DATA-PARALLEL SPATIAL JOIN ALGORITHMS*

Erik G. Hoel†
Geography Division
Bureau of the Census
Washington, D.C. 20233

Hanan Samet
Computer Science Department
Center for Automation Research
Institute for Advanced Computer Sciences
University of Maryland
College Park, Maryland 20742

Abstract – *Efficient data-parallel spatial join algorithms for* PMR *quadtrees and R-trees, common spatial data structures, are presented. The domain consists of planar line segment data (i.e., Bureau of the Census TIGER/Line files). Parallel algorithms for map intersection and a spatial range query are described. The algorithms are implemented using the SAM (Scan-And-Monotonic-mapping) model of parallel computation on the hypercube architecture of the Connection Machine.*

INTRODUCTION

The spatial join is one of the most common operations in spatial databases. This term is usually used in conjunction with a relational database management system [9]. In that context, a join is said to combine entities from two data sets into a single set for every pair of elements in the two sets that satisfy a particular condition. Traditionally, these conditions involve specified attributes that are common to the two sets. In the spatial variant of the join, the condition is usually interpreted as being satisfied (i.e., two elements are joined) when the elements of the pair cover some part of the space that is identical.

In this paper we compare the parallel implementation of a couple of variants of spatial join for databases consisting of collections of line segments that are organized using hierarchical spatial data structures. These collections correspond to maps. The first variant is a simple intersection operation which seeks to find all line segments of a given type that intersect line segments of another type. The second variant is a bit more complex. It seeks to find all line segments that lie within a given distance of line segments of another type. This is a spatial range query. It is also known as a corridor or a buffer zone in GIS [22], or image dilation in image processing [19]. For example, suppose that we have one map corresponding to the roads in the United States and another map corresponding to the border of Colorado. An instance of the first spatial join is a query that seeks to determine all roads that cross the border of Colorado, while an instance of the second spatial join is a query that seeks to determine all roads that lie within 10 miles of the border of Colorado.

We assume that the line segments are represented using hierarchical spatial data structures [20, 21]. In this paper we focus on representations that sort the data with respect to the space that it occupies. This results in speeding up operations involving search. The effect of the sort is to decompose the space from which the data is drawn (e.g., the two-dimensional space containing the lines) into regions called *buckets*. One approach known as an R-tree [13] buckets the data based on the concept of a minimum bounding (or enclosing) rectangle. In this case, lines are grouped (hopefully by proximity) into hierarchies, and then stored in another structure such as a B-tree [8]. The drawback of the R-tree is that it does not result in a disjoint decomposition of space — that is, the bounding rectangles corresponding to different lines may overlap. Equivalently, a line may be spatially contained in several bounding rectangles, yet it is only associated with one bounding rectangle. This means that a spatial query may often require several bounding rectangles to be checked before ascertaining the presence or absence of a particular line.

The second approach is based on a decomposition of space into disjoint cells. Each line is decomposed into disjoint sublines such that each of the sublines is associated with a different cell. There are a number of variants of this approach. They differ in the degree of regularity imposed by their underlying decomposition rules and by the way in which the cells are aggregated. The price paid for the disjointness is that in order to determine the area covered by a particular line, we have to retrieve all the cells that it occupies. The approach that we study here is known as a PMR quadtree [17]. It is based on recursively decomposing the space into four equal area blocks on the basis of the number of lines that it contains. The decomposition process can be implemented by a tree structure.

We use the SAM (Scan-And-Monotonic-mapping) model of parallel computation [3]. It is defined by one or more linearly ordered sets of processors which allow element-wise and scan-wise operations to be performed. A *scan* operation [4, 16] takes an associative operator $\bigoplus$, a vector $[a_0, a_1, \cdots, a_{n-1}]$, and returns the vector $[a_0, (a_0 \bigoplus a_1), \cdots, (a_0 \bigoplus a_1 \bigoplus \cdots \bigoplus a_{n-1})]$. Both within and between each linearly ordered set of processors, monotonic mappings may also be performed. A *monotonic mapping* is defined as a mapping between two linearly ordered processor sets where the destination processor indices are a monotonically increasing function of the source processor indices. Being a subset of the scan-model [5], the SAM model considers scan operations as taking unit time, thus allowing sorting

*This work was supported in part by the National Science Foundation under Grant IRI-90-17393.

†Also with the Center for Automation Research at the University of Maryland.

operations to be performed in $O(\log n)$.

There are a number of possible ways to implement the spatial join operations. In order to compare the two representations we try to implement comparable algorithms. We have chosen a bottom-up approach as we are working with data parallel algorithms which are based on assigning a processor to each object (i.e., line segment in our case). Of course, a top-down approach [6] could also have been used; we leave this to future work. The simplest algorithm checks every line segment against every other line segment for intersection or satisfaction of the within condition. Ideally, we would like to take advantage of the decomposition of underlying space from which the lines are drawn and avoid testing lines that cannot possibly intersect. This is quite easy when using the PMR quadtree as it provides a sort of the underlying space and a partition into disjoint blocks. Moreover, since it is easy to identify blocks in the two maps that correspond to the same parts of the underlying space, we can avoid checking for the intersection of lines that cannot possibly intersect. However, this is not possible for the R-tree as the bounding rectangles are not disjoint. In addition, the R-tree data structure does not contain any information as to which bounding rectangles in one map overlap with bounding rectangles in the other map. This means that little of the search space can be pruned while performing the operations. The problem is that although the R-tree's main utility is to enable the user to distinguish easily between occupied and unoccupied regions in a particular map, it does not provide a means of correlating the contents of one map with another map. Unfortunately, this is exactly the ability that is needed to implement spatial join algorithms efficiently. This places the R-tree is at a considerable disadvantage in comparison to the PMR quadtree.

The rest of this paper is organized as follows. Section 2 deals with the PMR quadtree. It contains a definition of its sequential and parallel variants as well as a description of an algorithm for constructing the latter. This is followed by the algorithms for the two spatial join algorithms. Section 3 is concerned with the R-tree and is organized in the same way as Section 2. Section 4 compares the two parallel data structures in terms of performance data for the two spatial join algorithms on a Thinking Machines CM-5 parallel computer. Section 5 contains concluding remarks as well as a discussion of topics for future research.

Sequential PMR Quadtrees and R-trees

The first spatial data structure that we consider is the PMR quadtree. It is a member of a family of data structures that adaptively sort the line segments into buckets of varying size. There is a one-to-one correspondence between buckets and blocks in the two-dimensional space from which the line segments are drawn. There are a number of approaches to this problem [20]. They differ by being either vertex-based or edge-based. Their implementations make use of the same basic data structure. All are built by applying the same principle of repeatedly breaking up the collection of vertices and edges (making up the polygonal map) into groups of four blocks of equal size (termed *siblings*) until obtaining a subset that is sufficiently simple so that it can be organized by some other data structure.

