SAND2019-6773C

A Parallel Graph Algorithm for Detecting Mesh Singularities in Distributed Memory Ice Sheet Simulations

Ian Bogle
boglei@rpi.edu
Rensselaer Polytechnic Institute
Troy, NY

Karen Devine
Mauro Perego
Sivasankaran Rajamanickam
kddevin,mperego,srajama@sandia.gov
Sandia National Laboratories
Albuquerque, NM

George M. Slota
slotag@rpi.edu
Rensselaer Polytechnic Institute
Troy, NY

ABSTRACT

We present a new, distributed-memory parallel algorithm for detection of degenerate mesh features that can cause singularities in ice sheet mesh simulations. Identifying and removing mesh features such as disconnected components (icebergs) or hinge vertices (peninsulas of ice detached from the land) can significantly improve the convergence of iterative solvers. Because the ice sheet evolves during the course of a simulation, it is important that the detection algorithm can run *in situ* with the simulation — running in parallel and taking a negligible amount of computation time — so that degenerate features (e.g., calving icebergs) can be detected as they develop. We present a distributed memory, BFS-based label-propagation approach to degenerate feature detection that is efficient enough to be called at each step of an ice sheet simulation, while correctly identifying all degenerate features of an ice sheet mesh. Our method finds all degenerate features in a mesh with 13 million vertices in 0.0561 seconds on 1536 cores in the MPAS Albany Land Ice (MALI) model. Compared to the previously used serial pre-processing approach, we observe a 46,000x speedup for our algorithm, and provide additional capability to do dynamic detection of degenerate features in the simulation.

KEYWORDS

biconnectivity, climate modeling, graph algorithms

ACM Reference Format:
Ian Bogle, Karen Devine, Mauro Perego, Sivasankaran Rajamanickam, and George M. Slota. 2019. A Parallel Graph Algorithm for Detecting Mesh Singularities in Distributed Memory Ice Sheet Simulations. In *48th International Conference on Parallel Processing (ICPP 2019), August 5–8, 2019, Kyoto, Japan.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3337821.3337841

1 INTRODUCTION

Modeling sea-level rise (SLR) is important in climate modeling. A major factor contributing to SLR is mass loss from the Greenland and Antarctic ice sheets [4]. To predict SLR accurately, ice sheet dynamics are simulated using large-scale parallel computational models (e.g. [5, 9, 11]). Typically, these models assume that ice flow is reduced by friction with bedrock, and fail to simulate regions, such as icebergs, where the ice is floating and partially or completely detached from the land ice. It is therefore important to detect these regions during simulation so that they can be properly handled.

Most large scale ice sheet models employ finite-element or finite-volume discretizations using unstructured ice sheet meshes. These are either 2D or 3D meshes generated by extruding 2D basal meshes. Therefore, detecting regions that are problematic for the solvers reduces to detecting "degenerate features" in 2D basal ice sheet meshes as detailed in [18]. These degenerate features can develop over the course of an ice sheet simulation; therefore, it is of paramount importance that the detection can be efficiently performed at runtime on meshes held in distributed memory.

In our study, we consider only 2D conformal meshes, such as the basal triangular and quadrilateral meshes used in the Albany Land Ice (Albany-LI) [17] velocity solver component of the MALI model [9]. Zou et al. [20] considered the detection of degenerate features in non-conformal block-structured grids; their proposed method works efficiently for data generated by adaptive mesh refinement algorithms. However, they detect only a subset of the degenerate features that we are required to detect.

The degenerate mesh features we must detect are similar to the biconnected components of a graph. Biconnected components are maximal subgraphs of a graph such that the removal of any single vertex will not disconnect the subgraph. Zou et al. only considers detection of connected components. Connected components are maximal subgraphs such that at least one path exists between all vertex pairs within the subgraph.

We view a mesh as an undirected graph, with mesh vertices corresponding to vertices in the graph, and graph edges defined by the element connectivity in the mesh. Finding the biconnected components of the graph reveals vertices that are single points of failure, such as floating chunks of ice that are about to become icebergs. While our study focuses on ice sheet simulations, biconnectivity algorithms are also useful for fault tolerance in ad hoc networks [12], and detecting mechanisms in meshes for structural dynamics [6]. There are shared-memory algorithms for graph biconnectivity (e.g., Tarjan and Vishkin [15]), but to support parallel ice sheet simulations such as Albany-LI, distributed-memory algorithms that do not rely on a global view of the graph are needed.

Our Contributions: We present an efficient, distributed-memory algorithm for detecting degenerate features in ice sheet meshes. We implemented our algorithm in the Zoltan2 [2] graph algorithms library and demonstrated it with the Albany-LI solver. We show that our algorithm is fast enough to run at every step of a simulation, taking at most 0.4% of the runtime of a single solver step in Albany-LI using a 13M element mesh. We explore the algorithm's performance with synthetic meshes in which we vary the size and number of degenerate features, and show that the algorithm performs well for meshes that approximate real ice sheet meshes.

2 BACKGROUND AND RELATED WORK

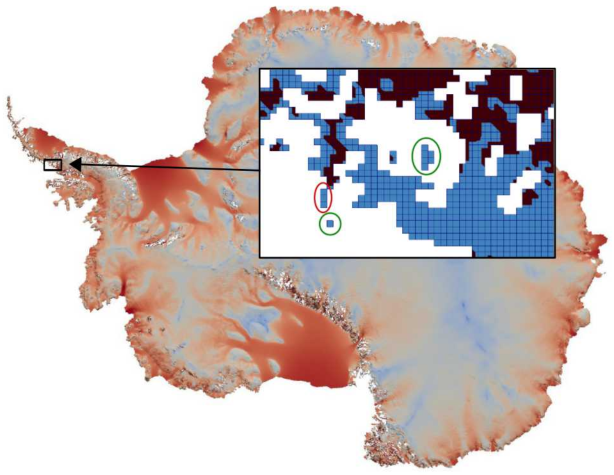
Figure 1: Antarctic Ice Sheet colored with ice surface velocity magnitude (red = fast, blue = slow), and mesh detail showing a realistic map of floating (light blue) and grounded (brown) ice and degenerate features marked with green (icebergs) and red (hinges) circles.

Degenerate features of an ice sheet mesh correspond to parts of the ice that are floating and either detached from grounded ice (icebergs) or loosely attached to it (hinged peninsulas). Examples of these features are shown in the mesh of Antarctica in Figure 1. We refer to vertices corresponding to places where the ice is in contact with the ground as "grounded" vertices and to vertices corresponding to places where the ice is floating as "floating" vertices. Icebergs (i.e., floating islands) are connected components of the mesh that are constituted entirely of floating vertices. Hinged peninsulas, or "hinges," are portions of the mesh that are constituted of floating vertices and that can be disconnected from the mesh by the removal of a single vertex. Both icebergs and hinged peninsulas can negatively impact solver convergence and need to be identified as degenerate features. In fact, in these regions, ice flow equations are typically ill-posed because the ice velocity is known only up to translations and/or rotations.

