A Systematic Strategy for High Performance GIS

Liujian Qian Donna J. Peuquet

Department of Geography
The Pennsylvania State University
University Park, PA 16802 USA
qian/peuquet@geog.psu.edu

ABSTRACT

High Performance Computing is becoming increasingly important to large GIS applications, where the ability to store and access huge amounts of social and environmental data is crucial. In this paper we propose a systematic strategy using what we term a *virtual grid*; a quadtree-based decomposition of space used for the balanced allocation of distributed storage space. We also introduce the concept of a *quadtree spatial signature* as a highly compact spatial index for storage and retrieval. Our proposal is generic in the sense that it addresses problems at all levels of a high performance spatial database system, from the physical implementation of data storage and access methods to the user level for spatial query and analysis, with the underlying parallel computing model being an increasingly popular Network of Workstations (NOW).

1 Introduction: Parallelism in Different Levels of GIS

Efficient handling of spatial data is a growing concern for GIS as the amount of data available, indeed necessary, for addressing urban and environmental issues continues to increase. Terrabytes of data are now available from various governmental agencies. All levels of a GIS, from data storage to in-memory spatial operations and algorithms, can benefit from efficient partitioning and parallel computing strategies. A number of articles have been published in the topic of parallel strategies for GIS. Most of these, however, have concentrated on individual spatial data structures or algorithms [Wag92], [DDA92]. The more broad-ranging problem toward High

Performance GIS (HPGIS), however, is how to partition large amounts of complex and often highly interrelated data so that multiple computers can each be assigned a fraction of the data and complete the task cooperatively and effectively regardless of the task.

The physical implementation of any file structure involves allocating disk storage in units of fixed size. These are called disk blocks, pages or buckets, depending upon the level of description. We will use the term buckets. Conceptually, a bucket is simply a unit of storage containing a number of data records. Significantly enhanced performance in accessing large amounts of data can be achieved if the multiple buckets needed to satisfy a single query can be accessed simultaneously (i.e., in parallel). This can be achieved by distributing the set of buckets representing an individual data layer over multiple physical storage units. For locationally-based queries, maximal efficiency is achieved when the data are evenly distributed among the buckets on the basis of their locational value. However, the geographic distribution of data elements is typically highly variable, and often very clustered. Moreover, geographic distributions tend to be variable over time. Most existing GISs do not presume any correspondence whatsoever between conceptual ordering and physical distribution in storage. Since most GISs store data in unordered pages of a file, some kind of spatial indexing must be employed in order to avoid the inspecting large portions of the database unnecessarily. The *virtual grid* organization described in this paper is proposed as an effective method for mapping of the geographical distribution of a data layer to a physical storage distribution. The method proposed has it's root in a balanced file structure called Grid File originally developed for a non-spatial context in [NH84].

Spatial indexing structures currently used for geographic databases include K-D-B trees, R-trees, Grid Files, and quadtrees, among many others (see [Sam90] for a thorough review). These indexing structures usually store key-pointer pairs where the 'key' is the identifying spatial attribute or shape, and 'pointer' is the address of the whole record stored in the unordered data file. The idea behind all spatial indexing schemes is to subdivide a large search space into multiple smaller search spaces, so that only those potentially relevant (and much smaller) parts need to be actually examined for given query predicates.

There are important differences among the ways various indexing structures split the search space. The two fundamental approaches, following the two basic types of geographic data models, can be described as space-based vs. object-based partitionings [NH84]. An index structure where partitions are designed to contain, and to not subdivide objects, such as in R-trees [Gut84], has object-based partitions. Other examples of this partitioning strategy are K-D trees [Ben75] where the partitioning boundaries are drawn based on the location of the point data being indexed.

If, on the other hand, the partition boundaries are drawn symmetric to all dimensions regardless of any particular object's location it is a space-based partitioning scheme. Quadtrees are the best-known example of this partitioning strategy. Another distinction can be made among space-based partitioning techniques depending on whether or not the partitioning is regular. The area quadtree is a regular space-based partition index, because it always splits an area into four equal parts. In contrast, the Grid File as adopted in another high-performance GIS context [CSZ93] is an irregular space-based partitioning scheme, since it divides the space into variable intervals along each dimension. Below is a figure that illustrates these different partitioning schemes. Figure 1(a) represents an example partitioning for an R-tree, where each rectangle is a minimum rectangle that contains a set of objects or smaller bounding rectangles. Figure 1(b) represents the partitioning for a K-D tree. The irregular spatial subdivision of a Grid File is depicted by Figure 1(c) while the Figure 1(d) illustrates the regular spatial subdivision of an area quadtree.

