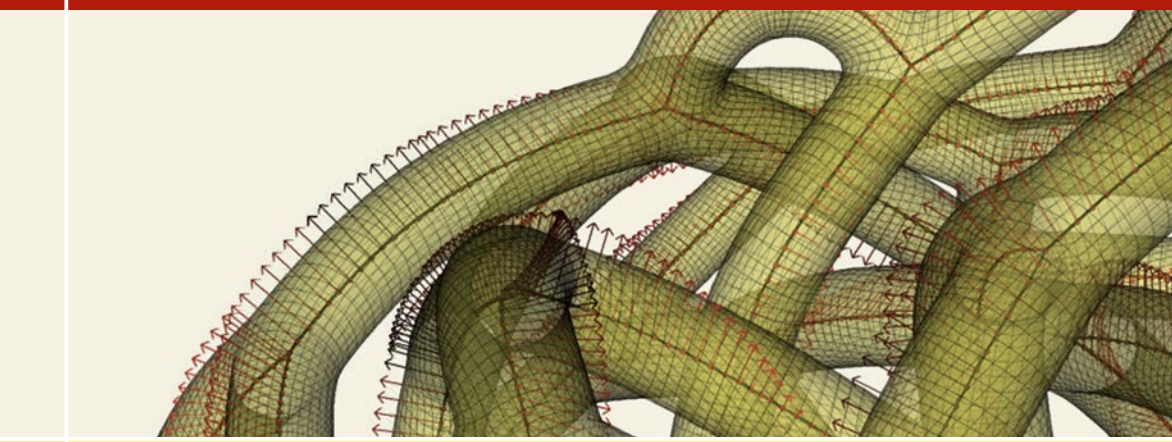
Mathematics and Visualization



Janine Bennett · Fabien Vivodtzev
Valerio Pascucci *Editors*

Topological and Statistical Methods for Complex Data

Tackling Large-Scale, High-Dimensional, and Multivariate Data Spaces

Springer

Mathematics and Visualization

More information about this series at
http://www.springer.com/series/4562

Janine Bennett • Fabien Vivodtzev •
Valerio Pascucci
Editors

Topological and Statistical Methods for Complex Data

Tackling Large-Scale, High-Dimensional, and Multivariate Data Spaces

With 120 Figures, 101 in color

Springer

Editors
Janine Bennett
Sandia National Laboratories
Livermore, CA
USA

Fabien Vivodtzev
CEA CESTA, CS 60001
Le Barp CEDEX
France

Valerio Pascucci
School of Computing and SCI Institute
University of Utah
Salt Lake City, UT
USA

ISSN 1612-3786 ISSN 2197-666X (electronic)
ISBN 978-3-662-44899-1 ISBN 978-3-662-44900-4 (eBook)
DOI 10.1007/978-3-662-44900-4
Springer Heidelberg New York Dordrecht London

Library of Congress Control Number: 2014955717

Mathematics Subject Classification (2010): 54-02, 62-02, 62-07

Printed on acid-free paper

Springer is part of Springer Science+Business Media (www.springer.com)

Preface

The continued improvements in high performance computing and high resolution sensing capabilities are resulting in data of unprecedented size and complexity. Historically topological and statistical techniques have been deployed as independent alternatives in the analysis of a variety of data types. However, the continued increases in size, dimensionality, and number of variables create new challenges that traditional approaches cannot address. New methods that leverage the mutual strengths of both topological and statistical techniques are needed to support the management, analysis and visualization of such complex data.

In an effort to characterize the current challenges and research trends, and to foster collaborations, we organized the Workshop on the Analysis of Large-scale, High-dimensional, and Multivariate Data using Topology and Statistics, held June 12–14 in Le Barp, France. Around 30 researchers from 20 European and American universities, companies, and national research laboratories were in attendance. The program comprised 18 presentations, including a keynote talk by Herbert Edelsbrunner from the Institute of Science and Technology Austria, titled "Persistent Homology: Theory and Practice." A number of interesting challenges were addressed during the workshop, with presentations covering a wide range of topics, including topological techniques for large data, high-dimensional data analysis, computational challenges, multivariate visualization and analysis techniques.

In this book, we present 16 peer-reviewed chapters, divided into 6 parts as the outcome of this workshop. Parts I and II focus on large-scale data, Parts III and IV focus on multivariate data, and Parts V and VI focus on high-dimensional data. The chapters in Part I include recent results in the area of in-situ and distributed analysis. We start with a distributed-memory algorithm for labeling connected components in simulation data (Harrison et al.), followed by a discussion of in-situ visualization in fluid mechanics (Ribes et al.). Part I concludes with a survey of recent discoveries in sublinear algorithms for extreme-scale data analysis (Seshadhri et al.). Part II focuses on the efficient representation of large functions and includes

a report on optimal general simplification of scalar fields on surfaces (Tierny et al.), an algorithm for piecewise polynomial monotonic interpolation of 2D gridded data (Allemand-Giorgis et al.), and a technique for shape analysis using real functions (Biasotti et al.). The chapters in Part III focus on structural techniques for multivariate data. This part includes a survey on 3D symmetric tensor fields that highlights what we know and where to go next (Zhang and Zhang), followed by a comparison of Pareto Sets and Jacobi Sets (Huttenberger and Garth), and a report on deformations preserving total curvature (Berres et al.). Part IV focuses on classification and visualization of vector fields, and includes a presentation of Lyapunov time for 2D Lagrangian visualization (Sadlo), followed by a survey of geometric algebra for vector field analysis and visualization (Ausoni and Frey). This part concludes with a report on a technique for computing accurate Morse-Smale complexes from gradient vector fields (Gyulassy et al.). Part V includes chapters focused on the exploration of high-dimensional models, including a presentation of high-dimensional sampling techniques (Ebeida et al.), and a report on the realization of regular maps of large genus (Razafindrazaka and Polthier). Lastly, Part VI presents recent results in the analysis of large, high-dimensional systems, and includes a technique for faster localized solving of systems of equations (Anthony et al.), followed by a system for ensemble analysis of electrical circuit simulations (Crossno et al.).

