The structure of INMOST program platform and its usage for numerical modeling problems[*]

A. Danilov[1], K. Terekhov[1,2], I. Konshin[1], Yu. Vassilevski[1]

Institute of Numerical Mathematics RAS[1], Stanford University[2]

The INMOST program platform allows a user to work with distributed data on general meshes. The description of platform structure, the interrelation of mesh elements, the work with ghost cells, distribution and redistribution of mesh data is presented. Special aspects of the program platform implementation and usage as well as advantages over the analogous libraries are analyzed. For one of the specific tasks the exploiting of the INMOST program platform is demonstrated on the all stages of numerical modeling: distributed meshes construction, attachment of data to the mesh elements, the use of the mesh data for problem discretization, as well as the parallel solution of resulting linear systems.

1. Introduction

The amount of software for unstructured mesh generation, mesh adaptation, numerical analysis and graphical visualization is huge. The problem sizes and the computational power of modern computer systems increases rapidly, hence recent software development requires the use of parallel algorithms with distributed data. All these applications, undoubtedly, have a common set of needs for representing and manipulating distributed unstructured meshes. The typical infrastructure needed by these applications is a set of data structures for representing the mesh and software mechanisms for accessing and modifying mesh data. A mesh representation consists of topological mesh entities (vertices, edges, faces, cells) and topological adjacencies (interentity connections). The combination of a mesh representation and a set of access mechanisms for mesh data is called a mesh framework or meshing infrastructure. Although the need for common infrastructure to enable the rapid and efficient development of mesh based programs is obvious, no single framework is considered flexible enough and commonly accepted. One reason for the scarcity of general meshing infrastructures is that the needs of the various meshing and analysis applications vary widely and there is little agreement on the computational efficiency of different mesh data representation. Therefore, a large number of mesh representations are in use in the computational community, each tailored to a specific application. Some simple numerical analysis programs use only a minimal representation consisting of elements (quads, tetrahedra, etc.) defined by nodes (or points). Other more sophisticated applications, like complex finite volume schemes for general polyhedral meshes find it useful to exploit a much wider mesh representation consisting of a full set of mesh entities with an extensive set of topological adjacencies [1]. Therefore, to gain widespread acceptance, it is important to have a full mesh framework which allows applications to access all types of mesh entities. At the same time, the infrastructure should be lightweight and efficient to have sufficient utilities for real world applications.

Fortunately, the need for general parallel meshing infrastructure is increasingly being recognized and several development efforts have been introduced in the last few years. These include the following packages: MSTK (Mesh Toolkit) [2], STK (Sierra Toolkit) [3], MOAB (A Mesh-Oriented datABase) [4,5], FMDB (Flexible distributed Mesh DataBase) [6]. Some of these packages does not support dynamic modification of the mesh, some of them does not provide enough parallel functionality like several layers of ghost cells, some of them are still not reliable. Due to lack of appropriate parallel mesh framework for complex numerical analysis applica-

[*]The work was supported in part by Russian Foundation for Basic Research (RFBR) grants 14-01-00830 and 15-35-20991.

tions in early 2010s our group decided to design a new parallel platform with flexibility and efficiency in mind – INMOST (Integrated Numerical Modeling Object-oriented Supercomputing Technologies).

INMOST is a newly developed, flexible and efficient numerical analysis framework that provides application developers low-level infrastructure for reading, writing, creating, manipulating and partitioning distributed unstructured meshes without having to design and implement their own mesh data structures. INMOST provides a convenient infrastructure for matrix assembling hence simplifying rapid development of discretization techniques. INMOST also provides a framework for parallel linear system solvers including third party solver packages PETSc and Trilinos. Newly developed parallel linear system solvers may be easily incorporated or built upon INMOST infrastructure.

In this paper, a brief description of INMOST is provided including details on the design of the framework and the software mechanisms for interacting with it. INMOST is in active development and some of the capabilities are not described here since their functionality may change in near future. However, the core functionality is considered stable and is described in this paper. INMOST is currently used to implement several applications such as safety analysis of nuclear and radioactive facilities, free surface computational fluid dynamics modeling, and black oil modeling within Institute of Numerical Mathematics RAS, Nuclear Safety Institute RAS and Stanford University. INMOST is currently available to general public under Modified BSD License at http://www.inmost.org/.

