Some Issues in the Development of Overset Grid CFD using One-Sided Communication

YikLoon Lee  
Naval Air System Command  
Patuxent River, MD 20670

ABSTRACT: Overset (overlapping) grids are very useful for CFD problems with multiple moving bodies. The parallelization of the grid connectivity algorithm with a two-sided communication model like MPI, however, imposes formidable challenge. One of the reasons is that only a few remote data are known to be needed at any one time in the cell search process. Another reason is the need to immediately ‘push’ some variables whenever they are determined. The one-sided model is well suited for this problem. For boundary value interpolation, however, its advantage is being offset by the need for more synchronization and a lot more data to be passed. This paper describes the parallelization of a Navy overset CFD code using Co-array Fortran on the Cray X1E and its comparison with a two-sided model.

KEYWORDS: Overset grid connectivity, one-sided communication

1. Introduction

The design and generation of a suitable grid that covers the computational region in the numerical simulation of partial differential equations (PDE) is the most labor intensive and time consuming part of the Computational Fluid Dynamics (CFD) analysis process. This is especially so for problems with moving body components and complex geometries. Some applications of this type that are of practical interest to the aerospace community include rocket stage separation, pilot ejection, rotorcraft [1], aircraft store separation, turbomachinery, airdrop, etc. It has also seen applications in other areas such as ships/submarines, road vehicles, biological flow, environmental flow, oceanography, etc.

Overset Grids

The grid generation task for these problems can be made simpler through the use of overset (i.e. overlapping) component grids [2, 3]. In this method, the entire grid is selectively broken into many smaller and arbitrarily overlapping parts generated independently of each other. Overset grids can be used for bodies like a wing, tail, nose, blade, store, etc., or for refining a flow feature like a vortex, shock, etc (see Fig. 1 for a 2D example). Because of their arbitrary motion and orientation, these component grids would generally overlap each other. The entire region would be free of any void or gap if one of them is a large background grid. Otherwise care should be exercised to avoid any gap.

While there is some grid redundancy and wastage in the overlapping region, this is usually small and typically not much more than 5-15% of the total grid points. Overset grids are important for moving bodies with very large motion because they are allowed to move past each other without having to repeatedly and increasingly deform the grid in order to maintain water-tight connection between cells. They are especially well suited for rotorcraft and other problems with large and high speed grid motion. It is noted here that another method that has been used for moving bodies is the unstructured dynamic remeshing [4]. Interestingly, it too, like overset grids, reportedly requires one-sided communication for its development.
Since the component grids that make up the entire grid are independently generated, many grid boundaries arise that need boundary conditions from possibly anywhere in the flow field. These include newly generated boundaries around holes or cuts in the grids (called hole fringes) which are frequently generated in overset grids (Fig. 2 and 3), especially if relative motion is present. The overset grids method, in a nutshell, is all about locating and supplying these boundaries with the correct boundary condition.

The simplicity of overset grid generation, however, comes with the hefty price of determining exactly where the component grids overlap – which boundary and hole fringe point falls in which cell (called the donor cell) and where in that cell, and which points fall inside the bodies (e.g. the aircraft) and can be discarded. This information is required to facilitate exchange of boundary condition among component grids. The operation to determine this is called overset grid connectivity and has to be performed every time step for moving body problems. The input data required are, first and foremost, the cells’ nodal coordinates. The output from the algorithm is a complete list of these boundary and hole fringe points (their indices and grid numbers) and their donor cells (indices, grid numbers and locations of the points in the cells).

Fig. 1 A 2D example of overset grids – multiple holes with complicated shape cut out in each of 5 grids (3 nested square, 2 wing airfoils). Hole points are mostly redundant.

Fig. 2 The fringe around the hole cut out in the square grid is the newly generated boundary.

Fig. 3 A 3D example of holes in 3 rectangular overset grids hollowed out by two aircraft.
From the point of view of the code developer, technical codes can be broadly classified into either one of two categories – a) numerically intensive and b) logically complex. A solver that attempts to solve a set of governing equations belongs to a). Its development is usually logically easier than a code belonging to b). However, its verification and validation is harder and, in some problems, uncertain and inconclusive. Also the code is usually smaller in size but needs a much longer execution time even on a supercomputer. A logically complex code, in contrast, has mostly the opposite characteristics – a logically much more difficult development effort (with few or no equation to solve) but with easier verification criteria. The code could be large and may not need a supercomputer to run. An example would be an automated baggage handling system. The overset connectivity algorithm is most appropriately classified as b) logically complex. Like a), however, its execution time requires a supercomputer.

