Domain coupling with the DOVE scheme

Andrei V. Smirnov, Hanzhou Zhang

West Virginia University
Morgantown, WV 26506, U.S.A.

Abstract

Key words: CFD, domain decomposition, overlapping domains, unstructured meshes

1 Background

Numerical solutions of partial differential equations (PDE), are typically resource- and time-consuming. The reason is that the discretization of PDEs usually results in large linear systems of equations. To improve accuracy and increase the speed of execution it is desirable to partition one large-scale problem into many smaller subproblems, and solve it in a parallel manner. The emergence of inexpensive Beowulf clusters and message passing standards makes parallel computing even more attractive.

Domain decomposition is one of the most significant techniques to solve PDEs on distributed memory multi-processor systems. It is based on partitioning the computational domain into several sub-domains. The original problem can be then reformulated on sub-domains as a family of subproblems of reduced size [1]. In PDEs leading to elliptic problems the application of the technique can be especially challenging since sub-dividing the domain leads to much slower global convergence rates [2]. The solution of the original PDE is typically obtained by a preconditioned Krylov space method, such as conjugate gradient method, generalized minimal residual algorithms. The pre-conditioner is produced by using the solutions to subproblems.

Email addresses: asmirnov@wvu.edu (Andrei V. Smirnov,), zhang_hanzhou@hotmail.com (Hanzhou Zhang).
URL: mulphys.com (Andrei V. Smirnov,).

Sub-domains may be overlapping or non-overlapping. The domain decomposition methods for overlapping sub-domains are known as Schwarz methods. The earliest domain decomposition algorithm, Schwarz alternating method, was introduced by H. A. Schwarz in 1869 [3]. Its sequential nature makes it not ideal for parallel computations [4]. The additive Schwarz method proposed by Dryja and Widlund [5], makes parallel computing of subproblems possible (see also [6]). Another type of Schwarz method, the multiplicative Schwarz method, is directly generalized from the original Schwarz alternating method [7].

Iterative sub-structuring methods [8–10], or Schur complement methods, are methods for nonoverlapping domain decomposition where sub-domains are separated from one another by interfaces. The main technique of these methods is to reduce the differential problem to an interface problem, which is defined in terms of the Steklov-Poincaré operator. In certain cases, sub-structuring methods and Schwarz methods can be unified.

Domain decomposition typically involves a decomposition of the geometric data structures, or decomposition of the mesh. The mesh decomposition is usually subjected to some optimality criteria such as load balancing. It can can be viewed as a graph partitioning problem. Unfortunately, this graph partitioning is a NP-complete problem []. And there are almost no heuristics that can solve this problem efficiently.

When domain decomposition is applied to an unstructured mesh the realization of conformal mapping will often lead to highly irregular boundaries for each sub-domain, which in turn will create extra errors in the gradient approximations. One possibility to avoid this is to construct all parallel domains independently and then combine them together using inter-domain interpolation. This approach is conceptually simple and can be used to construct domains of fairly complex topologies and large sizes. It naturally provides for smooth boundaries of each sub-domain. The price for this convenience is the extra computational effort required to do variables interpolation at the communication boundaries. Considering that this approach is similar to an assembly procedure, in contrast to a dissection approach of a conventional domain decomposition, we call it *domain coupling* rather than decomposition [11].

In this work a complete solution to a general domain coupling problem was developed and used to solve a complex flow problem in a multi-processor environment by means of message-passing interface. Section 2 gives the outline of the method, with some of the results presented in Sec.3 and details on implementation provided in the Appendix. The perspectives of this technique for complex problems of continuum mechanics involving moving boundaries are outlined in Sec.4.

2 Method

We consider a solution of a continuum dynamics problem on M domains distributed over N computing nodes or *processors*. The *computational model* of a physical problem is introduced by the following definitions.

(1) **Element** is a geometrical primitive. Elements can be of four different ranks, which identify their dimensions: 0 (node), 1 (edge), 2 (face), 3 (cell). Nodes are points in space defined by their coordinates. All elements of higher rank are defined by connecting them to lower-rank elements, which are then called *sub-elements*. In some instances elements may also be connected to the elements of the same rank which are called *neighbors* or to the elements of higher rank (*super-elements*). Thus, an edge is a line segment connecting two points, face is a polygon defined as a connected set of nodes or edges, and cell is a connected set of faces, edges or nodes forming a polyhedron. A connectivity can be *static* or *dynamic*, i.e with a constant or a variable number of connecting elements. Elements with dynamic connectivities can represent continuum as well as discrete medium.

Elements of rank 2 and higher are characterized by their *order*, which is equal to the number of the sub-elements of lower rank they are connected to. This quantifies them as different order polygons or polyhedrons, such as triangles, cubes, tetrahedrons, femtohedrons, hexahedrons etc. In addition to this each element is characterized by its *type*, which determines its relation to other elements and variables. For example, a node can be internal, boundary, near-boundary etc.

There can be several types of *boundary elements*, depending on their proximity to the boundary. The proximity determines if the element is connected to the boundary directly (boundary-type) or indirectly, by neighboring other boundary-elements (near-boundary type). There can be several degrees of boundary proximity depending on the rank, order and the type of boundary connectivity. Thus nodes located at the boundary are of boundary type, and those not located at the boundary but having a boundary super-element are of a near-boundary type. Cells having only a face, an edge or a node, belonging to the boundary, will all have different degrees of boundary proximity.

(2) **Mesh** is a single-connected set of elements, where every element can be reached from any other element of the same rank through the neighbor connectivity information.

(3) **Boundary** is a set of all boundary elements. The *order* of the boundary is determined by the degree of boundary proximity of its elements. The n-th order boundary of mesh M is denoted as $\partial^n M$.

(4) **Overlap** of mesh A by mesh B is a set of boundary elements, ∂B, of a mesh B and cells of mesh A, such that the elements ∂B themselves or

some of their sub-elements are inside the cells of mesh A. These cells of A are called the (*host-cells*). In mathematical notation the overlap is defined as $O(A,B) \equiv A \bigcap \partial B$. According to this definition the overlap is not commutative, i.e. $O(A,B) \neq O(B,A)$. The first argument of the overlap $O(A,B)$, i.e. A, is the object considered to be *overlapped* and the second, B, is *overlapping*. In what follows we shall also refer to $O(A,B)$ as *domain overlap*.