Fig. 1 Workshop participants, June 2013

In summary, this book brings together recent results from some of the most prominent and recognized leaders in the fields of statistics, topology, and computer science. The book's contents cover both theory and application, providing an overview of important key concepts and the latest research trends. The algorithms detailed in this book are broadly applicable and can be used by application scientists to glean insight from complex data sets.

Acknowledgements

The Workshop on the Analysis of Large-Scale, High-Dimensional, and Multivariate Data Using Topology and Statistics was co-organized by the French Atomic Energy Commission and Alternative Energies (CEA), Sandia National Laboratories, and Kitware, SAS. The organizing committee greatly acknowledges Julien Jomier and Charles Marion for their administrative support in preparation for the workshop, Sophie Vivodtzev for her logistical help, and thanks all of the workshop participants for making it such an enjoyable and informative event. Lastly, the editors would like to thank all of the contributors of this volume for their insightful contributions.

Livermore, CA, USA — Janine Bennett
Le Barp, France — Fabien Vivodtzev
Salt Lake City, UT, USA — Valerio Pascucci
July 2014

Contents

List of Contributors

Léo Allemand-Giorgis Inria - Laboratoire Jean Kuntzmann, University of Grenoble, Grenoble, France

Eleanor Anthony Mathematics Department, University of Mississippi, MS, USA

Chantal Oberson Ausoni Sorbonne Universités, UPMC Univ Paris 06, Institut du Calcul et de la Simulation, Paris, France

Muhammad A. Awad Department of Naval Architecture and Marine Engineering, Alexandria University, Alexandria, Egypt

Janine C. Bennett Sandia National Laboratories, Livermore, CA, USA

Anne Berres University of Kaiserslautern, Kaiserslautern, Germany

Harsh Bhatia SCI Institute, University of Utah, Salt Lake City, UT, USA

Silvia Biasotti CNR-IMATI, Genova, Italy

Ryan Bleile The University of Oregon, Eugene, OR, USA

Georges-Pierre Bonneau Inria - Laboratoire Jean Kuntzmann, University of Grenoble, Grenoble, France

Peer-Timo Bremer Lawrence Livermore National Laboratory, Livermore, CA, USA

Andrea Cerri CNR-IMATI, Genova, Italy

Hank Childs The University of Oregon and Lawrence Berkeley National Laboratory, Eugene, OR, USA

Patricia J. Crossno Sandia National Laboratories, Albuquerque, NM, USA

Andrew A. Davidson Department of Electrical and Computer Engineering, University of California, Davis, CA, USA

Mohamed S. Ebeida Computing Research, Sandia National Laboratories, Albuquerque, NM, USA

Bianca Falcidieno CNR-IMATI, Genova, Italy

Pascal Frey Sorbonne Universités, UPMC Univ Paris 06, Institut du Calcul et de la Simulation, Paris, France

Sorbonne Universités, UPMC Univ Paris 06, Laboratoire Jacques-Louis Lions, Paris, France

Yvan Fournier EDF R&D, Chatou, France

Kelly Gaither The University of Texas at Austin (Texas Advanced Computing Center), Austin, TX, USA

Christoph Garth University of Kaiserslautern, Kaiserslautern, Germany

Sheridan Grant Mathematics Department, Pomona College, Claremont, CA, USA

Peter Gritzmann Fakultät für Mathematik, Technische Universität München, München, Germany

David Günther Institut Mines-Telecom, Telecom ParisTech, CNRS LTCI, Paris, France

Attila Gyulassy SCI Institute, University of Utah, Salt Lake City, UT, USA

Hans Hagen University of Kaiserslautern, Kaiserslautern, Germany

Stefanie Hahmann Inria - Laboratoire Jean Kuntzmann, University of Grenoble, Grenoble, France

Cyrus Harrison Lawrence Livermore National Laboratory, Livermore, CA, USA

Ming-Yu Hsieh Sandia National Laboratories, Albuquerque, NM, USA

Lars Huettenberger University of Kaiserslautern, Kaiserslautern, Germany

Warren L. Hunt Sandia National Laboratories, Albuquerque, NM, USA

Julien Jomier Kitware, Villeurbanne, France

Benjamin Lorendeau EDF R&D, Clamart, France

Ahmed H. Mahmoud Department of Naval Architecture and Marine Engineering, Alexandria University, Alexandria, Egypt

Shawn Martin Sandia National Laboratories, Albuquerque, NM, USA

Scott A. Mitchell Computing Research, Sandia National Laboratories, Albuquerque, NM, USA

John D. Owens Department of Electrical and Computer Engineering, University of California, Davis, CA, USA

Valerio Pascucci School of Computing and SCI Institute, University of Utah, Salt Lake City, UT, USA

Anjul Patney Department of Electrical and Computer Engineering, University of California, Davis, CA, USA

