Evolution of EDF's *Code_Saturne* industrial CFD tool: numerical and HPC-oriented aspects

Yvan Fournier, EDF

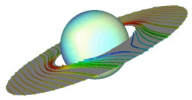
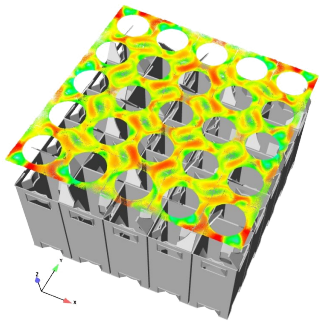
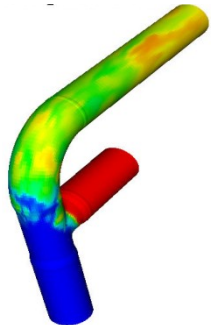
August 17, 2015

Outline

- GENERAL FEATURES
- APPLICATION EXAMPLES
- PERFORMANCE
- HPC ISSUES
- WORK IN PROGRESS AND ROADMAP

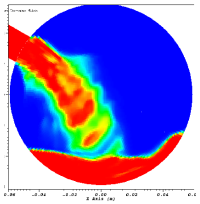
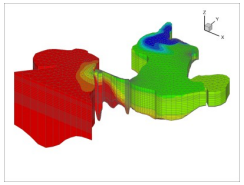
Code development at EDF R&D (1/2)

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

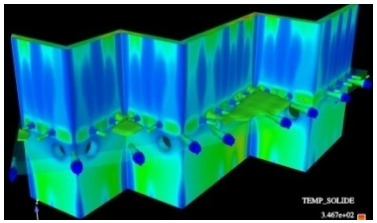
- NEPTUNE_CFD
  - multiphase CFD, esp. water/steam
  - property of EDF/CEA/AREVA/IRSN

NEPTUNE

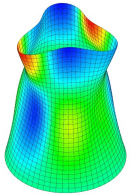



- SYRTHES
  - thermal diffusion in solid and radiative transfer
  - property of EDF, open source (GPL)
  - http://rd.edf.com/syrthes

SYRTHES

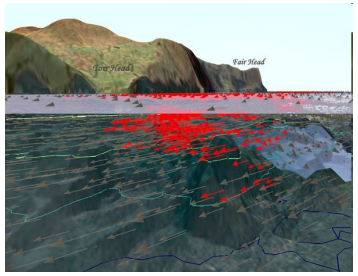


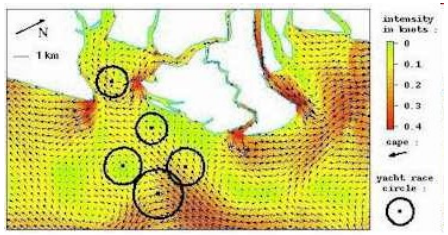
- *Code_Aster*
  - general usage structure mechanics
  - property of EDF, open source (GPL)
  - http://www.code-aster.org

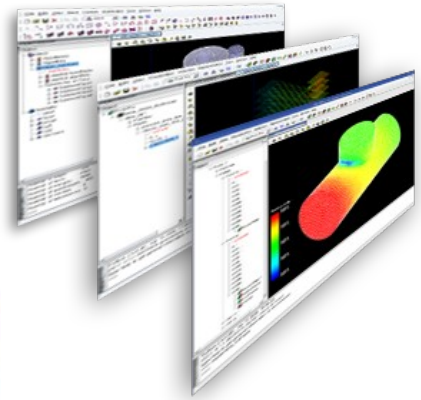
Code development at EDF R&D (2/2)

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

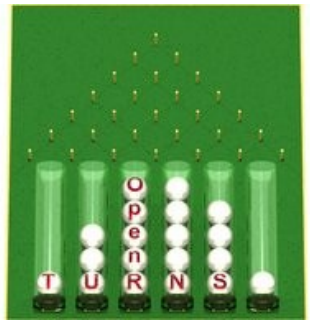




- SALOME platform
  - integration platform (CAD, meshing, post-processing, code coupling)
  - property of EDF/CEA/OpenCascade, open source (LGPL)
  - http://www.salome-platform.org

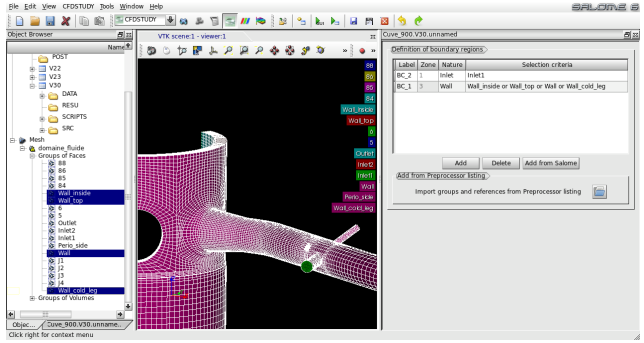


- Open TURNS
  - tool for uncertainty treatment and reliability analysis
  - property of EDF/CEA/Phimeca, open source (LGPL)
  - http://trac.openturns.org

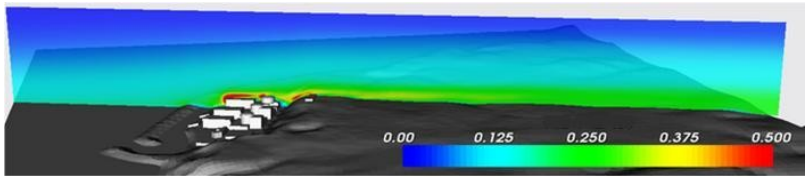


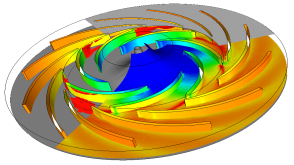
- and many others
  - Neutronics, electromagnetism
  - component codes, system codes
  - ...

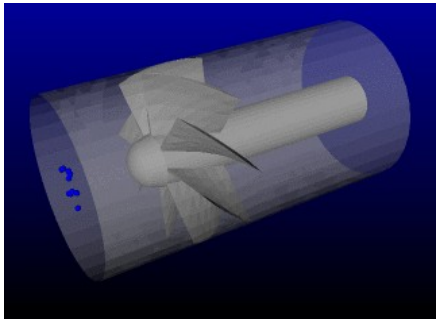
Code_Saturne
EDF's general purpose CFD tool



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







Numerical features of note to HPC aspects

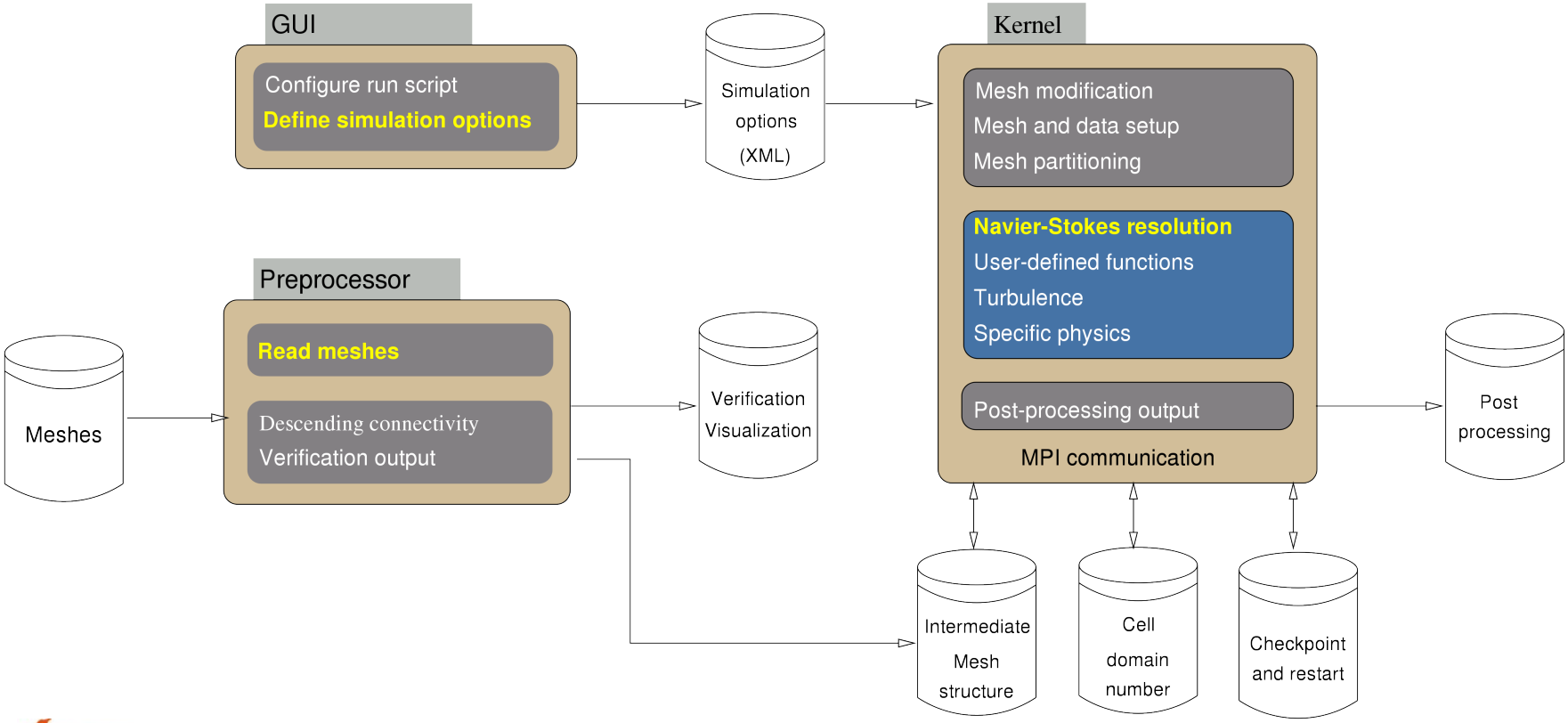
- Segregated solver
  - All variables are solved independently, coupling terms are explicit
    - components of the velocity are coupled, leading to a matrix with dense 3x3 blocks on the diagonal, and with a multiple of 3x3 Identity matrix off diagonal (specific storage)
    - recent addition of similar 6x6 block structure for RSM models
  - 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
- Experiments with a fully coupled solver have been done outside EDF, but this may *not be easily generalizable* to all physical models
  - would require a robust and scalable coupled solver
  - expected performance would be lower in most cases

Other numerical features of note to HPC aspects

- 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 and latency are more 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
  - For most operations outside linear solvers, a face→ cells based structure is used
    - analogous to FEM “edge-based” structures (the graph edges are the FV cell's faces)
- As we use mostly sparse matrix-vector and BLAS-1 (vector-vector) operations, expected performance can be determined more by benchmarks such as the stream benchmark (http://www.cs.virginia.edu/stream/ref.html#counting) or the HPCG benchmark (http://hpcg-benchmark.org/) than the HPL (High Performance Linpack) benchmark or machine peak performance
  - The roofline model explains this in an intuitive manner
  - http://crd.lbl.gov/departments/computer-science/performance-and-algorithms-research/researc
  - http://crd.lbl.gov/assets/pubs_presos/PMBS14-Roofline.pdf

