Numerical Implementation of Overlapping Balancing Domain Decomposition Methods on Unstructured Meshes

Jung-Han Kimn[1] and Blaise Bourdin[2]

[1] Department of Mathematics and The Center for Computation and Technology, Louisiana State University, Baton Rouge, Louisiana 70803, USA, kimn@math.lsu.edu
[2] Department of Mathematics, Louisiana State University, Baton Rouge, Louisiana 70803, USA, bourdin@math.lsu.edu

Summary The Overlapping Balancing Domain Decomposition (OBDD) methods can be considered as an extension of the Balancing Domain Decomposition (BDD) methods to the case of overlapping subdomains. This new approach, has been proposed and studied in [4, 3]. In this paper, we will discuss its practical parallel implementation and present numerical experiments on large unstructured meshes.

1 Introduction

The Overlapping Balancing Domain Decomposition Methods (OBDD) is a two level overlapping Schwarz method. Its coarse space as well as the projection and restriction operators are based on partition of unity functions. This new algorithm has been presented in [4, 3]. More recently, it has also been extended to the Helmholtz problem (see [5, 3]).

The main goal of this paper is to present an efficient and scalable implementation on large unstructured meshes. The proposed algorithm does not require the construction of a coarse mesh and avoids expensive communication between coarse and fine levels. The implementation we present works on an arbitrary number of processors and does not requires a *a priori* manual decomposition of the domain into subdomains. It relies heavily on the construction of overlapping subdomains and associated partition of unity functions. These functions are used both as a communication mechanism between coarse and fine levels, and as the generating functions for the coarse space. More details on two level overlapping Schwarz methods with partition of unity–based coarse space can be found in [8, 9, 7].

1.1 Notations and presentation of the method

All along this paper, we focus on the implementation of the Poisson problem with Dirichlet boundary condition on a polygonal domain Ω. Given a function $f \in H^{-1}(\Omega)$, and $\partial\Omega_D \subset \partial\Omega$ with a finite number of connected components, we want to solve the problem

$$-\Delta u = f \text{ in } \Omega, u = u_0 \text{ on } \partial\Omega_D. \tag{1}$$

Let $\mathcal{T}$ be a conforming mesh of Ω with N_e elements and N_v vertices, partitioned into N parts $\mathcal{T}_i, 1 \leq i \leq N$ with N_e^i elements and N_v^i vertices. For any $k \in \mathbf{N}$, the overlapping mesh $\mathcal{T}_i^k$ is sub-mesh of $\mathcal{T}$ whose vertices are either in $\mathcal{T}_i$ or linked to a vertex of $\mathcal{T}_i$ by at most k edges. We denote by Ω_i and Ω_i^k the geometric domain associated with these meshes. Lastly, let A be the matrix associated to a discretization of (1). In our experiment, we used a finite element method with linear elements, but this is not a requirement of the method.

The construction of the Overlapping Balancing Domain Decomposition method is similar to that of the well-known Balancing Domain Decomposition method. Its main ingredient is the construction of partition of unity $\theta_i, 1 \leq i \leq$ such that $\theta_i > 0$ on $\mathcal{T}_i^k$, and $\theta_i = 0$ on $\mathcal{T} \setminus \mathcal{T}_i$. Using the function θ_i, we define N weight matrices D_i of size $N_v \times N_v$ whose diagonal vectors are the θ_i.

In this method, the dimension of the coarse space is equal to the number of processors, and the associated matrix A_c is given by

$$A_c(i,j) = \theta_i^T A \theta_j. \tag{2}$$

On each subdomain, the local problems involve solving a local version of (1) with homogeneous Neumann interface conditions. Of course, this is a singular problem, however one can show that the partition of unity functions θ_i generate the null space of the associated local matrix A_i, from which one can easily derive compatibility conditions.

For more details on the theoretical aspect of the method, and for a precise description, refer to [5] and [3].

2 Implementation of the OBDD method on Unstructured Meshes

The Overlapping Balancing Domain Decomposition Method was implemented using an existing parallel finite element package previously written by the second author. The implementation we describe in the sequel is general enough that it should be fairly easy to reproduce in any finite element code. However, some of the technical choices detailed later are dependent on the software packages we used. Namely the unstructured two and three dimensional meshed were generated using Cubit, developed at Sandia National Laboratories [6],

and the internal mesh representation is based on the EXODUS II libraries, also from Sandia National Laboratories. The automatic domain decomposition was obtained using METIS and ParMETIS [2]. Lastly, we used PETSc [1] for all distributed linear algebra needs, and most communication operations.

The OBDD itself was implemented as a shell preconditioner in PETSc

2.1 Construction of the overlapping subdomains and the partition of unity functions

The first step toward the implementation of the OBDD method is to construct the overlapping subdomains and the partition of unity functions, using a non-overlapping domain decomposition computed with METIS. The following algorithm does that in a fully distributed and scalable way.

