

Parallel and Multilevel Algorithms for Computational Partial Differential Equations

Peter K Jimack*

School of Computing, University of Leeds,
Leeds, LS2 9JT, UK

Email: pkj@comp.leeds.ac.uk Tel.: +44 113 343 5464

ABSTRACT

The efficient and reliable solution of partial differential equations (PDEs) plays an essential role in a very large number of applications in business, engineering and science, ranging from the modelling of financial markets through to the prediction of complex fluid flows. This paper presents a discussion of alternative approaches to the fast solution of elliptic and parabolic PDEs based upon the use of parallel, adaptive and multilevel algorithms. Mesh adaptivity is essential to ensure that the solution is approximated to different local resolutions across the domain according to its local properties, whilst the multilevel algorithms ensure that the computational time to solve the resulting finite element equations is proportional to the number of unknowns. Applying these techniques efficiently on parallel computer architectures leads to significant practical problems. Difficulties addressed in this paper include how to handle the coarse grid operations efficiently in parallel and the dynamic load-balancing problem that arises when the finite element mesh is adapted.

Keywords: Partial Differential Equations, Parallel Computing, Multilevel Algorithms, Adaptive Mesh Refinement.

1. INTRODUCTION

In this paper we discuss the efficient numerical solution of elliptic and parabolic partial differential equations (PDEs) based upon the combination of three core ingredients: multilevel solvers, mesh adaptivity and parallel computing. Each of these topics have been actively and broadly studied in their own right in recent years and so it would be unrealistic to attempt to provide a comprehensive introduction to any of them in a short paper such as this. It is clear however that the use of any of these techniques within a computational algorithm has the potential to yield significant enhancements in computational efficiency. Combining any two of these approaches allows the possibility of further efficiency gains at the expense of increased programming complexity, whilst the use of all three has the potential for yet more improvement in performance provided that a number of challenging technical difficulties can be overcome successfully. In this paper we present some of these technical issues and discuss the author's experiences in attempting to address them.

Section 2 below briefly introduces multilevel solvers for elliptic PDEs. In Section 3 the use of mesh adaptivity is discussed, along with its integration with a fast multilevel solution algorithm. Sections 4 and 5 consider the application of these techniques to parallel/distributed computer architectures, and the paper concludes with a brief discussion.

2. MULTILEVEL ALGORITHMS

When solving elliptic PDEs using finite difference (FD) or finite element (FE) discretizations there is a need to produce a computational mesh and then to solve a large sparse system of algebraic equations corresponding to that mesh. It has long been understood that standard direct methods (based around Gaussian elimination) are very inefficient for these equations when the mesh becomes very fine since the solution time grows as $O(N^3)$, where N is the number of degrees of freedom. For structured meshes it is possible to make use of the bandedness of the corresponding algebraic equations in order to improve matters considerably. Similarly, for unstructured meshes it is possible to order the unknowns so as to approximately minimize the fill-in that arises with a direct solver. In neither case however is it possible to solve the algebraic equations at a cost that is even close to $O(N)$ as N becomes large. Standard iterative methods also suffer from a superlinear increase in cost as N grows, due mainly to the condition number of the algebraic equations deteriorating as the mesh spacing becomes small.

One class of solver that is capable of obtaining solutions to the discrete systems of FD or FE equations at a computational cost that approaches $O(N)$ is based upon what are known as multilevel algorithms. The most well-known of such methods are possibly the multigrid algorithms described in sources such as [8,26] for example. The underlying philosophy behind multilevel algorithms is to decompose the problem, and the solution, into components which occupy different levels of a suitable hierarchy. In the case of multigrid these levels are sequences of finer and finer meshes and the solution may be thought of as being built up of contributions from each of these. In the case of a finite element discretization for example, the FE space on each uniformly refined mesh contains each of the spaces corresponding to the coarser meshes. This idea may be generalized to other nested sequences of subspaces too [9,28]. Multilevel additive and multiplicative Schwarz algorithms are also closely related to the classical multigrid approach, as described in [21] for example.

In the following two sections we describe some multilevel algorithms in a little more detail in the context of local mesh refinement and parallel implementation respectively.

