Domain Decomposition and Skeleton Programming with OCamlP3l

F. Clément, V. Martin, A. Vodicka, R. Di Cosmo, P. Weis

published in

Parallel Computing:
Current & Future Issues of High-End Computing,
Proceedings of the International Conference ParCo 2005,
G.R. Joubert, W.E. Nagel, F.J. Peters, O. Plata, P. Tirado, E. Zapata
(Editors),
John von Neumann Institute for Computing, Jülich,
NIC Series, Vol. **33**, ISBN 3-00-017352-8, pp. 811-818, 2006.

Domain Decomposition and Skeleton Programming with OCamlP3l

F. Clément[a], V. Martin[a], A. Vodicka[a], R. Di Cosmo[b] P. Weis[c]

[a]Projet Estime, INRIA-Rocquencourt, B.P. 105, F-78153 Le Chesnay Cedex, France.

[b]Case 7014, Université de Paris 7, 2 place Jussieu, F-75251 Paris Cedex 05, France.

[c]Projet Cristal, INRIA-Rocquencourt, B.P. 105, F-78153 Le Chesnay Cedex, France.

When simulating large scale phenomena, it is natural to divide the domain of computation into subdomains. Then, besides the issue of the existence of a global solution to the coupled problem, arises the difficulty of its implementation. The OCamlP3l system provides a structured approach to parallel programming using skeletons and template based compilation techniques: designing and debugging is performed on a sequential version, and the parallel version is automatically deduced by a recompilation of the source program. We present the application of a domain decomposition method to a 3D flow problem around a deep underground nuclear waste disposal.

1. Introduction

This work deals with the simulation of flow and transport in porous media to study the feasibility of an underground nuclear waste disposal. Reliable simulations are crucial to forecast the behavior of contaminants in the geological layers, and thus should be computed in 3 dimensions on very long time scales (transport simulations over 10 million years), with very different length scales (going from the meter to 20 or 30 kilometers), involving possibly different physics. A natural way of treating such a large scale problem is to divide it into smaller subproblems, and then to couple them.

We restrict ourselves in this paper to the efficient simulation of stationary flow. The Darcy equations are solved by using a non-overlapping domain decomposition method that allows the treatment of non-matching grids. This coupling method is based on Robin interface conditions and was studied in [1]. This nonconforming domain decomposition method is very practical as it allows the separate meshing of the subdomains: for example, a local refinement around the underground storage, and the rest of the domain can be meshed with a coarser mesh, following the geological layers.

Code coupling and parallelism are a very demanding implementation task, especially when the codes to couple have been developed separately. The main difficulty is the fine tuning of the communications between the codes. Most of these aspects actually correspond to a sequence of generic basic tasks that should be automatically set up by a program, or better via compilation. This is where OCamlP3l enters the picture. Objective Caml (or OCaml) is a fast modern high-level functional programming language based on formal semantics. It is particularly well suited for the implementation of complex algorithms. As an example, the OCamlP3l system, see [3], provides a structured approach to parallel programming using skeletons and template based compilation techniques, see [6]. It brings all the piping capabilities we need to implement the communications, and moreover, it offers the parallelization for free: designing and debugging is driven on a sequential version, and the parallel version is automatically deduced by a simple recompilation of the same source program.

We present first the domain decomposition method in Section 2, then Section 3 is devoted to the OCamlP3l system and Section 4 to the implementation of the coupling algorithm, and finally in Section 5, we show numerical results for both extensibility tests and a realistic 3D flow computation.