The PMR quadtree [17, 18] is edge-based. It makes use of a probabilistic splitting rule. A block is permitted to contain a variable number of line segments. The PMR quadtree is constructed by inserting the line segments one-by-one into an initially empty structure consisting of one block. Each line segment is inserted into all of the blocks that it intersects or occupies in its entirety. During this process, the occupancy of each affected block is checked to see if the insertion causes it to exceed a predetermined *splitting threshold.* If the splitting threshold is exceeded, then the block is split *once*, and only once, into four blocks of equal size. The rationale is to avoid splitting a node many times when there are just a few very close lines in a block. In this manner, we avoid pathologically bad cases. For more details, see [17].

A line segment is deleted from a PMR quadtree by removing it from all the blocks that it intersects or occupies in its entirety. During this process, the occupancy of the block and its siblings (the ones that were created when its predecessor was split) is checked to see if the deletion causes the total number of line segments in them to be less than the predetermined splitting threshold. If the splitting threshold exceeds the occupancy of the block and its siblings, then they are merged and the merging process is recursively reapplied to the resulting block and its siblings. Notice the asymmetry between the splitting and merging rules.

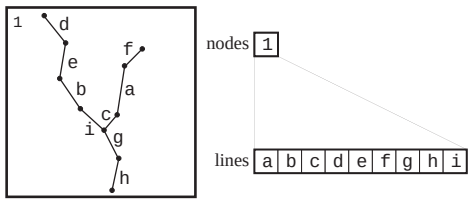

Figure 1: Initial PMR quadtree processor assignments.

1.0

Data-Parallel PMR Quadtree

The algorithm for building a parallel PMR quadtree proceeds as follows. Initially, a single processor is assigned to each line in the data set, and one processor to the resultant PMR quadtree as depicted for the sample data set in Figure 1. Via a downward scan operation, the number of lines associated with the single node processor (9 in the example) is determined and then passed to the node processor. If the number of lines associated with the node processor exceeds the splitting threshold (in our example, the splitting threshold is 2), then the

node must be split into four subnodes and each of the lines must be regrouped. according to which nodes it intersects. The regrouping is achieved with a series of un-shuffle operations as described in greater detail in conjunction with the presentation of the parallel R-tree build process in Section 3.1. Note that a line may span two nodes, thus requiring the line to be duplicated and hence an additional processor in the line processor set is allocated for it (termed *cloning* [3]). For example, consider line a in Figure 2 which intersects both the NW and the NE nodes in the quadtree. The result of the first node subdivision is shown in Figure 2.

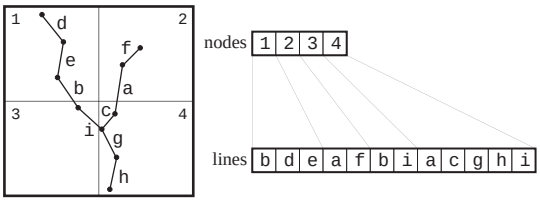

Figure 2: Result of the first node subdivision, line duplication and un-shuffling.

1.0

Continuing with this iterative process, each line segment group (i.e., a group of line segment processors) determines the number of lines it contains, and then communicates the count to the associated node processor. For example, in Figure 2, the first line segment group transmits a count of three to node 1, the fourth line segment group transmits a count of two to node 2, etc. Each of the node processors then determines whether or not the transmitted line count exceeds the splitting threshold. If the splitting threshold is exceeded, the node will subdivide and the associated lines will be regrouped according to which of the resulting subnodes they intersect. If Figure 2, the NW and SE nodes will subdivide, resulting in the situation depicted in Figure 3.

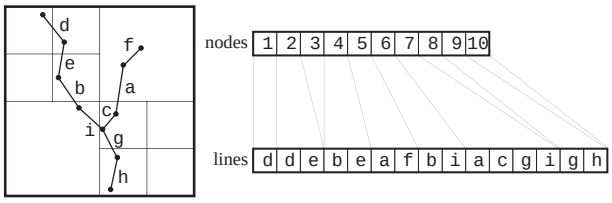

Figure 3: Result of the second node subdivision.

1.0

This iterative subdivision process will continue until all nodes in the parallel PMR quadtree have a line count less than or equal to the splitting threshold, or the maximal resolution of the quadtree has been reached (i.e., a node of size 1×1). Note that in the degenerate case, even at the maximal resolution of the quadtree, it is possible that the number of lines associated with a node exceeds the splitting threshold. For practical splitting thresholds (i.e., eight and above), this situation is exceedingly rare and will not cause any algorithmic difficulties provided that the quadtree algorithms do not assume an upper bound on the number of lines associated with a given node.

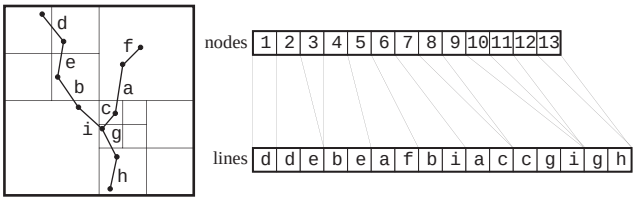

Figure 4: Result of the PMR quadtree build process.

1.0

The result of the third and final subdivision for our example data set is shown in Figure 4. Note that one of the quadtree nodes (node 9) still has its splitting threshold exceeded. To facilitate the discussion of the algorithms, this node will not be further subdivided.

The data-parallel PMR quadtree building operation is of complexity $O(\log n)$, where each of the $O(\log n)$ subdivision stages requires $O(1)$ computations (a constant number of scans and re-shuffles).

PMR Quadtree Map Intersection

In the following algorithm description, assume that we are starting with two data-parallel PMR quadtrees; one termed the *source* quadtree, and the second termed the *target* quadtree. The quadtrees are of equal size (i.e., they represent the same $2^n \times 2^n$ area). The source quadtree will contain the reference set of lines from which to intersect against (e.g., the border of the city), and the target quadtree contains the lines which will be determined to intersect the objects in the source quadtree (e.g., the roads found in the county).

1.0

Given the data-parallel source and target PMR quadtrees, we first establish a correspondence between the source and target quadtree nodes. This will facilitate the lessening of communication between the two quadtrees when performing the actual intersection. While establishing the source and target node correspondence a third temporary set of quadtree nodes (termed the mapping quadtree), is employed. The mapping quadtree is discarded following completion of the operation.

1.0

The mapping quadtree initially consists of a single large node, equal in physical size to the exterior dimension of the source and target quadtrees (i.e., $2^n \times 2^n$). The single mapping node is associated with the entire collection of both source and target quadtree leaf nodes (as an example, consider the situation depicted in Figure 5). The mapping quadtree nodes (of which

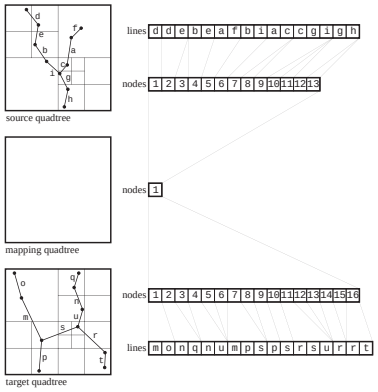

Figure 5: Example of a source and target PMR quadtree (each with a splitting threshold of 2) and a mapping quadtree prior to the first mapping node subdivision.

