An Object-Oriented Environment for Sparse Parallel Computation on Adaptive Grids

Salvatore Filippone
IBM Italia, Roma, Italy-00144
sfilippone@it.ibm.com

Michele Colajanni
University of Modena, Modena, Italy-41100
colajanni@unimo.it

Dario Pascucci
University of Rome "Tor Vergata"
Roma, Italy-00133

Abstract

Many numerical solutions of large scale simulation models require finer discretizations in some regions of the computational grid. When this region is not known in advance, adaptive meshing *is the most convenient approach because it focuses the computational efforts on the most significant subdomain(s). However, leaving the tasks of implementing adaptive meshing capabilities to the programmer would make the parallelization too much complex. We propose an approach based on an object-oriented library that brings the adaptive meshing capabilities to a wide user community without deteriorating much performance. The software framework includes a runtime support that detects the region requiring a dynamic grid refinement, manages reconfigurable data structures and masks any dynamic reconfiguration to the high-level code.* [1]

1. Introduction

Many engineering and scientific computing applications center around the solution of partial differential equations. The main problem with the most common discretization techniques (i.e., finite difference and finite elements) is the difficulty of a preliminary choice of the appropriate resolution in the computational grid. For example, for a *convection-diffusion* equation applied to an environment with an ongoing chemical reaction, such as a flame front, it is very difficult to choose the proper spacing for a uniform finite difference grid because the presence of a region of high physical activity requires a finer discretization of the grid in that region. Typically, it is impossible to design an appropriate computational grid at compile-time because most real applications do not allow to foresee the location of the activity front. Moreover, for many problems the localization of this region is among the most important results of the simulation itself. There are two viable alternatives: either to refine the grid for the whole computational domain or to refine just the subdomain that requires a more accurate solution namely, *adaptive meshing*. This solution has a lower computational cost. However, it requires a runtime system that automatically detects the region where the refinement is needed, supports dynamic reconfigurations of the data structures used to implement the equation solver, and masks these changes to the high-level code. These tasks make the parallelization of applications with adaptive meshing very complex. We propose an approach based on an object-oriented library that facilitates the implementation of such applications on distributed memory parallel machines. Our solution is based on the PSBLAS library that already contains basic linear algebra routines for sparse computations on MIMD distributed memory computers [7] and a new object-oriented interface implemented in Fortran 90 [9]. The key extension presented in this paper is the automatic support for dynamic grid refinement that provides the user with all data structures and operations necessary to work on dynamically refined computational grids.

The standardization of numerical operations on sparse matrices has received a lot of interest [8, 3]. Moreover, various tools are now available [2, 12], and a number of papers have recently outlined the convenience of object-oriented methods for PDE solvers [14, 5]. However, to the best of our knowledge, the software environment proposed in this paper is the first that brings together a support for adaptive meshing, an object-oriented interface implemented in Fortran 90, and a communication layer based on MPI.

The paper is organized as follows. In Section 2 and 3,

[1]This paper is partially supported by the Ministry of University and Scientific Research (MURST) in the framework of the project "Design Methodologies and Tools of High Performance Systems for Distributed Applications".

we outline the supports for adaptive meshing and the PSBLAS library, respectively. In Section 4, we describe the library interface. In Section 5, we present some experimental results. In Section 6, we summarize the results of this paper.

2. Supports for Dynamic Grid Refinement

The quality of the solution achieved by the iterative methods depends on the number of steps executed during the process; this in turn depends on the tolerance and convergence criterion. Typically, there is a relation between the specified tolerance and the quality of the solution, but even a stringent tolerance may not be enough to get accurate results because of the problems with the discretization of the differential equation. Therefore, it is attractive to dynamically refine the computational mesh around the points for which the quality of the computed solution is not satisfactory. Indeed, it is conceivable to mix solution and refinement steps, starting from a coarse mesh, letting the solution process itself indicate where it is necessary to apply more effort, and then reusing the already computed solution.

Moreover, in a time dependent problem, a dynamic refinement scheme such as this one can easily cope with changes in the location of the high activity zones due to the results of the computation. An example is given by Figure 1. A full exploitation of the computational potential of the algorithm requires not only to refine the grid in certain areas, but also to (re-)coarse them whenever the high-activity front has moved away.

