Finding Regions of Interest on Toroidal Meshes*

Kesheng Wu‖ Rishi R. Sinha† Chad Jones‡ Stephane Ethier§ Scott Klasky¶
Kwan-Liu Ma‡ Arie Shoshani‖ Marianne Winslett**

February 9, 2011

Abstract

Fusion promises to provide clean and safe energy, and a considerable amount of research effort is underway to turn this aspiration into reality. This work focuses on a building block for analyzing data produced from the simulation of microturbulence in magnetic confinement fusion devices: the task of efficiently extracting regions of interest. Like many other simulations where a large amount of data are produced, the careful study of "interesting" parts of the data is critical to gain understanding. In this paper, we present an efficient approach for finding these regions of interest. Our approach takes full advantage of the underlying mesh structure in magnetic coordinates to produce a compact representation of the mesh points inside the regions and an efficient connected component labeling algorithm for constructing regions from points. This approach scales linearly with the surface area of the regions of interest instead of the volume as shown with both computational complexity analysis and experimental measurements. Furthermore, this new approach is 100s of times faster than a recently published method based on Cartesian coordinates.

1 Introduction

Computer simulations are providing vital details to complex processes such as protein folding (Shea & Brooks 2001), turbulent combustion (Peters 2000), supernova explosion (Röpke & Hillebrandt 2005) and nuclear fusion (Parker et al. 1993, Budny 1994, Zweiback et al. 2002). Often critical details are many orders of magnitudes smaller than the overall system scale. For example, in magnetic confinement fusion, the scale of microturbulence can be 4 to 6 orders of magnitude smaller than the size of the tokamak, while being the main cause of energy losses in the device. A faithful simulation must capture the microturbulence as well as the overall system dynamics, which leads to very large and complex simulation programs with enormous amounts of output data. Locating the essential information about the key features in mountains of data is a fundamental challenge.

In many such simulations, these features are located within some sub-domains of a feature space or small volumes in the real space generally known as *regions of interest*. For example, in a study of laser wakefield particle accelerators, these regions of interest contain accelerated particle bunches in a four-dimensional feature space (Rübel et al. 2009). In a simulation of turbulent combustion involving a diesel-air mixture, the regions of interest might be *ignition kernels*, volumes of space where the auto-ignition starts (Koegler 2001). When studying the stability of magnetic confinement for fusion, the regions of interest could be regions with high electric potential because they are associated with zonal flows critical to the stability of the magnetic confinement (Lin et al. 1998, Wang et al. 2005). This type of search is often performed interactively, which places a stringent requirement on the response time. In this paper, we propose a

‖Lawrence Berkeley National Lab
†Work performed while visiting Lawrence Berkeley National Lab. Current address: Microsoft Research
‡University of California, Davis
§Princeton Plasma Physics Lab
¶Oak Ridge National Lab
**University of Illinois at Urbana-Champaign
*This work was supported in part by the Director, Office of Advanced Scientific Computing Research, Office of Science, of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231. This research used resources of the National Energy Research Scientific Computing Center, which is supported by the Office of Science of the U.S. Department of Energy.

strategy to significantly speed up the extraction of these regions of interest from fusion simulations, and demonstrate that it achieves interactive response time on a typical desktop computer using data from a recent gyrokinetic simulation.

A common way to identify such regions of interest is through visualization (Crawford et al. 2004, Jones et al. 2008, Klasky et al. 2003). However, the output from visualization is not convenient for quantitative analysis. The regions of interest from this work can be more conveniently fed into other analyses. For example, one can compute aggregate quantities such as energy output from each region, and track the evolution of regions (Koegler 2001). On a regular mesh, there was a demonstration showing that regions of interest can be computed very quickly (Stockinger et al. 2005). A key motivation for this work is to develop an efficient technique for extracting regions of interest on an irregular mesh.

Our approach to extract regions of interest proceeds in three steps. Starting from a user-specified set of conditions, e.g., "potential $> 10^{-8}$," we first find all points satisfying the conditions, convert the points into a list of query lines (defined in Section 3.3), and then connect the query lines into regions in space. For the first step, we employ a new bitmap indexing technique that offers the best space-time trade-off among a class of similar methods. This class of methods have been studied with synthetic data (Wu et al. 2010), we are the first to use it on a set of application data. To connect the query lines, we adapt a connected component labeling algorithm to assign a unique label to each connected region (Dillencourt et al. 1992, Suzuki et al. 2003). By additionally taking full advantage of the regularity present in the toroidal mesh used for fusion simulation, we are able to achieve nearly 1000-fold speedup for the labeling step compared to a recently published result (Sinha et al. 2009). We also provide a thorough complexity analysis to support the effectiveness of our approach for identifying regions of interest.

The remaining of the paper is organized as follows. In Section 2, we review related work. We provide a description of our approach of the proposed method in section 3. We sketch out the key arguments for the overall computational complexity of the proposed approach in Section 4, and present a set of experimental measurements in Section 5. A summary of results in given in Section 6.

2 Background

In this section, we review the techniques for finding points and connecting points, and describe the simulation code that produced our test data. Some amount of details are given to provide the background information needed for understanding the proposed techniques and the complexity analysis.

Our work extends database techniques to the analysis of fusion simulation data, therefore, we use both database terms and physics terms. The simulation program is a particle-in-cell (PIC) fusion simulation involving both particles and fields (Lin et al. 1998). We organize the data into two database tables, one for the particles and the other for fields. The table for particles treats each particle as a row and each variable as a column. The fields are discretized onto an irregular mesh and the corresponding table has a row for each mesh point and a column for each field variable.

The above mapping is a conceptual mapping; we don't load the data into a database management system (DBMS), but keep the simulation data in files generated by the simulation program. This approach avoids a number of shortcomings identified in earlier studies (Musick & Critchlow 1999), and allows application scientists to continue work with their familiar data formats, such as HDF5 (The HDF Group 2007) and netCDF (Unidata 2007). Even though this work uses HDF5 files, it is straightforward to extend the techniques to data stored in other formats and applied to different scientific applications.

Database technologies generally treat rows in a table as independent. This is appropriate for particle data but not for field data. Our work extends indexing techniques to account for the connectivity among the field data to identify regions of interest.

2.1 Bitmap Indexes

In database terminology, an index is an auxiliary data structure used to accelerate certain operations on a dataset. The most commonly used index is a variation of B-tree (Comer 1979). B-tree and many other tree-based indexes are designed to be updated quickly as a data record is modified. For example, after an ATM transaction, the index about an account balance is to be updated immediately. However, in most scientific applications, the data records are not modified (Gray et al. 2005, Musick & Critchlow 1999). Therefore, an indexing technology with more emphasis on

		Equality				Range				Interval		
RID	**A**	=0	=1	=2	=3	≤ 0	≤ 1	≤ 2	≤ 3	$[0,1]$	$[1,2]$	$[2,3]$
1	0	1	0	0	0	1	1	1	1	1	0	0
2	1	0	1	0	0	0	1	1	1	1	1	0
3	3	0	0	0	1	0	0	0	1	0	0	1
4	2	0	0	1	0	0	0	1	1	0	1	1
5	3	0	0	0	1	0	0	0	1	0	0	1
6	3	0	0	0	1	0	0	0	1	0	0	1
7	1	0	1	0	0	0	1	1	1	1	1	0
8	2	0	0	1	0	0	0	1	1	0	1	1
		e_0	e_1	e_2	e_3	r_0	r_1	r_2	r_3	i_0	i_1	i_2

Figure 1: An illustration of three bitmap indexes for a column **A** with four distinct values from 0 to 3. RID is short for "row identifiers." The Equality, Range and Interval indexes are produced with different bitmap encoding methods known as Equality, Range and Interval encoding.

searching performance could be a better alternative. The most successful index of this type is the bitmap index (O'Neil 1987, Chan & Ioannidis 1998).

Figure 1 shows three versions of bitmap indexes on a tiny dataset with one column named **A**. We call the number of distinct values of a column the *column cardinality*, or simply the cardinality. The cardinality of **A** is four in this case. Figure 1 shows a logical view of three basic bitmap indexing methods labeled "Equality", "Range" and "Interval." In an Equality index, each bitmap is defined by an equality condition, such as "$\mathbf{A} = 2$" (O'Neil 1987); in a Range index, each bitmap is defined by a one-sided range condition, such as "$\mathbf{A} \leq 2$" (Chan & Ioannidis 1998); while in an Interval index, each bitmap is defined by a two-sided range condition that spans half of the values, such as "$0 \leq \mathbf{A} < 2$" (Chan & Ioannidis 1999). The numbers of bitmaps used by the Equality index and the Range are the cardinality of **A**, which is 4 in our example;[1] while the Interval index uses about half as many bitmaps.[2]

To resolve a range condition on **A** such as "**A** between 1 and 2," we retrieve some bitmaps from an index and perform bitwise logical operations to produce a bitmap representing whether each row satisfy the condition. For the specific example, we can either perform a bitwise OR on bitmaps e_1 and e_2 using the Equality index, perform a bitwise difference between r_2 and r_0 (r_2 AND NOT r_0) using the Range index, or simply retrieve i_1 from the Interval index. Because the bitwise logical operations are efficiently supported by computer hardware, these bitmap indexes are very fast at answering queries.

The example in Figure 1 uses an integer variable with a low column cardinality. In scientific applications, most data are floating-point values with very high cardinalities. For example, the test data used for this work contains about 750 data tables with 10 million rows each, most of the columns have more than 9 million distinct values. The Equality index and the Range index require more than 9 million bitmaps to index each of these columns. Without compression, they require 9 million bits to represent each value. Assuming each input value is stored in a 4-byte word, this index size is nearly a quarter of a million times larger than the base data. Such an index requires too much storage for practical uses.

A number of recent research efforts have addressed this issue (Wu & et al. 2009). A commonly used strategy to control the index size is to compress the bitmaps. There are a number of different specialized compression methods for bitmaps. Among them, the Word-Aligned Hybrid (WAH) compression method has been shown to produce a theoretically optimal indexing method (Wu et al. 2006). This same optimality holds for a handful of the B-tree variants, however, the compressed bitmap indexes are generally faster in answering queries than these B-tree variants (O'Neil 1987, Wu et al. 2002, Wu et al. 2004).