Let $\mathcal{T}$ be a part of the mesh of Ω we say that a vertex (resp. an element) of Ω is *local* to $\mathcal{T}$ if it belongs to $\mathcal{T}$. We say that an element of Ω is a *near* element for $\mathcal{T}$ if one of its vertices is local to $\mathcal{T}$. Similarly, we say that a vertex $v \in \Omega$ is a near vertex for $\mathcal{T}$ if it belongs to a near element for $\mathcal{T}$, but is not local for $\mathcal{T}$. Lastly, any vertex or element that is neither local nor a near is referred to as *distant*. With these notations, note that Ω_i^k is simply the union of all local and near vertices and elements of Ω_i^{k-1}. This is the essence of our iterative construction.

In the mesh representation system we used, we did not have access to the adjacency graph of the vertices, or the list an element neighbor for instance. Instead, the presented algorithm requires only for each processor to store the entire connectivity table of the mesh.

In order to construct the partition of unity functions and the overlapping subdomains simultaneously, each processor uses a temporary counter d_i of size equal to that of the total number of vertices. At the initial stage, one sets $d_i(v) = 1$ if v is local to Ω, and 0 otherwise. Then, one iterates the following process for $0 \leq j \leq k$: for $1 \leq l \leq N_e$, the element l is near Ω_i^j if $d_i(v) > 0$ at any of its vertices v. Using the connectivity table, compute then the list of all near vertices to Ω. Lastly increment $d_i(v)$ for all v local or near Ω_i^j. After k iteration, $d_i(v) = k + 1$ if $v \in \Omega_i$, $d_i(v) = 0$ if $v \notin \Omega_i^k$. At this point, all that remains to do is to set $\theta_i(v) = d_i(v) / \sum_{j=1}^{N} d_i(v)$.

Figure 1 illustrate the three steps construction of Ω_i^{k+1} out of Ω_i^k. The leftmost figure highlights the local vertices and elements for Ω_i^k. In middle the figure, one has identified the near elements for Ω_i^k. From these near elements, it is now easy to identify the near vertices, as illustrated in the right. All local and near elements for Ω_i^k are the local elements for Ω_i^{k+i}, so that process can be iterated as many times as necessary.

Note that this algorithm is very similar in spirit to a fast marching method (see for instance [10]). Indeed, the functions d_k are the distance to the non-overlapping domains, in a metric where $d(v_i, v_j)$ is proportional to the smallest number of edges linking two vertex v_i and v_j.

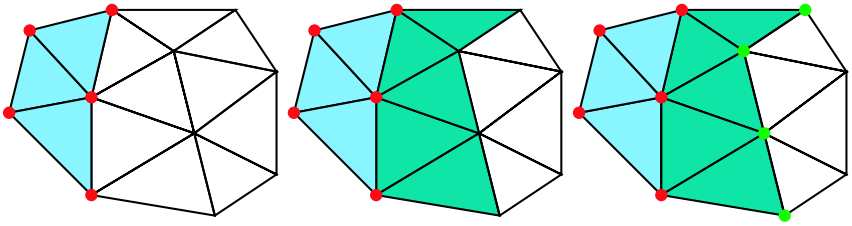

Fig. 1. Extension of the overlap in three steps

Note also that the complexity of this algorithm is independent on the number of processor, and that it requires communication only at its very final stage. The complexity of this algorithm is of order $\mathcal{O}(kN_e)$ and grows linearly with the size of the overlap. As demonstrated in the sequel, a typical overlap choice is of the order of 3 to 5, so the construction of the θ_i is very efficient. However, should one have access to the list of edges of the meshes, or the list of neighboring element to a given one, this complexity would be greatly reduced.

2.2 Coarse problem

The coarse matrix is given by $A_c(i,j) = \theta_i^T A \theta_j$. However, its construction does not require the actual computation of these matrix-vector products. Also, it is easy to see that A_c has a sparse structure, as $\text{supp}(\theta_i) \cap \text{supp}(\theta_j) \neq \emptyset$ only if $\Omega_i^k \cap \Omega_j^k \neq \emptyset$.

In our implementation, we first find all subdomains with non-empty intersection, which give the sparse structure of A_c. Then, for each processor, $A\theta_j$ is obtained from computing $A_j\theta_j$. Then we communicate this vector to all neighboring subdomains so that each processor can assembly its own line in A_c. This algorithm is extremely scalable since it involves only communications in between neighboring processors, and no "all to all" message passing. As illustrated in the experiments in the next section, the OBDD perform best with relatively small overlap. In this case, it is enough to build the adjacency graph of the non-overlapping domains, which is slightly faster. However, this is not true with very large overlap.

