CFD Parallel Simulation Using Getfem++ and Mumps

Michel Fournié[1], Nicolas Renon[2], Yves Renard[3], and Daniel Ruiz[4]

[1] Institut de Mathématiques de Toulouse, CNRS (UMR 5219), Université de Toulouse, France
[2] Centre de Calcul Inter Universitaire de Toulouse (CICT-CALMIP), France
[3] Institut Camille Jordan, CNRS (UMR 5208), INSA Lyon, France
[4] Institut de Recherche en Informatique de Toulouse, CNRS (UMR 5505), Université de Toulouse, France

Abstract. We consider the finite element environment Getfem++[1], which is a C++ library of generic finite element functionalities and allows for parallel distributed data manipulation and assembly. For the solution of the large sparse linear systems arising from the finite element assembly, we consider the multifrontal massively parallel solver package Mumps[2], which implements a parallel distributed LU factorization of large sparse matrices. In this work, we present the integration of the Mumps package into Getfem++ that provides a complete and generic parallel distributed chain from the finite element discretization to the solution of the PDE problems. We consider the parallel simulation of the transition to turbulence of a flow around a circular cylinder using Navier Stokes equations, where the nonlinear term is semi-implicit and requires that some of the discretized differential operators be updated and with an assembly process at each time step. The preliminary parallel experiments using this new combination of Getfem++ and Mumps are presented.

1 Introduction

In many applications, the solution of PDE problems with discretization using very fine grid meshes is required, as in the simulation of turbulent fluid flows or in the simulation of lightnings in plasmas, for example. The amount of unknowns in such cases can reach levels that make it difficult to hold the data in the memory of a single host and distributed computing may become intrinsically necessary.

In the following, we are interested in the particular case of the simulation of non-stationary turbulent flows, 2D or 3D, using Navier-Stokes equations where the nonlinear term is semi-implicit and requires that some of the discretized differential operators be updated with an assembly process at each time step. We exploit therefore the finite element environment Getfem++, which is a C++ library of generic finite element functionalities, that enables to express in a simple

[1] http://home.gna.org/Getfem/
[2] http://Mumps.enseeiht.fr/

way the finite element discretization of PDE problems, including various types of 2D or 3D finite elements, and that can be used for many PDE applications. Additionally, Getfem++ allows for parallel distributed data manipulation and assembly, and this is one of the key features to address appropriately the type of problems that we are interested in.

For the solution, at each time step, of the large sparse linear systems arising from the finite element assembly, we also use the multifrontal massively parallel solver package Mumps, which implements a parallel distributed LU factorization of large sparse matrices. Since Mumps enables to specify, as input, an already distributed sparse linear system, it was easy for us to integrate the Mumps package into Getfem++ in order to design a complete and generic parallel distributed chain from the finite element discretization to the solution of the PDE problems. This is developed in more details in sections 2 and 3. The main contribution of this work is to propose a coupling between Getfem++ and Mumps in an advantageous way.

We introduce the non-stationary turbulent flow problem in section 4. The results are presented in section 5, where we try to analyze the potential and limitations for parallelism of this general computational chain made of Getfem++ and Mumps together. We briefly discuss the identified bottlenecks and possible evolutions of this platform for improved performances.

2 The Getfem++ Library

Getfem++ (Generic Toolbox for Finite Element Methods in C++) is a finite element library, freely distributed under the terms of the Gnu Lesser General Public License (LGPL license).

It aims at providing some standard tools for the development of finite element codes for PDEs, and in particular:

1. proposing a generic management of meshes: arbitrary dimension, arbitrary geometric transformations,
2. providing some generic assembling methods,
3. implementing many advanced methods (mixed methods, mortar elements, hierarchical elements, X-FEM,...), and simplify the addition of new methods,
4. interpolation methods, computation of norms, mesh operations (including automatic refinement), boundary conditions,
5. proposing a simple interface under Matlab© and Python thus giving a possibility to use Getfem++ without any knowledge of C++,
6. post-processing tools such as extraction of slices from a mesh ...

Getfem++ can be used to build very general finite elements codes, where the finite elements, integration methods, dimension of the meshes, are just some parameters that can be changed very easily, thus allowing a large spectrum of experimentations with arbitrary dimension i.e. not only 2D and 3D problems (numerous examples are available in the tests directory of the distribution).