Other techniques for addressing the size issue of bitmap indexes include more complex bitmap encoding methods and binning (Stockinger & Wu 2006). The three different indexes in Figure 1 represent three different basic bitmap encoding methods known as the Equality encoding, the Range encoding and the Interval encoding. Among them,

[1] Often the last bitmap in the Range index is ignored because it contains only 1s.

[2] For a column with cardinality C, each bitmap in an Interval index corresponds to a range with $\lceil C/2 \rceil$ distinct values; a total of $C - \lceil C/2 \rceil + 1$ bitmaps are used (Chan & Ioannidis 1999).

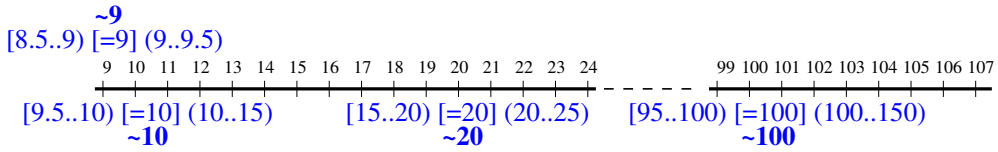


Figure 2: An illustration of 1-digit binning. The values following the symbol $\sim$ are 1-digit-precision reference values. A bracket indicates the closed end of a range, and a parenthesis indicates the open end of a range. For example, the range $[8.5..9)$ includes its left end but excludes its right end.

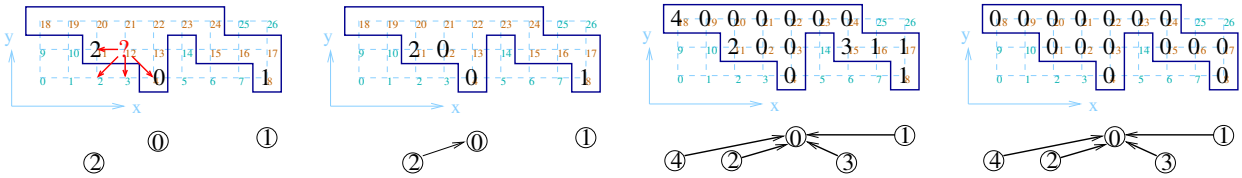
the Range index and the Interval index require the least number of bitmaps to answer an arbitrary query (Chan & Ioannidis 1998, Chan & Ioannidis 1999), and the Equality index has the smallest sizes after compression (Wu et al. 2010). Because the Range index and the Interval index are hard to compress, their sizes can be much larger than the base data for a high cardinality column. However, when they are small enough to fit on disk, they can answer queries much faster than the Equality index on average. There are different ways of composing more encoding methods from these basic encoding methods (Chan & Ioannidis 1998, Sinha & Winslett 2007, Wu et al. 2010). The indexing methods to be discussed later use a small number of bitmaps with the Range encoding or the Interval encoding and leave the majority of the bitmaps in the Equality encoding. Similar multi-level bitmap indexes have been studied before (Sinha & Winslett 2007, Wu et al. 2001), but, the particular variations have only been recently established through a theoretical analysis (Wu et al. 2010). We will describe our two-level methods in more detail in Section 3.2 and demonstrate their effectiveness in Section 5. This work is the first to apply such techniques to an application dataset.

Instead of producing a bitmap for each distinct value, we could represent a group of values with a bitmap. A simple way of implementing this idea is to bin the base data and use a bitmap to represent each bin. This reduces the number of bitmaps used in an index. A disadvantage is that the binned index only answers some of the queries precisely, for others it has to go back to the base data to get precise answers. This extra step can be expensive (Rotem et al. 2005, Wu et al. 2008). However, in many cases, such an index is able to answer all user queries without going back to the base data. For example, even though scientific data may be computed and stored in high precision, many of the analyses, especially exploratory analyses, access data with relatively low precision conditions. It is common for a user to draw an isocontour of pressure equal to 10^5 Pascal or to find regions of interest where temperature is greater than 2×10^3 degrees. In each case, the values specified by the user only have one significant digit. Based on this observation, we can bin the values to one or two significant digits and build indexes on the binned values. In later tests, we bin the values with one-digit precision to ensure that the Range index and the Interval index are not too large.

This low-precision binning strategy has not been described elsewhere, so we give a brief illustration in Figure 2. This binning procedure first reduces the precision of a value to the specified number of digits, for example, rounding the value 11 to 1-digit precision produces 10, and we call the reduced precision value the *reference value*. For each reference value, we produce three bins for values less than, equal to, or greater than the reference value. This allows the binned index to any range conditions involving a reference values, such as "$A > 10$," "$A < 10$," "$A = 10$," "$A \geq 10$," and "$A \leq 10$."

2.2 Finding Regions of Interest

Finding regions of interest is a common task in image processing, computer graphics, and many other applications (Samet 1990). A number of indexing techniques can accelerate these operations (Samet 1984). Most of these techniques, including R-tree, kD-tree and Pyramid Tree, partition space in a hierarchical manner. The main limitation of these methods is that the dimensions (i.e., columns) involved in a query must be the same as those used for partitioning. For exploratory queries where the columns involved vary from one query to another, these indexes are not effective. Furthermore, the effectiveness of these indexes deteriorates as more dimensions are used for partitioning. As the number of dimensions reaches 10 or more, these multi-dimensional indexes are less efficient than simply scanning the base data. Because a typical scientific data set contains many columns and the typical queries are ad hoc in nature, using bitmap indexes to work on each column independently is more effective.



(a) examining point 12 (b) assigned a label to 12 (c) scanned all points (d) final labels

Figure 3: Schematics of a two-pass connected component labeling on a regular mesh. Each point is denoted by an intersection of dashed lines and identified by a number in a smaller font. The label assigned to each label is a number in a larger font. The labels in circles connected with arrows represent the union-find data trees. All labels in a tree are to be replaced with the root of the tree.

After finding the points with the bitmap indexes, the next step is connecting points into regions. The approach we take is generally known as *connected component labeling*, which assigns a unique label to each connected region (Dillencourt et al. 1992, Suzuki et al. 2003, Chang et al. 2004). The sequential algorithms for connected component labeling can be divided into three classes, multi-pass algorithms, two-pass algorithms, and one-pass algorithms. The first two classes share the following scanning procedure: examine each point one after another, if there is any neighbor with a label already, it takes on one of their labels; if there is no neighbor with a label, it receives a new label. When a point is found to have neighbors with different labels, then we say all the labels are equivalent to each other. The labeling algorithms differ in how they select the representative and how to remember the equivalence information among the labels. The multi-pass algorithms either do not record the equivalence information or do not keep the complete equivalence information; they are typically simple but slower than a two-pass algorithm.

A two-pass labeling algorithm is depicted in Figure 3 (Wu, Otoo & Suzuki 2009). This example uses a 9×3 regular mesh marked by the dashed lines. The points inside the solid lines form a region of interest. Figure 3(a) shows the scanning procedure after we have examined the first 12 points and assigned label 0 to point 4, label 1 to point 8, and label 2 to point 11. To emphasis that these labels may change later, we call them *provisional labels*.

In the scanning procedure, to assign a label to point 12, we examine its neighbors that have already been processed. These points are 2, 3, 4 and 11. In this example, points 4 and 11 are inside the regions of interest and have received labels 0 and 2. In this case, we choose the smaller labels as the provisional label for point 12. To record the fact that we have chosen to use label 0 as the representative of label 2, we have label 2 pointing to label 0 with an arrow in Figure 3(b). We say that labels 0 and 2 are equivalent to each other.

The most effective data structure for representing this equivalence information is the *union-find* data structure, which supports merging of equivalent sets (union) and locating the representative of each equivalent set (find) (Tarjan 1975, Galil & Italiano 1991). The union-find data structure is commonly implemented with a collection of nodes that point to each other. This implementation produces many small objects in memory. Operations on such a data structure need to chase the pointers to unpredictable locations, which is usually slow. In addition, the process of allocating many small objects can also be time-consuming. In this work, we use an implicit union-find data structure implemented in a simple array (Wu, Otoo & Suzuki 2009). This implicit union-find data structure avoids the explicit pointers and improves the performance of connected component labeling algorithms. Additional information about this data structure is given in Section 3.7.

The one-pass algorithms include the region growing algorithm and the contour tracing algorithm (Chang et al. 2004). They go through the data records once to assign their labels, however they access data in an irregular manner, such one-pass algorithms could be slower than two-pass algorithms because the irregular memory accesses are much more expensive than sequential accesses. Furthermore, some of these one-pass algorithms are only applicable to certain types of data. For example, the contour tracing algorithm can only be used for two-dimensional (2D) images. Overall, the two-pass labeling algorithms with the implicit union-find data structure is flexible and efficient for our work.

To achieve better performance, we avoid enumerating the points inside the regions of interest by organizing the adjacent points into *query lines*. In the example shown in Figure 3, there are 5 such query lines: the first one contains

point 4 alone, the second one contains point 8, the third one contains points 11 through 13, the fourth one contains point 15 through 17, and the fifth one contains points 18 through 24. Earlier work demonstrated the effectiveness of this approach with empirical evidences (Shima et al. 1990, Wu et al. 2003, Sinha et al. 2009). Because our approach for finding points naturally groups adjacent points together, we can effectively produce query lines instead of lists of data points. Furthermore, we provide an analysis of the computational complexity for the labeling step.

To assign a label to each point, we need to find its neighbors. In the above example, it is straightforward to determine the neighbors of each point. However, this may not be the case if the points are stored in a different order or the points are not on a regular mesh. The mesh structure used in fusion simulations is irregular. In earlier work on fusion data, the connectivity information among the data records is stored in a connectivity graph that requires many megabytes of space to store (Klasky et al. 2003, Sinha et al. 2009). More importantly, the neighbors of adjacent points are not necessarily neighbors, which require the labeling procedure to examine each mesh point separately. We propose a way to identify the neighbors and avoid examining each mesh point separately.

For visual presentation, the outline of a region of interest is similar to an isocontour, therefore, we briefly mention isocontouring algorithms. Isocontouring is a common visualization technique with active research effort (Bajaj et al. 1996, Cignoni et al. 1997, Livnat et al. 1996, Schreiner et al. 2006, Kloetzli et al. 2008). Earlier studies of zonal flows make extensive uses of isocontours (Lin et al. 1998, Wang et al. 2005). With an efficient indexing structure such as interval tree and span space, the execution time of an isocontouring algorithm is proportional to the size of the isocontour, which is theoretically optimal. On data from regular meshes, a two-pass connected component labeling algorithm was shown to have the same optimal scaling property with experimental measurements (Wu et al. 2003). However, the regions of interest is a more general tool than isocontouring. With an efficient labeling procedure, it is even possible to find regions faster than identifying the isocontours (Stockinger et al. 2005). In this work, we extend the labeling algorithm to work with an irregular mesh.