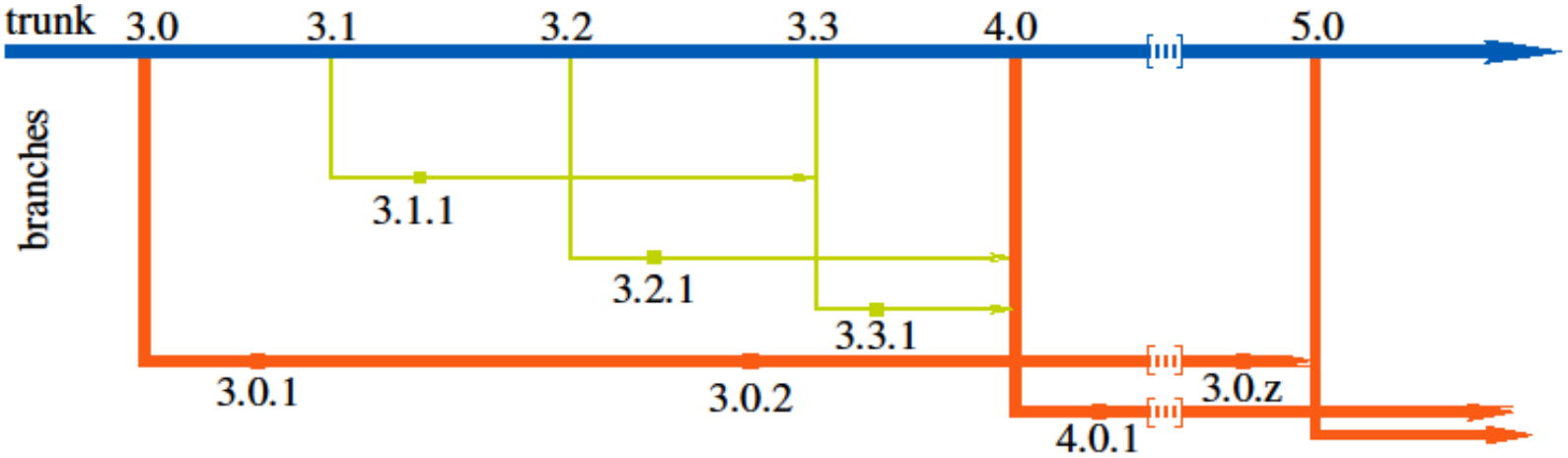
Code_Saturne toolchain

- 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



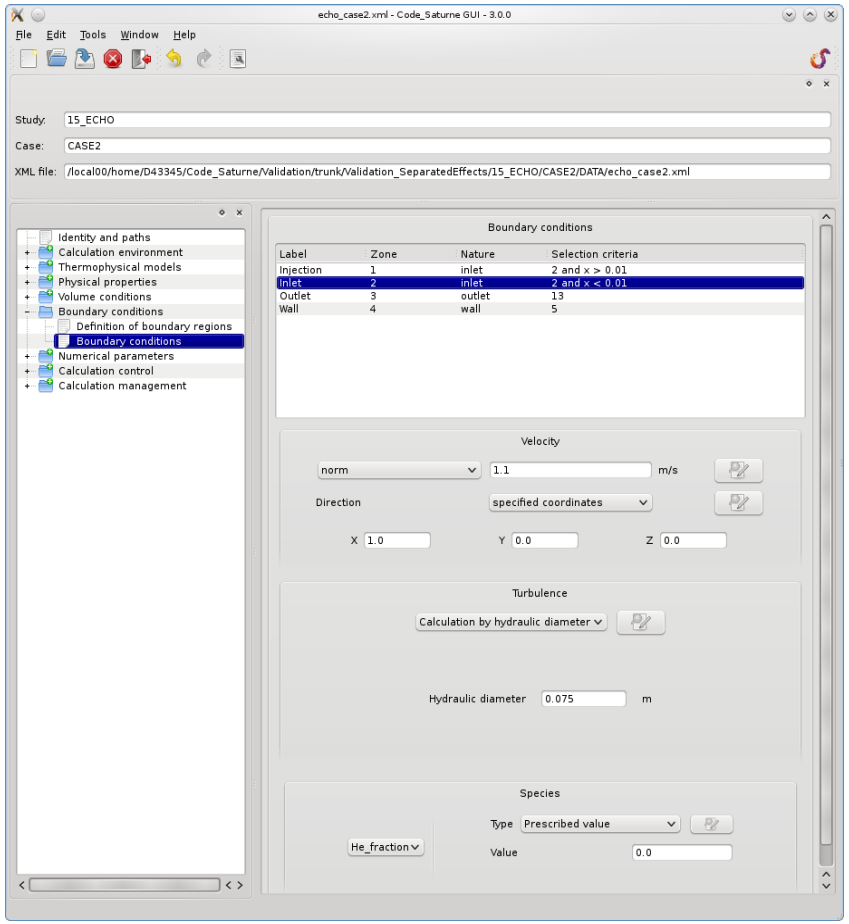
Code_Saturne VVUQ and versioning

- Quality assurance and VVUQ described in detail in other talks here
- Simple versioning scheme, with different kinds of versions "*x.y.z*"
  - Production (LTS) version every two years (*x*), with Verification & Validation summary report
  - Intermediate version every six months (*y*), with non-regression tests
  - Corrective versions when needed (*z*), for bug fixes and ports
- Data setup incompatibility-introducing changes allowed between all x or y versions
  - corrective (z) releases stable
  - GUI has automatic data update features
  - user subroutines must be updated/rewritten



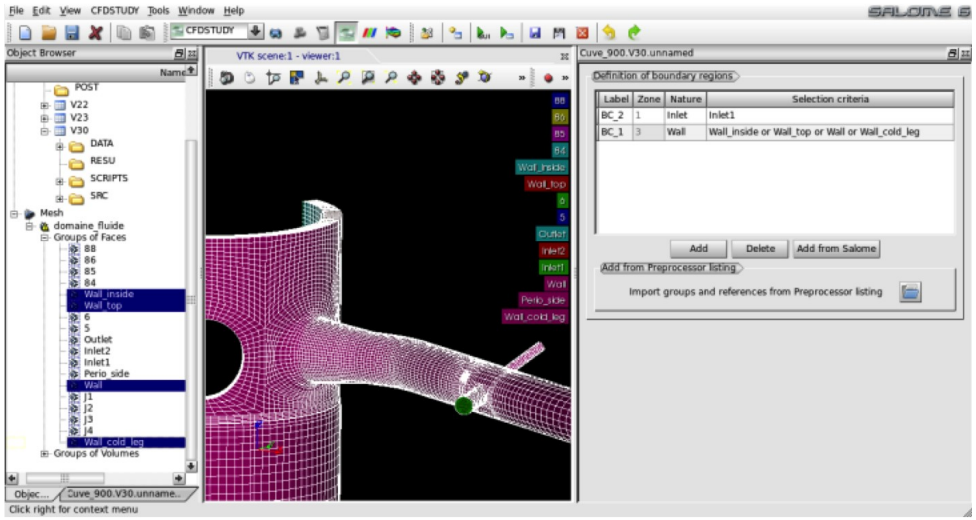
Required Environment: **Linux**, **Unix**, **OS-X**, Window (from laptop to supercomputer)

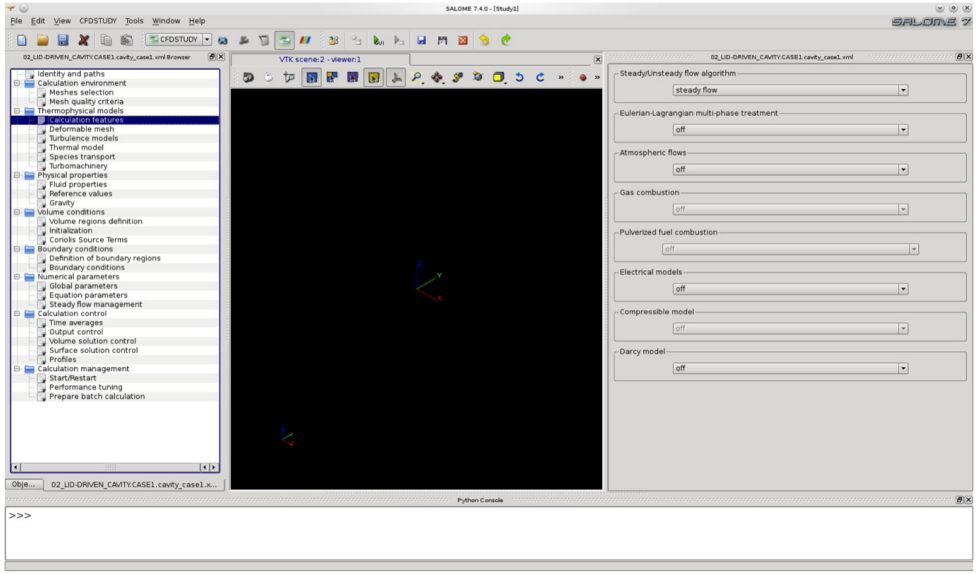
- Pre-requisite: compilers
  - C (gcc, xlc, icc, …)
  - Fortran (gfortran, xlf, ifort, ...)
- Pre-requisites for parallel computing:
  - MPI library : Open MPI, MPICH, ...
  - PT-SCOTCH or ParMETIS partitioner (optional)
- Pre-requisites for GUI:
  - Python, Qt4, SIP, libxml2
- Optional data exchange 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



Increasing use of the SALOME platform

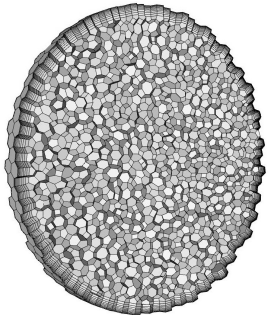
- Provides a complete (mesh to visualization) environment
  - www.salome-platform.org
- Some parts of SALOME are better at HPC than others
  - Integration does not get in the way of HPC aspects





Supported Meshes

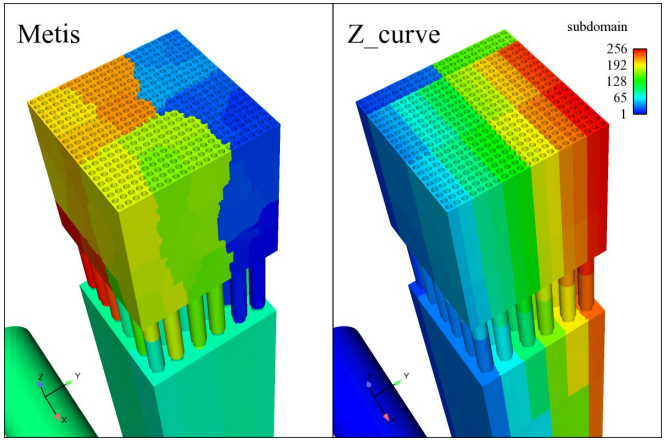
- Mesh generators
  - SALOME SMESH (http://salome-platform.org)
  - GAMBIT (Fluent), ICEM-CFD, ANSYS meshing
  - Star-CCM+
  - Simail: easy-to-use, with command file, but no CAD
  - I-deas NX
  - Gmsh
- Formats
  - MED, CGNS, Star-CCM+, Simail, I-deas universal,
  - GAMBIT neutral, EnSight Gold
- Cells: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



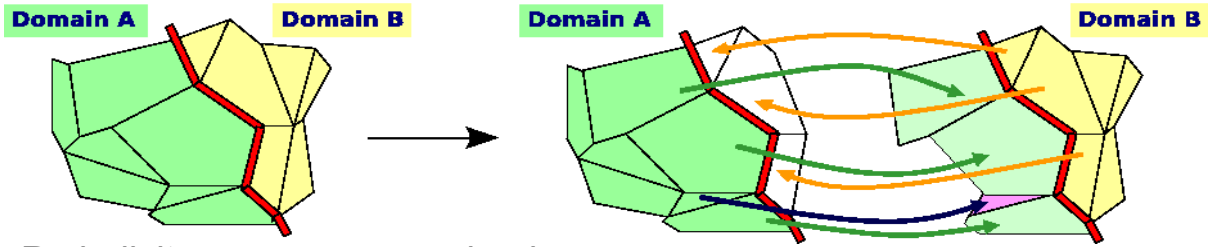
Installation on HPC resources

- 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

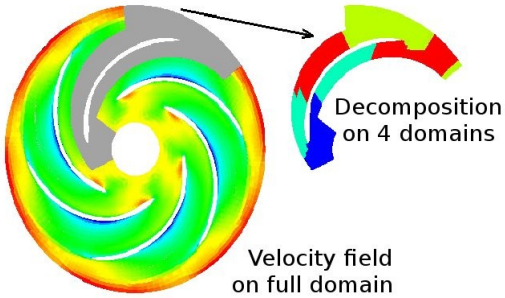
Parallelism and periodicity (1)



- 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

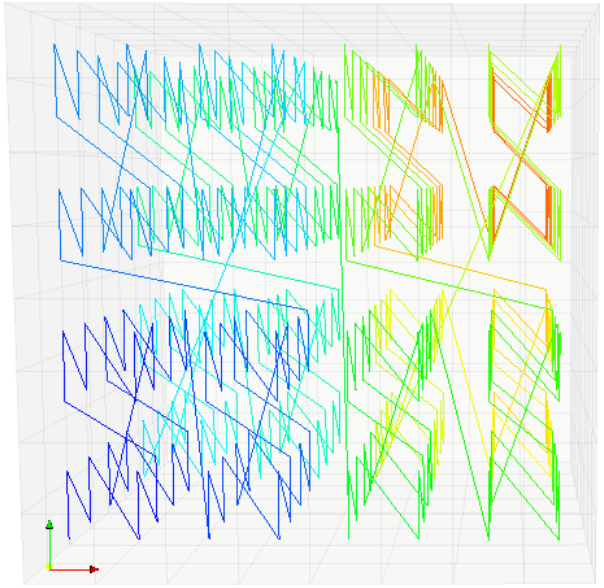


  - Periodicity uses same mechanism
    - True geometric periodicity (not a BC)
    - Vector and tensor rotation may be required (semi-explicit tensor component coupling in rotation)
  - Input output is partition independent
- Recently added OpenMP for compute-intensive sections

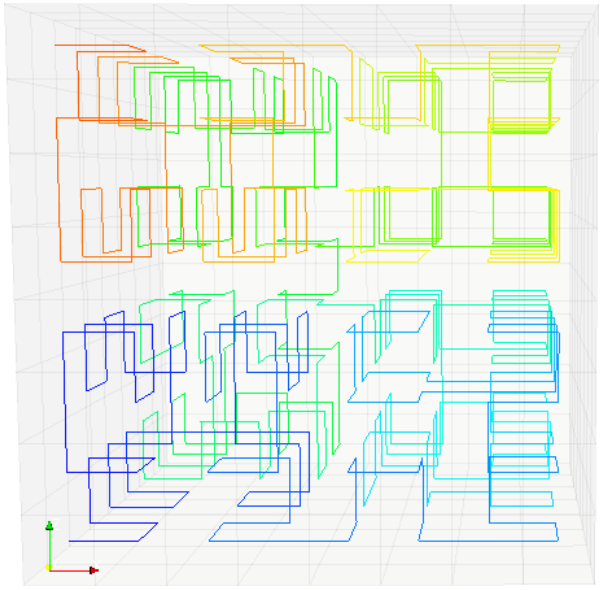


Mesh partitioning (1)

- 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



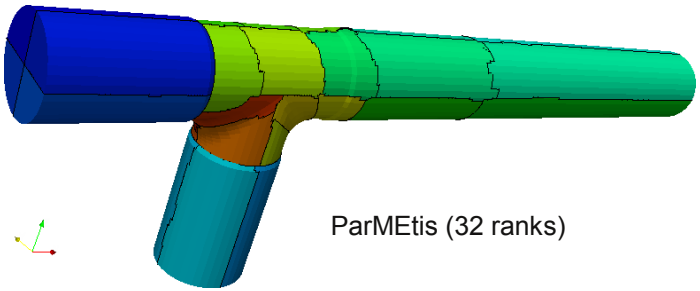
Morton curve



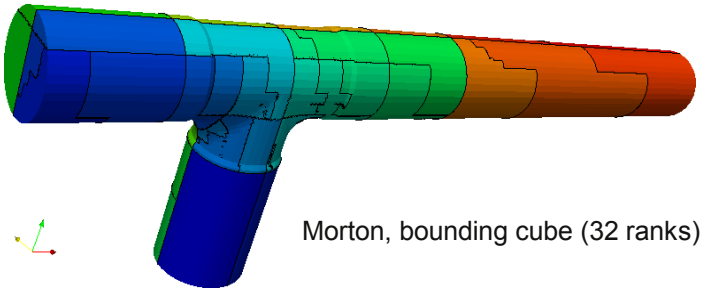
Hilbert curve

Mesh partitioning (2)

