A Parallel Unstructured Mesh Infrastructure

Seegyoung Seol, Cameron W. Smith, Daniel A. Ibanez, Mark S. Shephard
Scientific Computation Research Center,
Rensselaer Polytechnic Institute
Troy, New York 12180
Email: seols@rpi.edu

***Abstract*—Two Department of Energy (DOE) office of Science's Scientific Discovery through Advanced Computing (SciDAC) Frameworks, Algorithms, and Scalable Technologies for Mathematics (FASTMath) software packages, Parallel Unstructured Mesh Infrastructure (PUMI) and Partitioning using Mesh Adjacencies (ParMA), are presented.**

***Index Terms*—unstructured mesh; architecture-awareness; two-level mesh partitioning; threading; partition improvement;**

I. Introduction

Unstructured meshes, often adaptively defined, can yield required levels of accuracy using many fewer degrees of freedom at the cost of more complex parallel data structures and algorithms. The DOE SciDAC FASTMath Institute efforts in this area are focused on the parallel unstructured mesh data structures and services needed by the developers of PDE solution procedures [1]. Current FASTMath efforts are addressing parallel techniques for mesh representations on exascale computers, dynamic load balancing, mesh optimization, mesh adaptation, mesh-to-mesh solution transfer, and a massively parallel unstructured CFD solver. This paper discusses two core components of the FASTMath unstructured mesh developments:

- PUMI a parallel infrastructure with a general unstructured mesh representation and various operations needed for interacting with meshes on massively parallel computers.
- ParMA dynamic load balancing operations through the direct use of mesh adjacency information in PUMI's unstructured mesh representation.

The design of an infrastructure for unstructured meshes on massively parallel computers must consider the mesh information management, the operations to be carried out on the meshes, and the complications introduced in the scalable execution of those operations in parallel. Since the goal is to support the full set of operations needed in a simulation workflow from the generation of the mesh through the post processing of the solution information solved for, it becomes clear that a mesh representations that can effectively provide information on any order mesh entity, as well any adjacencies of mesh entities, is needed. The minimal requirement of such a mesh representation is complete representation with which the complexity of any mesh adjacency interrogation is O(1) (i.e., not a function of mesh size) [2]. Since it is highly desirable to support mesh modification during a simulation workflow, the mesh infrastructure needs to efficiently support dynamic mesh updates.

Given the dominance in the past decade of message passing parallelism, the most common form of parallel mesh decomposition is to partition the mesh into parts such that workload is balanced and the communications between parts is minimized. A second form of decomposition that can be advantageous for intra-process threaded operations using a shared memory is coloring into small independent sets. Although it may be possible to get the highest degree of parallel efficiency on parallel computers with high core count nodes using a combination of the two approaches, the amount of code rewrite needed for applications to take full advantage would be large. In addition, the support of two forms of parallel representation will introduce additional data storage and data manipulations during execution. Thus the current development effort in PUMI is focused on supporting a two level partitioning of the mesh where message passing is used at the node level and threading is used at the core level.

Another key component of supporting unstructured mesh workflows is the ability to apply dynamic load balancing on a regular basis (assuming it can be fast enough). In case of a fixed mesh, this can be driven by the fact that different operations in the workflow can require a different partitioning of the mesh to maximize scaling of that operation. For example, one step in a multi-physics analysis may be using a cell centered FV method where work load balance is based on the mesh regions only, while another step may be using second order FE on the same mesh where vertex and edge balance is more important to scaling than region balance. Of course, the application of operations like mesh adaptation will change the mesh in general ways thus requiring dynamic load balancing before any analysis operation is carried out on the adapted mesh.

II. PUMI

The three data models central to the numerical solution of PDEs on unstructured meshes are the geometric model, the mesh and the fields. The geometric model is the high-level (mesh independent) definition of the domain, typically a non-manifold boundary representation [3]. PUMI interacts with the geometric model through a functional interface that supports the ability to interrogate the geometric model for the adjacencies of the model entities and geometric information about the shape of the entities [2], [4], [5]. The mesh is the discretization of the domain into a form used by the PDE analysis procedures. The fields are tensor quantities that define

978-0-7695-4956-9/13 $26.00 © 2013 IEEE
DOI 10.1109/SC.Companion.2012.135

IEEE computer society

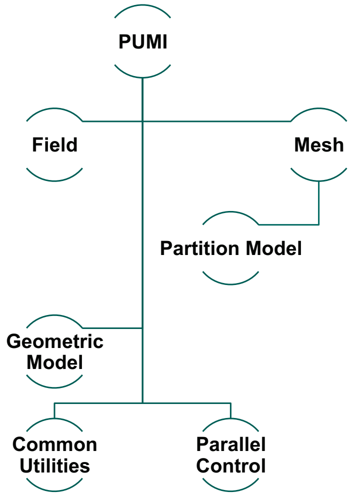


Fig. 1: Software structure of PUMI

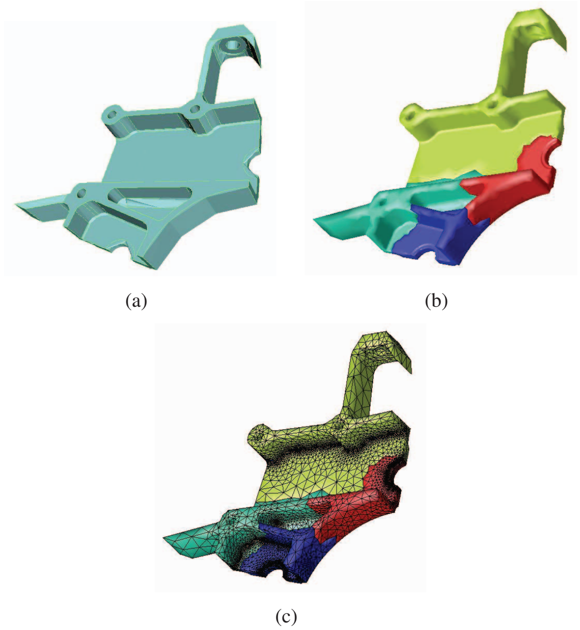
the distributions of the physical parameters of the PDE over domain (mesh and geometric model) entities.

