Thirteenth International Conference on Domain Decomposition Methods
Editors: N. Debit, M.Garbey, R. Hoppe, J. Périaux, D. Keyes, Y. Kuznetsov

28 Ahpik: A Parallel Multithreaded Framework Using Adaptivity and Domain Decomposition Methods for Solving PDE Problems

A. Ben-Abdallah[1], A.S. Charão[2], I. Charpentier[3], B. Plateau[4]

Introduction

Domain decomposition methods are a valuable approach when solving partial differential equation (PDE) problems on parallel computers. In this paper, we focus our attention onto parallelization strategies for these numerical methods when dealing with irregular applications, more specifically when adaptive refinement techniques [Ver96] are applied to PDE problems involving unstructured meshes. A parallel object-oriented framework called AHPIK [CCP00] has been developed to cope with such irregular behaviors of simulations relying on domain decomposition. It provides general abstractions that are suitable for solving PDE problems on distributed memory machines using finite difference or adaptive finite element discretizations, along with overlapping or nonoverlapping, synchronous or asynchronous domain decomposition methods. One of the main features of AHPIK is the use of multithreading techniques on distributed memory machines (thus scalable) together with a message passing library (MPI). This offers a degree of freedom for traditional parallel solvers, where subdomain computations are scheduled in the context of heavyweight processes which are assigned to a given processor once for all. The use of multiple threads leads to programs that are flexible in terms of data exchange, facilitating a task scheduling with potential for masking communication overhead. Moreover the object-oriented techniques used in AHPIK make reusability, flexibility and expressiveness of source code easy.

Our goal in this paper is to show the efficiency of AHPIK concepts by comparing an AHPIK implementation with an original MPI code which solves an unsteady incompressible Navier-Stokes problem. The impact of our parallel strategy is investigated in two situations : we first consider a well-balanced distribution of the subdomains, then we induce an irregular parallel behavior by adding adaptive mesh refinement to the original code. This paper is organized in three sections: in the first one, domain decomposition methods and adaptivity techniques are discussed from a parallelism point of view. The second section presents the multithreaded framework AHPIK, while the third one provides performance results and analysis after a brief introduction to our trial application.

[1]Laboratoire de Modélisation et Calcul, IMAG, Grenoble, Adnene.Ben-Abdallah@imag.fr
[2]Laboratoire Informatique et Distribution, IMAG, Grenoble, Andrea.Charao@imag.fr
[3]Laboratoire de Modélisation et Calcul, IMAG, Grenoble, Isabelle.Charpentier@imag.fr
[4]Laboratoire Informatique et Distribution, IMAG, Grenoble, Brigitte.Plateau@imag.fr

Mathematical Methods

Domain decomposition methods (DD)

AHPIK can be used for a large variety of domain decomposition methods. In this paper, we describe its basic design ideas for the resolution of the Laplace equation by a dual Schur complement method.

In a bounded two-dimensional polygonal domain Ω, we consider the Laplace problem (1):
Find $u \in H_0^1(\Omega)$ *such that*

$$\left\{ \begin{array}{rcll} -\Delta u & = & f & \text{in } \Omega, \\ u & = & 0 & \text{on } \Gamma. \end{array} \right. \tag{1}$$

We denote by Γ the boundary of the domain and f is assumed to be square integrable. In the sequel, functions are supposed to belong to well chosen spaces, that is, the PDE problems have a unique solution.

Let choose a nonoverlapping domain decomposition $\{\Omega_k\}_{k=1,..,K}$ of Ω such that

$$\left\{ \begin{array}{l} \overline{\Omega} = \cup_{k=1,..,K} \overline{\Omega_k}, \\ \Omega_k \cap \Omega_l = \emptyset, \quad \forall (k,l) \in \{1,..,K\}^2,\ k \neq l, \end{array} \right. \tag{2}$$

We denote by Γ_k and γ_{kl} the boundaries of Ω_k that are respectively included in Γ and interfaces with other subdomains ($l = 1,..,K,\ l \neq k$) such that

$$\forall (k,l) \in \{1,..,K\}^2,\ k \neq l, \left\{ \begin{array}{l} \gamma_{kl} = \partial\Omega_k \cap \partial\Omega_l, \\ \Gamma^k = \partial\Omega_k \cap \partial\Omega. \end{array} \right. \tag{3}$$

