APPLICATION SPECIFIC MESH PARTITION IMPROVEMENT*

CAMERON W. SMITH†, MICHEL RASQUIN‡§, DAN IBANEZ†, KENNETH E. JANSEN§, AND MARK S. SHEPHARD†

Abstract. Partitioning unstructured meshes for parallel adaptive workflows running on leadership class systems requires the application of scalable partitioning tools that meet the needs of each workflow step. The combination of the (hyper)graph methods with two criteria partition improvement to hundreds of thousands of cores was previously presented [23, 51]. This work generalizes the partition improvement to multiple criteria and presents results to more than one million cores of Mira BlueGene/Q on meshes with over 12 billion elements.

Key words. partition improvement, graph/hypergraph, unstructured mesh, dynamic load balancing

AMS subject classifications. 65Y05, 68W10

1. Introduction.

Automated parallel simulation based engineering workflows operating on unstructured meshes require adaptive methods to ensure reliability and efficiency [42]. Given a model based problem definition [30, 41] an effective workflow automatically executes parallel mesh generation [45], analysis, and analysis-based mesh [7, 35] and/or model [33] adaptation. The analyze-adapt cycle is repeated until a desired level of solution accuracy is reached. Between, and within, each step is an opportunity to improve scalability and efficiency through dynamic partitioning.

Current dynamic load balancing methods do not effectively reduce imbalances to the levels needed by applications capable of strong scaling to the full size of leadership class petascale systems. This paper presents a scalable hybrid approach that quickly reaches the required imbalance levels for multiple criteria by pairing ParMA, Partitioning using Mesh Adjacencies, with current partitioning methods.

Section 2 reviews (hyper)graph, recursive sectioning geometric, and diffusive partitioning methods. Section 3 describes how the hybrid ParMA approach meets the requirements of dynamic partitioning. Section 4 reviews the partitioned mesh representation in the Parallel Unstructured Mesh Infrastructure (PUMI) [21, 40]. Sections 5 through 7 detail partition improvement procedures to support application specific partition requirements. Section 8 presents ParMA feature comparison tests, multi-criteria partitioning results on meshes with over 12 billion elements, and the scaling improvement of a CFD analysis on up to one million cores. Section 9 concludes the paper.

2. Unstructured Mesh Partitioning.

*Submitted to the journal's Software and High-Performance Computing section DATE; accepted for publication (in revised form) DATE, published electronically DATE. This research was supported U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research, under award DE-SC00066117 (FASTMath SciDAC Institute). We gratefully acknowledge the use of the resources of the Rensselaer Center for Computational Innovations and the Leadership Computing Facility at Argonne National Laboratory.

†SCOREC, Rensselaer Polytechnic Institute, 110 8th St., Troy, NY 12180 (smithc11@rpi.edu, ibaned@rpi.edu, shephard@rpi.edu)

‡Leadership Computing Facility, Argonne National Laboratory, 9700 South Cass Avenue, Bldg. 240, Argonne, IL 60439-4847

§University of Colorado Boulder, 1111 Engineering Dr, ECOT 126 Boulder, CO 80309 (michel.rasquin@colorado.edu, kenneth.jansen@colorado.edu)

2.1. (Hyper)Graph Partitioning. Graph-based partitioning methods define an assignment of weighted graph vertices to k parts such that each part has the same total weight and the inter part communication costs are minimized. A graph node represents a work unit and an edge between two vertices represents an information dependency. A graph is constructed from an unstructured mesh by selecting one type of mesh entity and mesh adjacencies between them to define graph nodes and edges. Although there are 12 different pairs of entity dimensions and first order adjacency relations that can define the graph, the dominant definition selects mesh elements for graph nodes and faces (edges) shared by elements in a 3D (2D) mesh for graph edges. The unique assignment of each mesh element to a part supports local element level computations that are typical of mesh-based analysis methods [19].

Parallel implementations of multi-level graph algorithms [24] have been demonstrated to support creating high quality partitions with tens of thousands of parts in a fraction of the time needed by most analysis procedures [26, 28, 39]. An extension of the graph-based partitioning methods are hypergraph methods. A hypergraph is defined as a set of weighted vertices and hyperedges. Hyperedges differ from graph edges in that they represent dependencies between multiple graph vertices and, in doing so, have the ability to better model the communication costs of an application [5, 6]. As with graph-based partitioning, the goal of hypergraph partitioning is to balance the vertex weight across the k parts while minimizing an hyperedge-based objective function. Boman and Devine propose constructing the hypergraph from an unstructured mesh by creating one hypergraph vertex for each mesh element, as is done in the graph-based construction, and a hypergraph edge connecting the elements bounded by each mesh vertex. As with graph construction, there are multiple options for defining which mesh entities and which entity adjacencies define the hypergraph, but the above definition is the most common for element-based analysis methods. Given the richer model, hypergraph methods are more computationally intensive in terms of operations and memory usage as compared to graph methods.

2.2. Geometric Partitioning. Geometric based recursive sectioning methods can quickly compute well-balanced partitions for a single entity type [3, 11, 36, 44, 50]. These methods represent information via spatial coordinates and their relations via the distance between them; the closer two pieces of information are the stronger their relation. Recursive coordinate [2], RCB, and inertial bisection [44, 50], RIB, methods recursively cut the parent domain; the former along a coordinate axis and the latter perpendicular to the parent domain's principal direction. Multi-sectioning techniques [11, 36] can be considered extensions of the recursive coordinate bisection methods as they define cuts along coordinate axis, but do so with multiple parallel cut planes at each recursion.

2.3. Diffusive Partitioning. Diffusive partitioning methods efficiently improve an existing partition by the transfer of load between adjacent parts. Transfers can be globally coordinated by computing a diffusive solution that minimizes either the total weight of transferred elements, or the maximum weight transferred in to or out from a part [17, 18, 29, 31, 37, 38, 46]. Alternatively, load can be iteratively transferred from heavily loaded to less loaded parts [10, 37, 49]. The latter approach can have significantly lower overall computational costs if the amount of transferred load and number of iterators are controlled. Careful selection of elements for migration between parts to satisfy the load transfer is typically through a heuristic that measures the improvement, or gain, in part quality [14, 27]. Zhou [23, 51] has shown these methods to be highly scalable given a sufficient distributed mesh representation.

3. Dynamic Partitioning Requirements. Relative to static partitioning, dynamic partitioning has more requirements. Hendrickson and Devine [16] define these requirements as: (1) balance the computational work, (2) reduce the inter processor communication costs, (3) modify the partition incrementally, (4) output the new communication pattern, (5) execute on parallel systems quickly, (6) consume small amounts of memory, and (7) provide an easy to use functional interface. Of the partitioning methods applicable to unstructured meshes [13], multi-level graph and hypergraph-based methods most effectively meet the requirements of dynamic partitioners on up to several thousand processes. Recursive sectioning geometric methods also meet the requirements at a lower computational cost than the (hyper)graph based methods, but at the expense of increased inter-part surface area. However, both (hyper)graph and geometric-based methods balance only one type of mesh entity, thus the balance of other types of mesh entities might not be optimal; especially at high processor counts and when the part surface area to volume ratio is larger. Multi-constraint partitioning is supported by Zoltan's recursive coordinate bisection implementation and by the multi-level (hyper)graph methods [1, 25, 39], but cannot account for the duplication of multiple mesh entity types across inter-part boundaries and the subsequent increase in imbalance due to this duplication. For applications, such as C^O finite elements, controlling this imbalance is critical to scalability. As these peak entity imbalances typically occur in a relatively small number of parts that are surrounded by less heavily loaded parts a diffusive procedure can be very effective [23, 51] at quickly reducing the imbalances. ParMA provides a diffusive partition improvement algorithm that works in conjunction with (hyper)graph and geometric methods to reduce the peak imbalance of multiple mesh entity types specified by the application.

As presented in the sections that follow, ParMA, partitioning using mesh adjacencies, combined with graph and geometric based partitioning methods provided by Zoltan [4, 12], satisfy the requirements for dynamic load balancing to over one million parts on meshes with over 12 billion elements by extending the work of Zhou [23, 51] to support multiple mesh entity types (vertices, edges, faces, and regions). Partition quality requirements 1 and 2 are satisfied by partitioning the mesh with a graph or geometric based partitioner and then running ParMA to reduce the imbalance of mesh entity types critical to the application, such as the entities used as degree of freedom holders in Finite Element Method analysis procedures, while maintaining, or improving communication costs. The incremental partition change requirement is implicitly satisfied by the definition of ParMA's diffusion procedure and recursive coordinate bisection, and data movement minimizing graph-based method objectives are provided by Zoltan's API. Requirement 4 is implicitly satisfied as applications in the workflow are driven from the partitioning of the mesh that ParMA produces. Performance requirements 5 and 6 are satisfied by ensuring fast ParMA diffusion, choosing the base partitioner suitable for the concurrency level, and as appropriate, running process-local instances of the base partitioner [51]. Lastly, requirement 7 is satisfied through Zoltan's API to interact with the mesh data structure and ParMA's direct use of mesh modification and query APIs.