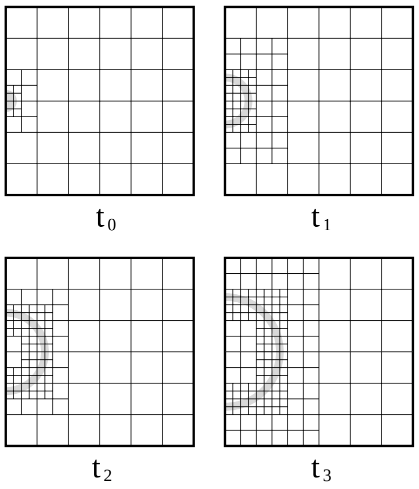

Figure 1. Evolution of a mesh in response to perturbations

In such a situation, balancing the load among the nodes becomes a big issue mainly because the best partition strategy heavily depends on the structure of the new matrix being generated. The optimal solution to this problem is equivalent to an optimal (incremental) graph partitioning problem, which is known to be an $\mathcal{NP}$-complete problem. Therefore, it is necessary to restrict our solution to some heuristics techniques. Two main techniques for implementing this general strategy are *local refinement* [4] and *moving mesh* [11, 1].

We have realized the support under the assumption that the physical model is implemented through a particular class of adaptive meshes, namely *recursive consistent meshes* [13]. They are obtained starting from a regular, rectangular mesh called the *root mesh*. Each cell may be divided into four rectangles called *descendant cells*, while the origin cell is also called *parent cell* or *super-cell*. Therefore to each cell we can assign a *level*. Cells in the root mesh are at level 0, whereas any descendant cell is at a level higher than the level of the parent cell. A cell with no descendants is called *terminal*.

A mesh is *consistent* if the size of any two adjacent terminal cells differs at most by a factor of 2. This is a very convenient property, because it drastically reduces the number of possible configurations in the neighborhood of a given grid point, thereby simplifying the discretization techniques and data structures. To maintain the consistency of a grid throughout the computation, it is necessary that the refinement strategy depends not only on the value of the indicator function, but also on the context of a given cell. The discretization process on a recursive consistent mesh is very similar to that of a uniform mesh, except for the boundaries between subgrids at different refinement levels. At such boundaries the application of a discretization scheme requires an interpolation to determine values in non-existing nodes. If we are working with a second-order discretization, it is necessary to employ a quadratic interpolation scheme to avoid a degradation in the accuracy. For the same reason, a finite element application would require additional continuity conditions.

3. PSBLAS Library

The Parallel Sparse BLAS library [7, 9] has been designed and implemented to alleviate the problem of a proper implementation of implicit solvers on distributed memory machines. It contains numerical kernels and support operations oriented to the implementation of iterative solvers for sparse linear systems (the main kernels in implicit PDE solvers). Once the user has satisfied the minimal requirements of choosing a (completely arbitrary) data distribution and writing the linear equations in terms of global indices, the PSBLAS library takes care of all data structures necessary to synchronize the parallel operations, the local renumbering of the equations and the local storage format of the matrices. A detailed description of the PSBLAS library can be found in [9].

The data structures play an essential role in the implementation of the PSBLAS environment. We rely heavily on the Fortran 90 capabilities to define the necessary interfaces. The PSBLAS library handles two main types of data structures: the (local part of) sparse matrices and the global matrix descriptors. The former describes the characteristics of the local storage necessary to perform the various operations from a local point of view, specific to each processor. The latter takes care of blending the various local views in a consistent way to define the global system of equations.

The format for the local storage is compatible with the proposal for the serial sparse BLAS [6, 8]. The relations among the various processors and the parts of the sparse matrix held by each processor are described in a decomposition data structure `DECOMP_DATA_TYPE` which contains the lists of indices defining the role of each grid point with respect to the current processor, and how/when to exchange data with other processors. A full description of the format of these vectors is contained in [9].

The basic data structures have been extended to handle a global matrix capable of dynamic refinements. For the *local* matrix storage we have identified two possible actions: oriented at the matrix level or at the single equation level. Our implementation allows a dynamic addition and deletion of submatrices. Single rows of a given matrix can only be modified, but never deleted. The resulting data structure is an array of sparse (sub)matrices. As in the static case, the library takes care of the relations between different processors in the parallel machine. Some of the data items contained in these structures have the same value across all processors, because they define global characteristics of the application. Other items have a local meaning only. In both instances, the pre-processing routines carry out extensive consistency checks during the setup phase.