Ali Pinar Sandia National Laboratories, Livermore, CA, USA

Konrad Polthier Freie Universität Berlin, Berlin, Germany

Faniry Razafindrazaka Freie Universität Berlin, Berlin, Germany

Alejandro Ribés EDF R&D, Clamart, France

J. Maurice Rojas Mathematics Department, Texas A&M University, College Station, TX, USA

Filip Sadlo University of Stuttgart, Stuttgart, Germany

C. Seshadhri Sandia National Laboratories, Livermore, CA, USA

Timothy M. Shead Sandia National Laboratories, Albuquerque, NM, USA

Milosz A. Sielicki Sandia National Laboratories, Albuquerque, NM, USA

Michela Spagnuolo CNR-IMATI, Genova, Italy

David Thompson Kitware, Inc., Carrboro, NC, USA

Julien Tierny CNRS LIP6, UMPC, Sorbonne Universites, Paris, France

CNRS LTCI, Telecom ParisTech, 46 Rue Barrault, 75013 Paris, France

Stanley Tzeng Department of Computer Science, University of California, Davis, CA, USA

Fabien Vivodtzev CEA: French Atomic Energy Commission and Alternative Energies, Le Barp, France

Jordan Weiler The University of Oregon, Eugene, OR, USA

Eugene Zhang School of Electrical Engineering and Computer Science, 2111 Kelley Engineering Center, Oregon State University, Corvallis, OR, USA

Yue Zhang School of Electrical Engineering and Computer Science, 3117 Kelley Engineering Center, Oregon State University, Corvallis, OR, USA

Part I
Large-Scale Data Analysis: In-Situ and Distributed Analysis

A Distributed-Memory Algorithm for Connected Components Labeling of Simulation Data

Cyrus Harrison, Jordan Weiler, Ryan Bleile, Kelly Gaither, and Hank Childs

1 Introduction

Parallel scientific simulations running on today's state of the art petascale computing platforms generate massive quantities of high resolution mesh-based data. Scientists often analyze this data by eliminating portions and visualizing what remains, through operations such as isosurfacing, selecting certain materials and discarding the others, isolating hot spots, etc. These approaches can generate complex derived geometry with intricate structures that require further techniques for effective analysis, especially in the context of massive data.

In these instances, representations of the topological structure of a mesh is often helpful (Fig. 1). Specifically, a labeling of the connected components in a mesh provides a simple and intuitive topological characterization of which parts of the

C. Harrison
Lawrence Livermore National Laboratory, Livermore, CA, USA
e-mail: cyrush@llnl.gov

J. Weiler • R. Bleile
The University of Oregon, Eugene, OR, USA
e-mail: jweiler@cs.uoregon.edu; rbleile@cs.uoregon.edu

K. Gaither
The University of Texas at Austin (Texas Advanced Computing Center), Austin, TX, USA
e-mail: kelly@tacc.utexas.edu

H. Childs (✉)
The University of Oregon, Eugene, OR, USA

Lawrence Berkeley National Laboratory, Berkeley, CA, USA
e-mail: hchilds@lbl.gov; hank@cs.uoregon.edu

J. Bennett et al. (eds.), *Topological and Statistical Methods for Complex Data*,
Mathematics and Visualization, DOI 10.1007/978-3-662-44900-4_1

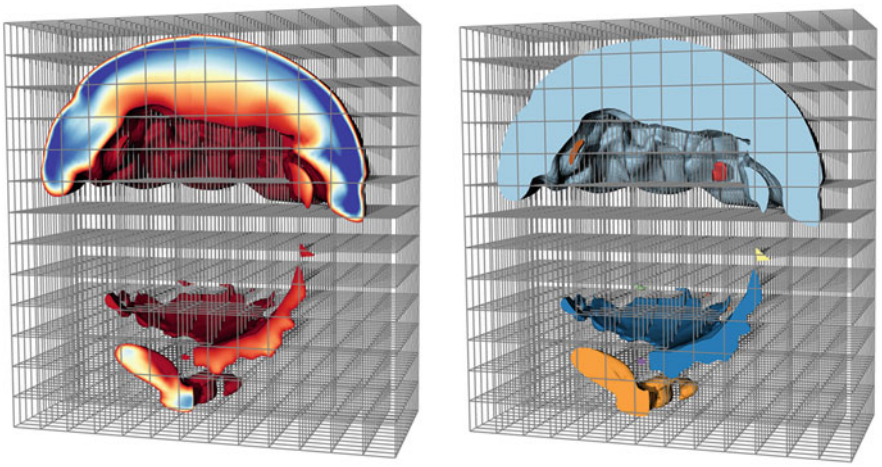


Fig. 1 (*Left*) Sub-volume mesh extracted from a 21 billion cell structured grid decomposed across 2,197 processors. (*Right*) Sub-volume mesh colored by the results from the connected components labeling algorithm described in this chapter

mesh are connected to each other. These unique sub-meshes contain a subset of cells that are directly or indirectly connected via series of cell abutments.

The global nature of connectivity poses a challenge in distributed-memory parallel environments, which are the most common setting for analyzing massive data. This is because massive data sets are typically too large to fit into the memory of a single processor, so pieces of the mesh are distributed across processors. Cells comprising connected sub-meshes may span any of the processors, but the relationships of how cells abut across processors frequently has to be derived. To deal with this problem, sophisticated techniques to resolve connectivity are necessary.