4. Partitioned Mesh Representation. PUMI, the parallel unstructured mesh infrastructure, provides a complete mesh representation that supports the efficient query of the intra and inter part mesh entity topology information [21, 40] needed by ParMA. The PUMI distributed mesh is the union of mesh parts. A mesh part is defined as a collection of mesh elements, faces M^2 in 2D and regions M^3 in 3D, assigned to a processing resource, typically a core or hardware thread. Within a part

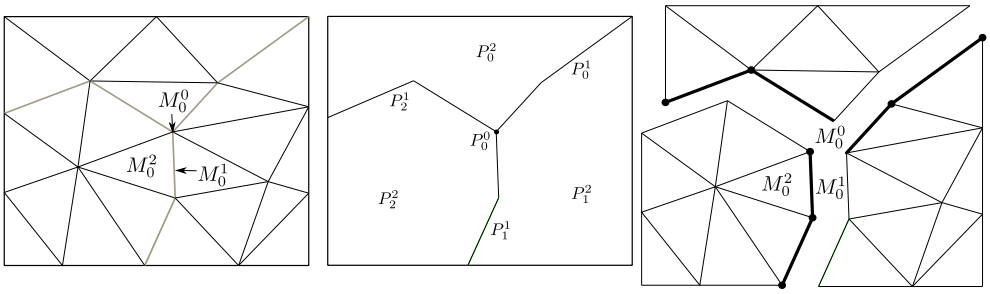


Figure 1: Example of a mesh (left), its partition model (middle), and its ownership (right). Discs and bold segments denote entity ownership.

the complete mesh representation provides mesh entity topology information through constant time adjacency queries. At the boundary of multiple parts mesh entities are copied, as shown for mesh vertex M_0^0 and edge M_0^1 in Figure 1, and tracked on each part through a remote copy object. Distributed mesh operations involving a mesh entity classified on the part boundary are coordinated through an ownership protocol; depicted by the discs and bold segments in Figure 1. Sets of mesh entities sharing common neighboring parts form a partition model entity and are defined to be classified on the partition model entity. For example, in Figure 1 mesh vertex M_0^0 is classified on the partition model vertex P_0^0, mesh edge M_1^1 is classified on the partition model edge P_1^1, and mesh face M_0^2 is classified on the partition model face P_2^2; respectively noted as $M_0^0 \sqsubset P_0^0$, $M_1^1 \sqsubset P_1^1$, and $M_0^2 \sqsubset P_2^2$. Information associated with entities classified on the partition model entities is exchanged through point-to-point neighborhood communications provided by PCU [20, 32].

5. Partition Improvement. ParMA reduces the peak partition imbalance of multiple entity types by iteratively migrating a small number of mesh elements from heavily loaded parts to neighboring parts with less load. The entity types to balance are defined by an application specified priority list. For example, if element>vertex is specified then the algorithm prioritizes element balance improvements over vertex improvements. To achieve element balance improvements vertex balance may be degraded, but vertex balance improvements cannot degrade the element balance. The balance of unlisted entity types (edges and faces in this example) are not considered and may be degraded. If vertex=element is specified, then the algorithm considers the balance of mesh elements and vertices equally important. The target imbalance for each listed entity type is specified by the application as $tgtImb^d$ where $d \leq d_{max}$ (the maximum dimension of a mesh). Applications which perform work on entities regardless of their ownership define the parts imbalance, I_p^d, as the weight of mesh entities of dimension d existing on part p divided by the average weight of dimension d entities per part. The maximum imbalance of dimension d entities across all parts is noted as I^d. When not specified, the weight of a mesh entity is set to one.

ParMA diffusive partition improvement is broken down into three stages. The first stage, targeting, determines how much load needs to be migrated and where it needs to go. The second, selection, marks elements for migration, and the third, migration, moves the marked elements to their defined destinations [21]. These stages are repeated for a specified number of iterations until the desired entity imbalance is reached, or no migration opportunities remain and stagnation occurs [23]. For

each entity type specified by the application, from high priority to low, the iterative process is repeated. Entities with equal priority are processed in order of increasing topological dimension as improvements in lower dimension entity imbalances are often observed to improve the balance of higher dimension entity types.

6. Targeting. Parts with an entity imbalance greater than the specified imbalance, $tgtImb^d$, are defined as heavily loaded parts. A lightly loaded part is defined based on the partition improvement objectives. If the application requires vertex=edge>element then migration to decrease element imbalance should not increase the imbalance of vertices or edges. Thus, during element improvement a part is a target to receive elements if it has fewer vertices, edges and elements than the heavy part. Migration from heavy parts to less heavily loaded parts can decrease the number of required iterations versus a more restrictive criteria that only permits migration to a neighboring part if it is under balanced, $I_q^d < tgtImb^d$. For example, a partition containing a heavily loaded part surrounded by other heavily loaded parts would require more iterations with the restrictive criteria as diffusion of the interior part could only occur after the boundary parts have migrated sufficient elements to become lightly loaded.

The amount of load, l_{pq}^d, migrated from a heavily loaded part p to a neighbor q during improvement of mesh entities of dimension d is defined as

$$l_{pq}^d = \alpha * sf * \left(\sum_i w(M_i^d \in p) - \sum_i w(M_i^d \in q) \right), \tag{1}$$

where $w(M_i^d)$ is the application specified weight associated with a given entity i, α is a diffusion rate limiting constant $\in (0, 1]$, and, in 3D, sf is the ratio of mesh faces shared by parts p and q to the total number of faces classified on partition boundaries of p. The surface area bias sf helps define load transfer requirements that can be satisfied in a single iteration by selecting elements for migration that are classified on the part boundary. A large transfer across a small boundary will not only take several iterations to satisfy, it will also lead to a large increase in the number of entities classified on the part boundary as each iteration will 'tunnel' into the part. The entity selection process is detailed in Section 7.

The affect of *alpha*, the diffusion rate limiting constant, on the run time and vertex imbalance is respectively shown in Table 1 and Figure 2 for a 2048 part mesh with an initial vertex imbalance of 46%. In this test setting α to 0.6 yields the fewest iterations and the shortest run time. Increasing or decreasing α from this value results in increased run time due to the migration of too many or too few elements in each iteration, respectively. The result of migrating too many elements in an iteration can be seen in the large oscillations in imbalance after ten iterations at α levels greater than 0.6. Given these observations α is set to 0.5 for all tests to allow at most 50% of the weight difference to be migrated in a single iteration and striking the balance between the total number of diffusive iterations and the imbalance reduction achieved in each iteration.

7. Entity Selection. Entity selection's primary objective is to reduce the imbalance of a given entity type. While selecting mesh elements for migration it is important to maintain inter-part boundaries with low surface area as an increase in the number of mesh entities classified on inter-part boundaries increases application communications, and in some cases, also the computational load [22]. Thus, entity selection's secondary objective is to reduce the number of mesh entities classified on

α	iterations	time (s)
0.2	45	19.3
0.4	31	14.0
0.6	27	12.5
0.8	27	13.5
1.0	31	16.6

Table 1: Diffusion iterations and run time for various α settings.

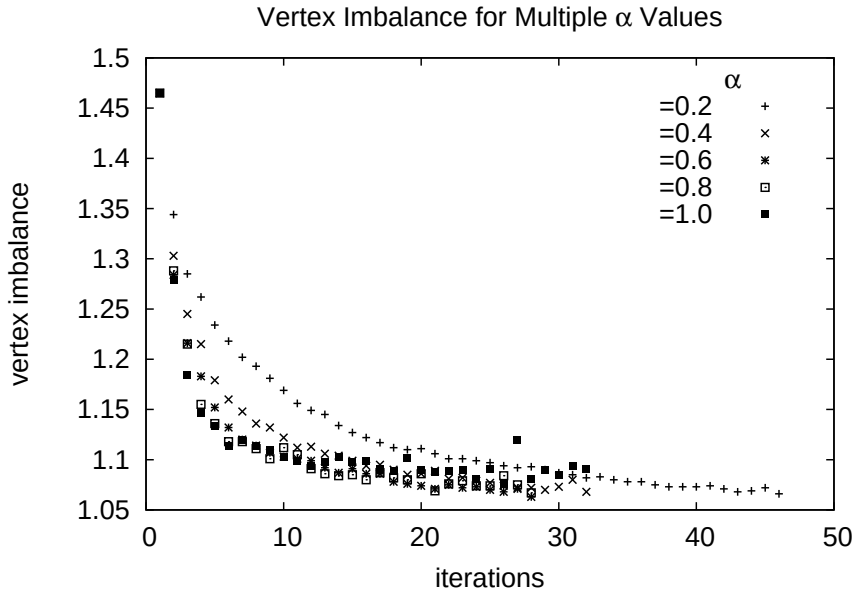


Figure 2: Affects of α on the number of iterations, run time, and vertex imbalance. The initial mesh has 2048 parts and a 46% vertex imbalance.

partition model entities of dimension $d < d_{max}$. To achieve these goals entity selection is driven by an entity level and a part level topological heuristic. The entity level heuristic considers the topology of the set of elements bound by a vertex classified on a partition model entity, a cavity, while the part level heuristic considers the connectivity of all the elements in the part.

