Parallel Support in MOAB

Introduction

This report describes the support for parallel computing in MOAB. Information is included from both the MOAB point of view (e.g. how parallel IO is done) and the application point of view (e.g. how this functionality is accessed by applications). This report also includes performance data for various computer architectures of interest to SciDAC and GNEP, for both moderate- and large-sized meshes. There are various axes of variability in parallel support in MOAB and how it is implemented. The important axes considered in this report are:

- *Target mesh decomposition & state:* Applications have varying requirements on the decomposition of mesh across processors. Some need the entire mesh on every processor, while others need an element-based decomposition. There are also details of what each processor knows about inter-processor interface mesh, and whether processors keep a copy of mesh close to the interface on other processors (ghost elements). These factors can affect the strategy used to load and initialize a mesh in parallel.
- *Machine architecture:* MOAB is meant to run in serial and in parallel, and for the latter is intended to support both cluster and large-scale parallel architectures. In general, we avoid any algorithms which will limit scalability to even millions of processors, at least where programming resources allow.
- *Parallel IO strategy:* Parallel IO is a major issue for most large-scale parallel computing applications. From the mesh point of view, this issue is more visible on the input side. Various strategies can be used to load and initialize a mesh in parallel. Our general strategy is to provide options which do not limit performance while also being relatively easy to use from an overall simulation process point of view.

This report is structured by the various types of functionality needed by parallel applications. The general approach used to represent parallel mesh and how that information is embedded in the MOAB data model is described in Section 2. Section 2 also describes the parallel architectures and the specific meshes used in this report to describe parallel performance. Parallel input and output is described in Section 3. How a parallel mesh appears in the data model, including locally owned mesh, ghost elements, and communication interfaces, is described in Section 4. Tools available for parallel communication in MOAB, including requesting the setup of ghost elements, are described in Section 5. Conclusions and future work appear in Section 6.

Details about the parallel methodology are discussed in the main body of the report. Performance on various parallel architectures appear with each functionality description. Those wanting to know simply how to access the functionality can skip directly to Appendix A.

General Approach

The general strategy for supporting parallel computing in MOAB is based on a distributed memory, MPI-based model, where each processor runs a single process. We assume each processor runs a single instance of MOAB. Information about the parallel nature of the mesh is embedded in the MOAB data model. For example, entity sets are used to represent both the portion of mesh local to a processor and mesh on inter-processor interfaces, with the latter linked as children to the partition sets. Tags are used to distinguish these sets from other types of sets in the mesh. While there are specific tag names used by convention to denote parallel partitions (PARALLEL_PARTITION) and interfaces (PARALLEL_INTERFACE), some parallel functionality is implemented to allow use of any application-specified tag to be used for this purpose. Unless otherwise stated, all functions called

through the MOAB interface are local, that is, they require no communication with other processors.

