EGADSlite: A Lightweight Geometry Kernel for HPC

Robert Haimes*
Aerospace Computational Design Laboratory
Massachusetts Institute of Technology, Cambridge MA, 02139
and
John F. Dannenhoffer, III†
Aerospace Computational Methods Laboratory
Syracuse University, Syracuse, NY, 13244

As attention is focused upon the "time to solution", it becomes obvious that the entire process must be taken into account – not just the cost and efficiency of the solver. Amdahl's Law tells us that any serial portion of the application will be the limiting factor in scalability. Therefore it does not matter how efficient a solver is if both the pre- and post-processing have not been given the same focus towards scalability. The most obvious way to insure that a scalable process exists is to view the process as an integrated whole and remove any serial portions. The work discussed in this paper makes geometry available in a parallel environment to support parallel mesh generation, solver-based grid adaptation, and the curving of linear meshes to support high(er) order spacial discretizations.

I. Introduction

NASA recently commissioned the study: "CFD Vision 2030 Study: A Path to Revolutionary Computational Aerosciences",[1] in which several technology gaps and impediments were identified. In section 5.3 (Autonomous and Reliable CFD Simulations), "Mesh generation and adaptivity" were specifically identified as concerns, owing both to "inadequate linkage to CAD" and "poor mesh generation performance and robustness". In section 5.1 (Effective Utilization of High-Performance Computing), the concern was raised that there was a "lack of scalable CFD pre- and post-processing methods".

The study projected that in the near future, CFD grids with 10^9 vertices (or more) would be commonplace, executing in High Performance Computing (HPC) environments with 10^3 to 10^5 compute nodes, each consisting of 10s of cores; graphic processing units (GPU) were also seen as technologies that might seriously change the HPC from its current state. Further, the study identified (in several places) the need for widespread adoption of grid adaptation and a transition from one-off analyses to design optimization.

Design environments (driven by optimization) pose a significant challenge to most current Computer-Aided Engineering (CAE) areas, including CFD. The challenge is one of automation. If many thousands of design cases are to be analyzed, an expert (or any) engineer cannot be *in the loop* providing updated geometry, meshes or the post-processing of results. This challenge suggests the goal of a process that is fully automated and therefore efforts should be made to attempt to achieve that goal.

*Principal Research Engineer, Department of Aeronautics and Astronautics, AIAA member.
†Associate Professor, Mechanical and Aerospace Engineering, AIAA Associate Fellow.

II. Current bottlenecks

The discussion thus far has focused on process and taking all of the above together, one can identify the following current bottlenecks:

- The grid generation systems are not distributed throughout the HPC environment, but rather frequently operate on one HUGE front-end machine; the grid is then partitioned and distributed throughout the HPC system for execution by the parallelized CFD solvers. In addition to most likely growing bigger than any one computer could handle, this serial process is an impediment for good parallel performance (Amdahl's Law[2]).
- One of the reasons that the grid generation may not be distributed is because the underlying geometry systems are not distributed. Part of this is because the geometry systems were not built to exploit parallel and/or distributed computing environments. The other part is because most geometry systems are licensed, making their distribution across an HPC system impractical (if not prohibitively expensive).
- Even after the grid has been generated and distributed, there is a desire to adapt the grids on the processors on which the CFD solver is executing; hence the grid adaptation process must operate in a distributed manner. In the field, this is not too difficult, but for grid vertices near the surface, a major impediment to this process today is the central (i.e., non-distributed) nature of the geometry system.
- With the advent of higher-order methods, it may not be enough to simply evaluate the location of points on the surfaces of the geometry, but also the local surface slopes and curvatures in order to generate meshes with curved elements.

III. Technical Approach

The technical approach is to extend the "Engineering Sketch Pad" (`ESP`) to support the geometry needs in current and (hopefully) future HPC environments. In the section that follows, a brief description of the relevant existing capabilities will be described. This is then followed by several sections in which the specific technical tasks are discussed.

A. Existing capabilities

This work builds upon the "Engineering Sketch Pad" (`ESP`) system[3] that is currently in use at NASA, the Air Force Research Laboratory and other sites. As opposed to most commercially-available geometry-generating CAD systems, `ESP` was designed for geometries encountered in aerospace applications for which CFD (and Structural Analysis) computations are desired. `ESP` is also currently being used to generate multi-fidelity and multi-disciplinary geometries for the Air Force's "Computational Aircraft Prototype Syntheses" (CAPS) project. This `ESP` functionality allows for a single parameterized *model* to generate different disciplinary *views* of that model, each directly appropriate for the task at-hand.

