Computational Science and Discovery

High Performance Parallel Computing of Flows in Complex Geometries - Part 1: Methods

N. Gourdain[1], L. Gicquel[1], M. Montagnac[1], O. Vermorel[1], M. Gazaix[2], G. Staffelbach[1], M. Garcia[1], J-F. Boussuge[1] and T. Poinsot[3]

[1] Computational Fluid Dynamics Team, CERFACS, Toulouse, 31057, France
[2] Computational Fluid Dynamics and Aero-acoustics Dpt., ONERA, Châtillon, 92320, France
[3] Institut de Mécanique des Fluides de Toulouse, Toulouse, 31400, France

Contact: Nicolas.gourdain@cerfacs.fr

ABSTRACT
Efficient numerical tools coupled with high performance computers, have become a key element of the design process in the fields of energy supply and transportation. However flow phenomena that occur in complex systems such as gas turbines and aircrafts are still not understood mainly because of the models that are needed. In fact, most CFD predictions as found today in industry focus on a reduced or simplified version of the real system (such as a periodic sector) and are usually solved with a steady-state assumption. This paper shows how to overcome such barriers and how such a new challenge can be addressed by developing flow solvers running on high-end computing platforms, using thousands of computing cores. Parallel strategies used by modern flow solvers are discussed with particular emphases on mesh-partitioning, load balancing and communication. Two examples are used to illustrate these concepts: a multi-block structured code and an unstructured code. Parallel computing strategies used with both flow solvers are detailed and compared. This comparison indicates that mesh-partitioning and load balancing are more straightforward with unstructured grids than with multi-block structured meshes. However, the mesh-partitioning stage can be challenging for unstructured grids, mainly due to memory limitations of the newly developed massively parallel architectures. Finally, detailed investigations show that the impact of mesh-partitioning on the numerical CFD solutions, due to rounding errors and block splitting, may be of importance and should be accurately addressed before qualifying massively parallel CFD tools for a routine industrial use.

PACS classification: 47.11.-j

NOMENCLATURE

ALE	Arbitrary Lagrangian Eulerian
API	Application Program Interface
CFD	Computational Fluid Dynamics
DES	Detached Eddy Simulations
DTS	Dual Time Stepping
FLOPS	FLoating-point Operation Per Second
HPC	High-Performance Computing
I/O	Input/Output (data)
IPM	Integrated Parallel Macros
LES	Large Eddy Simulation
MPI	Message Passing Interface
MIMD	Multiple Instructions, Multiple Data
MPMD	Multiple Programs Multiple Data

PU	Processing Unit (*i.e.* a computing core*)
RANS	Reynolds-Averaged Navier-Stokes
RCB	Recursive Coordinate Bisection
REB	Recursive Edge Bisection
RGB	Recursive Graph Bisection
RIB	Recursive Inertial Bisection
SPMD	Single Program Multiple Data

1. Introduction

Computational Fluid Dynamics -CFD- is the simulation, by computational methods, of the governing equations that describe the fluid flow behaviour (usually derived from the Navier-Stokes equations). It has grown from a purely scientific discipline to become an essential industrial tool for fluid dynamics problems. A wide variety of applications can be found in the literature, ranging from nuclear to aeronautic applications. Indeed CFD flow solvers are considered to be standard numerical tools in industry for design and development, especially for aeronautic and propulsion domains. This approach is attractive since it reproduces physical flow phenomena more rapidly and at a lower cost than experimental methods. However, industrial configurations remain complex and CFD codes usually require high-end computing platforms to obtain flow solutions in a reasonable amount of time. Within the real of computing architectures, High Performance Computing -HPC- has no strict definition but it usually refers to (massively) parallel processing for running quickly application programs. The term applies especially to systems that work above 10^{12} FLoating-point Operation Per Second -FLOPS- but it is also used as a synonym for supercomputing. Two kinds of architecture are generally set apart. The first one is the vector architecture that has high memory bandwidth and vector processing capabilities. The second architecture relies on superscalar cache-based computing cores interconnected to form systems of symmetric multi-processing nodes and has rapidly developed over the past decade due to the cost effectiveness of its basic components. This tendency is best illustrated by the list of the Peak Performance Gordon Bell Prize, delivered each year at the supercomputing conference and which rewarded applications running on massively parallel scalar platforms for the last successive three years after a period dominated by vector supercomputers (www.top500.org). However some difficulties are still observed with cached-based processors since the common computing efficiency is less than 20% for most CFD applications (Keyes *et al.*, 1997). One of the reasons is that flow solvers process a large amount of data, meaning a lot of communication is required between the different cache levels. Indeed such parallel platforms impose to run tasks on a large number of computing cores to be attractive and provide new possibilities for CFD by reducing computational time and giving access to large amounts of memory.

Parallel platforms integrate many processors that are themselves composed of one or more computing cores (a processing unit -PU- corresponds to the task performed by one computing core). Programming for parallel computing where simultaneous computing resources are used to solve a problem can become very challenging and performance is related to a great quantity of parameters. With a relatively small number of PU (<128), calculations efficiency relies essentially on communications as they are the primary parameters that counterbalance the architecture peak power. When thousands of PUs are used, load balancing between cores and communications dominate and both aspects clearly become the limiting factors of computational efficiency. This paper focuses on the work done in flow solvers to efficiently run on these powerful supercomputers. Objectives are to show techniques and methods that can be used to develop parallel computing aspects in CFD solvers and to give an overview of the authors' daily experience when dealing with such computing platforms. There is no consensus today in the CFD community on the type of mesh or data structure to use on parallel computers. Typically, flow solvers are usually efficient to solve the flow in only one specific field of applications such as combustion, aerodynamics, etc. Two different flow solvers are considered here to illustrate the different tasks that have to be considered to obtain a good efficiency on HPC platforms, with a special interest for (massively) parallel computing: a multi-block structured flow solver dedicated to aerodynamics problems (*elsA*) and an unstructured flow solver designed for reactive flows

* In this paper, it is assumed that one computing core is dedicated to only one process

(AVBP). Mesh partitioning and load balancing strategies considered by both solvers are discussed. Impact of HPC oriented programming, code architecture and the physical solutions they provide are also described.

2. Presentation of the flow solvers

Modern flow solvers have to efficiently run on a large range of supercomputers, from single core to massively parallel platforms. The most common systems used for HPC applications are the Multiple Instruction Multiple Data -MIMD- based platforms (Flynn, 1972). This category can be further divided in two sub-categories:

- Single Program, Multiple Data -SPMD-. Multiple autonomous PUs simultaneously executing the same program (but at independent points) on different data. The flow solver *elsA* is an example of a SPMD program;
- Multiple Program, Multiple Data -MPMD-. Multiple autonomous PUs simultaneously operating at least two independent programs. Typically such systems pick one PU to be the host (the "master"), which runs one program that farms out data to all the other PUs (the "slaves") which all run a second program. The software AVBP uses this method, referred as the "master-slave" strategy.

The *elsA* software is a reliable design tool in the process chains of aeronautical industry and is intensively used by several major companies (Cambier and Gazaix, 2002; Cambier and Veuillot, 2008). This multi-application CFD simulation platform deals with internal and external aerodynamics from low subsonic to hypersonic flow regime. The compressible 3-D Navier-Stokes equations for arbitrary moving bodies are considered in several formulations according to the use of absolute or relative velocities. A large variety of turbulence models from eddy viscosity to full Differential Reynolds Stress models is implemented for the Reynolds-Averaged Navier-Stokes -RANS- equations (Davidson, 2003), including criteria to capture laminar-turbulent transition phenomena for academic and industrial geometries (Cliquet *et al.*, 2007). In order to deal with flows exhibiting strong unsteadiness or large separated regions, high-fidelity methods such as Detached Eddy Simulation -DES- (Spalart *et al.*, 1997; Deck, 2005) and Large Eddy Simulation -LES- (Smagorinsky, 1962) are also available. For computational efficiency reasons, connectivity and data exchange between adjacent blocks are set by means of ghost cells. High flexibility techniques of multi-block structured meshes have been developed to deal with industrial configurations. These advanced techniques include totally non-coincident interfaces (Fillola *et al.*, 2004) and the Chimera technique for overlapping meshes (Meakin, 1995). The flow equations are solved by a cell centred finite-volume method. Space discretization schemes include upwind schemes (Roe, 1981; AUSM, Liou, 1996) or centred schemes, stabilized by scalar or matrix artificial dissipation. The semi-discrete equations are integrated, either with multistage Runge-Kutta schemes or with a backward Euler integration that uses implicit schemes solved by robust Lower-Upper -LU- relaxation methods (Yoon and Jameson, 1987). For time accurate computations, the implicit Dual Time Stepping -DTS- method is available (Jameson, 1991). Several programming languages are used to develop the *elsA* software: C++ as main language for implementing the object-oriented design (backbone of the code), Fortran for CPU efficiency in calculation loops and Python for user interface. The code has already been ported on a very large variety of platforms, including vector and massively scalar supercomputers. Vector capabilities are still supported, but focus today is clearly on scalar parallel platforms with an increasing number of computing cores. To run in parallel, *elsA* uses a simple coarse-grained SPMD approach, where each block is allocated to a PU (note that in some circumstances several blocks can be allocated to the same PU). To achieve a good scalability, a load-balancing tool with block-splitting capability is included in the software.

The AVBP project was historically motivated by the idea of building a modern software tool for CFD with high flexibility, efficiency and modularity. Recent information about this flow solver can be found in Garcia (2009). This flow solver is used in the frame of cooperative works with a wide range of industrial companies. AVBP is an unstructured flow solver capable of handling hybrid grids of many cell types (tetra/hexahedra, prisms, etc.). The use of these grids is motivated by the efficiency of unstructured grid generation, the accuracy of the computational results (using regular structured elements) and the ease of mesh adaptation for industrial flow applications. AVBP solves the laminar and turbulent compressible Navier-Stokes equations in

two or three space dimensions and focuses on unsteady turbulent flows (with and without chemical reactions) for internal flow configurations. The prediction of these unsteady turbulent flows is based on LES which has emerged as a promising technique for problems associated with time dependent phenomena, and coherent eddy structures (Moin and Apte, 2006). An Arrhenius law reduced chemistry model allows investigating combustion for complex configurations. The data structure of AVBP employs a cell-vertex finite-volume approximation. The basic numerical methods are based on a Lax-Wendroff (1960) or a Finite-Element type low-dissipation Taylor-Galerkin (Donea, 1984; Colin and Rudgyard, 2000) discretization in combination with a linear-preserving artificial viscosity model. Moving mesh capabilities following the Arbitrary Lagrangian-Eulerian -ALE- approach (Hirt *et al.*, 1974) have been developed to allow the simulation of moving bodies (piston in a cylinder…). AVBP library includes integrated parallel domain partitioning and data reordering tools, handles message passing and includes supporting routines for dynamic memory allocation, parallel input/output -I/O- and iterative methods.

3. Parallel efficiency

In CFD, the parallel computing strategy is to decompose the original problem into a set of tasks. Simultaneous multiple computing resources are then used to solve the problem. Thus, a problem must be divided into N sub-problems (where N is typically the number of available PUs), and each part of the problem is solved by one of the PU concurrently. Ideally, the overall time $T_{parallel}$ of the computation will be $T_{sequential}$ /N, where $T_{sequential}$ is the calculation time for the problem using only one PU. The main issue is to first estimate the reliability of the parallel computing strategy before implementing the resolution of the problem and potentially wasting computer time. The various concepts used in such analyse are presented in this section.

3.1 Definition of the task scalability

Scalability measures whether or not a given problem can be solved faster as more PUs are used. The speedup S is defined as the ratio between $T_{parallel}$ and $T_{sequential}$:

$$S = T_{sequential} / T_{parallel} \quad \textbf{(1)}$$

It is used to quantify the scalability of the problem. Typically, a value of S equal to the number of available PUs corresponds to fully parallel tasks. The computing efficiency E corresponds to the value of S divided by the number of computing cores N (ideally, S=N so E=1). A poor efficiency means that the problem is not correctly balanced between all PUs (load balancing error) or that the time needed for communications is important compared to computing time. To estimate the theoretical parallel efficiency of a calculation, the most popular model is the Amdahl's law that gives the maximum theoretical speedup S_{Amdahl} defined by:

$$S_{Amdahl} = NP + (1 - P) \quad \textbf{(2)}$$

In Eq. (2), N stands for the number of computing cores, P is the part of the task that can be made parallel and (1-P) is the part that remains sequential. Amdahl's law assumes that all processes finish their task at the same time (P) and one process is then dedicated to a sequential task (1-P). This assumption of perfect load balancing is not true in most CFD applications, especially when an industrial configuration is considered. In fact, each process ends its task at a different time and the value of (1-P) is no longer meaningful. To deal with this problem, Amdahl's law can be extended by taking into account the working time of each process (Gourdain *et al.*, 2009). A typical scheme of a parallel computation is shown in Fig. 1, with the time spent by each process to perform its task.