The unstructured mesh representation is typically defined as a boundary representation using the base topological entities of vertex (0D), edge (1D), face (2D), region (3D) and their adjacencies [6], [7], [8], [9], [10]. A mesh entity is uniquely identified by its handle and denoted by M_i^d, where d is dimension ($0 \leq d \leq 3$) and i is an id. Each mesh entity maintains its association to the highest level geometric model entity that it partly represents, referred to as *geometric classification* [7]. The geometric classification is central to the ability to support automated, adaptive simulations. Three common utilities used by both the geometric model and mesh are: (i) Iterator: component for iterating over a range of data, (ii) Set: component for grouping arbitrary data with common set requirements, and (iii) Tag: component for attaching arbitrary user data to arbitrary data or set with common tagging requirements [11], [12], [13].

In addition, adaptive simulations in a parallel computing environment place extra demands to represent and manipulate the distributed mesh data over a large number of processing cores. PUMI supports a topological representation of the distributed mesh and efficient distributed manipulation functions through the use of the partition model [9], [10].

Figure 1 illustrates the software structure of PUMI consisting of the following six components.

- Common utility component is a helper component providing common utilities and services used in multiple other components such as iterator, set, and tag.
- Parallel control component is a helper component providing parallel-specific utilities and services such as communications and architecture-aware operations.
- Geometric model component provides a uniform interface for querying geometric model representations. It uses common utility and parallel control component.
- Partition model component is constructed based on the mesh distribution and provides mesh partitioning representation in topology to the mesh component for the support for efficient update/manipulation of mesh with respect to partitioning.
- Mesh component provides the storage and management of distributed unstructured meshes. It uses all components except for the field component.
- Field component provides the services for storage and management of solution information on the mesh. It uses common utility, parallel control and mesh components.


(a)

(b)

(c)

Fig. 2: Geometric model (a), partition model (b), and mesh distributed to four parts (c)

Figure 2 illustrates an example of a geometric model, a partition model and a mesh distributed to four parts (different colors represent different part). As illustrated, a partition model in the middle represents the mesh partitioning.

A. Part

When a mesh is distributed to N parts, each part is assigned to a process or processing core. A part is a subset of topological mesh entities of the entire mesh, locally identified by a part handle which is a pointer to the part instance, and globally identified by a unique integer id, denoted by P_i, $0 \leq i < N$. Figure 3 depicts a 2D mesh that is distributed to three parts on two processes such that the parts P_0 and P_1 are on process i and the part P_2 is on process j. The dashed line between P_0 and P_1 represents intra-process part boundary and the solid line between P_0 and P_2 represents inter-process part boundary. For part boundary entities which are duplicated on multiple parts, one part is designated as *owning part* and the owning part imbues the right to modify the part boundary entity. In Figure 3, the part boundary vertices and edges on owning part are illustrated with bigger circles and solid lines, respectively.

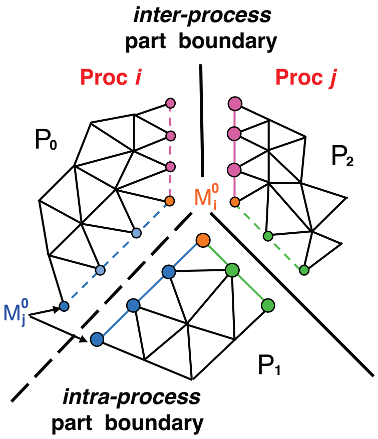


Fig. 3: Three-part distributed 2D mesh on two processes

B. Part Boundaries

Each part is treated as a serial mesh with the addition of mesh part boundaries to describe groups of mesh entities that are on the links between parts. Mesh entities on part boundaries, called *part boundary entities*, are duplicated on all parts for which they bound other higher order mesh entities [1], [9], [10], [14]. Mesh entities that are not on any part boundaries exist on a single part and called *interior* (with respect to the part) mesh entities.

For each mesh entity, the *residence part* [9], [10] is a set of part id(s) where a mesh entity exists based on adjacency information: If mesh entity M_i^d is not adjacent to any higher dimension entities, the residence part of M_i^d is the id of the single part where M_i^d exists. Otherwise, the residence part of M_i^d is the set of part id's of the higher order mesh entities that are adjacent to M_i^d. Note that part boundary entities share the same residence part if their locations with respect to the part boundaries are the same.

In 2D mesh illustrated in Figure 3, the part boundary entities are the vertices and edges that are adjacent to mesh faces on different parts. The residence part of M_i^0 and M_j^0 are $\{P_0$, P_1, $P_2\}$ and $\{P_0$, $P_1\}$, respectively.

C. Partition Model

For the purpose of representation of a partitioned mesh and efficient parallel operations, a partition model is developed [9], [10].

- Partition (model) entity: a topological entity in the partition model, P_i^d, which represents a group of mesh entities of dimension d or less, which have the same residence part. One part is designated as the owning part.
- Partition classification: the unique association of mesh entities to partition model entities.