(5) **Variable** is defined by a set or real numbers representing a property, such as pressure, temperature, velocity, stress, etc. A variable is defined on any of the elements, such as nodes, edges, faces or cells, and it inherits the type of element on which it is defined. Depending on the type of the element and the type of it's connectivity to other elements (static, dynamic) the variables can represent a continuum or a discrete property (density, particle size, etc.) A variable is also characterized by its *rank*, which determines its dimensionality: 0 - scalar, 1 - vector, 2 - matrix, $\ldots n$ - tensor of rank n.

(6) **Boundary variable** is a variable defined on boundary elements.

(7) **Iteration** is a procedure that updates some or all variables on the mesh according to a specified algorithm. Iterations are characterized by *order*, which is derived from the time accuracy of the discretizatin scheme.

(8) **Solver** is a sequence of iterations that compute the state of all variables at a specified physical time. A solver is supplied with the convergence and termination criteria.

(9) **Domain** is represented by a mesh, a set of variables and a solver. While mesh is a pure geometric construct, the variables and the solver represent the *model* of a specific physical problem. Each domain can only have one model, but each model can be represented by many domains. That is, domains relate to models in a one-to-one relationship, while models may have one-to-many relationship to domains. The set of all domains forms the *computational space*.

(10) **Processor** is a computational unit capable of executing sequential algorithms.

(11) **Processor map** is a distribution of domains among the processors. Domains relate to processors by one-to-one relationship whereas processors can have a one-to-many relationship to domains, which is to say that each domain can be computed by one processor only, but each processor can compute several domains. Domains computed on the same processor are called *near domains* and those computed on different processors - *far domains*.

The definitions above form the basis of a *multi-model*, *multi-domain* and *multi-processor* computational environment. In this formalism the total problem space can be represented by several domains supplied with the same or different physical models. The total set of domains is solved by a num-

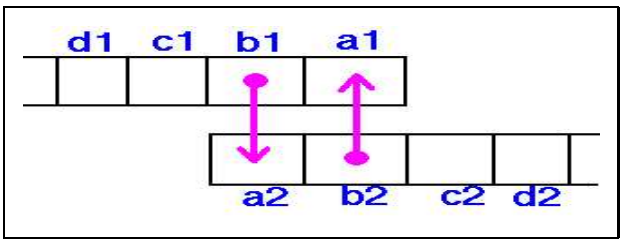


Fig. 1. Domain decomposition on co-located grids.

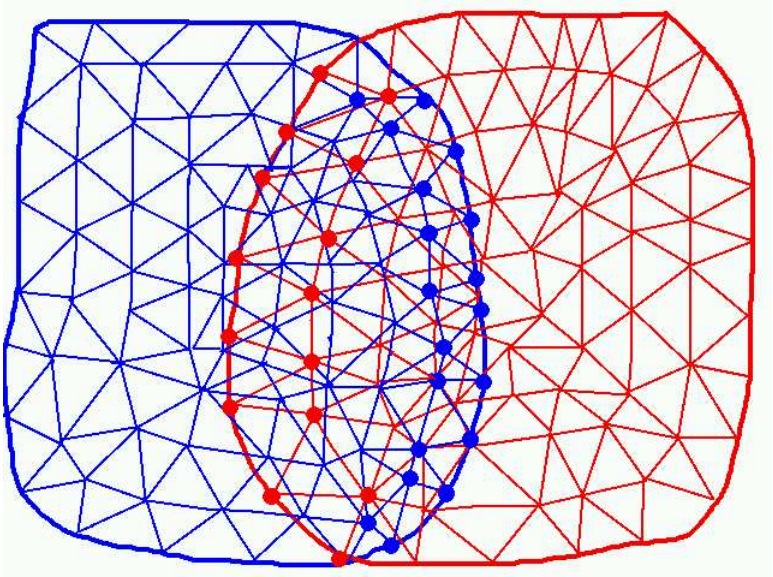


Fig. 2. Domain coupling on overlapping unstructured grids. Boundary nodes are marked with thick dots.

ber of processors, which does not have to be equal to the number of domains. This formalism was implemented in a multi-physics simulation system MulPhys [1] [12]. In what follows we shall describe the method of inter-domain data exchange in single and multi-processor environments. This method is based on the computational scheme for computing *domain coupling* through domain overlaps (DOVE) [13].

In contrast to the conventional domain decomposition method (Sec.1, Fig.1), where an initially constructed mesh for the computational space is split into domains, in the domain coupling method each domain is constructed independently and then *"sewed"* together with other domains in the areas of domain-overlaps. Figure 2 shows an example of the domain coupling on the boundaries of second order with four communication planes. As can be seen in the figure the positions of the nodes in both grids do not necessarily coincide in the region of overlap.

[1] mulphys.com

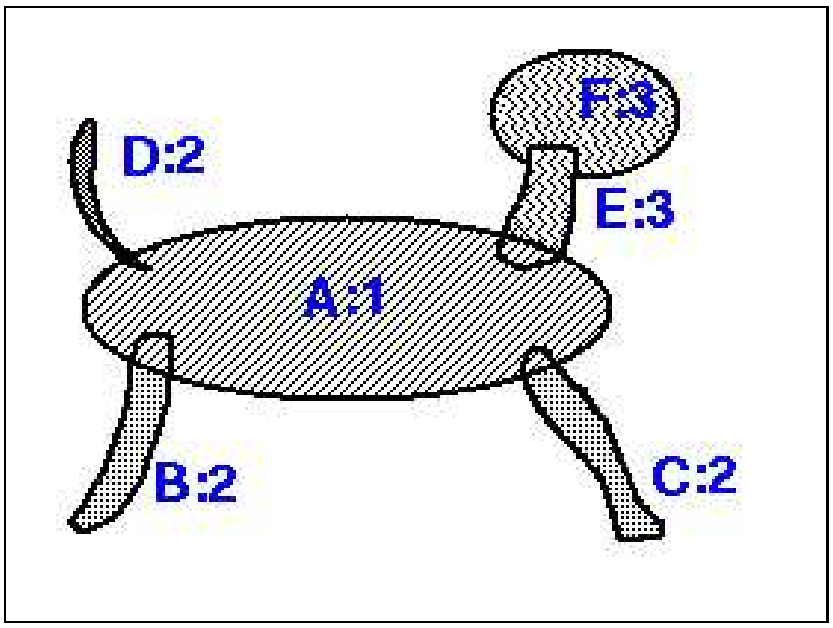


Fig. 3. Distribution of 6 domains over 3 processors: {1:A}, {2:B,C,D}, {3:E,F}.