By convention, the overall time related to the parallel computation $T_{parallel}$ is set to 1. The sequential computation time $T_{sequential}$ is thus estimated by adding the time spent by each process and is expressed by:

$$T_{sequential} = NP_0 + (N-1)P_1 + ... + 2P_{N-2} + P_{N-1} \quad \textbf{(3)}$$

with P_{i+1} the part of the work that is computed with (N-i) computing cores and $P_0+P_2+\ldots+P_{N-1}=1$. The strong speedup S_{EA} is directly found as the ratio between $T_{sequential}$ and $T_{parallel}$:

$$S_{EA} = \sum_{i=0}^{N-1} (N-i)P_{i+1} \quad \textbf{(4)}$$

This law is a simple extension of the Amdahl's law (which is found simply by taking $P_1=P_2=\ldots=P_{N-2}=0$), considering the "mean" problem. The main difficulty to apply the Extended Amdahl's law (EA) given by Eq. (4) is to estimate the exact value of the P_{i+1} (in fact, the computing time related to N-i processes), since it includes communication costs, load balancing errors, etc. As a first approximation, only the number of allocated mesh points is taken into account to evaluate the computing core load. In Fig. 2a, an example of a mesh point distribution by PU is shown for a turbine test case where a perfect load balancing can not be achieved. The mean value corresponds to the mean number of mesh points by PU.

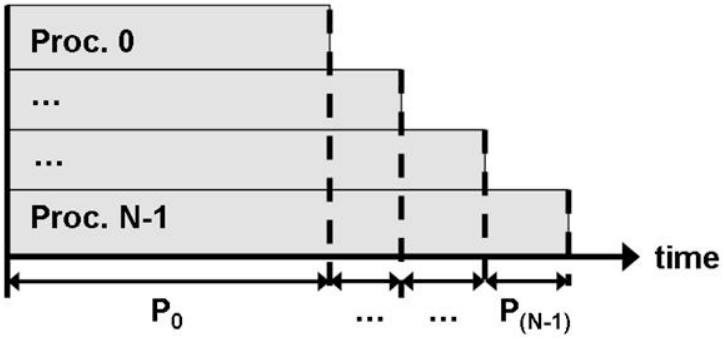


Fig. 1 Typical scheme of a parallel CFD computation.

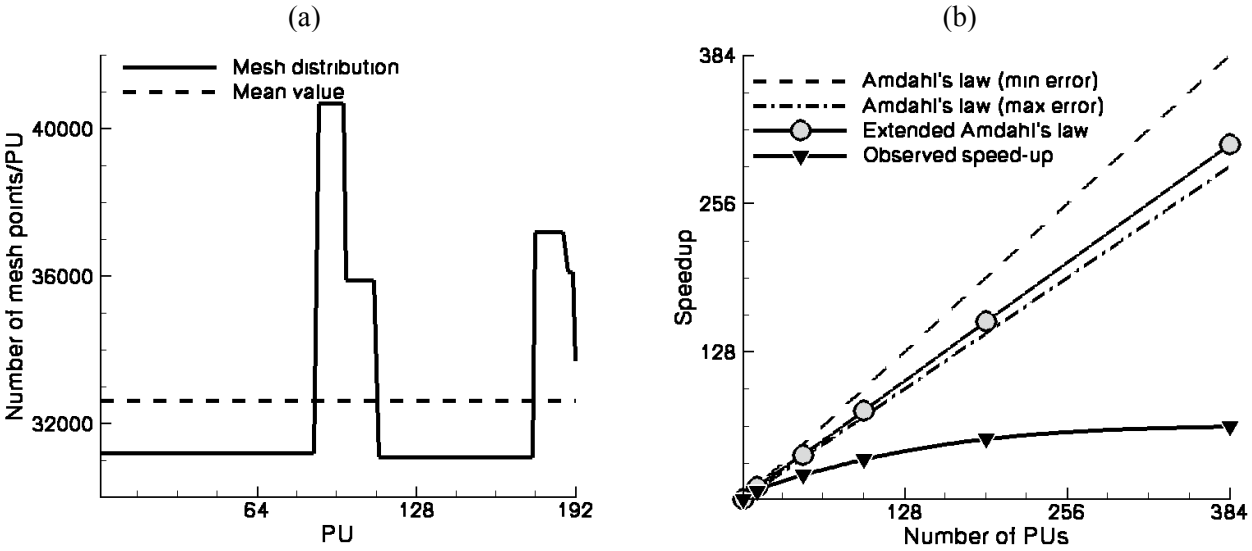


Fig. 2 Example of a mesh distribution (a) and related theoretical speedup with Amdahl's and extended Amdahl's laws (b). Turbine guide vane using a 10M cells grid and 1 to 384 PUs.

Speedups are estimated with Amdahl's and extended Amdahl's laws. Results are compared Fig. 2b with the observed speedup. For the classical Amdahl's law, the value of (1-P) can be chosen equal to the maximum or

the minimum load balancing error, divided by the number of computing cores. The Amdahl's law shows a quasi-linear behaviour for the minimum value of the load balancing error, while the extended Amdahl's law predicts values of the speedup right between the two bounds obtained with the Amdahl's law (as expected). While the predicted efficiency ranges from 75% to 100%, the observed efficiency is only around 17%. This example shows that the maximum load balancing error (that corresponds to the limiting PU) and the load balancing error distribution (that affects all PUs) have an influence on the parallel computing efficiency. However, it is clear that in some cases, other parameters such as communications must be considered to correctly predict the efficiency of a parallel task. In the present case, all the laws based only on geometric criteria (*i.e.* the mesh points distribution) failed to estimate the correct efficiency of the parallel computation.

3.2 Flow solver scalability

Usually, the parallel efficiency of a flow solver is evaluated with the "strong" speedup that represents the gain with respect to a sequential calculation. However, this criterion does not give always satisfying indications. First, a good speedup can be related to poor sequential performance (for example if a PU computes slowly with respect to its communication network). This point is critical, especially for cached-based processors: super-linear speedups are often observed in the literature with these chips, mainly because the large amount of data required by the flow solver leads to a drop down of the processor performance for a sequential task. For CFD applications, the potential solution is to increase the size of the cache memory and to reduce the communication time between the different cache levels. Second, the sequential time is usually not known, mainly due to limited memory resources. Except for vector supercomputer, the typical memory size available for one PU ranges from 512 Mb to 32 Gb. As a consequence, in most cases, industrial configurations cannot be run on scalar platforms with a single computing core and no indication about the real speedup can be obtained. A common definition of the speedup is often used by opposition to the "strong" speedup: the normalized speedup that is related to the time ratio between a calculation with N PUs and N_{min} PUs, where N_{min} is the smallest number of PUs that can be used for the application. The normalized speedup is thus useful to give an idea of the flow solver scalability.

A short overview of the normalized speedup obtained for many applications performed with *elsA* and AVBP is indicated in Fig. 3. While *elsA* shows a good scalability until 2,048 PUs, results for 4,096 PUs are quite disappointing.

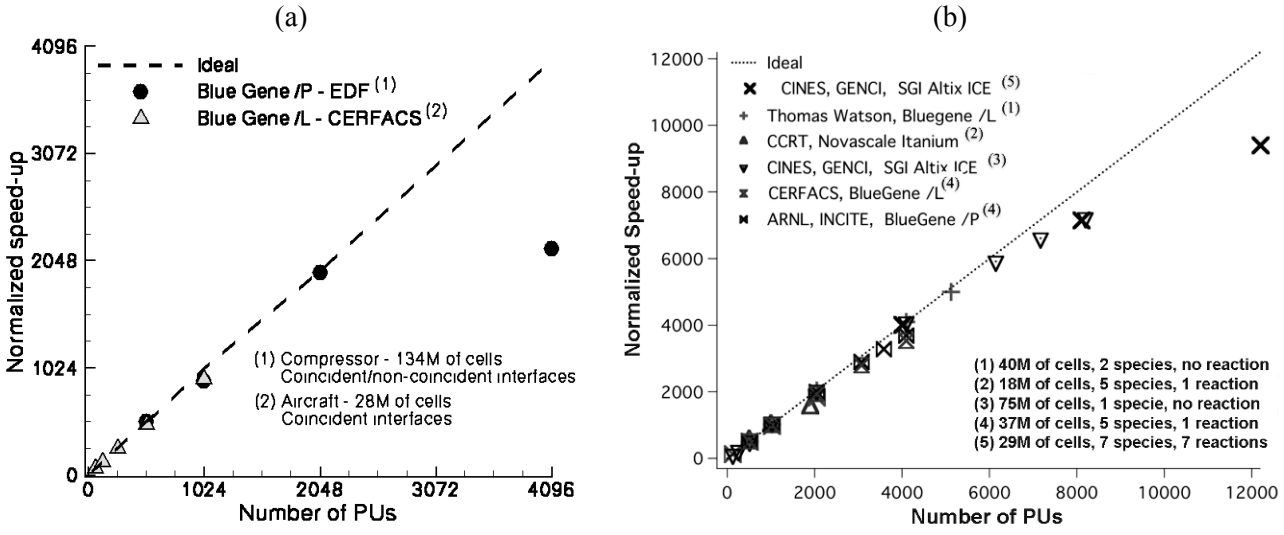


Fig. 3 Overview of the normalized speedup for applications with *elsA* (a) and AVBP (b).

This is mainly related to the difficulty for balancing a complex geometry configuration (non-coincident interfaces between blocks). Moreover the load balancing error can be modified during this unsteady flow simulation (due to sliding mesh). AVBP has a long experience in massively parallel calculations, as indicated

by the wide range of scalar supercomputers used. This flow solver shows an excellent scalability until 6,144 PUs since the normalized speedup is very close to ideal. Results are still good for 12,288 PUs (efficiency is around 77%). This decrease of performance for a very large number of PUs underlines a physical limit often encountered on massively parallel applications. Indeed for very large numbers of PUs the ratio between computation time and communication time is directly proportional to the problem size (number of grid points and unknowns). For this specific illustration, the configuration tested with 12,288 PUs corresponds to a calculation with less than 3,000 cells by PU or a too low computational workload for each PU compared to the amount of exchanges needed to proceed with the CFD computation. It shows that a given task is limited in terms of scalability and no increase of performance is expected beyond a given number of PUs.

4. Mesh partitioning strategies

The mesh partitioning is thus a challenging step to maintain good efficiency as the number of computing cores increases. Many claims of "poor parallel scaling" are often ill-found, and usually vanish if care is brought to ensure good load balancing and minimization of overall communication time. Indeed, the possibility to run a problem on a large number of PUs is related to the capacity of efficiently splitting the original problem into parallel sub-domains. The original mesh is partitioned in several sub-domains that are then (ideally equally) distributed between all PUs. One of the limitations is that the use of parallel partitioning algorithms often needed when using many computing cores remains very complex and this task is usually performed sequentially by one PU, leading to memory constraints. These constraints are much more important for unstructured meshes than for block-structured meshes, mainly because the mesh-partitioning of unstructured grids requires large tables for mesh coordinates and connectivity.

4.1 General organization

In most flow solvers, mesh partitioning is done in three steps: graph partitioning, nodes (re)ordering and blocks generation. An additional step is the merging of output data generated by each subtask (useful in an industrial context for data visualization and post-processing). As shown in Fig. 4, two strategies can be used: either the mesh-partitioning is done as a pre-processing step and all computing cores execute the same program during the computation (the SPMD approach used by *elsA*), or a master PU is designed to perform the mesh-partitioning task before or during the calculation (the MPMD approach used by AVBP). In the first approach (Fig. 4a), a new problem adapted to the desired number of PUs is first defined, and the parallel calculation is performed using all PUs for computation and input/output. However, this partitioning strategy does not allow a dynamic partitioning procedure which can be of interest for specific applications and that happens during the calculation, load balancing errors may vary during the calculation. The second approach (Fig. 4b) is more user-friendly (the mesh partitioning step is hidden to users) and is adapted to dynamic partitioning and (re)meshing (for moving bodies). During the computation, the master PU can be dedicated only to input/output or can be used for computation and input/output. However, during the computational steps, the communication strategy defining the exchanges between master and slave PUs can be quite complex and penalizing for parallel efficiency.

