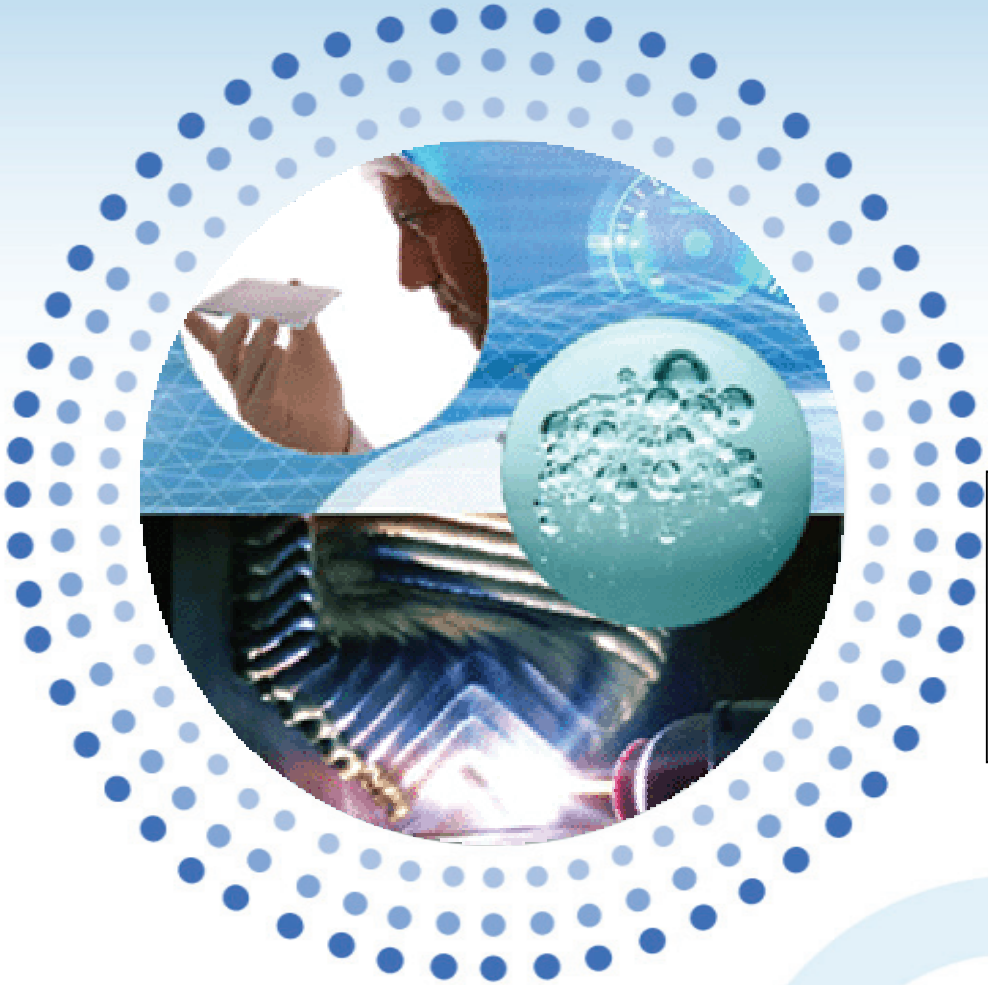


High Performance Calculation with *Code_Saturne* at EDF

Toolchain evoution and roadmap

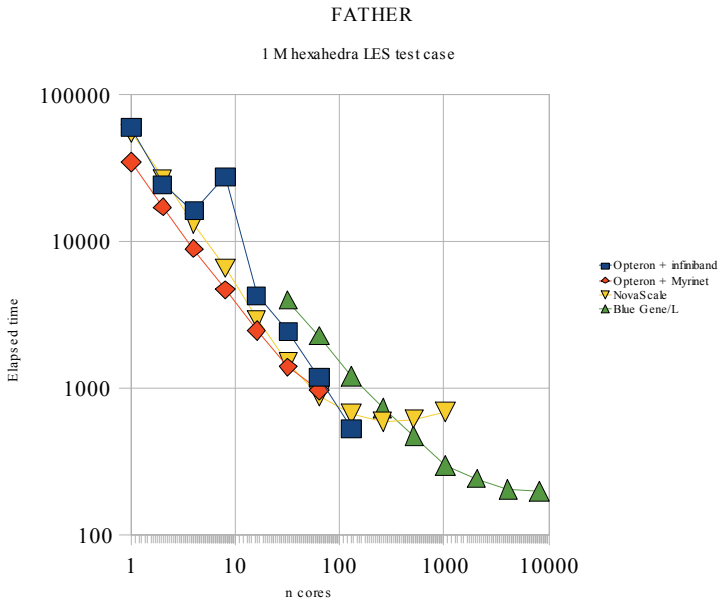
EDF R&D

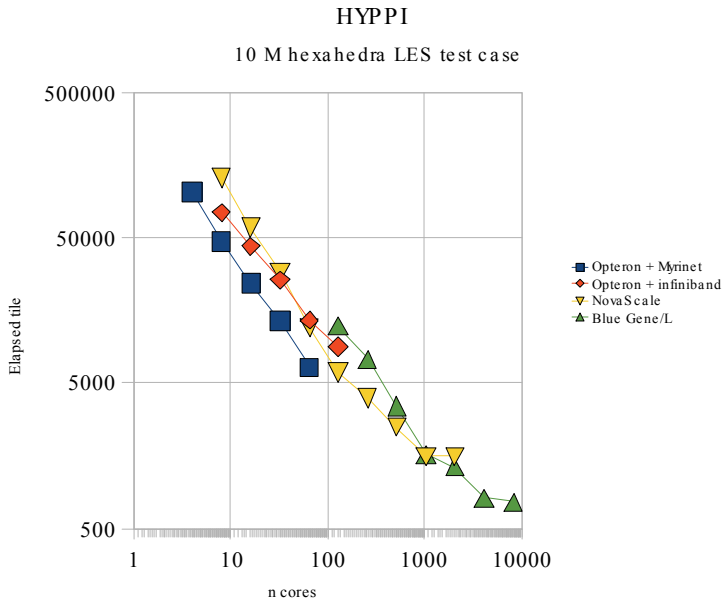
Code_Saturne Features of note to HPC

- Segregated solver
  - All variables are solved or independently, coupling terms are explicit
    - Diagonal-preconditioned CG used for pressure equation, Jacobi (or bi-CGstab) used for other variables
  - More important, matrices have no block structure, and are very sparse