Getfem++ has no meshing capabilities (apart from regular meshes), hence it is necessary to import meshes. Import formats currently known by Getfem++

are GiD[3], Gmsh[4] and emc2 mesh files. However, given a mesh, it is possible to refine it automatically.

Getfem++ has been awarded the second price at the "Trophées du libre 2000" in the category of scientific softwares[5].

2.1 The Model Description

Getfem++ provides a model description which allows to quickly build some finite element method applications on complex linear or nonlinear PDE coupled models. The basic idea is to define generic bricks which can be assembled to describe a complex situation. A brick can describe either an equation (Poisson equation, linear elasticity ...) or a boundary condition (Dirichlet, Neumann ...) or any relation between two variables. Once a brick is written, it is possible to use it in very different ways. This allows to reuse the generated code and to increase the capability of the library. A particular effort has been paid to ease as much as possible the development of a new brick. A brick is mainly defined by its contribution to the final linear system to be solved.

Here, we present some command lines used for a complete Poisson problem. After defining a `mesh` with boundary `DirichletBoundaryNum`, finite element methods `mfU` (Pk-FEM or Qk-FEM $\cdots$ for the unknowns) and `mfRhs` (for the right hand sides), an integration method `mim`, we can use the following main code.

```
getfem::model laplacian_model; // Main unknown of the problem
laplacian_model.add_fem_variable("u", mfU); // Laplacian term on u
getfem::add_Laplacian_brick(laplacian_model, mim, "u");

// Dirichlet condition.
gmm::resize(F, mfRhs.nb_dof());
getfem::interpolation_function(mfRhs, F, sol_u);
laplacian_model.add_initialized_fem_data("DirichletData", mfRhs, F);
getfem::add_Dirichlet_condition_with_multipliers(laplacian_model, mim, "u",
                              mfU, DirichletBoundaryNum, "DirichletData");

// Volumic source term and Neumann conditions are defined in the same way

// Solve linear system
gmm::iteration iter(residual, 1, 40000);
getfem::standard_solve(laplacian_model, iter);
std::vector<scalar_type> U(mfU.nb_dof());
gmm::copy(laplacian_model.real_variable("u"), U); // solution in U
```

The `standard_solve` command is linked to a solver method that can be an iterative one (using `iter`) or Mumps library (see section 3) when the installation is on parallel machine.

[3] http://gid.cimne.upc.es/
[4] http://www.geuz.org/gmsh/
[5] http://www.trophees-du-libre.org/

2.2 Parallelization under Getfem++

It is not necessary to parallelize all steps in the code, however the brick system offers a generic parallelization based on MPI (communication between processes), ParMetis[6] (partition of the mesh) and Mumps (parallel sparse direct solver). In this way, each mesh used is implicitly partitioned (using ParMetis) into a number of regions corresponding to the number of processors and the assembly procedures are parallelized. This means that the matrices stored are all distributed. In the current version of Getfem++, the right hand side vectors of the considered problem are transferred to each processor (the sum of each contribution is made at the end of the assembly and each MPI process holds a copy of the global vector). This aspect introduces some memory limitation but provides a simple generic parallelization with little effort. This can however be improved in further developments. Finally I/O have been parallelized too in order to prevent any sequential stage (avoid significant bottleneck when storing partial results during the simulations of large scaled problems).

2.3 Linear Algebra Procedures

The linear algebra library used by Getfem++ is Gmm++ a generic C++ template library for sparse, dense and skyline matrices. It is built as a set of generic algorithms (mult, add, copy, sub-matrices, dense and sparse solvers ...) for any interfaced vector type or matrix type. It can be viewed as a glue library allowing cooperation between several vector and matrix types. However, basic sparse, dense and skyline matrix/vector types are built in Gmm++, hence it can be used as a standalone linear algebra library. The library offers predefined dense, sparse and skyline matrix types. The efficiency of this library is given on the web site http://grh.mur.at/misc/sparselib_benchmark/. Getfem++ proposes classical iterative solvers (CG, GMRES, ...) and includes its own version of SuperLU 3.0. An interface to Mumps is provided to enable simple parallelization.