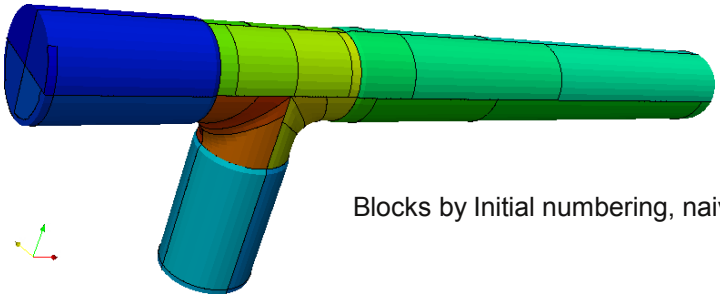
- Depending on mesh, results may vary
  - In some rare cases, even naive renumbering may be OK



ParMEtis (32 ranks)



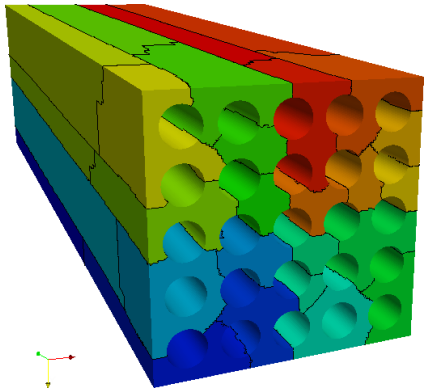
Morton, bounding cube (32 ranks)



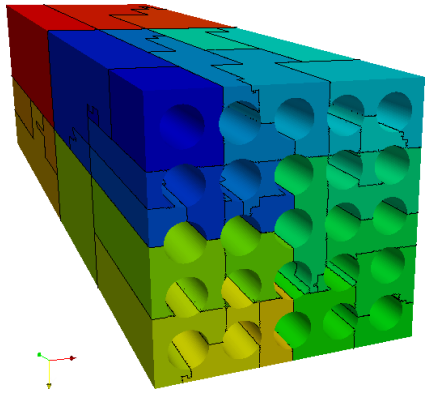
Blocks by Initial numbering, naive (32 ranks)

Mesh partitioning (3)

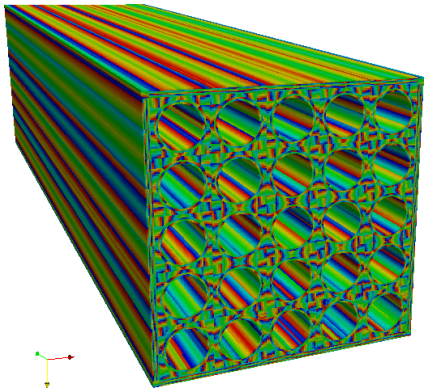
- In most cases, both are acceptable, while naïve partitioning is not
  - Always safe, 20 to 50% performance impact nonetheless



PT-Scotch (32 ranks)



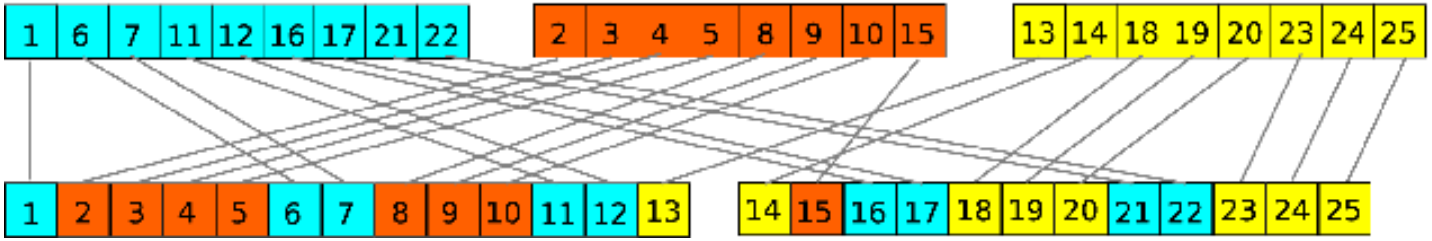
Hilbert, bounding box (32 ranks)



Blocks by Initial numbering, naive (32 ranks)

Parallelism and periodicity (2)

- 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)

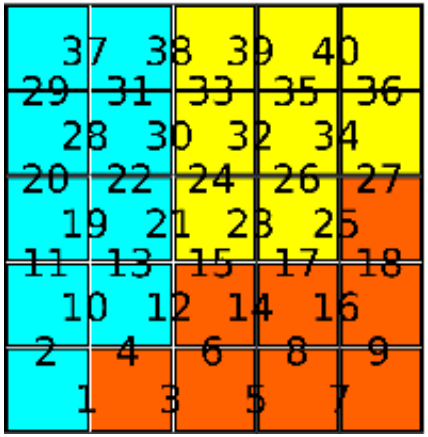


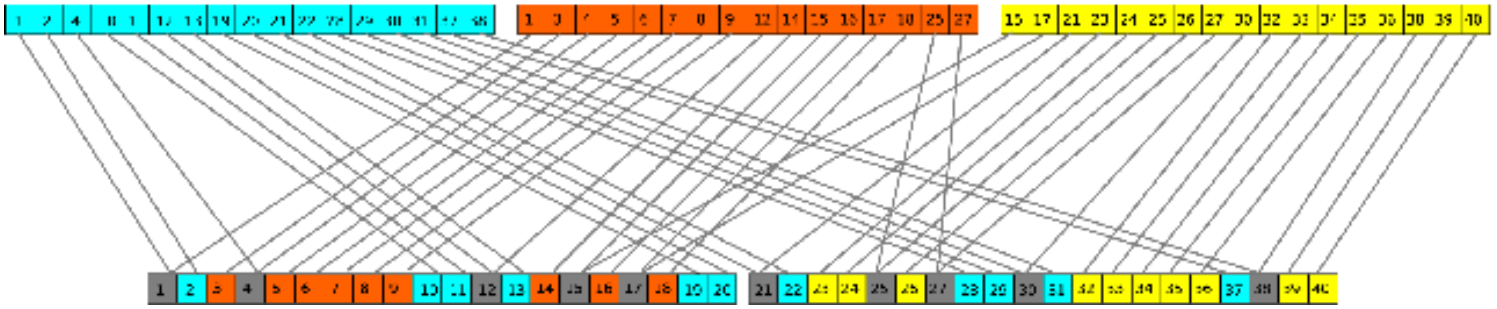
Parallelism and periodicity (3)

- 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
  - Similar to “assumed partition” algorithm

- 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

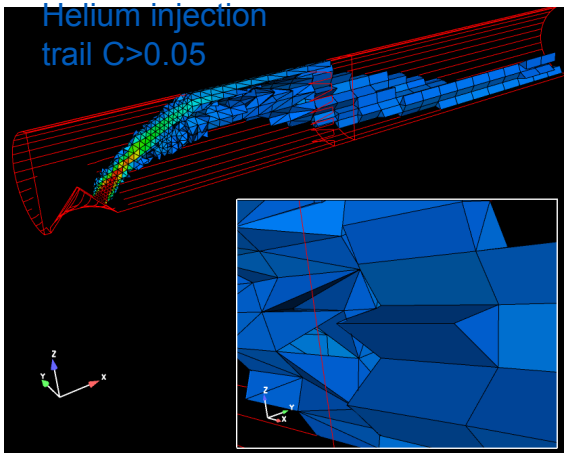


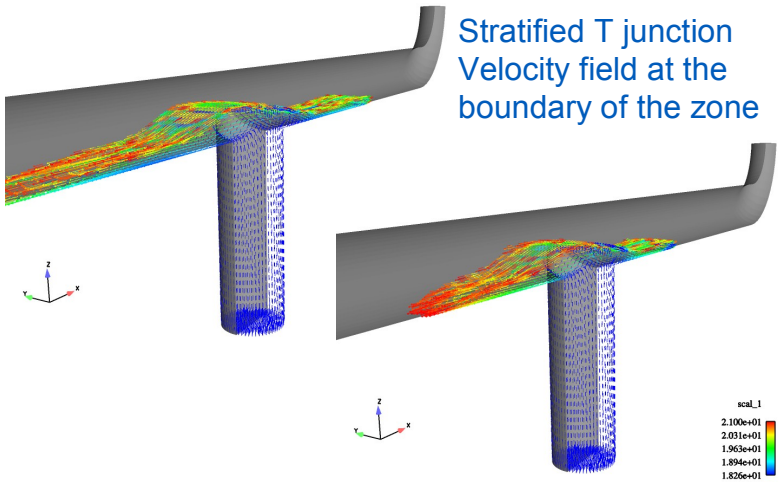


Postprocessing output

- 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

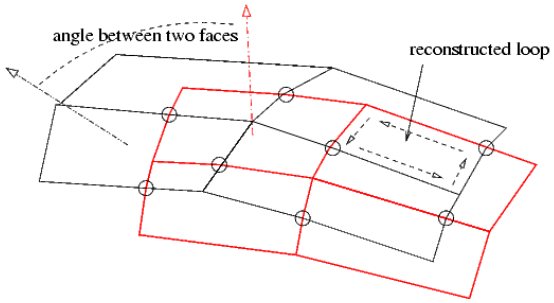
- This allows fine control of user output
  - Frequently output fields may be associated to partial data, on a possibly time-dependent mesh subset
    - Useful for example to output partial data at higher frequency so as to generate movies, with “reasonable” output volumes

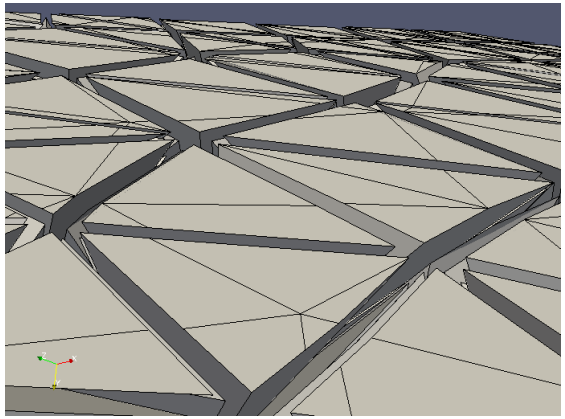


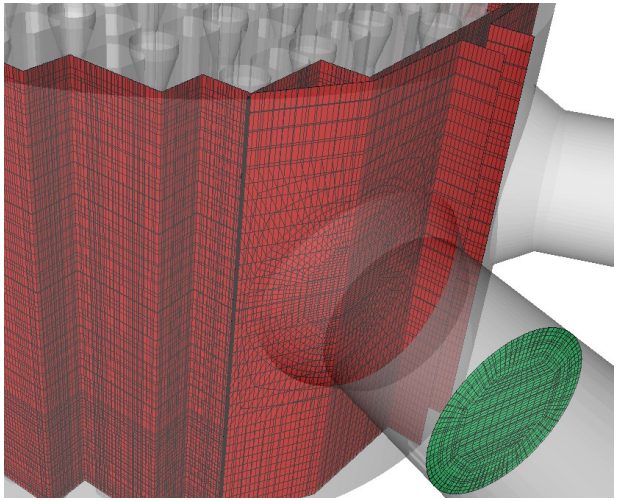


Mesh Joining (1)

- 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

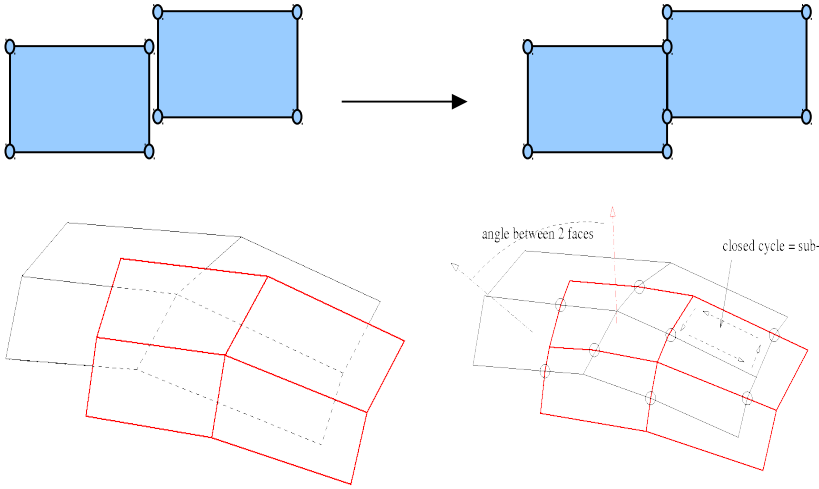






Mesh joining (2)

- 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
  - All ranks must take the same decision regarding merging a shared vertex
- 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.