During the iterations of the solver the variables in domain overlaps are exchanged in such a way as to provide a continuous solution across domain boundaries, following the alternating Schwarz method [**?**]. The advantage of this scheme is that it simplifies mesh generation since the conformity at the domain boundaries is no longer required. In addition to that, because the total mesh can be assembled by pieces, the method enables generation of meshes of large sizes, which can extend beyond the memory capacity of a single workstation.

The problem of domain coupling is reduced to constructing for each domain, D_i, overlaps with other domains: $O(D_i, D_j)_{i \neq j}$ (Sec.2, Def.4), and using this information to communicate the variables in the overlap areas between the domains during each iteration. This problem can be considered in both single and multi-processor modes. Generally, when M domains are distributed over N processors with $M > N$ (Fig.3) some of the domains will reside on the same processor (*near domains*: $\{E, F\}$, Fig.3). If such domains also happen to be overlapping, communicating the boundary overlap information will not require message-passing routines and the coupling can be done more efficiently than for the far-domains (Sec.2, Def.11)), like domains $\{A, B\}$, $\{A, C\}$, $\{A, D\}$, $\{A, E\}$ in Fig.3.

To solve this general problem of domain coupling a special computational procedure was developed. This procedure consists of initialization and communication steps. The initialization step enables automatic identification of overlapping regions and establishing their connectivities. The communication step uses this connectivity information and interpolation routines to exchange the variables in the areas of overlap. Two versions of this procedure

were implemented: for near and far domains (Sec.2, Def.11). While the first version can be used for a multi-block type mesh constructed on a single processor, the second version can be used in multi-processor computations on distributed memory platforms such as Beowulf clusters.

2.1 Initialization

The initialization procedure is used to determine for each domain A the overlap $O(A,B)$ with other domains as defined in Sec.2, Def.4. The overlap data consists of the coordinates of boundary nodes of domain B contained inside A and the pointers to the host-cells of domain A - one for each boundary node. According to this scheme the initialization procedure is split into two parts: identification of overlapping boundary points of the neighbor domain, and establishing of point-cell connectivities. The procedure of identification of boundary overlaps is the same for both near and far domains, and it is based on the solution of a classical inclusion problem in a polyhedron [14].

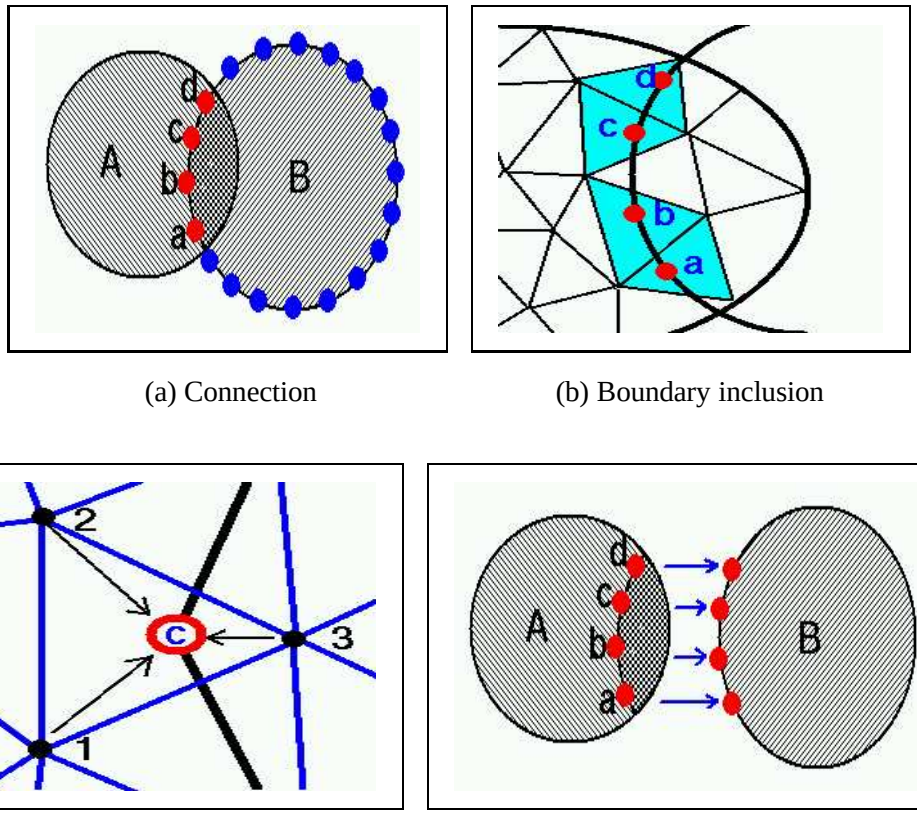
In order to establish the connectivities the array of boundary coordinates of each domain is sent to all other domains (Fig.4(a)), and the corresponding arrays are received from other domains (App.A.3). Each received boundary point is analyzed on whether it is inside of the current domain, and if so, the cell of the current domain is found, which hosts the boundary point (Fig.4(b), App.A.2). The coordinates of the boundary points of the overlapping domain together with their indexes as used on that domain, as well as the pointers to the host-cells of the overlapped domain are stored in a domain-overlap list (Dove, App.A.1) for subsequent inter-domain communication.