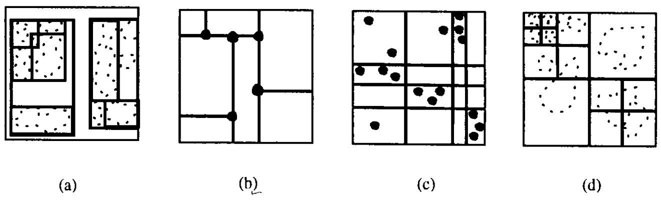

Figure 1: Patterns of Spatial Partitions

Although the R-tree, K-D tree and derivative methods have gained attention recently for indexing geographic databases for storage and retrieval, the area quadtree provides two significant advantages for parallel spatial indexing in a GIS context due to the regular subdivision of space. First, this allows for an even allocation of data records into buckets to be performed more easily. Although homogeneous areas or objects may be subdivided into different buckets for storage, the even allocation of data among multiple buckets becomes a more straightforward task. Second, the more direct mapping between geographic location and bucket allocation makes parallel retrieval of multiple layers on the basis of location a much simpler task. These features will be further explained in the discussion later.

2 Balanced Storage Allocation

In this section we describe the notion of the *virtual grid*, a quadtree based space decomposition and spatial data storage strategy.

2.1 Concept of the Virtual Grid Spatial Data Storage Model

The basic idea of the virtual grid is quite simple. The space or geographic area of interest is subdivided in the same manner as in normal area quadtree subdivisions. When to stop the decomposition is determined by the total volume of data within the resultant cells after each subdivision. Thus, an area with sparse data will result in few subdivisions and an area with a dense distribution will result in relatively more subdivisions. The result of the decomposition is a set of variably-sized tiles that covers entire area, with each tile corresponding to a leaf node in the quadtree. For each spatial tile, there is one data "bucket" associated with it that stores all the data which fall within the geographical area represented by that tile. We call our quadtree-based storage model a *virtual grid* for several reasons. First, the spatial resolutions of the stored data are not hierarchical, and the subdivision does not need to continue until only homogeneous spatial data are contained in the tile. The resulting set of partitions is a grid with variable-size tiles. Second, this irregular grid is a partitioning scheme only, with the individual tiles created by this partitioning potentially dispersed on different disks, as well as on different machines. Third, the subdivision bears no relation to the storage format used for the data themselves. The data within the subdivisions for any given data layer can be stored as vectors or pixels in accordance with the nature of the data.

Figure 2 illustrates the relationships between the individual tiles and data buckets for a given partitioning. The philosophy behind the use of the quadtree as a regular, hierarchical, location-based partitioning method stems from the simplicity of the area quadtree scheme, such as described in [Peu84]. The essential notion here is to provide a mechanism to even-out physical storage for what can be highly uneven and variable data distributions over geographic space, while still allowing efficient data retrieval for overlay and other layer-based operations.

Figure 2(a) represents a set of geographical data forming a layer of, say, land use on a set of islands. Figure 2(b) is the tiling of the corresponding layer, where there are 13 tiles numbered from 0 to 12. In figure 2 each tile is seen to correspond to a quadtree leaf node, with the whole quadtree (including both internal and leaf nodes) shown in Figure 2(c). If the data for a particular region are more dense than others, that region is always further divided into smaller tiles so that the volume of data contained within

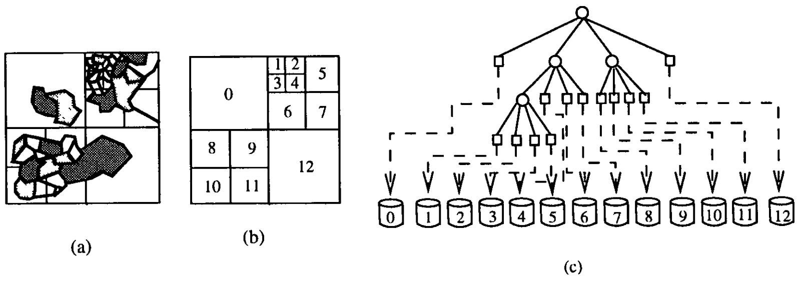

Figure 2: virtual grid, tiles, buckets and their relationships

any individual tile is approximately equal, regardless of the size of the geographical area represented.