This chapter explores an algorithm that operates on both structured and unstructured meshes and scales well even with very large data, as well as its underlying performance characteristics. The algorithm executes in multiple stages, ultimately constructing a unique label for each connected component and marking each vertex with its corresponding connected component label. This final labeling enables analyses such as: calculation of aggregate quantities for each connected component, feature based filtering of connected components, and calculation of statistics on connected components.

In short, the algorithm provides a useful tool for domain scientists with applications where physical structures, such as individual fragments of a specific material, correspond to the connected components contained in a simulation data set. This chapter presents the algorithm (Sect. 4), results from a weak scaling performance study (Sect. 5), and further analysis of the slowest phase of the algorithm (Sect. 6).

2 Related Work

The majority of research to date in connected components algorithms has been focused on computer vision and graph theory applications. This previous research is useful for contributing high-level ideas, but ultimately the algorithms themselves are not directly applicable to the problem considered here. Computer vision algorithms typically depend on the structured nature of image data, and so cannot be easily applied to unstructured scientific data. Graph theory algorithms are more appropriate, since the cell abutment relationships in an unstructured mesh can be encoded as an undirected graph representation. But this encoding results in a very sparse graph, with the edges having special properties—neighboring cells typically reside on the same processing elements, although not always—that graph theory algorithms are not optimized for. For more discussion of these algorithms, we refer the reader to [11]. That said, previous graph theory research on connected components has used the Union-find algorithm [8], which is also used for the algorithm described in this chapter. Further, the Union-find algorithm and data structures have been used in topology before, for the efficient construction of Contour Trees [5], Reeb Graphs [17], and segmentations [2, 3]. Union-find is discussed further in Sect. 3.1.

The algorithm described in this chapter is intended for distributed-memory parallelism. With this technique, Processing Elements (PEs)—instances of a program—are run on each node, or on each core of a node. By using multiple nodes, the memory available to the program is larger, allowing for processing of data sets so large that they cannot fit into the memory of a single node. Popular end user visualization tools for large data, such as ParaView [1] and VisIt [7], follow this distributed-memory parallelization strategy. Both of these tools instantiate identical visualization modules on each PE, and the PEs are only differentiated by the subportion of the larger data set they operate on. The tools rely on the data set being decomposed into pieces (often referred to as domains), and they partition these pieces over their PEs. This approach has been shown to be effective; VisIt performed well on meshes with trillions of cells using tens of thousands of PEs [6]. The algorithm described in this chapter follows the strategy of partitioning data over the PEs and has been implemented as a module inside VisIt. It uses the Visualization ToolKit (VTK) library [15] to represent mesh-based data, as well as its routines for identifying cell abutment within a piece. Finally, we note that other topology algorithms have also been ported to a distributed-memory parallel environment, specifically segmentations [18] and merge trees [13].

3 Algorithm Building Blocks

This section describes three fundamental building blocks used by the algorithm. The first is the serial Union-find algorithm, which efficiently identifies and merges connected components. The second is binary space partitioning trees, which enable

efficient computation of mesh intersections across PEs. The third is the concepts of exteriors cells and ghost data, which significantly accelerate the algorithm.

3.1 Union-Find

The Union-find algorithm enables efficient management of partitions. It provides two basic operations: UNION and FIND. The UNION operation creates a new partition by merging two subsets from the current partition. The FIND operation determines which subset of a partition contains a given element.

To efficiently implement these operations, relationships between sets are tracked using a disjoint-set forest data structure. In this representation, each set in a partition points to a root node containing a single representative set used to identify the partition. The UNION operation uses a union-by-rank heuristic to update the root node of both partitions to the representative set from the larger of the two partitions. The FIND operation uses a path-compression heuristic which updates the root node of any traversed set to point to the current partition root. With these optimizations each UNION or FIND operation has an amortized run-time of $O(\alpha(N))$ where N is the number of sets and $\alpha(N)$ is the inverse Ackermann function [16]. $\alpha(N)$ grows so slowly that it is effectively less than four for all practical input sizes. The disjoint-set forest data structure requires $O(N)$ space to hold partition information and the values used to implement the heuristics. The heuristics used to gain efficiency rely heavily on indirect memory addressing and do not lend themselves to a direct distributed-memory parallel implementation.

3.2 Binary Space Partitioning (BSP)

A binary space partitioning (BSP) [10] divides two- or three-dimensional space into a fixed number of pieces. BSPs are used in the connected components labeling algorithm described in this chapter to determine if a component on one PE abuts a component on another PE (meaning they are both actually part of a single, larger component). The BSP is constructed so that there is a one-to-one correspondence between the PEs and the pieces of the BSP tree. Explicitly, if there are N PEs, then the BSP will partition space into N pieces and each PE will be responsible for one piece. The PEs then relocate their cells according to the BSP; each cell is assigned a piece from the BSP based on its partition, and then that cell is sent to the corresponding PE.

It is important that the BSP is balanced, meaning that each piece has approximately the same number of cells. If disproportionately many cells fall within one piece, then its PE may run out of memory when the cells are relocated. As a result, the PEs must examine the cells and coordinate when creating the BSP.

The BSP construction and cell relocation can be very time consuming. More discussion of their complexity can be found at the end of this chapter (Sect. 6).

3.3 Exterior Cells and Ghost Cells

Exterior cells and ghost cells are used by the algorithm to reduce the amount of data needed to coordinate between PEs. Both techniques identify cells that are on the boundary of a PE's piece. Ghost cells identify exactly the cells on the boundary, while exterior cells identify a superset of the boundary cells.

