Parallel adaptive mesh refinement for incompressible flow problems

J. Cotela[*,**], R. Rossi[*,**], E. Oñate[*,**], N. Lafontaine[**] and S. Idelsohn[*,***]
Corresponding author: jcotela@cimne.upc.edu

[*] Centre Internacional de Mètodes Numèrics en Enginyeria (CIMNE).
Gran Capità s/n, Edifici C1 - Campus Nord UPC, 08034 Barcelona, Spain.
[**] Universitat Politècnica de Catalunya (UPC)
Jordi Girona 1-3, Edifici C1, 08034 Barcelona, Spain.
[***] ICREA Research Professor.

Abstract: We present an algorithm for the solution of the incompressible Navier-Stokes equations combined with local refinement of the mesh based on an error estimation. Our solution strategy is based on well-established CFD techniques and has been designed to be efficient and scalable in a parallel environment. This has proved to be specially challenging for the mesh refinement, for which we propose an algorithm designed to use as little non-local information as possible. We complement the discussion with a numerical example.

Keywords: parallel computing, incompressible Navier-Stokes, adaptive refinement.

1 Introduction

We begin by briefly describing our general framework. Our solver has been implemented within Kratos Multi-Physics [1], which provides an environment for the development of multi-disciplinary finite element solvers. One of our main design goals is to implement a strategy that can operate efficiently and scale both in shared and distributed memory parallel environments. In the case of distributed memory, we rely on Trilinos [2] to manage the matrices and vectors that are shared between the processes.

2 Problem Statement

The starting point for our problem are the incompressible Navier-Stokes equations

$$\rho \frac{\partial \mathbf{u}}{\partial t} + \rho \mathbf{u} \cdot \nabla \mathbf{u} - \nabla \cdot (2\mu \nabla^s \mathbf{u}) + \nabla p = \mathbf{f} \tag{1}$$

$$\rho \nabla \cdot \mathbf{u} = 0 \tag{2}$$

where $\mathbf{u}$ is the velocity, p is the pressure, $\mathbf{f}$ the vector of external forces and ρ and μ are the fluid's density and dynamic viscosity, respectively. Operator $\nabla^s := \frac{1}{2}\left(\nabla + \nabla^T\right)$ is the symmetric gradient.

Equations (1) and (2) constitute a non-linear system of partial differential equations. In order to solve it numerically, we need to introduce a linearization for the convective term and a scheme for time iteration, resulting in a system of the form

$$\rho\frac{\mathbf{u}_{n+1}}{\theta\Delta t}+\rho\mathbf{a}\cdot\nabla\mathbf{u}_{n+1}-\nabla\cdot(2\mu\nabla^{s}\mathbf{u}_{n+1})+\nabla p_{n+1}=\mathbf{f}_{n+1}+\rho\frac{\mathbf{u}_{n}}{\theta\Delta t} \tag{3}$$

$$\rho\nabla\cdot\mathbf{u}_{n+1}=0 \tag{4}$$

where $\mathbf{a}=\mathbf{u}^{(i-1)}$ is the convective velocity, taken as the velocity obtained in the previous iteration of the linearization loop. Note that, although we have used a simple time integration rule in the previous equations for clarity, the results shown in Section 5 have been obtained using the Bossak scheme [3], a member of the generalized-α Newmark family of methods.

Obtaining the weak form of equations (3) and (4) is the starting point for its solution using finite elements. For this we need to introduce a stabilized formulation. We have implemented both algebraic subgird scales (ASGS) [4] and quasi-static orthogonal subscales (OSS) [5]. Both methods are based on the affirmation that we can express the velocity field as $\mathbf{u}=\mathbf{u}_h+\tilde{\mathbf{u}}$, where $\mathbf{u}_h$ is the part of the continuous solution that can be reproduced by our mesh. The remaining part of the solution, the subscale, is then modeled as a local function of the momentum equation residual $\tilde{\mathbf{u}}\approx\tau_1\Pi(\mathbf{r}^{mom})$, where $\Pi(\cdot)$ projects the residual to the space that contains the subscales. The same reasoning is applied for the pressure and the mass equation residual.