Given a mesh and information about whether each mesh vertex is grounded or floating, our goal is to determine whether or not each mesh vertex is part of a degenerate feature.

Degenerate Feature Detections Algorithms: The previous detection algorithm for Albany-LI was implemented as a standalone MATLAB code [18] and used as a pre-processor to Albany-LI simulations. To remove floating islands, Tuminaro et al. used a Breadth-First Search (BFS) from every unvisited grounded vertex. Vertices that remained unvisited after the BFS operations had to be part of a floating island. Tuminaro et al. used graph coloring to remove hinged peninsulas. Each mesh vertex was colored so that no neighboring vertices had the same color. Then, for each color, they temporarily removed vertices of that color and re-ran the floating island algorithm; hinged peninsulas would become floating islands when the colored hinge vertices were temporarily removed. This approach generally had runtimes of several minutes when running on meshes with millions of elements.

A closely related problem was addressed in [20], considering meshes generated by adaptive mesh refinement (AMR) algorithms. This work found icebergs (but not hinged penisulas) in multi-level, multi-resolution meshes. Their method detects connected components at each level of the AMR structure and joins them together globally. To find connected components at a single level, they use the SAUF algorithm [19], a two-pass labeling algorithm that assigns temporary labels to each vertex on the first pass, uses a Union-Find data structure to determine which temporary labels are equivalent, and finalizes labels on the second pass. In the distributed implementation, Zou et. al run the first pass of SAUF on each process, considering only local vertices. Then they exchange ghost label information, and construct the final set of labels by sending all local Union-Find structures to an elected process on the same AMR level. After assigning labels on a level-by-level basis, they do a similar procedure to assign final labels for the entire AMR structure. Each unique label then corresponds to a connected component across the entire AMR structure. Similarly, Harrison et al. [7] present a connected component algorithm for 3D meshes based on union-find operations; they demonstrate its performance on 2197 processor with two-billion element meshes. However, these connected component approaches find icebergs only. Our biconnected component method detects both icebergs and floating peninsulas that are connected to the main ice sheet by a single point — features that create significant challenges for solvers.

Biconnectivity Algorithms: Graph biconnectivity decomposition algorithms seek to identify all maximal biconnected subgraphs as well as all cut vertices within some graph. Cut vertices, commonly referred to as articulation points or articulation vertices, are single vertices that disconnect the graph when removed.

Biconnectivity is a well studied problem, with a work-optimal serial algorithm presented by Hopcroft and Tarjan [10]. This work optimal algorithm uses Depth First Search (DFS) to identify biconnected components. DFS algorithms, however, are not easy to parallelize, as discussed by Tarjan and Vishkin [15]. Tarjan and Vishkin [15] present a parallel algorithm for finding biconnected components in a concurrent-read, concurrent-write parallel RAM model, where each processor has access to shared memory. This parallel algorithm reduces the problem of biconnectivity to the problem of connectivity in an auxiliary graph. The auxiliary graph can be constructed without using DFS, and its connectivity can then be found efficiently in shared memory.

Two shared memory algorithms for biconnectivity exploit simple graph operations such as BFS and color propagation [14] to identify articulation points. The BFS-based algorithm does BFS sweeps to determine whether children of certain vertices can reach other vertices on their parents' level; if so, the parent is not an articulation point. The coloring version also does an initial BFS; it then uses color propagation rules to prevent certain color labels from being passed through articulation points. Most recently, LCA-BiCC was proposed by M. Chaitanya and K. Kothapalli [3]. They observe that that finding bridges (edges whose removal disconnects the graph) is more parallelizable than finding articulation vertices. They use an arbitrarily rooted BFS on the input graph to find a set of non-tree edges (edges not in the BFS tree). Then they find the Lowest Common Ancestor (LCA) for each of the endpoints of the non-tree edges. They show that this set of LCA vertices is guaranteed to contain the set of all articulation points in the graph, although it can contain non-articulation points as well. Edges not visited in finding LCA vertices are bridges in the original graph; each bridge has endpoints that are articulation points. The algorithm is then applied recursively on the subgraphs formed by excluding the bridges. An efficient biconnectivity algorithm has not yet been demonstrated on a distributed graph representation. Distributed 2-edge connectivity algorithms exist, but these only find bridges.

One could use a biconnectivity algorithm to perform degenerate feature detection by identifying biconnected components and then testing each component to determine whether it is connected to ground. (Indeed, we used this strategy to verify the correctness of our proposed algorithm.) But by exploiting information about the mesh to identify potential articulation points and propagating the grounding information in the BFS operations, we can provide an efficient algorithm that is feasible for distributed memory.

3 ICE SHEET FEATURE DETECTION

Algorithm 1 shows a high-level view of our approach. The inputs of our algorithm are a mesh M and grounding information for each vertex indicating whether the vertex is grounded. From the mesh M, we can extract a graph $G = (V, E)$ in which vertices in V correspond to mesh vertices in M and edges in E correspond to mesh vertex adjacencies along element edges in M.

Algorithm 1 Degenerate feature detection algorithm

```
1: procedure PROP-ALG(Mesh $M$,$grounding\_info$)
2:     Compute set of potential articulation points using $M$
3:     Extract graph $G$ from $M$
4:     $labels \leftarrow \emptyset$
5:     Propagate initial $grounding\_info$ labels in $G$
6:     while Propagation is incomplete do
7:         Continue propagating, repropagate if necessary
8:     Return $labels$ indicating connection to ground
```

The first step in Algorithm 1 is to identify a set of potential articulation vertices in the mesh. For correctness, this set must include at least all true articulation points. The set may include vertices that are not true articulation points; indeed, the set of all mesh vertices can be used. However, the algorithm completes more quickly if the set of potential articulation points is close to the set of true articulation points, and we can exploit mesh information to closely approximate the set of true articulation points (Section 3.1).

Grounding information is then passed from grounded vertices to neighboring vertices via label propagation. Each vertex v has a label that is a structure of four vertex IDs: two representing grounded vertices to which there is a path from v in G, and two that record which vertices propagated the grounded vertex IDs to v. Grounded vertices' labels are initialized with their vertex IDs, and they propagate their vertex IDs to neighboring vertices in a breadth-first manner. When a vertex receives new grounded information, it stores the grounded vertex IDs it received and propagates them further (Section 3.2). Initial propagation halts at potential articulation points. Then propagation is restarted, with care taken to propagate grounding information correctly through potential articulation points (Section 3.3). Once all vertices have been labeled correctly, propagation ends and the labels indicate whether a vertex is grounded or not. Vertices with two grounded-vertex IDs in their labels have two paths to the ground, and thus, are not part of any degenerate features. Other vertices are part of degenerate features and will be removed by the ice sheet simulation.