High Performance Computing with *Code_Saturne*

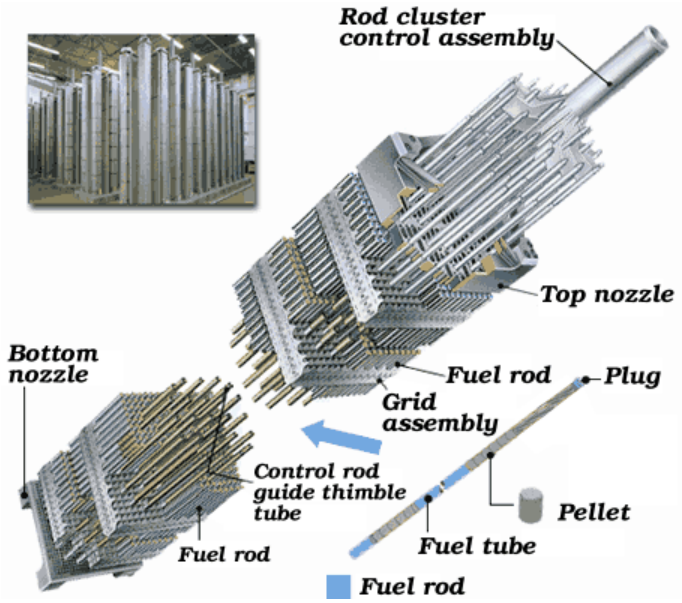
- *Code_Saturne* and NEPTUNE_CFD used extensively on HPC machines
  - EDF clusters (IBM Idataplex, Blue Gene/P; Blue Gene/Q)
  - CCRT calculation center (CEA based)
  - PRACE machines
    - Archer (EPCC), Jugene (FZJ), Curie (GENCI)
  - DOE machines (through INCITE access)
    - Jaguar (ORNL), MIRA (Argonne)

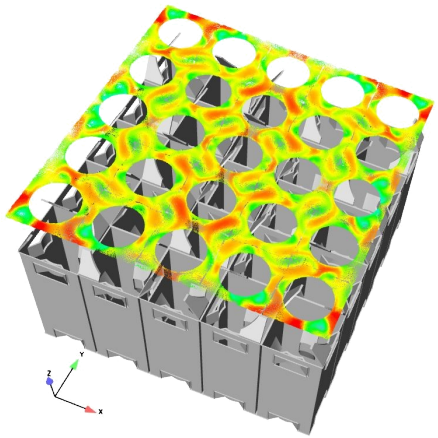
- Tests run by STFC Daresbury up to 7 billion cells on reference code
  - intensive work on parallel optimization and debugging loop
  - higher cell count reached with experimental global refinement from VSB
  - typical production studies use 10 to 50 million cells, with a few outliers in the 200-400 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

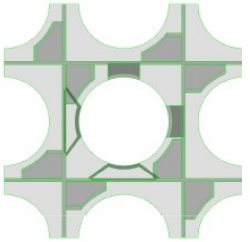
- *Code_Saturne* is one of the 12 codes selected for the PRACE and DEISA Unified European Application Benchmark Suite (UEABS)
  - along with QCD, NAMD, GROMACS, Quantum Espresso, CP2K, GPAW, ALYA, NEMO, SPECFEM3D, GENE, and GADGET
  - short list from 29 application codes in initial PRACE suite

EDF Applications: Fuel Assemblies

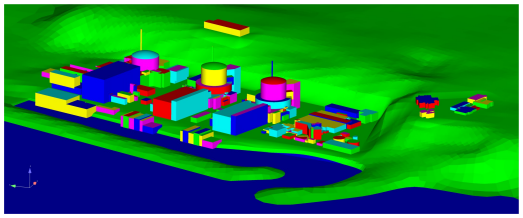
- 157 - 241 fuel assemblies in a PWR
- Each fuel assembly has 17*17 fuel rods or guide tubes and 8 to 10 grids
  - complex, non symmetrical geometry
  - Different vendors and models
- Many constraints / stakes
  - If head loss/lift too high, stronger springs needed to keep FA down, leading to possible 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



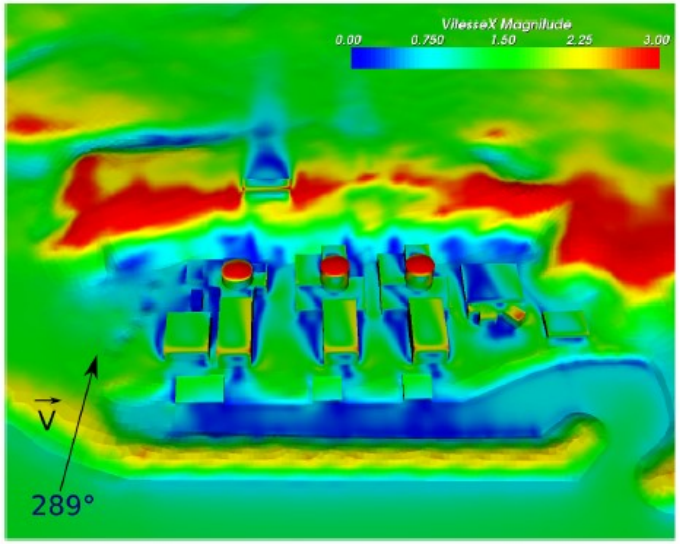


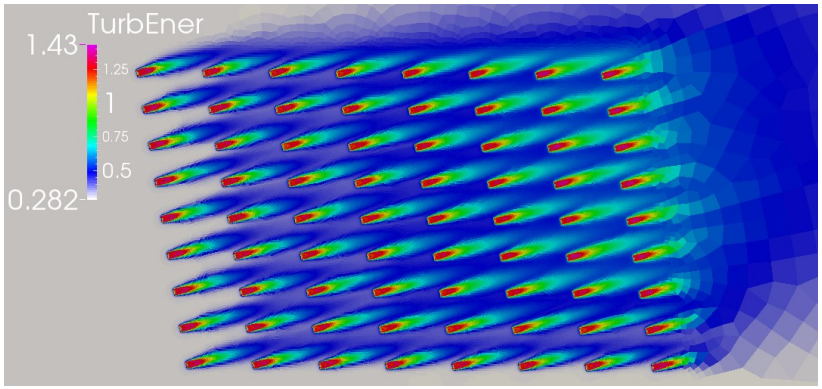


EDF Applications: meteo



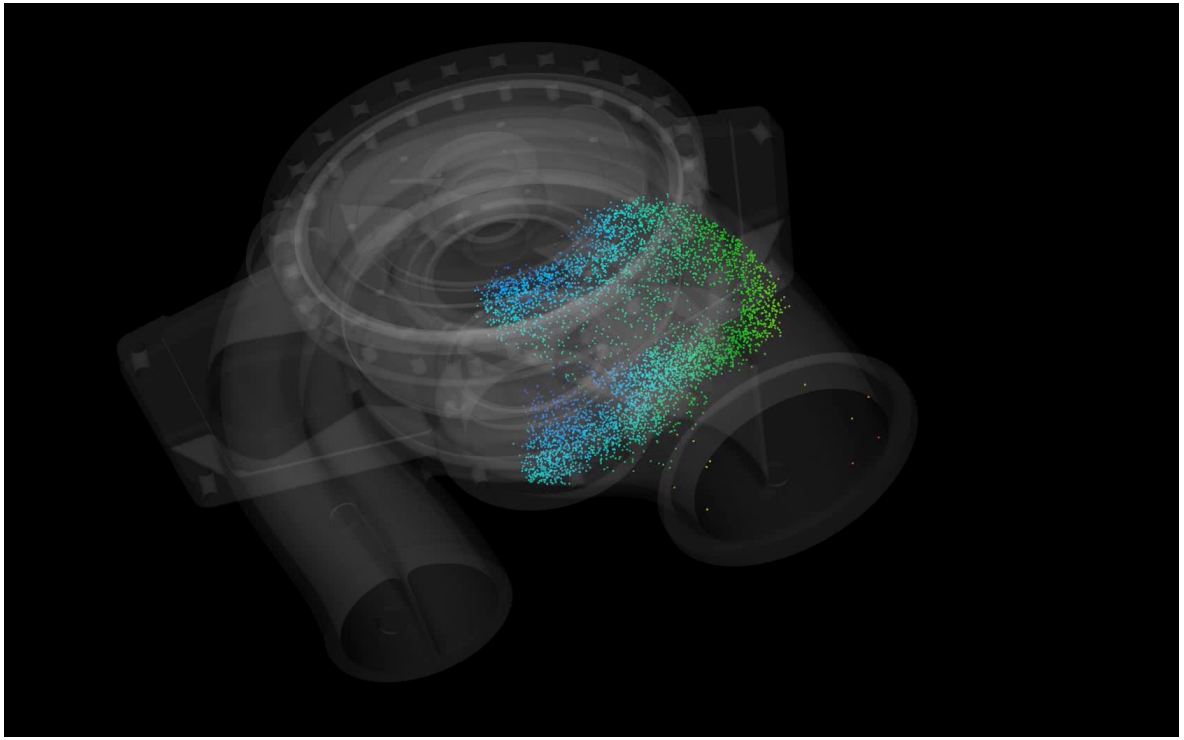
- Simulations for the atmospheric environment at the local and micro-scale (a few tens of meters to ~20km)
  - account for terrain, buildings
    - environmental impact of industrial sources, of road traffic, …
- all pollutant types: radionuclides, chemical or biological pollutants, heavy gases
- environmental 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

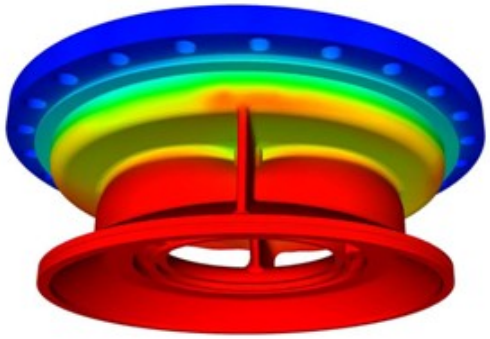


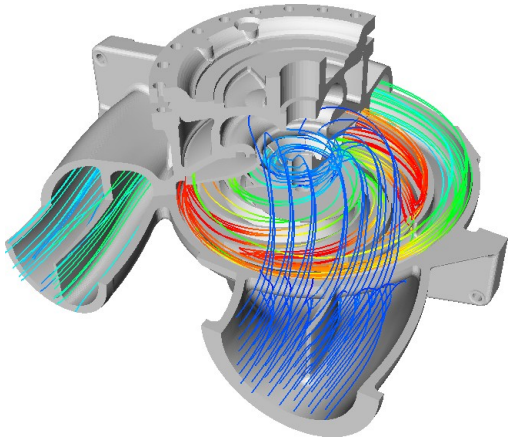


EDF Applications: CSS PUMP

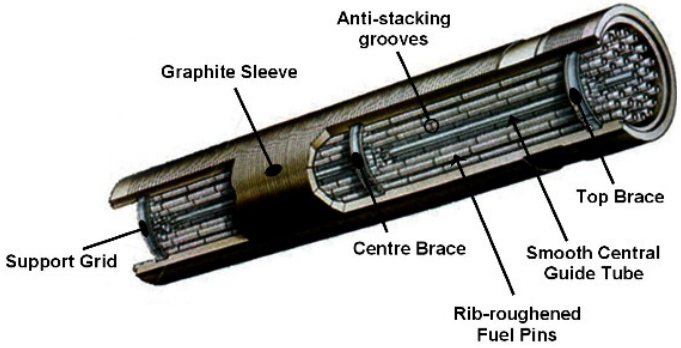
- Thermal shock
  - Large temperature gradient in the upper part of the lid
- Particle nocivity
  - Prediction of the particle distribution at the inlet of the lubrication system (in progress…)
- Cavitation model



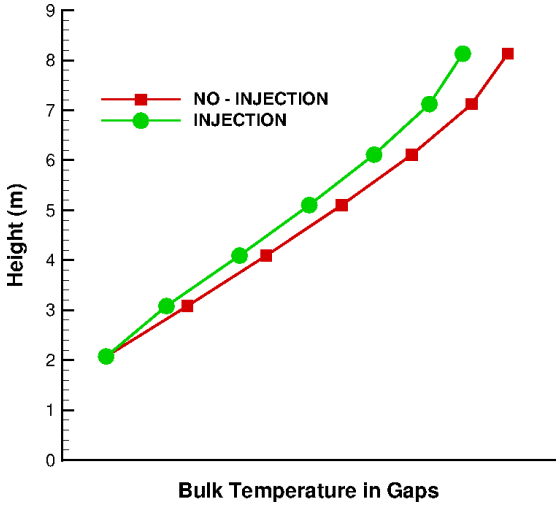


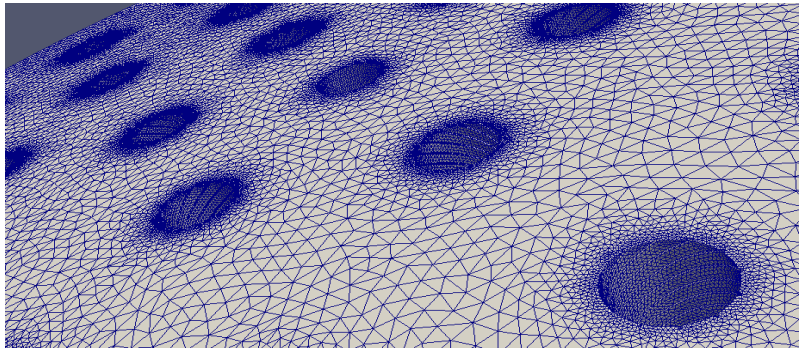


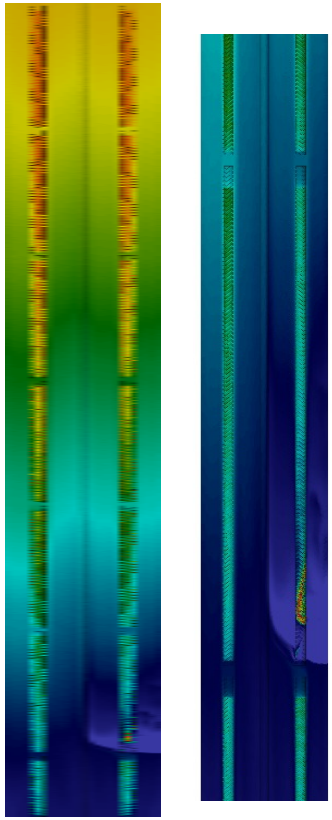
Example large scale application: Advanced Gas Reactor



