Query Processing with An FPGA Coprocessor Board

Jack S.N. Jean, Guozhu Dong, Hwa Zhang, Xinzhong Guo, and Baifeng Zhang
Department of Computer Science and Engineering
Wright State University
Dayton, Ohio 45435, USA
{jjean, gdong, hwazhang, xguo, bzhang}@cs.wright.edu

Abstract

In this paper a commercial FPGA coprocessor board is used to accelerate the processing of queries on a relational database that contains texts and images. FPGA designs for text searching and image matching are described and their performances summarized. A potential design for a database JOIN operator is then studied. A query optimization preprocessor is then proposed.

Keywords: Field Programmable Gate Array (FPGA), Configurable Computing, Text and Image Database, Query Optimization

1. Introduction

Computers with FPGA coprocessor boards have been used to accelerate many different applications in the past, especially those that do not involve massive amount of floating point computations [1, 2, 3, 4]. Query processing on text and image databases provides ample opportunities for acceleration with such coprocessor boards. In this paper a commercial FPGA coprocessor board is used to accelerate the processing of queries on a relational database that contains texts and images. In [5], ASIC chips were developed for image and string matching processing and a hybrid and reconfigurable board architecture was proposed that combined those ASIC chips and FPGAs/EPLDs for multimedia applications. The difficulty of that approach is that the functionality of the ASIC chips needs to be versatile enough to cover the application domain.

In Section 2, FPGA designs for text searching and image matching are described and their performances using a Xilinx XC4085 on a commercial FPGA board are summarized. The designs are used to accelerate queries on an Oracle 8i database. A potential design for a database JOIN operator is then studied. In Section 3, a query optimization preprocessor is proposed. Section 4 concludes the paper.

2. Text Searching and Image Matching

In order to evaluate the proposed approach, a test platform was used which was a 600 MHz Pentium III personal computer with an Annapolis Micro Systems's WildForce^{TM} FPGA board. On that computer a small text/image database was set up and a GUI interface was developed in Microsoft Foundation Class to display image/text data records. Queries to access the database were implemented based on ODBC (Open Data Base Connectivity) API functions. Three different operations were studied, text searching, image matching, and JOIN operator. They are described in the following subsections.

2.1 Text Searching

The database contains records that include some photos and resumes. A keyword string can be input through the GUI and used to match against each text file. All the records whose employee resume files include the keyword were displayed.

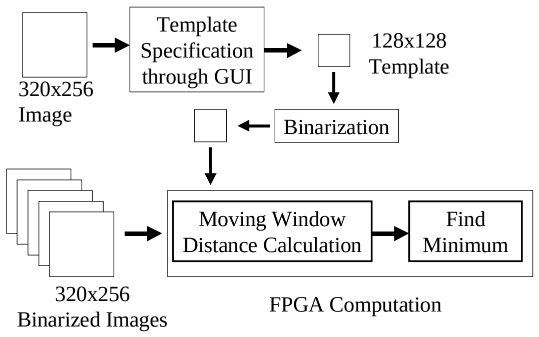
An efficient keyword-matching algorithm called the KMP algorithm (invented by Knuth, Morris, and Pratt) was implemented on the PC. The algorithm does not use more than M+N character comparisons to match a keyword of M characters against a string of N characters.

Two FPGA designs for the comparison of a keyword against a text file were implemented, one a straightforward hardware implementation of the KMP algorithm and the other a brute force usage of parallel comparators to implement the naïve text search algorithm (of O(M*N) complexity sequentially). It was found that while the KMP algorithm is more efficient on sequential machines, it enjoys no advantage in FPGA implementation. In fact, it suffers from a couple of drawbacks: (1) Its control circuit is more complicated and (2) it is more difficult for the FPGA to grab external data every clock cycle because whether it needs new data or not depends on the computation on the current data. As a result, without having a complicated data buffer, the KMP FPGA implementation can only grab one new data once every three clock cycles. By contrast, the brute force parallel comparator implementation simply grabs new character every clock cycle. In addition because the memory port can provide four consecutive characters in one clock cycle, four copies of the same hardware can be used to take advantage of that data parallelism with the brute force implementation. Because of these reasons the parallel comparator implementation is considered superior to the KMP one.

