Advances in Engineering Software 000 (2017) 1–21

Contents lists available at ScienceDirect

Advances in Engineering Software

journal homepage: www.elsevier.com/locate/advengsoft

Coupling parallel adaptive mesh refinement with a nonoverlapping domain decomposition solver

Pavel Kůs[a,b,*], Jakub Šístek[a,c]

[a] Institute of Mathematics of the Czech Academy of Sciences, Žitná 25, 115 67 Prague, Czech Republic
[b] Max Planck Computing and Data Facility, Max Planck Institute, Gießenbachstraße 2, 85748 Garching bei München, Germany
[c] School of Mathematics, The University of Manchester, Manchester, M13 9PL, United Kingdom

ARTICLE INFO

Article history:
Received 31 October 2016
Revised 10 March 2017
Accepted 26 March 2017
Available online xxx

Keywords:
Adaptive mesh refinement
Parallel algorithms
Domain decomposition
BDDC
AMR

ABSTRACT

We study the effect of adaptive mesh refinement on a parallel domain decomposition solver of a linear system of algebraic equations. These concepts need to be combined within a parallel adaptive finite element software. A prototype implementation is presented for this purpose. It uses adaptive mesh refinement with one level of hanging nodes. Two and three-level versions of the Balancing Domain Decomposition based on Constraints (BDDC) method are used to solve the arising system of algebraic equations. The basic concepts are recalled and components necessary for the combination are studied in detail. Of particular interest is the effect of disconnected subdomains, a typical output of the employed mesh partitioning based on space-filling curves, on the convergence and solution time of the BDDC method. It is demonstrated using a large set of experiments that while both refined meshes and disconnected subdomains have a negative effect on the convergence of BDDC, the number of iterations remains acceptable. In addition, scalability of the three-level BDDC solver remains good on up to a few thousands of processor cores. The largest presented problem using adaptive mesh refinement has over 10^9 unknowns and is solved on 2048 cores.

1. Introduction

Adaptive mesh refinement (AMR) is a well-established technique in the framework of the finite element method (FEM). Its use in combination with massively parallel calculations is, however, still rather limited. This should not be surprising, since developing an efficient parallel code employing a general AMR is highly nontrivial and requires meeting several contradicting requirements. In particular, keeping the load of CPU cores balanced is the main issue during the parallel process. Other issues involve management of hanging nodes (or enforced refinements) and a choice of a suitable algebraic solver.

In this contribution, we present an implementation of a massively parallel AMR code. It is coupled to the parallel linear system solver based on nonoverlapping domain decomposition method. The impact of AMR on the solver is evaluated. A subsequent goal of the paper is convincing the reader that hanging nodes treatment and its implementation can be relatively easy and straightforward, as opposed to the general feeling that nonconforming meshes are best to be avoided. The simplicity is achieved by allowing only so called first-level hanging nodes and equal order elements.

The goal of an AMR strategy is to improve the approximate solution in those parts of the domain, where the behaviour of the exact solution is complex, such as close to singularities or within boundary and internal layers. The structure of the mesh becomes complicated while the degrees of freedom are efficiently distributed and their number is kept low. On the other hand, relatively simple, often structured, globally fine meshes are typically preferred in massively parallel codes, resulting in very large number of degrees of freedom.

An integral part of parallel computations is partitioning the computational mesh into subdomains. The strategy employed in our work leads to parts with approximately equal number of elements. Although this requirement may seem rather simple, partitioning adaptively refined meshes in a scalable way on thousands of CPU cores is still a challenging task.

Perhaps the simplest strategy suitable for structured meshes is dividing them into geometrically regular subdomains. However, it cannot be used for adaptively refined subdomains, where the number of elements in different regions can differ largely, leading to huge load imbalance. A widely adopted alternative is translating the problem of dividing the mesh into partitioning the graph of

* Corresponding author.
E-mail addresses: kus@math.cas.cz (P. Kůs), sistek@math.cas.cz (J. Šístek).

http://dx.doi.org/10.1016/j.advengsoft.2017.03.012
0965-9978/

the mesh. The graph partitioning is then performed by standard libraries, such as *METIS* [19] or *ParMETIS* [20]. While the latter is meant for parallel repartitioning, the approach is not scalable to thousands of cores as required by massively parallel adaptive simulations.

A considerably simpler approach is based on partitioning space-filling curves, as it is done in the *p4est* library [10,17], one of the building blocks of our implementation. Its scalability has been tested in [10] up to hundreds of thousands of cores, and its use for development of a parallel FEM library *deal.II* is described in [5]. The authors use a globally assembled system matrix distributed by rows. The main difference of our work seems to be the use of the *subassembled* system matrix, i.e. matrix only assembled subdomain-wise, as it is common to nonoverlapping domain decomposition methods. An interface degree of freedom is present in two or more subdomains, and its assembly is not finalised over the interface. The use of this matrix format naturally avoids partitioning of matrix rows of the fully assembled matrix and thus circumvents the issue with the assembly at hanging nodes in the vicinity of interface described in [10].

On the other hand, properties of subdomains play a more important role in our approach. The drawback of the use of space-filling curves for mesh partitioning is the poor shape of subdomains. Moreover, disconnected subdomains composed of several independent components are common. One of our goals is to investigate how this type of subdomains affects the performance of the nonoverlapping domain decomposition solver.