- 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

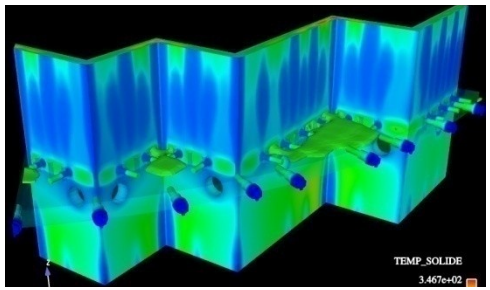




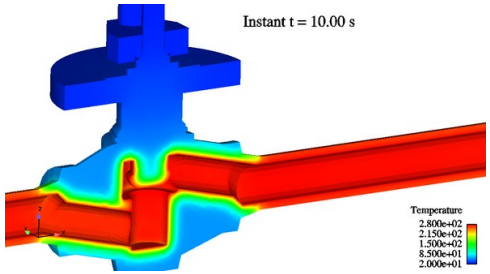


Parallel Code coupling (1)

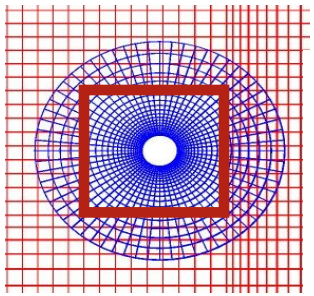
- Parallel n to p coupling using "**Parallel Location and Exchange**" sub-library
  - Uses MPI
  - Fully distributed (no master node requiring global data)
  - Successor to /refactoring of FVM (also used in Cwipi)
    - Core communication in PLE, rest moved to code
  - Also now used/tested by ALYA

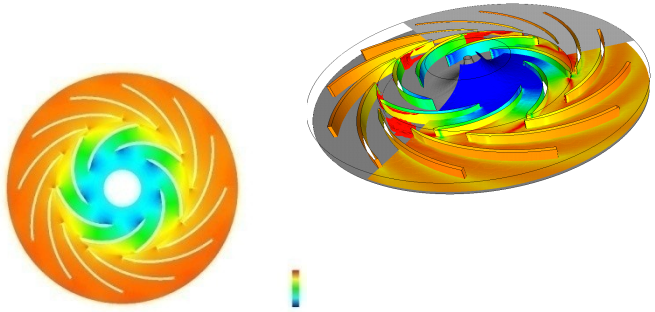


- SYRTHES (conjugate heat transfer)
  - Coupling with (parallel) SYRTHES 4 rshows good scaling using 128+ processors
  - Scaling of initial location not so good with current algorithm on 100's of ranks on BG/Q
    - need to upgrade with neighbor search from mesh joining



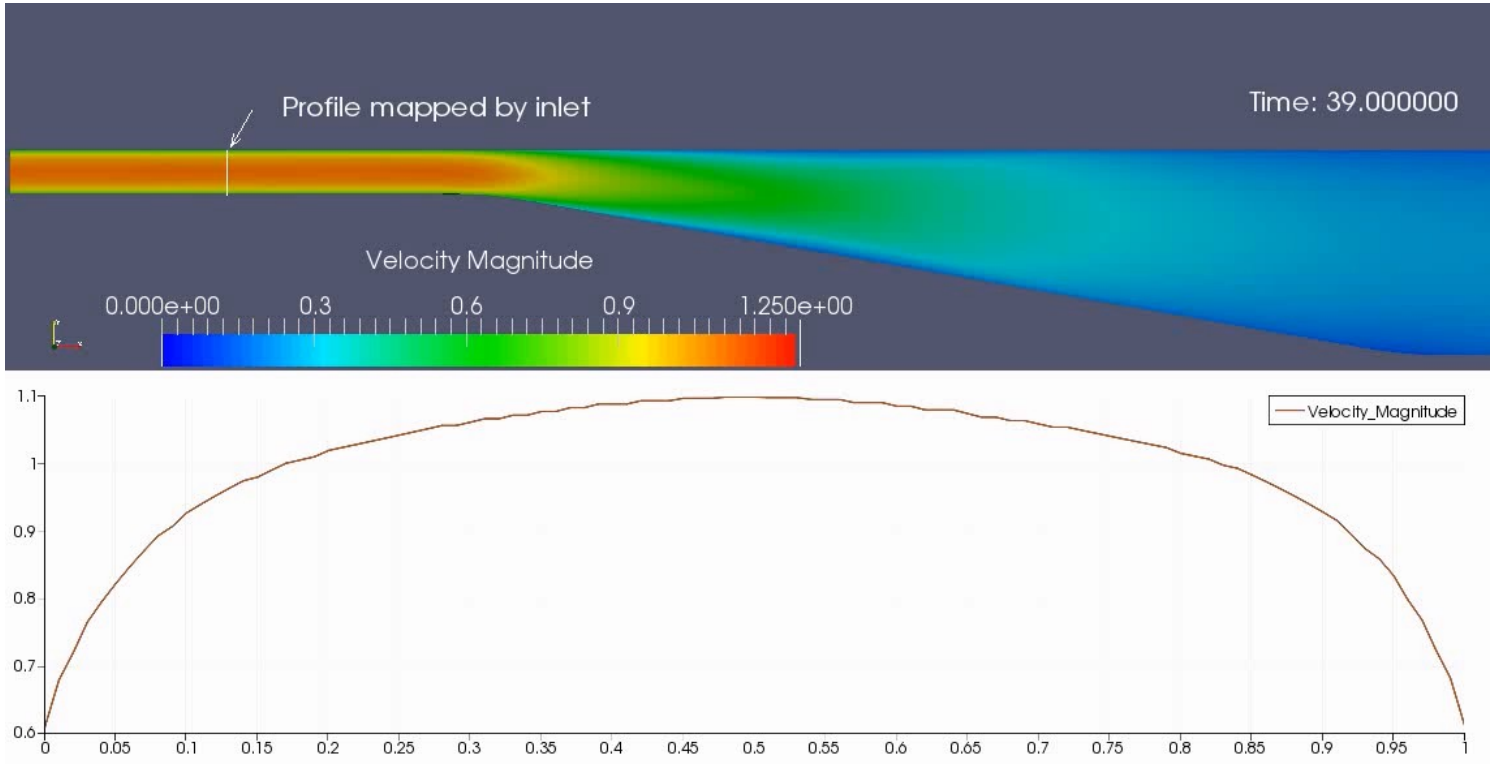
- *Code_Saturne*
  - RANS/LES
    - Different turbulence models and time steps in fixed overlapping domains
  - Turbomachinery (non-joining variant)
    - Same turbulence model
      time step, moving subdomain





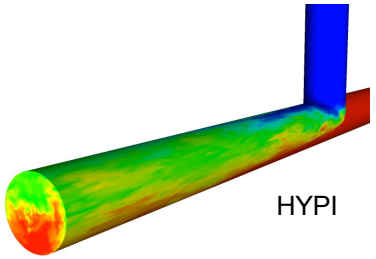
Parallel Code coupling (2)

- 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



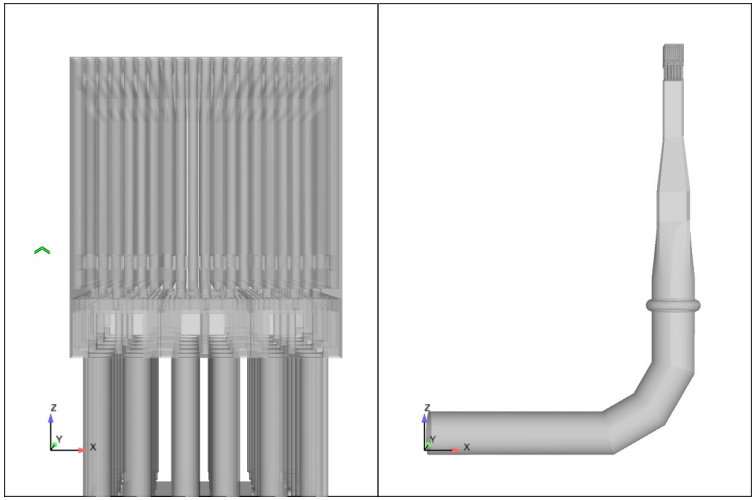
Performance benchmarks

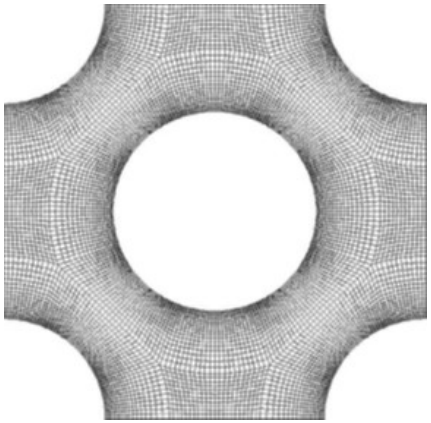
- To compare performance over several versions, we use several benchmark cases:
  - HYPI, based on a 10-million cell mesh, LES computation
  - BORA3x7, based on a 10-million cell non-conforming mesh, RANS or RSM computation
  - BUNDLE, a scalable mesh based on the experiment of Simonin and Barcuda



HYPI

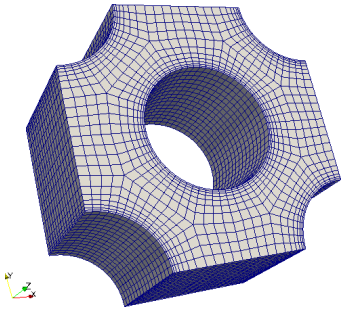
BORA3x7

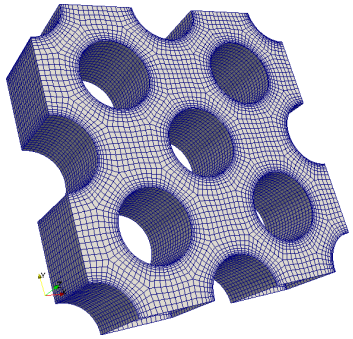


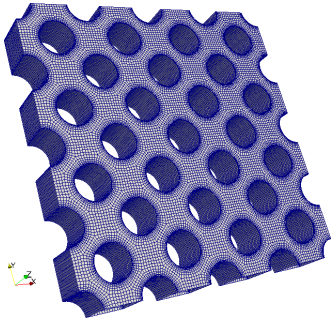


BUNDLE base stencil
(100004x128 hex)