3.1 Identifying Potential Articulation Points

When considering a geometric 2D mesh, all articulation points will be located on the boundary. Given the elements of a mesh, it is straightforward to compute the boundary edges of the mesh; the boundary edges are those that are not shared by two elements. Then, given the list of boundary edges, we determine which vertices are potential articulation points by looking at the number of boundary edges incident to each boundary vertex. If a boundary vertex has two incident boundary edges, it cannot be an articulation point; its two incident edges must either come from adjacent elements sharing some other edge incident on that boundary vertex or exists on a corner of the mesh. If a vertex has more than two incident boundary edges, it is a potential articulation point. For example, in Figure 2 (top), vertex B is a potential articulation point because it has more than two boundary edges, while vertex F is not.

3.2 Label Propagation Rules

All grounded vertices initialize the grounded vertex IDs in their labels to their own vertex IDs. During label propagation, vertices share their grounding information with neighboring vertices. Algorithm 2 shows the rules used to update a neighboring vertex's label. Vertices that are not potential articulation points may give their neighbors all (zero, one or two) of the unique grounded vertex IDs that they have. The neighboring vertices track from which vertices they received each grounded vertex ID. If a neighboring vertex already has two grounded vertex IDs, it is "full" and is not updated.

We ensure that potential articulation points send only one vertex ID to each neighbor, as an articulation point can contribute only one point of contact to the ground to any neighbor. Full potential articulation points pass their own vertex ID as a grounded vertex, rather than either of the grounded vertex IDs that they store. "Half-full" potential articulation points (those having only one grounded vertex ID in their label) pass their one grounded vertex ID. Because labels contain the vertex ID of the vertices giving a stored grounded vertex ID, a potential articulation point can determine whether or not it has previously given the neighbor a grounded vertex ID.

3.3 Propagation on Two Frontiers

We use two queues to manage propagation along two "frontiers": propagation from potential articulation points (*art_frontier*) and propagation from other vertices (*frontier*). As shown in Algorithm 3,

Algorithm 2 Function for updating neighboring vertices labels during propagation

```
procedure Give-Labels(curr_vtx, neighbor)
    if curr_vtx ∈ Potential_Articulation_Points then
        if curr_vtx hasn't sent anything to neighbor before then
            if curr_vtx has two grounded vertex IDs then
                curr_vtx gives neighbor its own vertex ID as
                a grounded vertex
            else
                curr_vtx gives its only grounded vertex ID
                to neighbor
    else
        curr_vtx gives neighbor any grounded vertex IDs
        that neighbor doesn't have
```

the two queues separate the potential articulation points from the other vertices while propagating. Initially, grounded vertices are placed in the appropriate queue (*frontier* or *art_frontier*, depending on whether they are or are not potential articulation points). Our algorithm begins propagation from the *frontier* queue; when that queue is empty, it swaps to the *art_frontier* queue and resumes propagation. This swap-and-propagate pattern continues until both queues are empty, meaning no labels changed in the previous iteration. Separating potential articulation vertices from non-articulation vertices allows potential articulation points to accrue as much grounding information as possible before passing that grounding information along. In particular, it increases the likelihood that potential articulation points will have full labels before they propagate their values, which reduces the total number of propagation iterations we need.

Algorithm 3 BFS-based label propagation algorithm

```
procedure BFS-Prop(frontier,art_frontier,labels,G=(V,E))
    if frontier.empty() then
        swap(frontier,art_frontier)
    while !frontier.empty() do
        curr_vtx ← frontier.pop()
        for all neighbors n of curr_vtx do
            Give-Labels(curr_vtx,n)
            if n's label changed then
                if n ∈ Potential_Articulation_Points then
                    art_frontier.push(n)
                else
                    frontier.push(n)
        if frontier.empty() then
            swap(frontier,art_frontier)
```

3.4 Multi-Phase Conditions

Although rare in ice sheet meshes, features such as chains of articulation points can occur and require special handling to correctly identify all degenerate features. These situations require additional phases of the propagation algorithm. Figure 2 shows an example. To determine whether additional propagation is needed, we identify

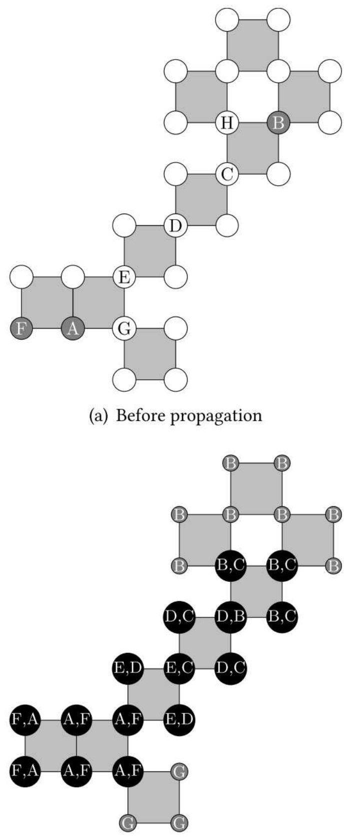

Figure 2: An example demonstrating label propagation

potential articulation vertices that have two grounded vertex IDs in their label and a half-full neighbor vertex whose only grounded vertex ID is not the potential articulation point's ID. We add all such potential articulation points to a frontier, reset labels that have only one grounded vertex ID, and propagate from this new frontier. Algorithm 4 shows this addition to BFS-Prop. The multi-phase approach is required only when certain topological conditions exist in the mesh and there is a large difference in shortest path distance between two grounded vertices and non-grounded vertices in the mesh. These conditions are rarely, if ever, encountered in practice, but we provide mitigation to ensure correctness.

3.5 Propagation Example

Figure 2 shows an example that illustrates parts of our algorithm. Initially, only grey vertices A, B, and F are grounded; all white vertices are not grounded initially. Labels for A, B and F are initialized to their respective IDs. Vertices A and F are placed in the *frontier* queue, since they are grounded and not potential articulation points. As a potential articulation point, vertex B is placed in *art_frontier*.

Vertex A propagates its label to its neighbors and adds its neighbors to the appropriate queues depending on their potential articulation status. In particular, A gives F its label so that F is full with label (F, A). Vertex F then propagates its full label to its neighbors,

Algorithm 4 Driver for BFS-Prop, Complete, Multi-Phase Solution

```
procedure BFS-Prop-Driver(labels, G=(V,E))
    art_frontier ←grounded potential articulation vertices
    frontier ←grounded vertices that are not potential
            articulation points
    BFS-Prop(frontier,art_frontier,labels,G)
    while true do
        for all potential articulation points $v \in V$ do
            if $v$'s label is full then
                for all neighbors $n$ of $v$ do
                    if $n$'s label is half-full and doesn't have
                      $v$'s ID then
                        art_frontier.push(v)
        if art_frontier.empty() then
            break
        for all $v \in V$ do
            if $v$'s label is half full then
                clear $v$'s label
        BFS-Prop(frontier,art_frontier,labels,G)
```

and the neighbors propagate (A, F) to their neighbors. (Note that labels (F, A) and (A, F) are equivalent.) The neighbors of A and F in *frontier* then propagate label (A, F) to potential articulation points E and G, giving them both full labels.