In order to test "case insensitive" keyword search, all the texts in each text file were converted to lower cases right before the keyword comparison. It was found that on the PC for a specific case with 22 Kbytes of text, it took 1.5 seconds for "case conversion" and 0.01 seconds for "keyword matching." An FPGA design that performs both the case conversion and the keyword matching was implemented. The design, utilizing 1,035 configurable logic blocks (CLBs), runs at 10 MHz and processes four characters per clock cycle. Since the FPGA design spends less than 0.01 seconds for both computations, the speedup is more than 150. Note that the majority of the computation is on the case conversion instead of the keyword matching.

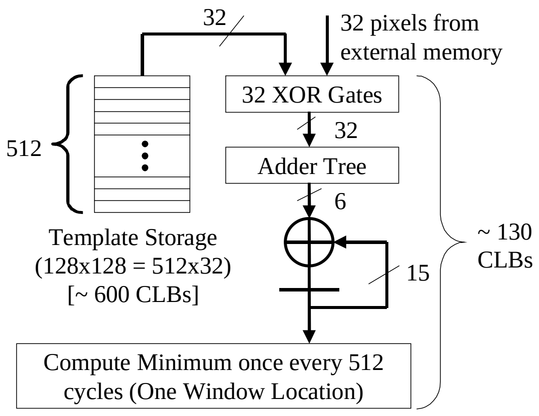
2.2 Image Matching

With the GUI, a user specifies a 128 x 128 area out of a 320 x 240 image and uses that area as a template to compare to all the images in the database. The record whose image *best matchs* with the template (within an "acceptable" degree of matching) is displayed. To simplify and speed up the matching process, the gray scaled images were preprocessed and their pixels converted into binary values. The binary images are stored in the database together with the gray scaled ones. The template is also binarized using a threshold chosen based on the histogram of the template. The resulting system is as shown in the following figure.



In terms of the matching computation, it is a moving window based distance calculation. When the template is applied to a pixel location of an image, the distance is computed as the number of mismatches between the template and the image window. Because the template and the image are both binarized, a template pixel and an image pixel are considered mismatched if an XOR operation produces a result of one. The computation therefore lends itself to a very efficient Pentium implementation because an integer XOR instruction handles 32 pixels in parallel. A 256-entry lookup table is used on the

Pentium code to count the number of ones in a 32-bit integer resulting from the XOR operation. With this efficient Pentium code, the search on the PC took around 8.48 seconds for nine images. The same search using a single computational unit on a XC4085 FPGA chip (with 3,136 CLBs) with a clock rate of 40 MHz on a WildForce board took around 3.91 seconds. Therefore the speedup is around 2.17. The FPGA design is as shown in the following diagram.


32
32 pixels from external memory
32 XOR Gates
32
Adder Tree
6
512
Template Storage (128x128 = 512x32) [~ 600 CLBs]
15
~ 130 CLBs
Compute Minimum once every 512 cycles (One Window Location)

By using multiple computation units, the potential improvements of a future design on the current WildForce Board are estimated as follows. On a single FPGA chip, around 4 computational units can be built and the speedup will be close to 5 or above. With four FPGA chips per board, the speedup is expected to be around 10 or above. Here the speedup is not assumed to be linear because other overheads may degrade the performance when higher degree of parallelism is explored. Another point worth mentioning is that Xilinx Virtex FPGA chips will be more suitable for this design than the 4085 chips being used because the template could then be stored more efficiently (both in space and in speed) in on-chip block RAMs.