4.2 Structured multi-block meshes

Splitting the original multi-block structured mesh into a new set of several sub-domains is a necessary step to obtain a correct load-balancing. Some constraints have also to be taken into account at this stage such as multigrid capacities and connectivity between blocks (coincident or non-coincident interfaces). A major difficulty with multi-block meshes is related to the use of ghost cells that are needed for boundary condition treatments and information exchanges between blocks. Calculations that use thousands of PUs impose to split the original configuration in order to obtain a very high number of blocks, resulting into a number of ghost cells that can be largely increased. Ghost cells affect directly the memory requirements and the computational time (the same quantity of information is stored in "real" and ghost cells). With the *elsA* flow solver, rows of ghost cells are created for each block, in each direction. In the case of a block with N_i, N_j, N_k cells in each direction, the total number of ghost cells K_{ghost} is:

$$K_{ghost} = (N_i + 2N_{gc})(N_j + 2N_{gc})(N_k + 2N_{gc}) - (N_i N_j N_k) \quad \textbf{(5)}$$

with N_{gc} being the number of ghost cells rows (default value is 2).

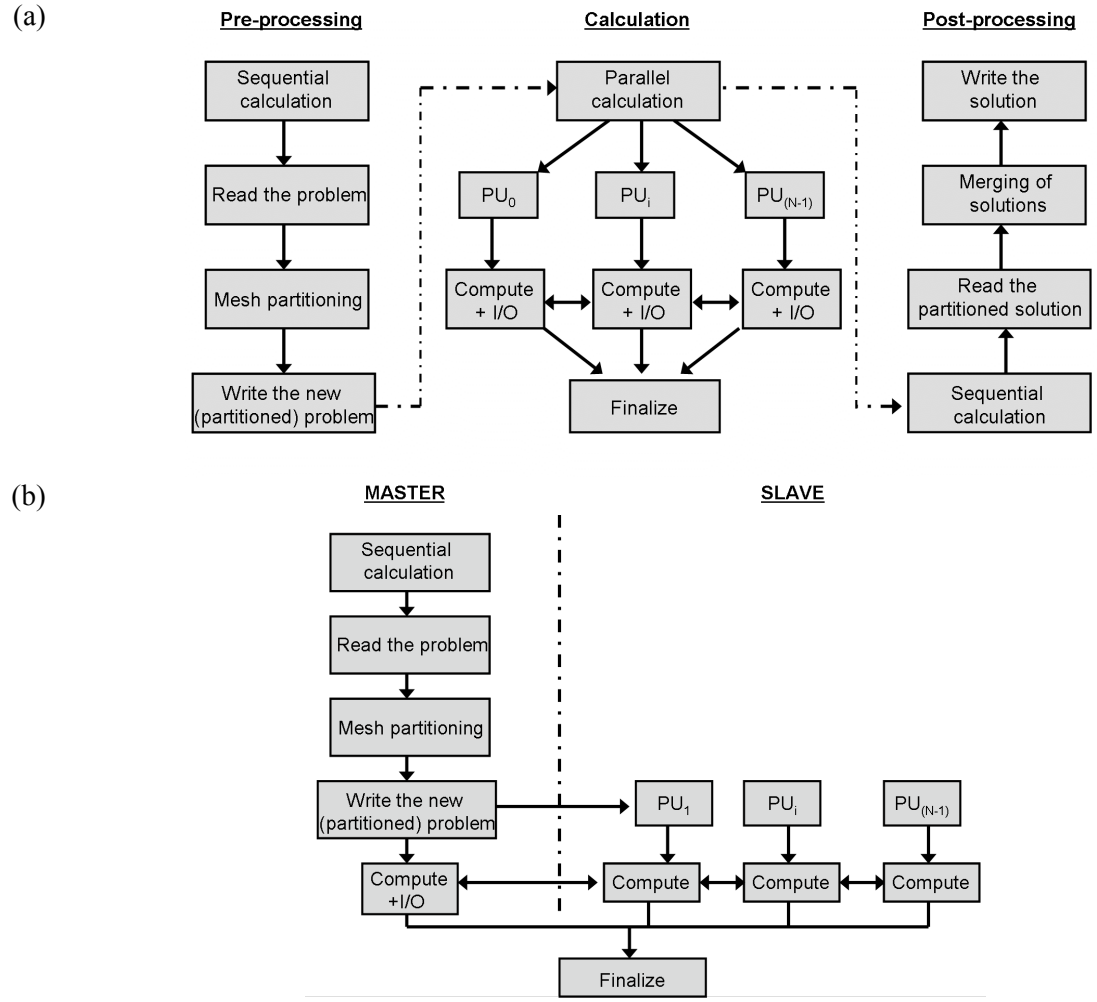


Fig. 4 Overview of the mesh partitioning strategies: (a) pre-processing step with a SPMD approach, (b) dynamic mesh-partitioning with a MPMD approach.

An example of the ghost cells effect is presented in Fig. 5 for a single cubic block with $N_i=N_j=N_k$. In Fig. 5a, the ratio between ghost cells and "real" cells is underlined for typical values of block dimensions in an industrial context (usually from 8^3 to 128^3). While ghost cells represent less than 10% of the total size of a block with 128^3 cells, the value reaches more than 40% for a block with 32^3 cells. For very small blocks typically used for technological effects (such as thick trailing edge, tails, tip leakage, etc), the number of ghost cells could be more than 200% of the number of "real" cells. The effect of block splitting on ghost cells is pointed out by Fig. 5b. It clearly shows the increase of the problem size after successive blocks splitting, for two initial block sizes (64^3 and 32^3). The law followed by the problem size is perfectly linear with a slope that depends only on the original block size. If an initial block with 64^3 cells is split into 16 new blocks of equal dimensions, the problem size has been increased by 190%. An example of the mesh splitting impact on industrial applications is shown in Fig. 6. The configuration is an industrial multi-stage compressor represented by an original grid composed of 2848 blocks and more than 130 millions cells. The original configuration is split and balanced over 512 PUs to 12288 PUs. As expected, the number of blocks and the size of the problem are directly correlated. For a reasonable number of PUs, the number of ghost cells has a relatively small impact on the size of the problem while the number of blocks is largely increased with

respect to the original configuration. For example, the size of the configuration corresponding to 4096 PUs is increased by +8% while the number of blocks is increased by +210%. In fact, only largest blocks have been split, leading to a small impact on the number of ghost cells. However, for 12288 PUs, both the number of blocks and the size of the configuration are significantly increased (resp. +590% and +37%). It is clear that this problem come from the use of ghost cells, which is not adapted to massively parallel calculations when block splitting is required. On the one hand, the suppression of this method would result in an increase of the parallel efficiency. On the other hand it will also considerably increase the sequential time. Indeed, the balance between both effects is not easy to predict and the result will depend on the configuration and the number of desired PUs.

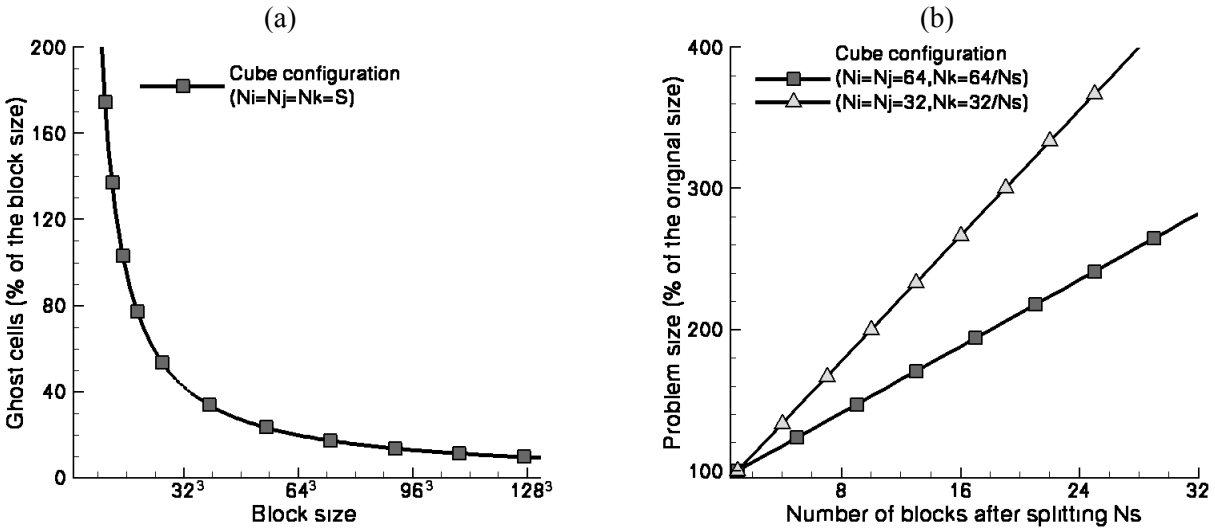


Fig. 5 Number of ghost cells with respect to the block size (a) and evolution of the problem size with respect to the number of blocks after splitting (b) - simulations performed with *elsA*.

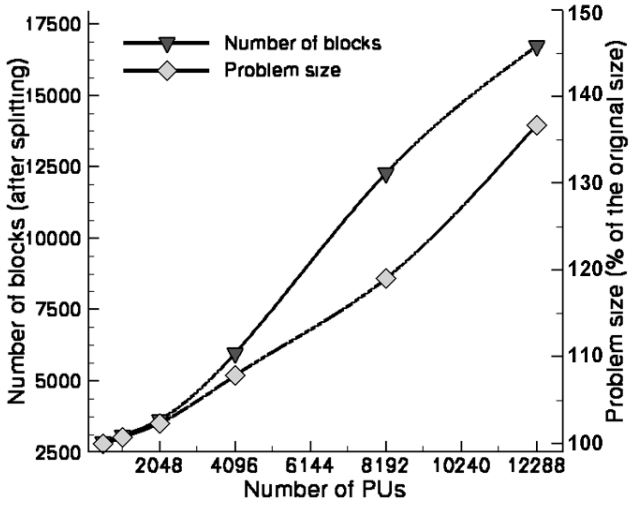


Fig. 6 Evolution of the problem size and number of blocks after splitting for an industrial compressor configuration (*elsA*, greedy algorithm).

For parallel computing applications, the program allocates one or more blocks to each PU with the objective of minimizing the load balancing error. To evaluate the load of PUs, the *elsA* software considers only the total number of cells Q_{total} (ghost cells are not taken into account). Then the ideal number of cells Q_{mean} allocated to one computing core depends only on the total number of computing cores N:

$$Q_{mean} = \frac{Q_{total}}{N} \quad \textbf{(6)}$$

The load balancing error related to a given PU is defined as the relative error between the allocated number of cells and the ideal number of cells Q_{mean} which would result into an optimal work load of the PU. The objective of mesh partitioning is to minimize the load balancing error. Two mesh partitioning algorithms are available in *elsA* that only consider geometric information (no information from the communication graph is used):

- The Recursive Edge Bisection -REB- algorithm works only on the largest domain (it splits the largest block along its longest edge until the number of blocks equals the desired number of PUs indicated by the user). This very simple algorithm is well adapted to simple geometries such as cubes but it usually gives poor quality results in complex configurations;
- The so-called greedy algorithm is a more powerful tool to split blocks and is based only on geometric criteria (Ytterstrom, 1997). This approach loops over blocks looking for the largest one (after splitting when necessary) and allocating it to the PU with the smaller number of cells until all blocks are allocated. An example is shown in Fig. 7 for a configuration with 5 initial blocks. Based on an objective of 16 PUs, the greedy algorithm successively splits the blocks to obtain a new problem with 19 blocks yielding a load balancing error of roughly 3%.

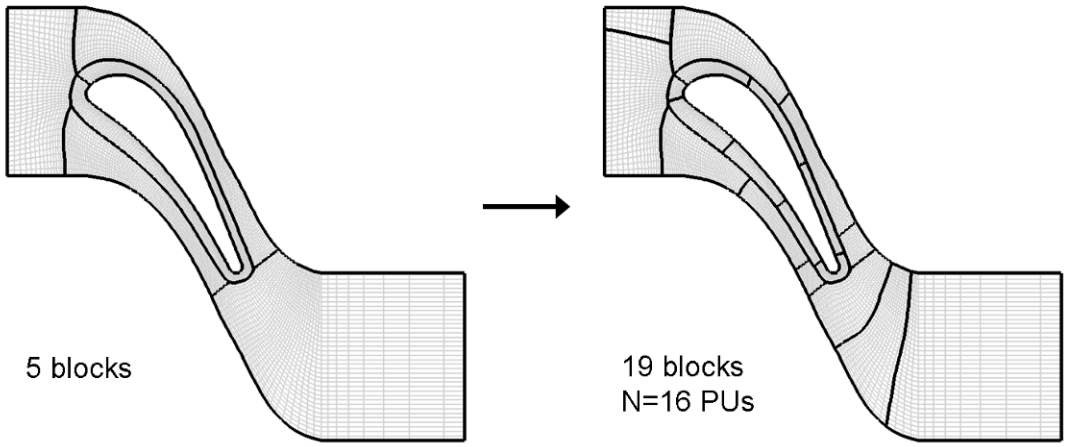


Fig. 7 Example of a multi-block mesh partitioning and load balancing with the greedy algorithm (*elsA* calculations).