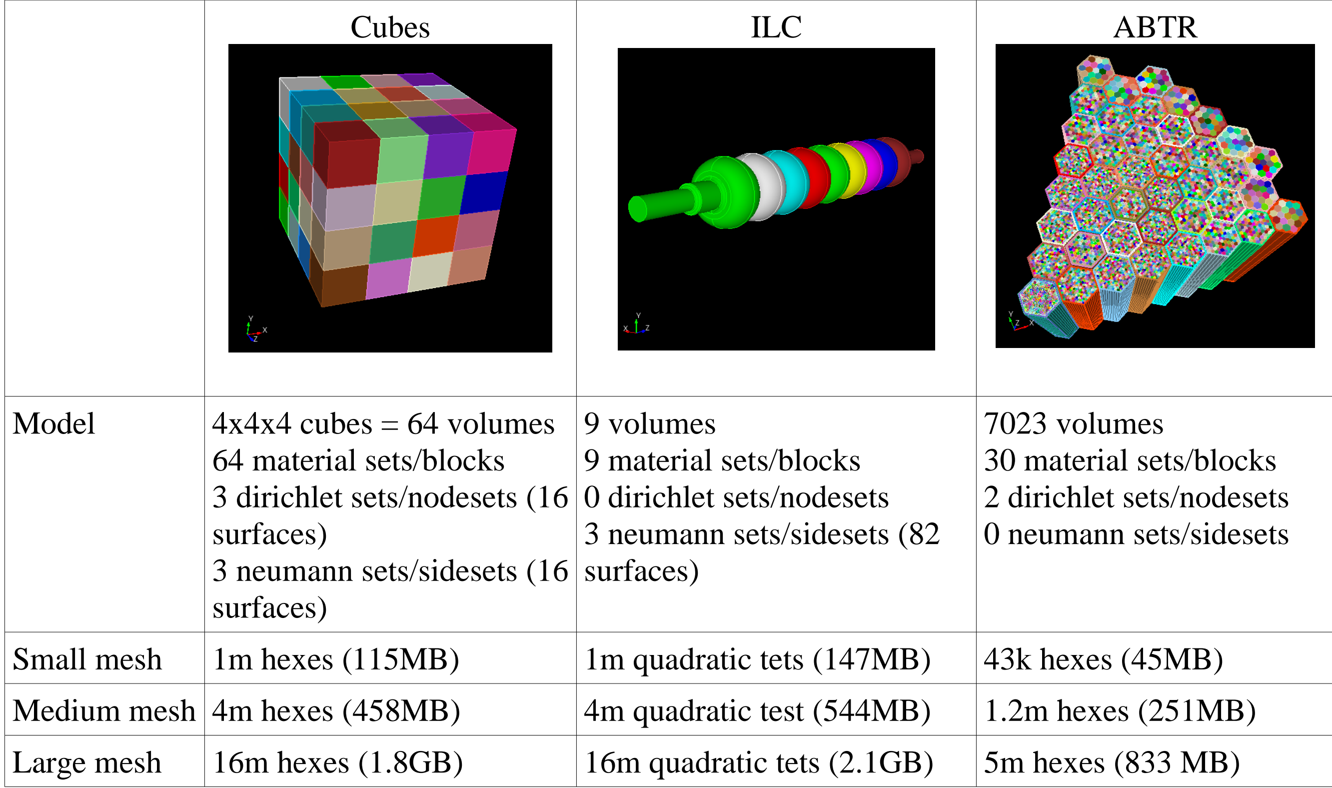
Benchmark Platforms

Three machines are used to characterize performance in this report. These machines span the spectrum of small multi-processor workstations to large-scale leadership computing platforms. Our goal is for MOAB to perform reasonably well in all these environments. The specific platforms used in this report are:

- ***Workstation:*** a dual-processor, quad-core workstation, with 4GB memory on each processor. The processors are 3GHz Intel Xeon 5365, with a SPEC-FP rating of ... This workstation has a peak Linpack rating of ...
- ***Cluster:*** a 128-node cluster, with xx dual-core processors per node and xx GB of memory per processor. Processors are xxxGHz Intel Xeon xxxx, with a SPEC-FP rating of ... The communication interface is base on ... This cluster has a peak Linpack rating of ...
- ***BGP:*** This is an IBM Blue Gene P, located at the Leadership Computing Facility at Argonne National Lab. This machine consists of ... processors each rated at xxx SPEC-FP.

Benchmark Meshes

We consider a small number of models with varying geometry and mesh complexity. The three geometric models and the meshes used for these models are shown in Table 1.



	Cubes	ILC	ABTR
Model	4x4x4 cubes = 64 volumes 64 material sets/blocks 3 dirichlet sets/nodesets (16 surfaces) 3 neumann sets/sidesets (16 surfaces)	9 volumes 9 material sets/blocks 0 dirichlet sets/nodesets 3 neumann sets/sidesets (82 surfaces)	7023 volumes 30 material sets/blocks 2 dirichlet sets/nodesets 0 neumann sets/sidesets
Small mesh	1m hexes (115MB)	1m quadratic tets (147MB)	43k hexes (45MB)
Medium mesh	4m hexes (458MB)	4m quadratic test (544MB)	1.2m hexes (251MB)
Large mesh	16m hexes (1.8GB)	16m quadratic tets (2.1GB)	5m hexes (833 MB)

Table 1: Test models and meshes used for benchmarking parallel IO with MOAB. File sizes shown for each mesh are for HDF5-based h5m files on a 64-bit linux workstation.

Parallel IO

We consider parallel input first. The reading and initialization of a mesh in parallel is controlled

primarily by how the mesh partition is specified, that is, what determines which part of the mesh goes on which processor. In general terms, a partition is simply a covering of some set of entities in the mesh; that is, a grouping of these entities into a collection of sets, where each entity is included in exactly one of the sets in this collection. This definition has two important implications. First, nothing has been stated about the characteristics of the partition in terms of its balance (the number of entities in each part or communication links between parts). This type of information is inherently application-specific, although there are some common characteristics requested by many applications. Second, the types of entities in the partition has not yet been specified. Traditionally, the elements of maximal dimension in a mesh are partitioned, but a partition could also be based on elements of several dimensions (e.g. elements and faces in a mesh), or entity sets representing some other grouping in the mesh. We have found it useful to implement parallel functionality based on more general concepts of a partition, allowing the application to choose the specific partition type, where possible.