3. MESH ADAPTIVITY

For many problems of practical interest the solution contains features that must be resolved on different length scales in different regions of the spatial domain. In such cases it is common to use mesh adaptivity in order to ensure that the solution may be calculated with sufficient accuracy throughout, but without the need to have an equally fine grid everywhere. Such an approach is certainly necessary in order to obtain the

* URL: <http://www.comp.leeds.ac.uk/pkj>

best possible computational efficiency for a given numerical simulation. One of the most popular forms of mesh adaptivity is known as h -refinement, [19,22], whereby an initial coarse mesh is locally refined to different levels in different regions, as dictated by some measure of the error in the representation of the solution throughout the computational domain. This approach may be applied to steady-state problems, using a succession of local refinements, or to time-dependent problems, using a sequence of refinements and derefinements as a transient feature moves within the spatial domain. An example of the latter is illustrated in Figure 1, which shows a mesh at two different times during the adaptive finite element simulation of the rapid solidification of a pure melt using a phase-field model [27] (courtesy of A. Jones). In either case, if the full mesh hierarchy is maintained then it is possible to combine this adaptive technique with the multilevel solution methods suggested in the previous section.

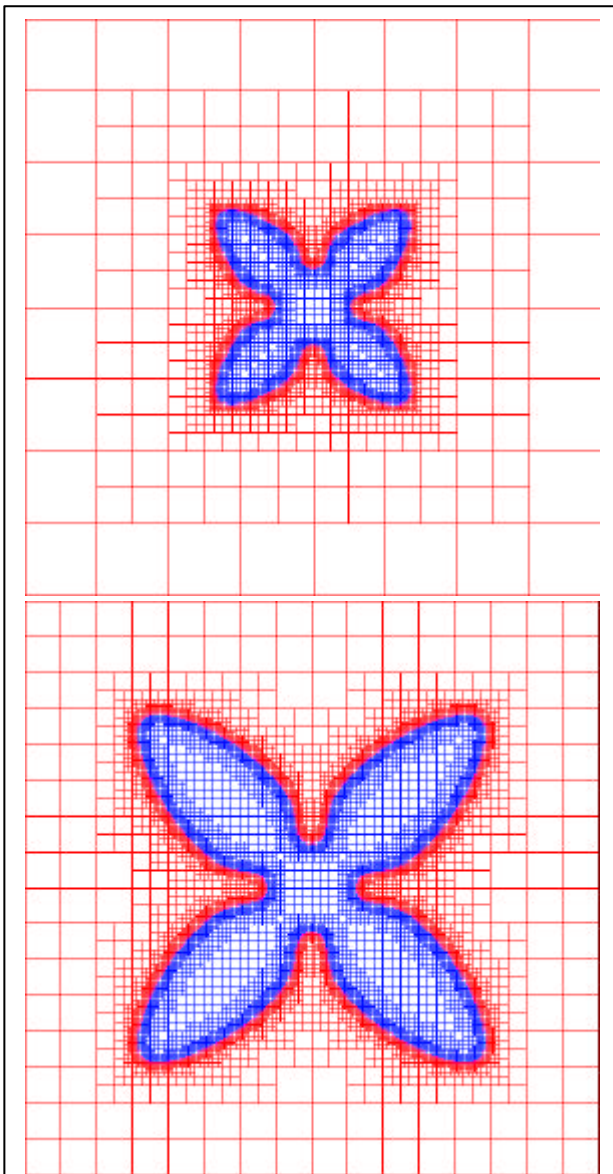


Figure 1: Finite element meshes obtained during the solution of a transient problem using local mesh refinement and derefinement (courtesy of A. Jones).

Work that combines local refinement with the use of multigrid solvers includes the domain decomposition approach of [12] (discussed in more detail in the next section) and the multigrid approaches of [7,10]. In [7] a multilevel adaptive technique (MLAT) is described in which artificial Dirichlet conditions are imposed at the boundary of all locally refined areas during the solution process. All smoothing at the refined mesh levels is then restricted to within these regions using these temporary boundary conditions. In [10] an alternative approach, known as the fast adaptive composite grid (FAC) method, is described. Here smoothing at each level takes place throughout the entire spatial domain, with suitable modifications made at the interfaces between different levels of mesh refinement. For the remainder of this section a very similar technique to this is described.

