Summary of Unstructured Mesh Parallel Preprocessor

Dimitri Mavriplis

12/18/2017

Overview

The pre_nsu3d parallel preprocessor performs the following functions:

1. Read input mesh in parallel
2. Construct line data structures for implicit line solver along prismatic cells
3. Partition mesh ensuring no lines are cut across processors
4. Generate coarse agglomerated multigrid levels (AMG)
5. Compute distance function
6. Write out all data in parallel needed by flow solver

Parallel partitioning is done by linking to the ParMETIS or SCOTCH libraries. ParMETIS is a product of the University of Minnesota, and SCOTCH is a product of INRIA in France.

The input formats supported in item 1 are:**cogsg** (VGRID native), **mcell.unf** or ***.mcell** (NSU3D native), **and b8.ugrid** (SOLID MESH stream IO). Cogsg and mcell files use the fortran sequential access unformatted format. Reading sequential access fotran unformatted files in parallel can be very inefficient with some compilers (notably ifortV13). Therefore pre_nsu3d reads these files as stream IO, taking into account the record length indicators in the sequential unformatted files. Record lengths larger than 2 GBytes are not supported. For VGRID files, this limit is hit around 125M cells, and for mcell.unf, around 500M cells of any single type (i.e. a hybrid mesh of 200M tets and 300M prisms will be readable). The reader will output a message if the record length is larger than the supported value. There is no limit on the size of the b8.ugrid files, except if any integer exceeds the values representable with 31 bits (around 2 billion). For VGRID and mcell files larger than the limit, a sequential converter code will be supplied to either write these as b8.ugrid files or as mcell8 files, which will also support 64 bit integers for very large meshes. Tecform and slice will be upgraded to read native b8.ugrid files as well.

Note that when cogsg (VGRID) files are read in, pre_nsu3d will automatically perform the prism merging operations in parallel and write out a new mcell.unf file which can be used thereafter. Since the resulting mcell.unf file has a small number of additional vertices compared to the original VGRID grid, the dist and amg files will be tagged with this node number and will report an error if read in with the original cogsg file. Therefore the mcell file should always be used once the merging operation has been performed.

At present, item 4 (AMG levels) is performed on a single computer node and item 5 (distance function) can be time consuming so these are done the first time a new grid is preprocessed, and the results are written out (in parallel) to a file. When the preprocessor is run again on the same grid, if these files exist, they are read in (in parallel) and used directly to generate the final output for the flow solver. Thus, meshes can be repartitioned in a matter of seconds or minutes for most cases once the AMG levels and distance function have been computed and stored to files.
The preprocessor has been tested on grids up to 120 million points (vertices) or 300 million cells (mixed tetrahedral, prisms and pyramids) for up to 32,678 partitions.

The memory requirements of the preprocessor run about 8 Gbytes per 1 million grid points (about 2Gbytes per 1M cells) which is distributed over the parallel machine.

The memory requirements for the AMG level construction are about 700Mbytes per 1 million grid points. Since this portion of the preprocessor executes sequentially, memory requirements may be a bottleneck for large grids. However, for most meshes adequate memory is available on a single multi-core node. As an example, the AMG portion of the preprocessor required 7Gbytes of memory and 82 secs to process a 10 million point mesh on a desktop Linux workstation.
The largest case tried to date was a 120 million point mesh that required 77 GBytes and about 15 minutes to generate the AMG levels on a fat memory cluster node. A fully parallel version of the AMG level generation is under construction.

Three methods are available for distance function calculation: a brute force method, a marching spheres search method, and an Eikonal equation solver**. (The previously supported octree method is currently disabled/not supported.)**
The brute force method takes significant amount of time but scales well on large numbers of cores. This method can be expected to require of the order of 30 minutes for many medium size meshes on moderate numbers of cores. This is the default method as it gives the exact distance function result and is the most robust overall.
The marching spheres method yields the same exact result as the brute force method but runs considerably faster by making use of an efficient search technique. This is enabled through the **drums** library. For large grids, this method is preferable to the brute force search method. The main issue with this approach is that it occasionally results in failure. In this instance, the brute force search method should be used.
The Eikonal equation solver relies on the solution of a PDE to obtain the distance function and scales well, although it requires cpu time equivalent to several hundred flow solver iterations. The Eikonal equation solver method yields an approximate distance function which will not be exactly the same as the other methods.
All methods are setup to write the distance function to a file and to use the file info once it has been generated rather than recalculating the distance function over again.

Sample Timings:

Some sample timings for relevant size meshes are given below.

10 million point mesh divided into 16 partitions on an 8 core workstation (using hyperthreading):

```
-------------------------------------------------------------------------
TOTAL WALL CLOCK TIME:                    0.2056E+03 secs
Time required for reading mesh:           0.2791E+01 secs;  Ratio:  0.01
Time required for partitioning mesh:      0.1277E+03 secs;  Ratio:  0.62
Time required for writing out mesh:       0.7210E+02 secs;  Ratio:  0.35
-------------------------------------------------------------------------
```

The time to partition is about 2 minutes. Note the write time is rather slow, over 1 minute. This is partly due to the state of the disk and other IO traffic and can vary significantly.

This timing includes reading AMG levels and dist function from files.
In the event these are not available (i.e on the first pass) the following additional resources were required:
Sequential AMG: 7GBytes of RAM, 82 seconds wall clock time
Distance function: 10 minutes of wall clock time

75 million point mesh divided into 2048 partitions on cluster:

```
 -------------------------------------------------------------------------
TOTAL WALL CLOCK TIME:                    0.1437E+03 secs
Time required for reading mesh:           0.1100E+02 secs;  Ratio:  0.08
Time required for partitioning mesh:      0.9842E+02 secs;  Ratio:  0.68
Time required for writing out mesh:       0.2419E+02 secs;  Ratio:  0.17
-------------------------------------------------------------------------
```

In this case the AMG and dist functions when run on the first pass required the following resources:
Sequential AMG: 48 GBytes of RAM, 10 minutes of wall clock time
Distance function: ~2-4 minutes of wall clock time (depending on convergence tolerance)

75M pt mesh 32768 partitions on 16384 cores

```
-------------------------------------------------------------------------
TOTAL WALL CLOCK TIME:                    0.4902E+03 secs
Time required for reading mesh:           0.4862E+02 secs;  Ratio:  0.10
Time required for partitioning mesh:      0.3027E+03 secs;  Ratio:  0.62
Time required for writing out mesh:       0.6369E+02 secs;  Ratio:  0.13
-------------------------------------------------------------------------
```

Note that the performance degrades somewhat when the number of partitions becomes very large. Also, this case is hyperthreaded (2 threads per core) so there is slowdown per core due to this as well. This was necessary in this case to get the larger number of partitions. Note the IO also degrades, and this is purely a machine specific issue. However, this is a relatively coarse mesh for the large number of partitions requested and finer meshes would likely scale better.

The partitions generated are of high quality and good scalability can be obtained from a CFD solver running on these partitions. As an example, the figure below shows the scalability obtained with NSU3D using up to 32,768 cores on the partitions generated with this preprocessing code. Note that this case contains only 2,200 points per core at 32,768 partitions and better scaling should be achieved with a finer mesh at this level.