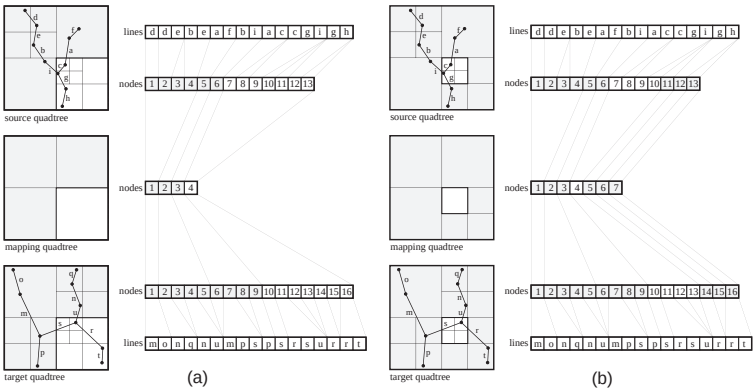

Figure 6: (a) Mapping quadtree nodes at the completion of the first subdivision phase with completed mappings shaded. (b) Mapping quadtree nodes at the completion of the second subdivision phase.

there is initially only one) are then repeatedly subdivided until each mapping node is associated with either a single source node, or a single target node. Essentially, the mapping nodes are subdivided until there is a one-to-one, one-to-many, or many-to-one relationship established between the source and target nodes. For our example dataset shown in Figure 5, the single mapping node is associated with thirteen source nodes, and sixteen target nodes, thus the mapping node must be split and the source and target nodes reassigned to the appropriate mapping node. The result of the first mapping node split is shown in Figure 6a. Continuing with this process, the shaded mapping nodes 1, 2, and 3 in Figure 6a have satisfied the termination condition and do not need to be further split. Mapping node 4 must further subdivide as it is associated with seven source nodes and seven target nodes. The result of mapping node 4 in Figure 6a splitting is shown in Figure 6b. The final mapping node split (of mapping node 4 in Figure 6b) results in the situation depicted in Figure 7a, where each mapping node corresponds to either a single source or a single target node.

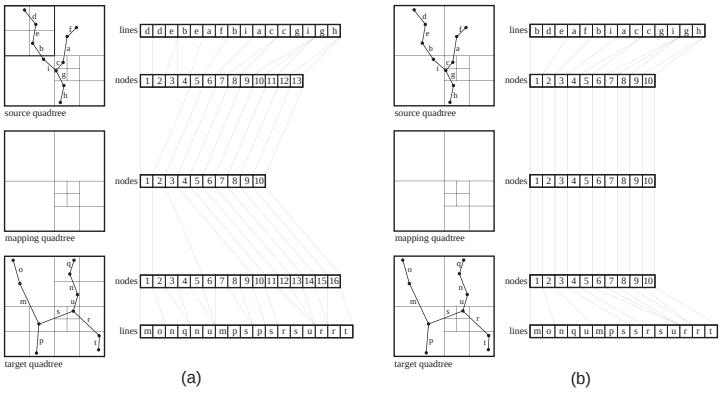

Figure 7: (a) Mapping quadtree nodes at the completion of the subdivision phase which creates one-to-one, many-to-one, or one-to-many mappings between the source and target quadtrees. (b) Quadtree nodes highlighting the one-to-one source to target node correspondence after node merging.

1.0

Once the mapping quadtree subdivisions are completed (see Figure 7a where each mapping node corresponds to either a single source or target node), there exists a one-to-one, many-to-one, or one-to-many association between source and target nodes. The source and target quadtree nodes are then merged as necessary in order to establish a one-to-one relationship between the nodes in the two input maps. For instance, if there are four source nodes associated with a single target node (refer to the first mapping node in Figure 7a), then the four source nodes (which will share the same parent node in the quadtree decomposition) are merged together (with duplicate line segments removed). This will result in a one-to-one correspondence between these source and target nodes (see Figure 7b). At the completion of the source and target node merging, the mapping quadtree may be discarded.

The actual process of determining the line segment intersections begins with each source node broadcasting the endpoints of all associated line segments (i.e., all the line segments that are found in the quadtree node) to the set of line segments in the associated target quadtree node. Figure 8 highlights the source to target line communication for the example dataset (note that the shaded nodes and lines represent inactive processors for which no action is necessary as there are either no source lines or target lines associated with the nodes). This is accomplished by the first line processor associated with each active source node (where an *active* line or node is defined as one that is participating in the current operation) passing its endpoints to the

first line processor in the corresponding target quadtree node. In Figure 8, the first line processors are shown with arrows emanating from them directed at the corresponding source nodes. The source line endpoint coordinates are then shared among all line processors in the associated target quadtree node via a sequence of scan operations.

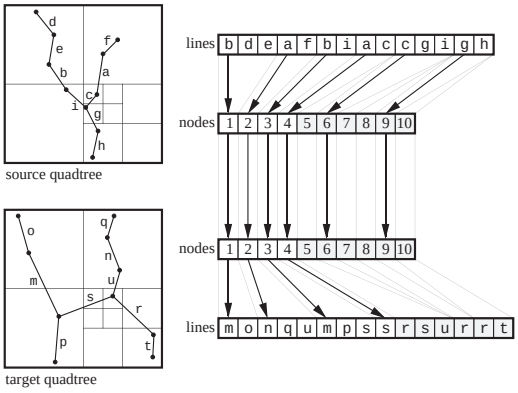

Figure 8: Source to target quadtree line endpoint communication channels for determining intersection. Shaded nodes and lines are not the recipient of any communications.

1.0

Each active target line processor then simultaneously determines whether or not the line that it represents intersects the broadcasted source quadtree line segment. If the target line intersects the broadcasted source line, the target line is so marked. Continuing this process, the collection of second line processors associated with each active source quadtree node passes their coordinates to the first processor in each active associated target node. Again, the line coordinates are then communicated among all line processors in the target node via a sequence of scan operations, and each target line processor determines in parallel whether or not it intersects the source line. Once all active source line processors have transmitted their coordinates to the associated target line processors, the intersection operation is complete and all target lines intersecting any of the source lines have been determined.

PMR Quadtree Spatial Range Query

The data-parallel PMR quadtree spatial range query algorithm proceeds in a fashion similar to the intersection algorithm where the quadtree decomposition is employed to maximize the number of parallel operations. In the following description, again assume that there are two input data-parallel PMR quadtrees of the same size. We will term the quadtree that contains the line segments to be expanded (also referred to as the *expansion set*) the source quadtree, and the quadtree containing the line segments from which to test for intersection with the expansion set the target quadtree.

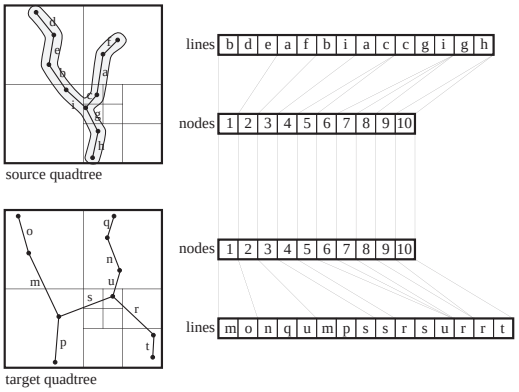

Figure 9: One-to-one source to target node mapping and region formed by the expansion set and expansion radius.