In [13] an adaptive multigrid tool is introduced for the solution of elliptic and parabolic PDEs in two dimensions, based upon the use of locally refined mesh hierarchies such as those shown in Figure 1. This is based upon the restriction that no two neighbouring elements of the refined mesh may differ by more than one level and so no elements can possibly have more than one *hanging node* on any edge (these hanging nodes are present only at the interface between different levels of refinement and in the case of the quadrilateral elements appearing in Figure 1 may be identified as the interior nodes that are surrounded by just three elements). In [13] the values of the solution at each hanging node is not a degree of freedom but is prescribed to be the average of the solution values at the two nodes at the end of the edge to which it is the midpoint. In the case of a bilinear FE approximation this ensures that the resulting space only contains continuous functions (i.e. it is conforming). It turns out to be relatively simple to modify the conventional full approximation scheme (FAS) multigrid algorithm in order to accommodate these constraints. The basis of the FAS scheme is shown in Figure 2 for a nonlinear elliptic PDE whose discretizations on two consecutive grids (coarse and fine) are given by $L^C(U^C)=0$ and $L^F(U^F)=0$ respectively. Note that only a two-grid version of the algorithm is provided here for simplicity but the generalization to more mesh levels, in the form of a standard V-cycle for example, is easily achieved by applying the same algorithm to the solution of the coarse grid system.

1. Pre-smooth on the fine grid using nonlinear Gauss-Seidel to obtain U^F .
 2. Restrict U^F and $R^F = L^F(U^F)$ to the coarse grid: yielding U^C and R^C .
 3. Set the coarse grid correction $C^C = L^C(U^C) - R^C$.
 4. Solve on the coarse grid: $L^C(V^C) = C^C$.
 5. Set $E^C = V^C - U^C$ (the coarse grid approximation to the error).
 6. Interpolate E^C to the fine grid: yielding E^F .
 7. Update the fine grid approximation: $U^F = U^F + E^F$.
 8. Post-smooth on the fine grid using nonlinear Gauss-Seidel.
- Figure 2: The standard FAS multigrid algorithm illustrated with just two levels.

The modifications to the standard FAS algorithm for the adaptive finite element scheme used in [13] may be summarized as follows. Firstly, it is necessary that the initial estimate of the fine mesh solution lies in the conforming finite element space. Secondly, the nonlinear smoother that is

applied on the fine mesh (steps 1 and 8 in Figure 2) is modified to always project the updated solution into this conforming space. Having obtained this solution and restricted it and the corresponding residual to the coarse mesh (step 2) the calculation of the coarse grid right-hand side (step 3) and the coarse grid solution (step 4) are both slightly modified to ensure that they are consistent with the conforming approximation. Finally, the modified nonlinear smoother is again applied to the corrected solution (step 8). With these modifications in place it is demonstrated in [13] that a nonlinear elliptic test problem may be solved adaptively such that: a) the adaptive solutions are just as accurate as the solutions on equivalent (much larger) uniform meshes; b) the cost of obtaining an adaptive solution is proportional to the number of degrees of freedom in the mesh; c) the solution may be approximated to a given accuracy at a substantially lower cost using the refinement algorithm. The extension to linear and nonlinear parabolic PDEs using implicit time-stepping is relatively straightforward and allows time steps of an arbitrary length to be selected (free from any stability restrictions for example).

4. PARALLEL IMPLEMENTATION ISSUES

So far we have considered multilevel algorithms and their potential for solving PDE problems in a time that is (close to) proportional to the number of unknowns. The need for mesh adaptivity has also been discussed and it has been shown that it is possible to combine this with a multigrid solver, for example, in a manner that preserves the optimal solution properties of the latter. The final component of an efficient PDE solver that we consider here is that of parallelism. This section therefore focuses on the application of multilevel and adaptive algorithms on distributed memory parallel architectures.

Parallel Multilevel Algorithms

The most straightforward approach to the implementation of a multilevel algorithm in parallel is based upon the use of a geometric partition of the domain. This approach is the most popular since it allows standard sequential algorithms, such as the FAS scheme of Figure 2, to be parallelized without any fundamental alterations. Furthermore, since the assembly of the underlying discrete systems of equations is usually undertaken using a decomposed domain, maintaining this strategy for the parallel solver minimizes the need for data movement within a distributed memory parallel code.