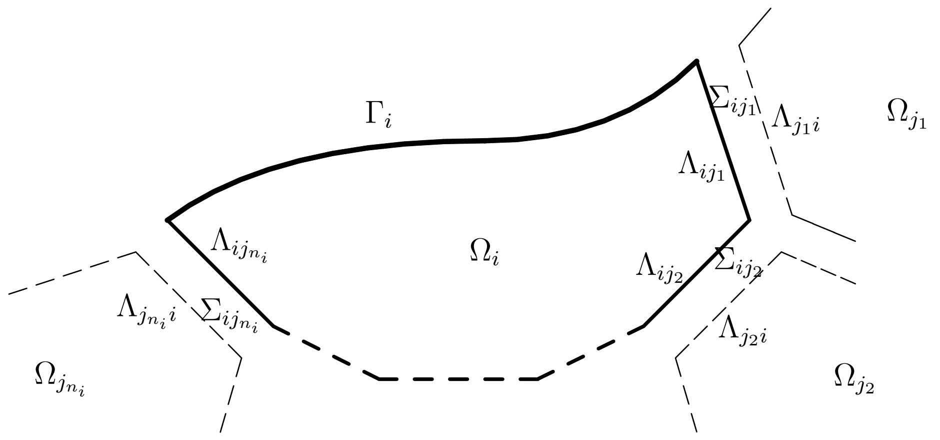

Figure 1. The subdomain Ω_i and its n_i neighbors Ω_{j_k}, $j_k \in \mathcal{N}_i$. The approximation spaces Λ_{ij_k} and $\Lambda_{j_k i}$ on each side of the interface Σ_{ij_k} are different.

2. Domain Decomposition

2.1. A model problem

Let Ω be a convex domain in $\mathbb{R}^d$, $d = 2$ or 3, and let $\Gamma = \partial\Omega$ be its boundary. We suppose that the flow in Ω is governed by a conservation equation together with Darcy's law relating the gradient of the pressure p to the Darcy velocity $\mathbf{u}$ via

$$\left\{\begin{array}{rcll} \operatorname{div}\mathbf{u} & = & q & \text{in } \Omega \\ \mathbf{u} & = & -\mathbf{K}\boldsymbol{\nabla} p & \text{in } \Omega \\ p & = & \overline{p} & \text{on } \Gamma, \end{array}\right. \tag{1}$$

where $\mathbf{K}$ is the permeability tensor, q a source term and $\overline{p}$ the given pressure on the boundary Γ.

The domain Ω is decomposed into n non-overlapping subdomains Ω_i with $i \in I = \{1, 2, \ldots, n\}$, and we denote by $\Gamma_i = \partial\Omega_i \cap \Gamma$ the *external boundaries*. Let $\Sigma_i = \partial\Omega_i \backslash \Gamma$ be the *internal boundary* of Ω_i, and let $\Sigma = \bigcup_{i\in I} \Sigma_i$ be the geometric *structure* of the decomposition. Then, we can define the geometric *interface* between neighboring subdomains Ω_i and Ω_j as $\Sigma_{ij} = \Sigma_{ji} = \Sigma_i \cap \Sigma_j$. We denote the number of neighbors of subdomain Ω_i by n_i and the set of their indices by $\mathcal{N}_i = \{j_1, j_2, \ldots, j_{n_i}\}$. See Figure 1 for an illustration in the 2D case. The set of all couples of neighbors, called *connectivity table*, is denoted by $\mathcal{N} = \{(i,j)/i \in I, j \in \mathcal{N}_i\}$, it is naturally ordered by the lexicographic order on $\mathbb{N}^2$. Let s be the permutation involution on $\mathcal{N}$ defined by $s(i,j) = (j,i)$.

2.2. A fixed point formulation

Let $\mathcal{T}_i^h$ be a conforming finite element partition of the subdomain Ω_i. We denote by $\mathcal{S}_{ij}^h$ the trace of $\mathcal{T}_i^h$ on Σ_{ij}. In general, $\mathcal{S}_{ij}^h \neq \mathcal{S}_{ji}^h$ when the meshes $\mathcal{T}_i^h$ and $\mathcal{T}_j^h$ do not match at the interface Σ_{ij}.

Let $\Lambda_{ij} = P_0(\mathcal{S}_{ij}^h)$ the space of facewise, or edgewise in 2D, constant functions on $\mathcal{S}_{ij}^h$ and set

$$\Lambda_i = \bigoplus_{j\in\mathcal{N}_i} \Lambda_{ij}, \qquad \Lambda = \bigoplus_{i\in I} \Lambda_i = \bigoplus_{(i,j)\in\mathcal{N}} \Lambda_{ij},$$
$$\tilde{\Lambda}_i = \bigoplus_{j\in\mathcal{N}_i} \tilde{\Lambda}_{ij} = \bigoplus_{j\in\mathcal{N}_i} \Lambda_{ji}, \quad \tilde{\Lambda} = \bigoplus_{i\in I} \tilde{\Lambda}_i = \bigoplus_{(i,j)\in\mathcal{N}} \Lambda_{ji} = \bigoplus_{(i,j)\in s(\mathcal{N})} \Lambda_{ij}.$$