7.1. Part Level Topological Distance Heuristic. The second objective of entity selection is to improve the part shape. This is accomplished by traversing the part boundary vertices in order of their graph distance from the topological center of the part when looking for elements to migrate. Thus as diffusive iterations are executed elements bounding vertices far from the part's core are migrated and the maximum graph distance of the part reduced; which at the limit encourages formation of parts with better volume to surface ratios and thus reduces communications.

Computing the topological distance requires addressing the full range of topological part characteristics. One such characteristic is the division of the part into multiple disconnected components; a component of a part is a set of elements in which there exists a path via M^{d-1} adjacencies between any two elements. Ideally a part

has only a single component, but this is not guaranteed by graph or geometric partitioning methods. Disconnected components are identified via a breadth first M^{d-1} adjacency based (faces in 3D) traversal starting at the first mesh element in the part. As elements are visited they are marked with the component id. When there are no more unmarked M^{d-1} adjacent elements to visit the component id is incremented and the traversal is restarted with an unmarked element. This process is repeated until all elements in the part are marked.

Graph distance computation begins by computing the center of each connected component via a breadth first traversal started from the component's boundary vertices. As the traversal tree grows inward the depth of each vertex in the tree is stored. When there are no more vertices to visit the traversal ends. From the set of vertices with the largest depth the first is chosen as the component's central vertex. In Figure 4a a part is shown with its component's core vertices marked with a 0. From these vertices Dijkstra's algorithm [9] is ran to compute the graph distance to all other vertices in the component.

An additional topological characteristic observed in the 60km 2D MPAS [8] ocean mesh of Figure 4 is the existence of components that have no mesh entities classified on the partition model boundary. Two of these 'isolated' components are shown in Figure 4b with dark shading. Since isolated components have no vertices classified on the partition model boundary their elements are not migrated during the diffusion process and as such are not distanced. Had partition improvement tests on the MPAS meshes described in Section 8.1 not been able to reach the specified imbalance levels, or if a significant portion of a part's weight was associated with isolated components, then some of those elements could be migrated to any part with less weight.

Within a component detection of local non-manifold [47] portions of the boundary is critical to ensure that the graph distance accurately records the shortest M^{d-1} adjacent path from the core to each vertex. For example, consider the 2D non-manifold vertex junction indicated by the arrow in Figure 4b and 4d. Here the paths from the core vertex marked in the upper portion of Figure 4a to either side of the junction will have significantly different lengths due to the large holes in the mesh formed by land masses. Detection of a non-manifold junction at a given boundary vertex, s, is through the breadth-first traversal of s's cavity vertices, rooted at the distance-1 parent of s. Vertices in the cavity are reachable via M^{d-1} adjacencies if the traversal can visit them without passing through s. For example, consider vertex s in the cavity depicted in Figure 3 to have the lowest distance in the priority queue of vertices being processed by Dijkstra's algorithm. The detection traversal starts at vertex p, the parent of s, by enqueuing vertices f and h. s is also edge-adjacent to p, by definition, but it is skipped as paths through it are not considered. The traversal continues by dequeuing a vertex and enqueuing its edge-adjacent vertices that have not been previously visited and are not s. Figure 3 depicts the depth of each edge in the traversal tree with hash marks. If there existed another element that was bounded by s that was also bounded by e and d, or h and b then the (e, d) or (h, b) edge would provide an edge-adjacent path from p to b, c and d and the junction would be identified as manifold.

Once the component vertex distances have been computed Algorithm 1 offsets them so that selection traverses all of one components boundary vertices before moving to another component. The procedure begins by sorting the components in order of descending depth, forming the list c. Next, on lines two through five the offset of the i^{th} component, r_i, is computed by summing the previous components max distance,

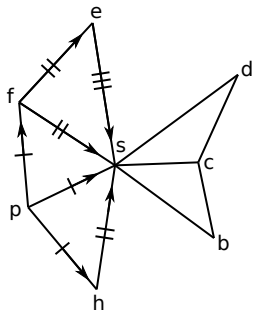


Figure 3: Detecting non-manifold component boundaries.

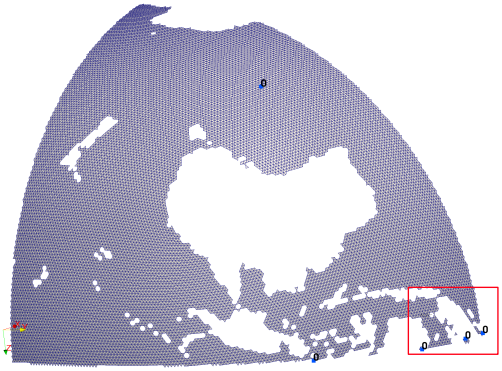
$max(R(M_j^0 \in c(i-1))$, for $j < i$, plus an upper bound on a component's distance, $maxDistIncrease$. Including $maxDistIncrease$ in the offset supports maintaining non overlapping distances between components of a part during distance recomputation for vertices migrated in the previous iteration. The final step on line nine loops over the components in ascending order of their depth and applies the offset to their vertices. This component traversal order, combined with the conditional checking that the current distance value is less than the offset, prevents the distance of vertices on the boundary of two components being offset multiple times. Figure 4c depicts the distance of the disconnected components before and after the offset is applied; for illustration purposes $maxDistIncrease$ is set to zero.

```
1  c ← sortDescending(components)
2  r_0 ← 0 //component zero's offset
3  for i ← 1 to numComponents do
4  |  r_i ← r_{i-1} + max(R(M_j^0 ∈ c(i − 1)) + 1 + maxDistIncrease
5  end
6  for i ← numComponents to 1 do
7  |  for M_i^0 ∈ c(i) do
8  |  |  if R(M_i^0) < r_i then
9  |  |  |  R(M_i^0) ← R(M_i^0) + r_i;
10 |  |  end
11 |  end
12 end
```

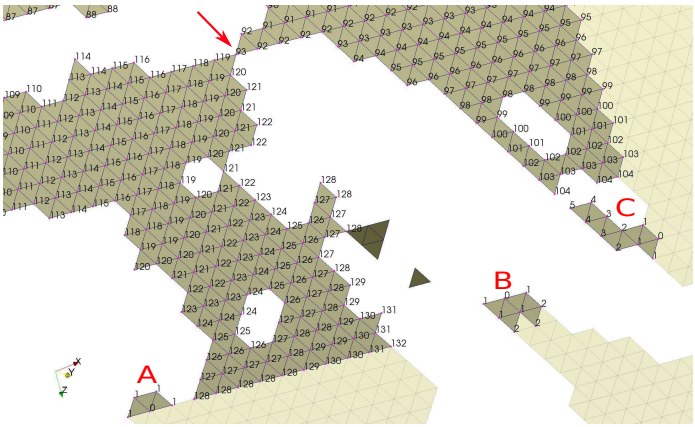
Algorithm 1: Vertex Distance Offset.

7.2. Entity Level Cavity Metric. Entity selection targets improving entity imbalance, the primary objective of partition improvement, by traversing the part's boundary vertices and evaluating a local topological heuristic. If only a few mesh elements are bounded by a vertex classified on a partition model entity then migrating the cavity of bounded elements will decrease the number of entities classified on partition model entities while decreasing the number of vertices on the source part. Conversely, migrating a cavity with several face connected elements can result in an increase in the number of mesh entities classified on partition model entities. Figures 5 (a-c) depict cavities of increasing size where the bounding vertex is marked with a disc, vertices classified on the partition model face P_j^2 bounded by parts P_0^3 and P_1^3

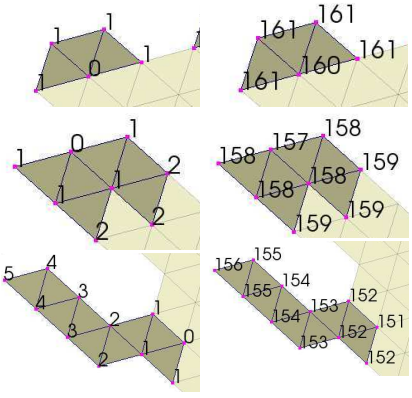
(a) Component core vertices.



(b) An edge-disconnected junction (arrow) and three disconnected components (A, B, C).



(c) Graph distance of disconnected components A, B, and C, before (left) and after (right) the offset is applied.



(d) Vertex graph distance near edge-disconnected junction.