For this scheme to work each processor should hold information on the distribution of domains over the processors. This information is loaded as a *processor map* (see Sec.2, Def.2) at the beginning of execution.

In the case of far-domains the initialization procedure can be communication intensive resulting in quadratic dependence of the number of sent messages on the number of communicating nodes. To reduce this communication burst a special deferred communication scheme is implemented (App.A.3).

2.2 Communication

The communication procedure (App.A.1) is called from the PDE solver whenever the values of the overlapped boundary variables need to be updated



(a) Connection (b) Boundary inclusion

(c) Cell interpolation (d) Communication

Fig. 4. DOVE scheme

with the corresponding interpolated values from the overlapped domain (Fig.4(d)). Usually this is required once per solver iteration.

During the inter-domain communication the values of the dependent variables are interpolated to the boundary points of the overlapping domains (Fig.4(c)) received during the initialization step (Sec.2.1). If the solver has the order of accuracy higher than one the communication scheme may involve both boundary and near-boundary elements. The order of proximity of near boundary elements will be selected according to the order of the discretization scheme used in the solver.

The interpolation is done inside each host-cell corresponding to each boundary node (Fig.4(c)). The pointers to the host-cells are retrieved from the domain overlap information acquired during the initialization step (Sec.2.1). A linear interpolation scheme is used for tetrahedral elements and a higher order scheme would be appropriate for higher-order polygons. The interpolated values are sent to the overlapping domains together with the indexes of the boundary nodes (Fig.4(d)). Correspondingly, the variables of the neighbor domain interpolated to the overlapped boundary of the current

domain are received from the neighbor domains, and used as boundary conditions on the overlapped boundary.

3 Results

The Dove scheme was tested in MulPhys environment for two cases: (1) a finite-element Poisson solver, and (2) a control-volume flow solver. Both solvers use tetrahedral meshes. All the variables were defined on the noes, which corresponds to the vertex-centered discretization scheme.

3.1 Poisson solver

The Poisson solver implments a solution of the bounary value problem for a scalar variable $P(\mathbf{x})$:

$$\begin{aligned} \Delta P(\mathbf{x}) &= f(\mathbf{x}) \\ P(\mathbf{x}) \mid_{\mathbf{x}\in\partial_0 D} &= f^0(\mathbf{x}) \qquad (1) \\ \frac{\partial P(\mathbf{x})}{\partial \mathbf{n}} \mid_{\mathbf{x}\in\partial_1 D} \equiv (\nabla P \cdot \mathbf{n})_{\mathbf{x}\in\partial_1 D} &= f^1(\mathbf{x}) \qquad (2) \end{aligned}$$

where $(* \cdot *)$ denotes a scalar product operation on two vectors, $\mathbf{x}$ represents a vector of coordinates $\mathbf{x} = \{x_i\}$, $i = 1:3$, $\mathbf{n}$ is the normal vector to the boundary $\partial_i D$ of the domain D with $i = 0$ representing Dirichlet and $i = 1$ - Neuman boundary. Functions f, f^i are the source-function inside the domain, f, and the boundary sources, f^i (i=0,1), respectively. The problem is discretized by a finite-element method (FEM) on a tetrahedral mesh.

First we considered two cases of heat transfer in a cubic box, which corresponds to the solution of a simple Laplace equation. The box was split into several domains, which were executed on different processors with the DOVE coupling scheme responsible for inter-domain communications. In the first case two opposite walls of the box were set at a constant temperatures with low and high values, and the other walls were made adiabatic. This situation should produce a temperature profile changing linearly inside the box from the cold to the hot wall. In the second case the side walls were set to the low temperature as well, which should produce a non-linear temperature decay from the hot to the cold wall.

Figure 5 presents the results of the first case for a single-, two-, and four-processor runs. As can be seen all curves agree well to the linear profile.

In the second case, (Fig.6) the continuity of the solution across the domain boundaries is observed as well.

In another test case we performed computations of a Poisson equation with a non-zero source term and constant value boundary conditions. The source was put inside the right of the cube. The cube was split into tree domains, which were coupled with the DOVE scheme and executed on different processors. Figure 7 shows the contour plots and cross-sectional temperature profiles for this case.

As can be seen in the figures there is an almost perfect coincidence in computed values for a single and multi-processor cases.

3.2 Flow solver

The flow solver used in this study [15] is based on the coupled solution of equations of mass and momentum conservation, augmented with the thermodynamic relation of state [2]

$$\dot{\rho} + (\rho u_i)_{,i} = 0 \tag{3}$$

$$\dot{u_i} + u_j u_{i,j} = \nu u_{i,jj} - \rho^{-1} p_{,i} \tag{4}$$

$$p = \rho R T \tag{5}$$

where ρ, u_i, p, ν are density, velocity, pressure and kinematic viscosity. For simplicity we consider the ideal-gas law as an equation of state with R as the gas constant, and T as the absolute temperature.

A stable discretization of the equation system (3)-(5) in a control-volume formulation is obtained by using the mass of the control volume, m as an independent variable: $m = \rho cv$, where cv is the control volume [15]. The usage of mass as an independent variable complicates the domain-coupling scheme since the control volumes of the sending and the recipient domains can be different. In this case the communicated variable should be the density: $\rho = m/cv$. Otherwise the mass on the receiving end should be derived from the pressure and the control volume using the equation of state (5). The last approach was adopted in our procedure.

