The Evolution of a Hierarchical Partitioning Algorithm for Large-Scale Scientific Data: Three Steps of Increasing Complexity

Chuck Baldwin, Tina Eliassi-Rad, Ghaleb Abdulla, Terence Critchlow
Lawrence Livermore National Laboratory
baldwin5@llnl.gov, eliassi@llnl.gov, abdulla1@llnl.gov, critchlow@llnl.gov

Abstract

As scientific data sets grow exponentially in size, the need for scalable algorithms that heuristically partition the data increases. In this paper, we describe the three-step evolution of a hierarchical partitioning algorithm for large-scale spatio-temporal scientific data sets generated by massive simulations. The first version of our algorithm uses a simple top-down partitioning technique, which divides the data by using a four-way bisection of the spatio-temporal space. The shortcomings of this algorithm lead to the second version of our partitioning algorithm, which uses a bottom-up approach. In this version, a partition hierarchy is constructed by systematically agglomerating the underlying Cartesian grid that is placed on the data. Finally, the third version of our algorithm utilizes the intrinsic topology of the data given in the original scientific problem to build the partition hierarchy in a bottom-up fashion. Specifically, the topology is used to heuristically agglomerate the data at each level of the partition hierarchy. Despite the growing complexity in our algorithms, the third version of our algorithm builds partition hierarchies in less time and is able to build trees for larger size data sets as compared to the previous two versions.

1. Three hierarchical partitioning algorithms

Scalable algorithms are needed to partition tera-scale data sets [1, 5]. This is especially true in scientific domains, where sizes of the data sets have grown exponentially in recent years. We describe the evolution of a hierarchical partitioning algorithm for large-scale scientific data sets. Specifically, large-scale simulation programs produce our data sets in mesh format. A data set in mesh format consists of interconnected grids of small zones, in which data points are stored. Figure 1 depicts the mesh produced from an astrophysics simulation of a star in its mid-life. Mesh data usually varies with time, consists of multiple dimensions (*i.e.*, variables), and can contain irregular grids. Musick and Critchlow provide a nice introduction to scientific mesh data [4].

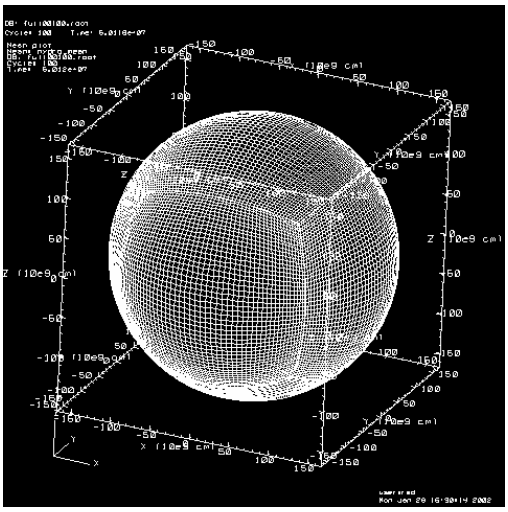

Figure 1. A Mesh Data Set Representing a Star

The first and simplest version of our partitioning algorithm employs a top-down partitioning technique by performing a four-dimensional bisection on the spatio-temporal space. The major advantage of this approach is the generation of a global decomposition of the data. However, this global partitioning comes with three major drawbacks. First, it is computationally too expensive to scale well to tera-byte data sets. This is largely due to its need to convert a mesh data file from its original simulation-specific format into a consistent vector-based representation. Second, it is not able to capture the information stored in the topology of a mesh data set. Lastly, the bisection procedure works best when there is a uniform density of grid cells throughout the whole problem domain. Typically, however, our domains have complex structures such as non-uniform distributions of grid cells, irregular boundaries, and unusual topologies. Figure 2 shows two examples of such domains.

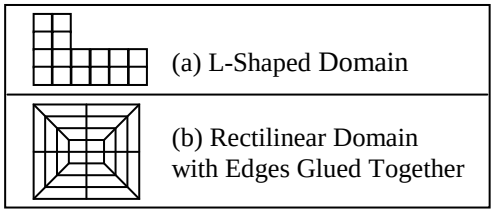

Figure 2. Examples of Complex Domain Structures

To address the above issues, our algorithm evolves to a bottom-up approach. First, however, we remove the time dimension from the partitioning space and redefine our partitions on the three-dimensional spatial structure of the data. This new partition space allows us to produce hierarchies that can easily be parallelized for data access.