A PDE problem is then defined on each subdomain and boundary conditions are prescribed on γ_{kl} ($(k,l) \in \{1,..,K\}^2,\ k \neq l$) to satisfy continuity constraints. A Lagrange multiplier (4) allows to write the variational formulation of the local PDE problem as:
Find u *in the appropriate space such that*

$$\left\{ \begin{array}{l} \displaystyle\sum_{k=1}^{K} \int_{\Omega_k} \nabla u_k . \nabla v_k dx = \int_{\Omega_k} f v_k dx,\ \forall v_k \text{ in the appropriate space}, \\ \displaystyle\int_{\gamma_{kl}} \mu(u_k - u_l) ds = 0 \quad \forall\ \mu \in H^{1/2}(\gamma_{kl}),\ \forall (k,l) \in \{1,..,K\}^2,\ k \neq l. \end{array} \right. \tag{4}$$

Problem (4) may be solved using Uzawa's method. Let f_k be the restrictions of function f to domains Ω_k, ν_k be the outer normals to Ω_k and λ_{kl}^0 ($(k,l) \in \{1,..,K\}^2$) be initial data. The knowledge of λ_{kl}^n at iterate n allows to compute u_k^n and λ_{kl}^{n+1} as solutions of (5) and (6):

$$\forall\ k \in \{1,..,K\}, \left\{ \begin{array}{rcll} -\Delta u_k^n & = & f_k & \text{in } \Omega_k, \\ \dfrac{\partial u_k^n}{\partial \nu_k} & = & \lambda_{kl}^n & \text{on } \gamma_{kl} \quad \forall l \in \{1,..,K\},\ l \neq k, \\ u_k^n & = & 0 & \text{on } \Gamma^k, \end{array} \right. \tag{5}$$

$$\lambda_{kl}^{n+1} = -\lambda_{lk}^{n+1} = \lambda_{kl}^n + \rho(u_k^n - u_l^n)_{|\gamma kl},\ \forall\ (k,l) \in \{1,..,K\}^2,\ k < l,$$
(parameter ρ has to be determined). (6)

The trace operator appearing in (6) is defined with respect to the domain decomposition method. The previous description then applies to both the dual Schur method when the global mesh is conform and the mortar method [BMP94] when meshes differ from one side of the interface to the other.

This iterative process may be seen as a composition of computational tasks T_{Ω_k} ($k \in \{1, .., K\}$) that solve local PDE problems (5) in domains Ω_k and computational tasks $T_{\gamma_{kl}}$ ($(k, l) \in \{1, .., K\},\ k \neq l$) that update the Lagrange multiplier (6) corresponding to interfaces γ_{kl}. The decomposition of DD algorithms into separate tasks is generic. It is already coded in AHPIK for the Schwarz overlapping method [Sch90][Lio88] (note that interface computations $T_{\gamma_{kl}}$ are empty in this case), the Schur and dual Schur complement methods and the mortar method. This approach is clearly extensible to the Dirichlet-Neumann method [MQ89]. This also applies when coding asynchronous algorithm (see for example [BT89]).

Adaptation of the discrete space

Standard *a priori* error estimates are sufficient to choose a discrete space convenient with respect to a desired accuracy. Nevertheless, in some cases, solutions may contain singularities, for which *a priori* estimates induce the refinement of all the domain for computing accurate solutions. Adaptivity is an alternate solution. It basically consists in an iterative method that computes local *a posteriori* estimates [Ver96] related to the solution at an iteration. They indicate the part of the mesh that need to be refined, thus allowing to compute a more accurate solution at a lower cost than if global refinement was used.

When solving a PDE problem in parallel via domain decomposition methods, the use of adaptive mesh refinement techniques leads to load imbalances among cooperating processors. The result is an important loss of efficiency since processors solving local PDE problem on coarse meshes may be idle, waiting for processors working on refined meshes. This is all the more true as soon as the chosen domain decomposition method is implemented with synchronous process. An interesting way to cope with this problem consists in assigning several subdomains to each processor, and let the computations be scheduled upon availability of the data they depend on. Doing so, idle times due to communications may be masked with computations. This approach can be coupled with load balancing strategies which allow to perform a new repartition of the subdomains over the processors. One can eventually "move" subdomains from one processor to another during the simulation if needed. As the management of such dynamic parallel behavior is usually cheaper with lightweight processes (threads) than with classical operating system processes, we propose to use the first ones. For a thorough discussion on the advantages of using threads for parallel irregular applications see, for example, [Chr96] and [BT98].