2.3 Gyrokinetic Toroidal Code

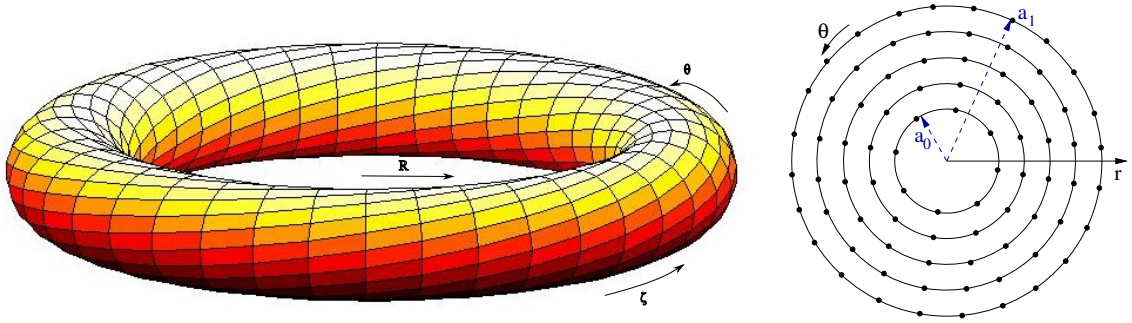
Next, we briefly describe the fusion simulation code used to generate our test data, the Gyrokinetic Toroidal Code (GTC). It was developed to study energy losses due to microturbulence in magnetic confinement fusion devices such as ITER.[3] It is a first principles code that solves the gyrophase-averaged kinetic equations for the hot plasma core of a tokamak fusion device using the particle-in-cell method (Lin et al. 1998, Lin & Lee 1995, Lee 1983, Lee 2004). By averaging the fast gyrating particle motion out of the kinetic equation, the gyrokinetic approach focuses on the physics relevant to the microturbulence without having to resolve the fast cyclotron motion of the individual charged particles. The highly scalable GTC runs on the cutting edge supercomputers and produces vast amounts of data (Ethier et al. 2008, Klasky et al. 2003). Analyzing the resulting data requires efficient methods for locating and examining these very large datasets. Therefore, most analysis and visualization tasks need to concentrate on regions with interesting features or activities. Quickly locating these regions is therefore an important part of analyzing GTC data. A unique challenge in this process is that the GTC mesh follows the configuration of the equilibrium magnetic field as its field lines twist around the toroidal geometry of the device. This twisted mesh allows GTC to reduce the computational work by 2 orders of magnitude. We seek to take advantage of the same mesh structure to accelerate the data analysis.

The mesh used for discretization twists with the magnetic field lines as illustrated in Figure 4(a). The major radius of the torus is denoted by R and the associated angle, known as the toroidal angle, is denoted by ζ. Because the motion along the toroidal direction is rapid but smoothly varying, the discretization of ζ is quite coarse.

A cross-section of the torus is called a poloidal plane. GTC discretizes the poloidal planes into equally spaced concentric rings and divides each ring into points. It keeps a constant arc length between the neighboring points in the magnetic coordinate system, resulting in more points on the outer rings than on the inner ones. The minor radius within the poloidal plane is denoted by r and the associated poloidal angle θ. An illustration of such a poloidal plane is shown in Figure 4(b).

Visualizing the GTC data has been an integral part of the data analysis (Crawford et al. 2004, Jones et al. 2007, Klasky et al. 2003). Common approaches include rendering the whole volume with transparency and displaying surface of selected regions. Rendering the whole volume requires a significant amount of data and is very time-consuming (Ma et al. 1994). Another shortcoming with this approach is that the resulting image or movie often

[3]More information about ITER can be found at `http://www.iter.org/`.



(a) field-line following mesh (b) points on a poloidal plane

Figure 4: An illustration of GTC mesh.

contains too much information and the viewer has to visually isolate the interesting features. With a large dataset, it is easy to miss important features. Most volume rendering software allows the user to provide a custom transfer function that completely suppresses the majority of the values. Such filtering can be more efficiently performed using a database index.

Displaying surfaces can avoid some of the problems associated with volume rendering, and many visualization systems are very efficient at such tasks (Bajaj et al. 1996, Kloetzli et al. 2008). However, such systems do not provide any assistance if the user needs to perform further computation on some interesting subset of the data.

Our approach provides an efficient way of selecting a subset of the data, which complements the existing visualization techniques (Post et al. 2003, Jones et al. 2007). After the selection process, only a smaller, presumably more interesting, volume of data is rendered or analyzed. This reduces the amount of data processed and reduces the visual clutter. Our approach also connects points into regions which provides additional flexibility for visualization and other analysis tasks, such as computing the total amount of heat generated from each connected region.

3 Approach

In this section, we give a detailed description of the approach we use to find regions of interest. We first give a brief outline of the key steps before going into the details.

3.1 Outline of the New Approach

A region of interest is defined by a user-specified set of conditions on some variables, such as, "potential $> 5 \times 10^{-8}$." A rendering of the boundaries of such regions is shown in Fig. 5, where each region is displayed with a different color. Our procedure to find regions of interest is divided into three steps. The first step utilizes bitmap indexes to locate all rows (i.e., mesh points) satisfying the set of given conditions. This step produces a compressed bitmap as the output. The second step turns this compressed bitmap into a list of query lines (defined in Section 3.3), and the third step assigns a label to each query line so that the connected query lines have the same label. These labels provide a unique identifier for each connected region.

Finding Points We use bitmap indexes to find points satisfying a set of conditions. The indexes are built for each column separately. Such an index can resolve a range condition on a column and produce a compressed bitmap as the answer. For a set of multi-dimensional range conditions, such as "temperature ≥ 1000 and pressure between 5×10^6 and 7×10^6," we invoke indexes on temperature and pressure separately to resolve their respective conditions, and then combine the two partial answers with a bitwise AND operation.

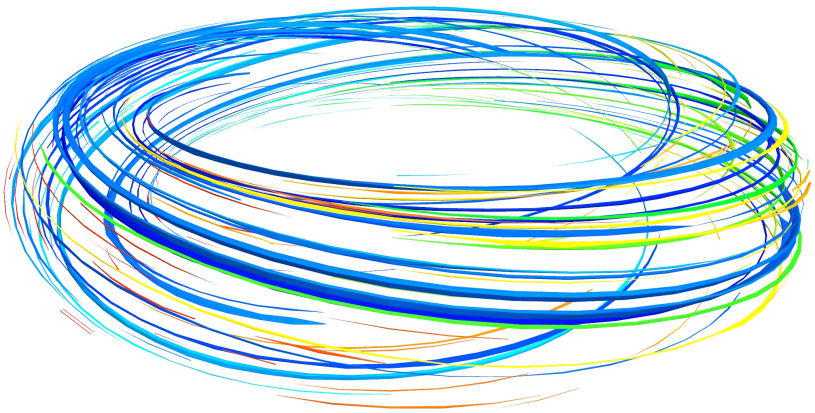


Figure 5: A rendering of boundaries of regions with high magnetic potential.

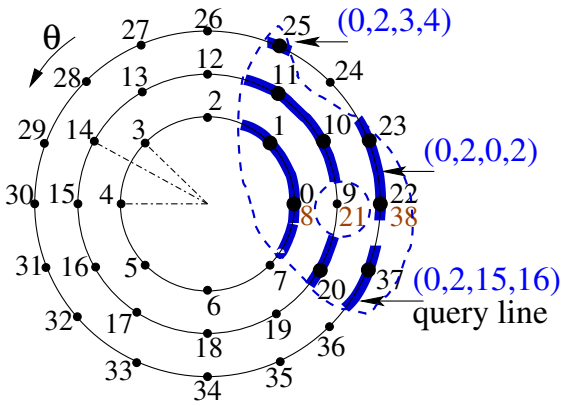


Figure 6: An illustration of row numbering of mesh points on a poloidal plane. A region of interest is outlined with dashed blue lines and the query lines are marked with thick blue lines. The three query lines on the outer ring are also labeled with their coordinates as described in Section 3.4. Points 8, 21, and 36 are ghost points.

Constructing Query Lines This step breaks the compressed bitmap produced from the first step into a list of query lines. A query line is a group of adjacent mesh points on the same mesh line. Because the compression method used is based on run-length code, these adjacent mesh points can be easily identified. Fig 6 provides an illustration of such query lines which will be further explained in Sections 3.3 and 3.4.

Connecting Query Lines To identify coherent regions in space, we assign a unique label to each of them. This is accomplished through connected component labeling. The particular labeling procedure used is a two-pass algorithm with an implicit union-find data structure (Wu, Otoo & Suzuki 2009). We further improve the efficiency of the step by exploring a new definition of connectivity among the mesh points.

GTC simulation performs most of its computation in the magnetic coordinate system because the mesh structure is considerably simpler in the magnetic coordinates than in the Cartesian coordinates. This motivated us to explore the connectivity among the mesh points in the magnetic coordinates as explained in Section 3.5. We have verified that this new definition leads to same or very similar regions of interest as the previous connectivity definition; an example of which is shown in Figure 5. This new definition turns out to be a key factor to the overall effectiveness of our approach.

Given the above outline, we next fill in the details about bitmap indexes, GTC meshes, and the connected component labeling algorithm on query lines.

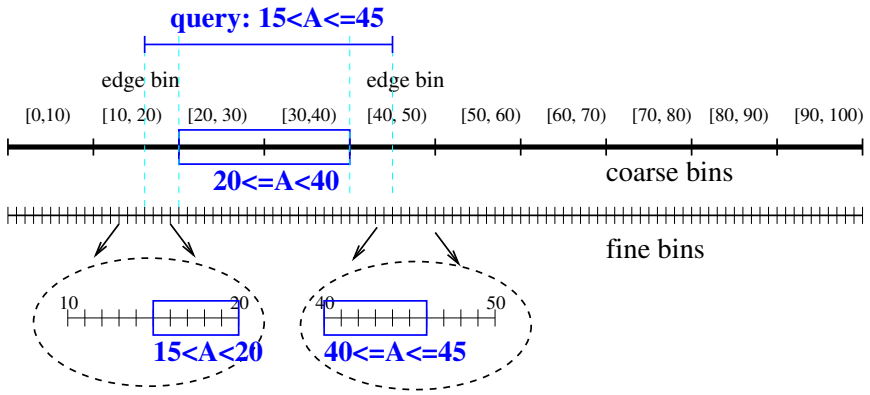


Figure 7: An illustration of two-level binning.