4. Library Interface

The software environment shields the user from most implementation details. It relies on Fortran 90 object-oriented features to encapsulate support tools and computational routines. To make the calling sequences as simple and consistent as possible, we extensively use the object-oriented capabilities of the language such as polymorphism and operator overloading. Moreover, we support static and dynamic distributed sparse matrices that contain either real or complex long precision numbers. The interface consists of two main classes of routines: *support tools* and *computational routines*.

The support tools are routines that enable the user to build a distributed application with minimal prerequisites. Basically, there are only two operations left to a user that is, deciding where each equation is allocated in the process grid, writing an equation in coordinate format.

`F90_PSSPDECINIT` Decomposition data structure initialization.

`F90_PSSPMATALL` Sparse (sub)matrix allocation. It appends a (new) sub matrix to the existing sparse matrix, allocating memory space as necessary.

`F90_PSSPMATINS` Sparse Matrix insertion. It defines the new coefficients entering the sparse matrices.

`F90_PSSPMATASB` (Sub)Matrix assembly. After the insertion phase, it assemble all contributions, doing format conversions where appropriate.

`F90_PSSPDECASB` Data assembly. It must be called before computations to achieve optimizations on the parallel environment structures.

`F90_PSSPMATUPD` Sparse (sub)matrix update. It updates an existing matrix instead of creating a new one (this operation is often convenient from a performance point of view).

`F90_PSSPMATCLN` Sparse matrix deletion. Subgrids and their corresponding submatrices may be deleted; for efficiency reasons, deletions may only be applied in a LIFO fashion.

`F90_PSSPDECFREE` Data structure deallocation. It prevents memory leaks.

The sequence of operations in a typical application is shown in Figure 2. Choice such as 'it is preferable to refine/update a matrix' or 'go forth and solve the corresponding system' should be taken under suitable criteria that depend on the application.

The interface of the computational routines is identical to that of the previous version of the PSBLAS library. The new data structures are handled by overloading the corresponding calling sequences and providing appropriate interfaces that manage the new data types. The main operations are outlined below.

- Sparse matrix-dense matrix products `PSSPMM`: $C \leftarrow \alpha P_R A P_C B + \beta C$
- Dense matrix sums `PSAXPBY`: $Y \leftarrow \alpha X + \beta Y$
- Scalar products `PSDOT`: $x^T y \quad$ or $\; x^H y$
- Dense vector 1, 2 and infinity norms `PSASUM`, `PSNRM2` and `PSAMAX`: $||x||$
- Sparse matrix infinity norm `PSNRMI`: $||A||_\infty$

For a detailed description of the calling sequences and parameters the reader can refer to the PSBLAS user's guide[2].

[2]URL: http://www.ce.uniroma2.it/psblas

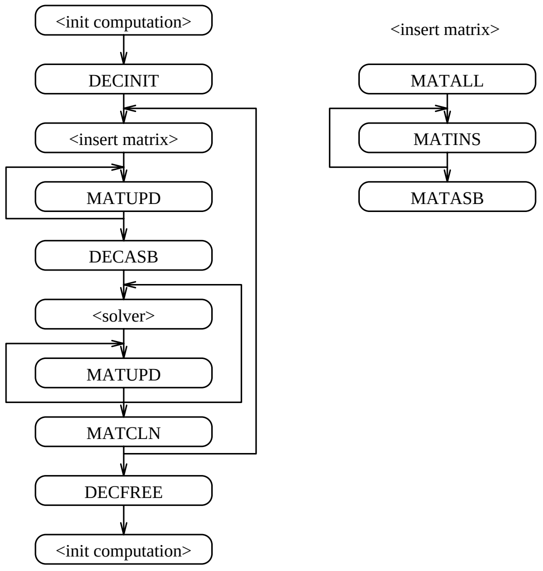

Figure 2. Typical application structure.