3 Mumps Library

Mumps ("MUltifrontal Massively Parallel Solver") is a package for solving systems of linear equations of the form $Ax = b$, where A is a general square sparse matrix. Mumps is a direct method based on a multifrontal approach which performs a direct factorization $A = LU$ or $A = LDL^T$ depending on the symmetry of the matrix. Mumps exploits both parallelism arising from sparsity in the matrix A and from dense factorization kernels. One of the key features of the Mumps package for the coupling with Getfem++ is the possibility to input the sparse matrices in distributed assembled or elemental format. This enables us to keep the already assembled and distributed matrices created under Getfem++ in place on the different processors and to call Mumps for solution without any data exchange or communication in between.

[6] http://glaros.dtc.umn.edu/gkhome/metis/parmetis/

Mumps offers several built-in ordering packages and a tight interface to some external ordering packages. The software is written in Fortran 90, and the available C interface has been used to make the link with Getfem++. The parallel version of Mumps requires MPI [7] for message passing and makes use of the BLAS [4], BLACS, and ScaLAPACK [1] libraries.

The system $Ax = b$ is solved in three main steps:

1. Analysis. An ordering based on the symmetrized pattern $A+A^T$ is computed, based on a symbolic factorization. A mapping of the multifrontal computational graph is then computed, and the resulting symbolic information is used by the processors to estimate the memory necessary for factorization and solution.
2. Factorization. The numerical factorization is a sequence of dense factorizations on so called frontal matrices. Task parallelism is derived from the elimination tree and enables multiple fronts to be processed simultaneously. This approach is called multifrontal approach. After the factorization, the factor matrices are kept distributed. They will be used at the solution phase.
3. Solution. The right-hand side b is broadcast from the host to the working processors that compute the solution x using the distributed factors computed during factorization. The solution is then either assembled on the host or kept distributed on the working processors.

Each of these phases can be called separately and several instances of Mumps can be handled simultaneously. Mumps implements a fully asynchronous approach with dynamic scheduling of the computational tasks. Asynchronous communication is used to enable overlapping between communication and computation. Dynamic scheduling was initially chosen to accommodate numerical pivoting in the factorization. The other important reason for this choice was that, with dynamic scheduling, the algorithm can adapt itself at execution time to remap work and data to more appropriate processors. In fact, the main features of static and dynamic approaches are combined, where the estimation obtained during the analysis is used to map some of the main computational tasks, and where the other tasks are dynamically scheduled at execution time. The main data structures (the original matrix and the factors) are similarly partially mapped during the analysis phase.

4 Navier-Stokes Simulation to Steady the Transition to Turbulence in the Wake of a Circular Cylinder

The transition to turbulence of the flow around a circular cylinder is studied by a two and a three-dimensional numerical simulation of the Navier-Stokes equations system. This simulation is a problem which has been extensively investigated and is still a challenge for high Reynolds number $Re = \frac{1}{\nu}$ when a rotation of the cylinder is considered. The numerical method used is based on splitting scheme and Neumann boundary conditions are imposed at the two boundaries in the

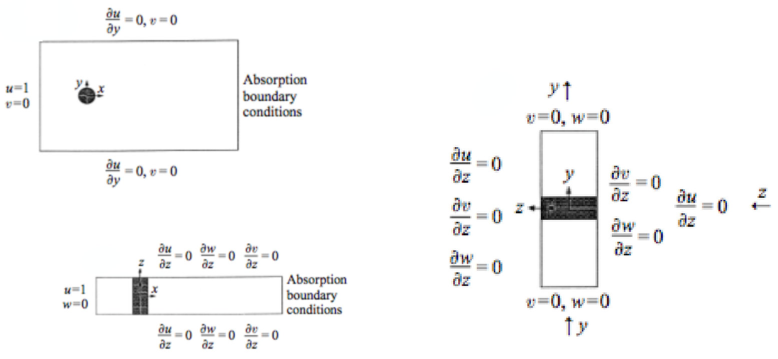
∂u/∂y = 0, v = 0
u=1
v=0
Absorption boundary conditions
∂u/∂y = 0, v = 0
∂u/∂z = 0 ∂w/∂z = 0 ∂v/∂z = 0
u=1
w=0
Absorption boundary conditions
∂u/∂z = 0 ∂w/∂z = 0 ∂v/∂z = 0
v=0, w=0
∂u/∂z = 0
∂v/∂z = 0
∂w/∂z = 0
∂v/∂z = 0
∂w/∂z = 0
∂u/∂z = 0
v=0, w=0

Fig. 1. Boundary conditions for the velocity components

spanwise direction and non-reflecting boundary conditions are specified for the outlet downstream boundary [6] see Figure 1.

