Implementation of a Spatial Data Structure on a FPGA

J. E. Rice, W. Osborn and J. Schultz

Department of Mathematics & Computer Science
University of Lethbridge
4401 University Drive, Lethbridge, AB, Canada,
T1K 3M4

***Abstract* - Many systems exist that store and manipulate data; however, many do no have sufficient support for spatial data Many data structures are proposed that are intended specifically for spatial data; however, software implementations have not performed as well as hoped. This work presents a feasibility study investigating the use of a FPGA for the implementation of a structure to support spatial search and retrieval.**

I. Introduction

Recent advances in the area of data storage have resulted in technology enabling institutions, companies, and individuals to store data in sizes never before envisioned. One area in particular that is leveraging this increase is the area of geographical information systems (GIS). A GIS manages spatial data. Unfortunately many data structures are not appropriate for spatial data. Recent work by Osborn [1] has produced a data structure intended specifically for spatial data. A limitation is that its software implementation is slow. In this work, we address this limitation by investigating the feasibility of implementing the data structure on a reconfigurable chip called a FPGA.

II. Background

A. *FPGAs*

One technique for accelerating computation is a (re)configurable hardware solution. This has been applied to various problems such as image compression [2] and string matching [3], as well as in other bioinformatics applications [5, 6, 7]. Reconfigurable computing utilizes the flexibility and processing power of reconfigurable devices such as Field Programmable Gate Arrays (FPGAs) to achieve an increase in performance. This flexibility allows the development and implementation of a custom hardware circuit as part of a solution. A FPGA consists of many programmable cells, which can be programmed for either I/O or functionality. Cells are comprised of look-up tables (LUTs), which can be programmed for various functions and interconnected in many ways that are determined by the place and route software.

Reconfigurable computing is generally used in a static or dynamic role. In static reconfigurable computing, the device is programmed once for the entire instance of an application. Dynamic reconfigurable computing programs the device many times, producing multiple hardware designs during execution. We are primarily interested in a static solution because reprogramming the FPGA dynamically incurs overhead. However, a dynamic solution is not ruled out at this stage in our investigations.

B. *Spatial Data Representation and Retrieval*

Spatial data is data that exists in multidimensional space. It ranges in complexity from simple points to objects 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 are the efficient retrieval of a specific object (*i.e*. exact match) and the efficient search for subsets of spatial objects (*i.e.* a region search).

Many one-dimensional hierarchical structures are proposed for retrieving spatial data [4]. 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.

III. The Project

C. *The 2DR-tree*

The 2DR-tree is used to create a two-dimensional hierarchical structure for retrieving objects in two-dimensional space, as opposed to forcing those objects to fit a one-dimensional structure. Using nodes with the same dimensionality as the object space can lead to significant improvement in retrieval performance [1]. The 2DR-tree maps

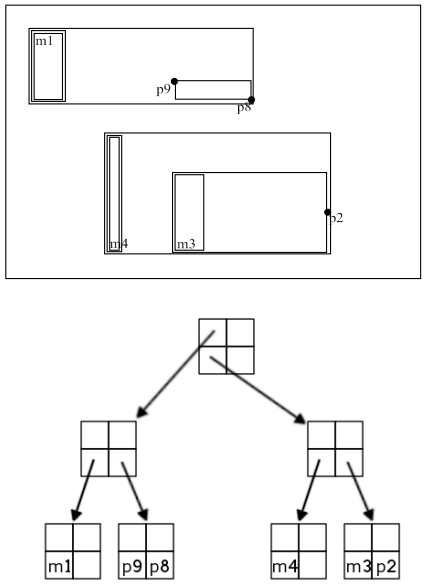

Figure 1. A sample object layout., with the resulting 2DR-tree.

each MBR to an appropriate location in a two-dimensional node, to reduce overlap and overcoverage.

Each node contains individual locations. Each node location holds the MBRs for an object or a region. A MBR consists of two coordinate pairs high (*x,y*) and low (*x,y*), or (*hx, lx, hy, ly*). At the leaf level, a MBR in two-dimensional space is placed in an appropriate location of a two-dimensional node. MBRs for regions are placed similarly in non-leaf nodes.

The 2DR-tree differs from other spatial data structures in how the node locations are arranged in a node. Instead of a node being modeled as a flat one-dimensional array, it is modeled as a two-dimensional array. Thus, 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.

Searching is done using a binary search strategy. The main task usually involves finding a split point; however we discuss in the following section the necessity for removing this when implementing the search on a FPGA.

D. *FPGA Implementation*

For this work we used the Virtex-II Pro FPGA Prototyping Station that is provided by the Canadian Microelectronics Corporation (CMC). It is an AMIRIX AP1000 development board that features the Xilinx Virtex-II Pro FPGA. It has 44,000 logic slices and features 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 to run simulations.

The basic unit in a 2DR-tree is the node location. With the node location a node of any order can be built. The nodes are then used as the building blocks for a 2DR-tree. It makes

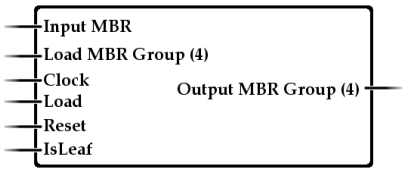

Figure 2. The input/output design for a 2*2 node.

sense for the unit representing the node location to contain most of the functional logic and carry out the bulk of the work (*i.e*. checking two MBRs for overlap). The other calculation performed in software is to find a dividing point for the node during 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 the input MBR for overlap with the stored MBR and give the appropriate output.

The output from a node location unit is a MBR. This may be empty if no overlap exists between the input and stored MBRs. If overlap exists, the output depends on whether the node location unit is marked as a leaf-node or not. An *isLeaf* marker is used to identify a leaf-node. A leaf-node will output the stored MBR, otherwise the input MBR is sent as output.