1.0

The algorithm begins by establishing a mapping between source and target nodes in an identical manner to that employed at the beginning of the intersection algorithm. Once the one-to-one source to target node mapping is established (refer to Figure 9 for an example source to target node mapping and an example expansion region which is denoted by the gray region superimposed on the source quadtree), the process of determining all target lines that intersect the region defined by the source line expansion set and the expansion radius proceeds in an iterative fashion.

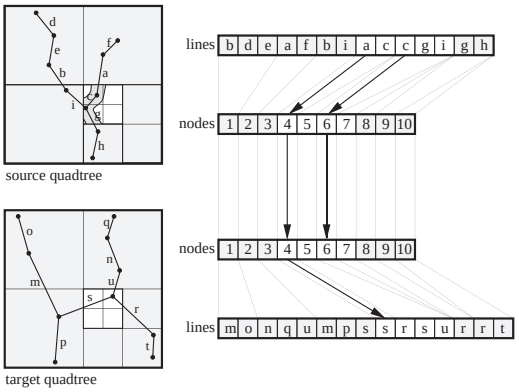

Figure 10: Active source and target nodes of minimal size (with other larger inactive nodes shaded) during the first iteration of line communications.

1.0

The spatial range query algorithm operates on a single size set of nodes at a time (i.e., all nodes of size $2^m \times 2^m$ where $m \leq n$), iterating upward from the smallest sized nodes to the root node in the quadtree representation. Given the set of the smallest sized nodes in the source quadtree (corresponding to the unshaded

nodes in Figure 10), in parallel, each broadcasts the endpoints of all associated line segments (i.e., all the line segments that are found in the quadtree node) to the set of line segments in the associated target quadtree node. This is accomplished in a similar fashion as was done with the intersection algorithm, with the first line processor associated with each active source node passing its endpoints to the first line processor in the corresponding target quadtree node. These coordinates are then shared among all line processors in the associated target quadtree node via a sequence of scan operations. Each active target line processor then simultaneously calculates the Euclidean distance between itself and the communicated source line. If the separation distance is less than the radius of expansion, the target line is marked as laying within the space defined by the source expansion set.

Continuing this process, the second line processor associated with each active source quadtree node passes its coordinates to the first processor in each active associated target node. Again, the line coordinates are then communicated among all line processors in the target node via a sequence of scan operations, and each target line processor calculates the distance between itself and the source line. Once all active source line processors have transmitted their coordinates to the associated target line processors, the communication stage for the currently active quadtree node size is complete. For example, only line segment s is found to intersect the expanded region in the first iteration.

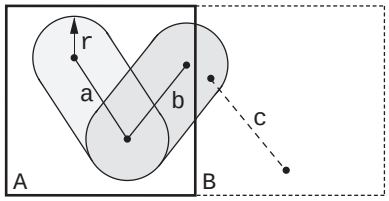

Figure 11: Example where one source line a may be deleted, and the second line b may not be deleted during the source node merge phase. For the given radius of expansion r, line b's expansion region might intersect a target line c in another currently unassociated node B.

1.0

Before the source node iteration continues (i.e., nodes of twice the current active node size are made active), each of the currently active source and target sibling nodes is merged. As an optimization to lessen the number of source line segment communications, all source line segments in the currently active source nodes whose distance from the border of their corresponding block is greater than the expansion radius are deleted. If a source line lies at distance less than the expansion radius from the border of the source node's corresponding block, then the source line must be retained for later rebroadcast. This is because a source line's region of expansion (the area within the expansion radius of the line) may intersect target lines that are not associated with the block corresponding to the source line's node (i.e., a target line may lie very close to the border in an adjoining node). For example, consider the situation depicted in Figure 11 of two line segments a and b in a source node corresponding to block A. Given the example situation (with the expansion radius r), a may be safely deleted as its expansion region can not possibly intersect any other blocks outside of A, while b may not be deleted as it's expansion region intersects other blocks (i.e.,block B). Note that there is no need to delete any target lines as all the target lines are checked for intersection with a source line in parallel. Thus removal of a target line does not affect performance. Of course, if there are more target lines than processors, then this may be a useful optimization.

The result of the node merging and line deletion is depicted for our example dataset in Figure 12. Note that no source lines were deleted as each of their expansion regions intersected the border of the blocks corresponding to their nodes in the initial quadtree.

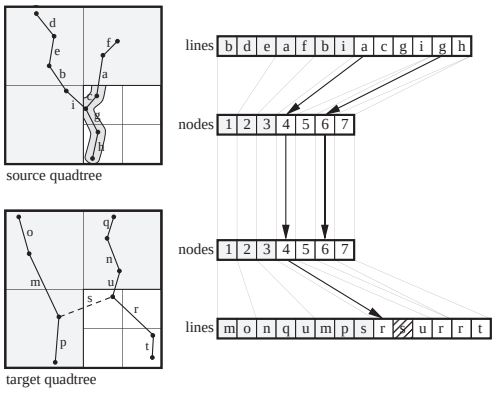

Figure 12: Active source and target nodes of minimal size (with other larger inactive nodes and lines shaded) after the first round of line communication and node merging. Note that the dashed target line s indicates that it was marked as intersecting the expansion region during the previous iteration.

1.0

Once all currently active source and target nodes have been merged, we then continue the above process, making all nodes of twice the size as the currently active nodes active (i.e., we are climbing one level of the quadtree as we move from the deepest node toward the root node). Once the level of the root of the quadtrees has been processed, the spatial range query operation is complete, with all lines in the target quadtree that intersect the source expansion set thus marked. Notice that during the algorithm a line in the target quadtree could be marked as intersecting the query region several times. However, the reporting of the intersection only happens once at the conclusion of the algorithm. This

is important as it avoids the need to eliminate duplicate answers [1]. Figure 13 depicts the situation immediately prior to the final round of source to target line communications.

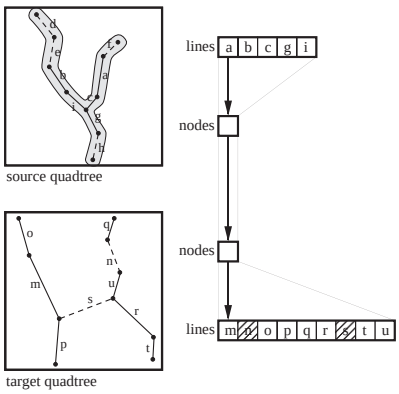

Figure 13: Active source and target nodes immediately prior to the final round of communication. Note that target lines n and s have been previously marked as intersecting the expansion region. Additionally, source lines d, e, f, and h were deleted during a prior source node merge phase as their expansion regions did not lay outside of the blocks corresponding to their source nodes.

1.0

R-trees

The R-tree and its variants are designed to organize a collection of arbitrary spatial objects (most notably two-dimensional rectangles) by representing them as d-dimensional rectangles. Each node in the tree corresponds to the smallest d-dimensional rectangle that encloses its son nodes. Leaf nodes contain pointers to the actual objects in the database, instead of sons. The objects are represented by the smallest aligned rectangle containing them. In our case, the object are line segments.

