Preprint
UCRL-JC-151476

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

C. Baldwin, T. Eliassi-Rad, G. Abdulla,
T. Critchlow

This article was submitted to
The 15th International Conference on Scientific and Statistical Data Base Management, Cambridge, MA, July 9-11, 2003.

U.S. Department of Energy

Lawrence
Livermore
National
Laboratory

April 16, 2003

This work was performed under the auspices of the United States Department of Energy by the University of California, Lawrence Livermore National Laboratory under contract No. W-7405-Eng-48.

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, 4]. 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. Mesh data consists of interconnected grids of small zones, in which data points are stored [3].

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 1 shows two examples of such domains.

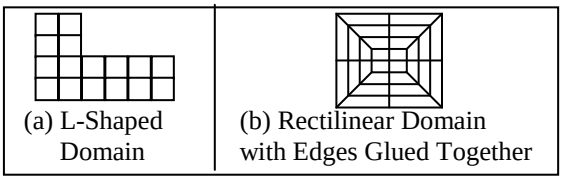

Figure 1. Examples of Complex Domain Structures

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. The second pass corrects 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. In the second pass, only neighbor relations are adjusted and not the coarse cells (which are defined in the first pass). A heuristically complex procedure utilizes the information about the (faces, edges, and corners of) neighbors of the coarse cells' descendents to establish the "correct" set of neighbors at the coarse level.

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. The data set is represented as zones. Djehuty-50gb has 7,840K zones, 18 variables, and 25 time steps.

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

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

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. 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. 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.

[4] 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.