2.2 Construction of a Virtual Grid

Now we consider how to construct a virtual grid for an individual layer of geographic data. It is worth mentioning that during the construction of the virtual grid we actually maintain a quadtree in memory to keep track of the levels and areas of the subdivision. As stated above, the level at which subdivision stops is totally determined by the amount of data contained within a tile. This amount is dictated by the size of a single bucket. All buckets are equal in their maximum size or capacity.

Given the bucket size, we now briefly describe how to store raw data records into a database in the sequence of Morton order, or equivalently speaking, how to the build the virtual grid (and hence the corresponding quadtree) in a bottom-up manner, in contrast to the normal top-down method.

The procedure is that we sort the raw data records first using the Morton order (or Z-order). This may involve calculating the Morton address for each record and sorting the records based on their addresses. Once sorted, we scan the records and add them sequentially into a bucket until it's capacity is reached. At this point we examine the Morton address of the last spatial object stored into the bucket, from which we are able to determine what is the smallest tile or quadtree node that should be associated with the bucket.

The following is an example showing how we load a set of spatial point data into a virtual grid using the bottom-up process.

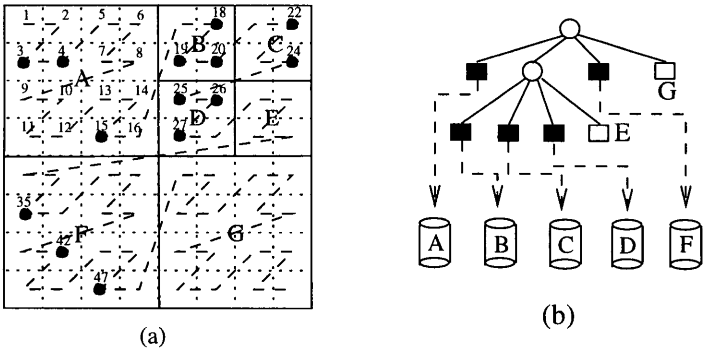

Figure 3: construction of a virtual grid

In Figure 3 we assume each bucket can hold 3 points, and the 15 data points to be stored have been labeled with Morton addresses and sorted. Upon the addition of the first three data points into a bucket, with the last point having a Morton address of 15, we form the corresponding tile A (which is associated with the bucket,) and it's three sibling quadrants, which have no buckets associated yet. Repeat this procedure and we eventually have 5 buckets with corresponding tiles being those black leaf nodes of the quadtree shown in figure 3.

The bucket size, as an important system parameter, has to be determined before raw data can be loaded into the database. As for most high performance database storage strategies, there is the issue of optimization: What bucket size provides the optimal performance given the distribution characteristics for a specific type of data? If the bucket size is too large, then the speed efficiency of doing operations in parallel is not fully realized. If the bucket size is too small, then the increased traffic of I/O operations caused by doing too many separate data access operations simultaneously can by itself slow down a computer, and indeed an entire computer network.

During the construction of a virtual grid, splitting of (large) objects may occur as a consequence of reallocating an overflow bucket. In our virtual grid storage model, when a bucket overflows, we subdivide it's corresponding tile into 4 equal quadrants; and reallocate the data in the bucket into at most four new buckets, depending on the data distribution in the original tile. During this subdivision and reallocation, if an object is large enough to cover more than one sub-tiles, we will split it and store partial objects into new buckets where they belong.

The last step of constructing a virtual grid will always be to construct a compact description of it, called a *quadtree spatial signature*, to be described in next section, and store the signature in the database catalog along with other meta information of a data layer.