3.2 Two-level Bitmap Indexes

In Section 2.1, we have briefly reviewed a number of different bitmap indexing methods, particularly, the three basic bitmap encoding methods shown in Fig. 1. In this section, we describe a set of two-level encoding methods that produce two-level bitmap indexes.

Figure 7 illustrates a two-level binning of values between 0 and 100. The coarse level has ten bins and the fine level has 100 bins. We take the whole numbers in the range as the bin boundaries for the fine level and the coarse bin boundaries are multiples of tens. Furthermore, we assume each bin includes its left boundary but excludes its right boundary.

We call the range specified by a query condition such as "$15 < A \leq 45$" a *query range*. The constants defining the range are called *query boundaries*; in this example, they are 15 and 45. To find records satisfying this particular range condition, we break the query range into three smaller ranges, "$15 < A < 20$," "$20 \leq A < 40$," and "$40 \leq A \leq 45$." In this process, we introduce an interior range using coarse bin boundaries. This interior range "$20 \leq A < 40$" overlaps with some coarse bins exactly and can be resolve with the coarse-level index. The other two narrower ranges fall within two coarse bins, which we call *edge bins*. We need the fine-level index to resolve these range conditions in the edge bins. If the fine-level index is insufficient to resolve the range conditions exactly, we will use the original data records to compute the exact answers (Wu et al. 2008).

After we decided how to bin the data, we choose how to generate the bitmaps. This process is referred to as bitmap encoding as discussed in Section 2.1. Among encoding methods, the Equality encoding produces bitmaps that compress well, but may require a large number of bitmaps to answer a query. In contrast, the Range encoding and the Interval encoding produce bitmaps that do not compress well, but only need one or two bitmaps to answer a query (Chan & Ioannidis 1998, Chan & Ioannidis 1999). With a multi-level binning, we can choose a different encoding method for each level. There have been a number of different empirical studies of multi-level bitmap indexes (Sinha & Winslett 2007, Wu et al. 2001). However, those studies were not able to determine the critical parameters such as the encoding methods, the number of levels and the number of coarse bins. Recently, a theoretical study by Wu et al. (2010) showed that two levels are sufficient and also produce a set of formulas for determining the number of coarse bins. That analysis also suggested the Equality encoding for the fine level and the Range or Interval encoding for the coarse level.

Typically, there are many bins in the fine level, and choosing the Equality encoding guarantees the resulting bitmaps can be compressed well, which keeps the fine-level index small. There are a small number of coarse bins, often less then 50, therefore, even though the Range encoding and the Interval encoding produce incompressible bitmaps, their total size is still modest. The overall index size is relatively small (Wu et al. 2010).

When using a two-level index to answer a query, the majority of the records are identified through the coarse-level index. Using Range encoding and Interval encoding at the coarse level allows us to identify these records quickly. The records in the edge bins need to be resolved with the Equality encoded fine-level index. However, because the range conditions in these edge bins are much narrower than the original range condition, the Equality encoded index

is efficient. To illustrate this point, let's consider the example shown in Fig. 7. To simplify the discussion, we assume that the data values are integers. To answer the query "$15 < A \leq 45$" with the fine index alone, we need to perform bitwise OR operations among 30 bitmaps. Using the same index to resolve the two range conditions in the edge bins, we only need to perform bitwise OR operations on 10 bitmaps, 4 bitmaps for "$15 < A < 20$," and 6 bitmaps for "$40 \leq A \leq 45$." In this example, the two bitmaps from the coarse level replace the 20 bitmaps from the fine level.

3.3 Magnetic Coordinates

The second step of our approach converts the compressed bitmap produced from the first step into query lines. Let i_ζ, i_r, and i_θ denote the 3D coordinates for the discretized values of the toroidal angle ζ, the minor radius r, and the poloidal angle θ in the magnetic coordinate system. A *query line* consists of set of mesh points with the same i_ζ and i_r, and consecutive i_θ. To construct a query lines, we effectively need to convert the row numbers represented by the compressed bitmap into i_ζ, i_r, and i_θ coordinates.

A bitmap index does not explicitly record the row numbers; instead, it relies on a fixed ordering of the rows. This order is same as how the values are ordered in the output files produced by the GTC simulations. Currently, a field variable is stored as a two-dimensional (2D) array with the first dimension for the poloidal planes and the second for points within a poloidal plane. Let n_ζ denote the number of poloidal planes and n_ψ denote the number of points in each poloidal plane. The number of poloidal planes used in a GTC simulation is typically modest, for example, there are only 64 planes in the test dataset used. In building a bitmap index, we linearize the points by assuming that the second dimension is the fast varying dimension, which matches the ordering of the points in data files. We call this number the row number. For any bitmap associated with the bitmap indexes, each position in a bitmap corresponds to a specific row number. In particular, in the bitmap representing a query result, if the ith row satisfies the query conditions, then the ith bit is set to 1.

Given a row number ℓ, the poloidal plane it belongs to is $i_\zeta = \lfloor \ell / n_\psi \rfloor$. Because the number of mesh points on each ring is different, in the output from GTC, the mesh points from different rings are packed together according to their minor radius r. Because the with the same radius r always have the same number of points, the position of the first point on each ring is the same for all poloidal planes. If we record these starting positions in an array s, then we can determine which ring the point ℓ belongs by doing a binary search for the offset $\ell - i_\zeta n_\psi$. For the three rings shown in Figure 6, the starting positions are (0, 9, 22, 39). An extra value is used at the end to simplify programming. Let m_r denote the number of rings in a poloidal plane. The binary search will access about $\log_2 m_r$ elements of the array s. However, because the arc length between two adjacent are about the same, we can derive a quadratic formula for s as follows. This formula predicts the value i_r to start the binary search and reduce the number of iterations of the binary search to 0 in most cases.

As illustrated in Figure 4(b) and 6, a poloidal plane is broken into concentric rings with points in a small ring numbered before those on a larger ring. The arc length between any two neighboring points on a ring are the same, therefore once the radius r is determined, the number of points can be computed. GTC specifies a minimum radius a_0, a maximum radius a_1, the number of rings to use m_r, and the number of points to use on the largest ring m_θ. With these parameters, the jth ring has a radius of $r_j = a_0 + (a_1 - a_0)j/m_r$, and the number of points on this ring is $n_j = m_\theta r_j / a_1$. There is a ghost point at the end of each ring that replicates the point with $i_\theta = 0$. The number of points on the first i_r rings is $s_{i_r} = \sum_{j=0}^{i_r-1}(n_j + 1) = i_r + \frac{m_\theta i_r}{a_1}\left(a_0 + \frac{a_1-a_0}{m_r}\frac{i_r-1}{2}\right)$. This quadratic formula can be used to compute the coordinate i_r of a point by computing the value of i_r that satisfy the following equation:

$$\ell - i_\zeta n_\psi = i_r + \frac{m_\theta i_r}{a_1}\left(a_0 + \frac{a_1 - a_0}{m_r}\frac{i_r - 1}{2}\right).$$

The above quadratic equation has one positive root and we round this positive root down to the nearest integer value as i_r.

In most case, the predicted i_r would be the correct value. However, because the GTC mesh always uses an even number of points on each ring, the computed i_r has a slight chance of being wrong. For this reason, we use the compute value of i_r as the starting value for our binary search procedure. In our experience, the binary search never needs more than one iteration.

Once the coordinate i_r is determined, the coordinate i_θ is given by $\ell - i_\zeta n_\psi - s_{i_r}$, where ℓ is the row number, i_ζ is the ζ coordinate, s_{i_r} is the number of points on the first i_r rings in a poloidal plane, and n_ψ is the number of points in a poloidal plane.

In addition to the ghost point on each ring, GTC mesh also contains a ghost plane that replicates the poloidal plane with $i_\zeta = 0$. This extra plane does not affects the computation of the coordinates, but does slightly modify the handling of wrap-around for the ζ and θ angle.

3.4 Query Lines

Now that we know how to convert a row number ℓ in magnetic coordinates (i_ζ, i_r, i_θ), we explain how the row numbers are extracted from a WAH compressed bitmap. By our definition, all points on a query line share the same i_ζ and i_r, therefore we only need to record them once. Furthermore, all points on a query line are adjacent to each other, therefore, we only need to record the range of their θ coordinates. More specifically, a query line is represent by four numbers (i_ζ, i_r, i_θ, j_θ), where i_ζ and i_r identify the ring, i_θ is the θ coordinate of the first point, and j_θ is the θ coordinate of the point just pass the end of the query line. In Figure 6, the three query lines on the outer most ring are expressed (0, 2, 0, 2), (0, 2, 3, 4) and (0, 2, 15, 16) using the 3D magnetic coordinates (assuming $i_\zeta = 0$).

The bitmap index software provides an iterator over a compressed bitmap that iterates through the positions of bits that are 1. A WAH compressed bitmap has two types of words, *literal words* and *count words*, where a literal word contains 31 literal bits (plus a bit to indicate that it is a literal word), and a count word represents a multiple of 31 bits that are all 0 or 1.[4] The iterator decodes these words one after the other. It reports the positions of the bits that are 1 in a literal word, skips the count word representing 0s, and reports the range of row numbers represented by the count word representing 1s. The range of row numbers is represented by two values, the row number of the first bit and the row number following the last bit represented by the count word.

For the individual positions generated from a literal word, we need to examine each position and decide whether to add it to an existing query line or start a new query line. Because the iterator decodes a bitmap in increasing row numbers, we only need to examine whether the current point is connected to the last query line (i_ζ, i_r, i_θ, j_θ). It is part of the query line if and only if it is on the same ring and its θ coordinate is j_θ.

For a range generated from a count word, we need to compute 3D coordinates corresponding to the start and end positions. The first point may be connected to an existing query line, which can be easily tested as with an individual point. The range may span multiple rings in a poloidal plane or even multiple poloidal planes, in which case, we need to generate a query line for each ring or part of a new ring it covers. This can be determined by differences between i_ζ and i_r of the start and end positions.