The profile of the flow depends on Re. For example, without rotation of the cylinder, for $Re = 200$, the flow becomes unsymmetric and unsteady and eddies are shed alternatively from the upper and lower edges of the disk in a time periodic fashion. This trail of vortices in the wake known as the Karman vortex street is illustrated in Figure 2. The qualitative results obtained are not presented in this paper.

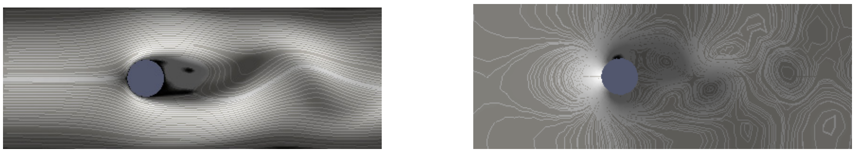

Fig. 2. Representation of the Navier-Stokes solution. On the left : Velocity u (norm, streamlines). On the right : Pressure p (iso-value).

4.1 Time Discretization

We consider the time-dependent Navier-Stokes equations written in terms of the primitive variables, namely the velocity $\mathbf{u}$ and the pressure p (ν is the viscosity) on a finite time interval $[0, T]$ and in domain $\Omega \subset \mathbb{R}^2$ or $\mathbb{R}^3$,

$$\begin{cases} \dfrac{\partial \mathbf{u}}{\partial t} - \nu \Delta \mathbf{u} + (\mathbf{u}.\nabla)\mathbf{u} + \nabla p = 0 \text{ on } \Omega \times (0, T), \\ \operatorname{div}(\mathbf{u}) = 0 \text{ on } \Omega \times (0, T), \\ \mathbf{u}(t = 0) = \mathbf{u}_0 \text{ and Boundary conditions on } \mathbf{u}. \end{cases} \tag{1}$$

We consider the incremental pressure-correction scheme which is a time-marching technique composed of sub-steps [5,2].

Let $\Delta_t > 0$ be a time step and set $t^k = k\Delta_t$ for $0 \leq k \leq K = T/\Delta_t$. We denote $\mathbf{u^n}$ and p^n approximate solutions of the velocity and the pressure at the time t^n. For given values of the velocity $\mathbf{u^n}$ and the pressure p^n, we compute an approximate velocity field $\mathbf{u}^*$ by solving the following momentum equations at t^{n+1} (with the same boundary conditions as $\mathbf{u}$)

$$\frac{\mathbf{u}^* - \mathbf{u}}{\Delta_t} + (\mathbf{u^n}.\nabla)(\mathbf{u}^*) = -\nabla p^n + \nu\Delta\mathbf{u}^*. \tag{2}$$

Notice that the incompressibility condition is not satisfied by $\mathbf{u}^*$, so we perform a projection step (onto a space of divergence free functions), by introducing an auxiliary potential function Φ solution of the Poisson equation (with homogenous boundary condition)

$$\nabla.\mathbf{u}^* = \Delta\Phi. \tag{3}$$

Then the true velocity is obtained by

$$\mathbf{u^{n+1}} - \mathbf{u}^* = -\nabla\Phi \tag{4}$$

and the pressure is given by

$$p^{n+1} = p^n + \frac{\Phi}{\Delta_t}. \tag{5}$$

The nonlinear term in (2) is treated with a semi-implicit discretization of the nonlinear term which implies to assemble this term at each time iteration. As opposed to the initial formulation of the problem the splitting scheme presents the advantage to uncouple the velocity and the pressure. This point implies that the dimension of the matrices appearing in the linear systems to be solved is reduced.