2.3 JOIN Operator

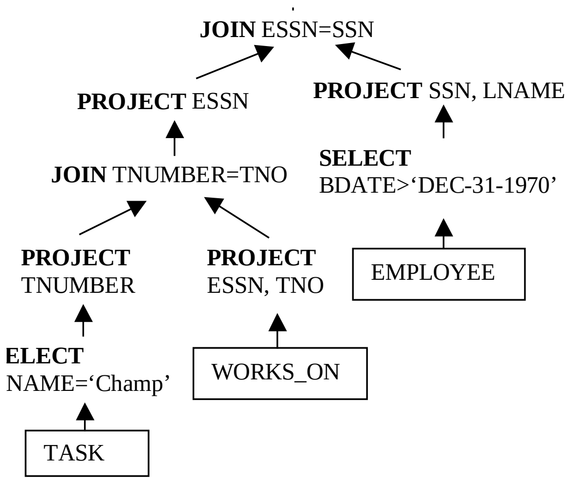
Typical database operations involve "JOIN" operations. A JOIN operation on two tables produces a table whose key values are in both tables. The operation is basically for each element in a sequence A, to find a corresponding matching element in another sequence B. In order to develop an efficient FPGA design for the JOIN operation, the following sequential algorithm was developed.

Suppose there are two sequences of integers, A and B. There are N elements in A, and M elements in B. Assume $N \ll M$. The algorithm to join the two sequences includes two steps: (1) Sort the sequence A : complexity $O(N \log N)$ and (2) For each B element, find the matching element in A by using a binary search : complexity $O(M \log N)$. Therefore the total complexity is $O(M \log N)$ that comes from the M binary searches. The algorithm was implemented and tested on an Oracle database. While a pure Oracle JOIN operation including data fetching took 27 seconds on a particular test case, the proposed sequential JOIN operation with data fetched through the same Oracle 8i database took only 22 seconds. This shows that it is possible to build our own JOIN operation that is as efficient as or even better than a pure Oracle JOIN operation (at least for the limited tests we ran).

An FPGA design to accelerate the JOIN operation, or more specially to accelerate the binary search has been proposed. The design uses a pipeline of comparators where each comparison result determines the operand to be sent to the next comparator stage. With a 40 MHz design, it is estimated that roughly 40 million integers can be processed each second (assuming each integer is compared to 1,024 sorted values). Since a program running on a 600 MHz Pentium III PC spends around 30 seconds on similar operations, the FPGA design can potentially lead to an order of magnitude improvement for the JOIN operation.

3. Query Optimizing Preprocessing

Once a database is created and in use, the main performance concern is about the processing, optimization, and execution of queries. A query can typically be represented as a "query tree," where each leaf node represents a relation (i.e., table) and each internal node represents a database operation. For example, the query tree as shown in the following figure corresponds to an optimized representation of the query **Q** (Find the last names of employees born after 1970 who work on a task named "Champ").



With optimization the initial query tree would use the Cartesian product of three tables, EMPLOYEE, WORKS_ON, and TASK, which may lead to a huge table. Therefore it is better to apply a set of rules to transform the initial query tree to an optimized tree that is more efficient to execute. The result is shown in the previous figure, where basic database operations such as **SELECT** and **PROJECT** are applied to individual tables so to reduce table sizes before the **JOIN** operation is applied to joining multiple tables. The process of converting an initial query tree to an optimized query tree is called query optimization. Since each basic database operation may be implemented in many different ways, the optimization also involves choosing the most efficient implementation for each operation. For example, there are at least six different ways to implement a **SELECT** operation, including a brute force linear search, a binary search, using a primary key or hash key, and so on, four different ways to implement a **JOIN** operation, and two different ways to implement a **PROJECT** operation. Query optimization in a commercial database is normally done by the database engine using built-in index structures on the host computer.

