options: they cannot be ignored
- Diminishing returns with current OpenMP implementation
- Preliminary work done in collaborative work, testing external libraries (CUSparse, ViennaCL) for linear solvers with matrices from current test cases
  - Single node at this point
    - not ghost cell synchronization / GPU ghost cell definition / GPU direct issues
  - CPU better for cases which fit well in cache
  - GPU better for larger local workloads
- Due to the large size of the code base, rewriting large portions of it to CUDA or OpenCL would be slower than the evolution of those languages
  - except for some kernels, directive-based approach seems the only solution
    - OpenMP again, OpenACC
- Not ready yet, but working on initial issues
  - PhD started with INRIA to study tasking and load balancing options at all levels
    - using microbenchmarks from *Code_Saturne* algorithms

External libraries

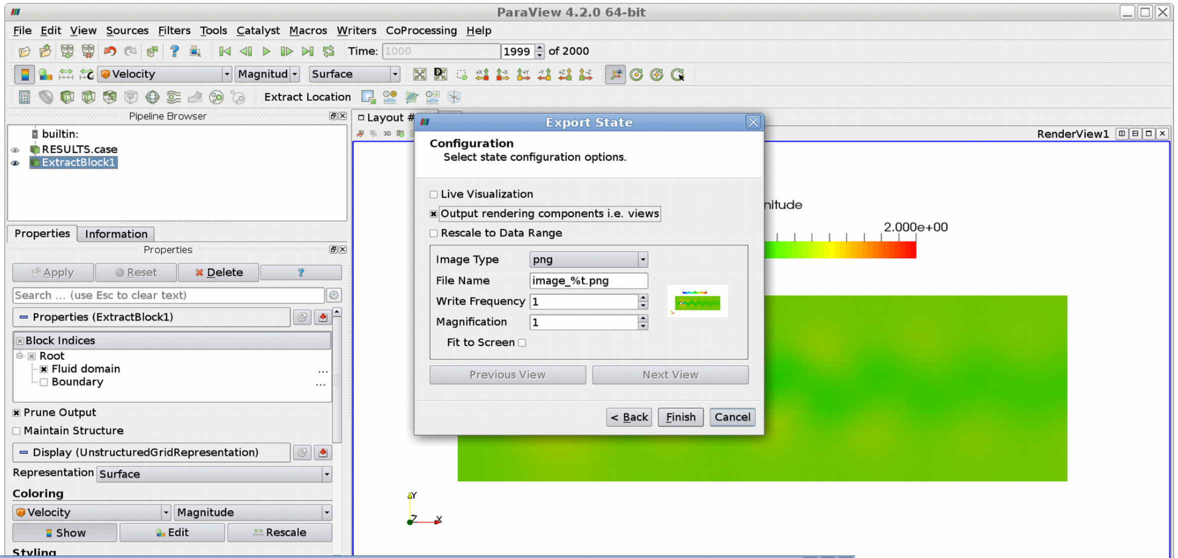
- Co-visualization and in-situ visualization now available using ParaView Catalyst
  - Easy to setup using ParaView wizard and Code_Saturne GUI
  - some bugs remain, but robustness has increased in the last year
    - mainly dependent on ParaView's progress

- Faster parallel IO may soon not be enough
  - Co-visualization is part of the solution
  - user subroutines also allow some simple in-situ analytics
  - we need to help users with meeting their analytics needs in situ, to avoid explosion of storage requirements
  - also need to change user habits…

- Leverage external libraries
  - So far, the few comparisons we have done between built-in solvers and those of external libraries have shown quite good performance of built-in features, without the overhead of external library deployment and synchronization
  - With increasing hardware, software, and solver complexity, we need to enable the use of HPC libraries such as PETSc, HYPRE, and Trilinos more easily, at least for testing
    - especially in the view of recent efforts relative to hybrid platforms in those libraries
  - 2014 refactoring of linear solver handling allows for easier integration
    - current testing of PETSc, Hypre, and Lis is underway

Co-visualization / in situ visualization

- Using ParaView Catalyst
  - Use ParaView wizard on initial post-mortem visualization
    - possibly on a coarser/ placeholder mesh
  - Then set writer format to Catalyst in *Code_Saturne*
    - possible with GUI
  - live connection also works
    - tested on workstations...