Unfortunately, obtaining good parallel efficiency for the distributed memory implementation of multigrid algorithms such as the FAS scheme is a challenging task. This is primarily due to the fact that a significant amount of computation must be undertaken on relatively coarse meshes which results in undesirably large communication to computation ratios at these levels. Furthermore as the number of processors grows, the coarsest mesh that can be split between the processors in such a way that they each have a non-trivial amount of computation to undertake, also increases in size. Hence the implementer is faced with a choice between making the coarsest mesh finer, which may adversely affect the performance of the multigrid algorithm, or building in processor redundancy (through idle processors or replication of work) at the coarsest levels. The latter approach is generally accepted as being more efficient overall, e.g. [14,16].

Another approach for obtaining optimal, or near-optimal, performance within a parallel solver is through the use of

multilevel domain decomposition methods (see, for example, [21]). These typically fall into two categories: multiplicative and additive. Multiplicative multilevel methods are essentially the same as multigrid methods but derived from a different perspective. Additive multilevel methods however are somewhat different, perhaps the most well known of these being the BPX algorithm of [6]. In this approach the computational work is undertaken independently on all mesh levels before being accumulated (rather than one level at a time for multiplicative methods). This allows more flexibility in the parallelization that just using a geometric decomposition (and, in particular, provides a mechanism for overcoming the coarse mesh issues discussed in the previous paragraph). Examples of the parallel implementation of BPX for uniform meshes can be found in [11,23].

Parallel Adaptive Algorithms

Parallel application of adaptive mesh algorithms, such as those discussed in Section 2, has been considered by a number of authors such as [18,20], for example, which both focus on parallel adaptivity for time-dependent problems in three space dimensions. This is clearly the most general case and therefore illustrates well the various difficulties that arise when implementing adaptive mesh algorithms in parallel.

The first issue that arises has already been discussed in the previous subsection and concerns the geometric partitioning of the mesh data structures. Assuming that local hierarchical refinement takes place, starting with an initial coarse mesh, it is necessary to decide if each element in the mesh hierarchy should always belong to the same processor as its parent (this is known as a vertical partition). If this is the case then the coarsest mesh must have sufficiently many elements for it to be partitioned across the processors with at least one element per processor. Furthermore, as we shall see below, this coarsest mesh must actually contain significantly more elements than there are processors. This assumption has been made in both [18,20].

A further difficulty that arises when undertaking mesh refinement (and coarsening) in parallel is that of maintaining consistent data at the partition boundary. It is very simple for each processor to adapt the interior of its mesh (i.e. those entities with no neighbours owned by another processor) in parallel provided these modifications have no effect on any entities that are not in the interior. In practice this situation occurs only rarely however, and so one of the major overheads in parallel adaptivity turns out to be that associated with communicating alterations to a data structure on one processor to all of the other processors that need to know about them. This task is often eased through the use of halo elements (sometimes referred to as ghost elements or cells) surrounding the sub-mesh that is stored on each processor. These halo elements are copies of the sections of the mesh that are immediate neighbours of the sub-mesh that are actually stored on other processors. Even with their use however these communications can lead to a significant overhead.

As well as the problem of ensuring the consistency of the distributed mesh data structures after parallel adaptivity has occurred, the other major overhead associated with parallel refinement and coarsening of meshes is that of maintaining a good load-balance for the parallel solver. In order to achieve this a parallel dynamic load-balancing algorithm is required which is capable of modifying an existing partition of a mesh so that: a) the total load on each processor is about the same; b) the partition boundary is as short as possible; c) the amount of

data that needs to be migrated is as small as possible. Of course these three criteria are not always mutually consistent and even when the third of them is dropped the resulting problem is known to be NP-hard. Despite this a number of relatively good parallel heuristics do exist (e.g. [15,25]) although it is always necessary to be sure that the costs associated with any migration of data between processors will be more than offset by the improved parallel performance on the new partition.

It should be noted that where a vertical partitioning strategy has been selected only mesh objects associated with the coarsest mesh may be migrated between processors. When such a migration occurs the entire mesh hierarchy beneath this coarse mesh entity must be transferred with it. This clearly means that obtaining a perfect load-balance is unlikely to be possible when some coarse mesh elements have been heavily refined, as in [24] for example. Moreover, a significant amount of communication will be associated with the migration of such data objects. For adaptive algorithms involving both mesh coarsening and refinement this communication may be minimized by first undertaking the mesh coarsening, then migrating the coarse grid data objects in a manner that improves the load balance (based upon knowledge of what refinement is about to take place), and then undertaking the local refinement on the repartitioned mesh. This approach has been implemented with noteworthy success in [18].