Figure 8 shows the velocity vector field in a simple composite domain consisting of two straight cylindrical pipes. A fluid flow was simulated in this domain by specifying a pressure drop between the inlet and the outlet of

[2] We use notation: $\dot{a} \equiv da/dt$, $a_{,i} \equiv \partial a/\partial x_i$

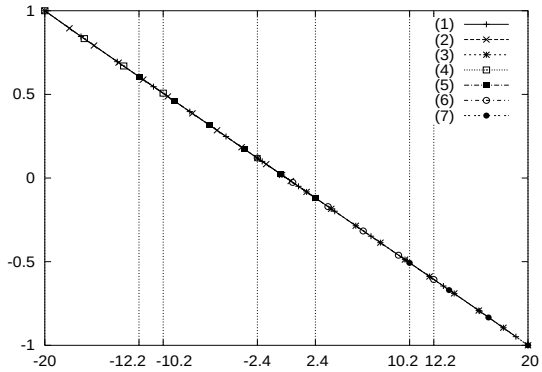
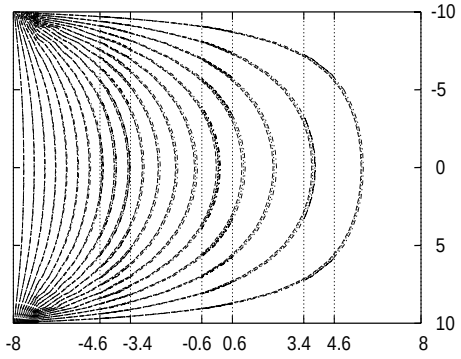
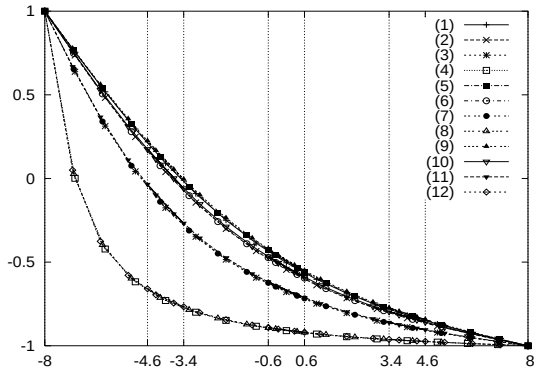


Fig. 5. Temperature decay in a cubic domain with adiabatic walls. (1):single processor run; (2)-(3):two-processor run; (4)-(7):four-processor run; Vertical lines: boundary places.

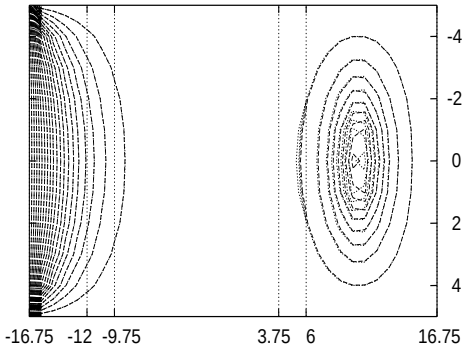


(a) Contours of constant temperature

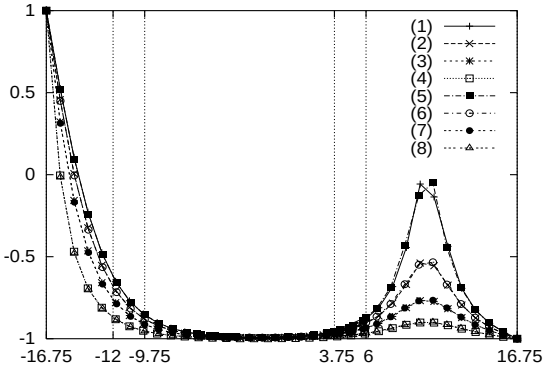


(b) Temperature profile at different cross-sections

Fig. 6. Temperature decay in a cubic domain with constant temperature walls. (1)-(4):single processor run; (5)-(8):two-processor run; (9)-(12):four-processor run; Vertical lines: boundary places.



(a) Contours of constant temperature



(b) Temperature profile at different cross-sections

Fig. 7. Temperature distribution with a source and constant-temperature walls. (1)-(4):single processor run; (5)-(8):three-processor run; Vertical lines: boundary places.

the straight pipe. In the present coupling scheme the degree of overlapping of the domains can be set arbitrarily, although from the consideration of memory utilization a smallest possible overlap consistent with the discretization scheme is preferred. A three-cell overlap was used in this case. The figure is centered on the region of overlap with two boundaries of the overlapping domains visible on both sides from the center. As can be seen the vector field shows a continuity across the domains.

The advantage of this scheme is especially evident in complex geometries with arbitrary connectivities between the domains. The next case illustrates this point. An example in Fig. 9 shows a more complex composite domain, consisting of two overlapping domains: a straight and a curved pipe. Domain coupling with the DOVE scheme enabled to simulate the continuous flow passage through the composite domain with deflection of some of the flow from the straight pipe to the curved one (Fig. 10). Since any overlap-

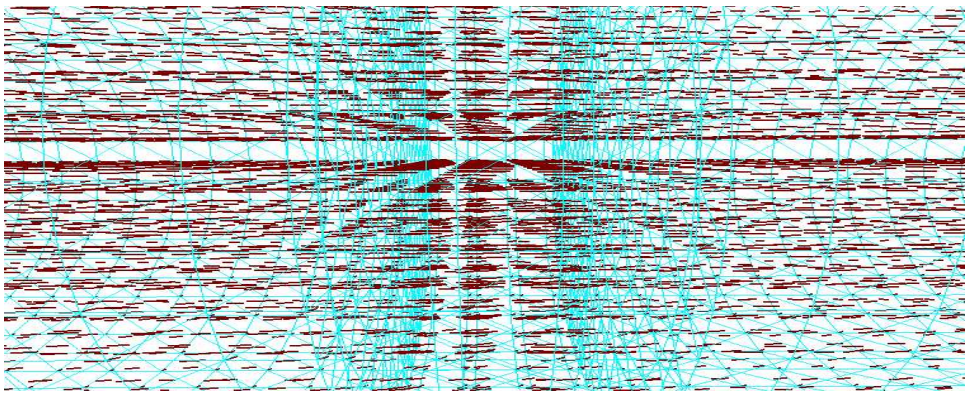


Fig. 8. Flow through two coupled straight pipes.

ping of two domains is detected automatically, topologically complex cases can be handled as easily as simple ones.