The global problem (1) can be rewritten as a transmission problem which states the same equations on each subdomain together with transmission conditions expressing continuity of the pressure and of the normal velocity across each interface Σ_{ij}. These transmission conditions can be rewritten as mixed Robin conditions with Robin coefficients $\alpha_{ij} > 0$ and $\alpha_{ji} > 0$ on each interface Σ_{ij}. Using a mixed finite element method with hybrid Lagrange multipliers, see [7], the discrete approximation of the local elliptic subproblems with Robin conditions take of the form

$$L_i v_i = (q_i, g(v_i)) \tag{2}$$

where v_i summarizes the velocity, pressure and Lagrange multiplier unknowns, and the function g expresses the Robin conditions on all the interfaces of the subdomain Ω_i.

Then, a simple method to solve all subproblems (2) for $i \in I$ is to solve iteratively the following fixed point problem: given $\lambda^0 \in \Lambda$, for $k \geq 0$, compute until convergence $\lambda_i^{k+1} = g(v_i^{k+1})$ where v_i^{k+1} is the solution of the local linear system

$$L_i v_i^{k+1} = (q_i, \lambda_i^k). \tag{3}$$

One can notice that matrix L_i is non-symmetric, hence when using an iterative method to solve (3), it is advisable to accelerate the convergence with a non-symmetric Krylov method such as BiCGStab or GMRes.

2.3. Interface operators

We introduce discrete Robin-to-Robin operators S_{iq_i} in each subdomain Ω_i defined by:

$$S_{iq_i} : \lambda_i \in \Lambda_i \longmapsto \mu_i = g(v_i) \in \Lambda_i \tag{4}$$

where v_i of the solution of the *inner* linear system

$$L_i v_i = (q_i, \lambda_i). \tag{5}$$

For all neighbors Ω_i and Ω_j, $(i,j) \in \mathcal{N}$, let $P_{i\to j}$ be the L^2-projection operator from $L^2(\Sigma_{ij})$ onto $\tilde{\Lambda}_{ij} = \Lambda_{ji}$. For all subdomain Ω_i, $i \in I$, let P_i be the tensor product $\bigotimes_{j\in\mathcal{N}_i} P_{i\to j}$. And let R_i (resp. $\tilde{R}_i$) be the restriction operator from Λ onto Λ_i (resp. from $\tilde{\Lambda}$ onto $\tilde{\Lambda}_i$).

Finally, we define the global operators

$$S_q = \sum_{i\in I} \tilde{R}_i^\top S_{iq_i} R_i : \Lambda \longrightarrow \Lambda \quad \text{and} \quad P = \sum_{i\in I} \tilde{R}_i^\top P_i R_i : \Lambda \longrightarrow \tilde{\Lambda} \tag{6}$$

and we set

$$A = \mathrm{Id}_\Lambda - sPS_0 \quad \text{and} \quad b = sPS_q(0). \tag{7}$$

With all these notations, solving the initial problem (1) is equivalent to find $\lambda \in \Lambda$ solution to the *outer* linear system

$$A\lambda = b. \tag{8}$$

Once again, since the matrix A is non-symmetric, it is advisable to accelerate the convergence of the resolution of (8) with a non-symmetric Krylov method such as BiCGStab or GMRes.

3. Skeleton Programming with OCamlP3l

Caml is a strongly-typed functional programming language from the ML (Meta Language) family. OCaml (Objective Caml) is an open source implementation of Caml developed at INRIA[1]. P3L (Pisa Parallel Programming Language) is a structured parallel programming language developed at the Department of Computer Science of the University of Pisa. It is based upon skeleton/templates and allows parallel programs to be developed composing a small set of primitive parallel forms[2]. OCamlP3l is a parallel programming system based on OCaml and P3l languages, providing seamless integration of parallel programming and functional programming and advanced features like sequential logical debugging of parallel programs and strong typing, useful both in teaching parallel programming and in the building of full-scale applications[3].

We briefly present now the OCamlP3l system, and the reader can refer to [2] for more details.

3.1. Three semantics

A distinctive feature of the OCamlP3l system, among all skeleton-based systems, is that the semantics of the skeletons is not hard-wired: the system allows the user to compile his code without any source modification using three possible semantics, i.e. three implementations of the skeletons. The *sequential semantics* produces an executable that can run on a single machine, as a single process, and easily debugged using standard techniques and tools for sequential programs. The *parallel semantics* produces a generic SPMD program that can be deployed on a parallel machine, a cluster, or a network of workstations. The *graphical semantics* produces a program that displays a picture of the parallel computational network that is deployed when running the parallel version.