Exterior cells are the cells that lie along the exterior of a volume, which does not necessarily strictly correspond to the exterior of the PE's piece. Consider the example of removing a material: the exterior cells of the remainder will likely have a portion along the PE piece boundary, but it will also likely have a portion along the interior of the piece, where the material interface lies.

"Ghost cells" are the result of placing a redundant layer of cells along the boundary of each domain. Ghost cells are either pre-computed by the simulation code and stored in files or calculated at run-time by the analysis tool. They are typically created to prevent interpolation artifacts at piece boundaries. More discussion of ghost cells can be found in [7] and [12].

Ghost cells are also useful for connected components labeling. They identify the location of the boundary of a piece and provide information about the state of abutting cells in a neighboring piece. Note that the results discussed in this chapter uses ghost cells that are generated at run-time, using the collective pattern described in [7], not the streaming pattern described in [12].

4 Algorithm

The algorithm identifies the global connected components in a mesh using five phases. It first identifies which pieces are at the boundary (Phase 1). It then identifies the connected components local to each PE (Phase 2) and then creates a global labeling across all PEs (Phase 3). It next determines which components span multiple PEs (Phase 4). Finally, it merges the global labels to produce a consistent labeling across all PEs (Phase 5). This final labeling is applied to the mesh to create per-cell labels which map each cell to the corresponding label of the connected component it belongs to. In terms of parallel considerations, Phases 1 and 2 are embarrassing parallel, Phase 3 is a trivial communication, Phase 4 has a large all-to-all communication, followed by embarrassingly parallel work, and Phase 5 has trivial communication following by more embarrassingly parallel work (Fig. 2).

Phase 1: Identify cells at PE boundaries: The goal of this phase is to identify cells that abut the spatial boundary of the data contained on each PE, which

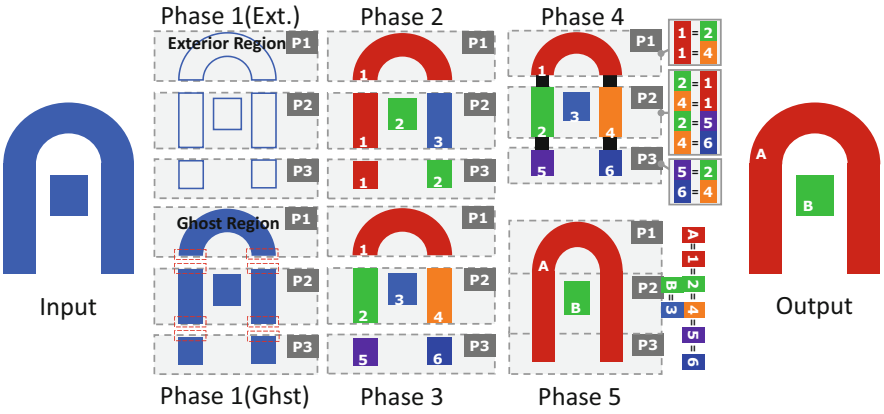


Fig. 2 Example illustrating the five phases of the algorithm on a simple data set decomposed onto three PEs. Phase 1 has two variants, and both variants are shown—"exterior cells" on the *top* and "ghost cells" on the *bottom*

enables reduced communication in Phase 4. We consider two methods for doing this: ghost data and exterior cells. The ghost data option marks the minimum number of cells to be considered, since ghost data always lies along the PE boundary. The exterior cells option marks more cells, since some cells are external to a component, but interior to the PE boundary; these cells cannot be distinguished and thus must be processed unnecessarily.

Ghost cells are not present in all data sets. The algorithm deployed in VisIt uses the ghost data option when ghost data is present, and falls back to the exterior cells option when it is not. However, we point out that the study described in this chapter shows the two variants to have very similar performance.

Phase 1, ghost cells option: Ghost cells are useful because they are always adjacent to boundary cells; finding the cells adjacent to ghost cells is equivalent to finding the list of cells on the boundary. Note that ghost cells cannot be used directly to represent PE boundaries since they themselves lack ghost data. For example, an isosurface operation on a ghost cell lacks the requisite additional ghost data to perform interpolation, and therefore does not have sufficient information to generate the correct contour. For this reason, all ghost cells are removed after the boundary is identified.

In pseudocode:

```
For each cell c:
  boundary[c] = false
  if (not IsGhostCell(c))
    For each neighbor n of c:
       if IsGhostCell(n):
            boundary[c] = true
RemoveGhostCells()
```

Phase 1, exterior cells option: Again, exterior cells are the cells that are on the exterior of the components. Only the cells on the boundary need to be considered in Phase 4, and these cells are a superset of the cells on the boundary. However, they are a subset of all cells and discarding the cells in the interior of the components substantially improves Phase 4 performance.

The benefit of this approach varies based on the data set. If a component has a high surface area to volume ratio, then proportionally less cells will be in the interior and the number of cells discarded is less. Further, the proportion of exterior cells that are not on the boundary compared to those that are on the boundary is data dependent. That said, a factor of 4× to 10× reduction is typical in the number of cells processed in Phase 4 by focusing on exterior cells.

The exterior cells can be calculated by using a standard "external faces" algorithm. For each face, look at the number of cells incident to that face (or each edge in two dimensions). The faces that have one cell incident to it are exterior, and so those cells are marked as exterior.