4.2 Space Discretization Using Getfem++

To define a weak formulation of the problem, we multiply the equations (2), (3) by a test function $\mathbf{v}$ and the equation (4) by a test function q belonging to a suitable space V and Q respectively. The spaces are chosen in such a way that the test functions vanish on the boundary position where a Dirichlet data is prescribed on $\mathbf{u}$ and we assume that p be average free. Then, we realize a Galerkin approximation. We consider two families of finite dimensional subspaces V_h and Q_h of the spaces V and Q, respectively. We refer to [5] for more details on those spaces. We consider finite element piecewise polynomial of degree 2 for the velocity and 1 for the pressure. More precisely, we consider the compatible couple of spaces Q2-Q1 (on quadrangular mesh) such that the Babuska-Brezzi condition be satisfied (inf-sup condition) [3].

5 Parallel Experiments

Computations were run on the CALMIP SGI Single System Image Supercomputer Altix 3700 with 128 Itanium 2processor (1,5 Ghz, 6 MB cache L3, single-core) and 256 GB RAM (one single address space). The Altix 3700 is a well known ccNUMA parallel architecture which can be efficient to run parallel applications. CALMIP stands for "Computations in Midi-Pyrénées" and has been founded in 1994 by 17 Research Laboratories of Toulouse University to promote, at the regional scale, the use of new parallel computing technologies in the research community.

The SGI Altix system is used through a batch scheduler system, namely Portable Batch System (PBSpro). The policy of the scheduler does not take into account spatial locality or affinity of the resources (cpus). However, we tried, as far as we could, to run jobs during periods of low charge of the whole system, in order to help locality between processes, but without any guarantee. Nevertheless we always used the command "*dplace*" to avoid undesired migration of MPI processes by the operating system. Installation of MUMPS on the SGI Altix System was made with SGI's Message Passing Toolkit v11 for MPI, SGI's Scientific Computing Software Library (SCSL) and SGI's Scientific Computing Software Library routine for Distributed Shared Memory (SDSM) for BLAS, BLACS and ScaLAPACK.

We must mention beforehand that Getfem++ is a C++ library, and since C++ codes suffer partly from the performance of the Itanium processor with the actual version of the compiler, the timings that we shall present might be improved. Still, our main purpose is to analyse the potential and limitations of the combination of Getfem++ with Mumps as a complete parallel platform for numerical simulation.

The integration of Mumps into Getfem++ naturally exploits the matrix distributed sparse format used in Getfem++. It was just necessary to transform the native compressed sparse columns format used in Getfem++ to the required general distributed sparse format in Mumps. This is done independently on each processor with the local data. As discussed in Section 2, Getfem++ holds so called "*global vectors*", that are in fact duplicated and synchronized on each processor as a generic and easy way to share data in parallel. Mumps inputs the right hand side vectors from the master processor, and may return distributed solutions. However, since these solutions are needed to build, in the time loop, the next right-hand sides, the solution vectors are therefore gathered onto the master processor. Nevertheless, to follow the requirements for data parallelism in Getfem++, we needed to declare both the right hand side and solution vectors as "*global vectors*". In that way, interfacing Mumps with Getfem++ was straightforward, but the many synchronizations implied by this handling of global vectors may be a bottleneck for efficiency. At any rate, we have incorporated these global data synchronizations in the various timings that are presented in the following. Further improvements of Getfem++ parallel platform may be obtained with alternative approaches for generic data exchanges.

As presented in section 4, the Navier-Stokes simulation involves a time-loop where the three sparse linear systems arising from (2), (3), (4) have to be solved at each time step, and with one of these three systems that needs (due to the nonlinear term) to be reassembled and factorized systematically. The other two systems need to be factorized only once, at the first iteration in time, and the three systems need to be analyzed only once since their sparsity structure remains unchanged. In Figure 3, where timings are shown, we indicate the average time spent in one time-loop (excluding the particular case of the first loop where extra computation is performed). We superimpose the elapsed time spent strictly in the Mumps specific routines (shown in dark red - bottom) with the global elapsed time per time-iteration (in light yellow - top). Speedups are given in terms of elapsed time after synchronization at the boundaries between the Getfem++ code and Mumps.