The second version of our algorithm (called *GRID*) utilizes a grid-based bottom-up partitioning approach. GRID constructs a hierarchy by systematically agglomerating the underlying Cartesian grid that is placed on a mesh data set. Specifically, a simple *coarsing strategy* starts at the initial grid configuration and iteratively produces *coarse* level collections of cells from *fine* level collections of grid cells. Unlike our top-down approach, GRID scales well to large data sets, deals effectively with irregularities of the grid, and produces hierarchies with better structure than the top-down algorithm. However, it is still not able to capture the topological information (*i.e.*, the true physical relationships of the grid cells) of a mesh data set

The third version of our algorithm, called TOPOLOGY, improves on the previous bottom-up approach by utilizing the intrinsic topology of the data given in the original scientific problem to build the partition hierarchy. TOPOLOGY uses a two-pass approach. In the first pass, each coarse cell is assigned the "best" neighborhood configuration (with respect to its rectilinear cell shape). This operation is a local search on the 2^N possible neighborhood configurations of a coarse cell, where N is the number of dimensions. For instance, in two dimensions, the four possible locations for a given cell (within a coarse agglomeration) are denoted by the grey boxes in Figure 3.

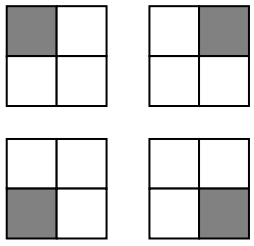

Figure 3. Four Possible Locations for a Cell within a Coarse Agglomeration in Two Dimensions

Since the first pass of TOPOLOGY is a local operation on cells, no information about the past and future agglomerations in other regions of the domain is taken into consideration when creating ancestor-descendent relationships. For this reason, some coarse agglomerations can result in trees that are non-binary, non-quad, or non-octree. For instance, it is easy to be in a situation (after the first pass) where the coarse cells are arranged as shown in Figure 4.

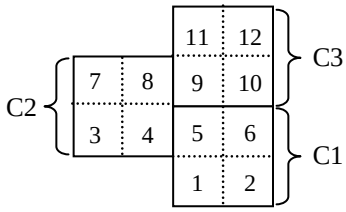

Figure 4. A Non-Quad Tree Coarse Cell Arrangement

The coarse cells (C1, C2, and C3 given by solid lines) have been arranged in such a way that indeterminate behavior for neighbors exists for the coarse cells. For example, C2 has two neighbors to its right. The second pass corrects such structural problems associated with indeterminate behavior for neighbors of coarse cells. In particular, the second pass has N-dimensional subphases. Each subphase, s, corrects the $(N - s)$ dimensional structures, planes, lines, and points. Each subphase uses information from all the previous subphases to correctly place the coarse cells. It is important to note that in the second pass, only neighbor relations are adjusted and not the coarse cells (which were defined in the first pass). For example, in two dimensions, the problem illustrated in Figure 4 can be fixed by (*i*) adjusting the face neighbors so that cell C2 "slides" down half of a coarse grid cell and (*ii*) making sure the neighbors for all local coarse cells reflect this slide (see Figure 5).

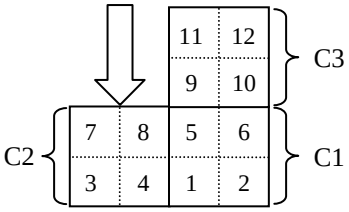

Figure 5. A Fix for a Non-Quad Tree Coarse Cell Arrangements

A heuristically complex procedure is used to compute these corrections. Our correction procedure utilizes the information about the (faces, edges, and corners of) neighbors of the coarse cells' descendents to establish neighbors at the coarse level. For instance, to find the neighbors for C2 (shown in Figure 4), we utilize the information for neighbors of cells 1, 2, 5, 6, 9, 10, 11, and 12.

In our topology-based algorithm, a new coarse level is created in the first pass and neighbors of coarse cells are identified in the second pass. The second pass rearranges the grid somewhat. The degree to which the domain of coarse cells is rearranged is bounded by the fine-cell sized moves. The degree to which a coarse level "fits" a fine level can be measured by the number of ancestor-descendent relationships that *are* established verses the number which *could be* established. This measure,

however, is not very precise since the ancestor cells are logically at coarser resolutions. In addition, since the ancestor cells' bounding boxes are approximated, they may geometrically be larger than the strict union of fine cells involved. In this case, there are no perfect spatial measurement that can capture what is being created. We have defined a reasonable measure, called *percentage filled*, which measures the non-empty volume of a cell.