Phase 2: Identify components within a PE: The purpose of this phase is for each PE to label the connected components for its portion of the data. As mentioned in Sect. 3.1, the Union-find algorithm efficiently constructs a partition through an incremental process. A partition with one subset for each point in the mesh is used to initialize the Union-find data structure. It then traverses the cells in the mesh. For each cell, it identifies the points incident to that cell. Those points are then merged ("unioned") in the Union-find data structure.

In pseudocode:

```
UnionFind uf;
For each point p:
   uf.SetLabel(p, GetUniqueLabel())
For each cell c:
   pointlist = GetPointsIncidentToCell(c)
   p0 = pointlist[0]
   For each point p in pointlist:
      if (uf.Find(p0) != uf.Find(p))
         uf.Union(p0, p)
```

The execution time of this phase is dependent on the number of union operations, the number of find operations, and the complexity of performing a given union or find. The number of finds is equal to the sum over all cells of how many points are incident to that cell. Practically speaking, the number of points per cell will be small, for example eight for a hexahedron. Thus the number of finds is proportional to the number of cells. Further, the number of unions will be less than the number of finds. Finally, although the run-time complexity of the Union-find algorithm is nuanced, each individual union or find is essentially a constant time operation, asymptotically-speaking. Thus the overall execution time of this phase for a given PE is proportional to the number of cells contained on that PE.

Phase 3: Component re-label for cross-PE comparison: At the end of Phase 2, on each PE, the components within that PE's data have been identified. Each of these components has a unique local label and the purpose of Phase 3 is to transform these identifiers into unique global labels. This will allow the

algorithm to perform parallel merging in subsequent phases. Phase 3 actually has two separate re-labelings. First, since the Union-find may create non-contiguous identifiers, it transforms the local labels such that the numbering ranges from 0 to N_P, where N_P is the total number of labels on Processing Element P. For later reference, we denote $N = \sum N_P$ as the total number of labels over all PEs. Second, the algorithm constructs a unique labeling across the PEs by adding an offset to each range. It does this by using the PE rank and determining how many total components exist on lower PE ranks. This number is then added to component labels. At the end of this process, PE 0 will have labels from 0 to $N_0 - 1$, PE 1 will have labels from N_0 to $N_0 + N_1 - 1$ and so on. Finally, a new scalar field is placed on the mesh, associating the global component label with each cell.

Phase 4: Merging of labels across PEs:

At this point, when a component spans multiple PEs, each PE's sub-portion has a different label. The goal of Phase 4 is to identify that these sub-portions are actually part of a single component and merge their labels. The algorithm does this by re-distributing the data using a BSP (see Sect. 3.2) and employing a Union-find strategy to locate abutting cells that have different labels. The data communicated involves cells, including their current label from Phase 3, although only the cells that lie on the boundary are needed to locate abutments. The cells identified in Phase 1 are used in the search process, but the cells known not to be on the boundary are excluded, saving about an order of magnitude in the number of cells considered.

The Union-find strategy in Phase 4 has four key distinctions from the strategy described in Phase 2:

- The labeling is now over cells (not points), which is made possible by the scalar field added in Phase 3.
- The algorithm now merges based on cell abutment, as opposed to Phase 2, where cells were merged if it had two points incident. This abutment captures any spatial overlap, be it at a face, a vertex, or one cell "poking" into another.
- Each cell is initialized with the unique global identifier from the scalar field added in Phase 3, as opposed to the arbitrary unique labeling imposed in Phase 2.
- Whenever a union operation is performed, it records the details of that union for later use in establishing the final labeling.

In pseudocode:

```
CreateBSP()
UnionFind uf;
For each cell c:
  uf.SetLabel(c, label[c])
For each cell c:
  For each neighbor n of c:
       if (uf.Find(c) != uf.Find(n))
            uf.Union(n, c)
            RecordMerge(n, c)
```

After the union list is created, the re-distributed data is discarded and each PE returns to operating on its original data.

Phase 5: Final assignment of labels: Phase 5 incorporates the merge information from Phase 4 with the labeling from Phase 3. Recall that in Phase 3 the algorithm constructed a globally unique labeling of per-PE components and denoted N as the total number of labels over all PEs. The final labeling of components is constructed as follows:

- After Phase 4, each PE is aware of the unions it performed, but not aware of unions on other PEs. However, to assign the final labels, each PE must have the complete list of unions. So Phase 5 begins by broadcasting ("all-to-all") each PE's unions to construct a global list.
- Create a Union-find data structure with N entries, each entry having the trivial label.

```
UnionFind uf
For i in 0 to N-1:
   uf.SetLabel(i, i)
```

- Replay all unions from the global union list.

```
For union in GlobalUnionList:
   uf.Union(union.label1, union.label2)
```

  The Union-find data structure can now be treated as a map. Its "Find" method transforms the labeling we constructed in Phase 3 to a unique label for each connected component.
- Use the "Find" method to transform the labeling from the scalar array created in Phase 3 to create a final labeling of which connected component each cell belongs to.

```
For each cell c:
  val[c] = uf.Find(val[c])
```

- Optionally transform the final labeling so that the labels range from 0 to $N_C - 1$, where N_C is the total number of connected components.

Note that the key to this construction is that every PE is able to construct the same global list by following the same set of instructions. They essentially "replay" the merges from the global union list in identical order, creating an identical state in their Union-find data structure.

5 Performance Study

The efficiency of the algorithm was studied with a performance study that used weak scaling on concurrency levels up to 2,197 cores (and 2,197 PEs) with data set sizes up to 21 billion cells. The study used Lawrence Livermore National Laboratory's

"Edge" machine, a 216 node Linux cluster with each node containing two 2.8 GHz six-core Intel Westmere processors. The system has 96 GB of memory per node (8 GB per core) and 20 TB of aggregate memory.