A key issue in the further development of OCamlP3l will be the proof of the adequation theorem stating the agreement between sequential and parallel executions: under reasonable assumptions, for any user program the two semantics should produce exactly the same result. Hence, the user has only to debug the sequential version.

3.2. Skeletons combinators

The OCamlP3l skeletons are compositional: the skeletons are combinators that form an algebra of functions and functionals called the *skeleton language* that define the parallel behavior of programs. More precisely, a skeleton is a *stream processor*, i.e. a function that transforms an input stream of incoming data into an output stream of outgoing data.

In version 2.0 of OCamlP3l, the eight combinators pertain to five kinds:

- the *task parallel* skeletons `farm` and *pipeline* model the parallelism of independent processing activities relative to different input data.
- the *data parallel* skeletons `mapvector` and `reducevector` model the parallel computation of different parts of the same input data.
- the *data interface* skeletons `seq` and `parfun` provide (dual) injection and projection between sequential and parallel worlds.
- the *parallel execution scope delimiter* skeleton `pardo` must encapsulate all the code that invokes `parfun`'s.
- the *control* skeleton `loop` provides the repetitive execution of a given skeleton expression.

[1]see `http://caml.inria.fr/`.
[2]see `http://www.di.unipi.it/˜susanna/p3l.html`.
[3]see `http://www.ocamlp3l.org/`.

The `mapvector` combinator computes in parallel a function over all the components of *vector* data items of the input stream. The expression `mapvector(`f,k`)` computes $(f(x_i^1),\ldots,f(x_i^n))$ over all (vector) data items $x_i = (x_i^1,\ldots,x_i^n)$ by having k independent processes that compute f over different components of the vector.

The `seq` combinator converts a sequential function into a node of the parallel computational network.

The `parfun` combinator converts a parallel computational network into a sequential stream processing function.

See [2] for the description of the other skeletons not used in the present application. All skeleton combinators allow for global or local initialization, meaning that independent processes will share, or not, their initialization data.

4. Implementation of the code coupling

4.1. Code to couple

The code to be (self-)coupled is the C++ solve_on_a_subdomain code that inputs the name of the file describing the 3D mesh associated with a subdomain, e.g. Ω_i, for $i \in I$. It reads this file from the disk and enters an infinite loop waiting for a keyword:

When given the keyword `"init"`, it computes a sparse LU factorization of the matrix L_i used to solve the inner linear systems (5), then computes and outputs $S_{iq_i}0_i$ needed for the computation of the right-hand side b given by (7).

When given the keyword `"loop"`, it inputs λ_i, then computes and outputs $S_{i0_i}\lambda_i$ needed for the computation of the matrix-vector product.

When given the keyword `"final"`, it inputs $\lambda_i^\star$, then computes and writes on the disk the solution $v_i^\star$ to the problem (5) associated with $S_i q_i \lambda_i^\star$.

It needs redirection of both its standard input and standard output. Moreover, the `"init"` phase is very costly and must be performed once and for all. Therefore, this code has to be *locally initialized*, and needs also to have the ability to be recalled, by having its I/O channels stored.

The implementation of non-symmetric Krylov method, e.g. BiCGStab, only requires a matrix-free matrix-vector product routine aa to compute the action of matrix A on any vector λ. The `parfun` skeleton will allow us to make this *ordinary routine* be a parallel computation network.

4.2. The coupling algorithm

Then, the coupling algorithm is the following.

Initialization

- define aa that applies in parallel matrix A given by (7) to any vector $\lambda \in \Lambda$.
- compute in parallel the inverse of the matrices L_i of the inner linear systems for all subdomains and the right-hand side b given by (7).
- choose $\lambda^0 = 0$.

Iteration

- run BiCGStab with the parallel matrix-vector product aa, the right-hand side b and the initial guess λ^0.
- name the solution $\lambda^\star$.

Finalization

- solve in parallel the inner linear systems associated with $S_q\lambda^\star$ and store the $v_i^\star$'s for all subdomains.