Time: 1000 1999 of 2000

builtin:
RESULTS.case
ExtractBlock1

Properties (ExtractBlock1)

Block Indices
- Root
  - ☒ Fluid domain
  - ☐ Boundary

☒ Prune Output

☐ Maintain Structure

Display (UnstructuredGridRepresentation)

Representation: Surface

Coloring: Velocity, Magnitude

Show | Edit | Rescale

Styling

Export State

Configuration
Select state configuration options.

☐ Live Visualization

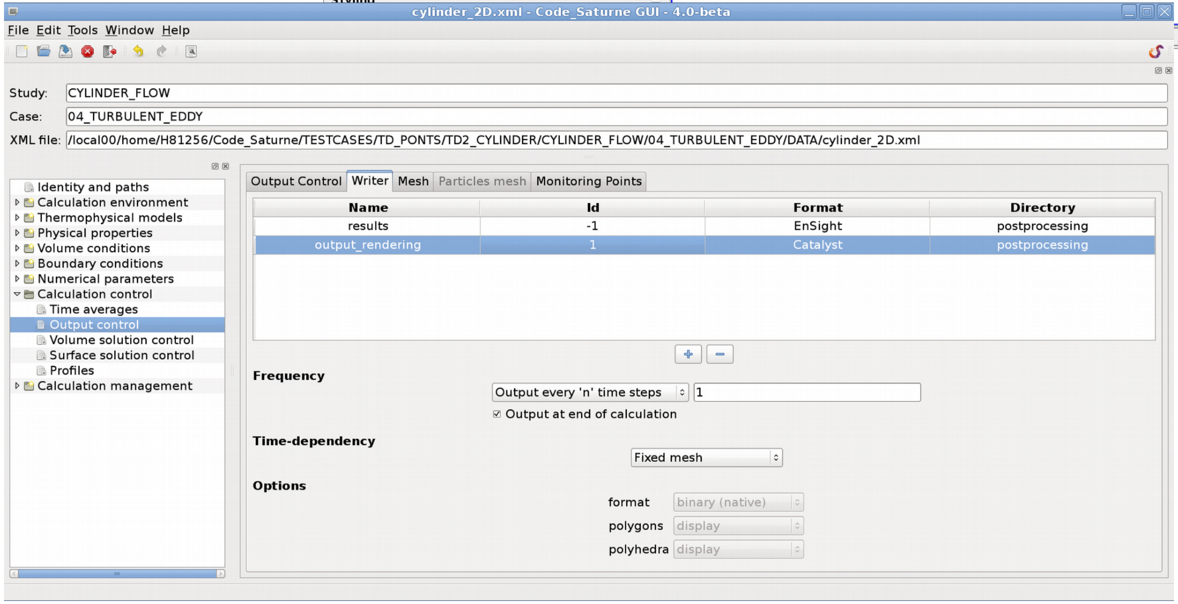
☒ Output rendering components i.e. views

☐ Rescale to Data Range

Image Type	png
File Name	image_%t.png
Write Frequency	1
Magnification	1
Fit to Screen	☐

Previous View | Next View

< Back | Finish | Cancel



Study: CYLINDER_FLOW

Case: 04_TURBULENT_EDDY

XML file: /local00/home/H81256/Code_Saturne/TESTCASES/TD_PONTS/TD2_CYLINDER/CYLINDER_FLOW/04_TURBULENT_EDDY/DATA/cylinder_2D.xml

Name	Id	Format	Directory
results	-1	EnSight	postprocessing
output_rendering	1	Catalyst	postprocessing

Frequency

Output every 'n' time steps: 1

☑ Output at end of calculation

Time-dependency

Fixed mesh

Options

format: binary (native)

polygons: display

polyhedra: display

Compatible Discrete Operators

- Medium term additions
  - Better mesh robustness
  - Preliminary work: Jérôme Bonelle and Alexandre Ern (2014). Analysis of Compatible Discrete Operator schemes for elliptic problems on polyhedral meshes. ESAIM: Mathematical Modelling and Numerical Analysis, 48, pp 553-581. doi:10.1051/m2an/2013104.
- Work on Compatible Discrete operator methods leads to different matrices
  - currently, lines match cells
    - diagonal entries match cells
    - non-diagonal entries match cell-cell boundaries (i.e. faces)
  - new discretizations may require additional contributions
    - cell-cell, face-face, cell-face terms in the same system
    - some lines are for cell connectivities, others for faces