5.1 Problem Setup

The data input came from a core-collapse supernova simulation produced by the Chimera code [4]. This data set was selected because it contains a scalar entropy field with large components that span many PEs. A data set was generated for each concurrency, using upsampling to ensure each PE would operate on a fixed number of cells. Interval volumes—the volume that lies between two isosurfaces, one with the "minimum" isovalue and one with the "maximum" isovalue—were extracted from the upsampled structured grid to create an unstructured mesh as input to the connected components algorithm.

The following factors were varied:

- Concurrency (12 options): Levels varied from 8 cores (2^3) to 2,197 cores (13^3).
- Data sets (2 options): Data sizes with one million cells per PE and 10 million cells per PE were run. Table 1 outlines the data sizes for the latter case.
- Phase 1 Variant (three options): Both the ghost cell and exterior cells variants of the algorithm were tested, as well as a variant with no identification of cells at PE boundaries (i.e., no Phase 1), since this variant was presented in previous work.

The cross product of tests were run, meaning $12 \times 2 \times 3 = 72$ tests.

Figure 1 shows rendered views of the largest interval volume data set used in the scaling study and its corresponding labeling result.

Table 1 Scaling study data set sizes for the runs with 10 million cells per PE. The study targeted PE counts equal to powers of three to maintain an even spatial distribution after upsampling. The highest power of three PE count available on the test system was $13^3 = 2197$ PEs, so PE counts from 8 to 2,197 and initial mesh sizes from 80 million to 21 billion cells were studied. The interval volume operation creates a new unstructured mesh consisting of portions of approximately 1/8th of the cells from the initial mesh, meaning that each core has, on average, 1.2 million cells

Num cores	Input mesh size	Interval vol. mesh size	Num cores	Input mesh size	Interval vol. mesh size
$2^3 = 8$	80 million	10.8 million	$8^3 = 512$	5.12 billion	621.5 million
$3^3 = 27$	270 million	34.9 million	$9^3 = 729$	7.29 billion	881.0 million
$4^3 = 64$	640 million	80.7 million	$10^3 = 1{,}000$	10 billion	1.20 billion
$5^3 = 125$	1.25 billion	155.3 million	$11^3 = 1{,}331$	13.3 billion	1.59 billion
$6^3 = 216$	2.16 billion	265.7 million	$12^3 = 1{,}728$	17.2 billion	2.06 billion
$7^3 = 343$	3.43 billion	418.7 million	$13^3 = 2{,}197$	21.9 billion	2.62 billion

5.2 Results

Figures 3 and 4 present the timing results from the cross product of tests. As expected, the timings for Phases 2, 3, and 5 are consistent between all variants of the algorithm. At 125 PEs and beyond, the largest subset of the interval volume on a single PE approaches the maximum size, either 1 million or 10 million cells depending on the study. For this reason, weak scaling for Phase 2 is expected. This is confirmed by flat timings for Phase 2 beyond 125 PEs. The ghost cell variant and exterior cell variant perform comparably in Phase 4, and both significantly outperform the variant with no boundary selection. These timings demonstrate the benefit of identifying per-PE spatial boundaries. The small amount of additional

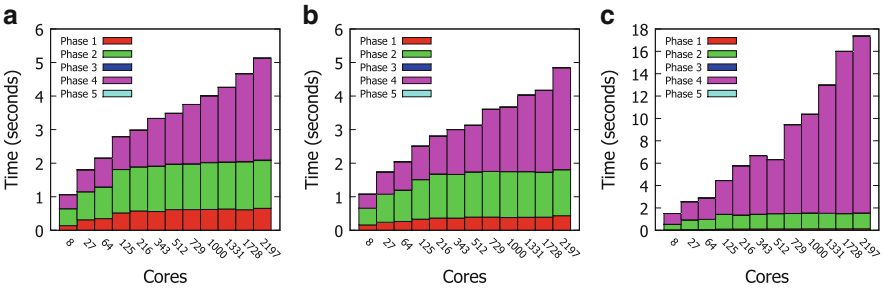


Fig. 3 Scaling study using one million cells per PE. Each figure corresponds to a variant for running Phase 1 and plots the timings for the five phases for each of the 12 concurrency levels for that variant. (**a**) shows the ghost cells variant. (**b**) shows the exterior cells variant. (**c**) shows the variant with no reduction of cells exchanged, which was presented in previous work and is included for comparative purposes. Figures **a** and **b** are very similar in performance and are on similarly scaled axes. Figure **c** performs significantly slower and is on a different scale. The time spent in Phase 1 for the ghost cell and exterior cell variants—which is not present in the third variant—leads to substantial savings in Phase 4. Phases 3 and 5 are negligible across all versions of the algorithm

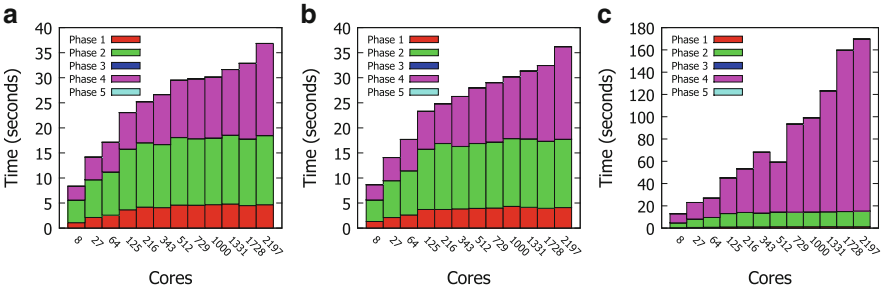


Fig. 4 Scaling study similar to that described in Fig. 3, except using 10 million cells per PE. As expected, the performance is proportional for the unoptimized variant of Phase 1, and nearly proportional for the two optimized variants

