Exploring Different Methods for 2DR-tree Binary Search on a FPGA

J. E. Rice
Dept. of Math & Computer Science
University of Lethbridge
Lethbridge, AB, Canada
Email: j.rice@uleth.ca

J. Schultz
Dept. of Math & Computer Science
University of Lethbridge
Lethbridge, AB, Canada
Email: jeremy.schultz@uleth.ca

W. Osborn
Dept. of Math & Computer Science
University of Lethbridge
Lethbridge, AB, Canada
Email: wendy.osborn@uleth.ca

***Abstract*— Many data structures are proposed for managing spatial data. However, they are limited in their software implementations. This paper analyses two hardware implementations of an existing spatial data structure using a FPGA.**

I. Introduction

Current technologies are allowing individuals, companies, and institutions to store data in amounts that 10 years ago would have boggled the mind. We have progressed from 5.25 inch floppy disks capable of storing less than 100 kilobytes of data, to hard disk drives that can store hundreds of gigabytes of data. Corporations and industry are making use of such technologies to store terabytes of data.

In this work we focus in geographical information system (GIS), an area that is coming into its own due to increased storage capabilities. A GIS manages spatial data, which, unlike traditional types of data, is comprised of multidimensional information. For example, one can use a longitude and latitude specification to specify a place on a globe. If one is referring to an archaeological digsite, then the depth of the object may also be specified.

Most current storage systems are not designed to efficiently store this type of data and so searches and retrieval of spatial information tends to be slow. Recent work by Osborn [1] has produced a data structure called the 2DR-tree, intended specifically for spatial data, with the intent of solving this problem. However, due to its complexity the software implementation does not perform to the needs that industrial applications require. In this work we address this limitation by investigating the feasibility of implementing the 2DR-tree on a reconfigurable chip known as a field-programmable gate array (FPGA).

II. Background

A. FPGAs

Hardware solutions, designed specifically for a particular problem, perform much faster than software solutions. However, it is extremely expensive to fabricate application-specific chips. Once fabricated the chip cannot be changed for use with any other type of problem. In the last 10 to 15 years, with the advent of (re)configurable hardware, it has become more common to identify within an algorithm some highly-intensive computations and off-load this portion of the processing to a chip that could be programmed for the specific problem. FPGAs allow a chip to be reused for a variety of problems, including image compression [2], string matching [3] and bioinformatics applications such as [4], [5], [6]. The field of reconfigurable computing allows us to utilize the flexibility and processing power of reconfigurable devices such as FPGAs to achieve an increase in performance.

A FPGA consists of programmable cells. Each cell can be programmed for either I/O or computational functionality. Cells can have various structures, but are often comprised of look-up tables (LUTs), which can be programmed to perform various functions and interconnected in as many ways as can be determined by the place and route software.

Reconfigurable computing is generally static or dynamic. In static reconfigurable computing, the device is programmed once for the entire instance of an application. Dynamic reconfigurable computing solutions re-program the device many times, producing multiple hardware designs during execution. We are primarily interested in a static solution as we do not wish to incur the overhead of reprogramming the FPGA on the fly. However, a dynamic solution is not ruled out at this stage in our investigations.

B. Spatial Data Representation and Retrieval

A spatial database contains data that is represented in multidimensional space. Spatial data can range in complexity from simple points to objects which are themselves composed of sub-objects, such as points, lines, or arbitrarily-shaped objects. For example, a town is represented with a point, while a province has many towns (*i.e.* points), cities (*i.e.* regions), and roads (*i.e.* linestrings). Two important issues for spatial data sets are the efficient retrieval of a specific object and the efficient search for subsets of spatial objects. Some common search types include [1]

- exact match searching, in which a matching point, object, or object approximation is found,
- region searching, where all points or objects that overlap a query region are found.

Many one-dimensional hierarchical structures are proposed for retrieving spatial data [7]. Most store minimum bounding rectangles (MBRs) of objects and the regions in space that contain objects. Their limitations include overcoverage of empty space, and overlap of the MBRs, which leads to multiple-path searching. Because we propose the use of a multidimensional structure to store spatial data we avoid these problems.

III. The Project

A. The 2DR-tree

This paper builds on work introduced by the authors in [8], which uses a two-dimensional hierarchical data structure for storage of objects in two-dimensional space. Using a hierarchical data structure with the same dimensionality as the data set leads to significant improvement in retrieval performance [1]. The 2DR-tree supports the mapping of each MBR at every level to an appropriate location in a two-dimensional node, which reduces overlap and overcoverage. In addition, it reduces the number of nodes that need to be checked during searches, thereby improving retrieval performance.