2. Experiments

Due to space limitations, we describe the performance of the GRID and TOPOLOGY algorithms on one scientific data set (see [2] for more results). Our data set, called *Djehuty-50gb*, depicts a star in its mid-life and contains readings in point locations of a continuous medium (see Figure 6). The data set is represented as zones (*i.e.*, small cubes with 8 nodes). Values of variables are associated either with each node (called a *nodal variable*) or with each zone (called a *zonal variable*). Djehuty-50gb has 7,840K zones, 18 variables, and 25 time steps.

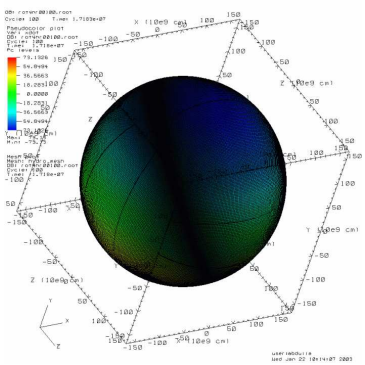

Figure 6. The Djehuty-50gb Data Set

Table 1 lists the number of levels needed to build a hierarchy for Djehuty-50gb along with their cell populations. TOPOLOGY requires more levels to complete the hierarchy than GRID since the former approach needs to consider the underlying topology each time it agglomerates cells. This consideration causes TOPOLOGY to reach levels where it can only agglomerate a few cells (as opposed to GRID where most of the time it can agglomerate many grid cells together).

Table 2 presents the minimum, average, and maximum number of children for nodes in each level of the GRID and TOPOLOGY hierarchies for Djehuty-50gb. As expected, the topology-based algorithm is able to agglomerate fewer cells (on average) as it builds a hierarchy. This is not the case for the grid-based algorithm, where the average number of children can equal the maximum number of children as the hierarchy is built (*e.g.*, level 3 in Table 2 — under the GRID column).

Table 1. Number of Cells per Level for Djehuty-50gb

Level	TOPOLOGY	GRID
0	7840K	7840K
1	980850	980000
2	125462	131144
3	17356	16393
4	2751	2401
5	578	400
6	166	108
7	54	20
8	24	4
9	9	1
10	5	—
11	2	—
12	1	—

Table 2. Number of Children in Djehuty-50gb's Hierarchies (TOP = TOPOLOGY)

Level	Min # of Children		Avg # of Children		Max # of Children	
	TOP	GRID	TOP	GRID	TOP	GRID
0	0	0	0.00	0.00	0	0
1	4	8	7.99	8.00	8	8
2	2	2	7.82	7.47	8	8
3	1	8	7.23	8.00	8	8
4	1	1	6.31	6.83	8	8
5	1	1	4.76	6.00	8	8
6	1	1	3.48	3.70	8	8
7	1	2	3.07	5.40	8	8
8	1	4	2.25	5.00	7	8
9	1	4	2.67	4.00	5	4
10	1	—	1.80	—	3	—
11	1	—	2.50	—	4	—
12	2		2.00		2	

Table 3 presents the minimum, average, and maximum percentage of non-empty space in the spatial bounding-box of cells in each level of the GRID and TOPOLOGY hierarchies for Djehuty-50gb. The minimum percentage filled shows the bad quality of the original grid structure. That is, there are fine grid cells (level 0) that are only 1.90% filled (*i.e.*, they are 98.10% empty!). These grid cells are in the boundaries of the star in our data sets. The maximum percentage filled illustrates the good quality of the grid structure. That is, there are fine grid cells (level 0) that do not have any empty space. These grid cells are in the center of the star in our data sets. But even then, the fine cells are not completely full (only 98.09%). The average percentage filled shows how close the fit of coarse cells are to their spatial bounding boxes (the higher the percentage the better the fit). The root cell has the best average percentage filled of about 52%.

Table 3. Percent Filled in Djehuty-50's Hierarchies (TOP = TOPOLOGY)

Level	Min % Filled		Avg % Filled		Max % Filled	
	TOP	GRID	TOP	GRID	TOP	GRID
0	1.90	1.90	33.38	33.38	98.09	98.09
1	0.98	2.04	31.24	31.27	96.93	96.00
2	0.50	2.31	30.75	29.96	95.32	92.36
3	0.26	3.48	28.46	30.25	86.62	93.06
4	0.14	1.95	24.08	26.91	71.51	74.76
5	0.08	1.42	17.65	24.20	64.25	58.92
6	0.20	1.42	14.56	14.27	55.66	72.40
7	0.38	4.36	14.58	20.50	38.49	44.11
8	2.43	37.68	12.75	38.81	34.22	42.18
9	0.44	52.35	14.52	52.35	32.09	52.35
10	0.44	—	16.52	—	34.83	—
11	10.01	—	30.90	—	51.79	—
12	52.35		52.35		52.35	