Lastly, since the dimension of the coarse problem is relatively small (recall that it is equal to the number of processors), we store it as a sequential matrix in one of the processors, so that each coarse solved can be performed using a direct solver.

2.3 Local problems

Our implementation uses PETSc which does not have data structures dedicated to overlapping submatrices. Therefore, we chose to reassembly the local

matrices A_j instead of extracting them from the global matrix A. Note that this has to be done only once, so it is not very expensive.

As we expect our algorithm to be very scalable, our goal is to use a large number of processors, which means relatively small local problems. For that reason, we use direct local solvers. The cost of the initial factorization is offset by the speed gain in solving the local problems multiple times.

Since we consider local problems with homogeneous Neumann interface conditions, the local matrices are singular. However, their null spaces are given by their associated partition of unity function (see [4, 3] for more details). In the implementation, we still had to add a small damping factor to the diagonal of the matrix, or the local factorization would sometimes fail. This damping factor is typically of order 10^{-10}.

3 Numerical Results

In the numerical experiment we present here, Ω is the square $[-5, 5] \times [-5, 5]$. We considered a homogeneous Dirichlet problem for two different right hand sides: $f(x, y) \equiv 1$ (Problem 1), or $f(x, y) = 1$ if $xy > 0$ and $f(x, y) = -1$ otherwise (Problem 2). The experiment we present, are based on solving both problems for various overlap size k and various mesh sizes. The larger mesh is made of approximately 1,000,000 vertices and 2,000,000 elements (*i.e.* $h \sim .01$). The second one is made of 450,000 vertices and 890,000 elements ($h \sim .015$), and the last one of 250,000 vertices and 500,000 elements ($h \sim .02$). We ran our test implementation on multiple other problems, and got very similar behaviors.

Table 1 display the evolution of the number of iterations of OBDD of Problem 1 and Problem 2. Along the horizontal lines, the ratio between the geometric size of the overlap and the size of the subdomains remains constant while the mesh size varies. As expected, the number of iterations does not change significantly. Along vertical lines, the number of processor is increased. As expected, the number of iterations decreases as long as the number of nodes in the overlap region remains small compared to that in the actual subdomain.

Table 1. Number of iterations for Problem 1 (Problem 2)

♯ of CPUs	Mesh1 (ovlp=10)	Mesh2 (ovlp=7)	Mesh3 (ovlp=5)
32	55 (51)	51 (51)	52 (51)
64	50 (49)	47 (44)	46 (44)
80	45 (43)	47 (46)	50 (48)

Figure 2 represent the evolution of the computation time and of the number of iterations as a function of the overlap. The simulation is computed on

the smaller of the three meshes, and on 32 processors. The first curve corresponds to the total time spend in the solver. In the second one, we subtracted the factorization time for the local matrices. As expected, the total time increases slightly with very large overlaps. However, most of this time increase is due to the local matrices factorization. Indeed, as the overlap size increases, the number of iteration decreases steadily. For all practical purposes, we found no reason to use large overlaps. Overlap sizes of 3-5 typically give the fastest convergence.

In Figure 3, we performed this experiment for various number of processors, on our largest mesh. The same conclusion holds in each case. In our implementation, we assumed that the overlapping subdomains associated to disjoint subdomains were also disjoint, which is not necessarily true with very large overlaps and very small subdomains. For that reason, we were not able to use large overlap sizes with 64 processors.

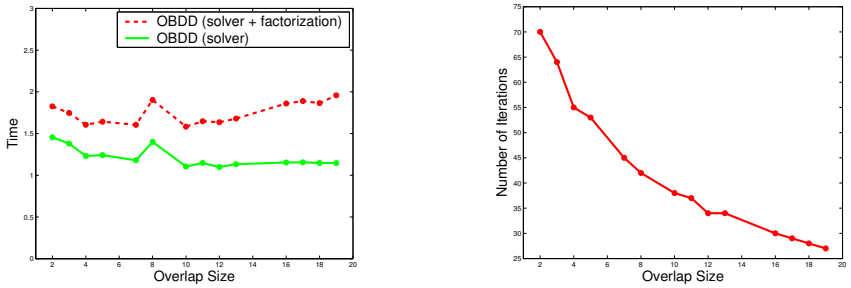

Fig. 2. Times and numbers of iteration versus the overlap size

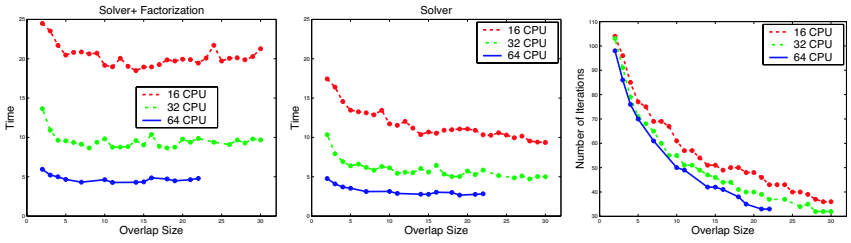

Fig. 3. Time and numbers of iteration versus the overlap size

In Figure 4, we demonstrate the scalability of the Overlapping Balancing Domain Decomposition method, and of our implementation. We solved again Problem 2 with various overlap sizes and up to 240 processors. As expected, both the total time and the number of iterations decrease with the number

of processors. Note also that the gain from an increase of the overlap size is quite minimal.