3 Quadtree Spatial Signature

The whole idea behind the *virtual grid* notion is that when time-consuming operations over large volumes of data are to be performed in parallel, using a "divide and conquer" approach, we need to physically map buckets onto multiple physical storage areas in order to achieve three things; 1.) physically balance the load for large data access operations, 2.) provide an efficient indexing mechanism so that the physical location in storage of any given data element can be determined quickly and with a minimum amount of disk access, and 3.) when doing spatial join of multiple layers, we can have an intuitive approach to associate geographically-relevant buckets from different layers onto multiple computers, due to the nature of the *virtual grid.*

In this section we present our method for achieving this, utilizing something we call the *quadtree spatial signature.*

We define a *quadtree spatial signature* as a compact mapping of where data elements are located spatially into a search path within quadtree-space. Since quadtree subdivision was used to create the data tiles, this provides a quick index that quickly eliminates consideration of any geographical areas that are "blank" as far as the data in question. It also provides a rapid conversion from geographic space to storage location, and due to the regularity, it provides an intuitive method of assigning buckets of different data layers based on geographical relationships, which will greatly enhance the effectiveness of parallel processing of multiple layers.

Given a quadtree, it's spatial signature is simply a set of bitmap strings, with each level of the quadtree having one bitmap string, as shown in Figure 4. For any given level of a quadtree, the bitmap string is an array of value '00', '01' or '11's, based on the type of a corresponding tree node in that level. An internal node is labeled as '01', while empty leaf nodes are labeled as '00', and black leaf nodes (which have data in corresponding buckets) are '11's. Note that even if, for a particular level, there is no tree nodes at all, we still assign '00's to their corresponding positions in the bitmap array, because our bitmap strings represent a complete quadtree.

At a first glance the size of our signature may seem to be quite large, since it records information for every nodes of a complete quadtree. But in implementation one can always use compression methods such as run-length encoding to store the bitmap, as there are many repeated '0's or '1's; and the size of such a quadtree signature is actually quite small (about less than hundred KBytes for a quadtree of 10 levels).

Our spatial quadtree signature has the following properties:

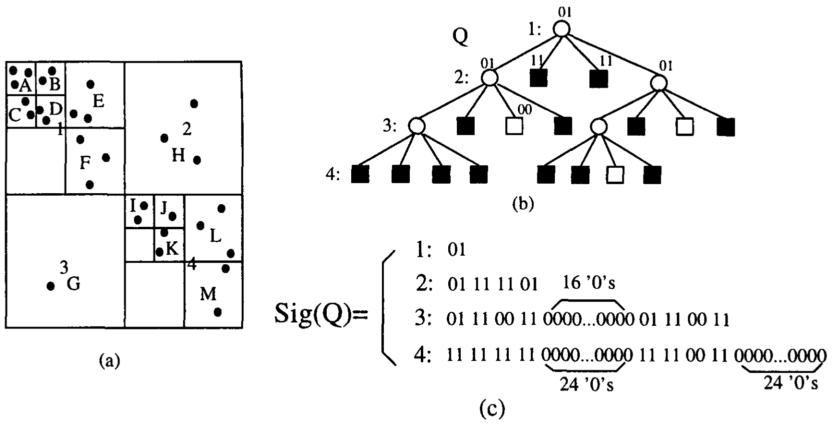

Figure 4: An example of quadtree spatial signature

- Morton addresses or Z-order numbers are used as an index inside the bitmap array of each level. The Morton address for each quadtree node in a given level is calculated using the spatial scale corresponding to that level. In this way the status of any tree node in that level can be obtained by calculating it's Morton address and index into the bitmap array. For example, in Figure 4, the bitmap string for level 2 is '01', '11', '11', '01', indicating that the four nodes (whose Morton addresses are labeled as 1,2,3,4 in Figure 4(a)) are internal nodes, leaf nodes, leaf nodes and internal nodes respectively.

- For an internal node in a higher level with Morton address N, the status of it's four quadrants can be obtained by looking up in the bitmap array of the next level at those positions from (N-1)*4 to N*4. Similar direct determination is possible from a child node to it's parent node.

- For a bitmap array, only those elements with value '11' are leaf nodes that have buckets associated with them. The number of tiles of a particular size in our virtual grid can be easily obtained by scanning the corresponding level's bitmap array and count the number of elements with value '11'.