Dropping the subindices in $\mathbf{u}_h$ and p_h, the weak form of the problem can be stated as:

Find the values of $\mathbf{u}_{n+1}$, p_{n+1} *that make the residuals given by equations (5) and (6) equal to zero for all suitable test functions* $\mathbf{v}$ *and* q,

$$\begin{aligned}\mathbf{r}^{mom}_{n+1}:=&\int_{\Omega}\mathbf{v}\,\mathbf{f}_{n+1}-\rho\,\mathbf{v}\,\partial_t\mathbf{u}_{n+1}-2\mu\nabla^{s}\mathbf{v}:\nabla^{s}\mathbf{u}_{n+1}-\rho\,\mathbf{v}\,\mathbf{a}\cdot\nabla\mathbf{u}_{n+1}+\nabla\cdot\mathbf{v}\,p_{n+1}\mathrm{d}\Omega \\ &+\sum_{e}\int_{\Omega^e}(\mathbf{a}\cdot\nabla\mathbf{v})\,\tau_1\,\Pi\left(\mathbf{f}_{n+1}-\rho\,\partial_t\mathbf{u}_{n+1}-\rho\,\mathbf{a}\cdot\nabla\mathbf{u}_{n+1}-\nabla p_{n+1}\right)\mathrm{d}\Omega^e \\ &-\sum_{e}\int_{\Omega^e}(\nabla\cdot\mathbf{v})\,\tau_2\,\Pi\left(\rho\,\nabla\cdot\mathbf{u}_{n+1}\right)\mathrm{d}\Omega^e\end{aligned} \tag{5}$$

$$r^{mass}_{n+1}:=-\int_{\Omega}\rho\,q\nabla\cdot\mathbf{u}_{n+1}\mathrm{d}\Omega+\sum_{e}\int_{\Omega^e}(\nabla q)\,\tau_1\,\Pi\left(\mathbf{f}_{n+1}-\rho\,\partial_t\mathbf{u}_{n+1}-\rho\,\mathbf{a}\cdot\nabla\mathbf{u}_{n+1}-\nabla p_{n+1}\right)\mathrm{d}\Omega^e \tag{6}$$

where we have assumed a linear interpolation, Ω denotes the entire domain and Ω^e a single element. $\partial_t\mathbf{u}_{n+1}$ is the acceleration, calculated according to the chosen time scheme, and τ_1 and τ_2 are stabilization parameters.

3 Decoupling the pressure equation

To complete the description of our solution procedure, we will briefly comment on the linear system defined by equations (5) and (6). This system combines velocity and pressure degrees of freedom and, as a result, will be poorly conditioned in general. This situation can be improved by the introduction of an uncoupling strategy, separating both variables into smaller, better conditioned linear systems. To do this, we rely on a pressure Schur complement approach for the pressure, similar to that presented in Chapter 2 of reference [6]. Our implementation of this strategy is described in [7], so we will just outline it here.

The problem defined can be recast in matrix form and solved for variations of its variables using Newton-Raphson iterations, which results in

$$\begin{bmatrix}\mathbf{K} & \mathbf{G}\\ \mathbf{D} & \mathbf{L}\end{bmatrix}\begin{bmatrix}\delta\mathbf{u}_{n+1}\\ \delta\mathbf{p}_{n+1}\end{bmatrix}=\begin{bmatrix}\mathbf{r}^{mom}_{n+1}\\ \mathbf{r}^{mass}_{n+1}\end{bmatrix} \tag{7}$$

with
$$\mathbf{K} := -\frac{\partial \mathbf{r}^{mom}_{n+1}}{\partial \mathbf{U}_{n+1}} \quad ; \quad \mathbf{G} := -\frac{\partial \mathbf{r}^{mom}_{n+1}}{\partial \mathbf{P}_{n+1}} \quad ; \quad \mathbf{D} := -\frac{\partial \mathbf{r}^{mass}_{n+1}}{\partial \mathbf{U}_{n+1}} \quad ; \quad \mathbf{L} := -\frac{\partial \mathbf{r}^{mass}_{n+1}}{\partial \mathbf{P}_{n+1}}$$

Now we can solve the system in a staggered way by introducing the intermediate result $\delta\hat{\mathbf{u}} := \mathbf{K}^{-1}\,\mathbf{r}^{mom}$ and manipulating the blocks in (7), with the following iterative scheme:

1. Obtain $\delta\tilde{\mathbf{u}}$ from $\mathbf{K}\,\delta\hat{\mathbf{u}} = \mathbf{r}^{mom}$.
2. Obtain $\delta\mathbf{p}$ from $\left(\mathbf{L} - \mathbf{D}\mathbf{K}^{-1}\mathbf{G}\right)\,\delta\mathbf{p} = \mathbf{r}^{mass} - \mathbf{D}\delta\hat{\mathbf{u}}$.
3. Obtain $\delta\mathbf{u}$ from $\mathbf{K}\,(\delta\mathbf{u} - \delta\hat{\mathbf{u}}) = -\mathbf{G}\,\delta\mathbf{p}$.
4. Repeat from 1 until convergence.