The 2DR-tree differs from other spatial data structures in that the node locations are arranged in a node. Instead of a node being modeled as one-dimensional, it is in the form of a two-dimensional array. A node can be broken down into individual node locations. These locations hold MBRs, which consists of two coordinate pairs high (x, y) and low (x, y). At the leaf level, an MBR represents an object, and its location in two-dimensional space is approximated in an appropriate location within a node. At the non-leaf level, an MBR represents a subspace that contains other MBRs, and its location is also approximated within a two-dimensional node. The order of a node is $X * Y$, where X is the number of node locations on the x-axis and Y is the number of node locations on the y-axis. Figure 1 shows the layout for some sample objects, and the corresponding 2DR-tree representing those objects is shown in Figure 2.

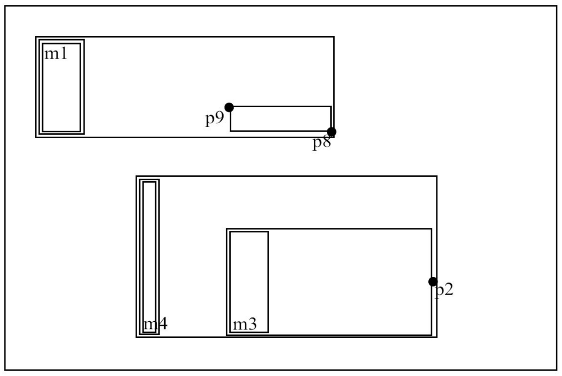

Fig. 1. An example layout of various objects.

Searching is done using a binary search algorithm. The main task generally involves deciding where to make the division; however as we will see in the following section the necessity for this is removed when implementing the search on the FPGA. In [8] we implemented a basic version of this concept. This paper builds upon this with a second implementation that solves many problems with our first method.

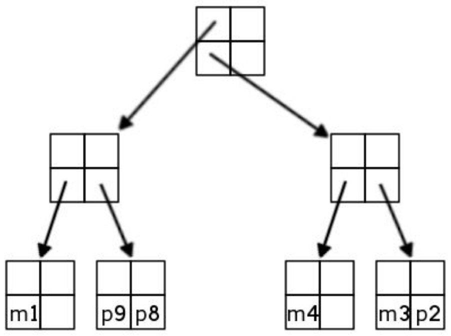

Fig. 2. The 2DR-tree storing the objects shown in Figure 1.

B. FPGA Implementation

For this work we used the Virtex-II Pro FPGA Prototyping Station, provided by the Canadian Microelectronics Corporation. It consists of an AMIRIX AP1000 development board that features the Xilinx Virtex-II Pro FPGA. It has 44,000 logic slices, two embedded IBM PowerPC hard macros and 1.4MB of on-chip RAM. We use the Xilinx ISE and Xilinx EDK software for development and simulations.

The FPGA implementation of the 2DR-tree structure closely matches the description in section III-A, except that the node location units also contain a pointer to related data about the object as well as logic to perform the bulk of the work (*i.e.* checking two MBRs for overlap). In software it is also necessary to determine a split point for the binary search. However, the FPGA can check all nodes simultaneously, and therefore we do not need to find a division; instead all nodes on one level can be tested at once. The node location unit is designed to test for overlap between a given input MBR and the stored MBR and give the appropriate output.

Along with the 2DR-tree, it is also necessary to implement controllers to manage the input and output. Details of these controllers are given in [8]. Using the node location unit as the foundation, a node of any order can be created. The only information stored at the node-level is the status of the node, *i.e.* leaf-node or not. The output from all node location units are passed out of the node unit as a group. The final step is to create, on the FPGA, a 2DR-tree built from the individual nodes. Two issues arise when creating the 2DR-tree. As the height of the tree grows the number of output MBRs increase drastically, as do the number of load MBRs. The input and output controllers solve the issue of handling the input and output loads, but tree size reamins a problem.

IV. Methods & Results

The goal of this work is to explore different FPGA implementations of the 2DR-tree binary search. In this section we present the two methods that were compared and some results for each. The first method is based on the idea of representing the complete 2DR-tree on the FPGA. However, in previous work we found that we quickly run out of room on the FPGA. The second method tries to overcome this shortcoming by using a queue of nodes to test for overlap.

A. Method 1: Complete Tree

In implementing the entire tree in hardware we gain an immediate speed increase from the ability to test several nodes for overlap at once. All nodes at each level in the 2DR-tree can be tested simultaneously. Assuming an order of $2*2$, then at the first level (the root node) one node is tested, followed by 4 at the next level, 16 at the level after that and so on.