BUNDLE
weak scaling scheme

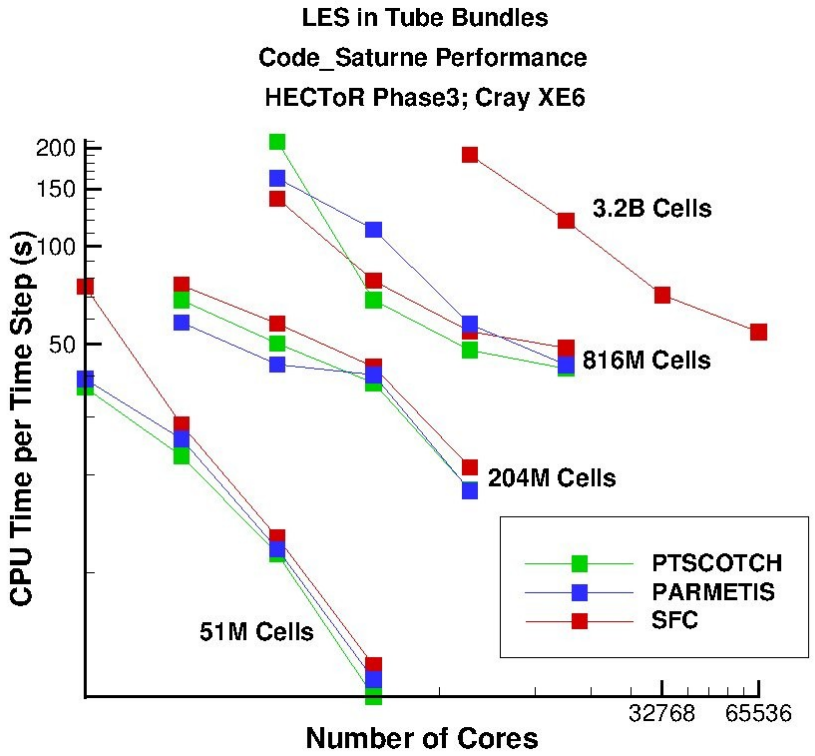


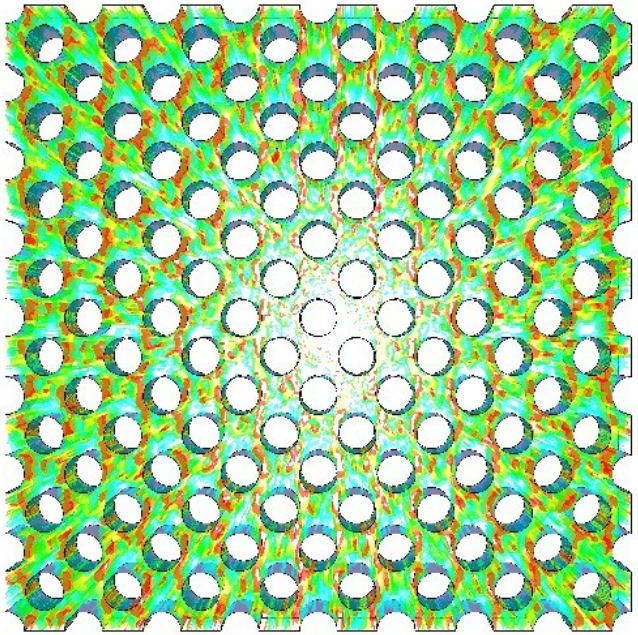




Scalability of *Code_Saturne*

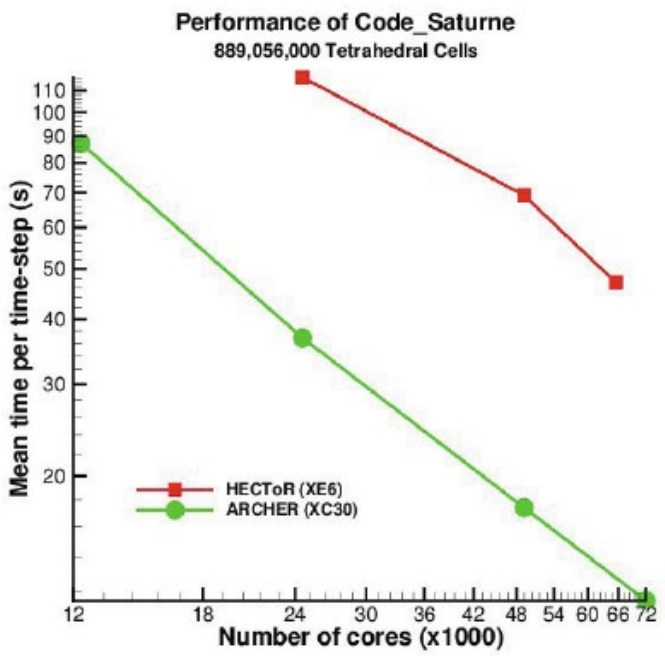
- Best scalability usually observed on Blue Gene and Cray machines
  - degrades faster on typical clusters
  - recent experience on X86/Inifiniband: do not always trust default MPI parameters…
- Scalability as a function of mesh size
  - At 65 000 cores and 3,2 Billion cells, about 50 000 cells / core

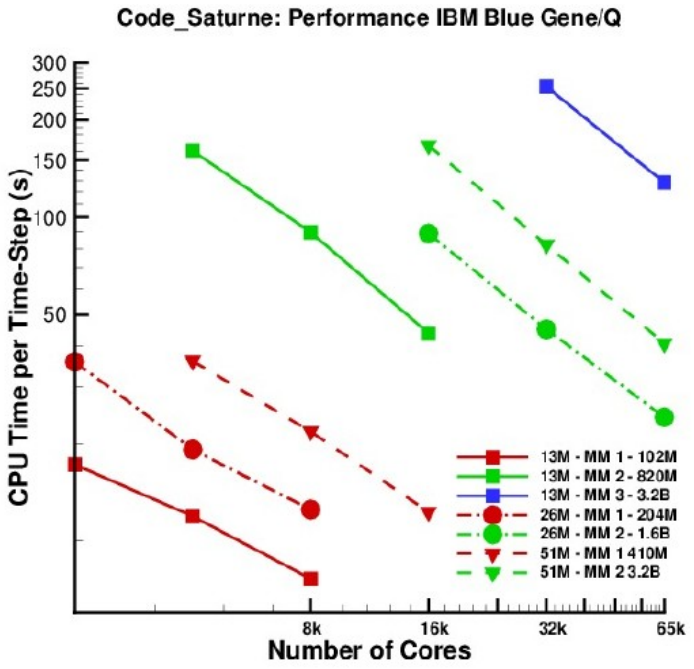




Scalability of *Code_Saturne*

- Best scalability usually observed on Blue Gene and Cray machines
- Scalability as a function of mesh size
  - At 65 000 cores and 3,2 Billion cells, about 50 000 cells / core





All to all algorithms (1)

- A new all to all API currently allows selecting from 2 algorithms (where deployed)
  - the current MPI_Alltoall / MPI_Alltoallv based algorithm
  - a Crystal router algorithm (similar to a binomial algorithm)
  - Other algorithms to be added (MPI-3 asynchronous barrier algorithm, SC14 advanced MPI tutorial)

- 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)

- 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
  - Some progress/interlocking issues also observed at high rank counts with some MPI libraries

- 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)

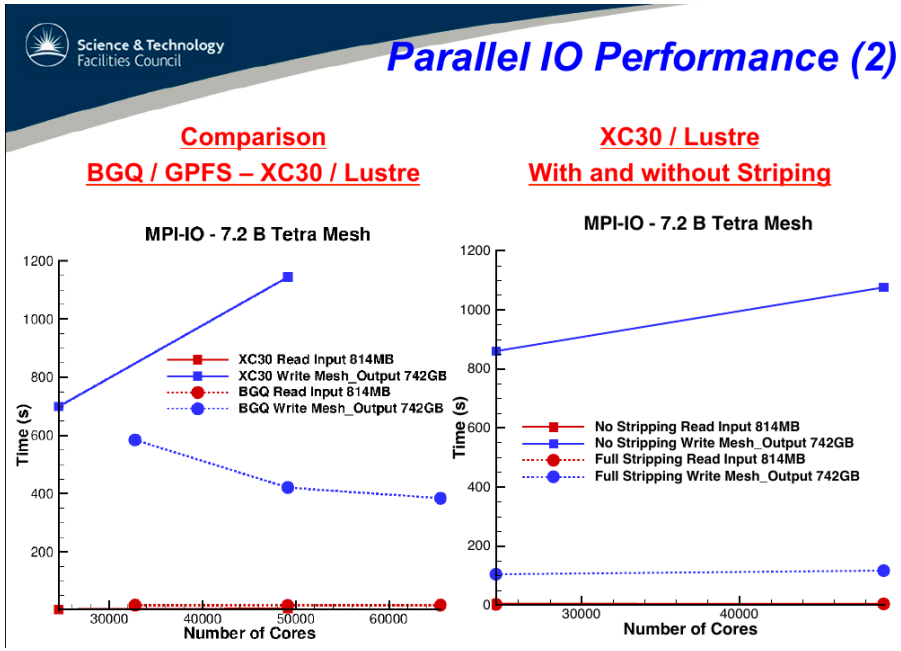
- In previous tests, we saw inverse scaling in some stages that are believed to be due to data redistribution
  - but those issues appear at much higher rank counts

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)

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



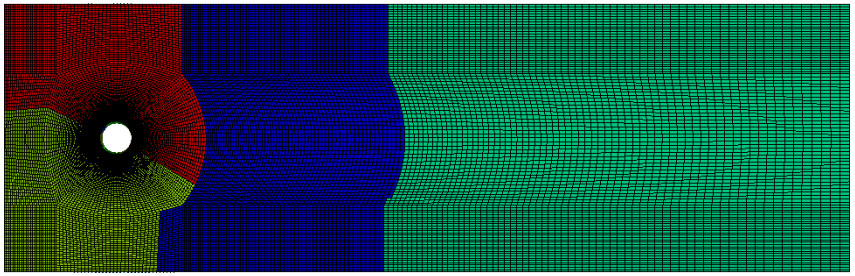
Hybrid Parallelism (1)

- 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

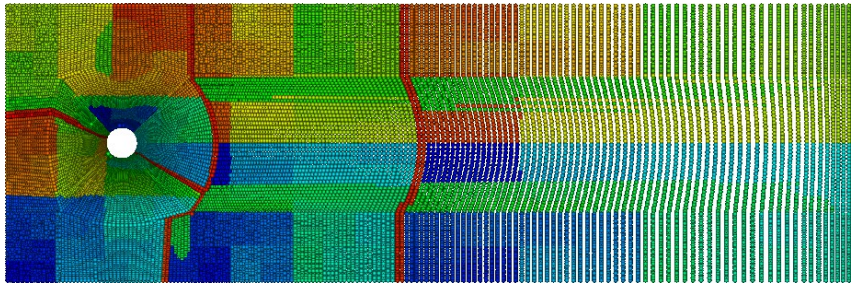
- OpenMP debugging still very painful
  - Valgrind DRD helps
    - very high overhead
    - requires compiling gcc with specific option
  - Writing C code with loop local variable definitions helps avoid missing “private” qualifiers
    - no equivalent in Fortran
    - alternative would be to use a tasking/data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

Mesh renumbering

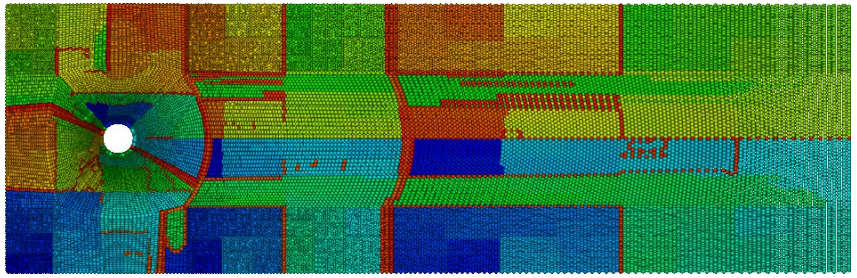
- Renumbering required to avoid thread write conflicts
  - due to the face-based nature of our loops
  - options to place halos-adjacent cells and faces last will also allow computation/communication overlap
    - MPI progress engines only recently asynchronous enough for this to be worthwhile
  - May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



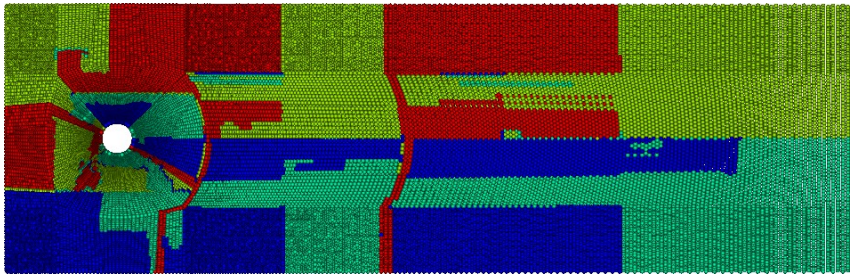
cell MPI rank id



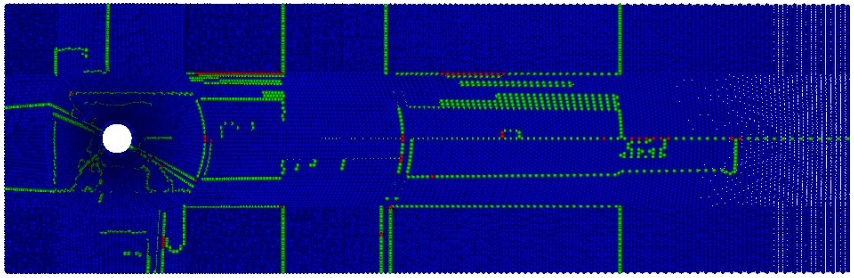
local cell id (Morton, ghost adjacent last)



local face id (Morton, ghost adjacent last)



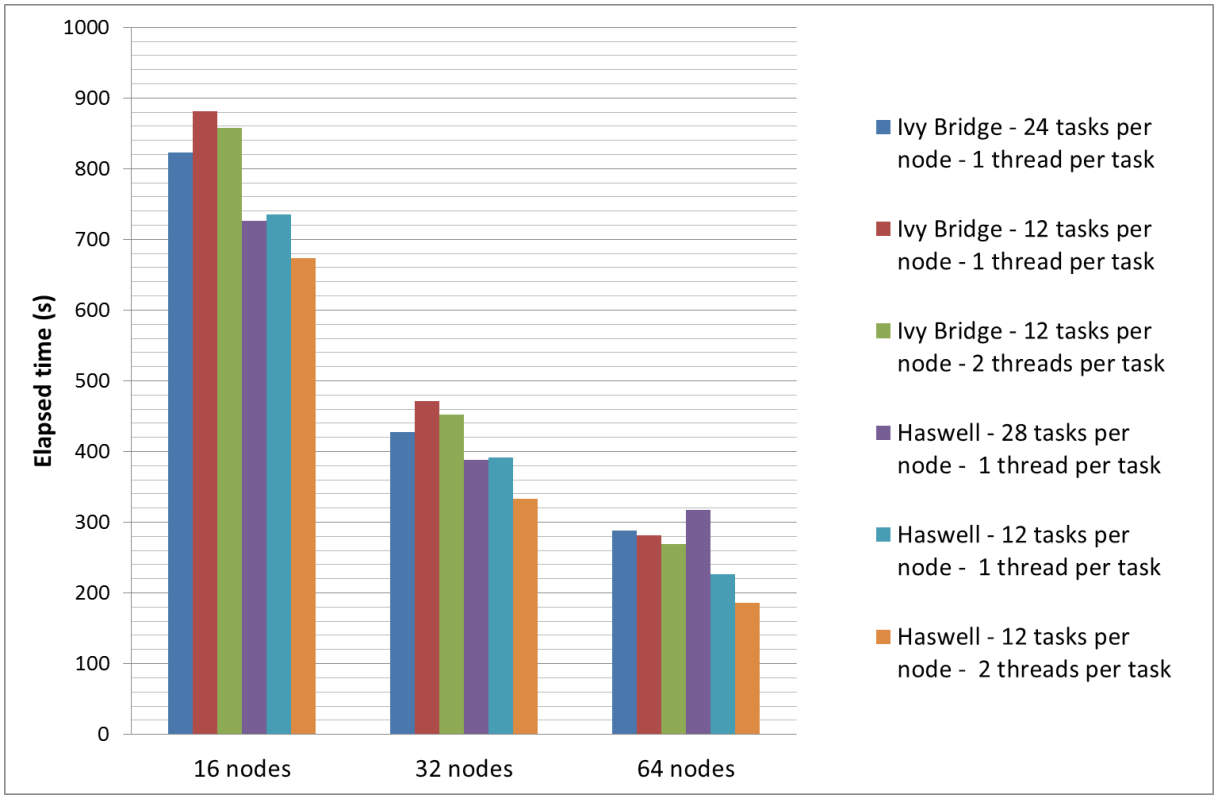
local face thead id (Morton, ghost adjacent last)