5. PARALLEL EXAMPLES

There exists very little software that attempts to combine the three features of multilevel algorithms, mesh adaptivity and parallel implementation in anything approaching their full generality. Noteworthy examples, combining all three aspects for linear elliptic problems, include [4,9]. More recently the work of [4] has been extended to deal with time-dependent problems, as illustrated in [5]. In each of these papers good performance has been reported but in the case of [4,5], for example, the software has required many years of development effort.

Recently, in [1,2], a rather different approach to this problem was proposed for the parallel, adaptive, multilevel solution of elliptic PDEs. This is designed to significantly reduce the development overhead associated with producing such software, provided that an efficient, sequential, adaptive multilevel solver is already available. The approach may be divided into three main steps as follows.

1. A preprocessing step in which the problem is first solved on a single (master) processor using a coarse mesh (which must contain significantly more than p elements, where p is the number of processors). An *a posteriori* error estimate is then used to assign a weight to each coarse mesh element before this mesh is partitioned into p sub-meshes, such that each sub-mesh has an approximately equal total weight.
2. Each processor is allocated a unique subdomain before going on to solve the whole problem independently using the sequential, adaptive algorithm starting with the entire coarse mesh. The only restrictions that are placed on these concurrent solves are that they may only refine inside the subdomain that belongs to the executing processor (or in its immediate vicinity in order to maintain a valid mesh) and that the target number of elements in the refined grid is approximately equal for each processor.

3. A communication step is now undertaken in which each processor exchanges information with its neighbours, describing its mesh at the interface between the subdomains. Each processor that finds a neighbour with a more highly refined mesh at their interface then adapts its mesh locally in order to ensure that the meshes match at the interface. The union of the refined meshes on each subdomain then defines the partitioned fine mesh over the whole domain. A coupled parallel solve is then undertaken

Figure 3 illustrates this meshing procedure for a simple two-dimensional problem involving two subdomains (above and below the solid bold line in the bottom two meshes). Local refinement has been undertaken in only a small part of the overall domain however the number of elements within each subdomain is approximately equal. Note that in all cases the meshes have been additionally refined in order to ensure that no edge of any element has more than one hanging node (as described in Section 3).