Overview of AHPIK

The AHPIK framework is basically composed of C++ classes that provide abstractions for developing PDE solvers based on domain decomposition methods. Two key abstractions in AHPIK are *internal* tasks and *interface* tasks. Internal tasks perform local computations, i.e., computations that require only local data within a subdomain. Tasks T_{Ω_k} identified in the previous section are examples of internal tasks: they solve local PDE problems, what usually needs solving the sparse linear equation system associated to each subdomain. Interface tasks,

on the other hand, carry out computations or updates over interface degrees of freedom. They require data from neighboring subdomains, as well as results of local computations performed by internal tasks. Tasks $T_{\gamma_{kl}}$ identified in the previous section are examples of interface tasks. Most domain decomposition methods can be described as an iterative process composed by interactions between these two types of tasks. The methods differ in terms of actual operations performed by internal and interface tasks and in the manner these tasks communicate and synchronize their execution.

Based on these ideas, AHPIK programming interface offers C++ classes which encapsulate various communication and synchronization patterns for internal and interface tasks. This includes synchronous and asynchronous algorithms, combined with different convergence control mechanisms. Writing a new domain decomposition solver then involves "filling in" the internal and interface tasks with computations, as well as specifying interface data objects that must be exchanged between processors solving neighboring subdomains. Actual communications are thus hidden from the user. Such high-level approach is achieved through object-oriented programming, which is employed in AHPIK as a means of providing strong separation between programming interface and parallel, multithreaded implementation.

Internally in AHPIK, each task is performed by a specialized thread. Additional sender/receiver threads are employed to carry out communication of boundary data needed for solving each interface problem. Threads are scheduled by the operating system upon availability of data. When a subdomain has more than one interface, interface computations can be performed in parallel by different threads as soon as their input data are available. Several subdomains can be assigned to each processor by multiplexing the set of threads performing internal and interface tasks. One can also solve uncoupled problems in parallel over the same subdomain. This is particularly interesting to efficiently exploit symmetric multiprocessor (SMP) architectures that are widely available nowadays. On such platforms, the different threads composing a parallel program can run simultaneously on different processors. Without multithreading, solving different problems at the same time over the same subdomain usually implies replicating some data. Multithreading techniques are currently used by other frameworks addressing the development of parallel PDE solvers, for example in [RHC+96] and [BBD+98]. Among these, AHPIK is distinguishable by combining multithreading with message-passing on distributed memory machines, and by being specially targeted to domain decomposition methods. The reader will find in [Cha01] a detailed description of the AHPIK framework as well as the basic ideas that have oriented its design and implementation.

Numerical Experiments

Our trial simulation model is a nonstationary incompressible flow around a cylinder with a circular cross section at Reynolds number $Re = 100$. This case corresponds to the 2D case of Schäfer and Turek's benchmark [ST96]. The flow is governed by the Navier–Stokes equations. The problem is solved using a parallel projection scheme based on mortar decomposition method, and a conjugate gradient method is used to solve the interface problem. Details are given in [Abd98].

The domain is divided into $K = 92$ nonoverlapping subdomains. A regular P_1-iso-P_2/P_1 mixed finite element triangulation is defined on each subdomain. One notices that we do not require the grids of each subdomain to match; the weak continuity through the subdomain interfaces is enforced by mortar functions. One of the particular points of this application

is that viscosity and incompressibility of the fluid are treated within two separate steps, and components of the velocity field can be computed in parallel.

Our first experiment consists in comparing the original implementation with AHPIK implementation in a case where the workload is well distributed over the subdomains. Such comparison is carried out over two different platforms : a PC cluster composed of uniprocessor nodes, and a SMP PC cluster comprising 2-processor nodes. Both clusters are homogeneous, but we notice that processors in the SMP PC cluster have higher clock speeds than the cluster of uniprocessor PCs. We use 22 nodes for each parallel execution and most nodes have 4 subdomains to solve. Figure 1 shows the duration of one iteration for both MPI-based original code and AHPIK implementation.