`ESP` is architected with a client-server model, where the back-end runs on most modern operating systems (Windows 7/8/10, MAC OSX, and Linux) and the front-end runs in modern web browsers (Firefox, Google Chrome, Safari, Internet Explorer), without the need for plug-ins.

Since the software being discussed here all runs on the compute node *back-ends*, further discussions will concentrate on the features of the "Open-source Constructive Solid Modeler" (`OpenCSM`)[4] and the "Engineering Geometry Aircraft Design System" (`EGADS`),[5] which are central to the computations in the *back-end*.

The distinguishing features of the `ESP` subsystems `EGADS` and `OpenCSM` include:

Feature-based parametric solid modeler This means that geometries (represented as BReps) are generated by executing build recipes (feature-trees) with a prescribed set of design parameters. The design parameters can be changed (by the user or an outside program) at any time to build a new configuration. For example, in a design setting, the optimizer may want to change the wing's aspect ratio, taper ratio, thickness, and/or camber to minimize drag with a prescribed lift. The BReps produced by `OpenCSM` are generally manifold (that is, watertight with all components connected), but non-manifold sheets (useful for wakes and some structural represnetations) and wires are also supported.

Full suite of feature-tree branch types This includes standard primitive solids, solids grown from sketches, applied features, Boolean operators, and transformations.

Compiled user-defined primitives and functions (UDPs & UDFs) Unlike other geometric modelers, `OpenCSM` provides the user with the ability to incorporate user-defined primitives. Currently, UDPs are available for generating an assortment of airfoil shapes (NACA-4, -5, and -6 series, Kulfan, and Parsec) and a variety of cross-sectional shapes often found in fuselages (four-quadrant super-ellipse, boxes with rounded corners, and Bezier surfaces). The UDPs are `EGADS` applets written in C/C++ and/or FORTRAN and then compiled into shared-objects (or DLLs). These UDPs, after run-time loading, act in the same way as any `OpenCSM` geometric primitive. UDFs are like UDPs but can take as input one or more existing BRep(s).

Scripted user-defined components (UDCs) These are usually small scripts of `OpenCSM` statements that are used for often-applied tasks. Examples of UDCs that ship with `ESP/OpenCSM` are standard aircraft components (wing, fuselage, duct, strut) as well as specialized UDCs for putting a deflected flap or deployed spoiler on an existing wing.

Configuration files that are readable ASCII text This gives the user the ability to write `OpenCSM` files from other applications. Also, a human readable file facilitates the ability to communicate the *design intent*.

Persistent attribution between models This allows users (or external programs) to place auxiliary information (such as required grid cell size) on a model and have this metadata persist through regenerations (because the feature-tree or design parameters changed) or across linked models (such as models with differing fidelities or suitable for different disciplines).

Sensitivities Rapid, accurate sensitivities of the BRep with respect to the design parameters. Many times this can be done analytically (without the need to regenerate), but there are some branch types for which `OpenCSM` employ finite-differences at this time.

Open-source `ESP` is distributed as source (C, C++, FORTRAN, JavaScript, html), with the non-viral GNU Lesser General Public License (version 2.1) as published by the Free Software Foundation. The only software dependency is `OpenCASCADE`,[6] which itself is licensed under LGPL 2.1.

B. Construction of "EGADSlite"

Though the `EGADS` portion of `ESP` can be used in isolation, it represents a huge amount of code which includes the ability to build geometry (from either a bottom-up or top-down perspective), tessellate the geometry for viewing (and other operations), placing annotations on parts of the BRep, and many other functions. `OpenCASCADE`, the only dependency for `EGADS`, can be difficult (if not impossible due to the C++ nature of the code-base) to port to novel architectures (such as GPUs). But, the requirements for mesh generation and mesh adaptation represent a small subset of the `EGADS` functions which primarily are the ability to parse the BRep, provide attributes, preform evaluations and inverse evaluations, and compute the in/outside predicates.