Next, from the *art_frontier* queue, vertices B, E, and G propagate to their neighbors; because E and G have full labels (A, F), they propagate their own vertex IDs (see Algorithm 2). Not-full neighbors of E and G are placed in the *frontier* queue (since their neighbors are not potential articulation points), and the swapping between the *frontier* and *art_frontier* queues continues.

We eventually reach the state in Figure 2 (bottom). This state is not the final state, as the vertices labeled B do not yet have full labels. However, at this point, the queues are both empty. Algorithm 4 (lines 7-12) finds that the vertices labeled B may not be in their final state, so a new round of propagation is initiated. All half-full labels are emptied, and propagation is restarted from vertex H. We do not need to restart propagation from vertex G, so the degenerate feature connected to G ends up with no labels.

4 DISTRIBUTED MEMORY IMPLEMENTATION

We have created a distributed memory implementation of our algorithm that is callable by parallel mesh-based applications. Because the application's mesh is likely already distributed to processors in a balanced manner for its computation, we use the same distribution of data to processors as the application. We assume the commonly used "owner computes" strategy for the application's distribution of mesh entities; that is, each processor identifies a set of unique vertices for which it is responsible. Our load balance and communication patterns are thus determined by the application's distribution of the data; however, since our runtimes are very small compared to the application's solve times, redistributing data within our method to adjust load balance is not worthwhile.

The application also provides all edges incident to its owned vertices, including edges to off-processor vertices. Then in each processor p, we store an "owned" graph vertex for each mesh vertex owned by p. We also create one layer of "ghost" vertices — copies of vertices that are owned by some processor $q \neq p$, and are neighbors of vertices owned by processor p. Using the edges provided by the application, we create a "local" graph consisting of the owned vertices, the ghost vertices and the edges.

In distributed memory, the label-propagation algorithm can maintain its "push" of label values to neighboring vertices without ill effects. We perform Algorithm 3 (BFS-Prop) independently on each processor's local graph, allowing labels to propagate to both owned and ghost vertices. Once local propagation stops, we communicate (via MPI point-to-point messages) to push the ghost vertices' labels to their owning processor. The owned labels are then made consistent with values received from the ghost copies. A global communication (all-reduce) is used to determine whether any processor had meaningful label changes due to the exchange of ghost information. If so, the ghost vertices are updated via point-to-point communication from their owners and propagation resumes.

We use existing classes from the Tpetra [1] package of Trilinos [8] to implement this strategy. We build Tpetra Maps to describe the distribution of the owned and ghosted vertices among processors. To store labels for owned and ghosted vertices, we use the Maps to create a Tpetra FEMultiVector — a distributed vector that provides ghost-exchange communication and ghost update capabilities. The FEMultiVector was designed for finite element assembly operations, allowing processors to contribute physical values to ghost vertices and sum the contributions from multiple processors. It is templated on a scalar type, which is usually *double* in physics simulations. We, however, provide our label structure as the scalar type, and overload its summation operator to use our Give-Labels function (Algorithm 2). To use the FEMultiVector's communication capabilities, we call FEMultiVector's method beginFill() before we start the local label propagation, and endFill() to communicate the ghost vertices' labels to their owners and "sum" their values into the owned versions using the overload summation operator. To communicate the owned labels back to the ghost copies, we use the FEMultiVector method doOwnedToOwnedPlusShared().

5 CORRECTNESS PROOFS

Here we will prove the correctness of our algorithm. First, we'll show that any vertex with two labels after propagation completes must have at least two internally-vertex-disjoint paths to grounded vertices. As mentioned in our algorithms description, the existence of at least two such paths indicates *grounded* status to the given vertex; we keep these vertices during simulation. We don't need to differentiate between vertices that were initially grounded or those that were marked grounded during propagation. Then, we'll show that our propagation rules guarantee any vertex with two internally-vertex-disjoint paths will end with two labels. Taken together, we have our primary proposition:

Proposition 5.1. *Under our label propagation rules, a vertex n will own two unique labels, l_1 and l_2 $\iff$ $\exists P_1, P_2$, two vertex-disjoint paths from n to two unique grounded vertices, v_1 and v_2.*

Proof: A is set of potential articulation points. B is a pseudo-BiCC component containing n. A pseudo-BiCC is a maximum biconnected subgraph bounded by potential articulation points. v_1 and v_2 are grounded vertices, potentially anywhere in the graph. Paths P_1, P_2 are termed (internally) vertex-disjoint if they have no vertex in common except n. By our propagation rules, n having some label l implies that $\exists P$, where P is a (minimal length) path tracing n to some grounded vertex $v \in G$, as labels only propagate along edges.

We first prove that the existence of two unique labels implies the existence of two vertex-disjoint paths to grounded vertices. Assume vertex n has two vertex identifiers in its label, v_1 and v_2.

Trivial Cases: The trivial cases are if both labels refer to vertices v_1 and v_2 which are neighbors of n, or if one label refers to n (i.e., n is grounded). The first case is obvious, and our propagation rules implies $\exists P_2$ from n to v_2 for the second case. This path would obviously be vertex-disjoint with empty path $P_1 = \{n\}$.

Nontrivial Case: Assume n has two labels referring to grounded vertices located anywhere within the graph. Per our propagation rules, this implies $\exists P_1, P_2$ from n to each of v_1, v_2. An equivalent statement to what we're trying to prove is that $\exists P = \{v_1, \ldots, n, \ldots, v_2\}$; i.e., a path from v_1 to v_2 containing n. Assuming this path doesn't exist, $\exists s$, where s is one vertex that P_1 and P_2 must traverse between both v_1 or v_2 and n. The removal of only s would therefore disconnect n from v_1 and v_2; hence, s is an articulation point and $s \in A$, as actual articulation points are a subset of A. By our propagation rules, s can only pass one label, so n having labels v_1 and v_2 is a contradiction, and therefore such an s can't exist.

We next show that the existence of two vertex-disjoint paths to grounded vertices for n implies that n terminates propagation with two labels. Assume that n has two vertex-disjoint paths, P_1 and P_2, to initially grounded vertices, v_1 and v_2. We consider three cases.

Case 1 – $n, v_1, v_2 \in B$: In this case, all three vertices considered are contained within the same pseudo-BiCC. Based on our propagation rules, the labels v_1 and v_2 will propagate from the grounded vertices. If we consider P_1 and P_2 as a minimal vertex-disjoint pair of paths, these labels will reach n without passing through some potential articulation point to another pseudo-BiCC; B itself is biconnected, which with our propagation rules guarantees the existence of some path through any three vertices contained within it.

Case 2 – $v_1 \in B$ and $v_2 \notin B$: In this case, v_2 is contained in B', a pseudo-BiCC distinct from B. v_1's label will propagate along P_1 within B to n unimpeded. v_2's label will begin propagating along P_2, which potentially passes through some number of potential articulation points. If B' neighbors B (*Case 2.1*) with some $x \in A, B, B'$, labels from both v_1 and v_2 will reach x. When the propagation frontiers swap, x will propagate its own label along the portion of P_2 to n, giving n its second label.