Figure 4 depicts the partition model of the mesh in Figure 3. The mesh vertex M_i^0, duplicated on three parts, is classified on the partition vertex P_1^0. Other part boundary vertices and edges (like M_j^0) are classified on partition edges. At the mesh entity level, the proper partition classification is a key to maintaining up-to-date residence part and owning part information. Partition classification also enables PUMI to support various capabilities for parallel unstructured mesh modification in an effective manner. These operations include:

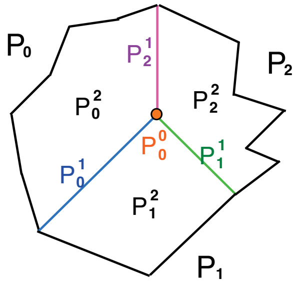


Fig. 4: Partition model of the mesh in Figure 3

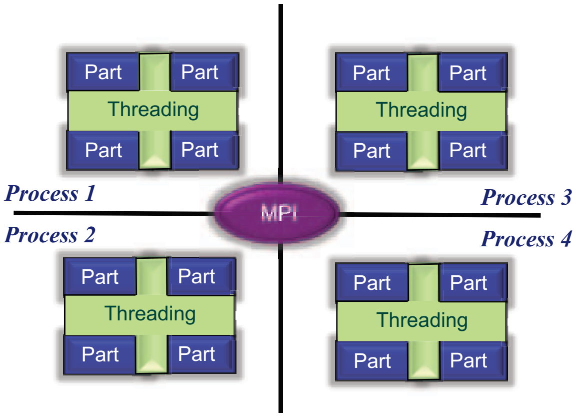
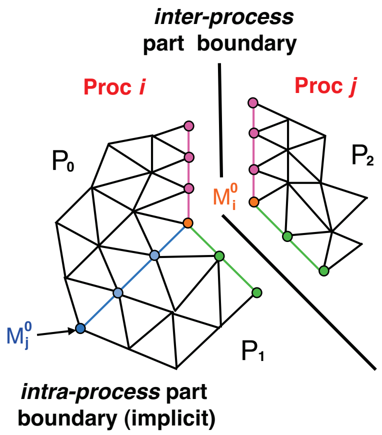


Fig. 5: Two-level mesh partitioning

- Mesh migration: a procedure that moves mesh entities from part to part to support (i) mesh distribution to parts, (ii) mesh load balancing, or (iii) obtaining mesh entities needed for mesh modification operations [10], [15].
- Ghosting: a procedure to localize off-part mesh entities to avoid inter-process communications for computations. A ghost is a read-only, duplicated, off-part internal entity copy including tag data.
- Multiple part per process: a capability to dynamically change the number of parts per process.

D. Two-level, Architecture-aware Mesh Partitioning

Recent supercomputers such as the Blue Gene/Q and Cray XE6, characterized by their hybrid shared and distributed memory architecture and fast communication networks [16], [17], offer new opportunities for improving performance by fully exploiting the process-level shared memory to reduce the total memory usage and communication time. Unstructured mesh infrastructure on modern computer systems requires two-level mesh partitioning with architecture awareness, thread management, efficient message passing, intra-proces and inter-process boundary entity differentiation, and intra-



inter-process part boundary
Proc *i*
Proc *j*
P₀
P₂
M_i^0
M_j^0
P₁
intra-process part boundary (implicit)

Fig. 6: Architecture-aware mesh partitioning

process shared entity access control. Architecture awareness supports mapping each MPI process to the largest hardware entity whose memory is shared (usually called a node) and each thread to the smallest hardware entity capable of independent computation (processing unit). The first mapping maximizes usable shared memory and the second mapping maximizes the use of processing resources.

In implementation, for effective manipulation of multiple parts per process, a mesh instance is created on each process and then the mesh instance maintains a list of part handles created on the process. A part handle maintains a set of mesh entity handles, pointers to the mesh entity data, per dimension. Regardless of its status (internal or part boundary), a mesh entity is created at most once on every *residence process*, which is a set of MPI process ranks where a mesh entity exists based on adjacency information. For an inter-process part boundary entity, one or more memory addresses of the duplicate copies on remote processes are stored as remote copies. The remote copy data structure is a pair of MPI rank and mesh entity handle on a remote process. Note that only the owning part can modify the entity (internal or on part boundary). Creation of a part boundary mesh entity, e.g. during mesh loading or migration, on a process i which is owned by a part in process j requires the following steps:

- On process i, a part handle with minimum part id creates the entity and the entity stores the entity copy address on process j as a remote copy.
- On process j, the owning part creates the entity and stores the entity copy address on process i as a remote copy.

Applying owning and minimum part id rule in entity modification and creation, inherently, the entity management on hybrid system is free of race condition.

Figure 5 depicts the two-level mesh partitioning in which communications are done through MPI message passing between remote parts and inter-thread message passing between local parts. Figure 6 illustrates architecture-aware mesh partitioning which differentiates intra-process and inter-process part boundaries. An intra-process part boundary entity (e.g. M_j^0) is shared by every local residence part, as such intra-process part boundaries do not explicitly exist. Whereas, an inter-process part boundary entity (e.g. M_i^0) is duplicated on every residence process.

In order to support architecture-aware, two-level parallelism effectively, parallel control functions are implemented as a library called Parallel Control Utilities (PCU). PCU functionalities include:

- Architecture topology detection: computing hardware information is obtained using `hwloc` [18]
- Message passing abstraction over shared memory and MPI-based message passing: in order to pass messages, applications (including the mesh component in PUMI) are presented with an API based on integer part ids, and then the PCU delegates to the appropriate messaging method (threading or MPI) based on the mapping of part handles to software threads.
- Creation and management of software threads within a process: PCU informs each running thread of its ID and the total number of threads in the job.
- Message passing control: message buffer management and message routing by hardware topology and neighboring process recognition (A process i neighbors process j over entity type d if they have d dimensional mesh entities on inter-process part boundary).
- Message round termination detection: most of the communication algorithms in PUMI are implemented as a series of collective message passing rounds. There are two algorithms for detecting the termination of such a round, depending on whether communication is confined to known neighborhoods. If this is the case, a local algorithm is used whose runtime is proportional to neighborhood size. When neighborhoods are not known, a non-blocking barrier is used to detect termination of the round. The latter algorithm has a runtime logarithmic over the total number of processes.