For Djehuty-50gb, the total number of cells in hierarchies built by TOPOLOGY and GRID are 8967258 and 8970471, respectively. The percentages of leaf and non-leaf cells for both hierarchies are about 87.4% and 12.6%, respectively. Note that even though the TOPOLOGY hierarchy is capturing more information from the data set, it has the same percentage of non-leaf cells as the GRID hierarchy. Moreover, the storage sizes for the hierarchies generated by both algorithms are the same (5.02 gigabytes). However, the memory requirement for TOPOLOGY is more than GRID since the former needs to establish the neighbor information for the coarse cells. For Djehuty-50gb, the memory requirements for TOPOLOGY and GRID are 6.46 and 5.12 gigabytes, respectively. Finally, the hierarchies for TOPOLOGY and GRID were built in 83.8 and 6187.9 minutes, respectively. This difference is partly due to the use of a simple linear search in the GRID algorithm.

3. Future work

In our top-down approach, we are considering more intelligent partitioning algorithms such as K-d and AVL trees. For our grid-based algorithm, we are exploring better searching strategies. For example, the simple linear search for cells can be replaced with hashing techniques that effectively decrease the number of cells to be examined. Our topology-based algorithm relies on strictly rectilinear (*i.e.*, hexahedral) meshes and an "octree like" structure for the agglomerated levels. To remove these strict assumptions, we are exploring the use of adaptive meshes. In particular, we are examining *ALE* (short for *Arbitrary Lagrangian–Eulerian*) meshes [3], where the grid cells (most importantly the finest level cells) can be of arbitrary polygonal shapes instead of hexahedrals. An ALE mesh will ensure that the grid better conforms to the complex shapes such as tetrahedrons. This new version of our topology-based algorithm will have a more complicated data structure but a less complex neighbor-finding algorithm.

4. Conclusion

Creating scalable partitioning algorithms is an important part of handling large-scale scientific data sets. We describe the three-step evolution of a partitioning algorithm for such data sets. Our first algorithm is a traditional top-down approach, which does not scale well to complex grid structures. The shortcomings of this approach lead to our GRID algorithm, which utilizes a bottom-up approach to construct hierarchies that conform to local grid structures (imposed on data sets by scientists). GRID generates hierarchies that are independent of the data's distribution over the initial grid. However, it does not utilize the topology of the data. Our TOPOLOGY algorithm alleviates this deficiency by considering the neighbor structure of fine cells in grid structures as well as the topological connections between these fine cells to create a hierarchy. Our experimental results show the effectiveness of our algorithms [2].

4. Acknowledgements

This work was performed under the auspices of the U.S. Department of Energy by the University of California Lawrence Livermore National Laboratory under contract No. W-7405-ENG-48.1. UCRL-JC-151476-REV-1. Our thanks to W.J. Arrighi, J.K. Durrenberger, R.T. Kamimura, N.A. Tang, and M.C. Thomas for their help.

5. References

[1] G. Abdulla, C. Baldwin, C., T. Critchlow, R. Kamimura, I. Lozares, R. Musick, N.A. Tang, B. Lee, and R. Snapp, "Approximate Ad-Hoc Query Engine for Simulation Data," *Proc. of the 1st ACM+IEEE Joint Conf. on Digital Libraries* (*JCDL*), ACM Press, 2001, pp. 255-256.

[2] C. Baldwin, T. Eliassi-Rad, G. Abdulla, and T. Critchlow, "The Evolution of a Hierarchical Partitioning Algorithm for Large-Scale Scientific Data," LLNL Technical Report, UCRL-JC-151476, 2003.

[3] R.L. Bowers, and J.R. Wilson, *Numerical Modeling in Applied Physics and Astrophysics*, Jones & Bartlett Publishers, Boston, 1991.

[4] R. Musick, and T. Critchlow, "Practical Lessons in Supporting Large-Scale Computational Science," *Proc. of SIGMOD Record*, Vol. 28, No. 4, ACM Press, 1999, pp. 49-57.

[5] W. Wang, J. Yang, and R. Muntz, "STING: A Statistical Information Grid Approach to Spatial Data Mining," *Proc. of the 23rd Int'l Conf. on Very Large Data Bases*, Morgan Kaufmann, 1997, pp. 186-195.