Note that in step 2 we use the approximation $\mathbf{K}^{-1} \approx (\mathrm{Diag}(\mathbf{K}))^{-1}$, as the inverse is unavailable in practice.

4 Refinement Strategy

The two key components of the refinement strategy are the refinement criteria and and the procedure by which this refinement is achieved. Regarding the former, we follow the ideas introduced in [8], which provide a "natural" way to estimate the error for variational multi-scale stabilization methods such as the ones we are interested in.

Broadly speaking, the error estimation strategy is based on the idea that, if our mesh is unable to properly represent the flow in the domain, the subscale velocity will be large when compared to the velocity field captured by the mesh. As seen in Section 2, the velocity subscale is modeled as being proportional to the residual of the momentum equation in each element, which means that it can be easily estimated a-posteriori.

Evaluating the subscale in each element we can detect areas where the resolution of the mesh is insufficient. We start by evaluating the norm of the solution and subscale velocities on the center of each element. Then an average velocity norm for the domain is obtained from the solution velocity values. The ratio between the subscale velocity and the average velocity is used to compute a dimensionless error estimate. Finally, all elements where this error estimate surpasses a pre-determined tolerance are refined.

The next step is to locally refine the mesh in a parallel environment, where the information each element has of its neighbors (which may be in different partitions) is limited and expensive to obtain. To this end, we have designed an algorithm that refines the mesh by splitting selected element edges in half. This divides each element into four triangles or eight tetrahedra depending on the spatial dimension of the problem, transforming the element's immediate neighbors accordingly.

We want to remark that, in the parallel data storage strategy used by Kratos Multi-Physics, elements are known by a single process, while nodes can be known to multiple processes. This conditions our refinement strategy, which must be adapted to this data structure. With this in mind, the algorithm performs the following steps:

- Build a (sparse) matrix of nodal connectivities, and mark the position i, j (for $i \leq j$, as it is symmetric) with the arbitrary value of `-1` if an edge between nodes i and j exists in the model. This matrix is built locally by each process and later assembled.
- Each process loops over its elements checking if the velocity ratio is greater than a given tolerance. When a candidate for refinement is found, its edges are marked in the nodal connectivity matrix with the value `-2`. This information is then transferred to other processes when necessary.

- Count the number of nodes to be added by each process and in total, and assign a range of node indices to each process.
- Each process marks the nodes it will create by assigning an index to the matrix position of the edge where it will be located. In case of edges shared between processes, the last one to assign an index to the new node will be its owner. This allows us to assign node ownership in a way that is deterministic between processes. New elements are always owned by the same process as the element they originate from.
- Each process creates the new nodes and elements it will own, initializing nodal data in new nodes by interpolation.
- Renumber nodes and elements so that their indices are correlative.

5 Numerical Example

As an application example, we reproduce here the results obtained for two-dimensional flow around a cylinder, using OSS stabilization, based on one of the examples in [5]. We started from an initial mesh of 3984 elements and simulated the first 60 seconds of flow, with a time step of 0.1 seconds. After an initial waiting phase, we start the error control routine, refining every 20 solution steps for a total of 40 refinement passes (although the refinement is limited to 3 passes over the same area).

The simulation was run, for test purposes, using 8 processes and distributed memory in our cluster at CIMNE. At the end of the simulation, we obtained a final mesh containing 11666 elements, which can be seen in Figure 1, distributed between the partions as shown in Table 1. We can see that the refined area coincides with the region where vortices develop, which is what we would expect.