A tree of height 3 was intially used for the FPGA 2DR-tree performance analysis. The height restriction is due to the space limitation of the FPGA. Adding another level would require a larger FPGA. Since storage of the MBRs for each node location accounts for most of the space requirements, we are able to fit a tree of height 4 on the FPGA by reducing the range of the integers in the MBRs. This was done by using an integer with 16 bits instead of 31 bits, since the sample data does not exceed the range of a 16 bit integer. Further reduction in the range would allow for more levels to be added to the tree, but for analysis purposes the two sample tree heights will suffice. Two techniques for generating timing results are used: first, the implementation was simulated using the Xilinx software, and then the method was implemented and tested on the FPGA. Simulations were run on the sample data with tree heights of both 3 and 4. A tree of height 3 has 21 nodes and 16 output nodes, while a tree of height 4 has 85 nodes and 21 output nodes. Both trees can be implemented with a clock cycle of approximately 12ns. Table I gives the results for performing searches with the varying height trees. For the FPGA implementation the total time is

Action	**FPGA Simulation**		**Software**	
	Height=3	**Height=4**	**Height=3**	**Height=4**
Initialize	6 ns	6 ns	N/A	N/A
Load	264 ns	1,032 ns	N/A	N/A
Search	48 ns	60 ns	N/A	N/A
Query Output	216 ns	792 ns	N/A	N/A
Total Time	534 ns	1,890 ns	13422 ns	20685 ns
Further Searches	264 ns	852 ns	no change	no change

TABLE I

SPEED FOR VARIOUS ACTIONS IN PERFORMING A BINARY SEARCH ON A 2DR-TREE (FPGA METHOD 1 COMPARED TO SOFTWARE).

based on loading each node in the tree, performing the search and querying all the output nodes. Further searches do not include loading the tree with nodes. The time it takes for the data to transfer to the host PC and FPGA is not included in these calculations. For comparison, the last two columns of Table I give the timing results for the software binary search. These results were obtained by runing the 2DR-tree Java code. The only modification to the code was to add instructions for gathering the time it takes for the searches. Since a search takes a fraction of a second, several hundred searches were performed and the average time was calculated from these results. There is a dramatic increase in speed using the FPGA, according to these simulated results. However, the results from execution of the binary search on the FPGA illustrate how much the data transfers slow down the search, as shown in Table II. Table II shows that incorporating the serial

Baud	**Iterations**	**Output Method**	**Avg. Time per Search (ms)**
9600	100	Full	46.8
9600	1000	Full	46.667
9600	100	None	17.267
9600	1000	None	17.334
57600	100	Full	15.667
57600	1000	Full	15.4
57600	100	None	2.81*
57600	1000	None	–

TABLE II

FPGA (METHOD 1) BINARY SEARCH PERFORMANCE INCLUDING SERIAL COMMUNICATIONS.

communication time requirements decreases the performance to a level far below the software results. Increasing the baud rate of the communications from 9600 to 57600 decreases the requirements, but not to an acceptable level. The output methods *full* and *none* refer to whether the search output is displayed. This allows us to determine the time required for querying for the output. With a baud rate of 57600 and with output method none, the Java serial driver gives errors and so therefore complete results could not be reported. In addition, the * indicates that not all iterations were completed for this data set.

From these results, running the binary search on the FPGA does not show any advantage. Increased communication speeds, such as sending information via PCI bus, are necessary. The difference between the software and FPGA binary search may also reduce as the height of the tree increases, since the FPGA search time does not dramatically increase as the height of the tree grows while this tends to be the trend for the software implementation.

B. Method 2: Overlap Queue

In the previous method there is a restriction on tree height, imposed by the data requirements and the size of the FPGA. One way around this restriction is to split the main tree into subsections and use the previous method to test each subsection. However, this could lead to loading of unneeded sections of the tree. This can be broken down even further. Instead of a complete subsection of the tree being tested for overlap a single node can be tested.

Using this method, the root node is added to a queue. The nodes in the queue are tested by a FPGA node. If a node location, from a non-leaf node, passes the overlap test, then the node it points to is added to the queue. Otherwise, if it is a leaf node and the node location passes the test, then that MBR is added to the found list. This cycle is continued until there are no nodes left in the queue. The pseudocode is:

```
Add the root node to a queue
While the queue is not empty
  Load and test the node on FPGA
  Query for the Node Location results
  If the Node Location passed the overlap test then
    If the node is a leaf-node then
      Add to the final results
    Else
```

```
      Add the child-node to the queue
    End if
  End if
```

Unlike method 1, which communcates with software strictly for input and output, this method relies on a software/hardware combination. The queue must be implemented in software and the overlap test would be performed on the FPGA. The queue software can, however, be run on the FPGA, since the board has a IBM PowerPC hard macro and onboard RAM.

The advantages to this method are that it behaves just like a binary search and takes up very little room on the FPGA. It can prune branches of the tree as it performs the search. It can also handle any size of tree. The only limitations are the size of the queue in software. Additionally, since this method only needs one node for testing, it frees up room for additional units that can then be used to speed up insertion or deletion operations on the 2DR-tree.