4.3. The OCamlP3l coupling code

The coordination code itself is so short that it can be presented in extenso. The parallel annotations, i.e. the OCamlP3l skeletons, are underlined.

```
    let solve_on_all_subdomains =
      let prog () =
        let cin = ref None and cout = ref None in
        (fun (i, task, li) ->
 5         let command = Dd.make_solve_on_a_subdomain_command i in
           let ic, oc = My_unix.spawn command cin cout in
           Dd.send_task oc task;
           Dd.send_boundary_values oc li;
           flush oc;
10         Dd.receive_boundary_values ic) in
      parfun (fun () -> mapvector (seq prog, Dd.number_of_processors));;

    pardo (fun () ->
      (* Initialization *)
15    let n = Dd.number_of_subdomains in
      let f = fun l ->
        Dd.permutation (Dd.projection (solve_on_all_subdomains l)) in
      let aa = fun l -> Dd.axpy (-1.) (f (Dd.loop_vector l)) l in
      let b = f (Dd.init_vector_of_size n) in
20    let l_0 = Dd.zero_vector_of_size n in
      let algorithm = Bicgstab.algorithm Dd.axpy Dd.dot in
      (* Iteration *)
      let l_star = algorithm aa b l_0 in
      (* Finalization *)
25    solve_on_all_subdomains (Dd.final_vector l_star));;
```

The `Dd` module is dedicated to domain decomposition. In particular, it delivers types for the unknown vector λ that ease the implementation of the operators R_i and $\tilde{R}_i^T$.

The function `solve_on_all_subdomains` (lines 1–11) correspond to the code to be coupled of section 4.1. It is defined defined through an encapsulation of a `mapvector` skeleton inside a `parfun` skeleton to allows parallel computation of the function over all components of a vector anywhere in the sequential code. It uses local initialization to reserve memory slots to store the I/O channels.

The domain decomposition algorithm is implemented inside the `pardo` scope delimiter (lines 13–25). It uses repeatedly the previously defined stream processing network. The function `f` (lines 16–17) defines the function $f = sPS_q$ for which we are searching the fixed point. It is used in both the body of the matrix-vector product `aa` (line 18) and the computation of the right-hand side `b` (line 19).

5. Numerical Tests

Numerical tests have been run on the Cristal Cluster deployed at INRIA-Rocquencourt. It is made of sixteen 2.8 GHz Intel Xeon bi-processors with 2 Gb of RAM each interconnected on a dedicated

Gigabit network. More details can be found in [4].

5.1. Extensibility results

We associated one processor to each subdomain, and we increased the number of processors while keeping a constant load per processor. All subdomains were made of about 50000 cells, in order to avoid swapping problems during the initialization phase of the local subproblems. The efficiency of the coupling algorithm is related to the way the global domain is decomposed, but this always led to a flat 30%-overhead in the conforming case, and no more than twice the time in the nonconforming case, which is not bad for up to 16 subdomains, i.e. up to 800000 cells.

The main drawback of the current version of OCamlP3l is the existence of a bottleneck since all communications are centralized in the `mapvector` skeleton. Obviously, this point was not crucial in our tests, but this may become an issue for larger tests. So a specific skeleton accounting for direct communication driven by a connectivity table is under development.

5.2. A realistic 3D flow simulation

We consider now the problem of a 3D flow simulation in a realistic porous media designed by ANDRA (the French National Agency for Radioactive Waste Management) to study the feasibility of an underground nuclear waste disposal, see [5]. We have simplified the model by making all the subdomains homogeneous, but the numerical difficulty remains the flat aspect of the domain, and the high heterogeneity between the subdomains as the permeability jumps by a factor of several orders of magnitude from a geological layer to the other.

The domain is defined by a global mesh provided by ANDRA containing about 400000 hexahedric cells. It is decomposed following the geology into 12 subdomains with matching meshes at the interfaces, see Figure 2a where the unit on the vertical axis is multiplied by a factor of 100 (the domain is actually flat). The pressure solution is displayed on Figure 2b.