If there are multiple pseudo-BiCCs between B and B', then at some $y \in A$ along P_2 labels from v_1 and v_2 will intersect. If P_1 and P_2 are both minimal paths and not just a minimal vertex-disjoint path pair (*Case 2.2*), y will propagate its label along P_2 toward n until it either reaches n or some $z \in A$, which will subsequently begin propagation of its label toward n. After some number of potential articulation points, two labels will reach $x \in A, B$, which will finally propagate its label the rest of the way to n along P_2 within B.

If P_1 and P_2 are not both minimal paths (*Case 2.3*) is when we might require multiple phases of iteration. The shorter path between v_1 or v_2 and n might propagate its label to "consume" all potential articulation points surrounding n, as demonstrated in Figure 2. Recall we start a new phase by clearing half-labels and re-propagating from the newly defined ground. In our new phase, we can guarantee that there exists some new ground v_3 that has a shorter path to n than v_1 or v_2 in the prior phase; at a minimum, the pseudo-BiCC where minimal paths between v_1, v_2 and n intersect will contain v_3. By guaranteeing that we decrease the distance of at least one minimal grounded path to n during each phase, we will eventually reach one of the prior cases.

Case 3 – $v_1 \notin B$ and $v_2 \notin B$: In this case, we can apply the logic used in *Case 2* on both P_1 and P_2. We omit it for brevity.

Note that in the above cases, if there are greater than just two vertex-disjoint paths from n, it simply follows that at least two unique labels will reach n. Conversely, we provide the following simple corollary to complete our correctness proof.

Corollary 5.2. *Under our label propagation rules, a vertex n will own less than two unique labels $\iff$ there exists less than two vertex-disjoint paths from n to unique grounded vertices.*

Proof: First assume that our algorithm terminates with n having less than two labels. If n has two or more vertex-disjoint paths to grounded vertices, then by Proposition 5.1, n will have ended up with two unique labels, a contradiction.

Next, to prove the other direction of our corollary, assume that n has less than two vertex-disjoint paths from n to unique grounded vertices. If no path exists, then no label could reach n. If $\exists P_1$ from n to grounded vertex v_1, then either $v_1 \in B$ or $v_1 \notin B$. If $v_1 \in B$, then its trivial to show that the only label that will reach n is from v_1, as either there are no other grounded vertices, or paths from n from these grounded vertices must pass through $v_1 \in A$. If $v_1 \notin B$, then P_1 must pass through some $x \in A, B$. As x is a potential articulation point, it will only pass a single label, regardless of how many other grounded vertices exist outside of B with a path to n through x.

5.1 Complexity Discussion

Generally speaking, our propagation algorithm utilizes a "frontier" in a similar fashion to breadth-first search. A vertex will be placed on the frontier at most twice as its label set is filled. As such, the number of propagations a vertex will send along an edge during a given phase is bounded by a maximum of two. The number of phases we require is bounded above by the cardinality of our potential articulation point set. So we can give a worst-case work complexity of $O(|E||A|)$, where E is the set of edges and A is the set of potential articulation points. However, we note that we have never required more than a single phase on real data or the synthetic data in our results. In practice, we see a *linear expected work complexity* on real-world ice sheet data of $O(|E|)$. For parallel time, the number of propagation iterations is dependent on the diameter of the graph d, which for rectangular meshes grows approximately with $O(\sqrt{|V|})$, where V is the set of vertices; note that each propagation iteration itself can be parallelized in $O(1)$ time on $O(|E|)$ processors. We therefore have a worst-case time of $O(d|A|)$ and expected time of $O(d)$ on $O(|E|)$ processors.

Table 1: Real (top) and synthetic (bottom) mesh data, including the numbers of vertices, elements, potential articulation points and vertices removed from the mesh.

Mesh	#Vertices	#Elements	Potential	#Removed
16km	52,465	51,087	21	0
8km	210,170	206,436	51	14
4km	841,346	831,173	174	6
2km	3,368,275	3,341,449	389	22
1km	13,479,076	13,413,766	606	65

Mesh	#Vtx	#Elems	#Potential	#Removed
	(max)	(min)	(max)	(max)
ground(2km)	3,364,589	3,354,197	260	21
numdegen	~3,364,589	1,563,762	15,800	2,683,500
numcomplex	~3,368,253	2,878,139	17,313	21
longdegen	~3,365,592	1,565,262	904,858	63,000
longcomplex	~3,368,235	2,870,687	25,229	21
syn-largest	16,236,896	16,186,433	1550	96

6 EXPERIMENTAL SETUP

The scaling results we present were obtained on AMOS, RPI's Blue Gene/Q housed at the Center for Computational Innovations. AMOS has 5K nodes with 80K cores and 80TB of RAM.

We used real meshes of Antarctica from the ProSPecT ice sheet project [13]. These meshes have geographic resolution from 16km to 1km. Smaller resolutions result in more refined meshes; thus, the number of elements ranges from 51K to 13.4M.

We also generated synthetic meshes by specifying the size of the central ice mass and the number and size of the degenerate and complex features. First, the central ice mesh was created by connecting a regular grid of vertices together in elements of four vertices. Then we create "complex" features, which are blocks that are similar to the central ice block, but smaller. These features are chains of elements; each one starts on the edge of the central ice sheet, has a number of intermediate ice elements, and then connects back to a different vertex on the edge of the central ice sheet. Degenerate features are similar, but they connect to the central ice sheet at only one point on its edge. Grounding information is generated randomly for the vertices in the central ice sheet and the complex features. Degenerate features are targetted to be removed, so we do not allow them to be grounded initially. As we varied the parameters we were testing, the numbers of vertices in meshes of equivalent resolution varied slightly; maximum values are reported in Table 1. Likewise, the orientation of the complex and degenerate features may vary slightly due to random generation.

For our scaling studies, we distributed mesh vertices equally among processors using "linear" distributions that assign $|V|/P$ vertices to each of P processors in the order of their global vertex ID numbers. These distributions are likely not optimal with respect to locality of vertices. However, because our method uses the same parallel distribution as applications calling it, we did not investigate optimal partitioning strategies in our tests.

7 RESULTS

To evaluate our method, we ran experiments on AMOS with the real and synthetic ice sheet meshes in Table 1. All tests correctly identified degenerated features in the meshes; thus, we focus on the performance of our method. We did weak and strong scaling studies, as well as specialized analyses to show how mesh features (number of grounded vertices, number and length of degenerated features, number and length of complex features) affect our algorithm.