    - Typically 7 non-zeroes per row for hexahedra, 5 for tetrahedra
    - Indirect addressing + no dense blocs means less opportunities for MatVec optimization, as memory bandwidth is as important as peak flops.
- Linear equation solvers usually 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

Current performance (1/2)

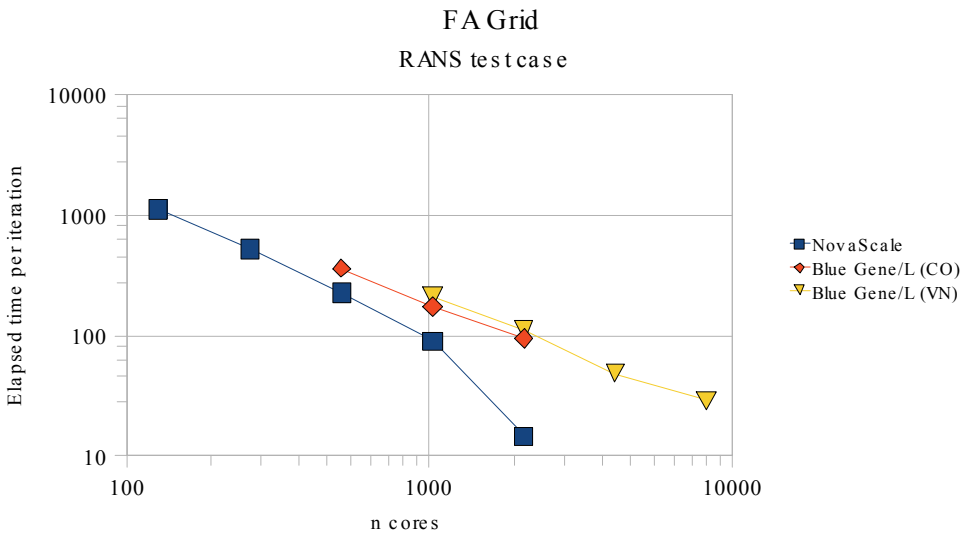
- 2 LES test cases (most I/O factored out)
  - 1 M cells: (n_cells_min + n_cells_max)/2 = 880 at 1024 cores, 109 at 8192
  - 10 M cells (n_cells_min + n_cells_max)/2 = 9345 at 1024 cores, 1150 at 8192