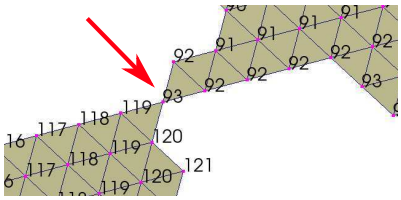


Figure 4: Components in one of four parts of the MPAS 60km [8] ocean mesh. Dark shaded elements are isolated and light shaded elements are on a different part.

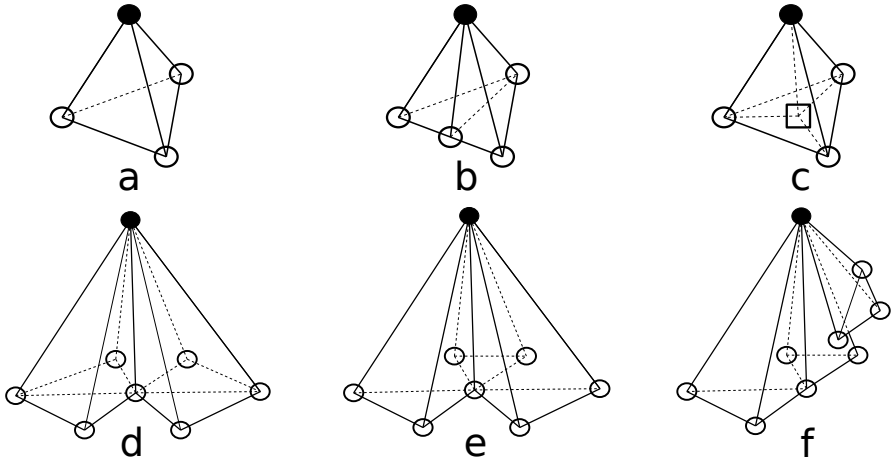


Figure 5: Vertex bounded cavities being migrated from part 0 to part 1.

	Cavity		
Entity Type	d	e	f
Vertex	1	1	1
Edge	5	7	8
Face	4	6	6

Table 2: Reduction in number of mesh entities classified on P_j^2 for cavities (d-f) in Figure 5.

with a circle, and in (c) a vertex classified on a partition model region, $M_i^0 \sqsubset P_0^3$, with a square. After migration of cavity (a-c) there is one less vertex classified on P_0^3. Migration of cavities (a) and (b) improves the part boundary by reducing the number of mesh entities classified on P_j^2; faces by two, edges by three, and vertices by one. Migration of cavity (c) does not change the number of entities classified on the part boundary since there is an entity added to the partition boundary for each one migrated. Note that the cavities (a-c) are face connected, i.e. there exists a path between any two regions in the cavity by traversing face adjacencies. A cavity that is not face connected can also reduce the number of vertices on the source part once migrated as they have a high surface area and will fill-in 'holes' in the neighboring part. Figures 5 (d-f) depict three face-disconnected cavities that, once migrated, reduce the number of vertices classified P_0^3 by one. Table 2 lists the substantial reduction in the number of mesh entities classified on P_j^2.

Cavities of elements are selected for migration in order of increasing size during topological distance ordered traversals of the part boundary vertices to reduce the number of mesh entities classified on the partition model boundary. The first traversal of the boundary will select only cavities comprised of single elements, followed by one and two element cavities in the second traversal (the single element traversal may have created new single element cases), and so on. Performance of vertex balancing to a target imbalance of 5% with the increasing cavity size approach is compared to a fixed max cavity size approach. In both approaches the maximum cavity size is 12 elements; the studies driving the choice of this size are not discussed here. The test mesh has

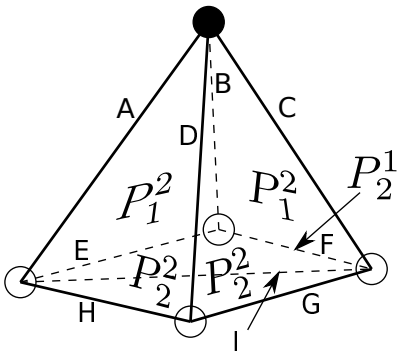


Figure 6: Counting mesh entity partition model classification to select either part 1 or part 2 for the migration of part 0 cavity elements. The cavity bounding vertex is marked with a disc.

228k elements partitioned to 2048 parts with RIB and has a 53% vertex imbalance, and a perfect element balance. Balancing with an increasing cavity size executes five iterations in 2.0 seconds on 2048 BlueGene/Q cores to reach the imbalance target while only increasing the element imbalance to 9%. The fixed cavity size balancing run achieves 5.7% vertex and 13% element imbalance after five iterations and continues for four more iterations before hitting the vertex imbalance target; its total run time is 3.4 seconds. During these iterations the increasing cavity size approach improves the part shape by reducing the average number of vertices per part by 3.4% while the fixed approach increases the average by 0.07%.

A cavity is migrated to the part with which it shares the most mesh edges, in 3D, and, in combination with the increasing cavity size selection, typically reduces the number of mesh entities classified on the part boundaries after migration. Counting shared edges avoids counting single vertices not bounding a higher order shared entity, the lowest dimension connection, while providing more information then the counting of shared faces, which only counts the highest dimension connections. Figure 6 depicts a two element cavity with entities classified on both partition model faces and edges; a more complex connectivity than those depicted in Figure 5. Specifically, the cavity has two faces shared with parts one and two, via classifications on partition model faces P_1^2 and P_2^2, respectively, and an additional classification of edge F on the partition model edge shared with part 2, P_2^1. Counting shared edges correctly identifies part two as the destination; it has six cavity edges versus part one only having five. The 'Sum' row of Table 3 lists the total cavity edge count on each part when the cavity elements are owned by part zero, the initial owner, and parts one and two, the two possible target parts. For this example migrating the cavity to part two reduces the total number of shared edges from 20 to 19; if part one were selected the total number of shared edges would increase by one. Note that the post migration entity existence counts can quickly be computed by determining which edges will remain on the source part, edges E, F, G, H, and I in this example, and then changing the total of the destination part to reflect existence of all the edges. In cases where a tie exists all the maximal edge sharing parts are considered and the first one that has remaining capacity is selected as the destination.

As the part boundary is traversed and the cavity heuristic selects elements for

		Cavity Owner								
		0			1			2		
	Part	0	1	2	0	1	2	0	1	2
Edge	A	1	1	1	0	1	1	0	1	1
	B	1	1	0	0	1	0	0	1	1
	C	1	1	1	0	1	1	0	1	1
	D	1	0	1	0	1	1	0	0	1
	E	1	1	0	1	1	0	1	1	1
	F	1	1	1	1	1	1	1	1	1
	G	1	0	1	1	1	1	1	0	1
	H	1	0	1	1	1	1	1	0	1
	I	1	0	0	1	1	0	1	0	1
	Sum	9	5	6	6	9	6	5	5	9

Table 3: Existence of edges on parts sharing edges with the cavity depicted in Figure 6. The column groups list the edge existence prior to migration of the cavity, Owner=0, and after migration to part N, Owner=N. An entry is '1' if the edge exists on the part. The last row lists the total number of cavity edges on each part.

migration the weight of selected entities are tracked to prevent migrating too much weight to the target parts. Tracking is based on the simple rule, rooted in the unique assignment of elements to parts, that an entity will not exist on the part if all the elements it bounds are marked for migration. Thus the weight tracking mechanism checks for this condition, and if satisfied, adds the entities weight to the running total for the given destination part.

During the balancing of lower priority entity types, i.e. elements during vertex > element balancing, the imbalance of higher priority entity types is preserved by canceling the migration of some elements. On the sending part this is done by iterating over the elements to be migrated and adding the higher priority entity weights if the entity is not on the part boundary with the target part. These weights are then sent to the respective targets. The target part then iterates over the incoming migration requests in descending order of the migration weight, accepts the request if capacity remains, reduces the remaining capacity accordingly, and sends the accepted weight to the sender. The sending part then traverses the list of migration elements in the order they were selected, descending distance from the parts topological core, and keeps elements in the list while the peer's higher priority entity weight capacity is not exceeded.

7.3. Stagnation Avoidance. A stagnation avoidance procedure stops execution of diffusion when the imbalance or part shape has not improved over several iterations. Specifically, a second order accurate backward finite difference [15] approximates the rate of change of the average part weight, *ents*, and the average number boundary mesh vertices per part, *sides*. Diffusion is stopped if the rate of change in *ents* and *sides* is respectively less than one percent of the target average, target imbalance multiplied by the initial average weight, and the initial average sides.