For the solution of the system of equations, we use the Balancing Domain Decomposition based on Constraints (BDDC) method [14] and its extension to multiple levels, the *Multilevel BDDC* [28,39]. The potential of the multilevel method to scale to 500 thousand cores was recently demonstrated in [4]. Another parallel implementation of multilevel BDDC is available in our open-source *BDDCML* library [35], the second building block of our implementation.

A crucial role in BDDC is played by constraints which enforce continuity of suitably defined *coarse degrees of freedom* across subdomains. Typically, these are point values at selected nodes called *corners* or averages over faces and/or edges among neighbouring subdomains. If a sufficient number of constraints is properly selected, the local Neumann problems on subdomains become uniquely solvable.

The strategy for handling disconnected subdomains employed in *BDDCML* is described. In fact, it is quite simple: a local graph of computational mesh is created for each subdomain and its continuity is analyzed. If more graph components are detected, each of them is treated separately during the selection of constraints. While this may lead to a certain load imbalance due to a sudden increase of number of constraints for disconnected subdomains, the principal amount of work, which is given by the size of each subdomain, remains unchanged. This effect is studied in detail in our paper.

The rest of the paper is organized as follows. In Section 2, we review adaptive mesh refinement strategies and describe a simple approach to dealing with hanging nodes. The BDDC method is recalled in Section 3 with the emphasis on accommodating disconnected subdomains within the method. Section 4 is devoted to combination of these concepts and discussion of specific issues arising in coupling AMR with a parallel BDDC solver. Finally, numerical results testing the developed implementation are presented in Section 5, and conclusions are drawn in Section 6.

2. Adaptive meshes and hanging nodes

The purpose of mesh adaptivity is to ensure precise resolution of details in troublesome areas of the domain while keeping the overall size of the resulting system within reasonable bounds. This often cannot be achieved by uniform refinements. The price to pay are several complications of the discretisation algorithm which have to be addressed. This is particularly true in the case of adaptivity in parallel setting as will be further elaborated in Section 4. There is, however, one particular issue, which is rather challenging even in the case of serial calculation: a proper treatment of the so called *hanging nodes*. These nodes appear when several smaller elements are adjacent to an edge (in 2D) or a face (in 3D) of a larger element, see Fig. 1. It is especially demanding when higher-order elements are used. Various techniques have been developed and used in different settings in the case of serial calculations. However, one has to be careful when dealing with adaptivity in parallel where hanging nodes might appear at subdomain interface.

Hanging nodes present a complication of the numerical scheme, and many authors proposed various techniques to avoid them from the beginning by introducing extra refinements. These *enforced* refinements are not necessary for a better precision of the solution but have the only purpose of keeping the mesh regular, i.e. face-to-face. Let us just mention the red–green algorithm, although many others exist. Nevertheless, these algorithms bring other disadvantages, and it seems that most approaches towards mesh adaptivity involve hanging nodes nowadays. The presence of hanging nodes is quite simply manageable when a discontinuous approximation, e.g. the discontinuous Galerkin method, is used. This can be employed for development of hybrid continuous-discontinuous methods as in [3], where discontinuous approximation is used at hanging nodes.

If continuous approximation is used, the use of *arbitrary-level* hanging nodes (see e.g. [33] for 2-D and [23] for 3-D results or recent works [40] or [36] for alternative approaches) becomes technically somewhat difficult. The *level* here corresponds to the number of subsequent refinements at one side of an element face that were needed for creation of the hanging node, see Fig. 1 for first and second level hanging nodes. The advantage of the arbitrary-level approach is that there are no additional refinements enforced by mesh regularity reasons. It leads, however, to a substantial algorithmic complexity due to the non-local character of constraints on continuity at higher-level hanging nodes and their propagation through the mesh [23]. This holds especially in 3D. For this reason, most authors resort to the use of first-level hanging nodes only (see, e.g., [13]), which seems to be the most feasible solution and which is used in our current work as well.

In the rest of this section, we describe in more detail the way hanging nodes are treated in our solver. The ideas are not new, but we recall them for the sake of completeness and in a way suitable for a subsequent coupling with nonoverlapping domain decomposition. We proceed by imposing three important restrictions, formulated as assumptions on the mesh.

Assumption 1 (2:1 mesh regularity)**.** Only first-level hanging nodes are present in the adaptive meshes.

This important assumption means that no more than two (four) other elements can be adjacent to an element edge (face) in 2D (3D), respectively, see Fig. 1. Consequently, a limited number of enforced refinements can appear in order to fulfill this assumption. This restriction is used by a majority of authors as a reasonable trade-off between performance gain and complexity of implementation.

Assumption 2 (Equal order of elements)**.** Finite elements have a uniform polynomial order.

This restriction rules out the superior *hp*-adaptivity and leaves us with the *h*-adaptivity, however including powerful higher-order approximations.

Please cite this article as: P. Kůs, J. Šístek, Coupling parallel adaptive mesh refinement with a nonoverlapping domain decomposition solver, Advances in Engineering Software (2017), http://dx.doi.org/10.1016/j.advengsoft.2017.03.012