3.5 Connectivity

In published work, the Cartesian coordinates are used define how the GTC mesh points are connected to each other, typically by tessellating space using the mesh points as input (Crawford et al. 2004, Jones et al. 2007, Sinha et al. 2009). This produces an irregular graph that connects the mesh points. However, as shown in Section 3.3, the mesh points can be described relatively easily in the magnetic coordinates. Defining the connectivity in the magnetic coordinates may be easier to use than the connectivity graph in the Cartesian coordinates. Furthermore, GTC primarily uses the dynamics in the magnetic coordinate system to compute the motions of particles, using a connectivity based on the magnetic coordinates matches better with the dynamics of the simulation. For these reasons, we explore the option of defining connectivity in the magnetic coordinates.

Within a two-pass connected component labeling procedure, the connectivity information is only used during the first pass and only the neighbors with smaller row numbers are actually examined. In the example shown in Figure 3, when deciding what label to assign to point 12, we only examine points 2, 3, 4, and 11. For this reason, it is more important to determine the neighbors with smaller row numbers efficiently. The connectivity to those with larger row numbers is implicitly defined by the reciprocal relation of the connectedness, i.e., if a point i is connected to point j ($i > j$), then point j must be connected to point i.

[4] This assumes that a word contains 32 bits.

We define the connectivity among GTC mesh points for three separate cases: points on the same ring, points on the adjacent rings, and points in the adjacent poloidal planes. First, for points on the same ring in a poloidal plane, adjacent points are considered connected. These points are also neighbors in row number ordering, and are considered as connected in other work as well (Crawford et al. 2004, Sinha et al. 2009).

The points from adjacent rings in the same poloidal plane are considered connected if their θ angles are the nearest neighbors. For example, in Figure 6, we consider point 14 as connected to points 3 and 4 because the θ angle of point 14 is the closest those of points 3 and 4. In general, a point with coordinate (i_ζ, i_r, i_θ) is connected to two points with coordinates (i_ζ, $i_r - 1$, j_θ) and (i_ζ, $i_r - 1$, k_θ) where the θ angles of the latter two are the two closest to the θ angle of the former. The twisting of the poloidal plane is determined by the external magnetic field and is given by a set of formulas. The exact θ value for each point can be computed through these formulas. With these formulas we can compute the two neighbors on ring (i_ζ, $i_r - 1$) quickly. Because the values of j_θ and k_θ are always consecutive integers, we need to compute only one of them. We choose to compute j_θ as the point with the largest θ that is not more than that of i_θ, and assign k_θ as $j_\theta + 1$ (with possible wrap-around on the ring).

When the points i_θ and j_θ have exactly the same θ angles, the above procedure still selects two neighbors. In the example shown in Figure 6, the θ angle of point 15 is exactly the same as that of point 4, by our definition, point 15 is connected to points 4 and 5. The connectivity defined by this procedure is different from the graph produced by a typical tessellation procedure; this difference is most prominent for points in the inner-most ring when the ring has only a small number of points. However, in practice, the inner-most ring always has dozens of mesh points, which is to ensure meaningful discretization of the ring. The new connectivity produces the same regions or nearly the same regions.

Because the continuity along the ζ direction is very strong, we only consider (i_ζ, i_r, i_θ) to be connected to ($i_\zeta \pm 1$, i_r, i_θ). Such connectivity along the ζ direction is weaker than typically used by those that discretize space with tetrahedrons or hexahedrons (Crawford et al. 2004, Sinha et al. 2009). However, as shown in Figure 5, the regions of interest found with our connectivity definition produces the familiar isosurface. The main reason for this outcome is that the new connectivity follows the twisting of the magnetic coordinates and therefore the dynamics of the system.

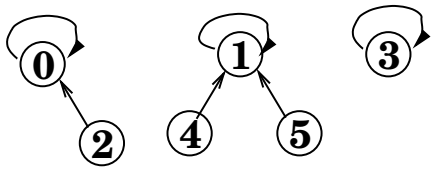
3.6 Provisional Labels

After converting a compressed bitmap representing the query results into a set of q query lines, the connected component labeling algorithm assigns q labels (one for each query line) to identify which region they belong to. From the conversion process, the query lines are ordered according to ζ, r and θ coordinates. Our labeling algorithm Scan plus Array-based Union-Find (SAUF) proceeds in three loops (Wu, Otoo & Suzuki 2009). The first loop iterates over all query lines, assigns a provisional label to each query line, and at the same time records the equivalence information among the provisional labels. We will discuss the details of these operation in the remainder of this section. The second loop determines the final label for each group of equivalent labels, which involves operations on the union-find data structure explained in the next section. The third loop assigns the final labels to each query line, which is a simple loop through every query line to replace its provisional label with the fine label.

Our labeling procedure uses integers as labels and the first query line receives the provisional label 0. For each remaining query lines, we examine whether it connects to any neighbors that already have a provisional label. If there are none, it receives a new label. Otherwise, it takes on one of their labels, and the labels assigned to the neighbors are marked as equivalent.

By construction of query lines, aside from the wrap-around at 2π, a query line can not connect to other query lines on the same ring. By our definition of connectivity given in Section 3.5, for each query line we only need to check for its neighbors on the ring just below it, and on the ring with the same radius in the preceding poloidal plane. We examine the query lines one poloidal plane at a time, and then within a plane one ring at a time. We use two pointers to track the query lines in the preceding plane and on the ring below.

We denote the current query line being examined as (i_ζ, i_r, i_θ, j_θ). To determine whether any query lines on the ring (i_ζ, $i_r - 1$) connects to it, we first compute the range of θ coordinates on the ring (i_ζ, $i_r - 1$), which is a range defined by the smallest i'_θ that connects to first point (i_ζ, i_r, i_θ) and the largest i''_θ that connects to the point (i_ζ, i_r, $j_\theta - 1$). After determining i'_θ and i''_θ, the pointer to the query lines on ring (i_ζ, $i_r - 1$) can be moved forward until the θ coordinate of the query line overlaps with i'_θ and i''_θ. Each query line with overlapping θ coordinate is a connected



(a) union-find tree

provisional label	0	1	2	3	4	5
representative	**0**	**1**	**0**	**3**	**1**	**1**

(b) array representation

Figure 8: An example union-find forest and its array representation.

neighbor of the current query line. We move the pointer forward one query line at a time until the one it points to no longer overlaps with the current query line.[5]

In the example shown in Figure 6, for query line (0, 1, 1, 3) (including points 10 and 11), $i'_\theta = 0$ (corresponding to point 0) and $i''_\theta = 2$ (corresponding to point 2). The query line (0, 0, 0, 2) overlaps with this range of θ coordinates 0 and 2, and therefore is connected with (0, 1, 1, 3). For query line (0, 1, 11, 12) including point 20, we have $i'_\theta = 7$ and $i''_\theta = 8$. Because the ring (0, 0) has only 8 points, i''_θ wraps around to 0, which means the query line (0, 0, 0, 2) (containing point 0) is connected to (0, 1, 11, 12).

To determine whether any query lines from the preceding plane are connected, we only examine the query lines on ring $(i_\zeta - 1, i_r)$ with overlapping θ coordinates. Recall that we denote the current query as $(i_\zeta, i_r, i_\theta, j_\theta)$. We maintain a pointer to query lines on ring $(i_\zeta - 1, i_r)$ in a way similar to the one on ring $(i_\zeta, i_r - 1)$. We move this pointer find the first query line on ring $(i_\zeta - 1, i_r)$ whose j_θ that is no less than the value of i_θ for the current query line. From this point on, all query lines pointed by this pointer overlaps with the current query line as long as its i_θ value is less than j_θ of the current query line and it remains on ring $(i_\zeta - 1, i_r)$.

3.7 Implicit Union-Find

While discovering which neighbors are connected to the current query line, we may find these neighbors having different provisional labels. These provisional labels are equivalent to each other and we use a union-find data structure to store the equivalence information. Conceptually, the union-find data structure represents a forest of rooted trees as shown in Figure 8(a). However, with integer labels, it is possible to use an array to represent these trees as shown in Figure 8(b) (Wu, Otoo & Suzuki 2009). In our case, a node in the union-find tree represents a provisional label and a link points to an equivalent label. We interpret the link as pointing to another node that is more "representative." If there is no label that is more "representative," then it points to itself and the root of a tree. We call the array R as a short-hand for "representative."

The union-find data structure supports three operations: add a new node, find the representative of a node, unite two nodes to mark them as equivalent. The operation of adding a new node extends the array R by one element, and new element records its position in the array (i.e., pointing to itself). The find operation follows the link from the given node to the root, which can be interpreted as finding the current representative of the given provisional label. The union operation starts with two input nodes, finds their respective representatives, and makes one of them pointing to the other. We always choose the root with the small label as the root of the combined tree.

There are a number of advantage for using array-based union-find data structure. Because it avoids the need of pointers, it is more compact than the pointer based implementations. It also avoids the problem of allocating a large number of small objects (nodes) that takes time to create and access (Wu, Otoo & Suzuki 2009).

4 Cost Analysis

We outline next an analysis of the cost of each operation. Overall, the procedure of finding regions of interest completes in $O(q)$ time, where q is the number of query lines in the regions of interest. Because the number of query lines is

[5]We need to move the pointer back one query line if there is any query line on ring $(i_\zeta, i_r - 1)$ after the matches. This is to accommodate the situation where a query line on ring $(i_\zeta, i_r - 1)$ may be connected to two or more query lines on ring (i_ζ, i_r).

never more than the number of points in the regions with neighbors outside of the regions, we can regard $O(q)$ as $O(s)$, where s is the number of points with external neighbors, which can be regarded as surface size. On regular meshes, similar approaches have been shown to outperform a well-known isocontouring algorithm with $O(s)$ time complexity (Stockinger et al. 2005). This analysis further demonstrates that such an approach is efficient for certain irregular meshes as well.

4.1 Size of Query Result

To provide a bound on the cost of using the bitmap indexes, we first examine the relation between the query result size and the number of query lines q. For a bitmap that represents q query lines, we are mainly concerned about the maximum size of the bitmap representing the result from index operations because the cost of these index operations are related to the size of the result bitmap.