2. Platform structure

Only one general assumption on the mesh type is in place. The volume mesh is considered to be a conformal mesh with arbitrary polyhedral cells, i.e. any two cells either do not have common points, either have one common vertex, either have one common edge, either have one common face. INMOST platform have a modular structure, allowing development of new modules and interfacing with third party software packages. The two core modules – mesh framework module and linear system solver module will be covered in the current paper.

INMOST mesh is a dynamic distributed database representing unstructured grid. Operations of mesh modification and parallel mesh redistribution for load balancing require special data structure which is capable of data relocation, pruning and compactification. Fast data access is one of the key features of INMOST. The direct memory access interface is used avoiding unnecessary data copying. This requires special attention in platform implementation ensuring a continuous direct element data access during mesh modification at least until the mesh element is moved or deleted. Memory fragmentation during massive mesh modifications becomes a significant concern. INMOST platform utilizes a set of average size memory blocks for big data arrays. Each block is considered nonrelocatable, hence ensuring continuous direct access, the blocks may be freed only during data compactification. INMOST implementation of the mesh data structure is simple, flexible and memory efficient with wide range of supported functionality. The detailed analysis of different mesh representations and supporting algorithms was performed by R. Garimella in his work [7]. INMOST mesh framework is based on full mesh representation with circular adjacencies vertex–edge–face–cell–vertex providing the balance between memory requirements and parallel algorithms efficiency.

INMOST platform provides infrastructure for parallel grid generation. The user may rely on internal topology consistency checks and geometry calculators. The user controls which type of computed metric data for each type of mesh entities should be precomputed, cached and updated on modification. The minimalistic parallel structured mesh generator will be presented below. Optionally the user can import and export the mesh in commonly used formats, internal cross architecture portable binary format is used to store the full set of mesh features.

One of the key points in parallel implementation of numerical analysis applications is the idea of domain decomposition and overlapping grids with ghost cells [8]. Ghost cells on one process are the cells mirroring the corresponding cells from another process. Different discretization schemes may require different types and width of overlapping layers. In INMOST each mesh entity have exactly one process owner, and a specific state for each sharing process: “owned” for entities, which only the current process owns, “shared” for own entities, which have copies on other processes, and “ghost” for copies of entities from other processes. Each shared entity also stores the list of processes it is copied to. INMOST automatically computes and distributes the ghost cells given the number of layers and the connectivity type: neighbours through the faces, edges or vertices. Special attention is given to handling of multiple ghost layers, since cross-process transfers occur commonly. The user can also provide an explicit ghost map for specific applications.

The second key point in effective parallel computation is the load balancing. INMOST provides flexible interface to repartitioning and redistribution of the mesh. The user chooses one of the internal partitioners, the external widely used partitioners Zoltan and Parmetis, or provides the partitioning map explicitly. The redistribution process effectively utilizes the ghost layers if any were computed in advance and ensures the ghost layers structure is preserved after the redistribution.

INMOST platform provides flexible infrastructure to organize mesh entities into hierarchical sets and associate user data with mesh entities. The user data is identified by named tags. Each tag may be associated with vertices, edges, faces, cells, sets, entire mesh or any combinations of them. Each tag stores real, integer or byte data or references to elements. Fixed and dynamic length arrays are natively supported. Tags may be dense, i.e. all appropriate mesh entities have the associated tag, or sparse, i.e. only some of the entities store the tag data. Markers are used in the same way as a boolean type tags. The tags data is mirrored on demand to ghost cells, in addition INMOST supports common reduce operations to gather data from ghost cells back to owner.

INMOST solver class provides methods for sparse matrix assembling without prior knowledge of matrix sparsity structure. The linear solver infrastructure may be used to develop and implement specific iterative linear solver with special preconditioner. Several internal solvers are provided natively, and convenient interfaces to PETSc and Trilinos solver packages are included as well.

The detailed algorithms and data structures are presented in [9].