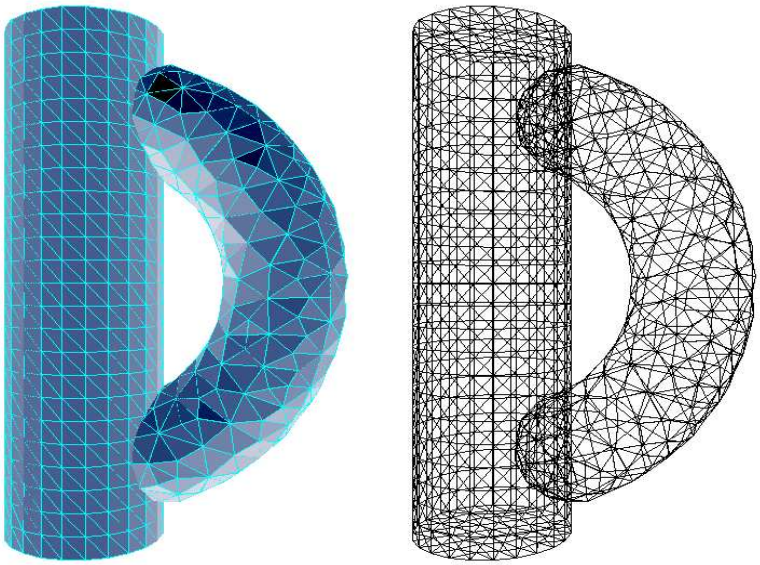
The cases with more than two domains are handled in an analogous manner, as long as there are no more than two domains overlapping in the same region of space. Figure 11 show an example of a more complex domain where an extra square box was added on the top of the pipe with the flow outlets at two opposite sides of the box. Coupling of multiple domains may create problems if more than two domains overlap in the same region of space. This situation can still be handled by designing a ”*shadowing*” strategy, whereby a domain that was *overshadowed* by another domain is rendered inactive in the region of overlap. Such a strategy is being currently worked out.

4 Conclusions and Future Work

An inter-domain coupling scheme is proposed which enables automatic detection of domain overlaps and and setup of inter-domain communication for overlapping 3D domains of arbitrary geometry. The scheme was tested on the cases of a finite-element Poisson solver and a control-volume flow solver.

One of the advantages of the proposed scheme is that it can be used to construct large and complex domains without the need to store all the grid on a single computing node at the same time. Thus a fairly large computational domains, which may not fit into a memory of a single processor can still be constructed by patching together smaller sub-domains.

As was noted in Sec.3, overlapping of more than two domains in the same region of space should be handled in a special way to avoid ambiguity of in the solution. This can be done by introducing the overlap-order sequence,



(a) Boundary (b) Mesh

Fig. 9. Complex domain coupling.

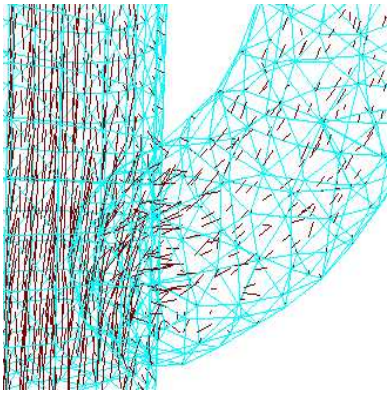


Fig. 10. Flow through the coupled domains.
A simulation is available at www.mulphys.com/dove

so that each domain will have only one overlapping and one overlapped domain.

The DOVE scheme can be easily extended to handle dynamic overlapping regions, when the domains may change shape or move with respect to each other. In this case the Dove initialization functions should be called from within the main solver at certain times to update the changing geometry of the overlap region and re-generate the connectivity arrays. In this case the initialization step (Sec.2.1) will become a part of the solver iterator. Thus, the realization of dynamic overlaps will be rather straightforward. It will enable modeling of many dynamic events that are usually considered tough for

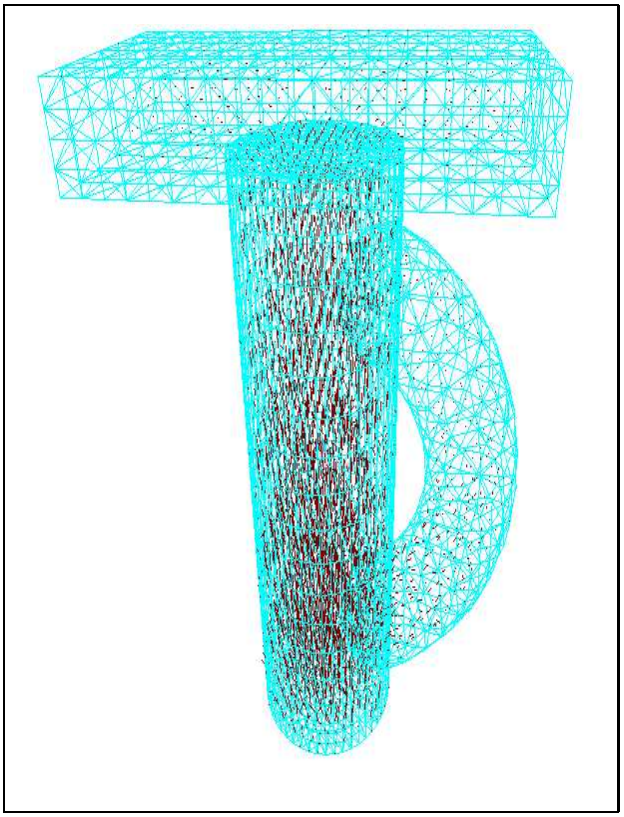


Fig. 11. Flow through three coupled domains.

conventional CFD: mitosis (cell division), munitions released from bomb-bays, etc.

Because of the non-conformal nature of the decomposition scheme and the consequent need for interpolation the order of accuracy of the solution in the boundary plane may degrade. This can be somewhat offset by using higher order boundaries represented by near-boundary elements of various orders of proximity. This will still not be as exact as a solution based on conformal mapping, since it will not eliminate the diffusive effects of interpolation. Nevertheless the flexibility of the scheme to handle complex geometries, including moving domain boundaries and whole domains, along with its practicality in storing large data sets can be an attractive feature for some tough engineering problems.