A query line is represented as a group of consecutive 1s in a bitmap (Sinha et al. 2009). Under WAH compression, this group of 1s may take as much as three words, two literal words for the first and last few bits of the group and a count word (or a literal word) for the 1s in the middle. Assuming the answer to a query is represented by a properly compressed bitmap, it will have no more than $3q$ words if it represents q query lines.

It is possible for a very compact bitmap to generate many query lines. For example, a bitmap of all 1s might take only one word, but can produce many query lines. In addition, a literal word may be translated into a number of short query lines as well. This indicates that the actual sizes of bitmaps generated from the index operations may be much smaller than $3q$ words. However, we use the upper bound of $3q$ words in the remaining analysis to capture the worst case scenario.

4.2 Cost of Index Operations

Different type bitmap indexes have different performance characteristics. For this discussion, we distinguish between two types of range conditions, 1-sided range queries (1RQ), e.g., "potential $\geq -3 \times 10^{-8}$," and 2-sided range queries (2RQ), e.g., "$5 \times 10^{-11} \leq$ potential $\leq 3 \times 10^{-9}$." For regions of interest specified with one-sided range conditions, the Range index essentially contains a set of precomputed answers. The answer to such a query is either a bitmap from the index or an empty bitmap (i.e., no hit). Clearly, the Range index requires the least amount of work. Let w denote the number of words used by the result bitmap, the time to retrieve the bitmap is generally regarded as $O(w)$. Because w is no more than $3q$ words, the time to generate the answer from the Range index is $O(q)$.

In the above case, we have a theoretical bound on the computational complexity as well as the space complexity for answering a query. However, the actual bitmap size w can be much smaller than the maximum value of $3q$, and the observed index operation time may not be a linear function of q. For other types of queries and other indexes, we are not able to establish a rigorous theoretic bound, but the actual query response time is close to a linear function of q as shown in the next section. In the remaining of this section, we give some reasons for this observed linear relation.

For a two-sided range condition, both the Range index and the Interval index will access two bitmaps and perform a bitwise logical operation. According to the analysis by Wu et al. (2006), the resulting bitmap from this operation is at the maximum the total size of the two input bitmaps and the execution time is bounded by a linear function of the total size as well. Because the maximum size of the result bitmap is $3q$ words, the index operation time is $O(q)$. As we see from the timing measurements, the average index operation time is observed to be a linear function of q.

Even though the Range index and the Interval index can answer queries quickly, their sizes can be much larger than the base data. We designed the two-level indexes as a comprise between minimizing the index size and minimizing the operation time. The two-level indexes used in our study are composed of an Equality index at the fine level and a Range index or an Interval index at the coarse level. In this combination, the fine level index can accurately answer any query the whole index could. To answer a query, we examine all available options including the option of using the Equality index alone. Thus, the two-level indexes will never be more costly than the Equality index alone. Earlier analysis has proven that the WAH compressed Equality index can answer range queries in $O(p)$ time, where p is the number of points inside the regions of interest (Wu et al. 2006).

In many cases, the operations used by a two-level Interval-Equality index or Range-Equality index can be understood as follows. The time needed to operate on the coarse-level index is nearly a constant because the operation usually involves a bitwise logical operation on two uncompressed bitmaps. The time required by the fine-level index

is proportional to the mesh points involved in the operations. For relatively slow-varying field data, the coarse-level index can locate the majority of the points inside the regions of interest, and the fine-level index only needs to resolve a relatively small number of points along the boundary. We expect the time required by the fine-level index to be proportional to the number of points on the boundary of the regions. Overall, the time to operate on the Equality index may be close to a linear function of q instead of p.

4.3 Cost of Building the Query Lines

In Section 3.4, we discussed the procedure of converting a bitmap to a list of query lines. Here we briefly review the cost of that procedure, which first extracts row numbers from the compressed bitmap and then construct the query lines. Sinha et al. (2009) has discussed the same procedure in a different context.

The procedure to extract the row number works on each WAH compressed word one after another. Each literal word in the compressed bitmap is converted to a list of row numbers corresponding to the positions of the bits that are 1, and each count word is converted to a range with a starting position and an ending position. In this process, the most expensive word to decode is a literal word with all 1s. It is possible to allocate enough space to store the row numbers before decoding any compressed word, therefore the cost of recording the positions of these bits can be regarded as a constant. The total cost of this operation is $O(w)$, where w is the number of words used in a compressed bitmap.

After the row numbers are extracted, we need to convert them to the magnetic coordinates. The process given in Section 3.3 effectively requires a constant cost per row number. These conversions cost $O(w)$. The process of conversion produces q query lines and the time to record the query lines is $O(q)$. Overall, the total cost producing the query lines is $O(q)$.

4.4 Cost of Labeling

The connected component labeling algorithms are often analyzed with binary images, in which case the optimal time complexity is $O(N)$, where N is the number of the pixels in the image (Fiorio & Gustedt 1996, Chang et al. 2004, Wu, Otoo & Suzuki 2009). Because our input to the connected component labeling procedure is a list of q query lines, we can complete the labeling procedure in $O(q)$ time.

A detailed analysis of the labeling algorithm called Scan plus Array-based Union-Find (SAUF) on binary images was given by Wu, Otoo & Suzuki (2009). In their case, a label is assigned to each pixel, while in our case, a label is assigned to each query line. After replacing 'pixel' with 'query line', their arguments should carry over to our case, therefore the overall labeling cost should be $O(q)$. Here we briefly outline the analysis and refer the interested readers to Wu, Otoo & Suzuki (2009) for more details.

Recall the labeling algorithm proceeds in three loops: (1) assign provisional labels and record label equivalence, (2) determine the final label for each provisional label, and (3) replace all provisional labels with their final values. The loop 3 iterates over all q labels and replace the provisional label with the final label. The operation of computing the final label from the provisional label with the implicit union-find data structure is simply reading an element of the "representative" array R, therefore, the total cost of this loop is $O(q)$. The operations for the loop 2 is performed on the union-find data structure is a simple loop over the array R (Wu, Otoo & Suzuki 2009). Because the number of elements in array R is guaranteed to be no more than q, the cost of this loop is also $O(q)$.

The loop 1 iterates over the query lines, and in each iteration it performs a fixed amount of work for the most part and the only exception is the number of neighboring query lines examined. The number of neighbors per query line is not a constant, however, the total number of neighbors visited is a linear function of q. By the definition of connectivity given in Section 3.5, a query line may be a neighbor to another one on the same ring at most once through the shared the ghost point, a query line may be a neighbor to another one on a larger ring in the same poloidal plane or on the same size ring in the next poloidal plane if their θ coordinates have overlap. We need to count how many times a query line can be a neighbor on these two rings.

As mentioned in Section 3.6, while assigning a provisional label to a query line, we maintain two pointers to the query lines, one for those on the ring just below the current ring and one for those in the poloidal plane just before the current plane. Each of these pointers marches forward as the pointer to current query line moves. Therefore, each pointer passes through no more than q query lines. In addition, after assigning a provisional label to a query line, each

of the two pointers may move back one step, which causes each pointer to repeat the visit of no more than q query lines. Altogether, the total number of neighboring query lines accessed is no more than $4q$.

Given that the labeling algorithm accesses no more than $4q$ neighbors, it will perform no more than $4q$ union and find operations. It will generate no more than q provisional labels, therefore the union-find data structure will have no more than q elements. The cost of each individual union and find operation is not guaranteed to be a constant, however, as shown by Wu, Otoo & Suzuki (2009), the total cost of all of these operations is bounded by a linear function of number of times these operations are invoked. In other word, the amortized cost per operation is a constant. Therefore, the overall cost involving the neighbors is $O(q)$ and the cost of the whole labeling procedure is $O(q)$.

Had we chosen to work with the mesh points instead of query lines, we can follow the similar arguments to show that the time complexity of the labeling procedure is $O(p)$, where p is the number of points in the regions of interest.

4.5 Total Cost

For the three steps to find regions of interest, the last two steps clearly take $O(q)$ time. In later tests, we measure these two steps together to simplify the discussion.

For the first step, there are several possible bitmap indexes that can be used and each of them has a different performance characteristics. For regions of interest defined with one-dimensional one-sided range conditions, the Range index is shown to take $O(q)$ time; however, such an index is often very large. Therefore, we prefer two-level indexes with a judicious use of the Range index or the Interval index on the coarse level. In these cases, the time spent on the coarse level index can be regarded as a constant. Because the fine-level index is likely used to resolve points on the boundary of the regions, we anticipate that the time used by the two-level indexes may follow a linear relation with q. In the next section, we will examine whether or not this is true with measurements on a set of real application data.

5 Performance Measurements

In this section, we present a set of timing measurements to support the analysis. As a point of reference, we also show the time needed to work with individual points instead of query lines and with a different connectivity defintion.

5.1 Test Data

The test data we use contains 750 time steps stored as HDF5 files, one for each time step. We view each time step as a database table. For this test, we only index one scalar field, the magnetic potential, from each time step. We index all data points stored in the HDF5 file, including the ghost points. This allows us to directly work with the output from the simulation program. In contrast, most earlier work needed to remove the ghost points, which requires additional processing of the data before they can be used (Sinha et al. 2009).

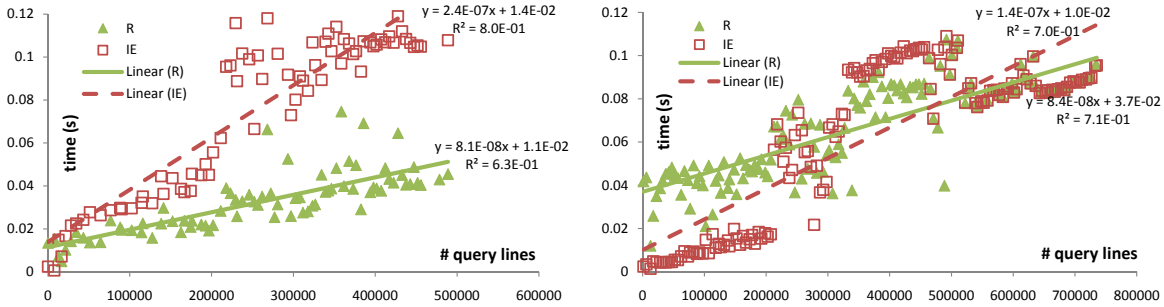
The GTC mesh used for the test data contains slightly less than 10 million points, divided into 64 poloidal planes (along with a ghost plane). Each poloidal plane has a little over 151,000 points, divided onto 192 rings, with the largest having 1,408 points. The twisting of the poloidal planes follow a quadratic formula, which requires three constants. To determine the connectivity among the points, we need to store these parameters along with the array s that records the starting position of each ring in a poloidal plane (which requires 193 integers). Altogether we need to store a little over 200 values instead of 60 million values for the adjacency matrix of a triangulated mesh formed from the GTC mesh points.