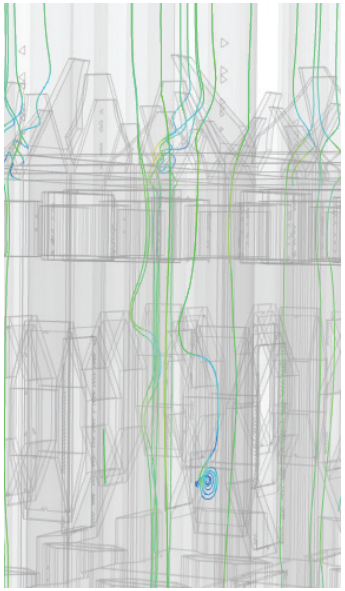




Current performance (2/2)

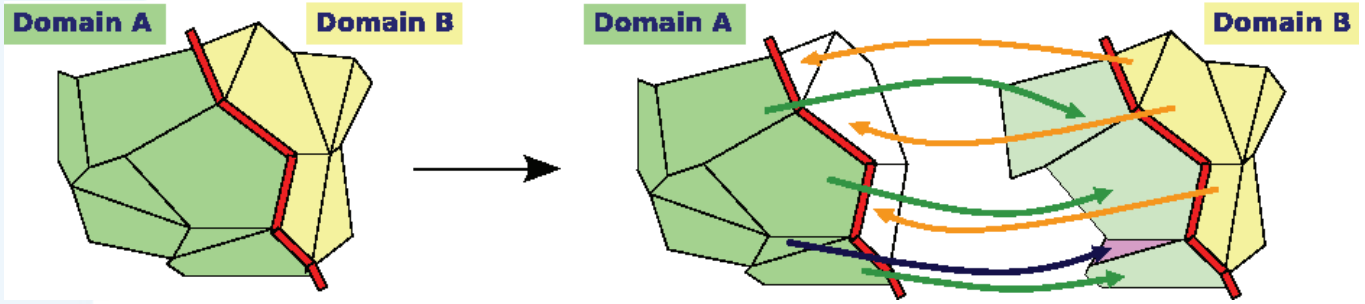
- RANS, 100 M tetrahedra + polyhedra (most I/O factored out)
  - Polyhedra due to mesh joinings may lead to higher load imbalance in local MatVec for large core counts
    - 96286/102242 min/max cells/core at 1024 cores
    - 11344/12781 min/max cells cores at 8192 cores





Base parallel operations (1/4)

- Distributed memory parallelism using domain paritioning
  - Use classical “ghost cell” method for both parallelism and periodicity
    - Most operations require only ghost cells sharing faces
    - Extended neighborhoods for gradients also require ghost cells sharing vertices



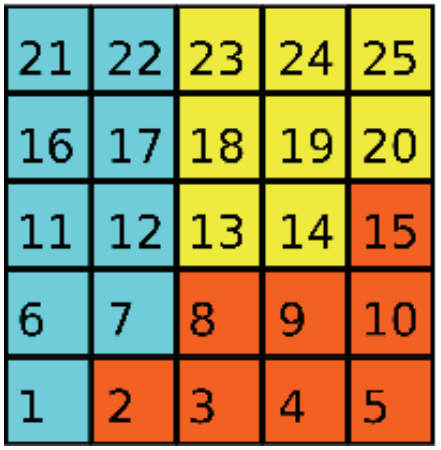
    - Global reductions (dot products) are also used, especially by the preconditioned conjugate gradient algorithm

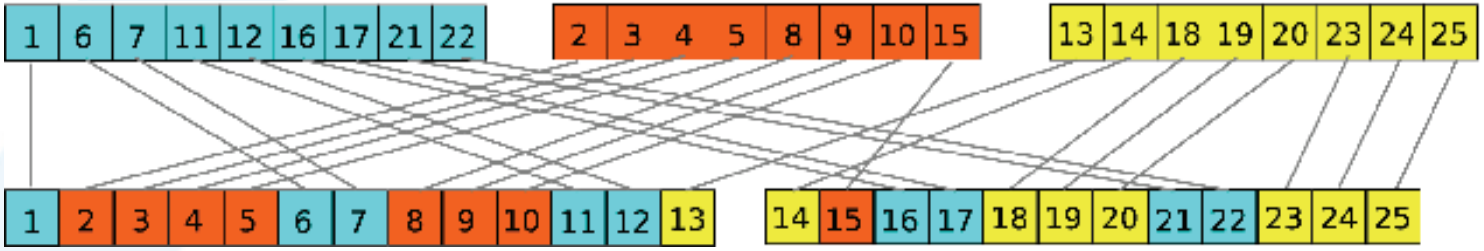
Base parallel operations (2/4)