We have run experiments for a 2D simulation with three different mesh sizes, one of 450K nodes, a medium one of 900K nodes, and a larger one of 1.8M nodes (the nodes numbers correspond to the velocity unknowns), and we varied the number of processors from 4 to 64 (maximum current user limit in the PBS queue management system). The generic loop in time involves, as already said, one sparse matrix FEM assembly for the system associated with equation (2), performed in parallel with the Getfem++ functions, and one factorization with Mumps for the same system, plus three solutions for the linear systems associated with equations (2), (3), and (4). Equation (5) involves a simple linear combination of vectors and is performed with a collective and reduction operation (MPI_Allreduce). We mention that some little and marginal extra operations are performed at each time-loop to recover lift and drag coefficients useful for qualitative analysis of the properties of the flow simulation. It appears, from the timings shown in these figures, that the matrix re-assembly benefits directly from parallel computation, since we can observe in all cases substantial CPU-time reduction for the operations excluding Mumps specific ones, illustrated by the reduction of the dark red part in the bar charts with increasing numbers of processors. The Mumps operations, which involve three solutions for one factorization only, benefit much less from parallelism except when the size of the matrices become sufficiently large so that the computations are mostly dominated by the factorization of the system arising from (2). This can be easily explained by the fact that the granularity of a solution phase in Mumps is much less than that of the factorization phase, and the benefit from different levels of parallelism (sparsity structure plus BLAS kernels) is much less in the solution

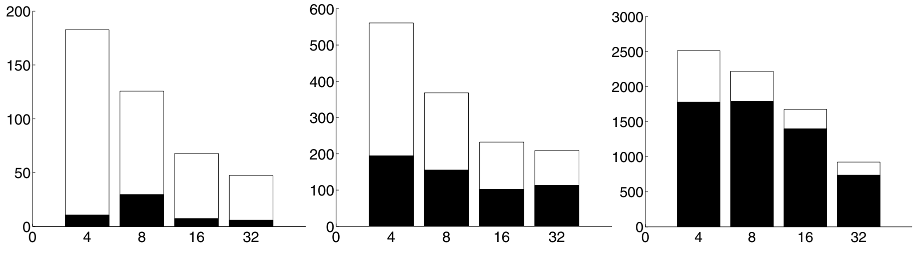

Fig. 3. Average timings with respect to the number of processors. Left : 450K nodes - Center : 900K nodes - Right : 1.8M nodes. Black : Mumps part - White : outside Mumps.

phase in general. Additionally, in the case of the largest dimensional experiment, the time spent in the Mumps operations specifically becomes dominant with respect to the rest. This particular observation is favorable to the situation of larger simulations (3D cases) where we may expect indeed that the gains in parallelism of the code will be mostly driven by the parallelism in the Mumps factorization.

Tables 1, 2, 3 give details about the computational timings in the different parts of the code, as well as some speed up information to estimate the degree of parallelism that can be achieved in each sub-part and in the combination of Mumps plus Getfem++. It is clear that effort has to be paid to improve in particular the parallelism in the solution phase of Mumps. This is under current investigation, and we shall benefit from the future releases of Mumps. The case of 4 processors shows better performances with respect to the others. This might be explained by the architecture of the Altix machine, but we do not have the explicit clue for this.

To get better understanding of these timings and speed-ups, we detail in Table 4 the contribution of the various subparts in the time-loop to the total time, including thus the timing for the assembly at each time step of the nonlinear term for the velocity, the extra computations for the construction of the systems themselves (including right hand sides), the timing for the factorization plus solution with Mumps of the system associated to equation (2), and the timings for the solution only of the systems linked to (3) and (4). The total computational time in Mumps additionally includes the extra operations when chaining the results of one equation to the

Table 1. Timings and Speed Ups for 450K nodes test case

Number of Processors	4	8	16	32
Time in Mumps operations	10.7	29.7	7.5	6
Speed Up in Mumps part	1	0.36	1.43	1.78
Time spent outside Mumps	172	96	60.3	41.4
Speed Up in ops. outside Mumps	1	1.79	2.85	4.16
Total Time per Iteration	182.7	125.7	67.8	47.4
Global Speed Up per Iteration	1	1.45	2.7	3.85

Table 2. Timings and Speed Ups for 900K nodes test case

Number of Processors	4	8	16	32
Time in Mumps operations	195.6	155.3	102.4	113.2
Speed Up in Mumps part	1	1.26	1.91	1.72
Time spent outside Mumps	365.3	212.8	129.9	96
Speed Up in ops. outside Mumps	1	1.72	2.81	3.80
Total Time per Iteration	560.9	368.1	232.3	209.2
Global Speed Up per Iteration	1	1.52	2.41	2.68