7.1 Feature Detection Performance

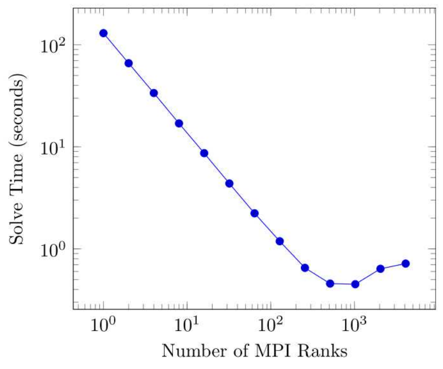

Figure 3: Strong scaling: runtime time using the largest real ice sheet mesh, *1km*

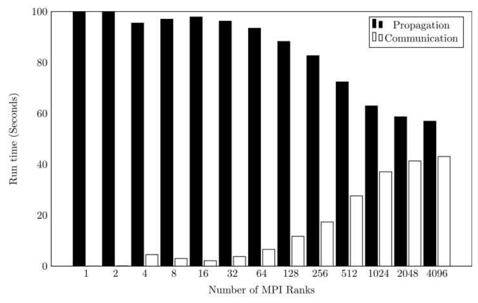

Figure 4: Strong scaling: percentage of runtime time for propagation and communication with real mesh *1km*

7.1.1 Strong Scaling. The largest real mesh, *1km*, has over 13.4 million vertices. Strong scaling for this mesh is nearly perfect up to 512 MPI ranks, as shown in Figure 3. Beyond 512 ranks, communication increases, and computation takes longer due to a larger number of frontier switches in the propagation phase. The breakdown of computation and communication times is shown in Figure 4.

The largest synthetic mesh, *syn-largest*, was slightly larger than our largest real case at 16 million elements. The results in Figure 5 show good strong scaling as well. There is little reduction in speedup as the number of MPI ranks approaches 4096; the extra vertices may aid the scaling, or our mesh generator may not perfectly replicate the features of the real ice sheet mesh. Figure 6 shows how much time our algorithm spends propagating, and how much time our algorithm spends communicating while solving a synthetic test

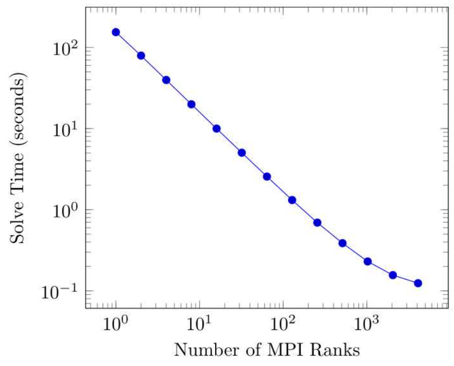

Figure 5: Strong scaling: runtime time for the largest synthetic ice sheet mesh, *syn-largest*

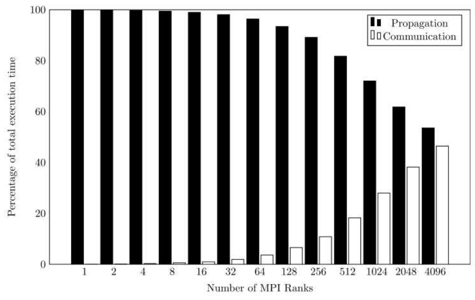

Figure 6: Strong scaling: percentage of runtime for propagation and communication with synthetic mesh *syn-largest*

case. The time spent communicating remains nearly constant as the number of processors increases, and the computation time reduces nearly by half when we double the number of processors.

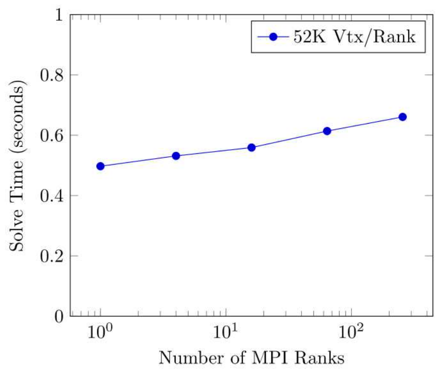

Figure 7: Weak scaling: runtime time for real meshes *16km, 8km, 4km, 2km, and 1km*

7.1.2 *Weak Scaling.* The availability of real ice sheet meshes with varying resolutions enables effective weak-scaling experimentation. As mesh resolution is halved in each dimension, the number of elements increases by roughly four (see Table 1, top). Our weak-scaling tests on the real meshes *16km, 8km, 4km, 2km and 1km* assigned roughly 52K vertices per MPI rank on AMOS. Figure 7 shows that our algorithm's weak scaling is good on real data.

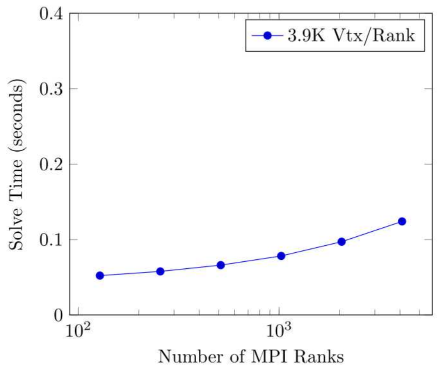

Figure 8: Weak scaling: runtime time for synthetic meshes

Our weak-scaling tests on synthetic meshes assigned roughly 3.9K vertices per MPI rank. Figure 8 shows our algorithm has good weak scaling on the synthetic data, even with a small workload.

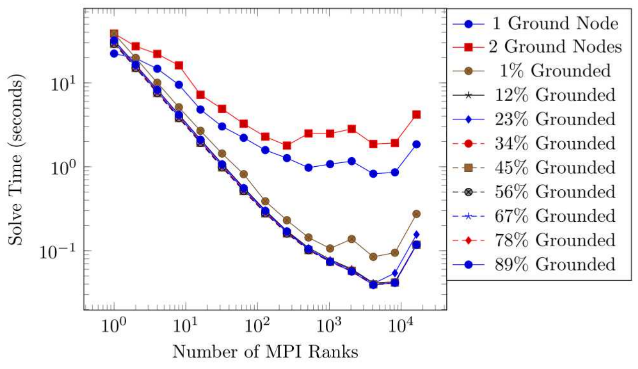

Figure 9: Scaling with different numbers of initial grounded vertices with synthetic mesh *grounded(2km)*

7.1.3 *Scaling vs. Mesh Complexity.* The sizes and numbers of mesh features and grounded vertices can impact the performance of our algorithm. To study this impact, we modified the synthetic 2km mesh to vary its characteristics.

First, in the *ground(2km)* mesh, we varied the number of vertices in the mesh that were initially grounded, from the extreme case of only one initially grounded vertex to the case typical in real ice sheet meshes where 89% of vertices are initially grounded. Figure 9 shows that as the number of initially grounded vertices increases, the runtime of our algorithm decreases, as fewer vertices' grounding state needs to be determined and grounding information is available across more processors initially. In the extreme cases with very few initially grounded vertices, strong scaling is poor because few processor have grounding information initially; most grounding information needs to propagate via communication. However, with 12% or more initially grounded vertices, all tests exhibited good strong scaling similar to the realistic 89% grounding case.