- Use of global numbering
  - We associate a global number to each mesh entity
    - A specific C type (`fvm_gnum_t`) is used for this. Currently an unsigned integer (usually 32-bit), but an unsigned long integer (64-bit) will be necessary
      - Face-cell connectivity for hexahedral cells : size 4.n_faces, and n_faces about 3.n_cells, → size around 12.n_cells, so numbers requiring 64 bit around 350 million cells.
    - Currently equal to the initial (pre-partitioning) number
  - Allows for partition-independent single-image files
    - Essential for restart files, also used for postprocessor output
    - Also used for legacy coupling where matches can be saved

Base parallel operations (3/4)

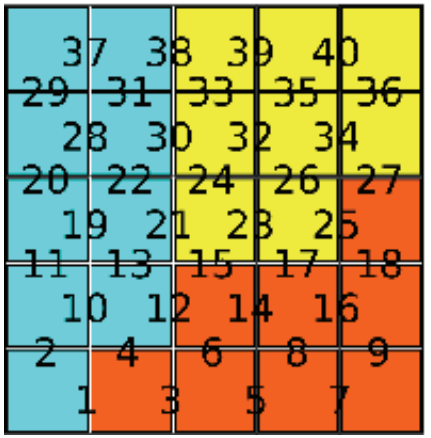
- Use of global numbering
  - Redistribution on n blocks
    - n blocks ≤ n cores
    - Minimum block size may be set to avoid many small blocks (for some communication or usage schemes), or to force 1 block (for I/O with non-parallel libraries)

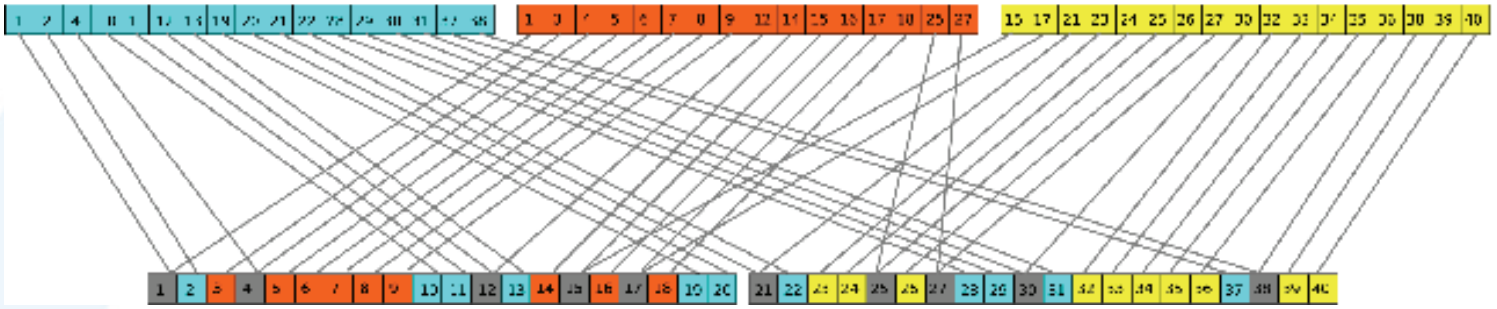




Base parallel operations (4/4)

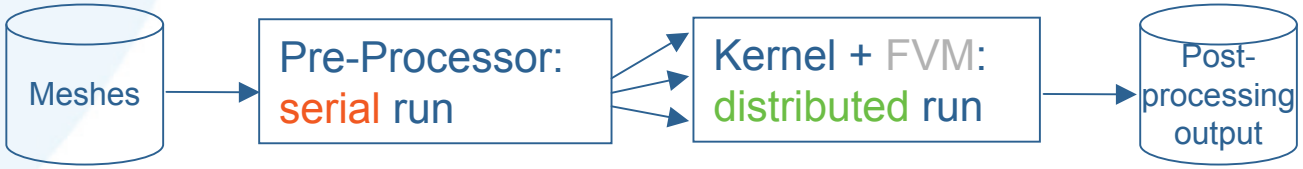
- Conversely, simply using global numbers allows reconstructing neighbor partition entity equivalents mapping
  - Used for parallel ghost cell construction from initially partitioned mesh with no ghost data
- Arbitrary distribution, inefficient for halo exchange, but allow for simpler data structure related algorithms with deterministic performance bounds
  - Owning processor determined simply by global number, messages are aggregated