The MPI-only version of PUMI with the mesh representation in Figure 3 is being used on a number of adaptive simulation applications. Figure 7 illustrates (a) initial mesh (b) close view of initial mesh, (c) adapted mesh, and (d) close view of adapted mesh for a supersonic flow past a scramjet [19]. Figure 8 illustrates three adapted meshes tracking the motion of particles through a linear accelerator [20]. The MPI-only version has been used on meshes of billions of elements up to 512K cores of Blue Gene/Q (K represents thousand). PUMI's mesh representation is under improvement toward architecture-aware, two-level mesh partitioning with the PCU. With hybrid multi-threaded/MPI version, an initial, experimental test of mesh loading on a Blue Gene/Q demonstrated faster communications than the MPI-only version.

III. ParMA

Unstructured distributed meshes in parallel adaptive simulation workflows need to satisfy varying requirements. Partitions for mesh adaptation require, at a minimum, that the resulting adapted mesh fits within memory, while partitions for PDE

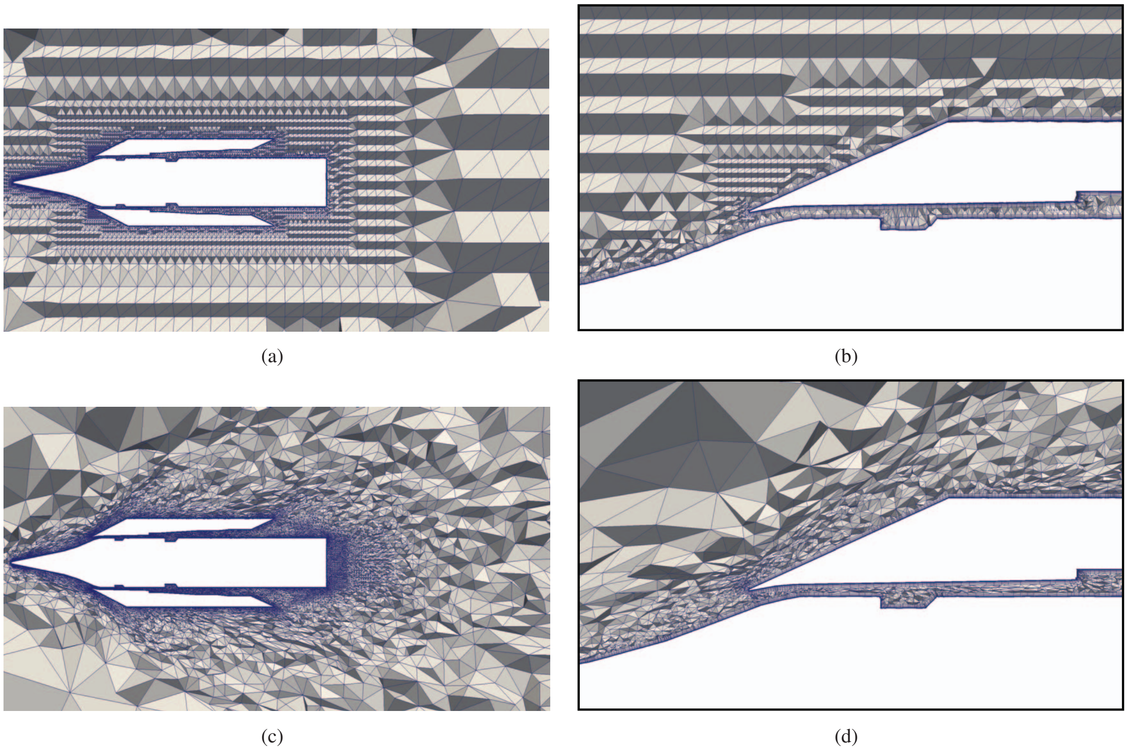


Fig. 7: Initial and adapted mesh on scramjet model

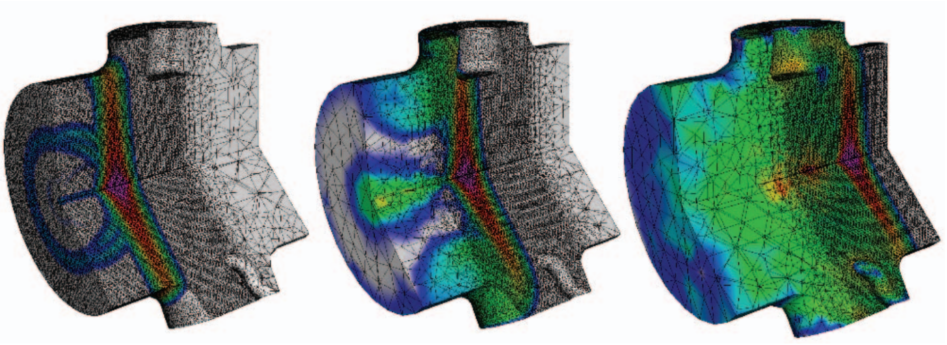


Fig. 8: Three adapted meshes on particle model

analysis require accounting for the balance of multiple entity types, those associated with degrees of freedom, to maximize scaling. In both cases peaks determine performance; valleys may leave a process idle or consuming very little memory, while peaks will leave the *majority* of processes idle or exhaust available memory. Therefore, reduction of peaks for each step in a workflow is critical for efficient performance.

A partitioning algorithm capable of reducing imbalance peaks must also account for the connectivity of the unstructured mesh such that the part boundaries are optimized. As was described in section II-B, the distributed representation of the mesh among processes creates duplicated mesh entities along the part boundaries. Thus, the amount of communications across partition model boundaries will increase as the part boundary gets 'rougher', an increase in boundary surface area in 3D, or length in 2D. The most powerful parallel unstructured mesh partitioning procedures are the graph and hypergraph-based methods as they can explicitly account for application defined imbalance criteria via graph node weights, and one piece of the mesh connectivity information via the definition of graph edges. Hypergraph-based methods can further optimize the partition boundaries at the cost of increased run-time over the graph-based methods [21]. Faster partition computation is available through geometric methods, and for certain applications are desirable. However, as they do not account for mesh connectivity information, the quality of partition boundaries can be poor. The problems observed in the partitions produced by the graph and hypergraph-based methods are limited typically to a small number of heavily loaded parts, referred to as spikes; scalability of applications is then reduced by these spikes.