8. Results.

8.1. Feature Tests. ParMA vertex > element improvement was run on the 497,058 element MPAS North America 15km to 75km graded ocean mesh partitioned to 1,024 parts. The initial partition generated with local ParMETIS part k-way has a vertex and element imbalance of 36% and 17%, respectively, and on average, 280

vertices per part. Configuration 1 serves as the baseline for feature inclusion as it uses mesh iterator based part boundary vertex traversal, disables detection of non-manifold part junctions, a fixed cavity size for selection, and when balancing elements, does not cancel selections to help preserve vertex imbalance. Configurations 2, 3, 4, and 5 add the listed features that were previously described. Figure 8 depicts the relative change in partition quality versus the initial ParMETIS partition after ParMA vertex balancing, with a target imbalance of 5%, and ParMA vertex > element balancing, also with a 5% target imbalance. Partition quality is measured by the average number of parts neighbors per part, counted via shared vertices, 'avgNB/part', the average number of vertices (edges) per part, 'avgVtx(Edge)/part', and the entity imbalance, I^d. For each of these measures a value of one indicates no change from the initial partition, while a value greater (lower) than one indicates an increase (decrease) in the measure relative to the initial partition. In Figure 8a the partition quality after vertex balancing improves as features are added. The average number of neighbors, vertices, and edges per part increases by one percent or less with all features enabled. Relative to the over 20% decrease in vertex imbalance these increases are negligible.

Vertex > element balancing, Figure 8b, further improves the partition quality as features are enabled. After element balancing with no features enabled, Configuration 1, the element imbalance reduces from 5% to 3% at the cost of a vertex imbalance increase from 5% to 20%. Enabling topological distance traversal, Configuration 2, reduces the average number of disconnected components per part from Configuration 1's 50x increase versus the initial partition down to a 10x increase, as shown in Figure 8c. The large reduction in disconnected components reduces the surface area of the partition which results in a smaller increase in vertex imbalance to 13% after element balancing. Enabling the features of Configurations 2, 3, and 4 further reduces part surface area, as indicated by the reductions in average neighbors, vertices, and edges per part, and results in a 10% vertex imbalance. The final Configuration, 5, enables selection cancellation and limits the vertex imbalance increase to only 7%, reaches an element imbalance of 5%, and further improves the average neighbor and entities per part. Configuration 5 runs in 72% of the time of Configuration 1, 3.07 seconds versus 4.25 seconds. The faster run time is mainly the result of vertex balancing times reducing from 4.09 seconds to 2.82 seconds, and only a slight increase in the element balancing times from 0.16 seconds to 0.25 seconds.

The feature test was also run on the 3D 2.3 million element RPI Formula Hybrid suspension upright mesh, the geometric model depicted in Figure 7, with 2,048 parts and a 46% vertex and 10% element initial imbalance. In Configuration 1 balancing the mesh vertices to 10% increases the element imbalance to 26%. The subsequent element balancing reduces the element imbalance to 5%, and as features are enabled, further drives down the vertex imbalance to 9%. In Configuration 1 ParMA vertex>element improvement takes 17.8 seconds, but only reaches a vertex and element imbalance of 10% and 26%, respectively. Configuration 5 requires 37.6 seconds, a 2x increase over Configuration 1, and reaches respective imbalances of 9% and 5%. A critical difference of the upright tests to the MPAS tests is the large reduction in disconnected parts and the related decrease in the average neighbors and entities per part. Compared to the initial partition, Configuration 5 reduces the average neighbors, vertices, and disconnected components per part by 17%, 8%, and 93% respectively, and 3%, 2%, and 27% versus Configuration 1. This difference is mostly due to the change from 2D to 3D and the increased connectedness of the geometric model that enables more migration opportunities; the MPAS mesh has multiple geometric surfaces which only

Figure 7: Coarse mesh of the 2014 RPI Formula Hybrid suspension upright.

Configuration	Enabled Features
1	None
2	1 + topological distance traversal
3	2 + non-manifold feature detection
4	3 + increasing cavity size selection
5	4 + selection cancellation

Table 4: ParMA test configurations.

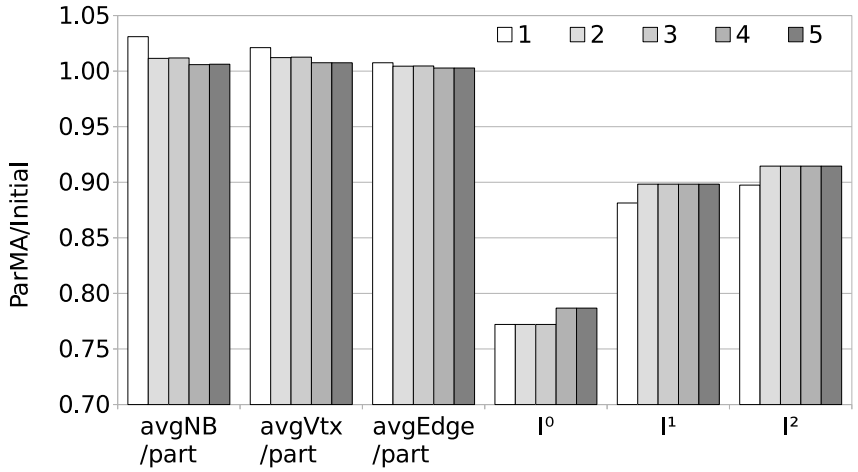
share one or two vertices with other surfaces.

The feature tests were run on N cores of the Rensselaer CCI BlueGene/Q where N is equal to the number of parts in the mesh.

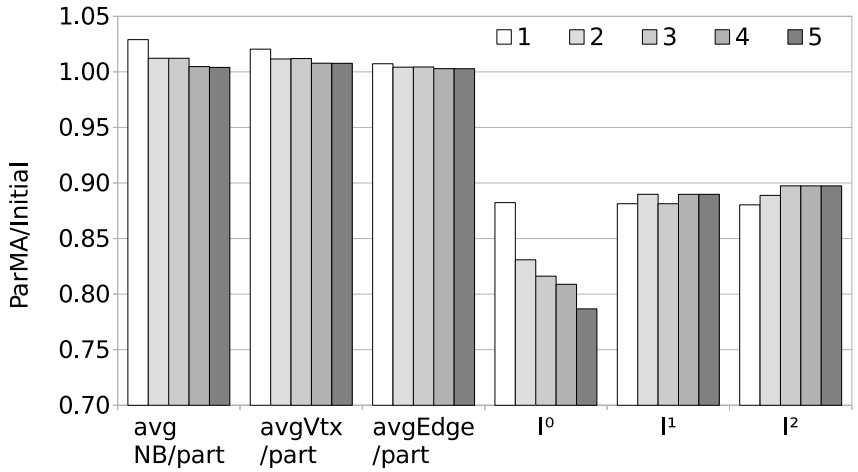
8.2. Multi Criteria Improvement. Analysis codes which have work associated with multiple entity types and have a non-uniform distribution of that work require multi criteria balancing. For example, consider an analysis using higher order shape functions that associate more degree of freedom holders with mesh edges. To account for the increased amount of work associated with mesh edges relative to mesh vertices during equation formation or equation solve weights can be applied to edges and vertices that are proportional to the work. The exact value of the weights is application specific and not discussed here.

Table 5 lists the partition improvement after running ParMA vertex = edge > element on a 2.3 million element, 2048 part, mesh of the RPI Formula Hybrid suspension upright of Figure 7. Mesh entity weights are set to one except on part zero where edges are given a weight of two; a bias representing a non-uniform distribution of work associated with edges. Two initial partitions were used in testing, one the result of mesh adaptation, adapt, and the other generated with RIB. Compared to the RIB partition, the adapt partition has a ten point higher element imbalance, and on average, four more neighbors and two more disconnected components per part. As a result of the poor adapt partition quality ParMA improvement on the adapt partition requires about 3.5x more time to run, 27.4 seconds versus 7.7 seconds, and has final entity imbalances, noted in the 'elements' row, a few points higher than the final ParMA imbalances of the RIB partition. Despite the disparity in initial quality and an initial edge weight imbalance of over 90%, ParMA reduces both cases average weight of vertices, edges, and faces per part by over 5%, 2%, and 1%, while reducing

(a) Partition quality after vertex balancing.



(b) Partition quality after vertex > element balancing.



(c) Disconnected components.