A Algorithms

The algorithms of setting and communicating domain overlaps below are given for the case of tetrahedral meshes and node-type variables of rank 0.

The boundary overlap data are stored in the list of domain overlaps, which is created during the initialization of domain overlaps (Sec.2.1). A member of this list is represented by the Dove structure below:

```
struct Dove // Domain overlap
{  int
      Nnodes, // number of overlapping nodes
              // of the neighbor domain
      I[];    // array of boundary node-indexes
              // of the neighbor domain
   float  X[]; // array of node-coordinates
               // of the neighbor domain
   Cell   *cell[]; // list of pointers to the cells
               // of this domain containing points
               // with the coordinates X
   Domain   *neighbor; // neighbor domain
   Dove   *next; // next overlap structure
};
```

As can be seen, it is connected in the linked list through `next` pointers. The *Node* and *Cell* structures can consist of this minimum set of data:

```
struct   Node
{
   int   type;
   float
      x[DIM],//Coordinates
      *var;  //Variable storage
};
struct   Cell
{
   unsigned int   index; //used in file IO
   Node   *vert[Nverts];
   Cell   *neib[Nfaces];  //neighbors
};
```

Here and in all the listings below the `DIM` constant is equal to 3. We also consider the case of node-based variables, corresponding to the vertex-centered discretization scheme.

The listing below shows the data exchange in the domain overlap region during each iteration of the solver for the far domains:

```
// Sending domain overlap:
for ( int i=0; i<ndomains; i++ )
{  if ( proc==domain[i].proc )
      domain[i].sendDove();
}
//Receiving domain overlap:
for ( int i=0; i<ndomains; i++ )
{  if ( proc==domain[i].proc )
   for ( int ivar=0; ivar<maxvar; ivar++ )
      domain[i].receiveDove();
}
```

The simplified versions of send/receive functions are implemented like this:

```
void Domain::sendDove()
{  for // Send overlap variables
   (  Dove *dove=dove_head;
      dove != NULL;
      dove = dove->next
   )
   {  // Variable buffer:
      float  *varbuf = new float[dove->Nnodes*Nvar];
      for( int inode=0; inode < dove->Nnodes; inode++ )
      {  getVar // Interpolate variables inside the cell
         (  dove->cell[inode], // host-cell
            dove->X+DIM*inode, // interpolation point
            Nvar,              // number of variables
            varbuf+Nvar*inode  // returned variables
         );
      }
      sendInt // Send variables
      (  dove->neighbor, // recipient domain
         Nnodes,  // number of nodes in the overlap
         dove->I  // indexes of boundary nodes
      );
      sendFloat // Send variables
      (  dove->neighbor, // recipient domain
         Nnodes*Nvar,    // number of variables to send
         varbuf          // send buffer
      );
   }
}
```

In this case variables `iproc`, `dove_head`, and `Nvar` are external to this function and are public variables of the domain.

Function `getVar(Cell *C, float X[], int n, float V[])` interpolates n variables from cell vertexes of cell `C` to the cell-internal point with coordinates `X[0]`, `X[1]`, `X[2]`, and stores the result at `V[0..Nvar-1]`. The variables are considered to have `rank=0` (Sec.2, Def.5).

Functions `sendInt, sendFloat` send arrays of integer and floating-point numbers to the overlapping domain. The host-processor of the recipient domain is determined from the processor map (Sec.2, Def.procmap). The send- functions implement the inter-processor data exchange, using a standard message passing interface, such as MPI.

Receiving of the domain overlap data can be by the following procedure:

```
void Domain::receiveDove()
{  //Receive overlap variables from connected
   // domains on other processors
   for
   (  Dove *dove=dove_head;
      dove != NULL;
      dove = dove->next
   )
   {  int Nnodes, Nvar,
          *I; // array of indexes to the boundary nodes
      float  *varbuf; // receive-buffer
      receiveInt( &Nnodes, &I );
      receiveFloat( &Nvar, &varbuf );
      for ( int inode=0; inode<Nnodes; inode++ )
      {  for( int ivar=0; ivar<Nvar; ivar++ )
         nodes[I[inode]].var[ivar]
         = varbuf[inode*Nvar+ivar];
      }
   }
}
```

Where `nodes[]` is an array of mesh-nodes of the current domain.

A.2 Cell search

The algorithm below solves a problem of locating a cell containing a point on a general plyhedral mesh. The algorithm uses the function `hostcell` that solves the inclusion problem on a polyhedron: given initial cell and a point it points to the initial cell or it's immediate neighbor-cell, which either contains the point or lies closer to the point than the initial cell. The function returns NULL if the point lies outside of the domain boundary.

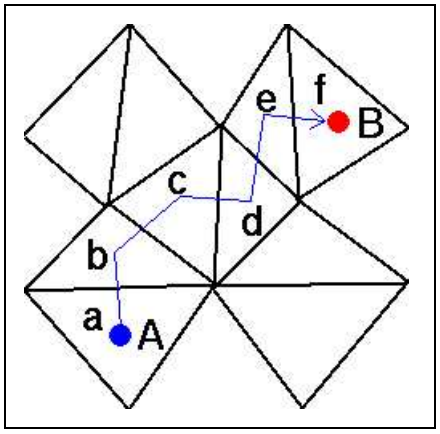


Fig. A.1. Host cell search. The search of the host-cell for point B goes through the intermediate cells $\{a, b, c, d, e, f\}$ in a linear manner.

```
Cell  *findcell
(  float *x,    // x[DIM] coordinates of a point
   Cell  *cell  // initial cell
)
{  //Locates cell containing point x
   int  counter=0;
   Cell  *next,*prev;
   next=prev=cell;
   while ((next=hostcell(x,cell))!=NULL)
   {  if (next==cell) return cell;
      if (next==prev) return NULL;
      prev=cell;
      cell=next;
   }
   return NULL;
}
```

The function `findcell` has two arguments: the coordinates of the point and the pointer to the initial cell, where the search will start. Usually, there exists a good guess of the initial cell for the search to start, like the previous host-cell of the particle in particle tracking problems, or the host-cell of the neighbor boundary node, in mesh-overlap problem. The algorithm will go in a linear path from the original cell to the new cell as illustrated in Fig.A.1. This makes the execution time of the algorithm linear in the number of cells of the mesh. The time of an exhaustive looping through all the mesh cells would be a cubic function of the mesh size.