A load signal and a load MBR are used to set the stored MBR value for a node location unit. When the load signal is sent the load MBR value is used to replace the current value in the stored MBR. The initial value of the stored MBR is an empty MBR. The last two incoming signals to a node location unit are the clock and reset signals. The reset signal is used in conjunction with the load signal to re-initialize the stored MBR to the empty MBR, or on its own to signal the node to output an empty MBR.

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 size of this group depends on the node order. It the node has an order of 2*2, then the output group will have 4 MBRs. The load MBRs are also grouped in the same manner. This allows for an entire node to load all of its node locations at once.

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.

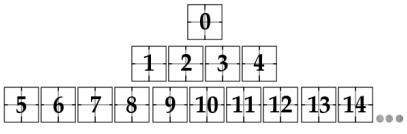

Figure-3. Numbering scheme for the 2DR-tree nodes

To solve these issues two controllers are needed, an input controller and an output controller. The input controller allows a single node to be loaded with a load MBR value, so the overall 2DR-tree on the FPGA only has one load MBR as an input line. Along with the load MBR value the node number is needed. Each node in the tree is numbered starting with the root node as 0 as shown in Figure 3. The reset and load signals for each individual node are also controlled by the input controller.

The output controller allows for a particular node output to be queried, which allows the FPGA 2DR-tree to have only one MBR output line. Each of the output MBRs for individual nodes are fed into the output controller as inputs. The load signal and node number are also provided as inputs to the output controller. If the load signal is set then the output MBR is an empty MBR, otherwise, the output from the node with the given node number is provided. These controllers solve the problem of having too many input and output lines for the FPGA 2DR-tree. It is now possible to create a functional tree to perform binary searches. Using this model a simulation using the Xilinx ISE was created for our testing and analysis purposes.

E. Results & Analysis

For the FPGA 2DR-tree performance analysis, we use trees of heights 3 and 4. These heights are chosen due to FPGA space limitations. We can store a height 4 tree by reducing the range of integers in the MBRs, as the sample data did not exceed the range of a 16 bit integer. Further reduction would allow for a deeper tree, but for analysis purposes the two sample tree heights will be adequate.

All of our results are from simulations carried out with the Xilinx design tools, and so the transfer time for sending data from the host system to the FPGA board is not taken into account. Table 1 shows the time required for each of the actions carried out by the FPGA-based 2DR-tree.

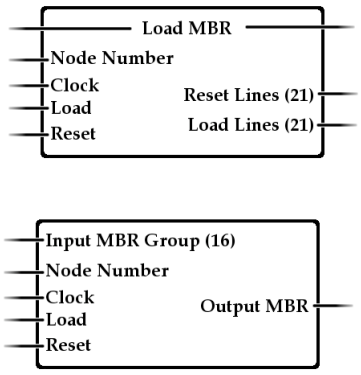

Figure-4. The input controller (top) and the output controller (bottom).

Table 2 shows the time required for the software binary search. The timing results for the software binary search are computed by using the Java code from Osborn's work [1]. The only addition was the code for timing the searches. Because searching takes a fraction of a second, several hundred searches were performed, and the average time was calculated.

Results show that the FPGA implementation will give a tremendous speed increase to the binary search. The concerning factor will be the time it takes for the data to transfer from the host machine to the FPGA board. Currently the transfer is performed via the serial interface, which can be rather slow for large amounts of data. We will investigate data transfer via the PCI Bus, which will produce faster data transfer speeds.

IV. CONCLUSIONS & FUTURE WORK

This work shows the feasibility of using a FPGA to improve search speeds on a spatial database. Work is continuing in many areas. We will incorporate communication times into our results. We will have the host machine run a Java application that will communicate with the FPGA board. This application will load the FPGA 2DR-tree, run a binary search and query for results. The communication between the application and the FPGA board can be done via the serial port or PCI bus, as investigations allow.

Also, additional logic can be added to support unbalanced trees and self-checking nodes. The current implementation requires that the 2DR-tree be balanced. Extending the logic to allow for leaf-nodes to occur any where in the tree would allow for more realistic sample data to be used. Self-checking nodes will allow for a more robust implementation and will be useful when logic is added for the insert operation. This means a node will have logic to ensure the MBRs within it conform to the required "node validity" presented by Osborn [1]. Finally, for a complete implementation of the 2DR-tree, insert and delete operations need to be incorporated into the design.

Table 1. FPGA 2DR-tree search performance.

Action	**Time (Height 3)**	**Time (Height 4)**
Initialize	6 ns	6 ns
Load	264 ns	1,032 ns
Search	48 ns	60 ns
Query Output	216 ns	792 ns
Total Time	534 ns	1,890 ns

Table 2. 2DR-tree software search performance.

Height	Iterations	Avg. Total Time
3	1,000,000	13,422.2 ns
3	5,000,000	13,100.04 ns
4	1,000,000	22,193.8 ns
4	5,000,000	20,684.96 ns

The current work on implementing a 2DR-tree on a FPGA shows much promise. The performance increase is significant. Further research will produce a functional 2DR-tree for binary searches. It can also lead to other speed increases for the 2DR-tree. The ideas presented in this paper can be applied and extended to increase performance for the binary search, insert and delete operations of the 2DR-tree.

REFERENCES

[1] Osborn, W. *The 2DR-tree: a 2-Dimensional Spatial Access Method*. PhD Thesis, University of Calgary, 2004.

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

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

[4] Gaede, V. and Guenther, O. *Multidimensional Access Methods*. ACM Computing Surveys, 30(2), pp. 170-231, 1998

[5] K. B. Kent, J. E. Rice, S. Van 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), pages 314–320, 2005.

[6] 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. Paper number 1480 (CDROM).

[7] 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.