3. User interface

```
#include "inmost.h"
using namespace INMOST;
int main(int argc, char *argv[])
{
  // Initialize INMOST activities
  Solver::Initialize(&argc,&argv,"");
  Mesh::Initialize(&argc,&argv);
  Partitioner::Initialize(&argc,&argv);
  ...
  // Finalize INMOST activities
  Partitioner::Finalize();
  Solver::Finalize();
  Mesh::Finalize();
  return 0;
}
```

Figure 1. Initialization and finalization of INMOST modules

The INMOST code is written in C++ language, it is cross-platform and may be built using wide range of compilers. Modular structure of INMOST code allows the user to enable and disable optional components including interfaces to third party software. The code is organized in such a way, that the user does not need to exploit any of MPI specific procedures in order to create a parallel application. If any INMOST module is used, it should be properly initialized and finalized before and after of its use respectively, refer to Fig. 1 for example of several modules usage.

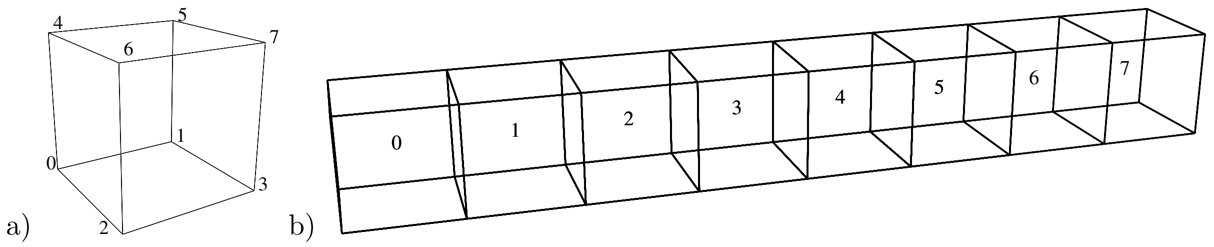

Figure 2. Minimalistic parallel cubic grid: a) one cell with local vertices, b) distributed grid with NP=8

```
Mesh *m = new Mesh();
int rank = m->GetProcessorRank(); // Get process rank
int size = m->GetProcessorsNumber(); // Get number of processes
ElementArray<Node> verts(m); // Create local vertices
for(int k=0; k<2; k++)
  for(int j=0; j<2; j++)
    for(int i=0; i<2; i++)
    {
      double xyz[3]={rank+i, j, k}; // Define node coordinates
      verts.push_back(m->CreateNode(xyz)); // Add new vertex
    }
// Define face connectivity template for cubic cell, rf. Fig. 2a
const int face_nodes[24] = {0,4,6,2, 1,3,7,5, 0,1,5,4,
                            2,6,7,3, 0,2,3,1, 4,5,7,6};
const int num_nodes[6]   = {4, 4, 4, 4, 4, 4};
m->CreateCell(verts,face_nodes,num_nodes,6); // Create the cubic cell
m->ResolveShared(); // Resolve duplicate nodes
m->Save("mesh.pvtk"); // Export mesh to parallel VTK format
delete m;
```

Figure 3. Minimalistic parallel mesh generation

In order to create a parallel mesh the user should set an MPI communicator, by default an alias INMOST_MPI_COMM_WORLD is provided for convenience. Once the communicator is set, the process may obtain its rank, and the total number of processes becomes known. We demonstrate the minimalistic distributed mesh generation process below (see Figs. 2b, 3). One way to create a polyhedral cell is to define all nodes, all edges using nodes, all faces using edges, and a cell using faces. Alternative short way is to define nodes, and create cell using connectivity template for its faces. In our example we use the second way: we define 8 nodes, and create a cell with six faces, each face containing four nodes, and their indices are prescribed by face_nodes template (see Fig. 2a). Once local grids are created we use ResolveShared() procedure to resolve duplicate nodes and assign global IDs to mesh entities. The distributed mesh is exported using Save() method.