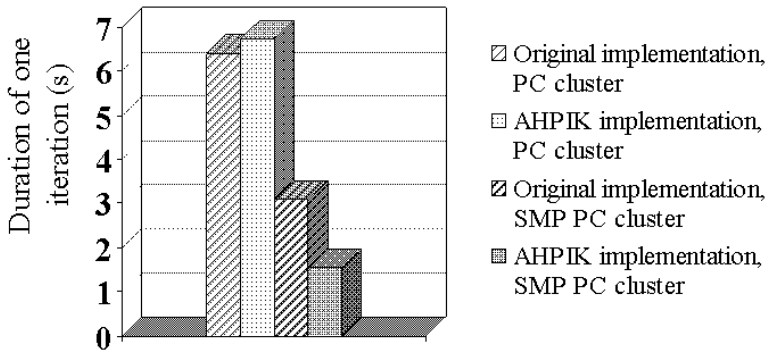

Figure 1: Results for a well-balanced distribution of the subdomains.

We see that the AHPIK version slightly decrease the performance as compared to the original implementation on uniprocessor. This can be explained by the good workload distribution that characterize this experiment. Indeed, when processor utilisation rate is high, using threads introduce overhead. In a multi-processor node, the AHPIK version produces better performance than the original implementation for identical execution parameters. We see that AHPIK implementation mixing threads and message passing automatically adapts to the multi-processor machine, while the original code keeps using only one processor.

Our second experiment consists in adding adaptive mesh refinement to either MPI-based original application and the AHPIK application. This introduces load imbalances as the number of degrees of freedom vary from one processor to another during the time iterative execution. To simplify implementation, we always refine a whole subdomain, thus we achieve the final mesh configuration within few adaptations. Figure 2 show results obtained when running the adaptive codes on each PC cluster platform. While results on the SMP cluster reproduce the behavior observed in the first experiment, results on the PC cluster composed of uniprocessor nodes show that, as long as the workload is unbalanced, the AHPIK implementation can reach or slightly surpass the performance of the MPI-based adaptive code. One can notice that this experiment reproduces a worst case situation as subdomain computations are coarse-grain and the interface solution scheme requires frequent global synchronizations. We expect better performance of the multithreaded version if synchronisation could be relaxed.

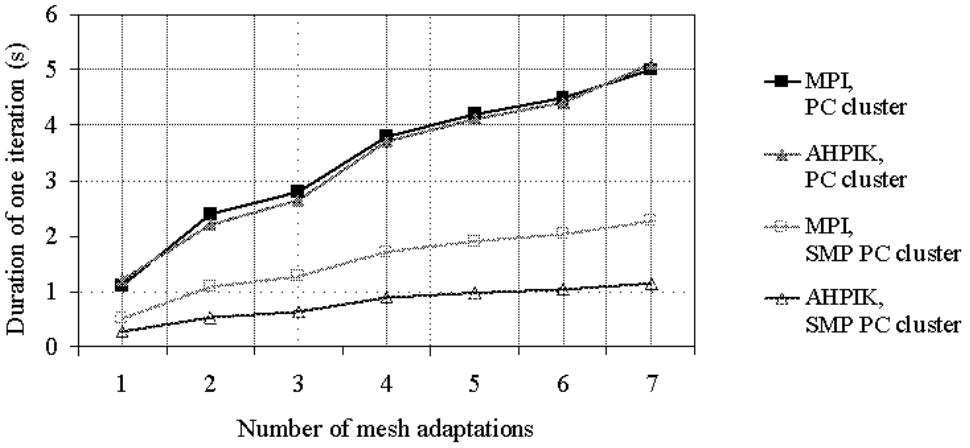

Figure 2: Results for the adaptive case.

1 Conclusion

In this paper we have introduced an object-oriented framework which uses multithreading combined with message-passing as a parallel implementation strategy for domain decomposition methods. The object-oriented approach provides general abstractions that are suitable for a variety of domain decomposition methods, including overlapping and nonoverlapping, synchronous and asynchronous methods. Such abstractions compose a programming interface where communication and synchronization details do not need to be hand-coded as in MPI-based applications.

We have investigated the performance of AHPIK compared to MPI-only domain decomposition implementation for an unsteady incompressible Navier-Stokes problem. Results show that multithreading associated to message-passing introduces more flexibility in parallel PDE solvers relying on domain decomposition, as subdomain computations are dynamically scheduled upon availability of data, and the resulting codes automatically adapt to different parallel architectures. This approach offers a potential for overlapping communication with computations when dealing with irregular applications, however the benefits of such technique are limited by the globally synchronous behavior of some numerical methods. In this sense, one of the important contributions of AHPIK rely on its support to multiple synchronization schemes that can be easily manipulated in the parallel code. This allows for an easy experimental evaluation of different numerical algorithms with different synchronization behaviors for solving a given problem.