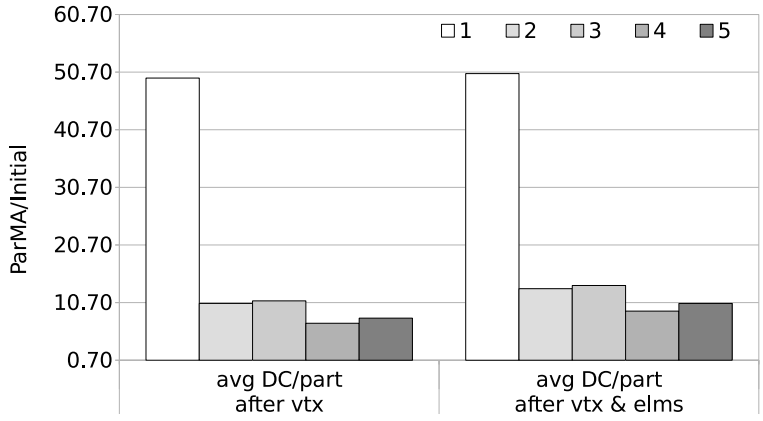
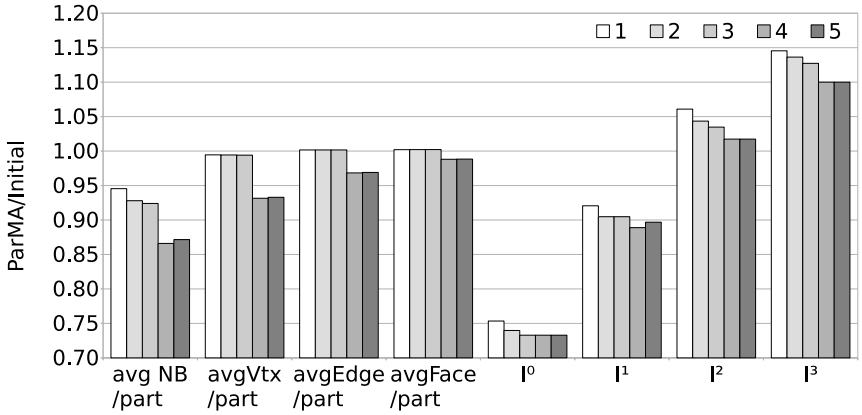
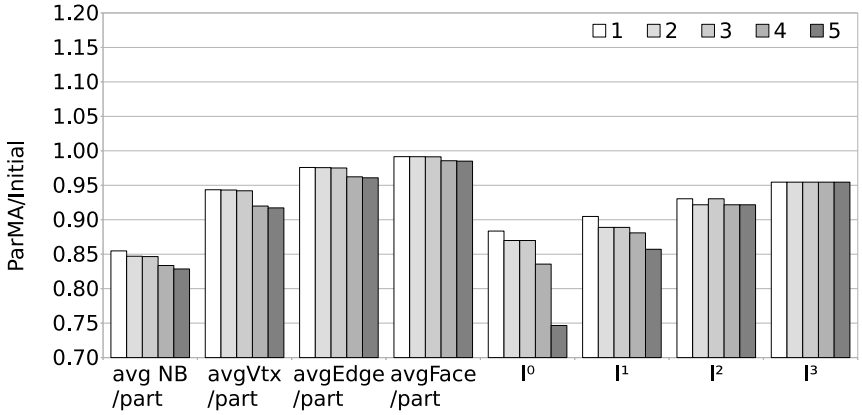


Figure 8: Partition quality of a 1,024 part MPAS North America 15km to 75km graded ocean mesh using the ParMA configurations listed in Table 4.

(a) Partition quality after vertex balancing.



(b) Partition quality after vertex > element balancing.



(c) Disconnected components.

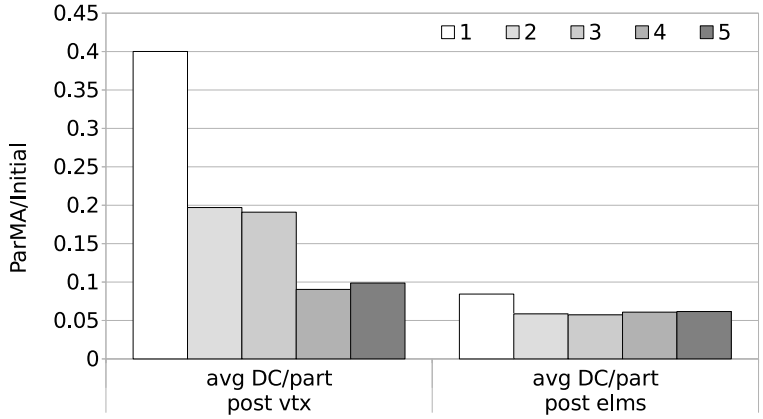


Figure 9: Partition quality of a 2,048 part RPI Formula Hybrid suspension upright mesh using the ParMA configurations listed in Table 4.

stage	avg/part vtx	avg/part edge	avg/part face	I^0	I^1	I^2	I^3	time (s)
adapt	**357.749**	**1741.012**	**2497.981**	**1.46**	**1.92**	**1.15**	**1.10**	
vertices	334.0	1687.7	2469.3	1.08	1.92	1.16	1.19	10.27
edges	330.5	1679.0	2464.3	1.09	1.06	1.09	1.13	6.88
elements	**328.829**	**1674.661**	**2461.637**	**1.09**	**1.06**	**1.07**	**1.08**	10.26
RIB	**350.457**	**1737.694**	**2503.369**	**1.53**	**1.92**	**1.10**	**1.00**	
vertices	337.8	1705.0	2483.4	1.04	1.95	1.07	1.10	3.01
edges	333.2	1692.8	2475.8	1.06	1.05	1.04	1.07	3.86
elements	**331.387**	**1687.526**	**2472.306**	**1.07**	**1.05**	**1.04**	**1.04**	0.85

Table 5: vertex=edge>element partition improvement on a 2.3 million element, 2048 part, mesh of the RPI Formula Hybrid suspension upright of Figure 7.

the entity imbalances to less than 9%. Critical to this result is ParMA's ability to diffuse away edge weight from the heavily imbalanced part zero while not overloading other parts; the number of mesh edges in part zero is reduced from 1674 to 901 and 1665 to 889 in the adapt and RIB partitions, respectively.

8.3. Partitioning to over one million parts. Several tests were run using ParMA on ALCF's Mira Blue Gene/Q where each hardware thread is assigned one part in the mesh. ParMA quickly reduces large imbalances and improves part shape of a 1.6 billion element mesh partitioned from 128K to 1M parts (approximately 1500 elements/part). (Note, K and M respectively represent multiples of 2^{10} and 2^{20}; i.e. 128K is 2^{17}.) The initial 128K partition has less than 7% imbalance for all entity types.

(i) Partitioning with global RIB completes in 103 seconds and results in a 209% vertex imbalance and a perfect element imbalance. ParMA runs on 1M processors in 20 seconds and reduces the vertex imbalance to 6%, only increases the element imbalance to 4%, and reduces the average number of vertices per part by 5.5%.

(ii) Partitioning with one serial instance of ParMETIS for each initial part completes in 9.0 seconds and results in a 63% vertex imbalance and a 12% element imbalance. ParMA runs in parallel on 1M processors in 9.4 seconds and reduces the vertex imbalance to 5%, the element imbalance to 4%, and reduces the average number of vertices per part by 2%.

Partitioning a 12.9 billion element mesh from 128K ($< 7\%$ imbalance) to 1M parts (approx. 12 thousand elements/part) using serial instances of ParMETIS completes in 60 seconds and results in a 35% vertex imbalance and an 11% element imbalance. Running ParMA in parallel on 1M processors reduces the vertex and element imbalances to 5% and reduces the average number of vertices per part by 0.6%.

Table 6 lists the number of elements, the initial and target part counts, and the initial entity imbalances, I^{0-3} for vertices, edges, faces and regions, respectively, for three partitions. Table 7 lists the results of ParMA runs on those partitions. Note, the column 'dec. (%)' lists the percentage decrease in the average vertices per part after ParMA relative to the partitioning stage, 'Split'.

8.4. CFD Scaling Improvement. As an example of ParMA's ability to improve simulations of very complex geometric models at extreme scale consider the geometry shown in Figure 10. Figure 10a and 10b respectively depict the surface of the vertical tail and rudder colored by part number and a detailed view of a complex geometric junction. Note that the mesh contains 1.2 billion tetrahedra. The starting

name	elements	parts	target parts	elms per tgt. part	I^0	I^1	I^2	I^3
small	1.6×10^9	2^{17}	2^{20}	1541.7	1.06	1.06	1.06	1.07
medium	12.9×10^9	2^{17}	2^{20}	12 333.8	1.05	1.06	1.07	1.07
large	12.9×10^9	2^{17}	2^{19}	24 667.6	1.05	1.06	1.07	1.07

Table 6: Initial meshes for upright tests.

scope	density	method	stage	avg vtx	dec. (%)	I^0	I^1	I^2	I^3	tot (s)
local	small	rib	Split	455.1		1.34	1.18	1.13	1.13	10.67
			ParMA	427.6	6.42	1.07	1.06	1.05	1.05	8.94
		pmetis	Split	427.0		1.63	1.32	1.13	1.12	8.99
			ParMA	418.8	1.97	1.05	1.05	1.04	1.04	9.48
	medium	rib	Split	2825.5		1.31	1.14	1.08	1.07	54.32
			ParMA	2752.0	2.67	1.06	1.05	1.04	1.05	48.81
		pmetis	Split	2687.7		1.35	1.14	1.11	1.11	59.81
			ParMA	2671.3	0.61	1.05	1.05	1.04	1.05	36.15
	large	rib	Split	5273.9		1.16	1.13	1.12	1.13	42.69
			ParMA	5122.9	2.95	1.05	1.04	1.04	1.05	52.87
		pmetis	Split	5132.4		1.21	1.09	1.10	1.10	37.02
			ParMA	5102.2	0.59	1.04	1.04	1.04	1.04	41.55
global	small	rib	Split	470.1		**3.09**	2.07	1.45	1.00	103.14
			ParMA	445.4	5.54	**1.06**	1.04	1.03	1.04	20.23
	large	rib	Split	5367.3		2.49	1.70	1.29	1.00	96.79
			ParMA	5228.8	2.65	1.05	1.02	1.03	1.04	379.84