ParMA, partitioning using mesh adjacencies, provides fast partitioning procedures for adaptive simulation workflows that work independently of, or in conjunction with, the graph/hypergraph-based procedures. ParMA procedures use constant time mesh adjacency queries provided by a complete mesh representation, and partition model information, to determine how much load must be migrated, the migration schedule, and which elements need to be migrated to satisfy that load, element selection. As graph/hypergraph-based use only a subset of the information provided by mesh adjacencies and the partition model, decisions can be made in the schedule and selection processes that can better satisfy the applications partition requirements. A key challenge is developing and adopting efficient algorithms required for partitioning without the classical graph data structure. Two ParMA procedures for partitioning are presented: multi-criteria partition improvement and heavy part splitting.

A. *Multi-criteria Partition Improvement*

ParMA multi-criteria partition improvement procedures take as input a partition with moderate imbalance spikes, and efficiently reduce them to the application specified level while meeting the partition requirements of the application. Efficient reduction of vertex imbalance spikes, with acceptable increases in element balance, for PHASTA CFD analysis with greedy diffusive procedures is demonstrated by Zhou in [21] and [22]. An extension to this work supports multi-criteria greedy partition improvement. An application executing the multi-criteria partition improvement procedure provides a priority list of mesh entity types to be balanced such that the imbalance of higher priority entity types is not increased while balancing a lower priority type.

The ParMA partition improvement procedure traverses the priority list in order of decreasing priority. For each mesh entity type the migration schedule is computed, regions are selected for migration, and the regions are migrated. These three steps form one iteration. When the application defined imbalance is achieved, or the maximum number of iterations is reached, the next mesh entity type is processed. If multiple mesh entity types share equal priority then those entities are traversed in order of increasing topological dimension.

For example, if the application specifies $Rgn > Face = Edge > Vtx$, there are two “>”, yielding three levels of priority. ParMA starts with the first level and improves the balance for mesh regions. Since mesh regions have the highest priority, the algorithm will do the migration for better region balance even if it may make the balance of other mesh entities (faces, edges and vertices) worse. After the iterations of region balance improvement, the algorithm moves to the second level, where the mesh faces and edges have the equal importance. During face balance improvement mesh elements are migrated if and only if they reduce the face imbalance and do not harm the balance for mesh regions and edges. The procedure for edge balance improvement is similar. When ParMA finishes the iterations for the second level it starts the iterations for the last level, vertex imbalance improvement. Since the vertex balance has the lowest priority, migration occurs if and only if it reduces the vertex imbalance with no harm to the balance of any other types of mesh entities.

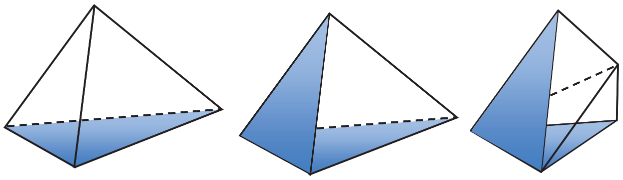


Fig. 9: Select mesh regions for migration that have more faces classified on the part boundary (translucent) then faces classified on the part interior (blue).

1) Candidate Parts: The ParMA algorithm reduces entity imbalance by migrating a small number of mesh elements from heavily loaded parts to the lightly loaded neighboring parts, which are called candidate parts. There are two categories for candidate parts: *absolutely* lightly loaded, and *relatively* lightly loaded. An absolutely lightly loaded part is one which either has fewer number of entities then the average across all parts, or has less then the application defined threshold for spikes. A relatively lightly loaded part is one which has fewer entities then a neighboring heavily loaded part. A candidate part must be lightly loaded, either absolutely or relatively, for all lesser priority mesh entity types then the mesh entity type being balanced. These categories of candidate parts improve the ability of the imbalance spikes to be diffused throughout the partition.

2) Mesh Element Selection: Mesh elements, and groups of mesh elements, referred to as cavities, are selected for migration if they will decrease the communication cost over part boundaries once migrated. To improve region balance, ParMA traverses the mesh faces classified on the part boundaries searching for those regions that have more faces classified on the part boundary then on the part interior. Figure 9 depicts three such examples of mesh regions that will be selected.

To improve the edge balance, ParMA traverses the mesh edges classified on the part boundary searching for those edges that bound fewer then two or three mesh faces. When such an edge is found the mesh elements bounded by both the mesh faces and the edge form a cavity selected for migration. This selection improves the edge balance with minimum effect to other types of mesh entities and usually improves the part boundary. Figure 10 gives an example in which edge M_0^1 has two faces (in (a)) and three faces (in (b)) classified on part P_0. In both cases, suppose P_0 is heavily loaded with edges, P_1 is a candidate part, and edge M_0^1 is classified on the part boundary of P_0 and P_1. In Figure 10 (a), edge M_0^1 bounds only two faces on P_0 and has only one adjacent region. In order to reduce the number of edges on P_0 the region adjacent to M_0^1 is migrated from P_0 to P_1. This migration only increases the number of region on P_1 by one and does not increase the number of faces and edges on part boundaries. In Figure 10 (b), edge M_0^1 bounds three faces and two regions on P_0. In order to remove M_0^1 from the part boundary between P_0 to P_1 both regions adjacent to M_0^1 have to be migrated from P_0 to P_1. The number of regions classified on P_1 is increased by two. The number of faces and edges classified on the part boundary increases by two and three, respectively. Migration of edges which bound more faces can significantly increase the number of mesh entities on the part boundary. The strategy of ParMA to find candidate vertices to improve the vertex balance is same as that developed by Zhou in [21].