**Communication Models**

In the context of numerical PDE, parallelization can be implemented using one of two models of processor communication (shown schematically in Fig. 4) – i) one-sided or ii) two-sided. While the latter is commonly implemented with the industry standard MPI library, the former, e.g. Co-array Fortran (CAF) [5], is based on a simple language extension to arrays called co-array. The parallel programming in the overset CFD code partly described in this paper is based on CAF. These two communication models give rise to very different algorithm, programming efficiency and communication efficiency for overset grids CFD code. The most significant conceptual difference is that, to establish communication, the two-sided model requires coordination with the remote processor while the one-sided model does not. In other words, remote data access in the one-sided model does not involve the remote processor, at least in concept. In a Cray X1, one-sided model is supported in hardware with its global memory address space. This difference in concept has particularly important implications for the overset grids methodology.

![Schematic diagram of one- and two-sided communication models](image_url)

**Data Communication**

Presently, most of the examples studied using PGAS (Partitioned Global Address Space) languages or one-sided communication models are non-overset grids such as that in [4, 6]. The PGAS and overset grids communities are relatively small and rarely interact with each other. The application of the one-sided model to overset grids using the grid connectivity algorithm as described [3] is virtually non-existent. But the combination of the two technologies promises to provide a rich environment for research and study.

Compared with conventional non-overset grids (structured or unstructured), the necessity of overset grid connectivity presents the code developer with a vastly different parallelization and data communication scenario. In a non-overset grid, the entire domain is decomposed into non-overlapping subdomains, each assigned to a different processor. An extra layer of ‘ghost’ cells, or points, are then duplicated at the subdomain boundaries for holding received data. Every boundary point and ‘ghost’ point has a one-to-one exact match with those in its neighboring subdomains, whose identities are always known and remain the same throughout the computation. There is, therefore, only one type of data communication in non-overset grids, namely, that for the boundary values in the solver. The data are transferred in one bulk communication at the end of every time step and are directly injected into the boundary points with no need for interpolation. Either one-sided or two-sided communication model would work well here. For non-overset grids, there is a clear separation between computation and communication.

In overset grids, on the other hand, the outer boundaries and hole fringes of the component grids generally do not match up with (i.e., lie on top of) any other points in the neighboring grids. Because of that, they must obtain their boundary conditions through interpolations from their bounding cells (called donor cells). Although this characteristic is the source of many challenges, it is also one of the reasons for the flexibility of overset grids. Unlike non-overset grids, the identities of the overlapping or neighboring grids are not known a priori and must be computed from the connectivity algorithm. Furthermore, they don’t remain the same for moving grids and must be repeatedly established at the beginning of every time step.

The primary difference with non-overset grids is the non-matching of boundary points and its associated need to interpolate boundary values instead of direct
injection of the values into the points. This leads to the communication step being tightly integrated with the solution interpolation, thus having a profound effect on processor synchronization as described in section 3.

In addition to the overset solver, the connectivity algorithm also needs to be parallelized. Like the solver, it is parallelized with the same distribution of the component grids. One of its primary and most time consuming operations is the search for the donor cells of all the boundary points. Thus from the operations of both the solver and connectivity algorithm in an overset CFD code, it is apparent that the two need different types of data communication: (i) the grid boundary values in the flow solver, and (ii) the cell coordinates in the grid connectivity that are needed for the determination of the boundary values. Like the non-overset case, either the one-sided or the two-sided communication model can be selected for the solver in (i). In the overset case however, these two models yield codes that have completely different and opposite characteristics. The distinction is so significant that it will be discussed further in section 3.

It is noted here that, like the decomposition of a non-overset grid described above, an overset component grid can also be further decomposed into subdomains. This is desirable when the grid is large and decomposing or splitting it into smaller subdomains for redistribution would improve load balance. The presence of decomposed grids within overset grids would then encompass all types of communication scenarios.

The main body of this paper is divided into two sections describing the data communication scenarios in the overset grids connectivity and boundary interpolation.