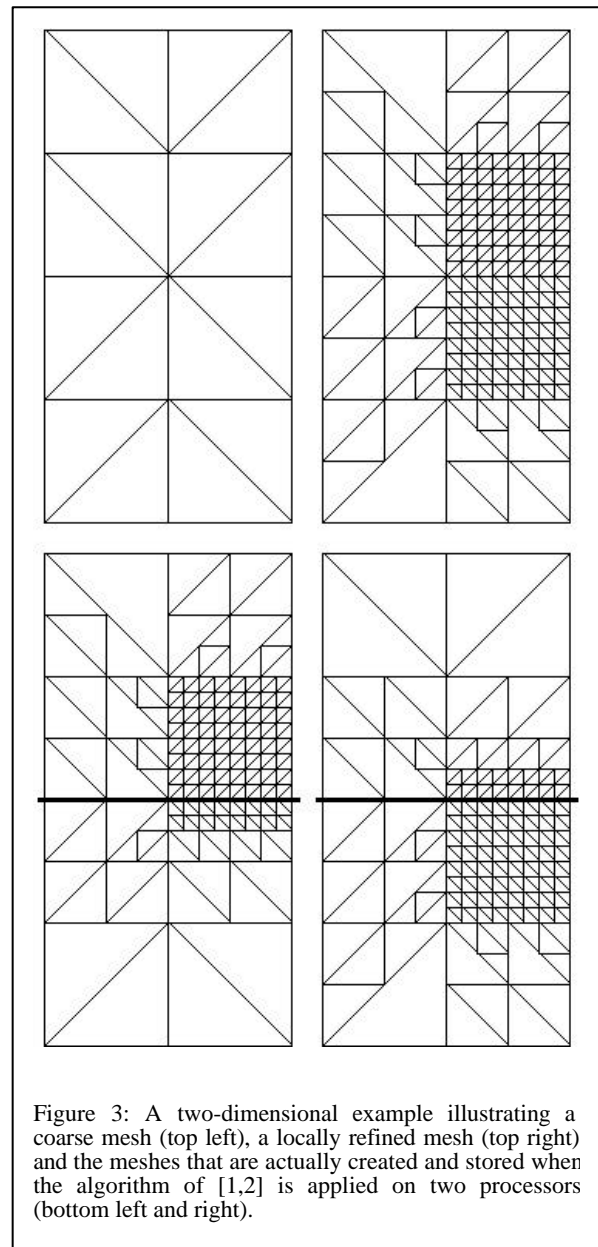


Figure 3: A two-dimensional example illustrating a coarse mesh (top left), a locally refined mesh (top right) and the meshes that are actually created and stored when the algorithm of [1,2] is applied on two processors (bottom left and right).

Having obtained a locally refined mesh on each processor, the final calculation that must be undertaken with this approach (the last part of step three above) is to solve the whole problem in parallel using these meshes. One approach for undertaking this parallel solve is proposed in [1,2] however we now describe an alternative that is introduced in [12]. Both of these approaches are based upon the theoretical results proved in [3] which demonstrate how the solutions on each processor may be combined in a manner that yields the overall solution at an optimal cost, subject to a small number of restrictions on the mesh. These results apply in both two and three dimensions.

The parallel algorithm described in [12] is now summarized. For the simplicity of the explanation it will be assumed that a piecewise linear finite element solution is being calculated using just two processors and that the underlying PDE is linear. For further clarity the reader may find it helpful to consider the two-dimensional case illustrated by the bottom two meshes in Figure 3. Each of the two processors must repeat the iteration shown in Figure 4 until convergence is achieved. As indicated above, in [3] it is proved that for a given class on self-adjoint elliptic PDEs this iteration converges at a rate that is independent of both N and p .

1. Given the latest estimate of the solution find the residual in the subregion owned by the processor.
2. Restrict the residual to the mesh on the other processor that covers the subregion owned by this processor, and send this to the other processor.
3. Receive the residual for the mesh not owned by this processor from the other processor.
4. Given the residual throughout the domain solve for an update of the solution on this processor's mesh using an optimal, multilevel, sequential algorithm.
5. Exchange the update values at the interface and take the average, then update the estimate of the solution.

Figure 4: Outline of the parallel domain decomposition algorithm used in [Refs] for the simplified case of just two processors.

In practice, in [12] the algorithm of Figure 4 is applied as a preconditioner to a GMRES solver rather than as a stationary iteration. Furthermore, it is applied to a class of convection-diffusion equations in three dimensions that is not covered by the underlying theory in [3]. Nevertheless, it proves to be surprisingly robust, as illustrated by the iteration counts shown in Table 1 that are typical of the results in [12]. Furthermore, very creditable parallel performances are recorded, including parallel speed-ups in excess of 12 when using locally refined meshes on 16 processors.

Elements / processors	2	4	8	16
2560	5	6	6	6
9728	5	6	6	6
38400	5	6	7	7
153088	6	7	7	7

Table 1: The number of iterations of the parallel domain decomposition algorithm required to solve a typical three-dimensional convection-diffusion problem in [12].

The iteration counts shown in Table 1 illustrate that the number of iterations of the parallel solver that are required to obtain a converged solution is essentially independent of the level of the finest mesh and the number of subdomains used. Hence, provided the sequential solver used on each processor (at step 4 of the algorithm in Figure 4) has a computational cost of $O(N)$, the total cost of the parallel algorithm will also be approximately proportional to N . This approximation is based upon the assumption that the same coarse grid is always used and that the cost of matching the independently generated meshes at the subdomain boundaries, plus subsequent communication costs, are also $O(N)$. The quality of the load balance that results from the decomposition of the original coarse mesh will not affect this optimal asymptotic performance however it will affect the parallel efficiency which forms part of the constant of proportionality.