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

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



Predicting performance for PCG-type operations

- The situation is getting worse (60% improvement/year in microprocessor performance, less than 10% per year for memory access time)

Computer	Cores	Rpeak (Pflop/s)	HPL Rmax (Pflop/s)	HPCG (Pflop/s)	Peak / HPCG	HPL / HPCG
Tianhe-2 NUDT, Xeon 12C+ Xeon Phi 57C	3120000	54.90	33.84	0.58	94.66	58.34
K computer, SPARC64 VIIIfx	705024	11.28	10.51	0.46	24.48	22.81
Titan - Cray XK7 , Opteron 6274 16C, NVIDIA K20x	560640	27.11	17.59	0.32	84.12	54.58
Mira - BlueGene/Q, Power BQC 16C	786432	10.07	8.59	0.17	60.28	51.42
Trinity - Cray XC40, Xeon E5-2998 v3	301056	11.079	8.10	0.18	68.13	49.82
Stampede - Xeon E5-2680 8C + Xeon Phi SE10P	522080	8.52	5.17	0.10	88.02	53.39
ARCHER - Cray XC30, Intel Xeon E5 v2 12C	118080	2.55	1.64	0.08	31.56	20.33
Earth Simulator - NEC SX-ACE	8192		0.49	0.06	0.00	8.43
Curie thin nodes - Bullx B510, Xeon E5-2680 8C	77184	1.67	1.36	0.05	32.69	26.65
Sunway TaihuLight, Sunway MPP	10646600	125.43	93.02	0,37	337.90	250.58

External libraries

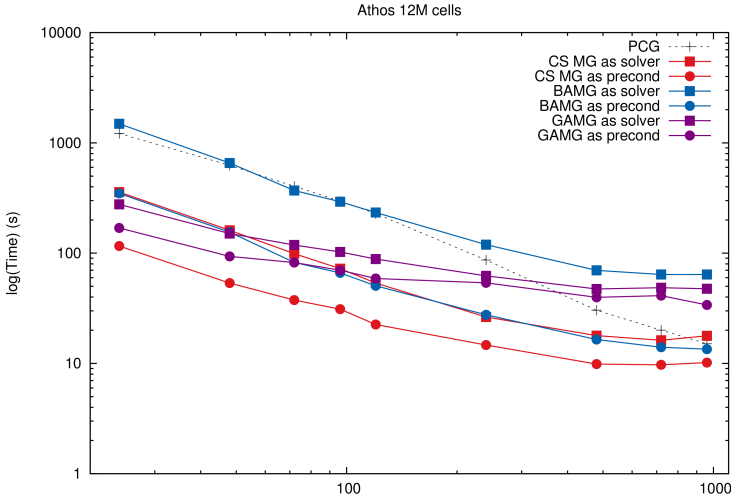
- Co-visualization and in-situ visualization now available using ParaView Catalyst
  - Easy to setup using ParaView wizard and Code_Saturne GUI
  - some bugs remain, but robustness has increased in the last year
    - mainly dependent on ParaView's progress

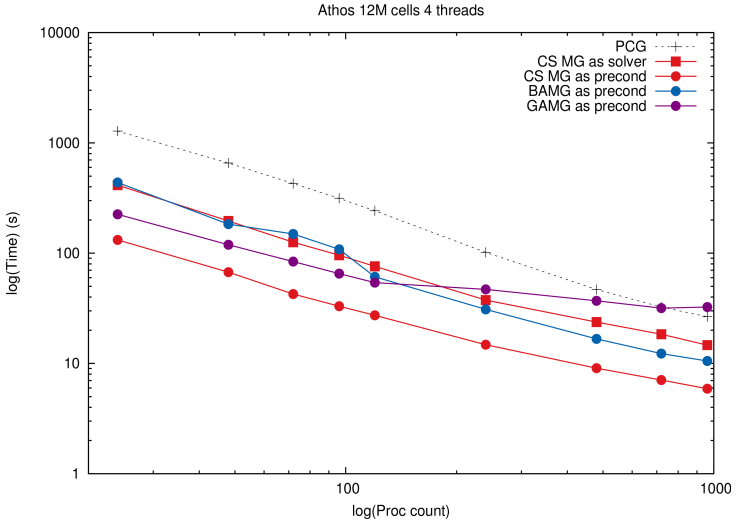
- Faster parallel IO may soon not be enough
  - Co-visualization is part of the solution
  - user subroutines also allow some simple in-situ analytics
  - we need to help users with meeting their analytics needs in situ, to avoid explosion of storage requirements
  - also need to change user habits…

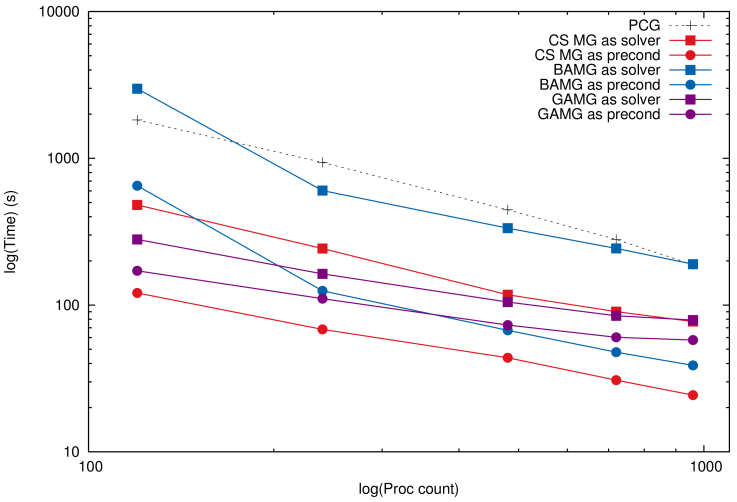
- Leverage external libraries
  - So far, the few comparisons we have done between built-in solvers and those of external libraries have shown quite good performance of built-in features, without the overhead of external library deployment and synchronization
  - With increasing hardware, software, and 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
  - PETSc integration optional since Aug 2015
    - Hypre (direct), and Lis tested spring 2015, will be added as options soon

Comparison of internal solvers to reference libraries (1)

- BUNDLE test case, comparison of multigrid on Ivy-bridge + infiniband cluster

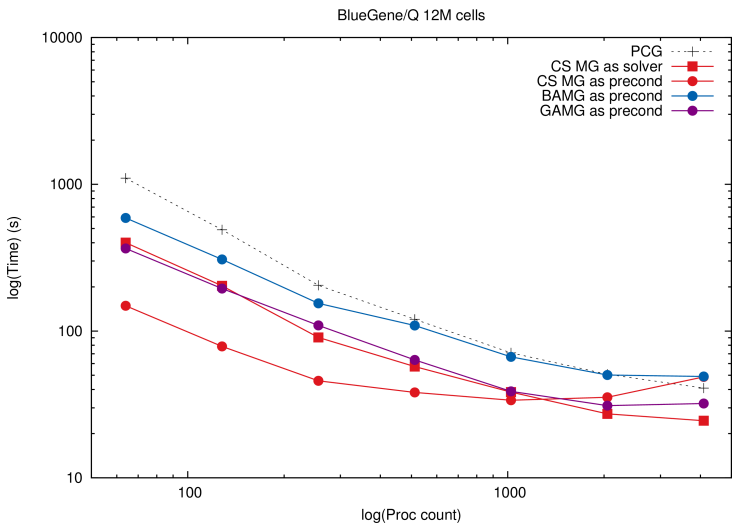


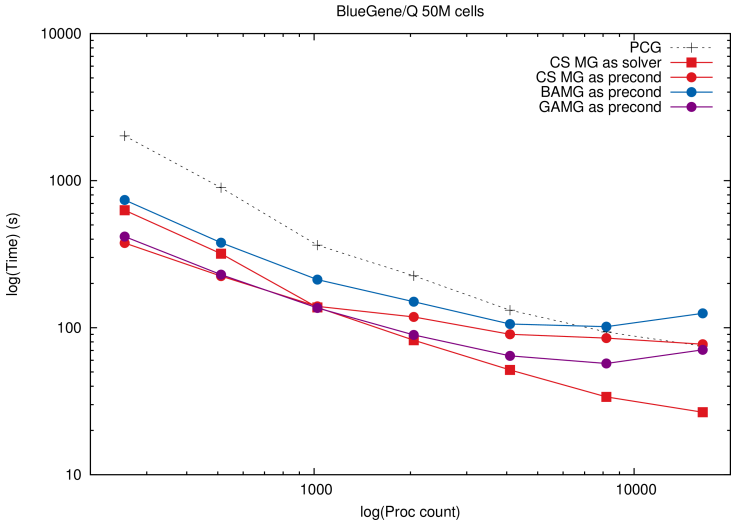


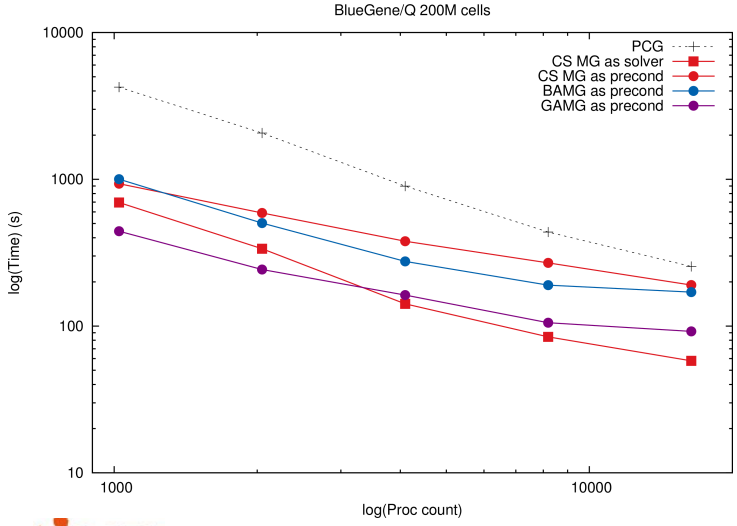


Comparison of internal solvers to reference libraries (2)

- BUNDLE test case, comparison of multigrid on Blue-Gene/Q (4 threads)

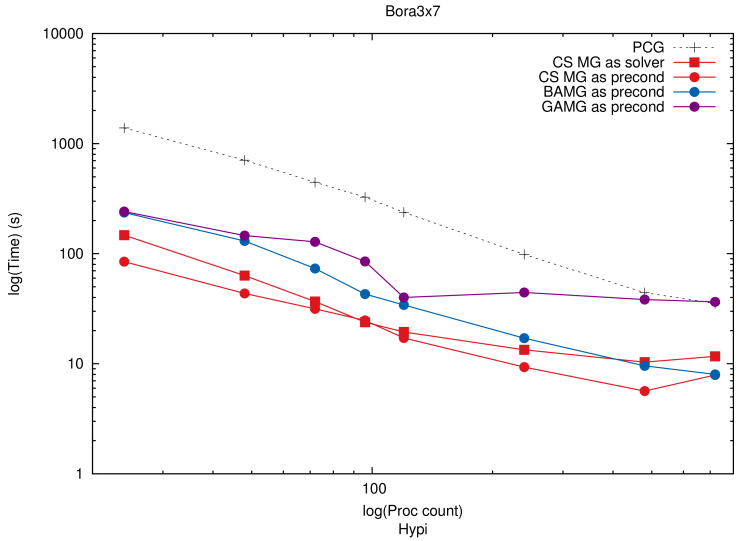


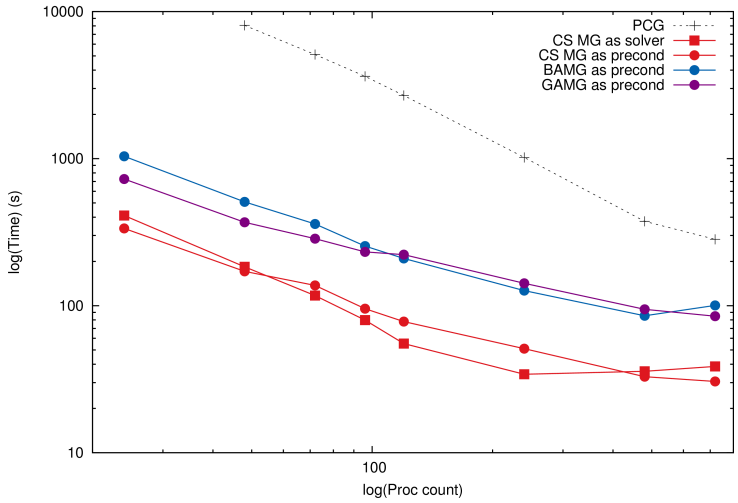




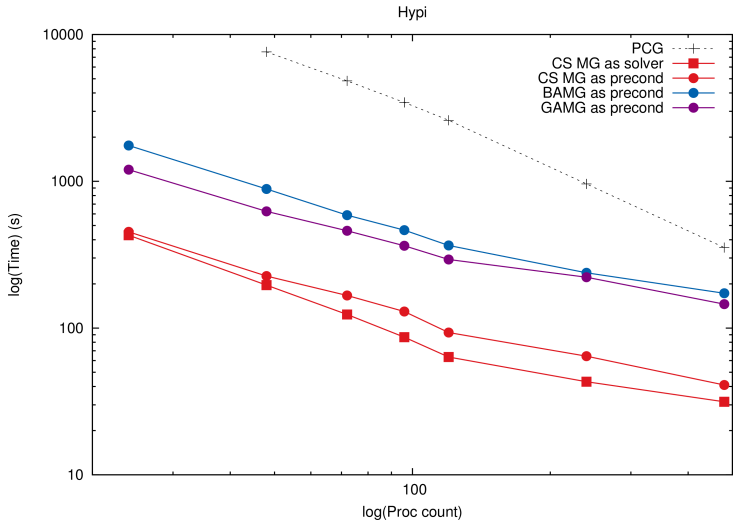
Comparison of internal solvers to reference libraries (2)