The definition of an R-tree is similar to that of a B-tree. All leaf nodes appear at the same level. Each entry in a leaf node is a 2-tuple of the form (R, O) such that R is the smallest rectangle that encloses the line segment O (where O points to the actual line segment). Each entry in a directory (non-leaf) node is a 2-tuple of the form (R, P), where R is the minimal rectangle enclosing the rectangles in the child node pointed at by P. An R-tree of order (m, M) means that each node in the tree, with the exception of the root, contains between $m \leq \lceil M/2 \rceil$ and M entries. The root node has at least 2 entries unless it itself is a leaf node.

1.0

The algorithm for building an R-tree is analogous to that employed when building a B-tree. Line segments are inserted into leaf nodes. The appropriate leaf node

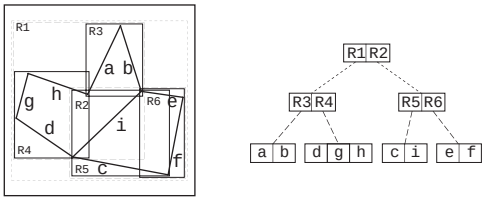

Figure 14: Example order $(1, 3)$ R-tree.

is determined in a top-down fashion by traversing the R-tree starting at its root and at each step choosing the subtree whose corresponding bounding rectangle will have to be enlarged the least by the addition of the line segment. Once a leaf node is determined, a check is made to see if the insertion of the line segment will cause the leaf node to overflow. If the leaf node is overflowing, it is then split and the $M + 1$ 2-tuples are redistributed among the two resulting nodes. Splits may propagate up the R-tree.

Note that the final shape of the R-tree will be dependent upon the insertion order of the line segments. In the data parallel environment [14], all line segments are inserted at the same time, and thus the final shape of the parallel R-tree will likely not be the same as its sequential counterpart.

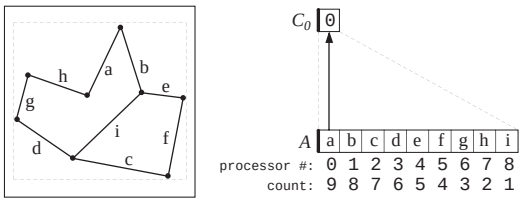

Figure 15: Initial R-tree processor assignments.

1.0

Data-Parallel R-trees

The parallel R-tree building algorithm proceeds as follows (for more details, see [15]). Initially, one processor is assigned to each line of the data set, and one processor to the resultant R-tree as depicted for a sample dataset in Figure 15. Our example assumes an order $(1, 3)$ R-tree. In the figure, the label A denotes the line processor set, C_0 denotes the R-tree node processor set, with the associated square region containing the identifier for the group of line processors associated with the R-tree node processor (i.e., the group of line processors beginning at processor 0 in A). We use the term *segment* to refer to the collection of line processors associated with a particular R-tree node processor. Within the line processor set A, the nine square regions contain the line identifiers and indicate which processor is associated with which line. A downward scan operation (i.e., a scan which returns the vector $[(a_0 \bigoplus a_1 \bigoplus \cdots \bigoplus a_{n-1}), (a_1 \bigoplus a_2 \bigoplus \cdots \bigoplus a_{n-1}), \cdots, a_{n-1}]$) is performed on the line processor set to determine the

number of lines associated with the single R-tree processor. This is shown in Figure 15 as the `count` field beneath the line processor set. The number of lines in the segment is then passed to the single R-tree node processor. If the number of lines in the segment exceeds M, then the R-tree root node must be split into two leaf nodes and a root node (as is similarly done with the sequential R-tree). The two new leaf nodes are inserted into the R-tree processor set, with the former root node/processor updated to reflect the two new children.

The topic of how to split an overflowing node has been the subject of much research on sequential R-trees (e.g., the R*-tree [2] is an R-tree variant that uses a more sophisticated node insertion and splitting algorithm than the conventional definition of the R-tree [13]). For the parallel R-tree, the node splitting algorithm first sorts all lines in the segment according to the left edge of their bounding boxes. A parallel upward scan operation (i.e., a scan which returns the vector $[a_0, (a_0 \bigoplus a_1), \cdots, (a_0 \bigoplus a_1 \bigoplus \cdots \bigoplus a_{n-1})]$), is used to determine the extents of the bounding box formed by all lines preceeding a line in the sorted segment. A downward scan will similarly determine the bounding box for all following lines in the segment. For all legal splits (i.e., where each of the two resulting nodes receives at least m/M of the lines being redistributed), the amount of bounding box overlap is calculated, with the split corresponding to the minimal amount of overlap being selected as the x-axis candidate. An analogous procedure is employed for the y-axis in obtaining the y-axis candidate split coordinate value. Once the two candidate split coordinate values are determined, the one corresponding to the minimal bounding box overlap is selected. In the event of a tie, some other metric such as choosing the split with the minimal bounding box perimeter lengths may be employed. This splitting algorithm is of complexity $O(\log n)$ at each stage of the building operation as we employ two $O(\log n)$ sorts and a constant number of scan operations.

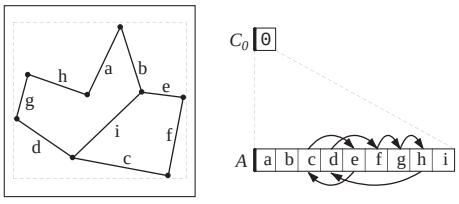

Figure 16: Un-shuffle operation.

1.0

Once the splitting axis and the coordinate value are chosen, an *un-shuffle* operation [3] (where two intermixed types are rearranged into two disjoint groups via two monotonic mappings) is used to concentrate those line processors together into two new segments, each of which will correspond to one of the two R-tree leaf node processors as depicted in Figure 16. For example, all lines which have a midpoint that is less than the split coordinate value are monotonically shifted toward the left, while those which are greater than the split coordinate value are monotonically shifted toward the right among the line processors. The result of the un-shuffle operation on the lines in Figure 16 is shown in Figure 17. Note that the root node of the R-tree is associated with two segments in the line processor set A (i.e., (a, b, e, h) and (c, d, f, g, i)), and must itself be subdivided in an analogous manner.

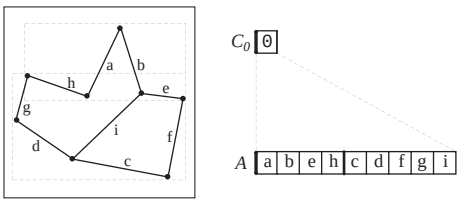

Figure 17: Result of the un-shuffle operation.

1.0

Thus, at this stage after the first root node split and line redistribution, we will wind up with two segments in the line processor set A, and two different R-tree processor sets C_0 and C_1 (each set corresponding to a node at a different height in the R-tree), as shown in Figure 18.

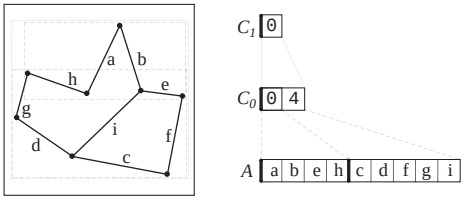

Figure 18: Completion of root node split operation.