It is known that the problem of finding the optimal set of independent edges is NP-complete (*i.e.* it is a non-polynomial and non-deterministic problem). More information about NP-complete problem can be found in Garey and Johnson (1979). With a few hundred computing cores, the greedy heuristic approach gives good results and more sophisticated methods would probably not significantly improve the graph coverage (Gazaix *et al.*, 2008). However for applications with thousands of PUs, some improvements should be considered. A simple thing to do is to order the edges according to their weight (based on the size of the interface meshes). In this case, the edge weights in one set are close to each other, and every message in one communication stage should take approximately the same amount of time to exchange. This method is expected to reduce the communication time.

4.3 Unstructured meshes

An example of an unstructured grid and its associated dual graph are shown in Fig. 8a. Each vertex represents a finite element of the mesh and each edge represents a connexion between the faces of two elements (and therefore, the communication between adjacent mesh elements). The number of edges of the dual graph that are being cut by partitioning is called an edge-cut. This subdivision results in an increase of the total number

of grid points due to a duplication of nodes at the interface between two sub-domains. The communication cost of a sub-domain is a function of its edge-cut as well as the number of neighbouring sub-domains that share edges with it. In practice, the edge-cut of a partitioning is used as an important indicator of partitioning quality for cell-vertex codes. The sparse matrix related to the dual graph of this grid is shown in Fig. 8b. The axes represent the vertices in the graph and the black squares show an edge between vertices. For CFD applications, the order of the elements in the sparse matrix usually affects the performance of numerical algorithms. The objective is to minimize the bandwidth of the sparse matrix (which represents the problem to solve) by renumbering the nodes of the computational mesh in such a way that they are as close as possible to their neighbours. This insures optimal use of memory of the computational resources and is mandatory in unstructured CFD solvers. In AVBP, node reordering can be done by the Cuthill-McKee -CM- or the reverse Cuthill-McKee -RCM- algorithms (Cuthill and McKee, 1969). In the case of AVBP, the RCM algorithm provides similar results at a lower cost in terms of time than the CM algorithm. An example of the effects of node reordering on the performance of a simulation with AVBP is presented in Fig. 9.

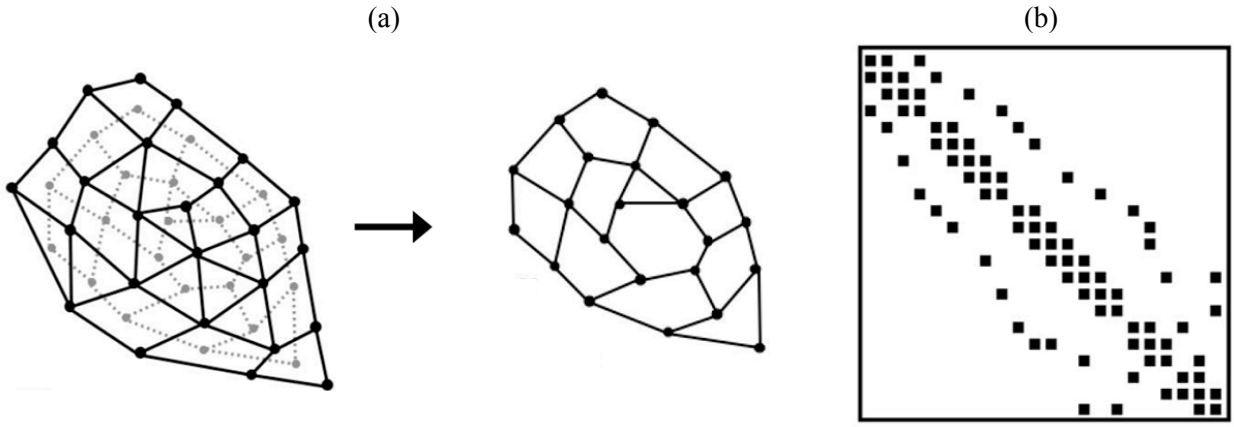


Fig. 8 Example of an unstructured grid with its associated dual graph and partitioning process (a) and the related sparse matrix (b).

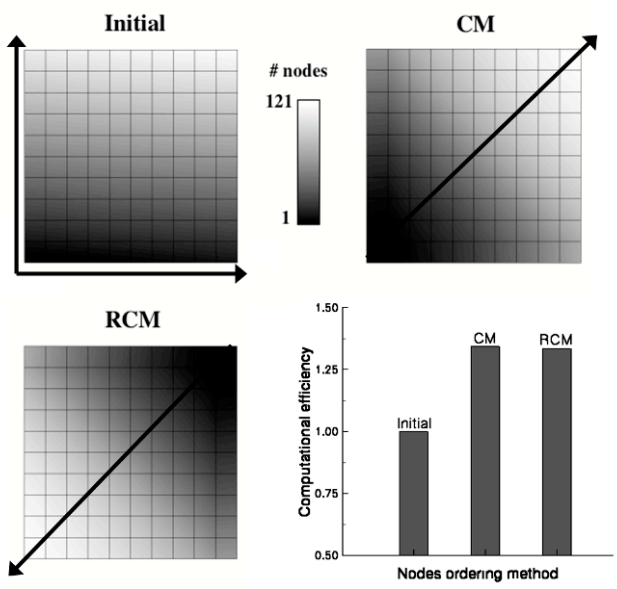


Fig. 9 Effect of the nodes ordering on the computational efficiency of AVBP. Cuthill-McKee (CM) and Reverse Cuthill-McKee (RCM) show an increase of the efficiency by +35%.

The configuration used in these tests is a hexahedron-based grid with 3.2 millions of nodes (and 3.2 millions cells). The impact of nodes ordering is measured through the computational efficiency, defined as the ratio between the computational time after reordering and before reordering. With 32 PUs, this efficiency increases by +35% with the CM or the RCM methods. These results show the importance of nodes ordering, in the case of unstructured CFD solvers especially on massively parallel machines to simulate large scale industrial problems. Different partitioning algorithms are currently available in AVBP and largely detailed in Garcia (2009). Two methods are geometric based algorithms, considering only coordinates and distances between nodes (Euclidean distance). A third one is a graph theory based algorithm that uses the graph connectivity information. These three first algorithms give satisfactory results for most applications but show limitations in terms of computational time and mesh partitioning quality for large size problems (such as the one usually considered with HPC). A fourth algorithm has thus been implemented (METIS) in AVBP (Karypis and Kumar, 1998), which is especially well adapted for Lagrangian and massively parallel calculations. A short description of these four methods is provided below:

- Recursive Coordinate Bisection -RCB- is equivalent to the REB algorithm available in the *elsA* software (Berger and Bokhari, 1987). The weakness of this algorithm is that it does not take advantage of the connectivity information;
- Recursive Inertial Bisection -RIB- is similar to RCB (Williams, 1991) but it provides a solution that does not depend on the initial mesh orientation (Fig. 10). The weakness of this algorithm is identical to the RCB method;
- Recursive Graph Bisection -RGB- considers the graph distance between two nodes (Simon, 1991). In AVBP, the determination of pseudo-peripheral nodes, which is critical for the algorithm, is obtained with the Lewis implementation of the Gibbs-Poole-Stockmeyer algorithm (Lewis, 1982). The interest of this method is that it considers the graph distance;
- The METIS algorithms are based on multilevel graph partitioning: multilevel recursive bisectioning (Karypis and Kumar, 1999) and multilevel k-way partitioning (Karypis and Kumar, 1998). As the partitioning algorithms operate with a reduced-size graph (Fig. 11), they are extremely fast compared to traditional partitioning algorithms. Testing has also shown that the partitions provided by METIS are usually better than those produced by other multilevel algorithms (Hendrickson and Leland, 1993).

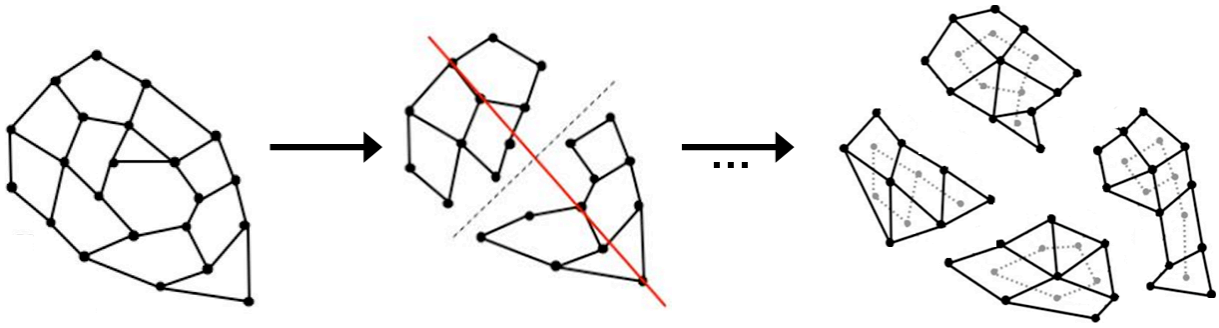


Fig. 10 Example of a partitioning process with an unstructured grid and the RIB algorithm (AVBP calculations).

It is not always easy to choose the best partitioning algorithm for a given scenario. The quality of the partition and the time to perform it are two important factors to estimate the performance of a partitioning algorithm. Complex simulations such as Euler-Lagrange calculations (two-phase flows) are good candidates to evaluate mesh partitioning algorithms (Apte *et al.*, 2003). During a two-phase flow simulation, particles are free to move everywhere in the calculation domain. As a first approximation, it is possible to assume that the particle density is constant in the domain. In this case, the number of particles in a domain depends only on the domain dimension (a large volume corresponds to a high number of particles). Results obtained with two different algorithms are presented in Fig. 12. The RCB algorithm takes into account only information

related to the number of cells while the k-way algorithm (METIS) uses more constraints such as particle density. A comparison of the two methods shows that the RCB method leads to an incorrect solution for load balancing (Fig. 13). With the RCB algorithm one PU supports the calculation for most the particles, Fig. 13a. The k-way algorithm gives a more satisfactory result by splitting the mesh in more reasonable dimensions, leading to a correct load balancing of the particles between PUs, Fig. 13b. Many tests have been performed with a wide range of sub-domain numbers, from 64 (2^6) to 16,384 (2^{14}), in order to evaluate the scalability of the different partitioning algorithms. The time required to partition a grid of 44 millions cells on 4096 sub-domains ranges from 25 minutes with METIS (k-way algorithm) to 270 minutes with the RGB algorithm (benchmarks performed on an Opteron-based platform).

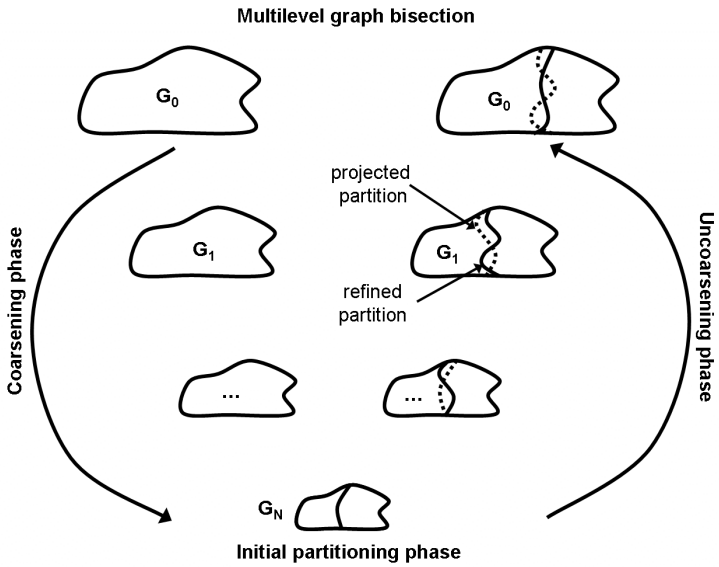


Fig. 11 Working scheme of the multilevel graph bisection. During the uncoarsening phase the dotted lines indicate projected partitions and solid lines indicate partitions after refinement (Karypis, 1999).