Input

When run in parallel, MOAB appears as a separate instance on each processor. Applications load mesh in parallel by passing special options to the MOAB load_file function, e.g.

```
const char *popt =   PARALLEL=BCAST_DELETE;PARTITION=MATERIAL_SET  ;
MBEntityHandle file_set;
MBErrorCode result = impl->load_file(  mymesh.cub  , file_set, popt);
```

Note that options are passed in a string, with each string delimited by a semicolon and option values appearing after '='.

Parallel input is controlled by the parallel load strategy and the partition designation, both input in the option string.

Parallel Load Strategy: The strategies for parallel mesh input are listed in Table 2. Each strategy is identified with a name, which is passed as a value of the PARALLEL option, e.g.
  PARALLEL=BCAST_DELETE .

Strategy Name	Description
BCAST	Read the mesh on the root and broadcast to all processors; every processor keeps a full copy of the mesh.
BCAST_DELETE	Like *BCAST* strategy, except that after mesh is broadcast, each processor deletes all mesh not assigned to it. Entity sets which are empty afterwards are also deleted, as are tags not set on any entities (DENSE-type tags with default values are kept).
READ_DELETE	Like *BCAST_DELETE*, except instead of the root reading and broadcasting the mesh, each processor reads the mesh concurrently.
READ_PARALLEL	True parallel read, where each processor reads only its portion of the mesh. This option is available only with HDF5-format files, and is implemented on top of Parallel HDF5. Collective communication is used to read file metadata information, then each processor reads different portions of the file concurrently. This strategy is described in more detail later in this report. (NOTE: this is not implemented yet).

Table 2: Parallel read strategies, requested by adding   PARALLEL=<strategy name>   as an option to the load_file function.

Partition Designation: A partition on a mesh is a collection of sets, with each processor responsible for one or more sets and each set assigned to exactly one processor. In MOAB, applications request a partition by identifying the tag name, the value(s), and whether sets with those tags and values are distributed across processors. The partition designation is controlled by the options listed in Table 3. Specifying a partition using a tag name and value allows an application to load a mesh in parallel before partitioning that mesh for parallel solution. This allows one to run the partitioning itself with a distributed mesh, for example.

Option	Value	Description
PARTITION	<tag_name>	Sets with the specified tag name should be used as partition sets
PARTITION_VAL	<val1, val2-val3, ...>	Integer values to be combined with tag name, with ranges input using val2-val3. Not meaningful unless PARTITION option is also given.
PARTITION_ DISTRIBUTE	(none)	If present, or values are not input using PARTITION_VAL, sets with tag indicated in PARTITION option are partitioned across processors in round-robin fashion.

Table 3: Options indicating the partition to be used in a parallel read and initialization.

Several example option strings controlling parallel reading and initialization are:

PARALLEL=READ_DELETE; PARTITION=MATERIAL_SET; PARTITION_VAL=100, 200, 600-700 : The whole mesh is read by every processor; this processor keeps mesh in sets assigned the tag whose name is MATERIAL_SET and whose value is any one of 100, 200, and 600-700 inclusive.

PARALLEL=BCAST_DELETE; PARTITION=PARALLEL_PARTITION, PARTITION_VAL=2 : The root processor reads the mesh and broadcasts it to all processors; this processor, whose rank is 2, is responsible for elements in a set with the PARALLEL_PARTITION tag whose value is 2, and deletes all mesh not contained in that partition. This would by typical input for a mesh which had already been partitioned with e.g. Zoltan or Parmetis.

PARALLEL=BCAST_DELETE; PARTITION=GEOM_DIMENSION, PARTITION_VAL=3, PARTITION_DISTRIBUTE : The root processor reads the file and broadcasts the whole mesh to all processors. If a list is constructed with entity sets whose GEOM_DIMENSION tag is 3, i.e. sets corresponding to geometric volumes in the original geometric model, this processor is responsible for all elements with index R+iP, i >= 0 (i.e. a round-robin distribution).

Output

For output, each processor can call MOAB's save_file function to save its local copy of the mesh to a file. To coordinate writing of the parallel mesh from all processors, the save_file function is passed an option string containing the PARALLEL sub-option. Note that currently, parallel saving only works when writing an HDF5 file, which normally has a .h5m file extension. This method will write a single file from all processors, without duplicated or ghosted entities and with common entity sets containing the union of those sets' contents over all processors.