Below is the algorithm for inter-domain boundary data exchange, which forms the outer shell of the procedure of setting up the domain overlap information. This procedure is used at the initialization stage of the Dove scheme (Sec.2.1). In this scheme instead of sending data to all processors at once, each processor sends data to a processor located n processors ahead in processor number range. Then it receives data sent by a processor located n processors behind. After that the whole process is repeated with a different n, untill n spans all the processor number range.

```
for ( int n=1; n<nproc; n++ )
{  int to = ( proc+n ) % nproc;
   for ( int idom=0; idom < ndom; idom++ )
   {  if ( proc == domain[idom].proc )
      for ( int j=1; j<ndom; j++ )
      {  int jdom = ( idom+j ) % ndom;
         if ( to == domain[jdom].proc )
            domain[idom].sendBoundary ( jdom );
      }
   }
   int from = ( proc+nproc-n ) % nproc;
   for ( int idom=0; idom<ndom; idom++ )
   {  if ( proc == domain[idom].proc )
      for ( int j=1; j<ndom; j++ )
      {  int jdom = ( idom+j ) % ndomains;
         if ( from == domain[jdom].proc )
            domain[idom].receiveBoundary ( jdom );
      }
   }
}
```

In this listing `iproc` is the current processor number, variables `to` and `from` identify the number of processors to send/receive data, `n%m` is a division-by-modulus operation of number `n` by number `m`, `idom` is the number of the domain residing on the current processor, and `jdom` is the number of the domain to communicate the boundary to/from. Functions `sendBoundary` and `receiveBoundary` implement the communication of boundary nodes and identification of domain overlaps as described in Sec.2.1, and App.A.2.

Figure A.2 illustrates this communication scheme for the case of `n=2`. The division by modulus used in the algorithm makes the processors arranged in a connected loop, which facilitates the processor skip operation.

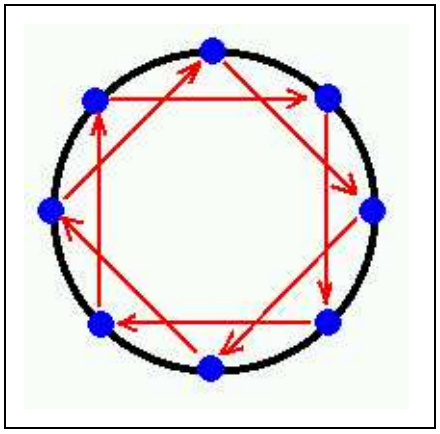


Fig. A.2. Interprocessor boundary exchange: Each processor skips over *n* processors when communicating data, where *n* changes from 1 to the maximum processor number (*n*=2 above).

This algorithm guarantees that the communication overhead is linearly dependent on the total number of communicating processors, as opposed to the quadratic dependence when the loop over n is not implemented.

The boundary information received by each processor is used to initialize the boundary overlap list (App.A.1).

References

[1] Alfi o Quanteroni and Alberto Valli. *Domain Decomposition Method for Partial Differential Equations*. Oxford Science Publications, 1999.

[2] Dihn, Q.V., Glowinski, R. and Periaux, J. Solving elliptic problems by domain decomposition methods with applications. In G. Birkhoff and A. Schoenstadt, editor, *Elliptic Problem Solvers II*. Academic Press, New York, 1984.

[3] H.A. Schwarz. Űber einige abbildungsdufgaben. *J. Reine Angew. Math.*, 70:105–120, 1869.

[4] X. Zhang. *Studies in Domain Decomposition: Multilevel Methods and the Biharmonic Dirichlet Problem*. PhD thesis, Courant Institute, New York University, 1991.

[5] O.B. Widlund M. Dryja. An additive variant of the schwarz alternating method for the case of many subregions. Technical Report 339, Department of Computer Science, Courant Institute, New York University, 1987.

[6] A.M. Mataokin, S.V. Nepomnyaschikh. A Schwarz alternating method in a subspace. *Soviet Mathematics*, 29 (10):78–84, 1985.

[7] P.L. Lions. On the schwarz alternating method i. In *First International Symposium on Domain Decomposition Methods for Partial Differential Equations*, pages 47–70. SIAM, 1988.

[8] J.H. Bramble, J.E. Pasciak, A.H. Schatz. The construction of preconditioners for elliptic problems by substructuring, I. *Math. Comp.*, 47:103–134, 1986.

[9] J.H. Bramble, J.E. Pasciak, A.H. Schatz. The construction of preconditioners for elliptic problems by substructuring, IV. *Math. Comp.*, 53:1–24, 1989.

[10] P.E. Bjørstad, O.B. Widlund. Iterative methods for the solution of elliptic problems on regions partitioned into substructures. *SIAM J. Numer. Anal.*, 23:1093–1120, 1986.

[11] A.V. Smirnov. Domain coupling with the DOVE scheme. In *Parallel CFD 2003*, Moscow, Russia, 2003. Russian Academy of Sciences.

[12] A.V. Smirnov. Multi-physics modeling environment for continuum and discrete dynamics. In *IASTED International Conference: Modelling and Simulation*, number 380-174, Palm Springs, CA, 2003.

[13] A.V. Smirnov, I. Yavuz, C. Ersahin, and I. Celik. Parallel computations of turbulent wakes. In *Parallel CFD 2003*, Moscow, Russia, 2003. Russian Academy of Sciences.

[14] Joseph O'Rourke. *Computational Geometry in C*. Cambridge Univ. Press, 1998.

[15] A.V. Smirnov, W. Huebsh, and C. Menchini. A flow-solver with flexible boundaries. In *IASTED International Conference*, number 380-252 in Modelling and Simulation, Palm Springs, CA, 2003.