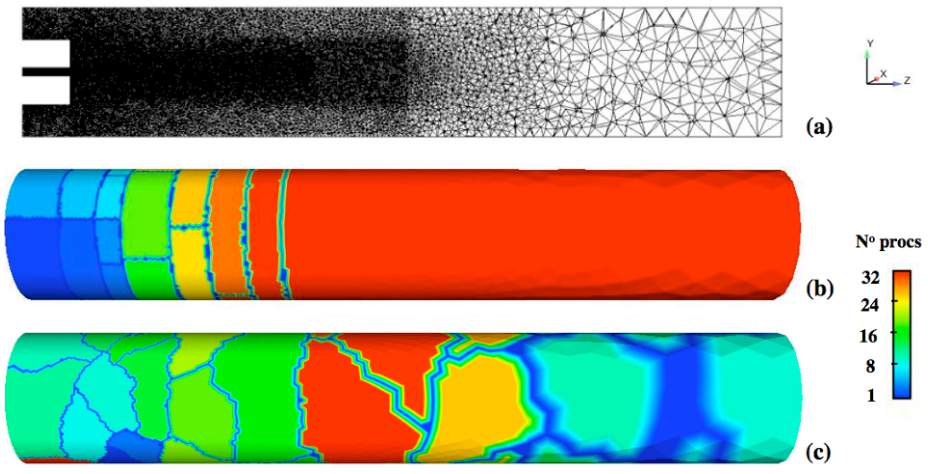
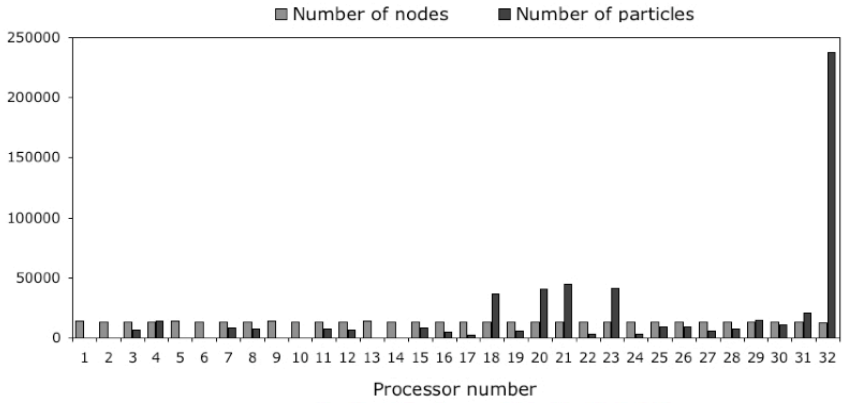


Fig. 12 Example of the mesh partitioning considering a two-phase flow configuration with AVBP (a) by using the RCB algorithm (b) and the k-way algorithm used by METIS (c).

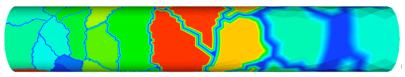
For most applications, METIS gives the fastest results, except for small size grids and for large number of PUs (for which the RCB algorithm is slightly better). To compare the quality of the mesh partitioning, the number of nodes after partitioning is shown in Fig. 14 for moderate (10 millions cells, 1.9 millions nodes) and large (44 millions cells, 7.7 millions nodes) grids. The reduction in the number of nodes duplicated with

the k-way algorithm of METIS is significant. The difference tends to be larger when the number of sub-domains increases, leading to an important gain in the calculation efficiency due to the reduction of the number of nodes and communication costs.

(a) RCB algorithm



(b) k-way algorithm (METIS)



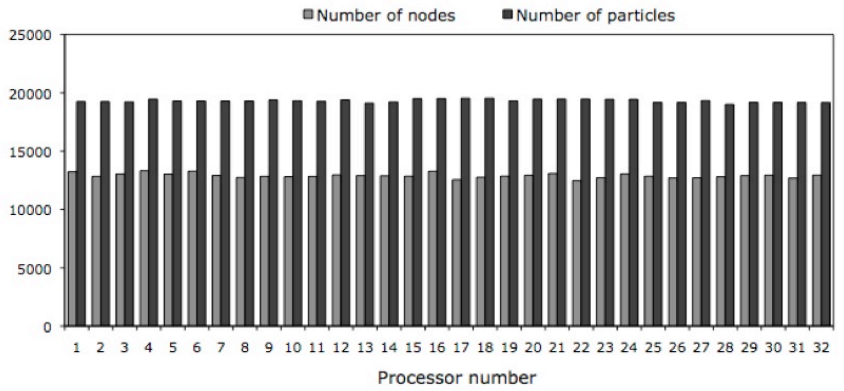


Fig. 13 Impact of the mesh-partitioning on the particle balance with AVBP: (a) RCB algorithm, (b) *k*-way algorithm (METIS).

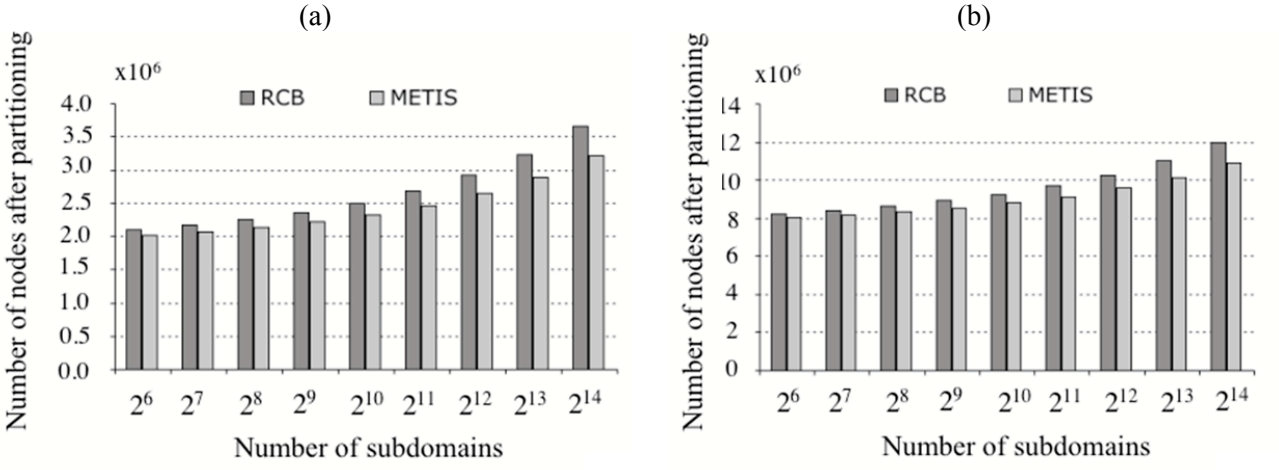


Fig. 14 Number of nodes after partitioning of combustion chamber meshes with AVBP by using RCB and *k*-way (METIS) methods: grids with 10 millions (a) and 44 million cells (b).

The previous example shows that a considerable improvement of the mesh-partitioning quality can be obtained by using multi-levels and multi-graphs algorithms. In fact for AVBP, the combination of the RCM algorithm (for node reordering) and the METIS algorithms provide the best results in terms of cost/parallel

efficiency ratio. Moreover, these algorithms usually provide a solution in a less amount of time than classical geometrical algorithms and they are particularly well fitted to parallel implementation. Indeed, the development of a parallel implementation for partitioning algorithms is a current challenge to reduce the computational cost of this step.

5. Communication and scheduling

For most CFD applications, processes are not fully independent and data exchange is required at some points (fluxes at block interfaces, residuals, etc.). The use of an efficient strategy for message passing is thus necessary for parallel computing, especially with a large number of computing cores. Specific instructions are given through standard protocols to communicate data between computing cores, such as MPI and OpenMP. Communication, message scheduling and memory bandwidth effects are detailed in this section.

5.1 Presentation of message passing protocols

MPI is a message passing standard library based on the consensus of the MPI Forum (more than 40 participating organizations). Different kinds of communication can be considered in the MPI Application Program Interface -API- such as point-to-point and collective communications (Gropp *et al.*, 1999). Point-to-point MPI calls are related to data exchange between only two specific processes (such as the MPI "send" function family). Collective MPI calls involve communication between all computing cores in a group (either the entire process pool or a user-defined subset). Each of these communications can be done through "blocking" (the program execution is suspended until the message buffer is safe to use) or "non-blocking" protocols (the program does not wait to be certain that the communication buffer is safe to use). Non-blocking communications have the advantage of allowing the computation to proceed immediately after the MPI communication call (computation and communication can be overlapping). While MPI is currently the most popular protocol for CFD flow solvers, it is not perfectly well adapted to new computing architectures with greater internal concurrency (multi-core) and more levels of memory hierarchy. Multithreaded programs can potentially take advantage of these developments more easily than single threaded applications. This has already led to separate, complementary standards for symmetric multiprocessing, such as the OpenMP protocol. The OpenMP API supports multi-platform shared-memory parallel programming in C/C++ and Fortran on all architectures. The OpenMP communication system is based on the fork-join scheme with one master thread (*i.e.* a series of instructions executed consecutively) and working threads in the parallel region. More information about OpenMP can be found in Chapman *et al.* (2007). Advantages of OpenMP are simplicity since programmer does not need to deal with message passing (needed with MPI) and allows an incremental parallelism approach (user can choose on which part of the program OpenMP is used without dramatic changes inside the code). However, the main drawbacks are that it runs only in shared-memory multi-cores platforms and scalability is limited by memory architecture. Today most CFD calculations require more than one computing node hosting few PUs, meaning OpenMP is still not sufficient to parallelize efficiently a flow solver aimed for massive industrial and realistic problems.

5.2 Blocking and non-blocking MPI communications

Communications in AVBP and *elsA* can be implemented either through MPI blocking calls or MPI non-blocking calls. MPI non-blocking is a good way to reduce the communication time with respect to blocking MPI calls but it also induces a non-deterministic behaviour that participates to rounding errors (for example residuals are computed following a random order). To overcome this problem, a solution is to use MPI blocking calls with a scheduler. The objective is to manage the ordering of communication for minimizing the global communication time. Such a strategy has been tested in *elsA*. As shown in Fig. 15, the scheduling of blocking MPI communications is based on a heuristic algorithm that uses a weighted multi-graph. The graph vertices represent the available PUs and therefore an edge connects two cores. An edge also represents a connectivity that links two sub-domains. A weight is associated to each edge, depending on the number of connection interfaces (cell faces). Since many sub-domains (*i.e.* block in the case of *elsA*) can be assigned to only one PU, two vertices of the graph can be connected with many edges, hence the name multi-graph. The underlying principle of the ordering algorithm is to structure the communication scheme into successive message passing stages. Indeed, the communication scheme comes into play at some steps during an iteration

of the solver, whenever sub-domains need information from their neighbours. A message passing stage represents therefore a step where all PUs are active, which means that they are not waiting for other processes but are instead concerned by message passing. Each communication stage is represented by a list containing the graph edges that have no process in common. In other words, if one edge is taken from this list, then PUs related to the corresponding vertices will not appear in other edges. The load-balancing process must ensure that all PUs send nearly the same number of messages (with the same size) so the scheduler can assign the same number of communication stages to them.

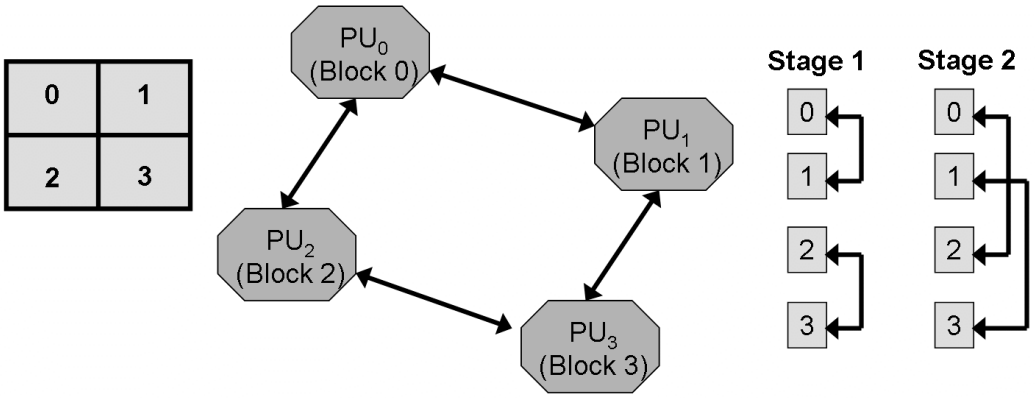


Fig. 15 Simplified example of a multi-graph used for the scheduling process in *elsA*.