6. DISCUSSION

This paper has attempted to introduce and discuss some of the main issues associated with the development of practical software, for the solution of elliptic and parabolic PDEs, that combines the theoretically attractive features of optimal multilevel fast solvers, adaptive mesh refinement and scalable parallel implementation. A variety of multilevel solvers exist for the efficient solution of many elliptic problems and these are also generally applicable for the solution of parabolic equations in combination with (unconditionally stable) implicit time stepping. The parallelization of such solvers is generally undertaken through a domain decomposition approach although more flexibility is possible for additive multilevel algorithms such as BPX: only the domain decomposition methodology is considered here however. When adaptive local refinement is undertaken the different levels in the mesh hierarchy no longer distribute elements uniformly throughout the spatial domain and so the strategy used for the geometric decomposition must become more sophisticated. The use of dynamic load-balancing has been discussed, especially in the context of parabolic problems for which the mesh must be permitted to coarsen in some regions as well as being allowed to refine in others. A suitable strategy is also required for the multigrid smoother used at each level of a locally refined mesh hierarchy: one such algorithm has been outlined based upon the hanging nodes not being treated as true degrees of freedom.

Examples are provided for which all of these considerations have been brought together within a single piece of software however it is noted that this is a highly sophisticated task requiring many years of programming effort. Motivated by this observation an alternative strategy, that is only presently suitable for the solution of elliptic problems, has been described. This strategy assumes that an efficient sequential solver already exists and seeks to integrate it within an outer iteration that is suitable for parallel and distributed computing. The issue of load balance is only addressed in a relatively simple manner however the advantages of this simplicity ensure that a practical parallel implementation may be achieved with relative ease. The two main technical issues that must be addressed are ensuring that the mesh is consistent at subdomain boundaries and accumulating the residual on each processor in the region outside of its own subdomain. In fact the domain decomposition style of algorithm described here has many similarities with an alternative parallel multigrid approach that has been developed in [17]. An interesting challenge that must now be addressed is this extension of this parallel adaptive solution strategy to time dependent problems.

ACKNOWLEDGEMENTS

I would very much like to thank the numerous co-workers with whom I have collaborated on much of the work described in this paper. These include Randy Bank, Martin Berzins, Chris Goodyer, Alison Jones, Sarfraz Nadeem and Mark Walkley.