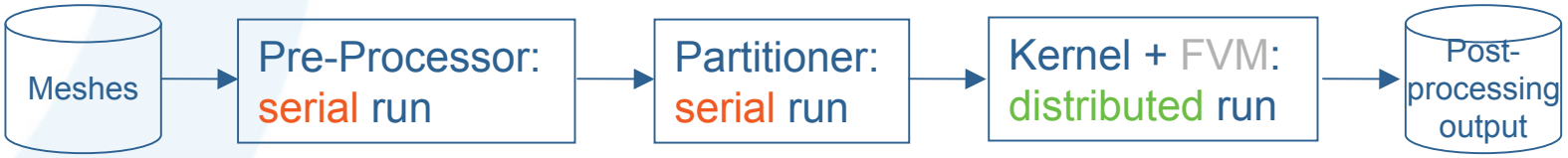
Toolchain evolution (1)

- Code_Saturne V1.3 added many HPC-oriented improvements compared to prior versions:
  - Post-processor output handled by FVM / Kernel
  - Ghost cell construction handled by FVM / Kernel
    - Up to 40% gain in preprocessor memory peak compared to V1.2
    - Parallelized and scales, though complex, as we may have 2 ghost cell sets and multiple periodicities
  - Well adapted for models up to 100 million cells (with 64 Gb preprocessing machine), or less
    - All fundamental limitations are pre-processing related



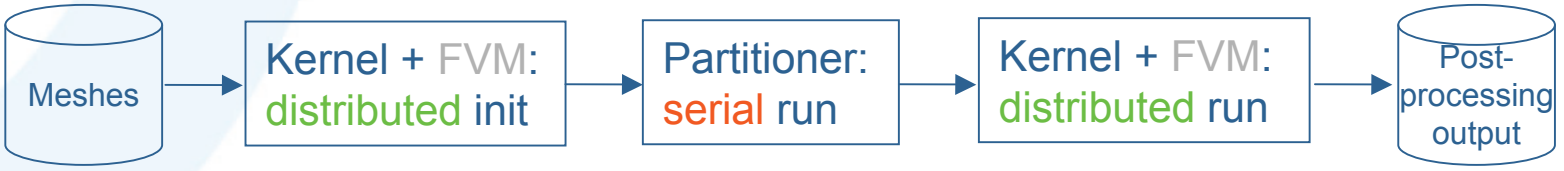
Toolchain evolution (2)

- Separate basic Pre-Processing from Partitioning in version 1.4
  - Allows running both stages on different machines
  - Allows using the same pre-processing run for different partitionings
    - Joining of non-conforming meshes usually requires only a few minutes under 10 million cells, but can be close to a day for for 50 million+ cell meshes built of many sub-parts
    - Partitioning requires slightly less memory than pre-processing



Toolchain evolution (3)

- Parallelize mesh joining for V2.0
  - Joining of conforming/non-conforming meshes, periodicity
    - Parallel edge intersection implemented, not merged yet
    - Sub-face reconstruction in progress
    - Current (serial) algorithm will probably remain available as a backup option for V2.0 (and be removed no sooner than 6 months after full parallel algorithm)
  - Allow restart in case of memory fragmentation due to complex mesh joining step



Toolchain evolution (4)

- Parallelize domain partitioning for V2.0
  - Replace serial METIS or SCOTCH with parallel versions
    - SCOTCH uses free CeCILL-C licence, whereas METIS is more restrictive, but SCOTCH does not yet seem as stable
      - Possibly due to METIS-wrapper, though feedback from *Code_Aster* seems to indicate the same
    - ParMetis partitioning reportedly of lower quality than serial METIS
      - Is it that bad ?
    - PT-SCOTCH does not suffer from this, but functionality is not complete yet

Parallelization of partitioning

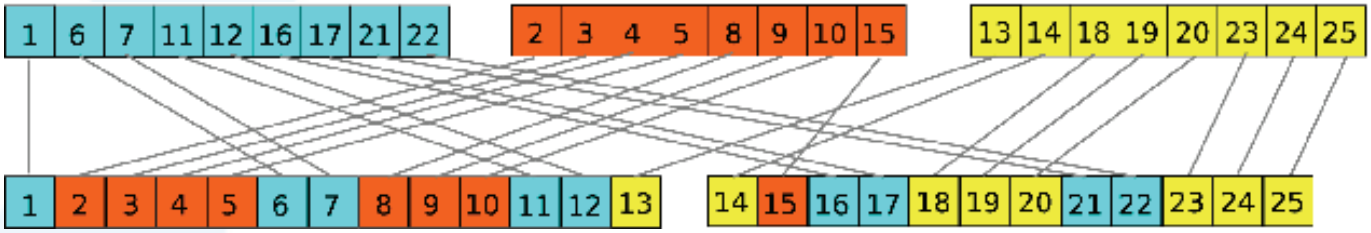
- Version 1.4 already prepared for parallel mesh partitioning
  - Mesh read by blocks in « canonical / global » numbering, redistributed using cell domain number mapping
  - All that is required is to plug a parallel mesh partitioning algorithm, to obtain an alternative cell → domain mapping
    - The redistribution infrastructure is already in place, and already being used
- Possible choice: use SANDIA's ZOLTAN library
  - May call METIS, SCOTCH, it's own high-quality hypergraph partitioning, recursive bisection or space-filling curves (Z-curve or Hilbert), LGPL license