Benchmarks have been performed with the structured flow solver *elsA* to quantify the influence of the MPI implementation on the point-to-point communications (coincident interfaces). In the current version of the flow solver, each block connectivity is implemented with the instruction "MPI_Sendrecv_replace()". This approach is compared with a non-blocking communication scheme that uses the "MPI_Irecv" and "MPI_Isend" instructions. Both implementations have been tested with and without the scheduler. The computational efficiency obtained with an IBM Blue Gene /L platform for a whole aircraft configuration (30M cells grid and 1,774 blocks) is indicated in Fig. 16. The reference used is the computational time observed with the standard blocking MPI instructions. As it can be seen, when the scheduler is used there is no difference in the efficiency with respect to the standard MPI implementation. However, without the scheduler, the computational efficiency with the blocking MPI implementations is reduced by 40% while no effect is observed with a non-blocking implementation.

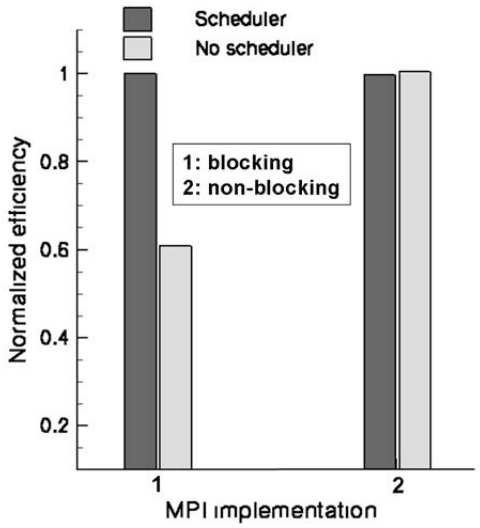


Fig. 16 Effect of the MPI implementation on the code efficiency (benchmarks are performed with *elsA* on an IBM Blue Gene /L platform).

These benchmarks clearly indicate that both implementations provide identical performance regarding the communication efficiency, at least for point-to-point communications. However, a scheduling step is required to ensure a good computing efficiency with MPI blocking calls. Particular care is necessary when the configuration is split to minimize the number of communications stages. The maximum degree of the graph used for scheduling is the minimum number of communication stages that may take place. It is thus important to limit the ratio between the maximum and minimum degrees of the graph vertices that indicates the rate of idleness of the computing core.

5.3 Point-to-point and collective MPI communications

Communications can also be done by using point-to-point communications and/or collective MPI calls. In the case of *elsA*, both methods are used: the first one for coincident interfaces and the second one for non-coincident interfaces. The strategy is thus to treat first the point-to-point communication and then the collective communications. As a consequence, if non-coincident interfaces are not correctly shared between all computing cores, a group of cores will wait for the cores interested by the collective MPI calls (it is also true for point-to-point communications). An example of the communications ordering obtained on an IBM Blue Gene /L system with the MPI Trace library is shown in Fig. 17. From left to right, time corresponds to one iteration of the flow solver (with an implicit scheme). The configuration is a multistage compressor that requires both coincident and non-coincident interfaces between blocks. The calculation has been performed with 4096 PUs but for clarity, data of Fig. 17 is presented only for 60 PUs (each line of the graph corresponds to one PU). The white region is related to the computation work while grey and black regions correspond to MPI calls. The collective communications are highlighted (in grey) and appear after the point-to-point ones (in black). The graph indicates how the management of these communications can reduce the parallel computing efficiency. MPI calls are related to the necessity for exchanging information, such as auxiliary quantities (gradients, etc.) at the beginning of the iteration or the increments of conservative variables ΔW_i during the implicit stages. At the end of the iteration, all the PUs exchange the conservative variables W and residuals R_W, using collective calls. In Fig. 17, two groups of processes are identified: group 1 is linked to coincident and non-coincident interfaces while group 2 is linked only to coincident interfaces. For example, when auxiliary quantities are exchanged, while group 1 is still doing collective MPI calls, group 2 has already started to compute fluxes. However during the implicit calculation, all processes have to be synchronized to exchange ΔW_i. This explains why the point-to-point communication related to group 2 appears so long: all the PUs associated to group 2 can not continue their work and are waiting for the PUs of group 1. In this example, collective MPI calls are not correctly balanced, increasing the communication time.

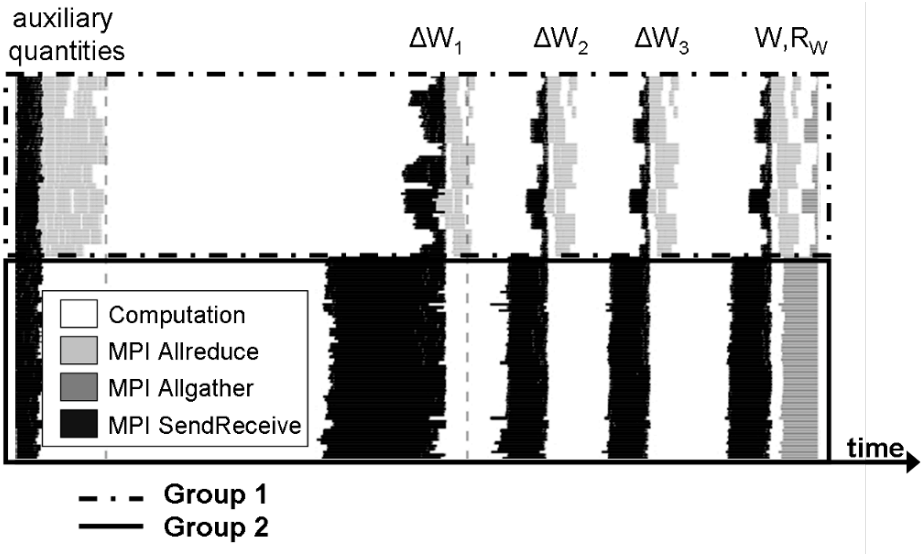


Fig. 17 Communication scheduling for one iteration with *elsA* on a Blue Gene /L system (N=4096 PUs) - each line corresponds to one PU, black is related to point-to-point communications and grey to collective communications.

In the unstructured flow solver AVBP, all communications between PUs are implemented through MPI non-blocking communications. AVBP does not use directly the syntax of the MPI library, but rather employs the Integrated Parallel Macros -IPM- library developed at CERFACS (Giraud, 1995) that acts as an intermediate layer to switch between different parallel message passing libraries. To highlight the role of communications, both point-to-point and collective calls have been tested. With point-to-point communication, all slave processes first send information to the master and then the master returns a message to all slave processes. It leads to the exchange of 2N messages (with N the total number of PUs). With collective calls communications, all processes have to exchange only 2.ln(N) messages. The second strategy minimizes the amount of information to exchange between computing cores through the network. Tests have been performed for a combustion chamber configuration up to 8192 PUs on a distributed shared memory supercomputer (SGI Altix, Harpertown cores). The grid of the test case is composed of 75 millions cells. The number of cells per computing core and the observed speedup are indicated in Fig. 18a and (resp.) Fig. 18b.

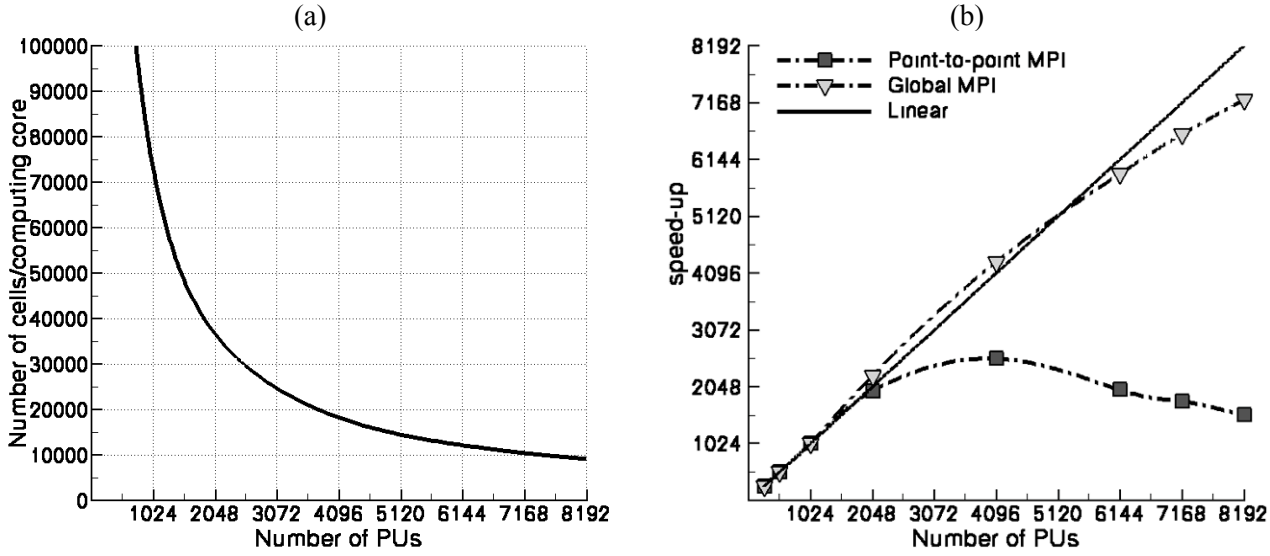


Fig. 18 Evolution of the number of cells by computing core (a) and effect of the communication strategy on the scalability (b) of AVBP (SGI Altix platform).

The communication strategy has a very small impact until 2048 PUs. When a higher number of PUs is used, the strategy based on point-to-point communication leads to a dramatic reduction of the parallel efficiency: the maximum speedup is observed with 4096 PUs before decreasing to 1500 with 8192 PUs. On the contrary, with a collective MPI communication, the speedup is almost linear (with 8192 PUs, the speedup is around 7200, *i.e.* an efficiency nearly 90%). With 2048 and 4096 PUs, the observed speedups are even higher than the linear law. This observation can be explained by the cache memory behaviour (PUs have a very fast access to data stored in a small private memory). When the problem is split into subtasks, the size of this data is reduced and it can thus be stored in this memory, explaining why calculations are sometimes more efficient than expected. Note also that AVBP shows good parallel performance with a very small number of cells per computing core (less than 10.000). To conclude, the previous benchmarks show the necessity of reducing the size and the number of communications, especially for some "sensitive" architectures. High-speed connections are also a crucial parameter for supercomputers when thousands of computing cores work concurrently.

6. Memory management

6.1 RAM requirements

With parallel computing, resources in terms of RAM memory can rapidly be a limiting factor. This observation is further emphasized by the fact that the global memory required by numerical simulations tends to increase with the number of PU used. Two strategies can be used. When it is possible, the first approach is

to compute most annex quantities (wall distance, connectivity, etc.) only one time before to store them. The main objective being to save computational time by increasing memory consumption thus requiring computing cored with large amount of memory. The second method is to compute most annex quantities at each iteration, which is interesting for code scalability (due to low memory consumption) but detrimental for computational time (at least when a sequential calculation is considered). The first method is used by *elsA*, while the second one is used by AVBP.

Due to the large number of libraries and code complexity, about 100 Mb of memory is needed for the executable file of *elsA*. In the case of a calculation with a large number of grid points (> 100 millions) and blocks (> 2000), the required memory could be as large as 300 Mb by computing core, before the first iteration of the solver! Implicit schemes are also available and used in *elsA* to reduce the computational time, requiring more memory than explicit schemes. For all these reasons, a multi-block flow solver such as *elsA* is not well adapted to supercomputers with a low distributed memory (<1Go/PU), especially if a limited number of computing cores is considered. The unstructured flow solver AVBP is implemented through a simpler architecture and only 10 Mb is enough to load the executable file. AVBP uses explicit schemes that require low memory. However, the storage of connectivity tables required for hybrid meshes leads to an increase of the memory consumption. The evolution of the consumed memory for both flow solvers during computations is indicated in Fig. 19 (in order to be as fair as possible, the memory required for mesh-partitioning is not taken into account). The memory consumption is not easy to compare directly since both flow solvers use different computing strategies. The configuration is either a combustion chamber with 10 millions cells (AVBP) either a turbine blade with 7 millions cells (*elsA*). In both cases the same numerical method is used (*i.e.* LES) and all simulations have been performed on a SGI Altix platform. For AVBP, results show that until 256 PUs, the slave process is the limiting factor while it becomes the master one with more than 256 PUs. The slope of the memory consumption is also different for both processes: while the memory needed by a slave process tends to rapidly decrease with the number of PUs (even if a plateau is observed after 1024 PUs), the memory dedicated to the master process follows a linear law with the number of PUs. For a very large number of PUs, it is clear that the master process will have a limiting effect. For *elsA*, the memory consumed by each computing core is related to block size, load balancing errors (more data can be stored on some computing cores) and connectivity (coincident/non-coincident interfaces). In practice, if the load balancing is correctly performed, all PUs require the same amount of memory (which is the case here). The effect of mesh partitioning on memory is also pointed out by Fig. 19: a configuration with fewer blocks is clearly less memory consuming (from 64PUs to 384PUs, the memory by PU is increased by +30%).