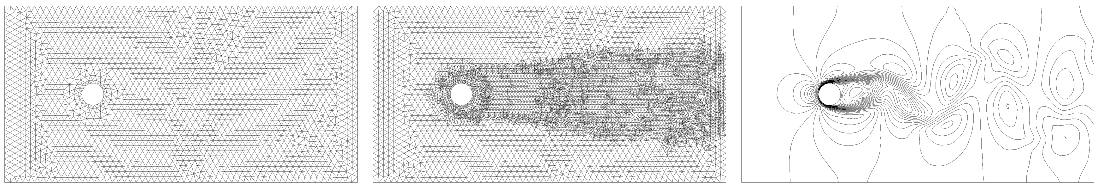
Figure 1: Initial and refined meshes and final velocity contours for the cylinder example.

Process	0	1	2	3	4	5	6	7	Total
Initial	496	510	501	493	494	490	505	495	3984
Final	1056	1871	836	740	1884	1694	1542	2043	11666

Table 1: Initial and final number of elements on each process.

6 Conclusions

We have presented a parallel CFD solver with the ability to locally refine the mesh according to an error estimation. The main challenge has been the implementation of a parallel refinement strategy. On this point, we have followed an approach that minimizes the amount of non-local information required and provides a consistent outcome in all processes.

However, there remain several open questions to be answered, specially with regards to the refinement strategy. A first issue is the error estimation. While the relative magnitude of the subscale velocities provides valuable insight on where to refine, there is no clear way to fix the tolerance over which refinement will start. For example, the magnitude of the subscale velocity can be expected to increase as we approach turbulent regime. Should the tolerance be adjusted accordingly?

An important question that has not been addressed here is that the refinement will be localized in specific areas of the domain and, as a result, the number of elements in each process will change. A crucial line of future work will be defining a strategy to preserve the load balance when this happens.

Finally, our refinement strategy does not attempt to control the quality of the refined mesh, which can be an issue if multiple refinement passes are performed over the same area. Unfortunately, this operation requires knowledge of the mesh geometry, which can be non-local in general, and will be difficult to obtain without a significant increase of communication along partition interfaces.

Acknowledgements

The authors wish to acknowledge the support of the Spanish *Ministerio de Ciencia e Innovación* through the E-DAMS project and the European Commission thorugh the Realtime project. The work of J. Cotela is funded by the Spanish *Ministerio de Educación* through a doctoral grant in the FPU program.

References

[1] P. Dadvand, R. Rossi, and E. Oñate, "An object-oriented environment for developing finite element codes for multi-disciplinary applications," *Archives of Computational Methods in Engineering*, vol. 17, pp. 253–297, 2010.

[2] M. A. Heroux, R. A. Bartlett, V. E. Howle, R. J. Hoekstra, J. J. Hu, T. G. Kolda, R. B. Lehoucq, K. R. Long, R. P. Pawlowski, E. T. Phipps, A. G. Salinger, H. K. Thornquist, R. S. Tuminaro, J. M. Willenbring, A. Williams, and K. S. Stanley, "An overview of the Trilinos project," *ACM Trans. Math. Softw.*, vol. 31, no. 3, pp. 397–423, 2005.

[3] W. Wood, "A unified set of single step algorithms. Part 4: Backward error analysis applied to the solution of the dynamic vibration equation," *International Journal for Numerical Methods in Engineering*, vol. 23, no. 5, pp. 929–944, 1986.

[4] R. Codina, "A stabilized finite element method for generalized stationary incompressible flows," *Computer Methods in Applied Mechanics and Engineering*, vol. 190, no. 20-21, pp. 2681–2706, 2001.

[5] R. Codina, "Stabilized finite element approximation of transient incompressible flows using orthogonal subscales," *Computer Methods in Applied Mechanics and Engineering*, vol. 191, no. 39-40, pp. 4295–4321, 2002.

[6] S. Turek, *Efficient solvers for incompressible flow problems: an algorithmic and computational approach*. Lecture notes in computational science and engineering, Springer-Verlag, 1999.

[7] J. Cotela, R. Rossi, E. Oñate, and P. Dadvand, "A comparison of different parallel techniques applied to the solution of the Navier-Stokes equations," in *Proceedings of the Second International Conference on Parallel, Distributed, Grid and Cloud Computing for Engineering*, 2011.

[8] G. Hauke, M. H. Doweidar, and M. Miana, "The multiscale approach to error estimation and adaptivity," *Computer Methods in Applied Mechanics and Engineering*, vol. 195, no. 13-16, pp. 1573–1593, 2006. A Tribute to Thomas J.R. Hughes on the Occasion of his 60th Birthday.