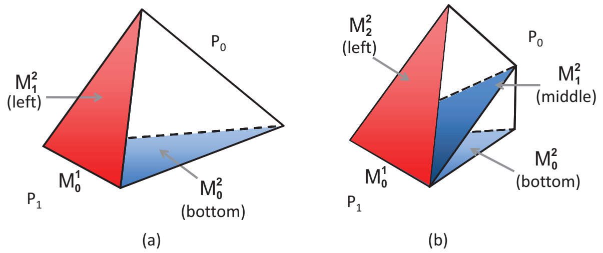


Fig. 10: An edge bounds two or three faces on part P_0

3) Multi-criteria Partitioning Tests: Multi-criteria partition improvement tests are performed on an abdominal aorta aneurysm (AAA) model depicted in Figure 11. A mesh with 133M tetrahedral elements (regions) is considered. Table I presents the tests done in this section. The first column labels the tests for referencing into Tables II and **??**. The first test, T_0, is the parallel partitioning with Zoltan Parallel Hypergraph partitioner PHG [23] to 16,384 parts. The remaining tests, T_{1-4}, apply ParMA multi-criteria partition improvement on the partition created in T_0 with the user inputs given in column two.

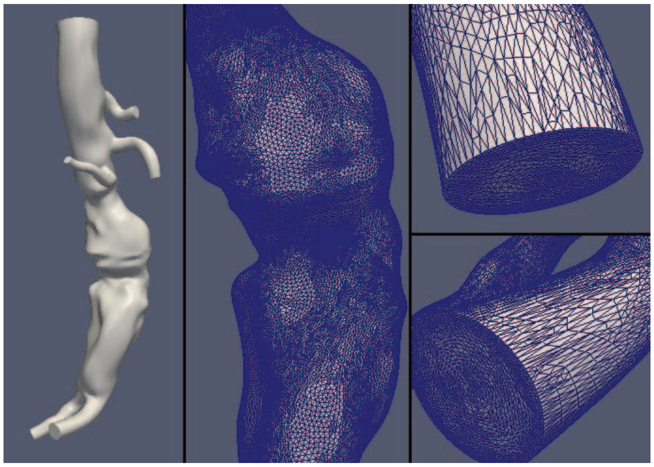


Fig. 11: Geometry and mesh of an AAA model

Table II presents the imbalance of each type of mesh entities for tests listed in Table I. The tests are performed on 512 cores on Jaguar (Cray XT5) at NCCS with the software's capability to handle multiple parts (32 parts) per process.

TABLE I: Tests and parameters for the partition improvement algorithms

Test	Method
T_0	Zoltan's Hypergraph
T_1	ParMA $Vtx > Rgn$
T_2	ParMA $Vtx = Edge > Rgn$
T_3	ParMA $Edge > Rgn$
T_4	ParMA $Edge = Face > Rgn$

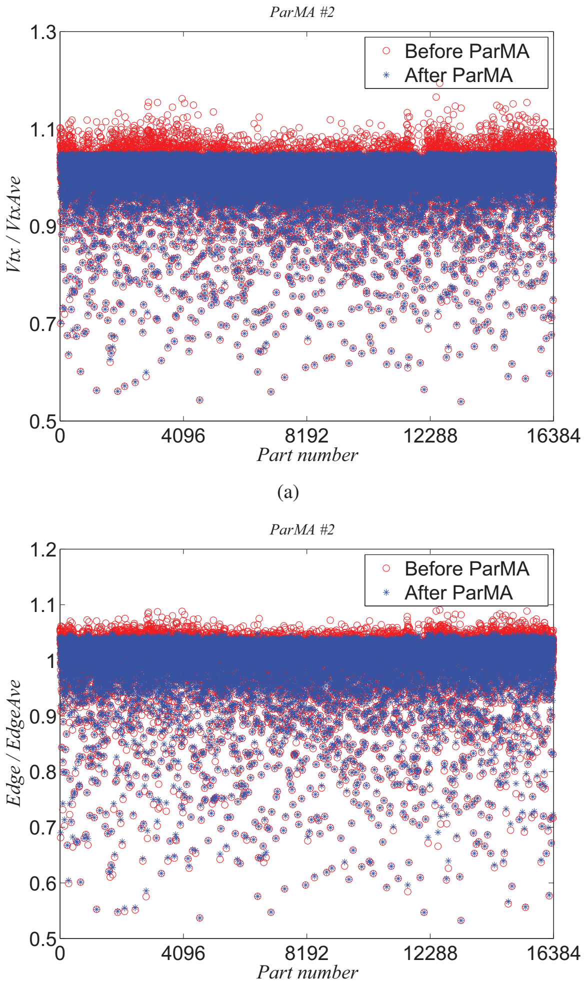
TABLE II: ParMA algorithm on a partition with 133M element mesh on an AAA model

AAA 133M	T_0	T_1	T_2	T_3	T_4
MeanRgn	8,177	8,177	8,177	8,177	8,177
Rgn Imb.%	4.3	4.99	5.99	5.98	5.93
MeanFace	17,315	-	-	-	17,309
Face Imb.%	5.39	-	-	-	4.97
MeanEdge	11,023	-	10,973	11,013	11,014
Edge Imb.%	9.07	-	4.91	4.99	4.99
MeanVtx	1,886	1,865	1,870	-	-
Vtx Imb.%	19.41	4.99	4.99	-	-

For comparison purposes, the imbalance ratios are all computed based on the mean values of the partition created in T_0, the 1st column of Table II. Test T_1 balances both regions and vertices where vertices are of higher priority then regions. ParMA reduces the vertex and region imbalance below the prescribed 5% tolerance. Furthermore, the average number of vertices after T_1 is 1,865 which is 1% smaller than the original value of 1,886. This is due to the careful selection of mesh entities to be migrated while improving the balance. Test T_2 improves the balance of regions, edges and vertices with the priority of vertices equal to edges, and larger then regions. ParMA reduces the balance for all three types of mesh entities below the prescribed tolerance. In tests T_3 and T_4, the balance of specified types of mesh entities satisfies the tolerance as well. Figure 12 demonstrates the normalized number of mesh vertices (left) and edges (right) on each part before and after the partition improvement in test T_2. In all four ParMA tests, the total number of mesh entities on part boundaries are reduced compared to the original partition, which reduces the communication load in the analysis.