5. Experimental Results

We have implemented a simple grid refinement strategy applied to various test cases. The experiments were carried out on an IBM SP2, with POWER2 wide nodes. In this section, we describe the details of the solution of a 2-D Dirichlet problem for a parabolic, time-independent differential equation of the convection-diffusion type ($0 < x, y < 1$):

$$R\frac{\partial u}{\partial x} = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + \omega^2[1 - e^{R(x-1)}]\sin(\omega y), \qquad (1)$$

with boundary conditions $u(x, 0) = u(x, 1) = u(1, y) = 0$, and $u(0, y) = [1 - e^{-R}]\sin(\omega y)$, so that the analytical solution is $u(x, y) = [1-e^{R(x-1)}]\sin(\omega y)$. The parameters in the equation have been chosen as $R = 35$, $\omega = \pi$. The solution has a steep gradient on the edge $x = 1$, and the analytical solution allows us to estimate the quality of the numerical solution.

The equation (1) has been discretized on a uniform orthogonal grid; the discretization is obtained through second-order formulas for first and second derivatives. On the root grid, we have one equation and one unknown for each point (k, m), $0 < k < K + 1$ and $0 < m < M + 1$. On the subgrids, it is possible to have equations for points with $k = 0, K + 1$, $m = 0, M + 1$, because the boundaries of a subgrid may be located internally to the parent grid.

To evaluate the grid refinement approach, we have implemented a test program employing a dynamic grid refinement even in a static scenario. In such a way, the user can only describe the accuracy required without specifying *a priori* grid resolution method. The environment will allocate the grid points in the most convenient way. The error indicator is based on the assumption that in an optimal grid the quantity [15]

$$\int_{r_i} D^2_{max} u \qquad (2)$$

should be uniform on all elements r_i, where

$$D^2_{max}u(x, y) = \max\{|u_{xx}(x, y)|, |u_{xy}(x, y)|, |u_{yy}(x, y)|\}.$$

During the computation, at the end of each solution step, all elements for which the value computed according to (2) is above the user defined $E_{threshold}$ are labeled and refined. The refinement process ends when the criterion is satisfied on all gridpoints, or too many refinement steps have been carried out without reaching convergence.

To implement efficient refinement steps, it is necessary to identify large *clusters* of gridpoints that have to be refined. We use a clustering algorithm that is frequently adopted for AMR techniques [4] because it represents a good compromise between simplicity and performance. This algorithm can be easily formulated in a recursive way, as shown in Figure 3. Given a grid (a), we determine the smallest rectangular subgrid (b) containing all labeled elements. If the percentage of labeled points in this grid is large enough, we apply the refinement step, otherwise we decompose in four subgrids (c), and recurse on them (d).

The linear systems resulting from the discretization process are solved through the BiCGSTAB method, with a block ILU preconditioner. The stopping criterion is based on the *normwise backward error* [10].

The workload is distributed by assigning one subdomain to each processor. The subdomains are chosen so that each of them contains an approximately equal number of points per refinement level. The experiments focus on two important aspects of system performance. We compare the adaptive solution with a uniform mesh of comparable precision in terms of gridpoints and execution time. Then, we study the scalability of our program with adaptive meshing for a growing number of processors. Table 1 shows the timing results for a series of tests. The same model problem has been solved with grids of various size on different processor configurations. The adaptive algorithms has been configured so as to execute three refinement/solve steps. The same quality of the solution would have required a uniform grid of the size indicated in the column labeled “Equiv. uniform grid.” The experimental results show that the adaptive algorithm

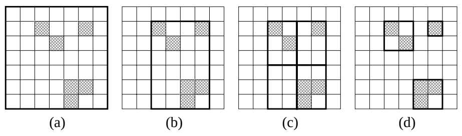

Figure 3. Clustering algorithm

Uniform grid solution			
Grid size	200×200	400×400	800×800
Linear system size	40000	160000	640000
1 proc [sec]	12.37	86.92	630.35
2 procs [sec]	8.90	56.05	369.06
4 procs [sec]	7.00	34.10	221.86
8 procs [sec]	7.50	26.01	131.51
Error ($\|\hat{\mathbf{x}} - \mathbf{x}\|_\infty$)	10^{-2}	$2 \cdot 10^{-3}$	$9 \cdot 10^{-4}$

Adaptive mesh			
Root grid size	100×100	200×200	400×400
Equiv. uniform grid	400×400	800×800	16000×16000
Size 1st system	10000	40000	160000
Size 2nd system	13624	53846	212888
Size 3rd system	22099	86587	337023
1 proc [sec]	12.32	89.10	855.47
2 procs [sec]	10.91	76.13	377.81
4 procs [sec]	11.65	43.62	223.89
8 procs [sec]	13.49	41.66	167.76
Error ($\|\hat{\mathbf{x}} - \mathbf{x}\|_\infty$)	$1.7 \cdot 10^{-3}$	$8.5 \cdot 10^{-4}$	$8 \cdot 10^{-4}$