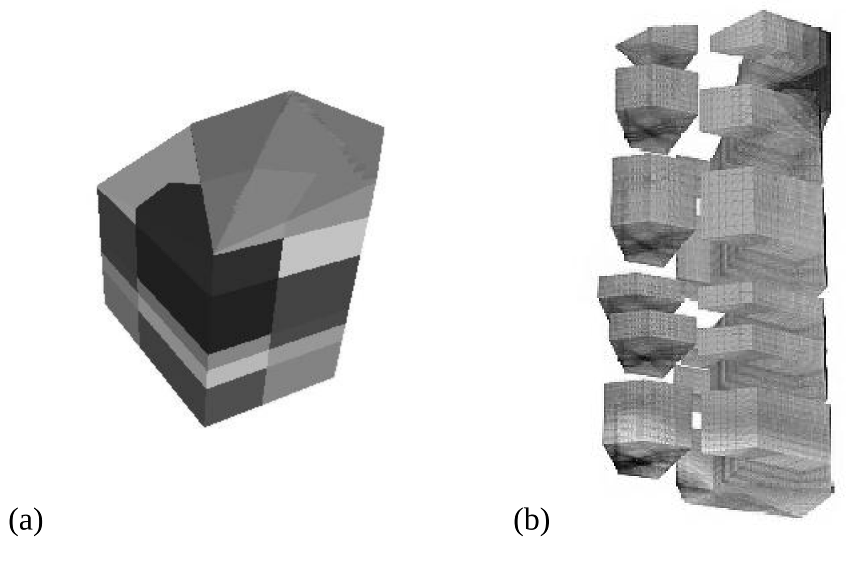

Figure 2. A realistic 3D flow simulation. (a) The domain. (b) A split view of the pressure.

The main difficulty here was to fine tune the Robin coefficients on each interface to insure convergence of the coupling algorithm. Our empirical tests based on the physics of the porous medium and the geometry of the domain led us to take into account the aspect ratio of the subdomains and the permeability jumps by choosing

$$\alpha_{ij} = \alpha_{ji} = \frac{2K_iK_j}{(K_i + K_j)L^{\Omega}_{ij}} \quad i, j \in I \tag{9}$$

where L^{Ω}_{ij} is the characteristic length of the domain Ω along the normal direction to the interface Σ_{ij}.

6. Conclusions

We have presented a non-overlapping domain decomposition method for non-matching meshes based on Robin interface conditions for the computation of 3D flow in porous media.

The main contribution of this paper is the way this algorithm is implemented. We have used the OCamlP3l parallel programming system that comes from the functional programming world. This system provides a structured approach to parallel programming obtained from skeletons and template based compilation techniques. The user describe its parallel algorithm by combining basic building blocks, then the (same) source code can be compiled either for sequential execution or for parallel execution, and both modes should always produce the same result. In short, this means that the user has never to take care of bugs specific to parallelism. 3D numerical results have shown the interest of the approach.

We are now working on both numerical and programming aspects: on one side, we are implementing another domain decomposition method based on Neumann-Neumann preconditioning with balancing, and on the other side, we are developing a new `mapvector` skeleton with communication capabilities based on a connectivity table that will reduce the bottleneck problem.

7. Acknowledgments

The authors thank Martial Mancip for providing an efficient implementation for a conservative L^2-projection routine, and ANDRA, the French National Agency for Radioactive Waste Management, for providing the 3D realistic test case.

References

[1] Todd Arbogast and Ivan Yotov. A non-mortar mixed finite element method for elliptic problems on non-matching multiblock grids. *Comput. Methods Appl. Mech. Engrg.*, 149(1-4):255–265, 1997. Symposium on Advances in Computational Mechanics, Vol. 1 (Austin, TX, 1997).

[2] F. Clément, R. Di Cosmo, Z. Li, V. Martin, A. Vodicka, and P. Weis. Parallel programming with the OcamlP3l system. Applications to numerical code coupling. Rapport de Recherche 5131, Inria, Rocquencourt, France, 2004.

[3] Marco Danelutto, Roberto Di Cosmo, Xavier Leroy, and Susanna Pelagatti. Parallel functional programming with skeletons: the OCamlP3l experiment. *The ML Workshop*, 1998.

[4] V. Martin. *Simulations multidomaines des écoulements en milieu poreux.* PhD thesis, Université de Paris 9, France, 2004.

[5] G. Pépin, B. Vialay, and D. Perraud. Cahier des charges relatif à la réalisation de calculs de transport de radionucléides avec le code castem2000. Cahier des charges, Andra, Châtenay-Malabry, France, March 2001. In French.

[6] F. A. Rabhi and S. Gorlatch, editors. *Patterns and Skeletons for Parallel and Distributed Computing.* Springer-Verlag, London, 2003.

[7] J.E. Roberts and J.-M. Thomas. Mixed and hybrid methods. In P.G. Ciarlet and J.L. Lions, editors, *Handbook of Numerical Analysis Vol.II*, pages 523–639. North Holland, Amsterdam, 1991.