The time usage of each test is presented in Table **??**. The time usage of ParMA partition improvement is much smaller than that of hypergraph method and it is negligible compared to the analysis.

Application of the iterative diffusive procedure is currently being tested on a 3 billion element mesh partitioned up to 1.5M parts for PHASTA CFD on Mira Blue Gene/Q. Initial tests specifying $Vtx > Rgn$ on the 1.5M part mesh improve vertex imbalance by more then 10%. This partition is created by locally partitioning each part of a 16,384 part mesh with Zoltan Hypergraph to 96 parts. The initial peak vertex imbalance of the 1.5M part mesh is 54% while the initial peak vertex imbalance of the 16,384 part mesh is 9%.



(a)

(b)

Fig. 12: Normalized number of vertices (a) and edges (b) on each part before and after the partition improvement in ParMA test T_2.

B. Heavy Part Splitting

The greedy iterative diffusive procedure presented in Section III-A is observed to not meet a target imbalance tolerance when the input partition is large and has multiple parts with the imbalance spikes neighboring each other. In [21] Zhou observed such an imbalance during local partitioning with greater then 80 parts per-process. Large entity imbalances are also likely in partitions with tens of thousands of parts where each part has fewer then two of three thousand mesh elements. For an extreme example of this, consider a 3 billion tetrahedron mesh partitioned to 1.5M parts. In this partition the average number of vertices per part is 576 and a 56% vertex imbalance occurs if one part has only 324 more vertices then the average. 324 vertices is approximately one 10 million'th of the total number of vertices. Given the sensitivity of each part to increases in entity count an approach is required that is more directed, and aggressive, then iterative diffusion.

Large imbalance spikes are also observed when predictively load balancing for mesh adaptation based on the estimated target mesh resolution at each mesh vertex. These spikes are common to analysis driven adaptation in which the regions of high mesh resolution change to capture some physical phenomena of interest, such as a phasic interface, or a shock front; some parts request large amounts of refinement while others request large amounts of coarsening. For example, consider a super-sonic viscous flow analysis on an ONERA M6 wing in which a shock front is resolved with a size field computed from the hessian of the mach number. Figure 13 depicts a histogram of element imbalance in an adapted mesh if no load balancing is applied prior to adaptation. Here the peak imbalance is over 400% and approximately 80 parts have an imbalance over 20%. Imbalanced parts cannot exist without reducing the number of elements in other parts an equivalent amount. As such, there are over 120 parts that have fewer then 50% of the average number of mesh elements.

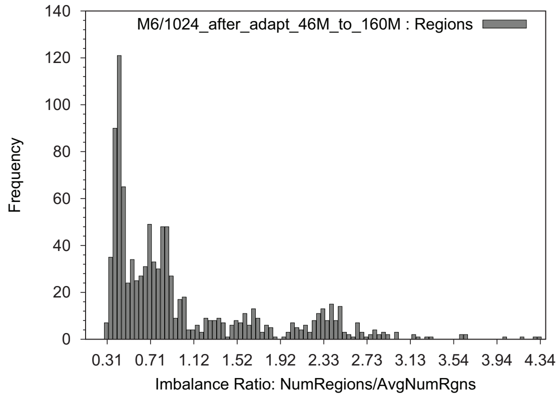


Fig. 13: Histogram of element imbalance (number of elements / average number of elements) in 1024 part adapted mesh on Onera M6 wing if no load balancing is applied prior to adaptation.

One possible approach to quickly reduce multiple large imbalance spikes is heavy part splitting. ParMA heavy part splitting reduces imbalance spikes by first merging lightly loaded parts to create empty parts, and then splitting heavily loaded parts into the newly created empty parts. The procedure begins by independently solving the 0-1 knapsack problem [24] on each part to determine the largest set of neighboring parts which can be merged while keeping the number of total number of elements less then the average. Next, a set of these merges that can be performed without conflicts, i.e. a part is merged only once, are found by solving for the maximal independent set. Lastly, heavily loaded parts are split as many times as required until there are either no heavy parts or empty parts remaining. As needed, heavy part splitting is followed by iterative partition improvement described in Section III-A.

IV. Closing Remarks

Two DOE SciDAC FASTMath software packages, PUMI and ParMA, are presented. The existing MPI-based PUMI demonstrated its effectiveness taking meshes of billions of elements from a few thousand parts to 1.5 million parts for ParMA and parallel adaptive simulations running on 512K cores of Blue Gene/Q (Mira). The current development includes two-level, architecture-aware mesh partitioning and various parallel control functionalities running on modern massively parallel computers.

PUMI and ParMA are open source and available at *http://www.scorec.rpi.edu/software.php*.

Acknowledgment

The authors would like to thank Alex Ovcharenko for his developments of PUMI message passing control, Min Zhou for her developments of ParMA multi-criteria mesh partition improvement, Michel Rasquin at University of Colorado at Boulder for Mira test data, and RPI Computer Science undergraduates Micah Corah and Ian Dunn for their development and testing efforts towards the hybrid PUMI implementation.

The authors also gratefully acknowledge the support of this work by the Department of Energy (DOE) office of Science's Scientific Discovery through Advanced Computing (SciDAC) institute as part of the Frameworks, Algorithms, and Scalable Technologies for Mathematics (FASTMath) program, under grant DE-SC0006617.

References

[1] D. of Energy's Scientific Discovery through Advanced Computing (SciDAC), "Frameworks, algorithms, and scalable technologies for mathematics (fastmath)," retrieved on October 1, 2012, from http://www.fastmath-scidac.org/.