REFERENCES

- [1] R.E. Bank and M. Holst, "A New Paradigm for Parallel Adaptive Meshing Algorithms", *SIAM J. Sci. Comput.*, Vol.22, 2000, pp.1411~1443.
- [2] R.E. Bank and M. Holst, "A New Paradigm for Parallel Adaptive Meshing Algorithms", *SIAM Review*, Vol.45, 2003, pp.291~323.
- [3] R.E. Bank, P.K. Jimack, S.A. Nadeem and S.V. Nepomnyaschikh, "A Weakly Overlapping Domain Decomposition Preconditioner for the Finite Element Solution of Elliptic Partial Differential Equations", *SIAM J. Sci. Comput.*, Vol.23, 2002, pp.1817~1841.
- [4] P. Bastian, S. Lang and K. Eckstein, "Parallel Adaptive Multigrid Methods in Plane Linear Elasticity Problems", *Numer. Linear Algebr.*, Vol.4, 1997, pp.153~176.
- [5] P. Bastian and S. Lang "Complex Benchmark Computations Obtained with the Software Toolbox UG: Simulation of Transport Around a Nuclear Waste Disposal Site", *Comput. Geosciences*, Vol.8, 2004, pp.125~147.
- [6] J.H. Bramble, J.E. Pasciak and J. Xu, "Parallel Multilevel Preconditioners", *Math. Comp.* Vol.55, 1990, pp.1~22.
- [7] A. Brandt, "Multi-Level Adaptive Solutions to Boundary Value Problems", *Math. Comp.*, Vol.31, 1977, pp.333~390.
- [8] W.L. Briggs, V.E. Henson and S.F. McCormick, *A Multigrid Tutorial* (Second edition, SIAM), 2000.
- [9] M. Griebel and G. Zumbusch, "Parallel Adaptive Subspace Correction Schemes with Applications to Elasticity", *Comput. Methods Appl. Mech. Engrg.*, Vol.184, 2000, pp.303~332.
- [10] L. Hart, S.F. McCormick and A. O'Gallagher, "The Fast Adaptive Composite-Grid Method (FAC): Algorithms for Advanced Computers", *Appl. Math. Comp.*, Vol.19, 1986, pp.103~125.
- [11] B. Heise and M. Jung, "Efficiency, Scalability, and Robustness of Parallel Multilevel Methods for Nonlinear Partial Differential Equations", *SIAM J. Sci. Comp.*, Vol.20, 1998, pp.553~567.
- [12] P.K. Jimack and S.A. Nadeem, "Parallel Application of a Novel Domain Decomposition Preconditioner for the Adaptive Finite-Element Solution of Three-Dimensional Convection-Dominated PDEs", *Concurrency Computat: Pract. Exper.*, Vol.15, 2003, pp.939~956.
- [13] A.C. Jones and P.K. Jimack, "An Adaptive Multigrid Tool for CFD Applications", in *Numerical Methods for Fluid Dynamics VIII*, ed. M.J.Baines et al (Institute of Computational Fluid Dynamics, Oxford), 2004.
- [14] J.E. Jones and S.F. McCormick, "Parallel Multigrid Methods", in *Parallel Numerical Algorithms*, ed. D.E.Keyes, A.Sameh and V.Venkatakrishnan (Kluwer, Dordrecht), 1997.
- [15] G. Karypis and V. Kumar, "Parallel Multilevel k-Way Partitioning Scheme for Irregular Graphs", *SIAM Review*, Vol. 41, 1999, pp.278~300.
- [16] J. Linden, G. Lonsdale, H. Ritzdorf and H. Schuller, "Scalability Aspects of Parallel Multigrid", *Future Gen. Comp. Sys.*, Vol.10, 1994, pp.429~449.
- [17] W. Mitchell, "A Parallel Multigrid Method Using the Full Domain Partition", *Electron. Trans. Numer. Anal.*, Vol.6. 1998, pp.224~233.
- [18] L. Oliker, R.Biswas and R.C. Strawn, "Parallel Implementation of an Adaptive Scheme for 3D Unstructured Grids on the SP2", in *Parallel Algorithms for Irregularly Structured Problems*, LNCS 1117 (Springer-Verlag), 1996.
- [19] N. Provatas, N. Goldenfeld and J. Dantzig, "Adaptive Mesh Refinement Computation of Solidification Microstructures using Dynamic Data Structures", *J. Comput. Phys.*, Vol.148, 1999, pp.265~290.
- [20] P.M. Selwood and M. Berzins, "Parallel Unstructured Tetrahedral Mesh Adaptation: Algorithms, Implementation and Scalability", *Concurrency Computat: Pract. Exper.*, Vol.11, 1999, pp.863~884.
- [21] B. Smith, P. Bjorstad and W. Gropp, *Domain Decomposition: Parallel Multilevel Methods for Elliptic Partial Differential Equations* (Cambridge University Press), 1996.
- [22] W.Speares and M. Berzins, "A 3-D Unstructured Mesh Adaptation Algorithm for Time-Dependent Shock Dominated Problems", *Int. J. Numer. Methods Engrg.*, Vol.25, 1997, pp.81~104.
- [23] M. Thess, "Parallel Multilevel Preconditioners for Thin Smooth Shell Finite Element Analysis", *Numer. Lin. Alg. Applic.*, Vol. 5, 1998, pp.401~440.
- [24] N. Touheed, P. Selwood, P.K. Jimack and M. Berzins, "A Comparison of Some Dynamic Load-Balancing Algorithms for a Parallel Adaptive Flow Solver", *Parallel Computing*, Vol.26, 2000, pp.1535~1554.
- [25] C. Walshaw, M. Cross and M.G. Everett, "Parallel Dynamic Graph Partitioning for Adaptive Unstructured Meshes", *J. Parallel Distrib. Comp.*, Vol.47, 1997, 102~108.
- [26] P. Wesseling, *Introduction to Multigrid Methods* (Wiley), 1992.
- [27] A.A. Wheeler, B.T. Murray and R.J. Schaefer, "Computation of Dendrites using a Phase-Field Model", *Physica D*, Vol.66, 1993, pp.243~262.
- [28] J. Xu, "Iterative Methods by Space Decomposition and Subspace Correction", *SIAM Review*, Vol.34, 1992, pp.581~613.



Peter K Jimack is Professor of Scientific Computing in the School of Computing at The University of Leeds, UK. He graduated from The University of Bristol, UK, with a BSc in Mathematics in July 1986 and a PhD in January 1990, beginning work at Leeds as a Lecturer in Mathematical Software later that year. He has published around 80 papers on topics involving the theory and application of the finite element method, with particular emphasis on adaptive methods and parallel algorithms. His recent publications include work on fast solvers and a wide variety of applications in Computational Fluid Dynamics.