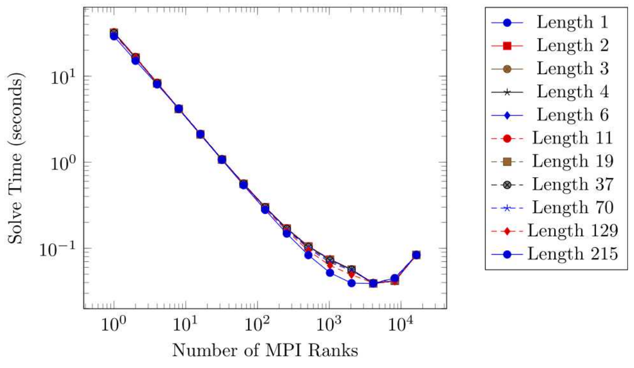

Figure 10: Scaling with different lengths of complex features using synthetic mesh *longcomplex*

Figure 10 shows our algorithm's behavior as the length of complex features increases. Varying the length of complex features from one to 215 elements shows little effect on the algorithm's runtime. Since vertices in complex features may be initially grounded, propagation through complex features is equivalent to propagation in the central ice mass. We do not measure the typical lengths of complex features in real meshes, but we can use the number of potential articulation points to get an idea. The real *2km* mesh has 389 potential articulation points; the synthetic mesh has 373 potential articulation points with a complex feature length of two.

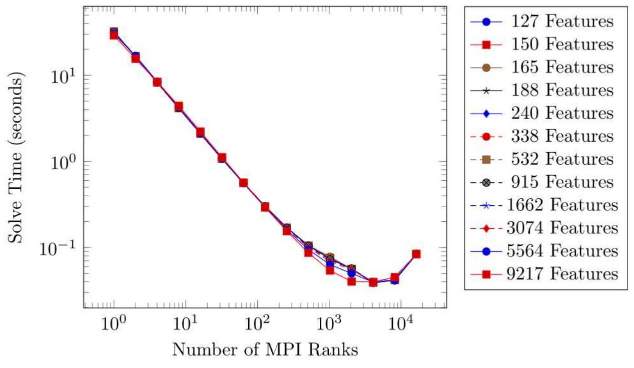

Figure 11: Scaling with different numbers of complex features with mesh *numcomplex*

Next, with mesh *numcomplex*, we varied the number of complex features in a 2km mesh from 127 to 9217. The results, shown in Figure 11, are similar to Figure 10. Adding complex features does not adversely affect our algorithm's running time, as the algorithm's runtime does not depend on the number of biconnected components. Similarly to above, we can use the number of potential articulation points to get an idea of how many complex features are realistic. There are 366 potential articulation points in the synthetic mesh that has 188 complex features, which is as close as we get to the 389 potential articulation points of the *2km* mesh.

While the lengths and number of complex features do not impact the algorithm's runtime, the length of degenerate features can have a dramatic impact. Results varying the degenerate feature length from 1 to 3000 in a 2km mesh are shown in Figure 12. For small degenerate lengths typical of real ice sheet meshes, the length of degenerate features has modest impact. But as the length grows to 3000 and above, the algorithm loses scalability. Since degenerate features do not have grounded vertices, the algorithm makes slower progress on features split over processor boundaries.

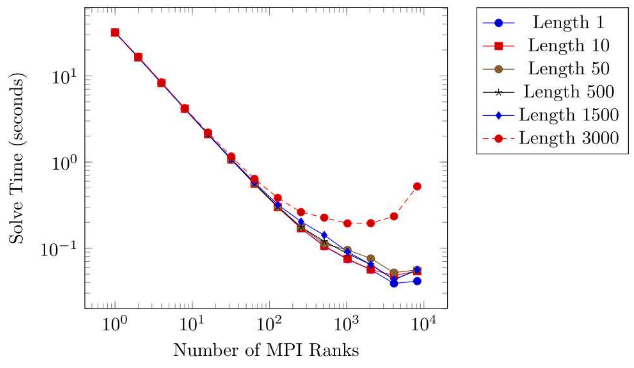

Figure 12: Scaling with different lengths of degenerate features with mesh *longdegen*

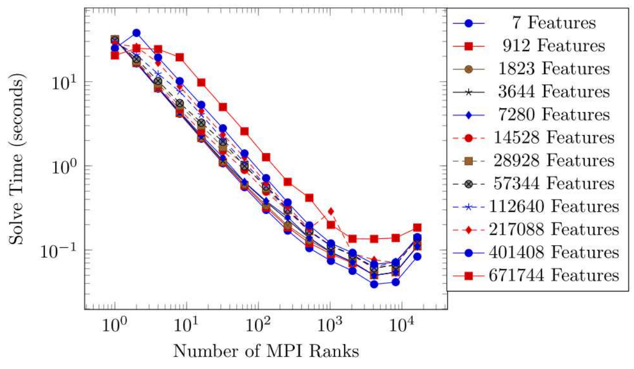

Figure 13: Scaling with different numbers of degenerate features with mesh *numdegen*

The number of degenerate features has a less dramatic effect on runtime, but as Figure 13 shows, the more degenerate features there are, the more slowly our algorithm runs. While scalability is unaffected by the number of degenerate features, the overall runtime increases with the number of degenerate features. We observe that the number of degenerate features in ice sheets typically is small. In the *1km* mesh, only 65 vertices are removed, so numbers of degenerate features over 100 are atypical of our real mesh data.

7.2 Application Results

We demonstrate the performance of our algorithm by incorporating it into the Zoltan2 graph algorithm package [2] of the Trilinos solver framework [8] and calling it from the Albany-LI component of the MPAS-Albany Land Ice (MALI) simulation code [9]. MALI is a high-fidelity, variable-resolution ice sheet model developed as part of the U.S. Department of Energy's Energy Exascale Earth System Model (E3SM). Albany-LI uses a conjugate gradient solver with a semicoarsening algebraic multigrid preconditioner [18]. The iterative solver is sensitive to degenerate features in ice sheet meshes. Thus, developers preprocessed their meshes with a degenerate-feature removal algorithm that was implemented in Matlab and run in serial before the simulation began; dynamic degerate-feature removal as the ice evolved during a simulation was not possible.

Mesh	Distributed (MPI Ranks)	Serial Matlab	Speedup
16km (real)	0.0176 s (6)	1.04 s	59×
8km (real)	0.0217 s (24)	5.65 s	260×
4km (real)	0.0414 s (96)	34.6 s	835×
2km (real)	0.0407 s (384)	245 s	6019×
1km (real)	0.0561 s (1536)	2630 s	46880×

Table 2: Execution time for our distributed method in Albany-LI, compared to the serial Matlab preprocessor.

With our new algorithm, serial preprocessing in Albany-LI is no longer needed. The mesh can be read into parallel processors, and degenerate features can be detected quickly at runtime. Our distributed implementation enables dynamic degenerate-feature removal to capture changes in the ice over the course of a simulation.

Table 2 shows the striking difference in runtime between our approach and the Matlab preprocessing approach. The preprocessing approach was run on a workstation with an Intel Xeon Gold 6146 CPU (3.20 GHz). Our distributed code was run in Albany-LI model on NERSC's Edison Cray XC30 supercomputer on varying numbers of MPI ranks. Albany-LI uses geometric partitioning (recursive inertial bisection [16]) to assign mesh elements to processors; our algorithm then used the distribution from Albany-LI. We see roughly 46,000× speedup in the highest resolution case. Moreover, our algorithm takes at most 0.4% of the time of a single simulation step. Thus, our algorithm is fast enough to be used dynamically in the simulation as needed.