- This will lead to a gradual switch to CSR or MSR type structures
  - already possible, but not the default
    - matrices build using native face-cell connectivity, with assignment/conversion cost
  - New systems with multiple blocks are directly built using those structures

- Building matrices using separate logical blocs (cell and face contributions for example) will lead to artificially increase cache-affecting distances
  - lines relative to a face adjacent to a given cell will be numbered after all lines related to cells, while some extradiagonal terms will link the 2
  - renumbering is expected to have a higher performance benefit in this case

Other future steps

- Pursue integration with the SALOME platform
  - Only integrate directly with components which are HPC compatible
    - Or in a manner compatible with HPC
    - Currently, Visualization (Paraview/ParaVis) is HPC compatible
    - Mesh is making progress
  - Coarser Integration (using files) for parts of the platform which are less HPC oriented
  - For future ensemble calculations, may benefit from OpenTurns integration for driving of uncertainty determination

- Mesh refinement (global multiplication or adaptation)
  - work on this in PRACE context at VSB / IT4I Ostrava
    - search for presentations by A. Ronowský
    - in collaboration with STFC

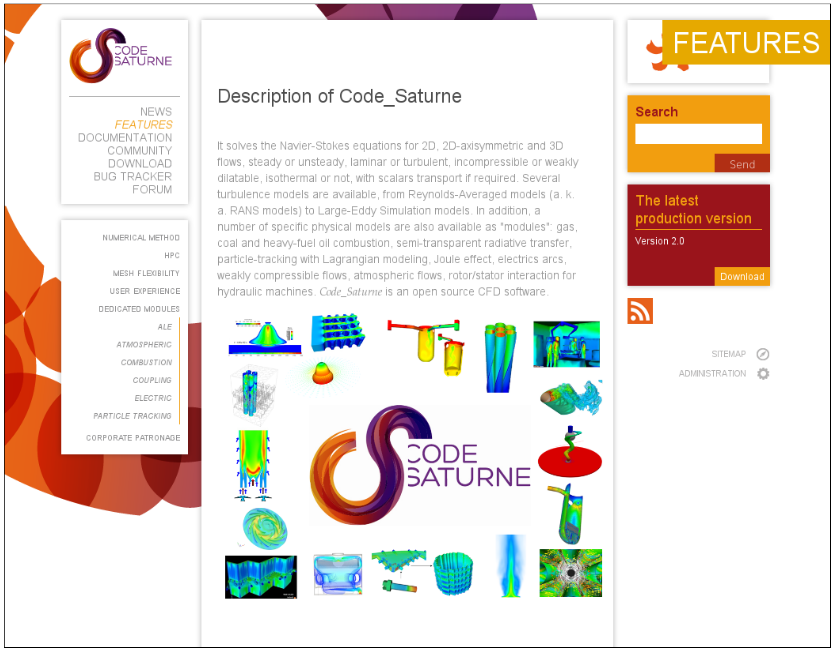
- Test parallel meshers / add readers if required

- Optimize for future ensemble calculations
  - Using in memory data staging (avoiding files) with ADIOS, HDF5, or similar technologies may mitigate IO volumes
  - Pseudo code coupling (actually postprocessing coupling) may allow determine key statistics with less I/O and archival
    - This needs to be done in a relatively fault-tolerant manner
      - one run crashing must not cause the loss of the whole ensemble
      - Nesting ?

Code_Saturne open source practical info

- Distribution of Code_Saturne
  - GPL license, auxiliary library (PLE) under LGPL license

- Code_Saturne EDF website
  - http://code-saturne.org
  - source download
  - Code presentation and documentation
  - Contact with EDF development and support team
  - Code_Saturne news
  - Forum and bug-tracker



Description of Code_Saturne

It solves the Navier-Stokes equations for 2D, 2D-axisymmetric and 3D flows, steady or unsteady, laminar or turbulent, incompressible or weakly dilatable, isothermal or not, with scalars transport if required. Several turbulence models are available, from Reynolds-Averaged models (a. k. a. RANS models) to Large-Eddy Simulation models. In addition, a number of specific physical models are also available as "modules": gas, coal and heavy-fuel oil combustion, semi-transparent radiative transfer, particle-tracking with Lagrangian modeling, Joule effect, electrics arcs, weakly compressible flows, atmospheric flows, rotor/stator interaction for hydraulic machines. *Code_Saturne* is an open source CFD software.

THANK YOU FOR YOUR ATTENTION

any questions ?