Table 7: X+ParMA vertex > element upright test results.

partition for this study was obtained through a series of steps. An initially balanced 4K part mesh became imbalanced due to the application of an error-based local adaptation as described in [7, 35]. RIB was applied globally to this adapted mesh to create an 8K part mesh whose quality was improved by applying ParMA such that the vertex and element imbalance was reduced to less than five percent. Starting from this 8K part mesh, ParMETIS part k-way [25] was applied locally to each part to create partitions of the mesh in powers of two from 16K parts to 1024K parts. A second set of partitions was created using ParMA to improve each partition. The case shown in Figure 10 includes 256K parts and thus not all part boundaries are discernible. The complexity of the geometry is even more apparent in the right half of this figure where a zoom on a clip plane cutting through the very small gap between the vertical stabilizer and the rudder is seen to contain many parts which are "cutoff" from the surrounding geometry and thus challenging to improve their quality.

The flow in this case is solved with PHASTA [48] and its strong scaling performance is highlighted in [34]. As this is an implicit solver, the equation formation scaling is primarily dictated by element balance while the equation solution scaling is primarily dictated by the vertex balance. As shown in Figure 11, through six part-count doublings, ParMA is able to improve both the element and the vertex imbalance. At one million parts ParMA reduces the imbalance from 11% to 4% and 86% to 6%, respectively. As expected, the impact on equation formation is significant but moderate (not shown) while the impact on equation solution is far more dramatic as can be seen in Figure 12. Here the ratio of the time PHASTA spends solving the equations (time using original partition divided by the time spent when using ParMA's

(a) Stabilizer and rudder. (b) Detail view of stabilizer-rudder junction.

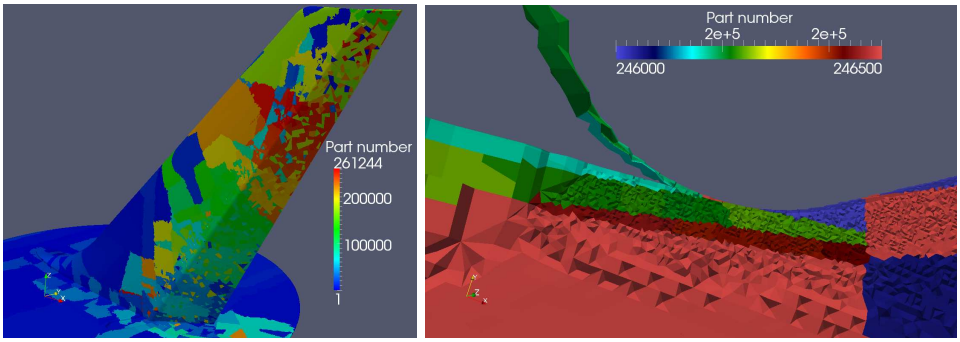


Figure 10: Full view (left) of the vertical stabilizer and rudder and a slice at their junction (right) colored by part number illustrating the complex geometry and small features of the fluid mesh.

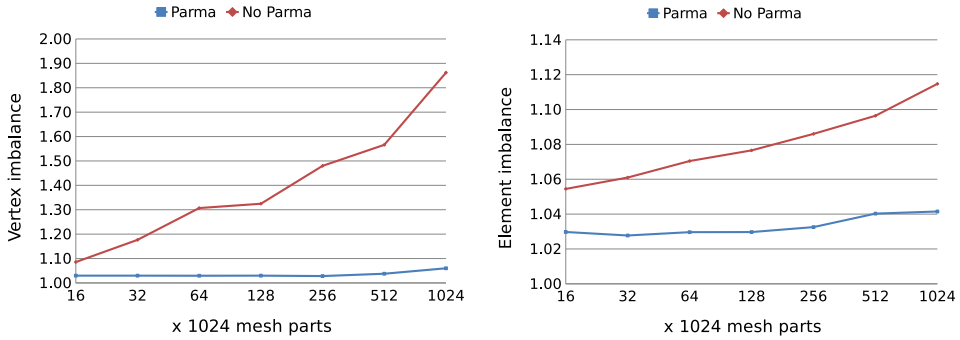


Figure 11: Evolution of the number of the vertex (left) and element (right) imbalance with and without ParMA.

improved partition) is plotted for each partition with one, two and four MPI processes per core on the BG/Q Mira system located at the Argonne Leadership Computing Facility. This is a direct measure of the improvement due to ParMA with up to 54% acceleration of the equation solve. As expected this metric is somewhat noisy since it is strongly affected not just by ParMA's improvement but also by the quality of the original partition. Here, partition quality is first due to the vertex imbalance which linearly influences the equation solution scaling but it is also affected by other partition quality metrics. Space constraints of this paper do not allow a thorough analysis and discussion of these secondary metrics, rather they will be discussed in a companion paper [43] where this example will be used to thoroughly study these metrics at extreme scale.

9. Closing Remarks. The ability to evenly distribute the work associated with mesh entities in parallel scientific unstructured mesh based applications is critical to scalability on massively parallel leadership class systems. ParMA, partitioning using

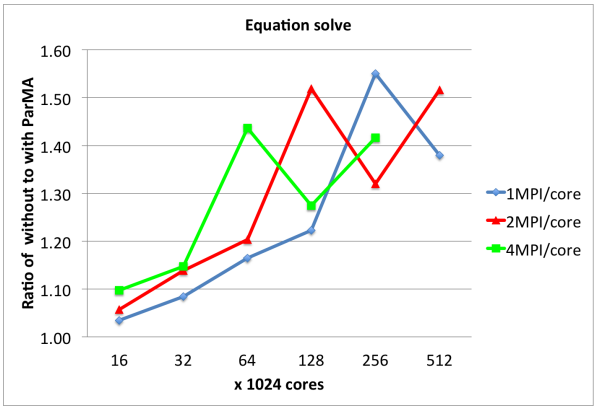


Figure 12: Improvement of PHASTA performance with ParMA.

mesh entities, coupled with graph and geometric based partitioning tools, provides fast partition improvement to meet this need on over one million processors. Ongoing efforts focused on analyzing the impact of PHASTA scalability with respect to specific partition quality metrics will be the focus of a companion paper [43].

REFERENCES

[1] Cevdet Aykanat, B Barla Cambazoglu, and Bora Uçar, *Multi-level direct k-way hypergraph partitioning with multiple constraints and fixed vertices*, Journal of Parallel and Distributed Computing, 68 (2008), pp. 609–625.

[2] Marsha J Berger and Shahid H Bokhari, *A partitioning strategy for nonuniform problems on multiprocessors*, Computers, IEEE Transactions on, 100 (1987), pp. 570–580.

[3] Erik G. Boman, Karen D. Devine, Vitus J. Leung, Sivasankaran Rajamanickam, Lee Ann Riesen, Deveci Mehmet, and Catalyurek Umit, *Zoltan2: Next-generation combinatorial toolkit.*, Trilinos Users Group Meeting, (2012).

[4] E. Boman K. 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`.

[5] Umit V Catalyurek and Cevdet Aykanat, *Hypergraph-partitioning-based decomposition for parallel sparse-matrix vector multiplication*, Parallel and Distributed Systems, IEEE Transactions on, 10 (1999), pp. 673–693.

[6] Umit V Catalyurek, Erik G Boman, Karen D Devine, Doruk Bozdağ, Robert T Heaphy, and Lee Ann Riesen, *A repartitioning hypergraph model for dynamic load balancing*, Journal of Parallel and Distributed Computing, 69 (2009), pp. 711–724.

[7] Kedar C Chitale, Michel Rasquin, Onkar Sahni, Mark S Shephard, and Kenneth E Jansen, *Anisotropic boundary layer adaptivity of multi-element wings*, in 52nd Aerospace Sciences Meeting (SciTech). AIAA Paper, vol. 117, 2014.

[8] Sea-Ice Modeling Team. Los Alamos National Laboratory Climate, Ocean, *MPAS-Ocean Model User's Guide. Version 2.0*, 2014. `http://oceans11.lanl.gov/mpas_data/mpas_ocean/users_guide/release_2.0/mpas_ocean_users_guide_2.0.pdf`.

[9] T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein, *Introduction To Algorithms*, MIT Press, 2001.

[10] George Cybenko, *Dynamic load balancing for distributed memory multiprocessors*, Journal of parallel and distributed computing, 7 (1989), pp. 279–301.