Table 3. Timings and Speed Ups for 1.8M nodes test case

Number of Processors	4	8	16	32
Time in Mumps operations	1786.8	1800.1	1400.6	739
Speed Up in Mumps part	1	0.99	1.28	2.42
Time spent outside Mumps	732.3	429.5	275.7	185.8
Speed Up in ops. outside Mumps	1	1.7	2.66	3.94
Total Time per Iteration	2519.1	2229.6	1676.3	924.8
Global Speed Up per Iteration	1	1.13	1.5	2.72

other, as indicated in section 4.1. We also give the time when summing and broadcasting the *global vectors* that correspond to the velocity and pressure vectors, and to the right hand sides for the three systems involved at each iteration in time. We have just extracted the timings for the medium size test case, on 4 and 16 processors, in order to get information about the relative weights of the different parts in the total computational timings. From these results we observe that the time for the extra computations are one of the main bottleneck. Future improvements should be brought in priority to that part but we still need to investigate in deeper details the mechanisms of data exchanges involved. The second axis for improvement may also be in a finer integration of Mumps in Getfem++, and in particular in the development of specific functionalities to address wether distributed right hand sides or distributed solution vectors.

We must mention that Mumps did not encounter specific numerical problems with the various systems involved. Indeed, the estimated number of entries for factors and the actual number on entries after factorization do not differ more than 1 or 2 percent in all cases. The maximum frontal sizes are 1457 for systems in equations (2) and (4), and 558 for system in equation (3), and the number of delayed pivots are 5760 and 0 respectively (for indication, the number of operations during node elimination are $3.135\,10^{10}$ and $1.432\,10^{9}$ respectively).

Futures developments will be concerned, as already mentioned, with the improvement of the parallelization in general and in particular the data exchanges. This may require new features such as sparse distributed right hand side and

Table 4. Detailed timings for 900K nodes test case on 16 processors

Number of Processors	4	16	Speed-Ups
Assembly of nonlinear term	284.7	72.2	3.94
Extra computations (sum and Broadcast)	80.6(0.89)	57.7(1.04)	1.4
Mumps timings for facto + solve of (2)	182.4	94.7	1.92
Mumps timings for solution of (3)	0.15	0.092	1.63
Mumps timings for solution of (4)	1.74	0.87	2
Time in Mumps operations	195.6	103.3	1.89
Time spent outside Mumps	365.3	129.9	2.81
Total Time per iteration	560.9	233.2	2.4

solution vectors in Mumps. Besides that, we shall improve the memory allocation in Getfem++ with a similar approach as for the sparse distributed vectors in Mumps. Finally we shall experiment with much larger 3D Fluids Dynamics test cases and with larger numbers of processors available on the new massively parallel machine in Toulouse (that will replace the Altix machine). We hope that these future developments will help to ensure some good scalability to address challenge problems.

References

1. Blackford, L.S., Choi, J., Cleary, A., D'Azevedo, E., Demmel, J., Dhillon, I., Dongarra, J., Hammarling, S., Henry, G., Petitet, A., Stanley, K., Walker, D., Whaley, R.C.: Scalapack users' guide (1997)
2. Braza, M., Persillon, H.: Physical analysis of the transition to turbulence in the wake of a circular cylinder by three-dimensional navier-stokes simulation. J. Fluid Mech. 365, 23–88 (1998)
3. Brezzi, F., Fortin, M.: Mixed and Hybrid finite element methods. Springer, Heidelberg (1991)
4. Dongarra, J.J., Croz, J.D., Duff, I.S., Hammarling, S.: Algorithm 679. a set of level 3 basic linear algebra subprograms. ACM Transactions on Mathematical Software 16, 1–17 (1990)
5. Ern, A., Guermond, J.L.: Theory and Practice of Finite Elements. Applied Mathematical Series, vol. 159. Springer, Heidelberg (2004)
6. Jin, G., Braza, M.: A non-reflecting outlet boundary condition for incompressible unsteady navier-stokes calculations. J. Comput. Phys. 107, 239–253 (1993)
7. Snir, M., Otto, S.W., Huss-Lederman, S., Walker, D.W., Dongarra, J.: Mpi: The complete reference (1996)