2. Communication in Overset Grids
Connectivity

As mentioned before, one of the main operations in the connectivity algorithm is the search for the donor cells of all boundary and hole fringe points. A simple and exhaustive search across the whole grid, however, is prohibitively expensive. The primary means to make the algorithm more intelligent and accelerate the operations is the determination of the overlap status of every component grid-pair. Knowledge of this status allows the elimination of unnecessary search. Of the several different possible statuses, this paper will describe only the most obvious one – the non-overlapping status. As detailed in the next two subsections, whereas local donor search needs to actively access remote data, the local processor is passively given the overlap status determined remotely. Since only one-sided model is practically feasible, two-sided model is not described or even attempted.

Donor Cell Search

Searching for the remote donor cell for a local boundary point is an iterative process (Fig. 5). A starting cell is first selected and tested for the inside/outside status of the point. The data that need to be accessed from the remote processor for the test are the cell’s nodal coordinates. If the point is found to be outside the cell, the search moves on to one of the neighboring cells, the choice of which is determined by the results of the test. The nodal coordinates of this neighboring cell are remotely accessed and tested again. The process repeats itself until either a donor cell is found or a dead end is reached. In a problem with moving grids, the identity of the donor cells for the boundary points may change every time step as these points move in and out of the cells.

It is clear from the above donor cell search process that only a small amount of data (i.e., nodal coordinates) are known to be needed at any one time. Their identities are determined from the outcome of the computation in the test and they are constantly being needed for all the boundary points and tests. Data access pattern like this cannot be efficiently facilitated by the two-sided communication model which is more suitable for bulk data transfer. Of course, all or most of the data can be transferred a priori in a bulk communication. But this is hardly an efficient solution in terms of memory and CPU time.

![Fig. 5 An iterative search for the donor cell](image)

Overlap Status

An overset grid system may contain dozens or even hundreds of grid components in any arbitrary arrangements. Many of them are far apart with no overlap and thus no possible donor cell. The expense of an exhaustive search for all donor cells from all available grids for a boundary point increases significantly with the number of grids even if the total number of points remain constant. For practical and economical implementation,
the connectivity algorithm must have a certain degree of intelligence to eliminate at least some fruitless search. This is accomplished through the determination of the grid overlap status.

As one example, consider the initial operation for a non-overlapping grid-pair – boundary points from the first grid searching for possible donor cells from the second grid. Not knowing that the two grids do not overlap, the search would proceed assuming otherwise until the last boundary point is completed and the non-overlapping status confirmed. Equipped with the new knowledge of the status, the subsequent reciprocal operation – boundary points from the second grid searching the first grid – can now be eliminated thus saving time.

If the coordinates of the two grids reside in different processors, then both reciprocal operations would have to proceed without the knowledge of the non-overlapping status until one of them is completed and the status revealed. In order to prevent further fruitless search in the slower operation, this status must be conveyed immediately to the remote processor to signify cease operation. The remote processor is being given this datum without any prior request or even knowledge that a new datum is needed. Because f this, efficient two-sided communication in this situation would be next to impossible.

**Index Range**

Another local quantity that similarly needs immediate transfer, but in this case to (not from) the remote processor, is the $ijk$ index range of a structured donor grid within which to perform the search. Without going into detail, it is also used for acceleration purpose. Like the overlap status, this quantity is being pushed to the remote processor as soon as it is determined. Likewise, two-sided communication is too inefficient and difficult if not impossible to implement.

All the data communication in the grids connectivity algorithm has no race condition problem [7]. This is certainly not the case for boundary conditions in the solver which will be described in the next section.

**3. Communication in Overset Grids Solver**

The need to interpolate boundary values of overset grids has a profound effect on the programming landscape and communication efficiency of the code. This section describes the dramatically different characteristics of the two communication models applied to boundary value interpolation.

**Two-sided Communication**

In a two-sided communication, the only way to remotely access the list of all the interpolated boundary values needed by a component grid is to pass a very long message in a bulk communication from each of the processors that possess the data pointed to by the list. The numbering order in the list is pre-determined in the grid connectivity algorithm. Each message consists of a non-contiguous subset of the list (like that shown in Fig. 6), i.e., it contains boundary values interpolated from only those donor cells that reside in the remote processor that sends the message. This subset changes every time step for moving body problems. Together, all the messages add up to the full list. It is important to note here that the values are interpolated remotely.