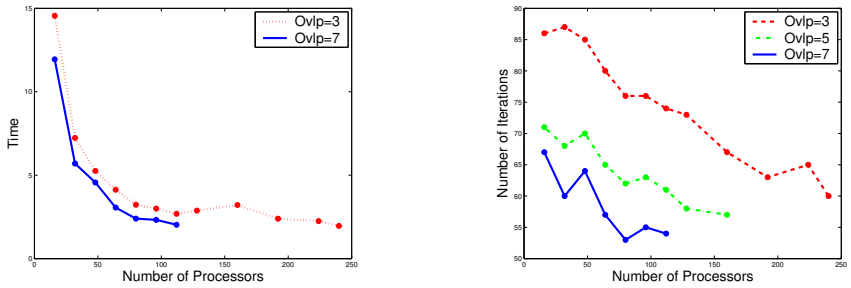

Fig. 4. Scalability of the algorithm and its implementation

The last figure is perhaps the most important. Here, we compared our method with a widely available solver. For our problem, we found that the best combination of solve and preconditioners in PETSc is the Conjugated Gradient with a block-Jacobi preconditioner, iterative local solvers, and incomplete LU local preconditioners. In figure 5, we confront the performances of the OBDD and block-Jacobi preconditioners. In all cases, our algorithm performs significantly better than the best available solver in PETSc.

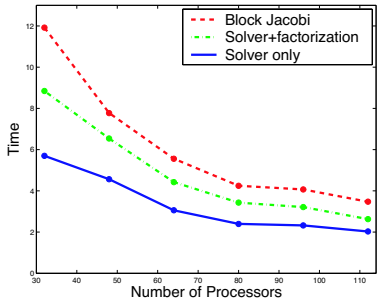

Fig. 5. Comparison of the OBDD and the block-Jacobi preconditioners

References

1. Satish Balay, Kris Buschelman, William D. Gropp, Dinesh Kaushik, Matthew G. Knepley, Lois Curfman McInnes, Barry F. Smith, and Hong Zhang. PETSc Web page, 2001. http://www.mcs.anl.gov/petsc.

2. George Karypis, Rajat Aggarwal, Kirk Schoegel, Vipin Kumar, and Shashi Shekhar. METIS Web page.
http://www-users.cs.umn.edu/ karypis/metis/index.html.
3. Jung-Han Kimn and Marcus Sarkis. OBDD: Overlapping balancing domain decomposition methods and generalizations to the Helmholtz equation. In *16th International Conference on Domain Decomposition Methods, New York*, 2005.
4. Jung-Han Kimn and Marcus Sarkis. Overlapping balancing domain decomposition methods for elliptic problems and for the Helmholtz equation. 2005. In preparation.
5. Jung-Han Kimn and Marcus Sarkis. Overlapping restricted sommerfeld domain decomposition methods and restricted coarse space: A numerical study for the Helmholtz equation. 2005. In preparation.
6. Steven J. Owen. Cubit project home page. http://cubit.sandia.gov.
7. Marcus Sarkis. Partition of unity coarse spaces: Enhanced versions, discontinuous coefficients and applications to elasticity. In Ismael Herrera, David E. Keyes, Olof B. Widlund, and Robert Yates, editors, *14th International Conference on Domain Decomposition Methods, Cocoyoc, Mexico*, 2002.
8. Marcus Sarkis. Partition of unity coarse spaces, fluid flow and transport in porous media: mathematical and numerical treatment. In *Contemp. Math., 295*, pages 445–456, Providence, RI, 2002. South Hadley, MA, 2001, Domain Decomposition Press.
9. Marcus Sarkis. A coarse space for elasticity: Partition of unity rigid body motions coarse space. In Zlatko Drmac et. al., editor, *Proceedings of the Applied Mathematics and Scientific Computing Dubrovnik, Croacia, June, 2001*. Kluwer Academic Press, 2003.
10. James A Sethian. A fast marching level set method for monotonically advancing fronts. *Proc. Nat. Acad. Sci. U.S.A.*, 93(4):1591–1595, 1996.