This method is not completely implemented on the FPGA. Further work on the software side of the method is required; however, some results can be calculated for the computations to be carried out in hardware. The clock speed used in the calculations is based on the previous methods, and thus is set at 12ns. The reason for using the previous method's clock cycles is because simulations of the overlap node do not give a clock cycle restriction. The single node has no complicated logic or mappings, and so should run at the FPGA's max speed. Comparing the estimated simulation timings of the two methods, the overlap queue method clearly is an improvement. This holds true even for the worst-case scenario where method 2 has to perform the overlap test on each node in the tree. It really shows its power if it can prune branches and does not test every node in the tree.

Action	**Height=3**	**Height=4**
Initialize	6 ns	6 ns
Single Test	12 ns	12 ns
Best-case (1 path)	36 ns	48 ns
Worst-case	252 ns	1020 ns
Total Time (Method: Complete Tree)	534 ns	1890 ns

TABLE III
ESTIMATED SIMULATION TIMINGS FOR METHOD 2 OF THE FPGA-BASED 2DR-TREE BINARY SEARCH.

As seen in the results from the last method, the communication latency increases the times for the FPGA methods considerably. Unfortunately, using serial communications there will be even more latency with method 2, since it depends on constant communication for each node. However one way to reduce the communication between the host PC and FPGA board is to load the tree in software on the FPGA. That means duplicating the tree to run on the IBM Power PC. The idea of creating the tree to run on the on-board processor of the FPGA board can be further extended. The insert and delete operations can be added to the software portion, and then certain node computations can be off-loaded to the FPGA. For instance, when inserting new data, it is necessary to determine where to add the data in the tree. The goal is to increase one node location's MBR to encompass the new data; however, the node location that requires the minimum amount of increase must be found via a greedy search. This operation can be incorporated into the FPGA and the speed increase will come from being able to run the test on all node locations concurrently.

V. Conclusion & Future Work

The problem is how to fit a highly data-intensive problem onto a platform that is intended for highly computationally-intensive problems. We propose two methods for using a FPGA to perform searches using a 2DR-tree structure.

Simulation of both FPGA-based methods for implementing and searching the 2DR-tree show the potential for speed-up. The major bottleneck in both methods is the communication latency between the host PC and the FPGA board. Currently, this communication is being performed via a serial cable. Future work will include using drivers for the FPGA board to communicate with the PC over the PCI bus. This would decrease the communication latency considerably.

The second method, using an overlap queue, shows even more promise. Once the communication barrier is overcome, not only should this technique speed up searches, but also we anticipate that it will allow for the inclusion of other required data manipulation operations such as inserts, update and deletes. Further development in implementing the 2DR-tree on the FPGA board using processor macros is of great interest, and could provide the solution to the problems we are encountering since it would reduce the communication between the FPGA board and the host PC to simple relays of the users' requests.

References

[1] W. Osborn, "The 2dr-tree: a 2-dimensional spatial access method," Ph.D. dissertation, University of Calgary, 2005.

[2] A. Simpson, J. Hunter, M. Wylie, Y. Hu, and D. Mann, "Demonstrating Real-Time JPEG Image Compression-Decompression Using Standard Component IP Cores on a Programmable Logic Based Platform for DSP and Image Processing," in *Proceedings of Field Programmable Logic (FPL) 2001, LNCS 2147, Springer-Verlag*, 2001, pp. 441–450.

[3] H. Lee and F. Ercal, "RMESH Algorithms For Parallel String Matching," in *Proceedings of the 3rd International Symposium on Parallel Architec tures, Algorithms and Networks (I-SPAN'97)*, 1997, pp. 223–226.

[4] K. B. Kent, J. E. Rice, S. V. Schaick, and P. A. Evans, "Hardware-Based Implementation of the Common Approximate Substring Algorithm," in *Proceedings of the Euromicro Symposium on Digital System Design: Architectures, Methods and Tools (DSD)*, 2005, pp. 314–320.

[5] J. E. Rice and K. B. Kent, "Systolic Array Techniques for Determining Common Approximate Substrings," in *Proceedings of the International Symposium on Circuits and Systems (ISCAS)*, 2006, cdrom paper 1480.pdf.

[6] K. B. Kent, R. B. Proudfoot, and Y. Zhao, "Optimizing the Edit-Distance Problem," in *Proceedings of the 17th International Workshop on Rapid System Prototyping (RSP)*, 2006, to appear.

[7] V. Gaede and O. Guenther, "Multidimensional Access Methods," *ACM Computing Surveys*, vol. 30, no. 2, pp. 170–231, 1998.

[8] J. E. Rice, W. Osborn, and J. Schultz, "Implementation of a Spatial Data Structure on a FPGA," in *Proceedings of the Second International Joint Conferences on Computer, Information, and Systems Sciences, and Engineering (CISSE)*, 2006, to appear.