`EGADSlite` is a lightweight ANSI-C version of the `EGADS` functions required by the meshing process, where the target is any high-performance computing equipment currently in use (including GPUs). Note that C++

can be used, but the API is essentially C. FORTRAN can also be used as the driving language with the EGADS FORTRAN binding library.

All of the EGADSlite function names are the same as those in EGADS, have the same signatures, and mostly act in the same manner including any side-effects (note that there are only 2 functions that differ in actions compared to their EGADS' namesake). This allows for code prototyping in the larger geometry environment, with confidence that, when pared down, the results will be the same. The initialization/model-read is one of the differences, where an EGADS application prepares the data for use by the EGADSlite package. This data can be written to disk or used live in a parallel setting. The latter option obviously requires a least a single node with full EGADS access. The small amount of data (in comparison to the surface discretization) can simply be broadcast throughout a high-performance machine so that all processors have the complete description of the analytic geometry and the BRep topology.

Clearly any dependence on OpenCASCADE needs to be removed in order to maintain the C (not C++) nature of the EGADSlite code base. This entailed rewriting all of the evaluation and inverse evaluation functions, which required covering the support of all of the curve and surface types used within.

The EGADSlite API is described below and is partitioned into sections having to do with functionality. For a more complete description of the API, see the EGADS documentation, which comes with the ESP distribution.[7] Overall, the EGADSlite function list contains no construction operations but only inquiry and read-only functions (for example, you can get any information about attributes, but you cannot add a new attribute or change one that exists).

1. Session Functions

The following subset of EGADS provides functions that open, control and close an EGADS session as well as provide dynamic memory arena access[a].

EG_alloc allocate memory controlled by EGADSlite

EG_calloc allocate and zero-fill a block of memory

EG_close close up and exit a context

EG_free free memory allocated by EG_alloc, EG_reall or EG_calloc

EG_open open and return a context object

EG_reall reallocates a bock of memory

EG_revision returns the version information

EG_setOutLevel sets the verbose level

2. Attribute Functions

It has become obvious during the use of ESP that attribution associated with the geometric model is essential for automation and design settings. Attribution allows for the placement of geometrically related metadata on the appropriate entities directly. This can then support the marking of boundary conditions, material properties, mesh spacings, and the like. This part of the interface is used to fully support the mission of EGADSlite and allow the application full access to this data.

EG_attributeGet retrieve a specific attribute for an object (by index)

EG_attributeNum returns the number of attributes associated with an object

EG_attributeRet get a specific attribute for an object (by name)

[a]Memory routines are required because EGADS and EGADSlite are built as shared objects/DLLs and Windows can't have an executable/DLL free memory allocated by another executable/DLL due to the lack of system level libaries.

3. Geometry Functions

These functions provide for the evaluation/inverse evaluation portion of the geometry kernel, in addition to other routines that can assist in mesh generation.

EG_arcLength returns the arclength of an curve/pcurve object

EG_curvature return the curvature and principle directions/tangents of a curve/surface

EG_evaluate returns physical coordinates and derivatives on an curve or surface

EG_getGeometry returns information about a geometric object

EG_getRange returns the valid range of a geometric object

EG_invEvaluate returns the result of inverse evaluation on an object

EG_invEvaluateGuess returns the result of inverse evaluation on an object given an initial guess

4. Topology Functions

The functions in this portion of `EGADSlite` allow for the parsing through the BRep Topology of the geometric model. This provides the routines that support the BRep hierarchy, information about neighbors and trimming. Some of the functions here were also listed above because they work on either geometry and/or topological entities.

EG_arcLength returns the arclength of an Edge object

EG_curvature return the curvature and principle directions/tangents of an Edge/Face

EG_evaluate returns physical coordinates and derivatives on an Edge or Face

EG_getBodyTopos returns topologically connected objects

EG_getBoundingBox computes the Cartesian bounding box around an object

EG_getEdgeUV computes on the Edge/pcurve to get the appropriate (u, v) on the Face

EG_getRange returns the valid range of an object

EG_getTolerance returns the internal tolerance defined for a object

EG_getTopology returns information about a topological object

EG_inFace computes the result of the (u, v) location in the valid part of a Face

EG_inTopology computes whether the point is on or contained within the object

EG_indexBodyTopo returns the index (bias 1) of the topological object in a Body

EG_invEvaluate returns the result of inverse evaluation on an object

EG_invEvaluateGuess returns the result of inverse evaluation on an object given an initial guess

5. Tessellation Functions

This subset of functions represent the complete `EGADS` tessellation package. This allows the user to build discrete versions of any geometry represented within the system. Also this turns out to be a great surrogate for a grid generation system to enable testing of the basic `EGADSlite` completeness and functionality.