These considerations lead us to conclude that the AHPIK approach offers a good compromise between performance and flexibility for implementing parallel PDE solvers based on domain decomposition. In a near future, we plan to use multithreading combined with message-passing to implement and evaluate dynamic load balancing strategies for adaptive PDE computations.

References

[Abd98]Adnene Ben Abdallah. *Méthode de projection pour la simulation de grandes structures turbulentes sur calculateurs parallèles*. PhD thesis, Université Pierre et Marie Curie, Paris, 1998.

[BBD+98]Federico Bassetti, David Brown, Kei Davis, William Henshaw, and Dan Quinlan. OVERTURE: An object-oriented framework for high-performance scientific computing. In *Proceedings of Supercomputing'98 (CD-ROM)*. ACM SIGARCH and IEEE, nov 1998.

[BMP94]Christine Bernardi, Yvon Maday, and Anthony T. Patera. A new non conforming approach to domain decomposition: The mortar element method. In Haim Brezis and Jacques-Louis Lions, editors, *Collège de France Seminar*. Pitman, 1994. This paper appeared as a technical report about five years earlier.

[BT89]Dimitri P. Bertsekas and John N. Tsitsiklis. *Parallel and Distributed Computation*. Prentice-Hall Inc., 1989.

[BT98]Pierre Eric Bernard and Denis Trystram. Report on a parallel molecular dynamics implementation. In E. H. D'Hollander, G. R. Joubert, F. J. Peters, and U. Trottenberg, editors, *Advances in Parallel Computing*, pages 217–220. North Holland, 1998.

[CCP00]Andréa S. Charão, Isabelle Charpentier, and Brigitte Plateau. A framework for parallel multithreaded implementation of domain decomposition methods. In E. H. D'Hollander, G. R. Joubert, F. J. Peters, and H. J. Sips, editors, *Parallel Computing: Fundamentals and Applications*, pages 95–102. Imperial College Press, 2000.

[Cha01]Andréa S. Charão. *Multiprogrammation parallèle générique des méthodes de décomposition de domaine*. PhD thesis, Institut National Polytechnique de Grenoble, 2001.

[Chr96]Nikos Chrisochoides. Multithreaded model for dynamic load balancing parallel adaptive PDE computations. *Applied Numerical Mathematics Journal*, 6:1–17, 1996.

[Lio88]Pierre-Louis Lions. On the Schwarz alternating method. I. In Roland Glowinski, Gene H. Golub, Gérard A. Meurant, and Jacques Périaux, editors, *First International Symposium on Domain Decomposition Methods for Partial Differential Equations*, pages 1–42, Philadelphia, PA, 1988. SIAM.

[MQ89]Luisa D. Marini and Alfio Quarteroni. A relaxation procedure for domain decomposition methods using finite elements. *Numer. Math*, (5):575–598, 1989.

[RHC+96]John V. W. Reynders, Paul J. Hinker, Julian C. Cummings, Susan R. Atlas, Subhankar Banerjee, William F. Humphrey, Steve R. Karmesin, Katarzyna Keahey, Marikani Srikant, and Mary Dell Tholburn. POOMA: A Framework for Scientific Simulations of Paralllel Architectures. In Gregory V. Wilson and Paul Lu, editors, *Parallel Programming in C++*, chapter 14, pages 547–588. MIT Press, 1996.

[Sch90]H. A. Schwarz. *Gesammelte Mathematische Abhandlungen*, volume 2, pages 133–143. Springer, Berlin, 1890. First published in Vierteljahrsschrift der Naturforschenden Gesellschaft in Zürich, volume 15, 1870, pp. 272–286.

[ST96]Michael Shäfer and Stefan Turek. Benchmark computations of laminar flow around cylinder. In *Flow Simulation with High-Performance Computers II*. Vieweg, 1996.

[Ver96]Rüdiger Verfürth. *A Review of A Posteriori Error Estimation and Adaptive Mesh-Refinement Techniques*. Wiley and Teubner, 1996.