Table 2 Cells exchanged in Phase 4 for each variant for the 10 million cell per PE test variant. The total listed for each variant is the percentage of the total number of cells for that concurrency level

Number of cores	Total cells	Ghost cell variant (%)	Exterior cell variant (%)	No optimization (%)
$2^3 = 8$	10.8M	1.2	11.1	100
$3^3 = 27$	34.9M	2.3	9.1	100
$4^3 = 64$	80.8M	2.4	7.7	100
$5^3 = 125$	155M	2.6	6.9	100
$6^3 = 216$	266M	2.6	6.2	100
$7^3 = 343$	419M	2.7	5.7	100
$8^3 = 512$	622M	2.7	5.4	100
$9^3 = 729$	881M	2.7	5.1	100
$10^3 = 1{,}000$	1.2B	2.7	4.9	100
$11^3 = 1{,}331$	1.6B	2.7	4.6	100
$12^3 = 1{,}728$	2.1B	2.7	4.5	100
$13^3 = 2{,}197$	2.6B	2.7	4.3	100

Table 3 Information about the largest component and about the number of global union pairs transmitted in Phase 5. There is a strong correlation the two, and the Pearson correlation coefficient between them is 99.4 %. The percentage of cores spanned by the largest component converges to slightly less than 25 %

Num cores	Num cells in largest comp.	Num cores spanned	Num global union pairs
$2^3 = 8$	10.1 million	4	16
$3^3 = 27$	132.7 million	17	96
$4^3 = 64$	176.7 million	29	185
$5^3 = 125$	146.6 million	58	390
$6^3 = 216$	251.2 million	73	666
$7^3 = 343$	396.4 million	109	1,031
$8^3 = 512$	588.9 million	157	1,455
$9^3 = 729$	835.5 million	198	2,086
$10^3 = 1{,}000$	11.14 billion	254	2,838
$11^3 = 1{,}331$	11.51 billion	315	3,948
$12^3 = 1{,}728$	11.96 billion	389	5,209
$13^3 = 2{,}197$	12.49 billion	476	6,428

preprocessing time required for Phase 1 creates significant reduction in the number of cells transmitted and processed in Phase 4, as shown in Table 2.

Although the amount of data per PE is fixed, the number of connectivity boundaries in the interval volume increases as the number of PEs increases. This is reflected by the linear growth in both the number of union pairs transmitted in Phase 5 and the number of cores spanned by the largest connected component (See Table 3).

6 BSP Generation

Phase 4 is the slowest part of our algorithm, and BSP generation is a significant portion of that time. In this section, we consider the techniques and performance considerations for BSP generation. Section 6.1 describes Recursive Coordinate Bisection (RCB), a technique for generating BSPs. RCB requires a data structure for doing spatial searches; Sect. 6.2 explores the relative advantages of octrees and interval trees.

6.1 Recursive Coordinate Bisection (RCB)

RCB [14] is an algorithm that takes a list of points and a target number of regions and generates a BSP that partitions space such that every region in the BSP contains approximately the same number of points. The list of points is distributed across the PEs, so the RCB algorithm must operate in parallel. Again, in this context, the target number of partitions is the number of PEs, so that each PE can own one region. It is important that each region contains approximately the same number of points, otherwise a PE might receive so many points that it will run out of memory.

RCB starts by choosing a "pivot" to divide space. In the first iteration, the pivot is a plane along the x-axis (e.g., "X = 2"). The pivot should divide space such that half of the point list is on either side of the plane. The algorithm then recurses. It embarks to find a plane in the y-axis for each of the two regions created by the initial split. These planes may be at different y locations. The algorithm continues iterating over the regions, splitting over X, then Y, then Z, then X again, and so on. At each step, it tries to split the region so that half of the points are on each side of the plane (with modifications for non-powers of two). This process is illustrated in Fig. 5.

A key issue for RCB is pivot selection. The pivot selection requires iteration, with each proposed pivot requiring examination of how many points lie on either side. Previous RCB constructions [14] have used randomized algorithms. These

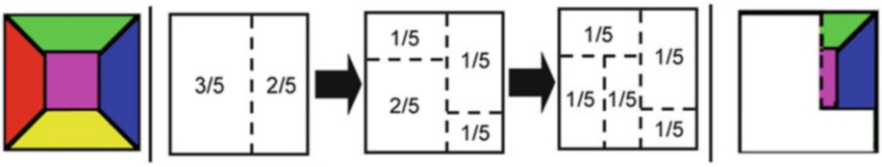


Fig. 5 RCB construction of a BSP-tree in a distributed memory setting. On the *left*, the decomposition of the original mesh. Assume the *red* portions are on PE 1, *blue* on 2, and so on. The iterative strategy starts by dividing in X, then in Y, and continues until every region contains approximately $1/N_{PEs}$ of the data. Each PE is then assigned one region from the partition and the data is communicated so that every PE contains all data for its region. The data for PE 3 is shown on the *far right*