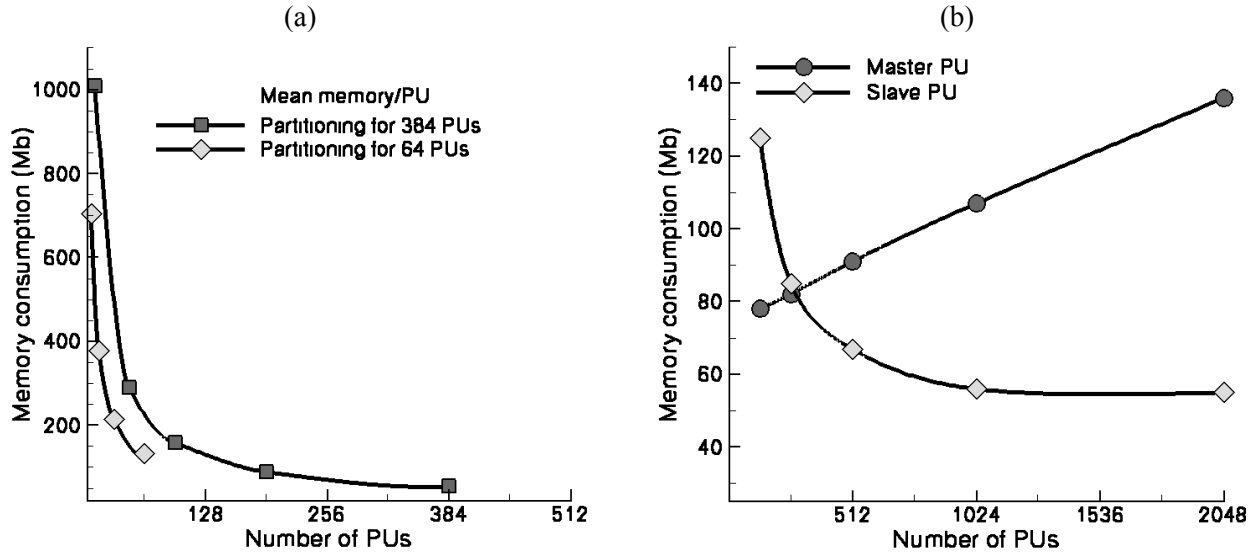


Fig. 19 Evaluation of the memory requirement by computing core with *elsA* (a) and AVBP (b).

6.2 Memory bandwidth

Another study has been done to measure the performance of the flow solvers regarding the memory bandwidth requirements. Benchmarks are performed on a distributed shared memory platform (SGI Altix, Harpertown cores) by using 1 to 8 PUs by computing node (the total number of PUs is kept constant). The normalized computational efficiency observed with the flow solvers *elsA* and AVBP are plotted in Fig. 20. Both flow solvers are affected by a reduction of performance when the number of cores by computing node is increased. AVBP shows a moderate decrease of the performance with 4 and 8 cores (-10% to -20%) while *elsA* exhibits a reduction from -15% (with 2 cores) to -50% (with 8 cores). This test shows that *elsA* is more sensitive to memory bandwidth than AVBP.

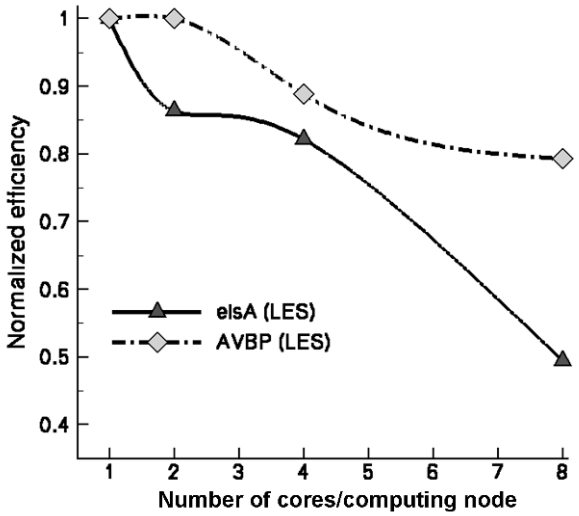


Fig. 20 Influence of the number of cores by computing node on the performance of the flow solvers *elsA* and AVBP (SGI Altix platform).

7. Impact on numerical solutions

Mesh-partitioning, node/block and communication ordering can affect the numerical solution. As previously mentioned, non-blocking MPI calls are responsible for non-deterministic behaviors and mesh-partitioning modifies the size of the initial problem. The effects of these features on rounding errors and implicit schemes are investigated in this section.

7.1 Mesh-partitioning and implicit algorithms

Implicit stage done on a block basis (such as in *elsA*) is a difficulty for mesh-partitioning. Block splitting performed to achieve good load balancing may reduce the numerical efficiency of the implicit algorithm. Simulations of the flow in a high pressure turbine have been performed (without block splitting) with different numbers of PUs, using RANS then LES. For both *elsA* tests, no difference is observed on the instantaneous solutions and convergence is strictly identical. A second set of simulation is performed by considering the same test case but with different mesh partitioning keeping an identical number of PUs. The unsteady RANS flow simulation is performed with *elsA* and results of tests are shown in Fig. 21. No significant difference on the convergence level between both configurations is observed until a normalized time of $t^+=10$. After $t^+=10$, although convergence levels are of the same order, differences are observed on the instantaneous residuals. In practice, no significant degradation on convergence level has ever been reported.

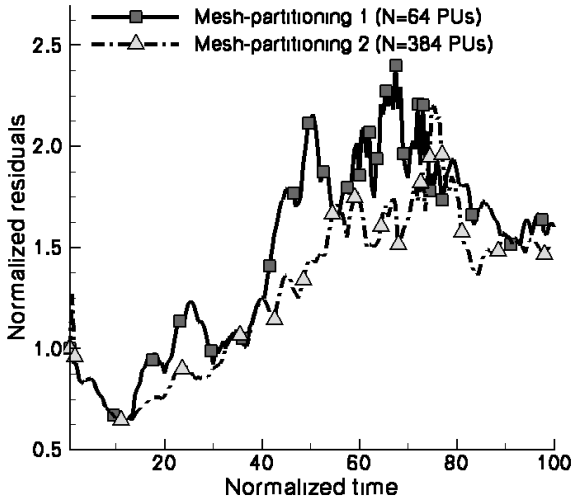


Fig. 21 Effect of mesh-partitioning on the convergence level with implicit schemes (unsteady flow simulation performed with *elsA*)

7.2 Rounding errors and LES

The effect of rounding errors is evaluated with AVBP for LES applications (Senoner *et al.,* 2008). Rounding errors are not induced only by parallel computing and computer precision is also a major parameter. However non-blocking MPI communications (such as used by AVBP) induces a difficulty: the arbitrary message arrival of variables to be updated at partition interfaces and the subsequent differences in the addition of the contributions of cell residuals at these boundary nodes is responsible for an irreproducibility effect (Garcia, 2003). A solution to force a deterministic behaviour is to focus on the reception of the overall contributions of the interface nodes (by keeping the arbitrary message arrival) and its posterior addition always in the same order. This variant of AVBP is used for error detection and debugging since it would be time and memory consuming in massively parallel machines. This technique is used to quantify the differences between solutions produced by runs with different node ordering for a turbulent channel flow test case, computed with LES. Mean and maximum norms are computed using the difference between instantaneous solutions obtained with two different node ordering and results are shown Fig. 22a. The difference between instantaneous solutions increases rapidly and reaches its maximum value after 100,000 iterations. The same test is performed for a laminar Poiseuille pipe flow, in order to show the role of turbulence, and results are compared with a fully developed turbulent channel flow in Fig. 22b. While instantaneous solutions for the turbulent channel diverge rapidly (even if statistical solutions remain identical), the errors for the laminar case grow only very slowly and do not exceed the value of 10^{-12} ($m.s^{-1}$) for the axial velocity component. This behaviour is explained by the fact that laminar flows do not induce exponential divergence of flow solution trajectories in time. On the contrary, the turbulent flow acts as an amplifier for rounding errors. The role of the architecture precision and the number of computing cores play an identical role. The use of high machine precisions such as double and quadruple precisions shows that the slope of the perturbation growth is simply delayed. The conclusion is that instantaneous flow fields produced by LES are partially controlled by rounding errors and depends on multiple parameters such as node reordering, machine precision and initial conditions. These results confirm that LES reflects the true nature of turbulence insofar as it may exponentially amplify very small perturbations. It also points out that debugging LES codes on parallel machines is a complex task since instantaneous results cannot be used to compare solutions.

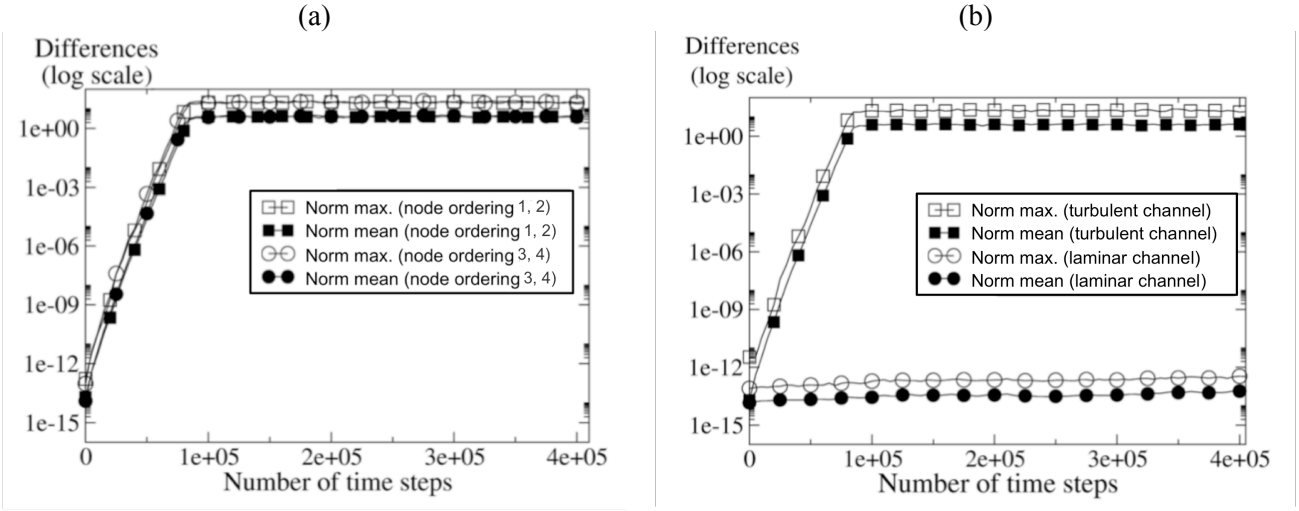


Fig. 22 Impact of rounding errors on numerical solution with AVBP: effect of node ordering (a) and turbulence (b).

8. Conclusion and further works

This paper has been devoted to programming aspects and the impact of HPC on CFD codes. The development and strategy necessary to perform numerical simulations on the powerful parallel computers have been presented. Two typical CFD codes have been investigated: a structured multi-block flow solver (*elsA*) and an unstructured flow solver (AVBP). Both codes are affected by the same requirements to run on HPC platforms. Due to the MPI-based strategy, a necessary step is to split the original configuration to define a new problem more adapted to parallel calculations (the work must be shared between all PUs). Mesh partitioning tools integrated in both flow solvers have been presented. Advantages and drawbacks of partitioning algorithms such as simple geometric methods (REB, RIB, etc.) or multilevel graph bisection methods (METIS) have been briefly discussed. However both flow solvers do not face the same difficulties.