[11] M. Deveci, S. Rajamanickam, K. Devine, and U. Catalyurek, *Multi-jagged: A scalable parallel spatial partitioning algorithm*, Parallel and Distributed Systems, IEEE Transactions on, PP (2015), pp. 1–1.

[12] Karen Devine, Erik Boman, Robert Heaphy, Bruce Hendrickson, and Courtenay Vaughan, *Zoltan data management services for parallel dynamic applications*, Computing

in Science & Engineering, 4 (2002), pp. 90–96.
[13] Jack Dongarra, Ian Foster, Geoffrey Fox, William Gropp, Ken Kennedy, Linda Torczon, and Andy White, eds., *Sourcebook of parallel computing*, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2003.
[14] C.M. Fiduccia and R.M. Mattheyses, *A linear-time heuristic for improving network partitions*, in 19th Design Automation Conference, IEEE, 1982, pp. 175–181.
[15] Bengt Fornberg, *Generation of finite difference formulas on arbitrarily spaced grids*, Mathematics of computation, 51 (1988), pp. 699–706.
[16] Bruce Hendrickson and Karen Devine, *Dynamic load balancing in computational mechanics*, Computer Methods in Applied Mechanics and Engineering, 184 (2000), pp. 485–500.
[17] YF Hu and RJ Blake, *An improved diffusion algorithm for dynamic load balancing*, Parallel Computing, 25 (1999), pp. 417–444.
[18] Y. F. Hu, R. J. Blake, and D. R. Emerson, *An optimal migration algorithm for dynamic load balancing*, Concurrency: Practice and Experience, (1998).
[19] Thomas JR Hughes, *The finite element method: linear static and dynamic finite element analysis*, Courier Dover Publications, 2012.
[20] Daniel A Ibanez, Ian Dunn, and Mark S Shephard, *Hybrid mpi-thread parallelization of adaptive mesh operations*, Parallel Computing. Submitted, (2014).
[21] Daniel A Ibanez, E Seegyoung Seol, Cameron W Smith, and Mark S Shephard, *Pumi: Parallel unstructured mesh infrastructure*, ACM Transactions on Mathematical Software, (2015).
[22] Kenneth E Jansen, Christian H Whiting, and Gregory M Hulbert, *A generalized-¡ i¿ α¡/i¿ method for integrating the filtered navier–stokes equations with a stabilized finite element method*, Computer Methods in Applied Mechanics and Engineering, 190 (2000), pp. 305–319.
[23] M. Zhou O. Sahni K.D. Devine M.S. Shephard K.E. Jansen, *Controlling unstructured mesh partitions for massively parallel simulations*, SIAM Journal on Scientific Computing, 32 (2010), pp. 3201 – 3227.
[24] G. Karypis and V. Kumar, *A fast and high quality multilevel scheme for partitioning irregular graphs*, SIAM Journal on Scientific Computing, 20 (1998), pp. 359–392.
[25] George Karypis and Vipin Kumar, *Multilevel algorithms for multi-constraint graph partitioning*, in Proceedings of the 1998 ACM/IEEE conference on Supercomputing (CDROM), IEEE Computer Society, 1998, pp. 1–13.
[26] ———, *Parallel multilevel series k-way partitioning scheme for irregular graphs*, Siam Review, 41 (1999), pp. 278–300.
[27] Brian W. Kernighan and S. Lin, *An Efficient Heuristic Procedure for Partitioning Graphs*, The Bell System Technical Journal, (1970), pp. 291–307.
[28] Dominique LaSalle and George Karypis, *Multi-threaded graph partitioning*, in Parallel & Distributed Processing (IPDPS), 2013 IEEE 27th International Symposium on, IEEE, 2013, pp. 225–236.
[29] Henning Meyerhenke, Burkhard Monien, and Stefan Schamberger, *Graph partitioning and disturbed diffusion*, Parallel Computing, 35 (2009), pp. 544–569.
[30] R.M. O'Bara, M.W. Beall, and M.S. Shephard, *Attribute management system for engineering analysis*, Engineering with Computers, 18 (2002), pp. 339–351.
[31] Chao-Wei Ou and Sanjay Ranka, *Parallel incremental graph partitioning using linear programming*, in Supercomputing'94., Proceedings, IEEE, 1994, pp. 458–467.
[32] Aleksandr Ovcharenko, Daniel Ibanez, Fabien Delalondre, Onkar Sahni, Kenneth E Jansen, Christopher D Carothers, and Mark S Shephard, *Neighborhood communication paradigm to increase scalability in large-scale dynamic scientific applications*, Parallel Computing, 38 (2012), pp. 140–156.
[33] Ekkehard Ramm, E Rank, R Rannacher, K Schweizerhof, E Stein, W Wendland, G Wittum, Peter Wriggers, and Walter Wunderlich, *Error-controlled adaptive finite elements in solid mechanics*, John Wiley & Sons, 2003.
[34] Michel Rasquin, Cameron Smith, Kedar Chitale, E Seegyoung Seol, Benjamin A Matthews, Jeffrey L Martin, Onkar Sahni, Raymond M Loy, Mark S Shephard, and Kenneth E Jansen, *Scalable Implicit Flow Solver for Realistic Wing Simulations with Flow Control*, Computing in Science & Engineering, (2014), pp. 13–21.
[35] Onkar Sahni, Jens Müller, Kenneth E Jansen, Mark S Shephard, and Charles A Taylor, *Efficient anisotropic adaptive discretization of the cardiovascular system*, Computer methods in applied mechanics and engineering, 195 (2006), pp. 5634–5655.
[36] Erik Saule, Erdeniz Ö. Baş, and Ümit V. Çatalyürek, *Load-balancing spatially located computations using rectangular partitions*, Journal of Parallel and Distributed Computing,

72 (2012), pp. 1201 – 1214.

[37] Kirk Schloegel, George Karypis, and Vipin Kumar, *Multilevel diffusion schemes for repartitioning of adaptive meshes*, Journal of Parallel and Distributed Computing, 47 (1997), pp. 109–124.

[38] ———, *Wavefront diffusion and lmsr: Algorithms for dynamic repartitioning of adaptive meshes*, Parallel and Distributed Systems, IEEE Transactions on, 12 (2001), pp. 451–466.

[39] ———, *Parallel static and dynamic multi-constraint graph partitioning*, Concurrency and Computation: Practice and Experience, 14 (2002), pp. 219–240.

[40] E. Seegyoung Seol, Cameron W. Smith, Daniel A. Ibanez, and Mark S. Shephard, *A parallel unstructured mesh infrastructure*, High Performance Computing, Networking, Storage and Analysis (SCC), 2012 SC Companion:, (2012), pp. 1124–1132.

[41] Mark S Shephard, M.W. Beall, R.M. O'Bara, and B.E. Webster, *Toward simulation-based design*, Finite Elements in Analysis and Design, 40 (2004), pp. 1575–1598.

[42] Mark S Shephard, Cameron Smith, and John E Kolb, *Bringing HPC to Engineering Innovation*, Computing in Science & Engineering, 15 (2013), pp. 16–25.

[43] Cameron W. Smith, Michel Rasquin, Dan Ibanez, Mark S. Shephard, and Kenneth E. Jansen, *Partition improvement to accelerate extreme scale cfd*, SIAM Journal on Scientific Computing, in preparation (2015).

[44] Valerie Taylor and Bahram Nour-omid, *A study of the factorization fill-in for a parallel implementation of the finite element method*, Int. J. Numer. Meth. Engng, 37 (1994), pp. 3809–3823.

[45] Saurabh Tendulkar, Mark Beall, Mark S Shephard, and KE Jansen, *Parallel mesh generation and adaptation for cad geometries*, in Proc. NAFEMS World Congress, 2011, pp. 2011–2.

[46] C Walshaw, Mark Cross, and MG Everett, *Dynamic mesh partitioning: A unified optimisation and load-balancing algorithm*, London: CMS Press, 1995.

[47] Kevin Weiler, *The radial edge structure: a topological representation for non-manifold geometric boundary modeling*, Geometric modeling for CAD applications, (1988), pp. 3–36.

[48] C. H. Whiting and K. E. Jansen, *A stabilized finite element method for the incompressible Navier-Stokes equations using a hierarchical basis*, International Journal of Numerical Methods in Fluids, 35 (2001), pp. 93–116.

[49] Marc H Willebeek-LeMair and Anthony P. Reeves, *Strategies for dynamic load balancing on highly parallel computers*, Parallel and Distributed Systems, IEEE Transactions on, 4 (1993), pp. 979–993.

[50] Roy D. Williams, *Performance of dynamic load balancing algorithms for unstructured mesh calculations*, Concurrency: Pract. Exper., 3 (1991), pp. 457–481.

[51] Min Zhou, Onkar Sahni, Ting Xie, Mark S Shephard, and Kenneth E Jansen, *Unstructured mesh partition improvement for implicit finite element at extreme scale*, The Journal of Supercomputing, 59 (2012), pp. 1218–1228.