Parallel Mesh Representation

The MOAB data model consists of mesh entities, mesh sets, tags, and the interface instance. Wherever possible, MOAB embeds data in this data model, to simplify the interface to these data. Following this principle, information about a parallel mesh is stored in MOAB using entity sets and tags. Retrieving that information requires simply knowing which tags are used to describe parallel information. These tags are described in Table 4. This information is also stored in the MBParallelConventions.h file in the MOAB source code.

To find a given piece of information about a parallel mesh, an application simply requests the entities with a given tag, and optionally a value of that tag. For example, to find the entities in a partition on the local processor, an application would call the get_entities_by_type_and_tag function in MOAB, passing the PARALLEL_PARTITION tag, MBENTITYSET as the entity type, and NULL as the value (or, if the application knows the partition comes from a mesh partitioner and that there is one partition per processor, the processor rank can be passed as the value). In most cases, tags will be set on entity sets, rather than individual entities, to reduce memory usage. Also, in many cases, those entity sets will not contain entities, but will contain entity sets, e.g. when an interprocessor interface corresponds to a geometric topology entity (like a model face). Therefore, applications should be careful to request entities recursively if they want the actual mesh entities contained by the set.

Tag Name	Assigned to...	Value holds...	Description
PARALLEL_ PARTITION	Entity sets	Rank of processor this Part is assigned to	Each set with this tag holds mesh assigned to a processor; this set is referred to as a Part in the partition. A mesh can have multiple partitions, but only one partition will be identified with the PARALLEL_PARTITION tag. A mesh entity can be part of exactly one Part in a given partition.
PARALLEL_ GID	Entities	Global id of (shared) entity	Global id used to identify entities appearing on multiple processors, either as shared or ghost entities.
PARALLEL_ SHARED_ PROC	Entity set	Ranks of 2 processors sharing this set	Stores 2 integers corresponding to two processors sharing entities in the set. If more than two processors share a set, use the PARALLEL_SHARED_PROCS tag instead.
PARALLEL_ SHARED_ PROCS	Entity set	Ranks of processors sharing this set	Stores several integers corresponding to processors sharing entities in the set. If only two processors share a set, use the PARALLEL_SHARED_PROC tag instead. The length of this tag is determined by the tool generating the partition, and can be found using the MOAB tag_get_size function.
PARALLEL_ OWNER	Entity or set	Rank of owning processor	For entities shared between processors, this tag denotes the processor which owns the entity. The owning processor is responsible for output related to that entity, and in many applications for computing field data for that entity.
PARALLEL_ GHOST	Entity or set	Rank of owning processor	Entities or sets with this tag are ghosted from the processor whose rank is stored in the tag.

Table 4: Tag names used to indicate parallel data. Tags are assigned to entities or entity sets. Presence of tag on an entity or set can be meaningful in itself, with tag value sometimes indicating addition information.

Query Functions

In most cases, applications can use existing set/tag functionality in MOAB to find information about a parallel mesh. However, there are some convenience functions which can make this job easier and which are implemented in MOAB's MBParallelData class (this class is located in the parallel/ subdirectory, and is not available unless MOAB has been compiled to use MPI). These functions are:

MBParallelData::get_partition_sets(MBRange &part_sets, const char *tag_name = NULL): get partition sets; if tag name is input, use that tag name to designate partition sets, otherwise use PARALLEL_PARTITION_TAG_NAME (defined in MBParallelConventions.h) to indicate tag name.

MBParallelData::get_interface_sets(std::vector<MBEntityHandle> &iface_sets, std::vector<int> iface_procs): get entity sets representing entities shared with other processors, and the processors those sets are shared with. Sets and processors are returned in order of increasing processor rank, therefore all sets interfaced with a given processor are adjacent in the returned list, and sets may be repeated in

this list if they are communicated with multiple processors.

Parallel Functions

Besides finding information about a partition and interfaces between partitions, there are other types of functionality commonly needed by parallel applications. Functions for this purpose provided by MOAB are described here, and typically appear in classes in the parallel/ subdirectory of the MOAB source code.

Ghost Entities & Data

get_ghost_entities(from_dim, to_dim, bridge_dim, proc)

exchange_ghost_data(tag(s))

Performance

Performance of reading and writing mesh in parallel for MOAB is measured using the platforms and models described earlier in this report.

Conclusions & Future Directions