1.0

The insertion algorithm will now proceed iteratively as before, with each segment determining the number of lines it contains, and transmitting the count to the associated R-tree processor. If the number of lines in the segment exceeds M, then the segment (and corresponding R-tree node processor) will be forced to subdivide (i.e., split). Note that this subdivision process may result in processors that correspond to internal nodes in the R-tree splitting themselves (with these splits possibly propagating up the R-tree). The building process will terminate when all nodes in the R-tree processor set have at most M child processors (either internal R-tree nodes or line processors), shown in Figure 19 for our example dataset. The R-tree root node corresponds to the single processor in set C_2, the leaf nodes are contained in processor set C_0, and all lines area grouped in segments of length less than or equal to 3 in processor set A (recall that we are dealing with an order $(1, 3)$ R-tree in our example).

1.0

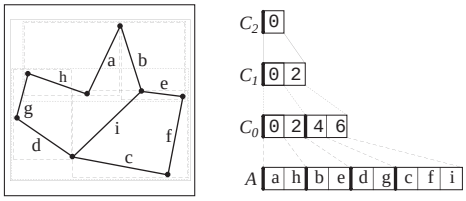

Figure 19: Completion of the data-parallel R-tree building operation.

The data-parallel R-tree building operation is of complexity $O(\log^2 n)$, where each of the $O(\log n)$ stages requires $O(\log n)$ computations (a constant number of scans along with two bounding box sorts).

R-tree Map Intersection

Given data-parallel source and target R-trees, a correspondence between the source and target R-tree nodes must be established which will be used to determine patterns of parallel communication. Basically, for each source R-tree leaf node,b s we must determine the intersecting target leaf nodes. Any source line in s might intersect another target line contained in a target leaf node that intersects s. For example, in Figure 20, source line segment c (contained in source node B), might intersect target line s (contained in target node L) as nodes B and L intersect. If a source and target leaf node do not intersect, then it is not possible for the associated contained lines to intersect. In Figure 20, source line c cannot intersect target line r (contained in target node N) as nodes B and N do not intersect.

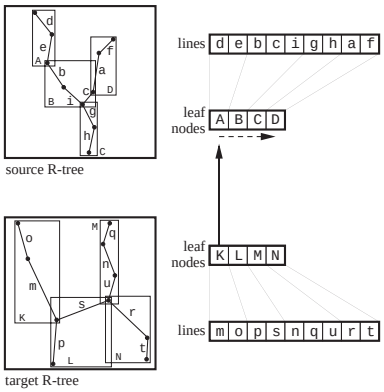

Figure 20: Example data-parallel R-trees for the same set of source and target lines as in Figure 1. Leaf node bounding rectangles are shown for both source (A, B, C, and D) and target (K, L, M, and N) R-trees. Internal R-tree nodes and bounding boxes are omitted for clarity.

1.0

For each source node, the process of determining which target leaf nodes it intersects is quite straightforward. In essence, each target node in turn will transmit its bounding rectangle coordinates to the first source leaf node in the arbitrary linear ordering (i.e., source leaf node A in Figure 20). These coordinates will then be shared among all source leaf nodes via a series of scan operations. Once each source leaf node knows the coordinates of the communicated target leaf node, in parallel, each source leaf node then determines whether or not it intersects the target node. If there is an intersection, then the index of the target node is appended to the source leaf node's list of target node intersections. This situation is depicted in Figure 20, where the communication path between first target leaf node (node K) and the first source leaf node (node A) is shown. The dashed arrow beneath the source leaf nodes represents the scan operation that is employed to share the target node bounding rectangle coordinates among all of the source leaf nodes.

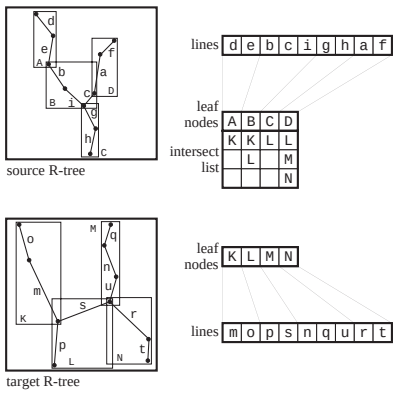

Figure 21: Source and target R-trees after all source and target leaf node intersections have been determined and recorded in the appropriate source nodes.

1.0

Figure 21 represents the situation found after all of the target leaf nodes have communicated their bounding rectangle coordinates to the source leaf nodes, and each source leaf node has compiled its intersection list. The intersection lists for each source leaf node are depicted as the collection of boxes beneath each source node identifier (e.g., source node B's intersection list contains target node identifiers K and L).

Once all source/target node intersections have been determined, the source leaf nodes will then transmit the endpoint coordinates of all contained line segments to all intersecting target leaf node line segments. Each target line then determines whether or not it intersects any source line segment.

Unlike the parallel PMR quadtree source to target node communication process, the non-disjoint irregular partitioning of space induced by the R-tree decomposition creates additional communication difficulties. In-

stead of all active source nodes communicating in parallel with the associated target node in the one-to-one mapping available in the PMR quadtree, the R-tree source to target leaf node communications are scheduled and made in an iterative process. This is due to the situation when multiple source nodes intersect a single target node (i.e., in Figure 21, source nodes B, C, and D each intersect target node L).

The scheduling problem is analogous to the Chromatic Index problem [12] where the set of source and target leaf nodes may be thought of as a set of vertices, and the intersections between the nodes as edges between the vertices. These edges and vertices form a bipartite graph, and it has been shown that there exists a polynomial time algorithm for scheduling the necessary communications [11].

In solving the communication scheduling problem, a non-optimal solution was chosen using a greedy approach. At each iteration of source to target communications, the first source node requiring communication selects the first target node in its intersection list (e.g., in Figure 22, source node A selects target node K). The next source node requiring communication then selects the first target node that has not been previously selected by the first source node (e.g., in Figure 22, source node B selects target node L). Continuing in this fashion, all following source nodes requiring communication select the first target node in their intersection lists that has not been previously selected by another source node during the current communication iteration.

Once all possible communications have been determined for the current iteration, each source leaf node that has a scheduled communication is made active (in Figure 22, this corresponds to source nodes A, B, and D). Using a technique similar to that employed in the PMR quadtree intersection algorithm, each active source node broadcasts the endpoints of all associated line segments (i.e., all the line segments that are found in the R-tree node) to the set of line segments in the associated target R-tree node. This is accomplished by the first line processor associated with each active source node passing its endpoints to the first line processor in the corresponding target R-tree node. In Figure 22, the first line processors (i.e., source lines d, b, and a) are shown with arrows emanating from them directed at the corresponding source nodes. Once the source line endpoint coordinates have been communicated to the first target line processor in the associated target R-tree node, they are then shared among all target line processors in the same target node via a sequence of scan operations.