Contrast this data access pattern with that used for global memory space in which many short messages are remotely accessed following the exact contiguous order of the list. Each message is a set of donor cell stencil values similar to that described in the ‘Donor Cell Search’ subsection above except that it may come from any remote processor.

Because of the severe constraint of one and only one long message, each remote processor must first pick and gather only the donor cells that it possesses from the full list of boundary points determined by the connectivity algorithm, interpolate each of them and then store them contiguously in a new array $Q_{send}(i,nproc)$, where $nproc$ indicates the ID of the processor that would receive the data, ready to be sent. The information in the list required for this gather operation is stored in $iBndPtDonor(:,nBndpt,nGrid)$, where $nBndpt$ counts the boundary points in grid $nGrid$. Once all the interpolated data are ready, a synchronization barrier in the code is executed to ensure that all processors have finished the interpolations and are ready to receive new boundary values before communications are initiated and old boundary values are overwritten. Data received are first stored in a separate array $Q_{recv}(i,nproc)$, where $nproc$ is the ID of the processor that sends the message, and then scattered, or distributed, to their respective boundary points.

CUG 2009 Proceedings 5 of 7
For processor 83 ← \[ \begin{array}{c} x \end{array} \] 1
42 ← \[ \begin{array}{c} x \end{array} \] 2
42 ← \[ \begin{array}{c} x \end{array} \] 3
42 ← \[ \begin{array}{c} x \end{array} \] 4
165 \[ \begin{array}{c} . \end{array} \] .
. \[ \begin{array}{c} . \end{array} \] .
. \[ \begin{array}{c} . \end{array} \] .
. \[ \begin{array}{c} . \end{array} \] .
42 \[ \begin{array}{c} . \end{array} \] .
. \[ \begin{array}{c} . \end{array} \] .
97 ← \[ \begin{array}{c} x \end{array} \] N

Fig. 6 Data in the array \( \text{iBndPtDonor}() \) are distributed non-contiguously to remote processors.

Since no more local solution data are needed by any other processor as soon as its gather operation to serve remote processors is complete, the local solver can begin the next time step and compute the new solution immediately after its scatter operation has updated its boundary values without having to execute another synchronization barrier. This is a significant advantage compared with one-sided communication that follows the exact order of the list, and a direct result of the fact that those non-contiguous interpolations needed locally are calculated continguously during the gather operation in the remote processor of the donor cells. With no more synchronization, the combined load balancing of the solver and the interpolation / communication steps becomes possible.

The same advantage offered by non-contiguous boundary interpolation described above is also a downside. Since all the remotely interpolated data needed by the local boundary points are scattered around in multiple non-contiguous blocks in an array (Fig. 6), the data in all the \( \text{Qrecv}(::, \text{nproc}) \) received are not stored in the same order as the original \( \text{iBndPtDonor}(::, \text{nbdpt}, \text{ngrid}) \). This requires additional bookkeeping, receiver point-counting, donor cell-counting, new arrays (with varying memory size for problems with moving grids) and most importantly, complex coding logic. Thus the implementation of this gather/scatter operation and optimization in the code becomes very tedious. The resulting code is also much less readable and understandable. In short, the different ordering of \( \text{Qrecv}() \) (gather/scatter operations) and \( \text{iBndPtDonor}() \) arrays results in extensive modification of the original serial (or global address space) code simply because only one bulk communication is allowed in a two-sided communication.

**One-sided Communication**

In contrast to the complex coding problem described above, one-sided communication as implemented in CAF is relatively simple. Besides the co-array declarations and load balancing subroutine, parallelizing the boundary interpolation part of the solver requires very minimal modification to only one statement, i.e., attaching a co-array to the solution array in the serial code where the donor cell data is accessed. In other words, \( \text{Q}((\text{nDonGr})\%\text{node}(i,j,k)) \) becomes \( \text{Q}((\text{nDonGr})[\text{nproc}((\text{nDonGr})\%\text{node}(i,j,k))] \).