Parallel I/O (1/2)

- Version 1.4 introduces parallel I/O
  - Uses block to partition redistribution when reading, partition to block when writing
    - Use of indexed datatypes may be tested in the future, but will not be possible everywhere
  - Fully implemented for reading of preprocessor and partitioner output, as well as for restart files
  - Infrastructure in progress for postprocessor output
    - Layered approach as we allow for multiple formats

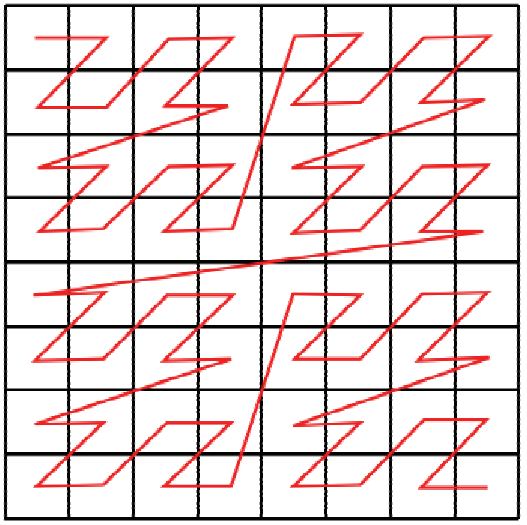
Parallel I/O (2/2)

- Parallel I/O only of benefit when using parallel filesystems
  - Use of MPI IO may be disabled either when building the FVM library, and for a given file using specific hints
  - Without MPI IO, data for each block is written or read successively by rank 0
    - Using the same FVM file functions



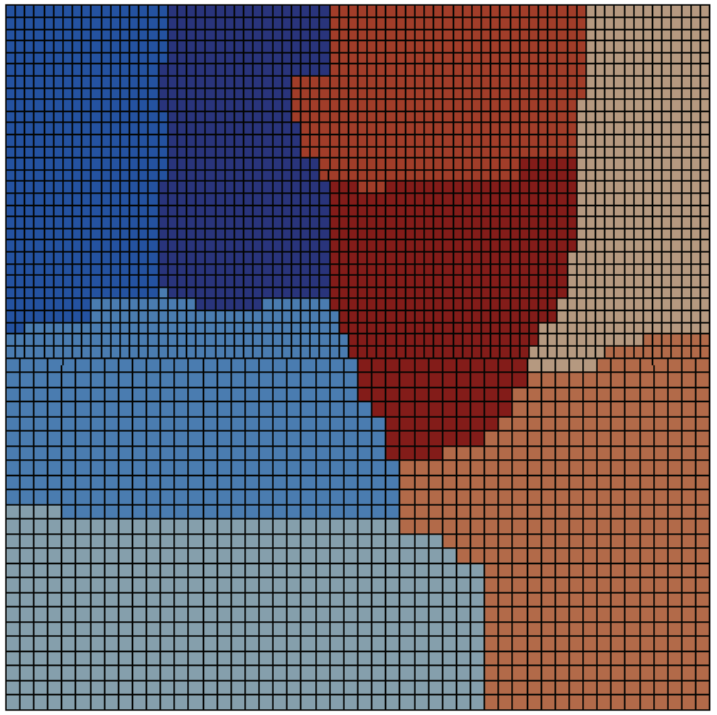
Possible future changes

- Base global (canonical) numbering not on initial number, but on space-filling curve
  - A domain partitioning method in its own right
    - Independent of processor count
  - Ensures block to partition mapping is « sparse » (i.e. only a few blocks are references by a partition, and vice-versa)
    - Block to partition mapping is 1-to-1 if no additional partitioning method is used



Load imbalance (1/3)

- In this example, using 8 partitions (with METIS), we have the following local minima and maxima:
  - Cells:
    416 / 440 (6% imbalance)
  - Cells + ghost cells:
    469/519 (11% imbalance)
  - Interior faces:
    852/946 (11% imbalance)
- Most loops are on cells,
  but some are on cells + ghosts,
  and MatVec is in cells + faces