1.0

Each active target line processor then simultaneously determines whether or not it intersects the broadcasted source R-tree line segment. If the target line intersects the broadcasted source line, the target line is so marked. Continuing this process, the collection of second line processors associated with each active source R-tree node (i.e., in Figure 22, source lines e, c, and f) passes it's coordinates to the first processor in each active associated target node. Again, the line coordinates are then communicated among all line processors in the target node via a sequence of scan operations, and each target line processor in parallel determines whether or not it intersects the source line. Once all active source line processors have transmitted their coordinates to the associated target line processors, the current iteration of communications is complete, and all active source nodes delete the target node that was the recipient of their communications from their intersection lists.

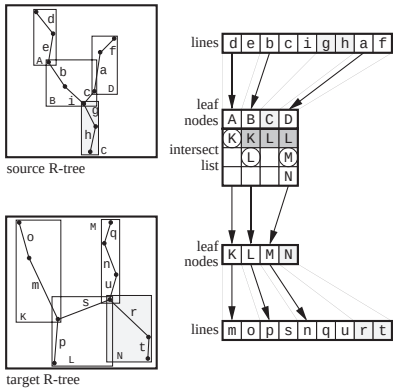

Figure 22: Source and target R-trees with first round of communication scheduling highlighted (inactive lines and nodes are shaded).

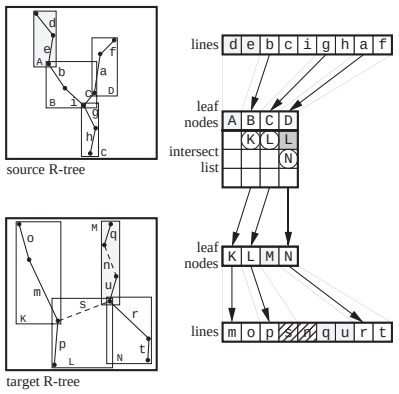

Figure 23: Source and target R-trees with second round of communication scheduling highlighted. Note that source node intersection lists have been modified following the first round communication depicted in Figure 10. Additionally, target lines marked as intersecting the expansion region are shown with covering diagonals (i.e., s and n).

1.0

Continuing with this process, the second round of source node to target node communications must be scheduled. Following an identical selection process as before, the first source node requiring a communication

makes its selection. Each following node (in our arbitrary linear ordering of source leaf nodes) requiring a communication first determines whether or not any of it's remaining intersecting target nodes has not been selected by a preceeding source node. If this is the case, the selection is made, and the following source nodes requiring communication make their selections (provided an unselected target node in their intersection lists is available). In Figure 23, the selected target nodes are shown in the intersection lists enclosed in circles.

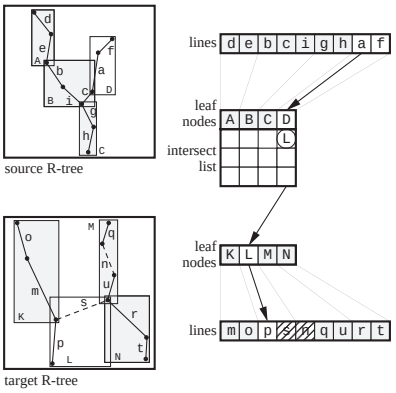

Figure 24: Source and target R-trees with final round of communication scheduling highlighted. Note that only source node D needs to communicate with a target node.

1.0

Once all source nodes have communicated their enclosing source lines with all of the target nodes in their intersection lists, and all intersecting target lines have been marked if they intersect a broadcasted source line, the intersection operation is complete. For our example data set, the third and final round of communications is shown in Figure 24, with only source node D broadcasting source lines a and f to the target line p contained in target node L.

R-tree Spatial Range Query

The algorithm for performing the spatial range query for data-parallel R-trees is very similar to that employed when computing the intersection as described previously. There are two small modifications that are necessary in adapting the R-tree intersection algorithm to a spatial range query algorithm.

1.0

The first modification involves the process of determining the target nodes with which each source node must communicate. In the intersection algorithm, this required finding all source/target node intersections. If the bounding rectangles for each source node are extended by the radius of expansion in each dimension prior to calculating all node intersections, one may ensure that no necessary source line to target line communications are overlooked.

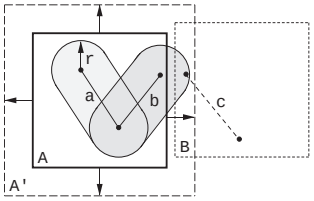

Figure 25: Example where the enlargement of the source R-tree bounding rectangle A by the expansion radius r (creating bounding rectangle A') prior to node intersection detection is necessary. The dashed bounding rectangle B represents a target R-tree node that contains target line c.

The second and final modification to the R-tree intersection algorithm concerns the source line to target line intersection calculation. Rather than determining whether or not two lines intersect, one would calculate the distance between the source and target lines. If the distance is less that the radius of expansion, the target line will clearly intersect the expansion region of the associated source line and should thus be marked. Once these two small modifications are made to the R-tree intersection algorithm, it will also function as a spatial range query algorithm.

Performance Comparison

In order to compare the behavior of the two data-parallel spatial decompositions and their performance on two map spatial join algorithms, each of the four described algorithms was implemented in C* on a Thinking Machines CM-5 (32 processor model). The data that we used consisted of maps containing planar line segment data from Bureau of the Census TIGER/Line files [7]. Our experiments used the map of Prince Georges County in Maryland as an example dataset.

In each of the spatial joins, the set of lines corresponding to railroads was chosen as the source dataset; with the set of lines corresponding to the road network in the county as the target dataset. In this case, the spatial join queries were trying to determine which roads were within a specified distance of a railroad line. The radius of expansion varied between 0 (corresponding to a map intersection query) and 50 units. Additionally, the splitting thresholds for the PMR quadtree varied between 8 and 32, while R-tree node capacities ranged from 10 to 50.

1.0

In Figure 26, the cpu times (in seconds) for the PMR quadtree variants of the spatial join operations are presented, with the vertical axis corresponding to the total cpu time, and the two other axes to the radius of expansion and to the splitting threshold. Observing the surface, we note that for the two map intersection query, a splitting threshold of roughly 14 to 16 corresponds to the smallest execution times. As the radius of expansion

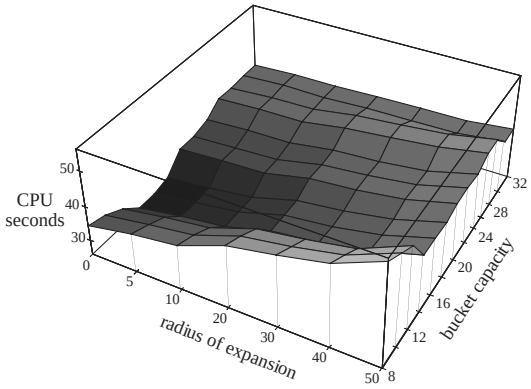

Figure 26: Execution time in seconds for the PMR quadtree spatial join algorithms for the example dataset (Prince Georges County, Maryland TIGER/Line file).

increases toward 50, these splitting thresholds continue to exhibit good performance although the performance advantage is not as great.