5.2 Finding Points

Bitmap indexes can locate points satisfying any conditions specified by the users. In the tests, we distinguish between 1-sided range queries (1RQ) and 2-sided range queries (2RQ). To ensure the Range index and the Interval index are relatively small, the constants used in these range conditions, known as query boundaries, have only one significant digit. To resolve these conditions accurately, we only need to bin the data with 1-digit precision. This option produces between 140 – 150 bins in most cases.

	Equality	Interval	Interval-Equality	Range	Range-Equality
index size (MB)	31.4	169	33.9	246	50.9
1RQ time (ms)	692	60	88	37	121
2RQ time (ms)	65	20	51	62	59
score (MBs)	2.04	3.38	1.73	15.25	3.00

Table 1: Average index sizes and query response time. Note that the original data value is stored as 8-byte floating-point values taking about 78.6 MB for about 9.8 million rows. The indexes are built on 1-digit precision bins that produced about 140–150 bins for a typical time step. The *score* is a product of the index size and the time to answer the 2-sided range queries (2RQ).



(a) One-sided range queries

(b) Two-sided range queries

Figure 9: Time to answer queries scales nearly linearly with the number of query lines for the Range (R) index and the Interval-Equality (IE) index. Note that the lines are produced with linear regression, and the values R^2 are the square of coefficients of correlation for the linear regressions.

Table 1 shows the average index sizes and time to process the range conditions with the indexes. For each type of index, the average size is computed over all data tables. The average time for index operations is computed over 20,000 random queries with different queries boundaries and on different data tables. For each test case, the time reported is the elapsed time in milliseconds (ms). For each set of query boundaries, we run through all variations of indexes and all data tables, which makes it hard for the file system to cache the index files. Therefore, the elapsed time reported includes the time to read the necessary portions of the index files to memory.

As expected, the Equality index is the most compact in size. The average size of an Equality index is about 40% of the original data size. The average size of an Interval index is more than five times the size of an Equality index, and a Range index is about twice as large as an Interval index because the Range index has about twice as many bitmaps. Had we used more bins, the sizes of the Range index and the Interval index would grow proportionally, while the Equality index would remain about the same size. The two-level indexes are somewhat larger than the Equality index, but are much smaller than the Interval index and the Range index. In particular, the average size of an Interval-Equality index is about 8% larger than that of an Equality index.

In terms of index operation time, the Range index requires the least amount of time for 1-sided range queries (1RQ) and the Interval index requires the least amount of time for 2-sided range queries (2RQ). On average, the time used by the two-level indexes are not significantly longer than the most efficient index. In general, the Interval-Equality index takes less time than the Range-Equality index, and both of them take less time than the Equality index.

To quantify the space-time trade-off among the different indexing methods, we define a score as the product between the index size and the query response time for two-sided range queries. This score remains the same if increasing the index size by a factor of α will decrease the query response time by the same factor α. A small score means an indexing method has a better space-time trade-off. Among the five indexing methods, we see that the Interval-Equality index has the lowest score, therefore, it has the best space-time trade-off among the methods tested.

To see the relation between the index operation time and the number of query lines, we plot these values in Figure 9,

				labeling time (ms)			speedup
				magnetic coordinates		Cartesian coordinates	
	# of points	# of query lines	# of regions	labeling query lines	labeling points	labeling query lines	
1RQ	5.39×10^6	3.11×10^5	609	45.6	519	44,060	966
2RQ	2.59×10^6	2.58×10^5	17400	32.0	257	19,016	549

Table 2: Summary information about performance of labeling procedures. The speedup compares the time of labeling the query lines in the magnetic coordinates against the same operation in the Cartesian coordinates.

along with their linear regression lines. We see that the points are fairly close to the straight lines. To quantify how the measured values fit the linear relation, we also report the coefficient of correlation R for each linear regression[6]. The maximum value of R is 1, which indicates the regression line fits the measurements perfectly. Because the coefficient R is greater than 0.8 ($R^2 > 0.64$) in most cases, we regard this as a strong support that the query response time grows linearly with the number of query lines.

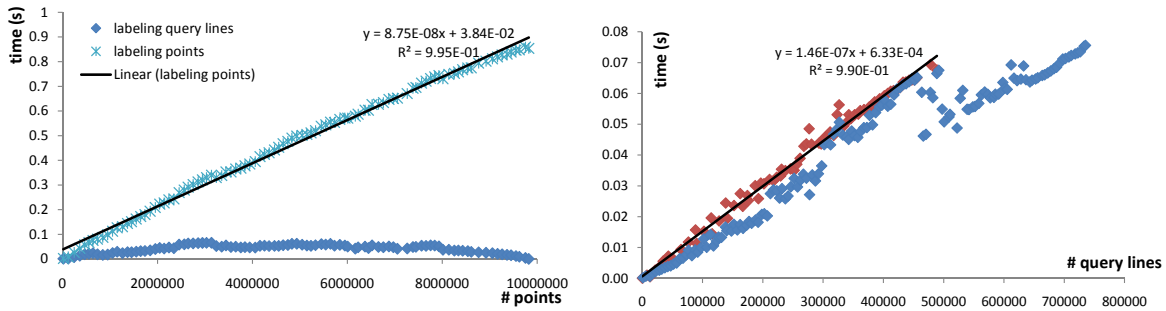
The measurements for the Interval-Equality (IE) index in Figure 9(b) show some systematic deviation from the regression line, here is an explanation. When there is a relatively small number of query lines, the fine-level Equality index is used to process the two-sided range conditions. In our current implementation, the necessary bitmaps can be retrieved with a single read operation, and we observe that the query response time is pretty close to be a linear function of q. As more query lines are produced from the queries, the coarse-level index are used as well. Once the coarse-level index is involved, we need to perform two to four more read operations depending on the query condition. One read operation is always needed to read the metadata about the coarse-level index. One or two read operations are needed to read one or two Interval encoded bitmaps from the coarse-level index. If we need to access fine-level bitmaps in two edge bins as shown in Figure 7, then we need an addition read operation to read the second set of fine-level bitmaps. The disk access latency, which is about 10 ms (0.01 s) each, contributes to a large part of the delay in query response time. Of course, as more bitmaps are involved, the operations on these bitmaps also take additional time.

5.3 Connecting Points

The second and third steps of our approach for finding regions of interest have simpler computational complexity and we measure their time together in the tests. The main operation in this part is performed with a connected component labeling algorithm named SAUF. In Table 2 we present a summary about the tests. Recall that the total number of mesh points is about 10 million. On average, the 1-sided range conditions selected about 5 million of them, which is half of the mesh points as expected. The 2-sided range conditions selected about 26% of the points, which is a little less the expected mean of 1/3rd. A query line produced by a 1-sided range condition has about 17.3 mesh points on average, while the same query line produced by a 2-sided range condition has only 10.0 mesh points. The bigger difference is that the query lines produced by the 2-sided range conditions are split into many more regions, almost 30 times more (28.6 times to be exact).

Early work on this subject defines the connectivity based on a mesh in Cartesian coordinates (Klasky et al. 2003, Sinha et al. 2009). Table 2, the column marked "Cartesian coordinates" displays the measured time used by the program from Sinha et al. (2009). In this work, we take advantage of the mesh structure in the magnetic coordinate system. The average timing information presented in Table 2 shows that using magnetic coordinates is hundreds of times faster than using the Cartesian coordinates. This is because the mesh generated in the Cartesian coordinates lacks the structure present in the magnetic coordinate system. Therefore it is much easier to perform connected component labeling in the magnetic coordinates.

[6]The linear regression is performed with Microsoft Excel, which outputs the square of coefficient of correlation rather than the coefficients of correlation.



(a) labeling time v. points

(b) labeling time v. query lines

Figure 10: Time to connect points in the regions of interest.

Working with the query lines is clearly more efficient than working with the mesh points. In our tests, labeling the query lines is 11 times faster for the set of 1-sided range conditions and 8 times faster for the set of 2-sided range conditions. Figure 10 shows the labeling time against the number of points and the number of query lines.

From Figure 10(a), we see that the time to label each point individually is a linear function of the number of points. This agrees with the theoretic computational complexity result of $O(p)$. In Figure 10(b), we show the time used to label the query lines. In this case, we separate out the measurements with 1-sided range conditions (in red) from those with 2-sided range conditions (in blue). We see that the red points are very close to the linear regression line (with the correlation coefficient $R = 0.995$ and $R^2 = 0.99$). On the other hand the time used to label the query lines produced with 2-sided range conditions are often less than that with 1-sided range conditions. From the summary in Table 2 we know that the query lines produced with 2-sided range conditions are generally shorter and connect to fewer query lines. While assigning a provisional label, each connected neighbor needs to be examined and some operations such union and find are performed. Having less neighbors would require less work in this process. Therefore, labeling query lines produced with 2-sided range conditions often takes less time as shown in Figure 10(b).

The slope of the regression line in Figure 10(a) indicates that on average it takes about 87.5 ns to label a mesh point; Figure 10(b) indicates that it takes about 146 ns to label a query line (for one-sided range queries). This suggests that working with a query line takes more time than working with a mesh point directly, however, the difference is less than a factor of two, which implies that as long as there are two points in a query line, working with query lines is more efficient.

In summary, the time to label query lines follows the $O(q)$ bound as the analysis predicted (Figure 10(b)), and labeling query lines is about ten time faster than labeling individual points. Furthermore, working with the magnetic coordinates can be hundreds of times faster than working with the Cartesian coordinates (Table 2).

6 Summary and Future Work

In this work, we investigate a strategy of extracting arbitrary regions of interest on toroidal meshes. We break the task into three steps: (1) find mesh points satisfying user specified conditions with bitmap indexes, (2) convert the bitmap answer into query lines, and (3) connect the query lines into regions in space. For the first step, we propose to use a two-level Interval-Equality index as a compromise between the large but fast Interval index, and the small but relatively slow Equality index. Our timing measurements show the Interval-Equality index takes about 2.5 times as long as the Interval index to answer a query on average, but its size is only 1/5th of the Interval index. Therefore, it offers a better space-time trade-off than the Interval index and the Equality index. Because the size of an Interval index grows proportionally with the number of bins, we have used a small number of bins in our tests. Had we used more bins, the advantage of the Interval-Equality index would be even more pronounced.