There is, however, a significant trade-off for this programming simplicity. It results in a lot more data to be communicated between processors and, indirectly, a necessary second synchronization barrier before the next time step that would increase processor idle time for any load imbalance during the interpolation / communication step. The second synchronization barrier is needed because the entire dataset in the interpolation stencil of the donor cell (which may be large for high order interpolation, e.g., \( 4^3 = 64 \) points) is passed to the requesting processor where the interpolant is then locally calculated. The interpolant is not calculated remotely because the remote processor would then be required to gather only the donor cells that it possesses which are scattered non-contiguously in the original boundary point list \( \text{iBndPtDonor}(::, \text{nbdpt}, \text{ngrid}) \), just like the tedious procedure in the two-sided communication scenario.

An unnecessarily large message consisting of the entire donor cell stencil for just one local boundary point is a problem that seems to affect only overset grids because of the need to interpolate from a remote stencil. Non-overset grids or any domain decomposition methods that do not need boundary interpolation should be free of this disadvantage.

Because the interpolations are calculated locally using remote data, the remote solver cannot begin the next time step even after it has long completed its own boundary value update, until all local interpolants have been computed. Otherwise the remote donor cell solution data for those unfinished local interpolations in the current time step might be overwritten by the solution of the next time step. This necessitates a second synchronization barrier that prevents combined load balancing of the solver and the interpolation / communication steps. The load imbalance may be tolerable if there are only a small number of processors in close proximity and linear interpolation is used. It can, however, become very expensive if this is not true. This is especially so for the very costly cubic interpolation with large stencil. Fortunately for the Cray X1, its built-
in hardware support for global address space reduces, but
not eliminates, the impact of this penalty.

One can, of course, choose to adopt the same
tedious procedure that the two-sided model requires but
use one-sided model to facilitate the bulk communication
[8]. This would, however, significantly nullify the
inherent advantage and defeat the purpose of the one-
sided model in overset grids. It is, nonetheless, an easy
way out for implementing the one-sided model in existing
codes that use two-sided model for boundary
interpolation.

Interim Summary

A brief summary of the pros and cons of one-
and two-sided communication models in their
applications to parallel numerical PDE using overset grids
as detailed above is appropriate. The two-sided model is
inefficient from the programmer's point of view, but it
results in a much smaller message size and needs only one
synchronization which, in turn, enables the combined load
balancing of computation and interpolation /
communication. The one-sided model, in contrast, has
exactly the opposite of the above three characteristics.

It is also appropriate to say that the non-
separation of the interpolation and communication steps
in overset grids raises significant coding and
communication issues in the parallelization effort that do
not arise in conventional (structured or unstructured) non-
overset grids.

4. Conclusion

The overset grids method is a powerful
technology for simulating moving body CFD problems.
It naturally solves some of the difficulties inherent in non-
overset methods. The burden, however, is shifted to rest
on the need for grid connectivity that requires a logically
complex algorithm and for boundary interpolation. Both
of these are absent in non-overset method.

Accordingly, overset grids require data
communication at two instances within a time step: A) grid
connectivity at the beginning and B) grid boundary
values at the end. This paper examined the characteristics
of one- and two-sided communication models applied to
both instances. The application of the one-sided model to
A) clearly results in programming efficiency and new
algorithmic capability that are difficult to match with the
two-sided model. Unfortunately, a similar application of
the one-sided model to B) is not as distinctively
beneficial. Specifically, its merit in programming
efficiency is being offset by the larger amount of data
transfer and higher number of synchronization points
required.

References

1. Lee, Y. and Baeder, J. D., “Vortex Tracking using
Overset Grids for Quad Tilt Rotor in Forward
Aerodynamics Conference, Providence RI, Aug. 16-
2. J. L. Steger, F. C. Dougherty and J. A. Benek, “A
Chimera Grid Scheme,” Advances in Grid
Generation, K. N. Ghia and U. Ghia (eds.), ASME
A New Approach to Overset Grid Connectivity,”
AIAA-2003-4128, AIAA 16th Computational Fluid
Dynamics Conference, Orlando FL, Jun. 23-27,
2003.
Application to Flapping Wing Micro Air Vehicles,”
25th Army Science Conference, Minneapolis MN,
Sep. 2006.
5. R. W. Numrich and J. K. Reid, “Co-Array Fortran
for Parallel Programming,” ACM Fortran Forum,
6. T. Wen, J. Su, P. Collela, K. Yelick and N. Keen,
“An Adaptive Mesh Refinement Benchmark for
Modern Parallel Programming Languages,” SC07
Parallel Programming,” Sun Microsystems
8. R. Biswas, et al. “NAS Experience with the Cray