There are two basic forces working against each other as the radius of expansion and splitting thresholds increase. First, with a larger radius, fewer source lines are deleted as the source nodes are merged (recall the situation described in Figure 11), thus resulting in more source line to target line endpoint transmissions. Second, as the splitting threshold increases for a fixed radius of expansion, we find fewer nodes but of larger capacity. The lessened node count results in a quadtree of shallower depth (which will result in fewer iterations of the spatial join algorithms), but each iteration will take longer as more source line segments need to transmit their endpoint coordinates to the target lines.

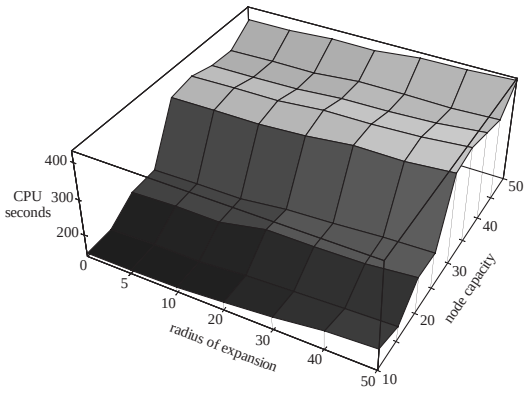

Figure 27: Execution time in seconds for the R-tree spatial join algorithms for the example dataset.

1.0

In Figure 27, the cpu times (in seconds) for the R-tree variants of the spatial join operations is presented, again with the vertical axis corresponding to the total cpu time, and the two other axes to the radius of expansion and to the node capacity (analogous to the splitting threshold in PMR quadtrees). The performance surface clearly indicates that R-trees with smaller node capacities (i.e., 10 or 15) exhibit execution times that are far less than for larger node capacities (i.e., 45 or 50). The reason for this large difference in performance is that smaller node capacities result in a finer decomposition of space; each of the smaller source nodes will intersect a smaller number of target nodes. With this finer granularity, there is increased opportunity of parallel communication when broadcasting the source lines to the appropriate target nodes.

Not surprisingly, execution times for a fixed node capacity tend to increase as the radius of expansion increases. Similar to what was observed with the data-parallel PMR quadtree, the increased radius results in increased numbers of source/target node intersections (as the source node bounding rectangles are expanded as is shown in Figure 25) and consequently an increased execution time.

When comparing the execution times of the PMR quadtree and R-tree spatial join implementations, it is apparent that the PMR quadtree offers significant performance advantages. This is primarily because the PMR quadtree yields a regular disjoint decomposition of space which facilitates increased amounts of parallel communication between source and target maps in comparison to the R-tree. The R-tree's non-disjoint, irregular decomposition of space suffers in the data-parallel environment.

One must keep in mind that these execution times are for the two map spatial joins. If one was to implement single map versions of the queries (i.e., given a single map containing line segments representing both roads and railways), the performance of the parallel R-tree would increase considerably, perhaps to a level comparable to that displayed by the parallel PMR quadtree. The single map spatial join algorithms is a topic for future research.

Concluding Remarks

Data parallel algorithms for computing a spatial join for the PMR quadtree and R-tree spatial data structures have been presented. The algorithms have been tested and revealed the superiority of the PMR quadtree. The main reason for this behavior is the fact that the PMR quadtree yields a disjoint decomposition of space while this is not the case for the R-tree. The problem is compounded when we wish to determine overlapping areas in the two maps that serve as operands to the spatial join. The algorithms that we studied are based on a bottom-up approach. It could be that this problem is avoided by use of a top-down approach. This is a subject for future work.

References

[1] W. G. Aref and H. Samet, Uniquely reporting spatial objects: yet another operation for comparing spatial data structures. *Proceedings of the Fifth International Symposium on Spatial Data Handling*, Charleston, SC, August 1992, 178–189.

[2] N. Beckmann, H. P. Kriegel, R. Schneider, and B. Seeger, The R*-tree: an efficient and robust access method for points and rectangles, *Proceedings of the SIGMOD Conference*, Atlantic City, NJ, June 1990, 322–331.

[3] T. Bestul, Parallel paradigms and practices for spatial data, Ph.D. dissertation, CS-TR-2897, Center for Automation Research, Computer Science Department, University of Maryland, College Park, MD, 1992.

[4] G. E. Blelloch and J. J. Little, Parallel solutions to geometric problems on the scan model of computation, *Proceedings of the International Conference on Parallel Processing*, St. Charles, IL, 1988, 218–222.

[5] G. E. Blelloch, Scans as primitive parallel operations, *IEEE Transactions on Computers*, C-38, 1989, 1526–1538.

[6] T. Brinkhoff, H. P. Kriegel,and B. Seeger, Efficient processing of spatial joins using R-trees, *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data*, Washington, DC, May, 1993, 237–246.

[7] Bureau of the Census, TIGER/Line Census Files, 1990 Technical Documentation, Washington, 1991.

[8] D. Comer, The ubiquitous B-tree, *ACM Computing Surveys 11*, 2 (June 1979), 121–137.

[9] R. Elmasri and S. B. Navanthe, *Fundamentals of Database Systems*, Benjamin/Cummings, Redwood City, CA, 1989.

[10] C. Faloutsos, T. Sellis, and N. Roussopoulos, Analysis of object oriented spatial access methods, *Proceedings of the SIGMOD Conference*, San Francisco, May 1987, 426–439.

[11] H. N. Gabow, Using Euler partitions to edge color bippartite multigraphs", *International Journal of Computational Information Sciences*, 5, 345–355.

[12] M. R. Garey and D. S. Johnson, *Computers and Intractibility: A Guide to the Theory of NP-Completeness*, Freeman, NY, 1979.

[13] A. Guttman, R-trees: a dynamic index structure for spatial searching, *Proceedings of the SIGMOD Conference*, Boston, June 1984, 47–57.

[14] W. D. Hillis and G. L. Steele, Data parallel algorithms, *Communications of the ACM*, 29, 12 (December 1986), 1170–1183.

[15] E. G. Hoel and H. Samet, Data-parallel R-tree algorithms, *Proceedings of the International Conference on Parallel Processing*, St. Charles, IL, August 1993, 49–53.

[16] F. T. Leighton, *Introduction to Parallel Algorithms and Architectures*, Morgan Kaufmann, San Mateo, CA, 1992.

[17] R. C. Nelson and H. Samet, A consistent hierarchical representation for vector data, *Computer Graphics 20*, 4 (August 1986), 197–206 (also *Proceedings of the SIGGRAPH'86 Conference*, Dallas, August 1986).

[18] R. C. Nelson and H. Samet, A population analysis for hierarchical data structures, *Proceedings of the SIGMOD Conference*, San Francisco, May 1987, 270–277.

[19] A. Rosenfeld and A. C. Kak, *Digital Picture Processing*, Academic Press, NY, 1982.

[20] H. Samet, *The Design and Analysis of Spatial Data Structures*, Addison-Wesley, Reading, MA, 1990.

[21] H. Samet, *Applications of Spatial Data Structures: Computer Graphics, Image Processing, and GIS*, Addison-Wesley, Reading, MA, 1990.

[22] C. D. Tomlin, *Geographic Information Systems and Cartographic Modelling*, Prentice Hall, Englewood Cliffs, NJ, 1990.