This portion of `EGADSlite` is the only part that is not *read-only*. Tessellation objects can be created and deleted.

EG_deleteEdgeVert delete an Edge vertex from a Body-based tessellation object

EG_getGlobal returns the point type and index (like from `EG_getTessFace`) with optional coordinates

EG_getPatch retrieves the data associated with the patch of a Face from the Body-based tessellation object

EG_getQuads retrieves the data associated with the quad-patching of a Face from a Body-based tessellation object

EG_getTessEdge retrieves the data associated with the discretization of an Edge from a Body-based tessellation object

EG_getTessFace retrieves the data associated with the discretization of a Face from a Body-based tessellation object

EG_getTessGeom retrieves the data associated with the discretization of a geometry-based object

EG_getTessLoops retrieves the data for the loops associated with the discretization of a Face from a Body-based tessellation object

EG_getTessQuads returns a list of the Face indices found in the Body-based tessellation object that has been successfully quadded

EG_initTessBody creates an empty (open) discretization object for a topological Body object

EG_insertEdgeVerts inserts verticies into an Edge discretization of a Body tessellation object

EG_localToGlobal perform local-to-global index lookup

EG_locateTessBody provide the triangle and the vertex weights for each of the input requests or the evaluated positions in a mapped tessellation

EG_makeQuads create quadrilateral patches of the indicated Face and update the Body-based tessellation object

EG_makeTessBody creates a discretization object from a topological Body object

EG_makeTessGeom create a discretization object from a geometry-based object

EG_moveEdgeVert moves the position of an Edge vertex in a Body-based tessellation object

EG_openTessBody opens an existing tessellation object for replacing an Edge or Face discretization

EG_setTessEdge sets the data associated with the discretization of an Edge for an open Body-based tessellation object

EG_setTessFace sets the data associated with the discretization of a Face for an open Body-based tessellation object

EG_statusTessBody returns the status of a tessellation object

6. Functions with Different Side-effects

The last set of two functions belong to the *Session* suite but are those that have the same signatures but act in a different manner.

EG_deleteObject delete an object
in `EGADSlite` only Tessellation Objects created during the session can be deleted

EG_loadModel load an return a model object
general file readers do not exist

C. Memory Footprint

Analyzing the amount of memory a library takes up in a running program is not simple. It depends on whether the libraries are linked in a static or dynamic manner, what functions are utilized and the size of internal objects (used in runtime). Because of these difficulties, the size (on disk) of the dynamic library or in the case of `EGADS` libraries (including `OpenCASCADE`) is used as a surrogate for memory footprint. The comparison can be seen in Table 1.

Table 1. Comparison of size

	# of Libraries	size (Kbyte)
`EGADS`	52	65384
`EGADSlite`	1	342

These numbers were derived from an Apple MACbook Pro running OSX 10.12.5 with Xcode 8.3.3 and OpenCASCADE version 6.6.0.

D. Timings

Tests of the `EGADSlite` evaluators and inverse evaluators show that this new subsystem is more robust than the equivalent parts of `EGADS` (which is based on `OpenCASCADE`) and between 10 and 100 times faster, as shown in Table 2. These tests were performed by taking tessellation points (for which the coordinates $[x, y, z]$ are known to lie exactly on the configuration), and then performing inverse evaluation (without an initial guess). An inverse evaluation is a local optimization that minimizes the distance between the target location and a position on the surface. This is solved by Newton-Raphson iterations and therefore require *good* first and second derivatives. The optimization works very well for most analytic surface types but can be problematic when applied to B-Spline (or NURBS) surfaces. Poorly constructed (or fit) B-Spline/NURBS surfaces can display oscillations that obviously generate local minima, which trap (or overshoot) the inverse evaluator. This problem can be mitigated by trying many seed points and selecting the closest result.