8 CONCLUSIONS AND FUTURE WORK

We have proposed an algorithm for identifying degenerate features in ice sheet meshes. We showed that our distributed memory implementation is efficient enough to be used at every step of an ice sheet simulation with negligible computational overhead. Our method scales very well for real and synthetic meshes. It also behaves well on synthetic meshes generated with an extreme scale and number of complex and degenerate features.

There are many directions for future work related to our baseline algorithm. We have explored generalizations of this algorithm that find all hinge points in a mesh; this capability is of interest to many other mesh-based scientific applications, such as [6]. Similarly, we have developed an extension of our algorithm to solve the biconnectivity problem on general graphs. Future work will explore optimizing these solutions and applying them to real-world applications. Additionally, we have discussed using a similar approach to identify triconnected components in graphs. These extensions are quite promising, as efficient distributed algorithms for biconnectivity and triconnectivity are not yet present in the literature.

9 ACKNOWLEDGMENTS

This work was supported in part by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research, Scientific Discovery through Advanced Computing (SciDAC) program through the FASTMath Institute under Contract No. DE-AC02-05CH11231 at Rensselaer Polytechnic Institute and Sandia National Laboratories and through the SciDAC ProSPect project at Sandia National Laboratories. Sandia National Laboratories is a multimission laboratory managed and operated by National Technology and Engineering Solutions of Sandia, LLC., a wholly owned subsidiary of Honeywell International, Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA-0003525.

REFERENCES

[1] C.G. Baker and M.A. Heroux. 2012. Tpetra, and the Use of Generic Programming in Scientific Computing. *Scientific Programming* 20, 2 (2012), 115–128.

[2] E.G. Boman, K.D. Devine, V.J. Leung, S. Rajamanickam, L.A. Riesen, M. Deveci, and U. Catalyurek. 2012. *Zoltan2: Next-Generation Combinatorial Toolkit.* Technical Report SAND2012-9373C. Sandia National Laboratories.

[3] M. Chaitanya and K. Kothapalli. 2016. Efficient Multicore Algorithms For Identifying Biconnected Components. *Int. J. Networking & Computing* 6 (2016), 87–106.

[4] J.A. Church, P.U. Clark, A. Cazenave, J.M. Gregory, S. Jevrejeva, A. Levermann, M.A. Merrifield, G.A. Milne, R.S. Nerem, P.D. Nunn, A.J. Payne, W.T. Pfeffer, D. Stammer, and A.S. Unnikrishnan. 2013. *Sea Level Change.* Cambridge University Press, Cambridge, UK and New York, NY, USA, Chapter 13, 1137–1216.

[5] S.L. Cornford, D.F. Martin, D.T. Graves, D.F. Ranken, A.M. Le Brocq, R.M. Gladstone, A.J. Payne, E.G. Ng, and W.H. Lipscomb. 2013. Adaptive mesh, finite volume modeling of marine ice sheets. *J. Comp. Physics* 232, 1 (Jan. 2013), 529–549.

[6] D.M. Day, M.K. Bhardwaj, G.M. Reese, and J.S. Perry. 2003. Mechanism free domain decomposition. *Comp. Methods. Appl. Mech. Engrg.* 192, 7-8 (2003).

[7] C. Harrison, J. Weiler, R. Bleile, K. Gaither, and H Childs. 2015. A Distributed-Memory Algorithm for Connected Components Labeling of Simuation Data. In *Topological and Statistical Methods for Complex Data*, J. Bennett, F. Vivodtzev, and V. Pascucci (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 3–19.

[8] M. A. Heroux and J.M. Willenbring. 2012. A New Overview of The Trilinos Project. *Scientific Programming* 20, 2 (2012), 83–88.

[9] M.J. Hoffman, M. Perego, S.F. Price, W.H. Lipscomb, D. Jacobsen, I. Tezaur, A.G. Salinger, R.S. Tuminaro, and T. Zhang. 2018. MPAS-Albany Land Ice (MALI): A variable resolution ice sheet model for Earth system modeling using Voronoi grids. *Geoscientific Model Development Discussions* (2018), 1–47.

[10] J. Hopcroft and R. Tarjan. 1973. Algorithm 447: Efficient Algorithms for Graph Manipulation. *Commun. ACM* 16, 6 (June 1973), 372–378.

[11] E. Larour, H. Seroussi, M. Morlighem, and E. Rignot. 2012. Continental scale, high order, high spatial resolution, ice sheet modeling using the Ice Sheet System Model (ISSM). *Journal of Geophysical Research* 117 (2012).

[12] R.E.N. Moraes and C.C. Ribeiro. 2013. Power optimization in ad hoc wireless network topology control with biconnectivity requirements. *Computers & Operations Research* 40, 12 (2013), 3188 – 3196.

[13] ProSPect 2019. ProSPect: Probabilistic Sea Level Projections from ice sheet and earth system models. https://doe-prospect.github.io

[14] G.M. Slota and K. Madduri. 2014. Simple Parallel Biconnectivity Algorithms for Multicore Platforms. In *Int. Conf. High Performance Computing (HiPC)*.

[15] R.E. Tarjan and U. Vishkin. 1985. An Efficient Parallel Biconnectivity Algorithm. *SIAM J. Comput.* 14, 4 (1985), 862–874.

[16] V. E. Taylor and B. Nour-Omid. 1994. A Study of the Factorization Fill-in for a Parallel Implementation of the Finite Element Method. *Int. J. Numer. Meth. Engng.* 37 (1994), 3809–3823.

[17] I. Tezaur, M. Perego, A. Salinger, R. Tuminaro, and S. Price. 2015. Albany/FELIX: A Parallel, Scalable and Robust Finite Element Higher-Order Stokes Ice Sheet Solver Built for Advance Analysis. *Geoscientific Model Dev* 8, 4 (2015), 1197–1220.

[18] R. Tuminaro, M. Perego, I. Tezaur, A. Salinger, and S. Price. 2016. A Matrix Dependent/Algebraic Multigrid Approach for Extruded Meshes with Applications to Ice Sheet Modeling. *SIAM J. Sci. Comput.* 38 (2016), C504–C532.

[19] K. Wu, E. Otoo, and K. Suzuki. 2009. Optimizing two-pass connected-component labeling algorithms. *Pattern Analysis & Applications* 12, 2 (01 Jun 2009), 117–135.

[20] X. Zou, K. Wu, D. A. Boyuka Ii, D. F. Martin, S. Byna, H. Tang, K. Bansal, T. J. Ligocki, H. Johansen, and N. F. Samatova. 2015. Parallel In Situ Detection of Connected Components in Adaptive Mesh Refinement Data. In *IEEE/ACM Int Symp Cluster, Cloud & Grid Computing*.