- BORA and HYPI test cases: note influence of restart (initial/established flow regime)





HYPI after restart (different flow regime)

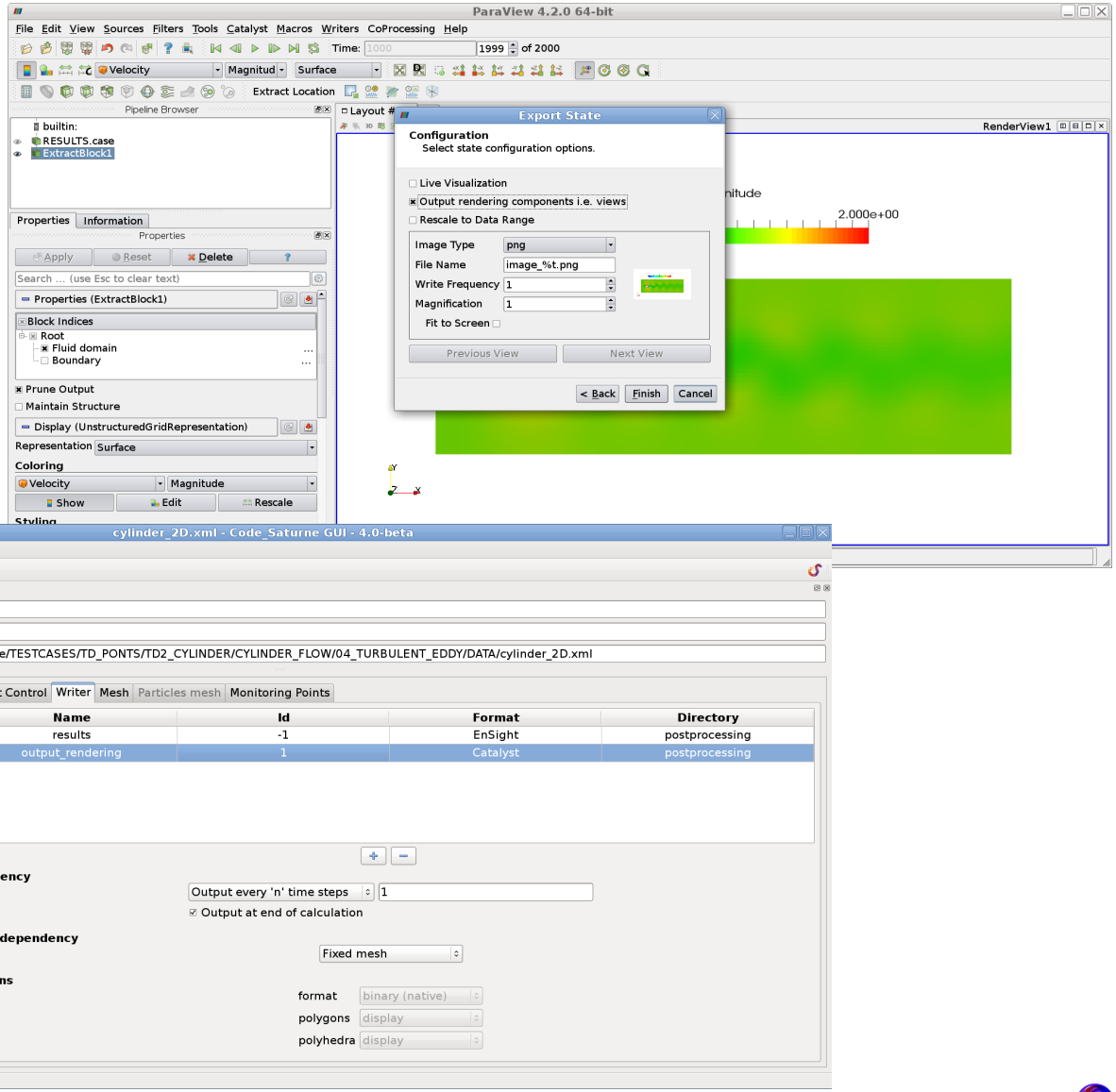


What about accelerators ?

- 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

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…
- Further work in the context of the AVIDO project (BPI)
  - EDF, Total, Kitware, UPMC-LIP6, INRIA



Next-Generation numerics in *Code_Saturne*

- Current discretization is based on a colocated cell-centered finite-volume approach (2-point schema)
  - with gradient reconstruction and explicit correction terms for mesh offsetting and non-orthogonalities
  - has served well up to now, but has some limitations
    - robustness relative to mesh quality could be improved
      - mitigation strategies are costly, or reduce precision
    - anisotropy and heterogeneity cannot be handled simultaneously

- Undergoing efforts in recent years to develop more robust methods
  - Improve the simulation of complex physics in complex geometries
    - Need a numerical scheme less sensitive to the mesh quality
    - Handle polyhedral and/or non-conforming meshes
    - Improve the physical fidelity
    - Keep an efficient solver (w.r.t memory and CPU)
  - we keep the unstructured mesh-based approach, with explicit boundaries
    - immersed-boundary methods explored in previous PHD work, not selected at this stage
    - LBM or SPH approaches tested in other EDF codes, and have strong points, but their current application domain would not completely cover *Code_Saturne* applications

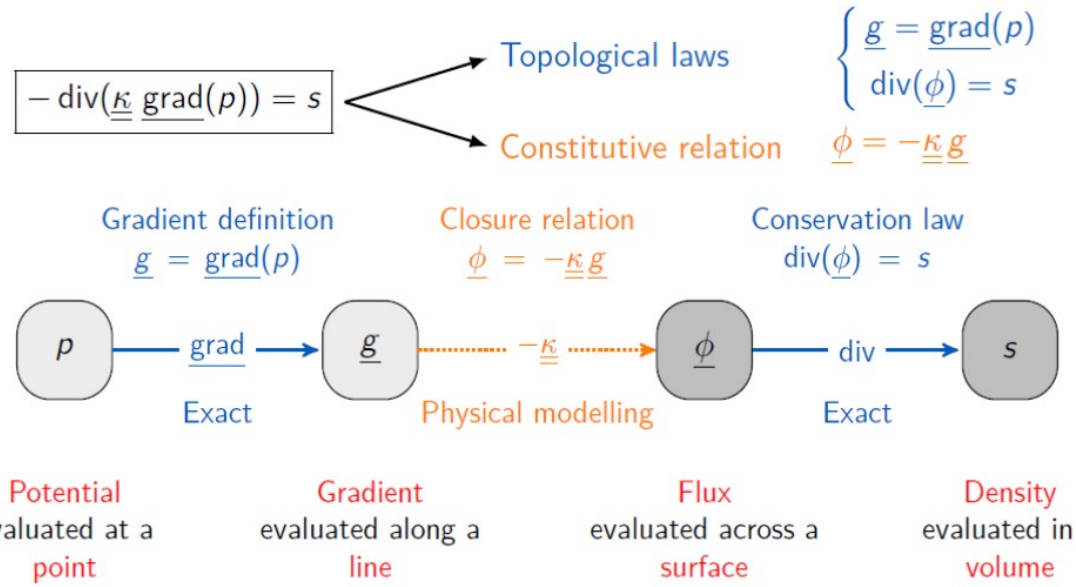
→ Development of the Compatible Discrete Operator (CDO) approach

Next-Generation numerics in *Code_Saturne*

- CDO schemes belong to compatible discretizations
  - Also called structure-preserving or mimetic discretizations
  - Aims: Preservation of the structural properties of PDEs at the discrete level
    - Satisfy exactly and locally conservation laws
    - Preserve properties of differential operators (adjunction, kernel. . .)
  - Pioneering works
    - Electromagnetics: Kron ’53 , Branin ’66 , Tonti ’74 and Bossavit ’88
    - Mathematics: Whitney ’57 and Dodziuk ’76
  - A deeper understanding of the links between mathematics/physics/geometry
    - Identify analogies between vector calculus, differential geometry and algebraic topology
    - Consider four types of fields according to their physical nature:
      – potential, circulation, flux and density fields
    - Make the distinction between
      – Topological laws: conservation laws and definitions of differential operators
      – Metric relations: closure or constitutive relations

CDO discretization process

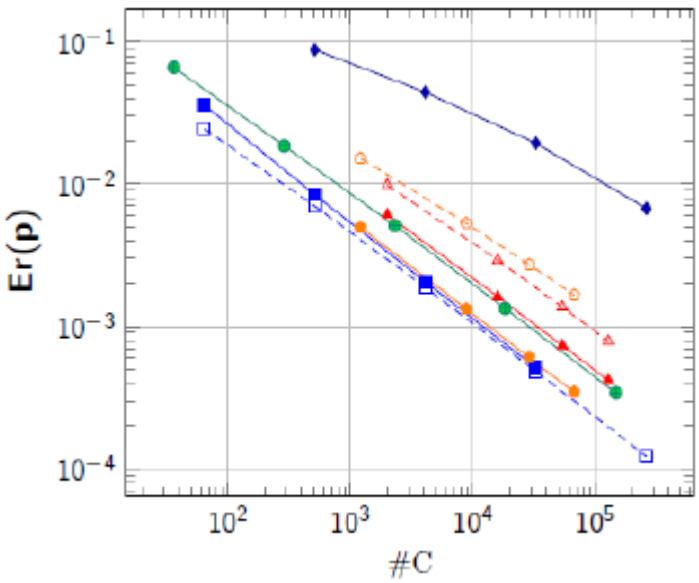
- several families of CDO schemes depending on where potential DoFs are located
  - vertex, edge, face, or cell-based schemes

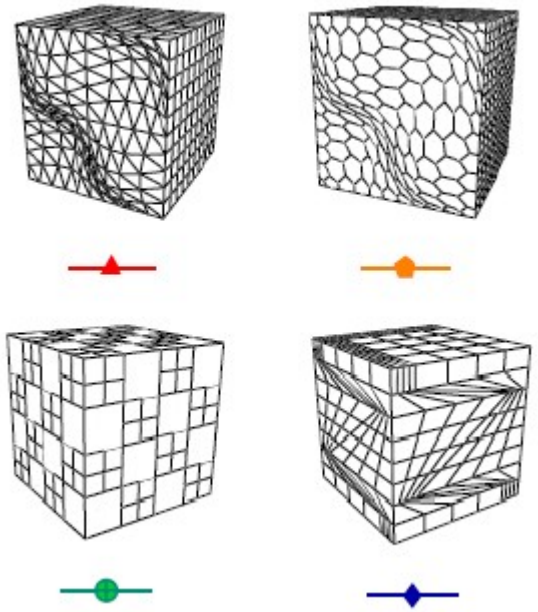


- For more
  - J. Bone Stokes equations" . PhD thesis, Universitée Paris-Est, (2014).
  - J. Bonelle and A. Ern. "Analysis of Compatible Discrete Operator Schemes for Elliptic Problems on Polyhedral Meshes."ESAIM: M2AN, 48:553–581 (2014)
  - J. Bonelle, E. Burman, P. Cantin and A. Ern "A vertex-based scheme on polyhedral meshes for advection-reaction equations with sub-mesh stabilization", submitted

First applications of the CDO approach

- Effective method on polyhedral mesh for advection-diffusion equations
  - application to hydrogeological flow test cases





- Encouraging results with CDO on hydrogeological flows
  - able to handle high heterogeneity and anisotropy
  - robust relative to non-conforming meshes

- Work in progress;
  - MPI Parallelization by end of 2016
  - Extension to Navier-Stokes over next few years

Parallel meshing / mesh modification

- Meshing is missing from most of the work presented
  - So far, parallel joining has enabled working around mesh size limitations
- Generating boundary layers inside the code to be added fall 2015
  - Doing it inside the code may *help* adapting the boundary layer to the turbulent model
    - But adaptive models are still essential
- Some open-source parallel meshers seem to be appearing
  - Polytope (https://bitbucket.org/jjphatt/polytope)
    - Open source, parallel layer to other serail open-source meshers
  - PMSH (https://code.google.com/p/pmsh/) tetra
- Favor open-source + library options
  - May go all the way up to linking inside code rather than adding IO

Mesh multiplication

- 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
- Work in progress
  - could allow starting from coarser meshes
    - great for mesh sensitivity studies
  - has already allowed weak scalability testing at a higher scale

Two elemental patterns: 26M → 13B				
no. mpi tasks	ref. levels	mesh refinement (s)	avg. time / step (s)	total computation (s)
524288	3	8.4	70	1032
1048576	3	15.4	50	1095
1572864	3	22.6	43	1398

Highest thread count attained (MIRA)

Highest mesh size attained (MIRA)

Two elemental patterns: 26M → 105B				
no. mpi tasks	ref. levels	mesh refinement (s)	avg. time / step (s)	total computation (s)
262144	4	8.39	701	3797
524288	4	8.92	376	2273

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

- 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 (in-situ/post-mortem hybrid)

- Don't forget code/resource manager integration
  - essential to ease of use
  - as resource managers and front-end/compute node differences become more complex, may need to define sub-steps
    - more of an issue on clusters where nodes are similar (with different environments) than on BG/Q

- Heuristics
  - How much ?

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

THANK YOU FOR YOUR ATTENTION

any questions ?