In this test, a failure in finding the correct result happens when the returned $[x, y, z]$ from the inverse evaluation differs from the known $[x, y, z]$ by more than 100 times the Face's internal tolerance. Failures noted in the table all had distances significantly larger than this tolerance. It is interesting to observe that failures in `EGADS/OpenCASCADE` and `EGADSlite` happened on different configurations. This is because different seeding algorithms are used. Note that overall, `EGADSlite` was as robust as `EGADS`, but generally about an order of magnitude faster in serial/single-threaded applications.

Table 2. Comparison of timings

	EGADS		EGADSlite	
Case	nFailure	time	nFailure	time
demo2	0	0.40	0	0.04
tutorial1_whole	0	8.48	0	1.04
design2	0	0.35	0	0.04
design3	0	2.56	0	0.16
tutorial2	0	1.04	0	0.16
tutorial3	0	98.96	0	2.51
myPlane	20	765.86	0	24.24
bottle2	0	8.14	0	0.83
wingMultiModel	0	38.30	0	0.64
bullet	102	1.75	0	0.11
connect5	0	0.47	0	0.03
group2	0	1.55	0	0.14
hl-crm-gapped-flaps	0	17115.71	38	1804.50
hl-crm-sealed-flaps			16	1783.64
jsm_case01			0	150.30
jsm_case02			0	153.27

E. Threading

The `EGADS` tessellator has been threaded from the most early revisions. Timing has shown that almost perfect scalability (based on the number of cores or hyper-thread availability) is usually achievable. It should be noted that this triangulator only performs forward evaluations of Edges, parameter-space curves (pcurves) and surfaces. Inverse evaluations are avoided due to their lack of speed and robustness. This scalability would lead one to believe that `OpenCASCADE` is both thread-safe and scalable. But it is observed that there tends to be a significant amount of code between individual evaluations.

During the analysis of the structured block grids for GMGW1[8] `EGADS` was initially used to remap the grid vertices, which were supposed to be on the geometry, onto the appropriate surface to determine conformity. This required only inverse evaluations of individual points against many of the surfaces in the configuration. It is odd that this phase of the analysis is required at all, but because there are no mesh generators that output the requisite information (Face/surface-patch and parametric coordinate data for each vertex in the patch), the data needs to be regenerated. Note that there is little other code between each inverse evaluation in this analysis.

The entry *hl-crm-gapped-flaps* of Table 2 displays the cost of doing the GMGW1 analysis on the coarse *overset* mesh. This is an extraordinary amount of time, and prompted the desire to improve the performance to do all of the other analyses. An obvious approach is to parallelize the operation by distributing the individual points across the available threads. When done with `EGADS` it was noted that the performance decreased significantly (verses the simple non-threaded application). This is NOT the desired outcome! During the examination of the machine's performance it was noted that about 150% of the CPU was being utilized (out of 800% – 4 cores with 2 hyper-threads each) and most of that was *system* time. The conjecture is that there is some MUTEX deep in the `OpenCASCADE` evaluators to allow for multi-threading (we get the correct answer), but not scalability. That is, each thread needs to wait until the critical portion of the code is available.

EGADSlite is designed to be thread safe without any low-level common code that needs *protection*. When threaded, the machine's appropriate scalability is achieved. These observations has prompted the following actions:

- The GMGW1 analysis for structured grids was performed using EGADSlite with threading. This further decreased the analysis time by the number of available threads on the machine.
- The OpenCASCADE evaluators and inverse evaluators were replaced by those constructed for EGADSlite in EGADS and are available at Rev 1.12 (and higher). This significantly improves the performance of all of ESP.