The mesh partitioning stage with unstructured grids (AVBP) is very flexible and usually leads to correct load balancing among all PUs. The computational cost of this partitioning stage can become very important. For example, a 44M cell grid requires 25 minutes and 650 Mb of memory to be partitioned with the most efficient algorithm (k-way, METIS) on 4096 PUs. Other difficulties can appear when dealing with two-phase flow simulations (with particles seeding) for example, reducing the algorithm efficiency. For most applications, the method that provides the best parallel efficiency is the use of the RCM algorithm combined with a multi-level graph-partitioning algorithm (such as proposed in the METIS software). Compared to unstructured flow solvers, it is very difficult to achieve a good load balancing with structured multi-block meshes (*elsA*) but partitioning algorithms require very low computing resources in terms of memory and computational time. A 104M cells grid is partitioned on 2048 PUs in less than 3 minutes and requires only 100 Mb of memory with the greedy algorithm.

Another important aspect is related to the time spent for communication between PUs. Both flow solvers use the MPI library. The working scheme of AVBP is based on a master-slave model that requires intensive communications between slaves and master processes, especially for input/output. To be more efficient, MPI communications are managed in AVBP through non-blocking calls. However, benchmarks showed that the implementation of MPI communications (collective or point-to-point calls) is critical for scalability on some computing platforms. A last difficulty is that non-blocking calls add “random” rounding errors to other sources such as machine precision. The consequence is that simulations considering the LES method exhibit a non-deterministic behaviour. Any change in the code affecting the propagation of rounding errors will thus have a similar effect, implying that the validation of a LES code after modifications may only be based on statistical fields (comparing instantaneous solutions is thus not a proper validation method). A solution to

minimize these rounding errors (at least for parallel computations) is to use blocking calls with a scheduler based on the graph theory (such as implemented in the flow solver *elsA*). This method gives good results in terms of communication time, even if some difficulties still exist for the treatment of global communications (non-coincident interfaces) that arise after the point-to-point communications (coincident interfaces). If all non-coincident interfaces are not correctly shared between PUs, this strategy leads to load balancing errors.

To conclude, work is still necessary to improve the code performance on current (and future) computing platforms. First, a parallel implementation of input/output is necessary to avoid performance collapse on sensitive systems, especially for unsteady flow simulations. Second, implementation with MPI is complex and at very low level. A solution that uses both OpenMP (for communication inside nodes) and MPI (communication between nodes) is probably a way to explore for improving code scalability with multi-core nodes. Finally improvement of mesh partitioning and load balancing tools will be necessary (dynamic/automatic load balancing). For AVBP, works focus on the improvement of the communication methods between master and slave computing cores. For *elsA*, current works focus on the communication strategies (suppression of collective calls for non-coincident interfaces) and on the mesh partitioning algorithms (integration of a communication graph). Work will also be necessary to implement massively parallel capacities for specific functionalities such as the Chimera method. The impact of ghost cells on HPC capabilities should be better considered for the development of future flow solvers. Finally, flow solvers face new difficulties such as the problem of memory bandwidth, which is expected to be crucial with the advent of the last supercomputer generation that uses more and more cores by computing node. According to recent publications (Murphy, 2007; Moore, 2008), adding more cores per chip can slow data-intensive applications, even when computing in a sequential mode. Future high-performance computers have been tested at Sandia National Laboratories with computing nodes containing 8 to 64 cores that are expected to be the next industrial standard. Results were disappointing with conventional architecture since no performance improvement was observed beyond 8 cores. This fact is related to the so-called memory wall that represents the growing disparity between the CPU and data transfer speeds (memory access is too slow). Because of limited memory bandwidth and memory-management schemes (not always adapted to supercomputers), performance tends to decline with nodes integrating more cores, especially for data-intensive programs such as CFD flow solvers. The key for solving this problem is probably to obtain better memory integration to improve memory bandwidth. Other architectures such as graphic processors (GPU) are also a potential solution to increase the computational power available for CFD flow solvers. However, such architectures would impose major modifications in the code, more adapted programming language and optimized compilers.

Acknowledgement

The authors would like to thank Onera, Insitut Francais du Petrole (IFP) and development partners for their sustained effort to make *elsA* and AVBP successful projects. Authors are also grateful to teams of CERFACS involved in the management and maintenance of computing platforms (in particular the CSG group). Furthermore, the authors would like to acknowledge all people that contribute to this paper by means of discussions, support, direct help or corrections. Special thanks to industrial and research partners for supporting code developments and permission for publishing results. In particular, authors thank Airbus, Snecma and Turbomeca for collaborative work around *elsA* and AVBP projects. Finally, the authors acknowledge EDF, Météo-France, GENCI-CINES and computing centres for providing computational resources.

Bibliography

[01] Apte, S. V., Mahesh, K., and Lundgren, T. A, “Eulerian-Lagrangian Model to Simulate Two-Phase Particulate Flows”, Annual Research Briefs, Center for Turbulence Research, Stanford, USA, 2003.

[02] Berger, M. J. and Bokhari, S. H., “A Partitioning Strategy for Non-uniform Problems on Multiprocessors”, IEEE Trans. on Computers, Vol. 36, pp. 570-580, 1987.

[03] Cambier, L. and Veuillot, J-P., "Status of the elsA CFD Software for Flow Simulation and Multidisciplinary Applications", 46th AIAA Aerospace Science Meeting and Exhibit, Reno, USA, 2008.
[04] Cambier, L., and Gazaix, M., "*elsA*: an Efficient Object-Oriented Solution to CFD Complexity", 40th AIAA Aerospace Science Meeting and Exhibit, Reno, USA, 2002.
[05] Chapman, B., Jost, G., van der Pas, R. and Kuck, D. J., "Using OpenMP: Portable Shared Memory Parallel Programming", MIT Press In Scientific Computation and Engineering Series, ISBN 978-0262533027, Cambridge, USA, 2007.
[06] Cliquet, J., Houdeville, R., and Arnal, D., "Application of Laminar-Turbulent Transition Criteria in Navier-Stokes Computations", 45th AIAA Aerospace Science Meeting and Exhibit, Reno, USA, 2007.
[07] Colin, O. and Rudgyard, M. "Development of High-order Taylor-Galerkin Schemes for Unsteady Calculations", J. Comput. Phys., Vol. 162, pp. 338–371, 2000.
[08] Cuthill, E. and McKee, J., "Reducing the Bandwidth of Sparse Symmetric Matrices", in Proceedings of the 24th National Conference of the ACM, pp. 157-172, 1969.
[09] Deck, S. "Numerical Simulation of Transonic Buffet over a Supercritical Airfoil", AIAA J., Vol. 43, pp. 1556-1566, 2005.
[10] Davidson, L. "An Introduction to Turbulence Models", Publication 97/2, Chalmers University of Technology, Sweden, 2003.
[11] Donea, J., "Taylor-Galerkin Method for Convective Transport Problems", Int. J. Numer. Meth. Fluids Vol. 20, pp. 101-119, 1984.
[12] Fillola, G., Le Pape, M.-C., and Montagnac, M., "Numerical Simulations around Wing Control Surfaces", 24th ICAS meeting, Yokohama, Japan, 2004.
[13] Flynn, M., "Some Computer Organizations and Their Effectiveness", IEEE Trans. Comput., Vol. C-21, pp. 948, 1972.
[14] Garcia, M., "Analysis of precision differences observed for the AVBP code", Tech. Rep. TR/CFD/03/84, CERFACS, Toulouse, France, 2003.
[15] Garcia, M., "Développement et validation du formalisme Euler-Lagrange dans un solveur parallèle et non-structuré pour la simulation aux grandes échelles", PhD thesis, University of Toulouse, 2009.
[16] Garey, M. R. and Johnson, D. S. "Computers and Intractability: a Guide to the Theory of NP-completeness", W.H. Freeman & Company, ISBN 0716710455, 1979.
[17] Gazaix, M., Mazet, S. and Montagnac, M., "Large Scale Massively Parallel Computations with the Block-structured *elsA* CFD Software", 20th International Conference on Parallel CFD, Lyon, France, 2008.
[18] Giraud, L., Noyret, P., Sevault, E. and van Kemenade, V., "IPM – User's guide and reference manual", Technical Report TR/PA/95/01, CERFACS, Toulouse, France, 1995.
[19] Gourdain, N., Montagnac, M., Wlassow, F., and Gazaix, M., "High Performance Computing to Simulate Large Scale Industrial Flows in Multistage Compressors", Int. J. of High Performance Computing Applications, accepted in 2009, to be published.
[20] Gropp, W., Lusk, E., Skjellum, A., "Using MPI, 2nd Edition: portable Parallel Programming with the Message Passing Interface", MIT Press in Scientific and Engineering Computation Series, ISBN 978-0262571326, Cambridge, USA, 1999.
[21] Hendrickson, B. and Leland, R., "A Multilevel Algorithm for Partitioning Graphs", Tech. Rep. SAND93-1301, Sandia National Laboratories, Albuquerque, USA, 1993.
[22] Hirt, C. W., Amsden A. A. and Cook J. L. "An Arbitrary Lagrangian-Eulerian Computing Method for All Flow Speeds", J. of Computational Physics, Vol. 14, p. 227, 1974.
[23] Jameson, A., "Time Dependent Calculations Using Multigrid, with Applications to Unsteady Flows Past Airfoils and Wings", 10th AIAA Computational Fluid Dynamics Conference, paper 91-1596, 1991.
[24] Karypis, G. and Kumar, V., "Multilevel Algorithms for Multi-Constraint Graph Partitioning", Tech. Rep. 98-019, University of Minnesota, Department of Computer Science/Army, HPC Research Center, USA, 1998.
[25] Karypis, G. and Kumar, V. "A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs". SIAM J. on Scientific Computing, Vol. 20, pp. 359-392, 1999.
[26] Keyes, D. E., Kaushik, D. K. and Smith, B. F., "Prospects for CFD on Petaflops Systems", Technical Report TR-97-73, Institute for Computer Applications in Science and Engineering, 1997.

[27] Lax, P. D. and Wendroff, B., "Systems of Conservation Laws", Commun. Pure Appl. Math., Vol. 13, pp. 217-237, 1960.
[28] Lewis, J. G., "The Gibbs-Poole-Stockmeyer and Gibbs-King Algorithms for Reordering Sparse Matrices", ACM Trans. Math. Software, Vol. 8, pp. 190-194, 1982.
[29] Liou, M. S., "A sequel to AUSM: AUSM+", J. of Computational Physics, Vol.129, pp.364-382, 1996.
[30] Meakin, R., "The Chimera Method of Simulation for Unsteady Three-Dimensional Viscous Flow", Computational Fluid Dynamics Review, pp. 70-86, 1995.
[31] Moin, P. and Apte, S., "Large Eddy Simulation of Realistic Turbine Combustors", AIAA J., Vol. 44, pp. 698-708, 2006.
[32] Moore, S. K., "Multicore Is Bad News For Supercomputers", Online article, http://www.spectrum.ieee.org/nov08/6912, 2008.
[33] Murphy, R., "On the Effects of Memory Latency and Bandwidth on Supercomputer Application Performance", IEEE 10th International Symposium on Workload Characterization, Boston, USA, 2007.
[34] Roe, P. L., "Approximate Riemann Solvers, Parameter Vectors and Difference Schemes", J. of Computational Physics, Vol. 43, pp. 357-372, 1981.
[35] Senoner, J-M., Garcia, M., Mendez, S., Staffelbach, G., Vermorel, O. and Poinsot, T., "Growth of Rounding Errors and Repetitivity of Large-Eddy Simulations", AIAA J., Vol. 46, pp. 1773-1781, 2008.
[36] Simon, H. D., "Partitioning of Unstructured Problems for Parallel Processing", Computing Systems in Engineering, Vol. 2, pp. 135-148, 1991.
[37] Smagorinsky, J. S., "General Circulation Experiments with the Primitive Equations: I. the Basic Experiment", Mon. Weather Rev., Vol. 91, pp. 99-163, 1963.
[38] Spalart, P. R., Jou, W.-H., Stretlets, M., and Allmaras, S. R., "Comments on the Feasibility of LES for Wings and on the Hybrid RANS/LES Approach", Advances in DNS/LES, Proceedings of the First AFOSR International Conference on DNS/LES, 1997.
[39] Williams, R. D., "Performance of Dynamic Load Balancing Algorithms for Unstructured Mesh Calculations", Concurrency: Practice, and Experience, Vol. 3, pp. 451-481, 1991.
[40] Yoon, S. and Jameson, A., "An LU-SSOR Scheme for the Euler and Navier-Stokes Equations", 25th AIAA Aerospace Sciences Meeting, paper 87-0600, Reno, USA, 1987.
[41] Ytterström, A., "A Tool for Partitioning Structured Multiblock Meshes for Parallel Computational Mechanics", Int. J. of High Performance Computing Applications, Vol. 11, pp. 336-343, 1997.