Load imbalance (2/3)

- If load imbalance increases with processor count, scalability decreases
- If load imbalance reaches a high value (say 30% to 50%) but does not increase, scalability is maintained, but processor power is wasted
  - Perfect balancing is impossible to reach, as different loops show different imbalance levels, an synchronizations may be required between these loops
    - GCP uses MatVec and dot products
  - Load imbalance might be reduced using weights for domain partitioning, with Cell weight = 1 + *f*(n_faces)

Load imbalance (3/3)

- Another possible source of load imbalance is different cache miss rates on different ranks
  - Difficult to estimate in advance
  - With otherwise balanced loops, if a processor has a cache miss every 300 instructions, and another a cache miss every 400 instructions, considering that the cost of a cache miss is at least 100 instructions, the corresponding imbalance reaches 20%

Multigrid (1/2)

- Currently, multigrid coarsening does not cross processor boundaries
  - This implies that on *p* processors, the coarsest matrix may not contain less than *p* cells
  - With a high processor count, less grid levels will be used, and solving for the coarsest matrix may be significantly more expensive than with a low processor count
    - This reduces scalability, and may be checked (if suspected) using the solver summary info at the end of the log file
- Planned solution: move grids to nearest rank multiple of 4 or 8 when mean local grid size is too small

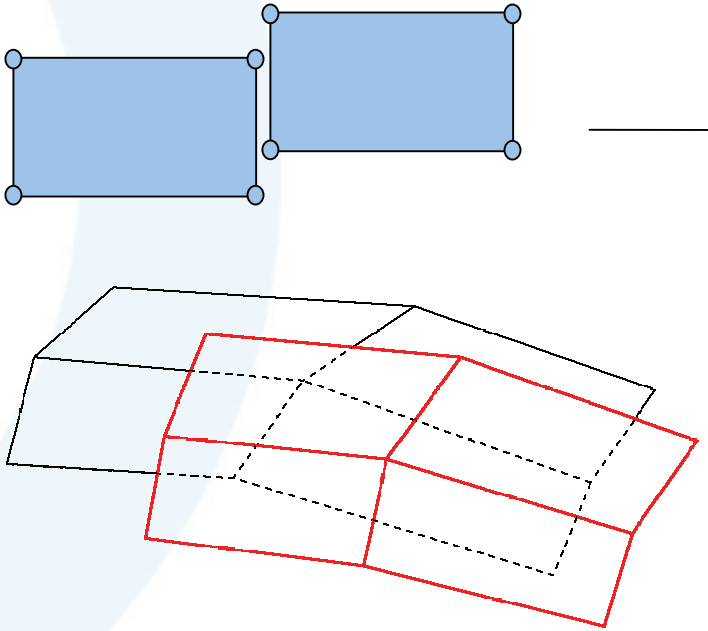
Multigrid (2/2)

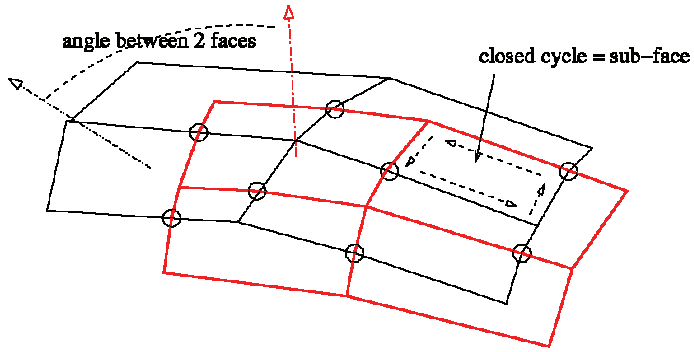
- Planned solution: move grids to nearest rank multiple of 4 or 8 when mean local grid size is too small
  - Most ranks will then have empty grids, but latency dominates anyways at this stage
  - The communication pattern is not expected to change too much, as partitioning is of a recursive nature (whether using recursive graph partitioning or space filling curves), and should already exhibit some sort of “multigrid” nature
  - This may be less optimal than some methods using a different partitioning for each rank, but setup time should also remain much cheaper
    - Important, as grids may be rebuilt each time step

Parallelization of mesh joining (1/2)

- Though mesh joining may not be a long-term solution (as it tends to be excessively used in ways detrimental to mesh quality), it is a functionality we can not do without as long as we do not have a proven alternative. Periodicity is also constructed as an extension of mesh joining.