Different discretization techniques result in matrix assembling. We demonstrate the template for finite volume (FV) scheme in Fig. 4. In general case matrix elements depend on the neighboring ones, thus a layer of ghost cells is usually needed. The method ExchangeGhost() is used to create one or several layers of ghost cells. We use two point flux approximation in FV scheme, hence for each internal face we need its area and barycenters of neighboring cells. Also we need cell volumes for right-hand side calculation. INMOST can precompute and cache the needed geometric information using PrepareGeometricData() method. Several entity iterators are available in INMOST. In our FV scheme we iterate over all faces and compute matrix co-

```
void matrix_assemble(Mesh *m)
{
  // Create a new tag for the solution phi
  m->ExchangeGhost(1,FACE);
  Mesh::GeomParam table;
  table[BARYCENTER] = CELL;
  table[MEASURE] = CELL | FACE;
  m->PrepareGeometricData(table);
  Solver::Matrix A;
  Solver::Vector x, b;
  for(Mesh::iteratorFace f = m->BeginFace(); f != m->EndFace(); ++f)
  {
    Cell r1 = f->BackCell(),  r2 = f->FrontCell();
    ... // Check for cells state r1->GetStatus(), r2->GetStatus()
    ... // Compute matrix coefficients
    int id1 = r1->GlobalID(),  id2 = r2->GlobalID(); // Get global ID
    A[id1][id2] += ...; // Fill matrix A coefficients
    b[id1] += ...; // Adjust RHS vector
  }
  ...  // Iterate over all cells and compute RHS
  Solver S(Solver::INNER_ILU2); // Specify the linear solver
  S.SetMatrix(A); // Compute the preconditioner for the original matrix
  S.Solve(b,x);   // Solve the linear system with the preconditioner
  Tag phi = m->CreateTag("Solution",DATA_REAL,CELL,NONE,1);
  for(Mesh::iteratorCell c = m->BeginCell(); c != m->EndCell(); ++c)
        if( c->GetStatus() != Element::Ghost )
                c->Real(phi) = x[cell->GlobalID()];
}
```

Figure 4. Assemble matrix and solver linear system

efficients for neighboring cells BackCell() and FrontCell(). Global cell identificators GlobalID() are used as indices to create matrix and right-hand side vector. Once the matrix is assembled we utilize internal BiCGStab(L) solver with second order ILU factorization as preconditioner. The computed solution is stored in tag “Solution”, a single real value attached to all cells.

More complicated examples and test cases are bundled in INMOST package and demonstrate in more detail mesh generation, mesh partitioning and redistribution, matrix assembling for FV scheme of diffusion problem and linear system solution using different solvers and packages. The user is advised to consult with online documentation available on project site and take a look at unit tests for mesh and solver modules.

4. Conclusions

The INMOST program platform is presented which allows a user to work with distributed data on general meshes. The internal platform structure is addressed showing the flexibility and efficiency of the infrastructure. Several user interface examples are used for minimalistic demonstration purposes of mesh generation, matrix assembling and linear system solution.

References

1. Danilov A.A., Vassilevski Yu.V. A monotone nonlinear finite volume method for diffusion equations on conformal polyhedral meshes // Russ. J. Numer. Anal. Math. Modelling. 2009. Vol. 24, N. 3. P. 207227.

2. Garimella R.V. MSTK – A Flexible Infrastructure Library for Developing Mesh Based Applications // 13th International Meshing Roundtable, September 19-22, 2004, Williamsburg, Virginia, USA, Proceedings. 2004. P. 203-212.

3. Edwards H.C., Williams A.B., Sjaardema G.D., Baur D.G., Cochran W.K. SIERRA Toolkit

Computational Mesh Conceptual Model. Technical Report SAND2010-1192, Sandia National Laboratories, 2010.

4. Tautges T.J. MOAB-SD: Integrated Structured and Unstructured Mesh Representation // Engineering With Computers. 2004. Vol. 20, N. 3. P. 286293.

5. Tautges T.J., Meyers R., Merkley K., Stimpson C., Ernst C. MOAB: A Mesh-Oriented Database. Technical Report SAND2004-1592, Sandia National Laboratories, 2004.

6. Seol E.S. FMDB: flexible Distributed Mesh Database for Parallel Automated Adaptive Analysis, Ph.D. Thesis, Rensselaer Polytechnic Institute, 2005.

7. Garimella R.V. Mesh Data Structure Selection for Mesh Generation and FEA Applications // International Journal of Numerical Methods in Engineering. 2002. Vol. 55, N. 4. P. 451–478.

8. Bertsekas D.P., Tsitsiklis J.N. Parallel and Distributed Computation: Numerical Methods. Prentice-Hall, 1989. 730 p.

9. Vassilevski Yu, Konshin I., Kopytov G., Terekhov K. INMOST a software platform and a graphical environment for development of parallel numerical models on general meshes. Moscow State Univ. Publ., Moscow, 2013, 144 p. (in Russian).