We adapted a two-pass connected component labeling algorithm with an implicit union-find data structure to identify connected regions. We study two choices of working with individual points and working with query lines. Because there are far fewer query lines than mesh points, working with the query lines is more efficient.

We have also explored the option of working with the magnetic coordinates instead of the Cartesian coordinates in defining the connectivity among the mesh points. Because the meshes used in the GTC simulation code has a much simpler description in the magnetic coordinates than in the Cartesian coordinates, operating in the magnetic coordinates is more efficient than in the Cartesian coordinates. In our tests, the average speedup is nearly a thousand for one-sided range conditions that produced the isocontours.

We have also given a computational complexity analysis for the three steps of extracting regions of interest, and show that it is possible to complete all steps in $O(q)$ time where q is the number of query lines. Because each query line has at least one point with neighbors outside of the regions of interest, the number of query lines is no more than the number of points on the surface of the regions. In other word, $O(q)$ time complexity can be regarded as $O(s)$ as well, where s is the number of mesh points in the regions that have external neighbors (or the surface size). The most efficient isocontouring algorithms can achieve $O(s)$ time complexity, but the regions of interest can be defined on more complex conditions than isocontours, and the resulting regions can be used as input to more analysis procedures than isocontours.

Using the magnetic coordinates gives us the largest gain in execution time. This appears to be something unique to the toroidal mesh. However, the general idea of taking full advantage of the regularity present in data is a theme common to most scientific approaches. One area of future work is to generalize this approach to other applications. Another area is to further quantify the strength and weakness of the two-level indexes. GTC simulation code has recently changed its output format (Lofstead et al. 2008). We are working on modifying our software to deal with the new file format. A limitation in our approach is that the angle-based connective definition does not deal with very small circles gracefully. It would be useful to explore alternative definitions.

References

Bajaj C L, Pascucci V & Schikore D 1996 *in* 'Volume Visualization Symposium' ACM pp. 39–46.

Budny R V 1994 *Nuclear Fusion* **34**(9), 1247–1262.

Chan C Y & Ioannidis Y E 1998 *in* 'Proceedings of the 1998 ACM SIGMOD: International Conference on Management of Data' ACM press New York, NY, USA pp. 355–366.

Chan C Y & Ioannidis Y E 1999 *in* A Delis, C Faloutsos & S Ghandeharizadeh, eds, 'SIGMOD 1999, Proceedings ACM SIGMOD International Conference on Management of Data, June 1-3, 1999, Philadelphia, Pennsylvania, USA' ACM Press New York, NY, USA pp. 215–226.

Chang F, Chen C J & Lu C J 2004 *Comput. Vis. Image Underst.* **93**(2), 206–220.

Cignoni P, Marino P, Montani C, Puppo E & Scopigno R 1997 *IEEE Transactions on Visualization and Computer Graphics* **3**(2), 158–170.

Comer D 1979 *Computing Surveys* **11**(2), 121–137.

Crawford D, Ma K L, Huang M Y, Klasky S & Ethier S 2004 *in* 'VIS '04' IEEE Computer Society Washington, DC, USA pp. 59–66.

Dillencourt M B, Samet H & Tamminen M 1992 *J. ACM* **39**(2), 253–280.

Ethier S, Tang W M, Walkup R & Oliker L 2008 *IBM Journal of Research and Development* **52**(1-2), 105–116.

Fiorio C & Gustedt J 1996 *Theor. Comput. Sci.* **154**(2), 165–181.

Galil Z & Italiano G F 1991 *ACM Comput. Surv.* **23**(3), 319–344.

Gray J, Liu D T, Nieto-Santisteban M, Szalay A, DeWitt D J & Heber G 2005 *SIGMOD Rec.* **34**(4), 34–41.

Jones C, Ma K L, Ethier S & Lee W L 2008 *Computing in Science and Engineering* **10**(4), 20–29.

Jones C, Ma K L, Sanderson A & Jr. L R M 2007 *Journal of Physics: Conference Series* **78**, 012033.

Klasky S, Ethier S, Lin Z, Martins K, McCune D & Samtaney R 2003 *in* 'SC'2003' IEEE/ACM SIGARCH Phoenix, AZ.

Kloetzli J, Olano M & Rheingans P 2008 *in* 'I3D '08: Proceedings of the 2008 symposium on Interactive 3D graphics and games' ACM New York, NY, USA pp. 45–52.

Koegler W S 2001 *in* 'Proceedings of IEEE Visualization '01' pp. 461–464.

Lee W W 1983 *Physics of Fluids* **26**(2), 556–562.

Lee W W 2004 *Computer Physics Communications* **164**(1-3), 244–250.

Lin Z, Hahm T Z, Lee W W, Tang W M & White R B 1998 *Science* **281**, 1835–1837.

Lin Z & Lee W W 1995 *Phys. Rev. E* **52**(5), 5646–5652.

Livnat Y, Shen H W & Johnson C R 1996 *IEEE Transactions on Visual Computer Graphics* **2**(1), 73–84.

Lofstead J F, Klasky S, Schwan K, Podhorszki N & Jin C 2008 *in* 'CLADE '08: Proceedings of the 6th international workshop on Challenges of large applications in distributed environments' ACM New York, NY, USA pp. 15–24.

Ma K L, Painter J S, Hansen C D & Krogh M F 1994 *IEEE Comput. Graph. Appl.* **14**, 59–68. http://dx.doi.org/10.1109/38.291532

Musick R & Critchlow T 1999 *SIGMOD Rec.* **28**(4), 49–57.

O'Neil P 1987 *in* '2nd International Workshop in High Performance Transaction Systems, Asilomar, CA' Vol. 359 of *Lecture Notes in Computer Science* Springer-Verlag pp. 40–59.

Parker S E, Lee W W & Santoro R A 1993 *Phys. Rev. Lett.* **71**(13), 2042–2045.

Peters N 2000 *Turbulent Combustion* Cambridge University Press.

Post F H, Vrolijk B, Hauser H, Laramee R S & Doleisch H 2003 *Computer Graphics Forum* **22**, 775–792.

Röpke F K & Hillebrandt W 2005 *Astronomy & Astrophysics* **431**(2), 635–645.

Rotem D, Stockinger K & Wu K 2005 *in* 'CIKM' ACM pp. 648–655.

Rübel O, Geddes C G R, Cormier-Michel E, Wu K, Prabhat, Weber G H, Ushizima D M, Messmer P, Hagen H, Hamann B & Bethel W 2009 *Computational Science & Discovery* **2**(1), 015005. http://stacks.iop.org/1749-4699/2/i=1/a=015005

Samet H 1984 *ACM Comput. Surv.* **16**(2), 187–260.

Samet H 1990 *Applications of spatial data structures: Computer graphics, image processing, and GIS* Addison-Wesley Boston, MA, USA.

Schreiner J, Scheiclegger C & Silva C 2006 *Visualization and Computer Graphics, IEEE Transactions on* **12**(5), 1205–1212.

Shea J E & Brooks, III C L 2001 *Annual Review of Physical Chemistry* **52**(1), 499–535.

Shima Y, Murakami T, Koga M, Yashiro H & Fujisawa H 1990 *in* 'Proceedings of 10th International Conference on Pattern Recognition' pp. 655–658.

Sinha R R & Winslett M 2007 *ACM Trans. Database Syst.* **32**(3), 16.

Sinha R R, Winslett M & Wu K 2009 *in* 'SSDBM' Vol. 5566 of *Lecture Notes in Computer Science* Springer pp. 130–147. http://dx.doi.org/10.1007/978-3-642-02279-1_10.

Stockinger K, Shalf J, Bethel W & Wu K 2005 *in* 'IEEE Visualization 2005, Minneapolis, MN, October 23-28, 2005'. LBNL report LBNL-57511.

Stockinger K & Wu K 2006 Idea Group, Inc. pp. 179–202. LBNL-59952.

Suzuki K, Horiba I & Sugie N 2003 *Comput. Vis. Image Underst.* **89**(1), 1–23.

Tarjan R E 1975 *J. ACM* **22**(2), 215–225.

The HDF Group 2007 'HDF5 user guide' http://hdf.ncsa.uiuc.edu/HDF5/doc/H5.user.html.

Unidata 2007 'The NetCDF users' guide' http://www.unidata.ucar.edu/software/netcdf/docs/netcdf/.

Wang W X, Lin Z, Tang W M, Lee W W, Ethier S, Lewandowski J L V, Rewoldt G, Hahm T S & Manickam J 2005 *Journal of Physics: Conference Series* **16**, 59–64.

Wu K, Koegler W, Chen J & Shoshani A 2003 *in* 'Proceedings of SSDBM 2003' Cambridge, MA, USA pp. 65–74. A draft appeared as tech report LBNL-52535.

Wu K, Otoo E & Shoshani A 2001 Compressed bitmap indices for efficient query processing Technical Report LBNL-47807 Lawrence Berkeley National Laboratory Berkeley, CA.

Wu K, Otoo E & Shoshani A 2002 *in* 'Proceedings of SSDBM'02' Edinburgh, Scotland pp. 99–108. LBNL-49627.

Wu K, Otoo E & Shoshani A 2004 *in* 'VLDB' pp. 24–35. LBNL-54673.

Wu K, Otoo E & Shoshani A 2006 *ACM Transactions on Database Systems* **31**, 1–38.

Wu K, Otoo E & Suzuki K 2009 *Pattern Analysis & Applications* **12**(2), 117–135. http://www.springerlink.com/index/B67258V347158263.pdf.

Wu K, Shoshani A & Stockinger K 2010 *ACM Trans. Database Syst.* **35**(1), 1–52. http://doi.acm.org/10.1145/1670243.1670245.

Wu K, Stockinger K & Shosani A 2008 *in* 'SSDBM'08' pp. 348–365. Preprint appeared as LBNL Tech Report LBNL-173E.

Wu K & et al. 2009 *in* 'SciDAC 2009'.

Zweiback J, Cowan T E, Hartley J H, Howell R, Wharton K B, Crane J K, Yanovsky V P, Hays G, Smith R A & Ditmire T 2002 *Physics of Plasmas* **9**(7), 3108–3120.

DISCLAIMER