- In addition, joining of meshes built in several pieces to circumvent meshing tool memory limitations is very useful.





Parallelization of mesh joining (2/2)

- Parallelizing this algorithm requires the same main steps as the serial algorithm:
  - Detect intersections (within a given tolerance) between edges of overlapping faces (done)
    - Uses parallel octree for face bounding boxes, built in a bottom-up fashion (no balance condition required here)
    - Preprocessor version used a lexicographical binary search, whose best case was O(n.log(n)), and worst case was $O(n^2)$.
  - Subdivide edges according to inserted intersection vertices
  - Merge coincident or nearly-coincident vertices/intersections
    - Requires recursive merge accept/refuse (done)
  - Re-build sub-faces

Parallel point location

- Parallel point location is used by the cooling tower and overset grid functionalities, as well as coupling with SYRTHES 4, neither of which is fully “standard” yet
  - Algorithm is parallel, but communication pattern could lead to some serialization
  - A simple improvement would be to order communication by communicating first among ranks in same half of rank set, then among ranks in other half, and ordering these communications recursively in the same manner
    - Load imbalance may be important, but serialization would be limited
- A rendez-vous approach using an octree or recursive bisection intermediate mesh may be more efficient, but is setup more complex and expensive

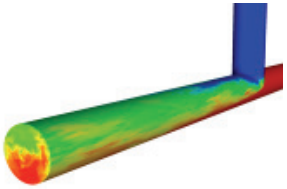
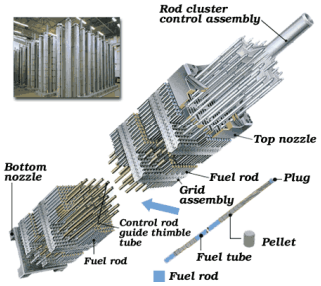
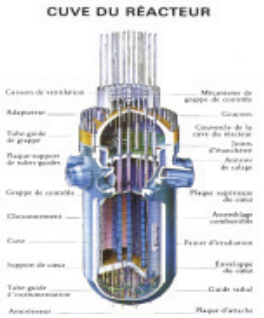
Hybrid MPI / OpenMP (1/3)

- Currently, only the MPI model is used:
  - By default, everything is parallel, synchronization is explicit when required
- On multiprocessor / multicore nodes, shared memory parallelism could also be used (using OpenMP directives)
  - Parallel sections must be marked, and parallel loops must avoid modifying the same values
    - Specific numberings must be used, similar to those used for vectorization, but with different constraints:
      - Avoid false sharing, keep locality to limit cache misses

Hybrid MPI / OpenMP (3/3)

- Hybrid MPI / OpenMP will be tested
  - IBM will do testing on Blue Gene/P
  - We also hope to provide OpenMP parallelism for ease of packaging / installation on Linux distributions
    - No dependencies on multiple MPI library choices, only on the gcc runtine
    - Good enough for current multicore workstations
      - Coupling with SYRTHES 4 will still require MPI

HPC roadmap: application examples

2003	2006	2007	2010	2015
Consecutive to the Civaux thermal fatigue event Computations enable to better understand the wall thermal loading in an injection. Knowing the root causes of the event ⇒ define a new design to avoid this problem.	 Computation with an L.E.S. approach for turbulent modelling Refined mesh near the wall.	 Part of a fuel assembly 3 grid assemblies	9 fuel assemblies No experimental approach up to now Will enable the study of side effects implied by the flow around neighbour fuel assemblies. Better understanding of vibration phenomena and wear-out of the rods.	The whole vessel reactor 

10^6 cells **3.10^{13} operations**	10^7 cells **6.10^{14} operations**	10^8 cells **10^{16} operations**	10^9 cells **3.10^{17} operations**	10^{10} cells **5.10^{18} operations**
Fujistu VPP 5000 1 of 4 vector processors 2 month length computation	Cluster, IBM Power5 400 processors 9 days	IBM Blue Gene/L « Frontier » 8000 processors # 1 month	**30 times** the power of IBM Blue Gene/L « Frontier » # 1 month	**500 times** the power of IBM Blue Gene/L « Frontier » # 1 month
# 1 Gb of storage 2 Gb of memory	# 15 Gb of storage 25 Gb of memory	# 200 Gb of storage 250 Gb of memory	# 1 Tb of storage 2,5 Tb of memory	# 10 Tb of storage 25 Tb of memory
Power of the computer	Pre-processing not parallelized	Pre-processing not parallelized Mesh generation	… ibid. … … ibid. … Scalability / Solver	… ibid. … … ibid. … … ibid. … Visualisation