IV. Distributed Approaches

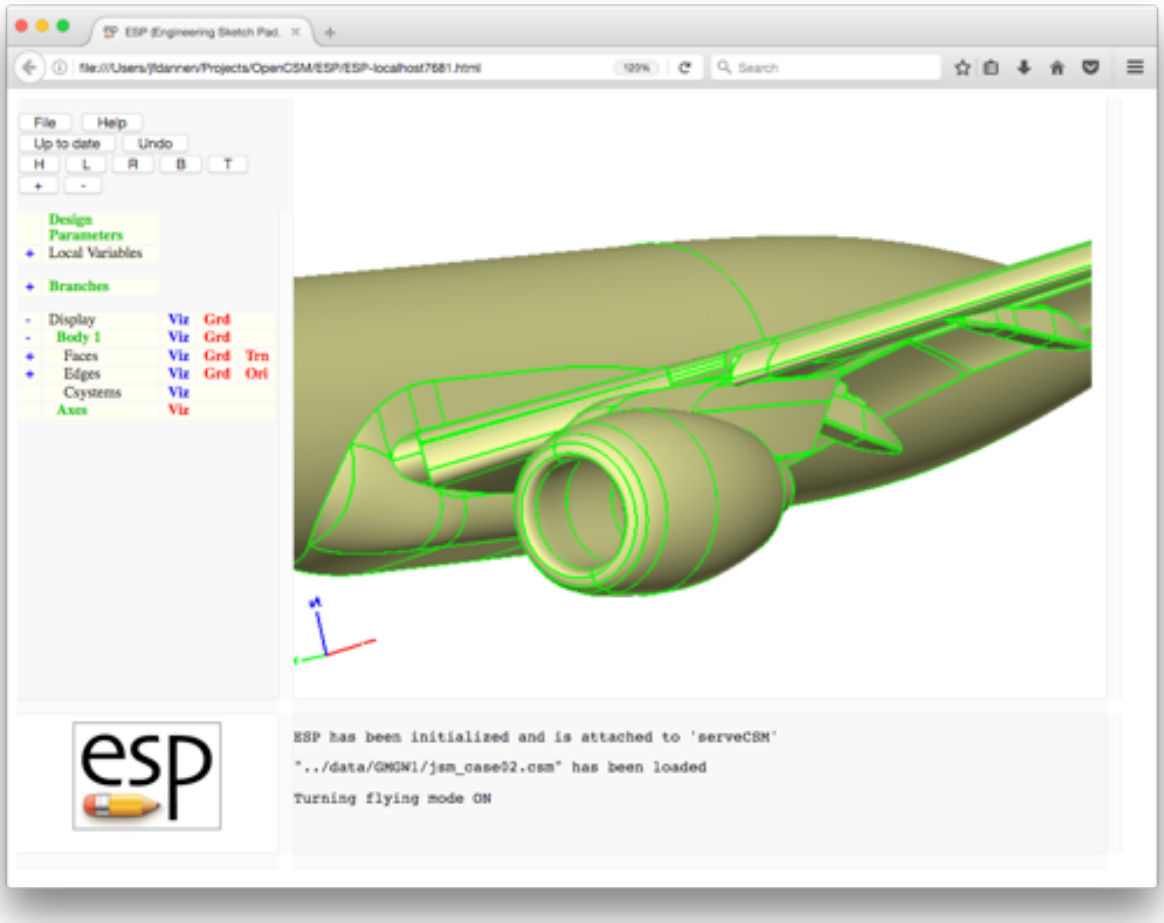

Figure 1. JSM configuration used in partition analysis

A. Simple MPI Implementation

The trivial setup for the distribution of geometric data is to take the EGADS model data in a *master* process (that is, the master process is an EGADS application) and distribute the information to all clients/*slaves*. This single geometry-server/multiple-client arrangement allows for the ability to load, create and modify the

geometry with EGADS and then broadcast the data to the EGADSlite clients to perform the mesh generation/node adaptation/grid curving *in-situ*.

This approach assumes that all computational clients/*slaves* have all of the geometric, topological and attribution data for the complete model available. This means that no further communication to the EGADS *master* is required until there is a change in geometry. But this may place a memory burden on the clients, though it is noted that all of the EGADSlite data is usually much smaller than the surface discretization's memory footprint.

B. Partition Analysis

In an attempt to understand the impacts of further partitioning the geometric data within a spatially decomposed CFD setting an analysis was performed. This partition analysis has been performed to determine how much of the geometry is contained within each of the partitions of a FUN3D flow calculation. With this information, we can assess the best and most appropriate distribution method (if any). The partition analysis was done using an instrumented version of FUN3D, created by NASA LaRC personnel to provided the raw partition data.

For a transport aircraft configured for high-lift operations (the AIAA High-Lift Prediction Workshop 3 JSM test case with nacelle and pylon – Figure 1), only a small portion of the entire configuration needs to be distributed to each processor. The results of this partition analysis is shown in Table 3.