[2] M. W. Beall, J. Walsh, and M. S. Shephard, "Accessing cad geometry for mesh generation," in *12th International Meshing Roundtable, Sandia National Laboratories*, 2003, pp. 2003–3030.

[3] K. J. Weiler, "The radial-edge structure: a topological representation for non-manifold geometric boundary representations," *Geometric Modeling for CAD Applications*, pp. 3–36, 1988.

[4] M. S. Shephard and M. K. Georges, "Reliability of automatic 3D mesh generation," *Computer Methods in Applied Mechanics and Engineering*, vol. 101, no. 1-3, pp. 443–462, December 1992.

[5] X. Li, M. S. Shephard, and M. W. Beall, "Accounting for curved domains in mesh adaptation," *International Journal for Numerical Methods in Engineering*, vol. 58, no. 2, pp. 247–276, 2003.

[6] K. K. Chand, L. F. Diachin, X. Li, C. Ollivier-Gooch, E. S. Seol, M. S. Shephard, T. J. Tautges, and H. Trease, "Toward interoperable mesh, geometry and field components for PDE simulation development," *Engineering with Computers*, vol. 24, no. 2, pp. 165–182, Nov. 2007.

[7] M. W. Beall and M. S. Shephard, "A general topology-based mesh data structure," *International Journal for Numerical Methods in Engineering*, vol. 40, no. 9, pp. 1573–1596, 1997.

[8] J. F. Remacle and M. S. Shephard, "An algorithm-oriented mesh database," *International Journal for Numerical Methods in Engineering*, vol. 58, no. 2, pp. 349–374, Sep. 2003. [Online]. Available: http://doi.wiley.com/10.1002/nme.774

[9] E. S. Seol, "FMDB: Flexible distributed Mesh DataBase for parallel automated adaptive analysis," Ph.D. dissertation, Rensselaer Polytechnic Institute, 2005. [Online]. Available: http://www.scorec.rpi.edu/reports/view_report.php?id=368

[10] E. S. Seol and M. S. Shephard, "Efficient distributed mesh data structure for parallel automated adaptive analysis," *Engineering with Computers*, vol. 22, no. 3-4, pp. 197–213, Nov. 2006.

[11] D. of Energy's Scientific Discovery through Advanced Computing (SciDAC), "Interoperable technologies for advanced petascale simulations (itaps)," retrieved on October 1, 2012, from http://www.itaps.org/.

[12] C. Ollivier-Gooch, L. F. Diachin, M. S. Shephard, T. J. Tautges, J. A. Kraftcheck, V. Leung, X. Luo, and M. Miller, "An interoperable, data-structure-neutral component for mesh query and manipulation," *ACM Transactions on Mathematical Software (TOMS)*, vol. 37, no. 3, 2010.

[13] T. J. Tautges, R. Meyers, K. Merkley, C. Stimpson, and C. Ernst, "MOAB: A Mesh Oriented Database (Sandia report)," Tech. Rep., 2004.

[14] J. E. Flaherty, R. M. Loy, C. Ozturan, M. S. Shephard, B. K. Szymanski, J. D. Teresco, and L. H. Ziantz, "Parallel structures and dynamic load balancing for adaptive finite element computation," *Applied Numerical Mathematics*, vol. 26, no. 1-2, pp. 241–263, Jan. 1998. [Online]. Available: http://linkinghub.elsevier.com/retrieve/pii/S0168927497000949

[15] F. Alauzet, X. Li, E. S. Seol, and M. S. Shephard, "Parallel anisotropic 3D mesh adaptation by mesh modification," *Engineering with Computers*, vol. 21, no. 3, pp. 247–258, Jan. 2006. [Online]. Available: http://www.springerlink.com/index/10.1007/s00366-005-0009-3

[16] IBM Inc., "Ibm system blue gene solution: Blue gene/q application development," retrieved on October 1, 2012, from http://www.redbooks.ibm.com/abstracts/sg247948.html?Open.

[17] Cray Inc., "Cray xe6 system," retrieved on October 1, 2012, from http://www.cray.com/Products/XE/CrayXE6System.aspx.

[18] O. M. Team, http://www.open-mpi.org/projects/hwloc/.

[19] A. Ovcharenko, K. Chitale, O. Sahni, K. E. Jansen, and M. S. Shephard, "Parallel adaptive boundary layer meshing for CFD analysis," *International Meshing Round Table, in press*, 2012.

[20] X. Luo, M. S. Shephard, L. Lee, L. Ge, and C. Ng, "Moving curved mesh adaptation for higher-order finite element simulations," *Engineering with Computers*, vol. 27, 2011, dOI:10.1007/s00366-010-0179-5.

[21] M. Zhou, O. Sahni, K. D. Devine, M. S. Shephard, and K. E. Jansen, "Controlling unstructured mesh partitions for massively parallel simulations," *SIAM J. SCI. COMPUT.*, vol. 32, no. 6, pp. 3201–3227, 2010. [Online]. Available: http://link.aip.org/link/?SJOCE3/32/3201/1

[22] M. Zhou, O. Sahni, T. Xie, M. S. Shephard, and K. E. Jansen, "Unstructured mesh partition improvement for implicit finite element at extreme scale," *Journal of Supercomputing*, vol. 38, no. 3, pp. 140–156, 2012, dOI:10.1007/s11227-010-0521-0.

[23] E. G. Boman, K. D. Devine, L. A. Fisk, R. Heaphy, B. Hendrickson, V. Leung, C. Vaughan, U. Catalyurek, D. Bozdag, , and W. Mitchell, "Zoltan home page," September 2011, http://www.cs.sandia.gov/Zoltan.

[24] H. Kellerer, U. Pferschy, and D. Pisinger, *Knapsack Problems*. Springer, 2004. [Online]. Available: http://books.google.com/books?id=u5DB7gck08YC