Table 1. Execution results

is superior in terms of performance given the same level of required precision. Indeed, the refinement strategy successfully identifies the high activity region in the computational domain. Scalability results are also quite satisfactory. Given the same number of grid points, the adaptive algorithm has the same behavior with respect to a growth in the number of processors. It is worth to remark that, with the same total number of grid points, the adaptive grid would provide a much better precision in the solution. For the adaptive case on a 400×400 root grid, we can also observe a slight superlinear speedup due to memory and cache effects.

6 Conclusions

Parallelization of numerical simulations with adaptive meshing is a very complex task. To bring this capability to a wide user community, we propose an object-oriented library that masks most implementation details related to dynamic mesh refinement. This support integrated into the PSBLAS library [7, 9] represents the first software framework for message-passing parallel computing that provides an automatic support for adaptive meshing, an object-oriented interface written in Fortran 90, and a communication layer based on MPI.

References

[1] D. C. Arney, J. E. Flaherty. An adaptive mesh moving and local refinement method for time-dependent partial differential equations. *ACM. Trans. Math. Softw.*, 16(1): 48-71, Mar. 1990.

[2] S. Balay, W. Gropp, L. Curfman McInnes, B. Smith. PETSc 2.0 - User Manual. Tech. Rep. ANL-95/11 - Revision 2.0.22, Argonne National Laboratory, 1995.

[3] R. Barrett, M. Berry, T. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, H. van der Vorst. *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods*, SIAM, 1993.

[4] M. J. Berger, J. Oliger. Adaptive mesh refinement for hyperbolic partial differential equations. *J. Comput. Phys.*, 53: 484-512, 1984.

[5] A. M. Bruaset, H. P. Langtangen. Object-oriented design of preconditioned iterative methods in diffpack. *ACM. Trans. Math. Softw.*, 23: 50-80, Mar. 1997.

[6] S. Carney, M. A. Heroux, G. Li, K. Wu. A revised proposal for a sparse BLAS toolkit. *SPARKER Working Note # 3*, Mar. 1994.

[7] F. Cerioni, M. Colajanni, S. Filippone, S. Maiolatesi. A proposal for parallel sparse BLAS. *Applied Parallel Computing* (Wasniewski et. al. editors), pp. 166–175, Lyngby, Denmark, August 1996. Springer-Verlag, LNCS No. 1184.

[8] I.S. Duff, M. Marrone, G. Radicati, C. Vittoli. Level 3 basic linear algebra subprograms for sparse matrices: A user-level interface. *ACM. Trans. Math. Softw.*, 23(3): 379-401, Sept. 1997.

[9] S. Filippone, M. Colajanni, S. Maiolatesi. PSBLAS: A Library for Parallel Linear Algebra Computation on Sparse Matrices. Tech. Rep. DISP-RR-98.5, Dept. of Computer Engineering, University of Roma Tor Vergata, Mar. 1998. (http://www.ce.uniroma2.it/psblas/submitted2.ps)

[10] N. J. Higham. *Accuracy and Stability of Numerical Algorithms*. SIAM, 1995.

[11] W. Huang, Y. Ren, R. D. Russell. Moving mesh partial differential equations (MMPDES) based on the equidistribution principle. *SIAM J. Numer. Anal.*, 31(3): 709-730, June 1994.

[12] S.A. Hutchinson, L.V. Prevost, J.N. Shadid, C. Tong, R.S. Tuminaro. Aztec User's Guide - Version 2.0. Sandia National Laboratories, July 1998.

[13] G. Kozlovsky. An interactive programming environment for solving partial differential equations using adaptive grids (with application to flame-flow interaction problems). Tech. Rep. CCNY-CS1291, The City University of New York, Dept. of Computer Sciences, New York, Dec. 1991.

[14] L. Machiels, M. O. Deville. Fortran 90: An entry to object-oriented programming for the solution of partial differential equations. *ACM. Trans. Math. Softw.*, 23: 2-49, Mar. 1997.

[15] W. F. Mitchell. A comparison of adaptive refinement techniques for elliptic problems. *ACM. Trans. Math. Softw.*, 15(4): 326-347, Dec. 1989.