Table 3. Partition analysis results for the JSM with nacelle and pylon in FUN3D

Number partitions	1	10	100	1000	10000
Partition w/o Points	0	0	1	164	5588
Avg Faces/Partition	413	54	10	2	0
Avg Faces/bbox	413	140	30	11	3
Max Faces/Partition	413	102	31	19	13
Max Faces/bbox	413	413	413	413	413

The original idea for distributing portions of the configuration to the geometry clients was to do this based upon the bounding box of the points associated with each partition. The results in the table show that this approach will result in 3–10 times as many Faces distributed as compared with distributing Faces based only on lists of Face indices. The problem with the *list-of-Faces* approach is that the geometry clients need to know the identity of the needed Faces, which is information not currently available in FUN3D – again information that has been lost during the meshing subprocess.

C. Should there be an EGADSair?

The data from Table 3 begs the question: For those memory sensitive HPC application, should the geometry related data be partitioned so that only the needed surfaces/Faces (and the lower hierarchal data) be contained within the client?

For this to work well the following needs to be noted:

Additional communication The required Face indices/objects are not know by the geometry system but are a function of the mesh. This would require a scanning of the mesh partition at client initialization to determine what subset of the Faces are needed. Additional API functions would be needed to transmit the requirements to the EGADS master and then only the preened geometric/topological and attribution data sent back to the requesting client. Obviously this functionality requires more communication and again information about the mesh/geometry association that the solvers don't usually have (and are not available from the grid generation software).

Partition rebalancing If adaptation is a part of the CAE process, the partitions lose their balance and groups of elements are sent around the subdomains in an attempt to rebalance the computational load. This can readjust the list of Faces required by each partition, which would require the local cache of geometric data to be updated. Again another API function may be required and more communication will need to occur.

Limited view of the geometric model This minimal view of the data required at each partition limits the functionality to queries that can only be local (at the Face level or below). For example: if during meshing a newly generated point needs to be compared to the enclosing volume (defined by a *solid*), the entire BRep must be available to perform the predicate.

V. Conclusions

A geometry system for use on HPC equipment, named `EGADSlite`, has been developed and is now a part of the `ESP` ecology of APIs, which is open-source and freely available at `http://acdl.mit.edu/ESP`. When compared with `EGADS` (which used `OpenCASCADE`), `EGADSlite` has been shown to be more robust and generally more than an order of magnitude faster for inverse evaluations (non-threaded). Further, `EGADSlite` is fully threadable and has nearly perfect parallel efficiency, making it ideally suitable for inclusion in HPC environments.

Acknowledgements

The authors thank Eric J. Nielsen of NASA Langley Research Center (LaRC) for his assistance in the `FUN3D` partition analysis.

This research is sponsored by NASA's Transformational Tools and Technologies (TTT) Project of the Transformative Aeronautics Concepts Program under the Aeronautics Research Mission Directorate. William T. Jones (NASA LaRC) is the technical point of contact.

References

[1]Slotnick, J., Khodadoust, A., Alonso, J., Darmofal, D, Gropp, W., Lurie, E., and Mavriplis, D., "CFD Vision 2030 Study: A Path to Revolutionary Computational Aerosciences", NASA/CR-2014-218178, March 2014.

[2]Amdahl, G.M., "Validity of the Single Processor Approach to Achieving Large-Scale Computing Capabilities", AFIPS Conference Proceedings (30): 483-485, 1967.

[3]Haimes, R. and Dannenhoffer, J.F., "The Engineering Sketch Pad: A Solid-Modeling, Feature-Based, Web-Enabled System for Building Parametric Geometry", AIAA-2013-3073, AIAA Computational Fluid Dynamics Conference, San Diego, CA, June 2013.

[4]Dannenhoffer, J.F., "OpenCSM: An Open-Source Constructive Solid Modeler for MDAO", AIAA-2013-0701, 51^{st} AIAA Aerospace Sciences Meeting, Grapevine, TX, January 2013.

[5]Haimes, R., and Drela, M., "On the Construction of Aircraft Conceptual Geometry for High Fidelity Analysis and Design", AIAA-2012-0683, January 2012.

[6]http://www.opencascade.org

[7]http://acdl.mit.edu/ESP/ESP.tgz

[8]Dannenhoffer, J.F., "Analysis of GMGW1 Structured Grids" (invited), To be given at AIAA SciTech 2018, January 2018.