- This set of bitmap arrays can serve as a spatial index; and mappings between '11'-valued elements (tiles) and actual storage buckets can be easily established based on the level number and array index for the elements. This removes the need for disk-based spatial indexing structures such as R-trees or K-D-B trees; and converts spatial search into in-memory calculations based on the set of bitmap arrays and searching rectangles (used in range search). The bitmap arrays also remove the need for maintaining expansive quadtree structures in memory, since all the information has been retained in the set of bitmap arrays.

- The data distribution information at different spatial scales (tree levels) are maintained in the different levels of bitmap arrays. This is useful for scale-sensitive operations.

One of the most important rules to follow in applying parallel strategies to spatial operations, is that the load allocation algorithm must take into account not only how to balance the amount of data to be processed individually, but also how to assure that the data allocated to a computer are as geographically clustered as possible in data-space.

Below we will briefly describe how to use our quadtree spatial signature in balancing load that meets these two requirements, on a simple but increasingly popular parallel computing model of NOW (Network of off-the-shelf Workstations), where a master computer allocates data buckets to multiple slave computers so that spatial operations can be performed locally and concurrently by each slave on the protion of data it receives.

The main allocation procedure (for single-layer based operations), is that for each bitmap array, we maintain a cursor indicating the next un-allocated bucket (tile) in that level. Starting from the left-most '11' element of all the arrays, which is the first tile in our virtual grid, advance the cursor in that array for following consecutive '11's, until the limit of bucket number for one computer is reached or an element of different value is encountered. In the first case, we allocate the set of buckets traversed to the first slave, and continue for the next slave; in the second case, where we encounter a different type of element, we will need to look at the array either above or below the current array, based on whether the element with non-'11' value is an empty node or internal node. We then advance the cursor in the other array in a similar manner until we find the first '11' elements, and records all the consecutive elements of '11's as the set of buckets to be allocated to next available slave computer. We then repeat this step until all the buckets have been allocated; or the available slave computers are exhausted.

For the example virtual grid of Figure 4, if we assume each processor can handle at most 3 buckets of data, then using the above algorithm will result the sets of tiles like (A,B,C), (D,E,F), (G, H, I), (J,K,L), and (M). This allocation is well-balanced and presents a good level of geographical adjacency among tiles within each working set.

4 Conclusion

In this paper we presented a systematic strategy for high performance GISs that need to deal with very large data volumes. We proposed a *virtual grid* structure for distributing the storage of unevenly distributed spatial data in an even way; and suggested that a simple structure called the *quadtree spatial signature* can be effectively used in guiding the dividing of work load for time-consuming spatial operations. Finally, we want to emphasize that our method may also be extended into hexagonal and triangular tessellations as well.

References

[Ben75] J. L. Bentley. Multidimensional search trees used for associative searching. *Communications of the ACM*, 18:509–517, 1975.

[CSZ93] Mark Coyle, Shashi Shekhar, and Yvonne Zhou. Evaluation of disk allocation mehtods for parallelizing spatial queries on grid files. In *Data Engineering Conference*, 1993.

[DDA92] Yuemin Ding, Paul J. Densham, and Marc P. Armstrong. Parallel processing for network analysis: Decomposing shortest path algorithm for MIMD computers. In *Proceedings of the 5th International Symposium on Spatial Data Handling*, pages 682–691, Charleston, South Carolina, 1992.

[Gut84] A. Guttman. R-trees: A dynamic index structure for spatial searching. *SIGMOD Record*, 14, No. 2:47–57, 1984.

[NH84] J. Nievergelt and H. Hinterberger. The grid file: An adaptable, symmetric, multikey file strcuture. *ACM TODS*, 9(1):38–71, 1984.

[Peu84] Donna J. Peuquet. A conceptual framework and comparison of spatial data models. *Cartographica*, 21:66 –113, 1984.

[Sam90] Hanan Samet. *The Design and Analysis of Spatial Data Structures.* Addison-Wesley, Reading, Mass., 1990.

[Wag92] Daniel F. Wagner. Synthetic test design for systematic evaluation of geographic information systems performance. In *Proceedings of the 5th International Symposium on Spatial Data Handling*, pages 166–177, Charleston, South Carolina, 1992.