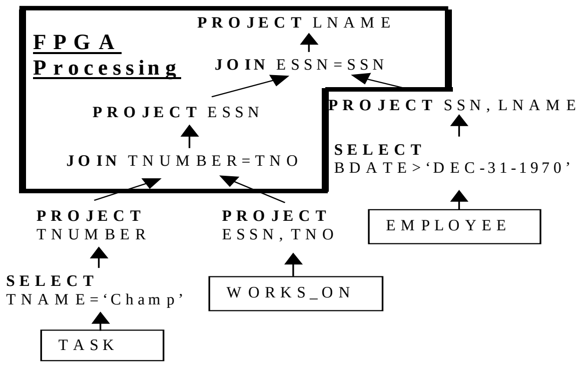
In order to use FPGA computing resources to drastically speed up query execution, we propose to develop a "query preprocessor" that optimizes queries by taking into account the availability of FPGA resources. The input to the query preprocessor is a query application in "embedded SQL," which is basically a program in the C language with SQL statements embedded in it. (Those SQL statements are enclosed by pre-compilation directives.) The "query preprocessor" will separate embedded SQL statements from the rest of the application and convert them into two parts, one for the FPGA resources and the other for the back-end database engine. The rationale is, because of the finite amount of FPGA resources, it may not be possible to squeeze all the query operations into the FPGA coprocessor board. In addition, in the case when there is not enough FPGA resources, the query preprocessor will need to evaluate whether to reconfigure the FPGA for more database operations or simply allocate those operations to the host machine. This also implies that optimization opportunities available through index are not wasted. More specifically, the steps involved in using the query preprocessor will be as follows.

1. The preprocessor isolates SQL statements from an embedded SQL application.
2. The preprocessor converts a SQL query to a query tree and transforms it to an optimized tree.
3. The preprocessor identifies "sub-trees" that can be accelerated with FPGA computing. Each sub-tree will be replaced with C codes that controls the FPGA coprocessor board while the remaining part of the tree will still be coded in SQL.

4. After being compiled, the final application code can be executed. A commercial database backend will be invoked for the SQL part and an FPGA board for the sub-tree C code. There will be parallel operations for the sub-trees allocated to the FPGA board.

Tree Partitioning

The query tree is partitioned into two parts, one for the FPGA and the other for a back-end database engine on the host machine, as shown in the following figure.



In this example, instead of sending the three tables to the FPGA board directly, we first apply **SELECT** and **PROJECT** operations on those three tables and then send to the FPGA board the resulting tables which are supposed to be much smaller. For example, after the table TASK goes through the **SELECT** TNAME= 'Champ' and **PROJECT** TNUMBER operations, the resulting table has only one single column, i.e., TNUMBER, and contains only those rows whose TNAME is 'Champ'. As shown in the figure the FPGA board is responsible for two **JOIN** operations and two **PROJECT** operations.

FPGA Computing

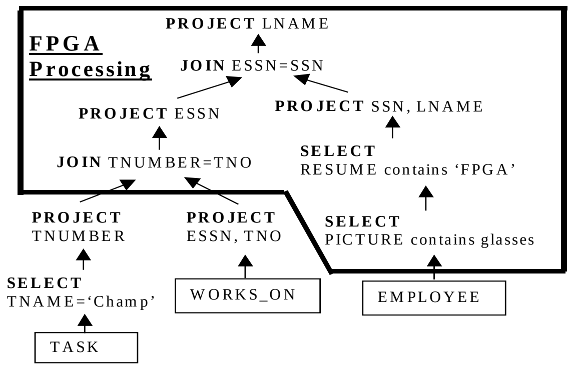
Depending on the size of the three tables sent to the FPGA board, different FPGA designs could be used. Computation of those **JOIN** and **PROJECT** operations can be pipelined to increase speedup.

For the **JOIN** TNUMBER=TNO operation, suppose there are only ten TNUMBER rows. In that case, ten comparators can be used for every incoming TNO to see if there is a match. However, since the number of TNUMBER rows cannot be pre-determined, a more complicated design that includes comparators, FIFOs, and a controller will be more appropriate. The speedup will depend on how many comparators can be squeezed into the hardware.

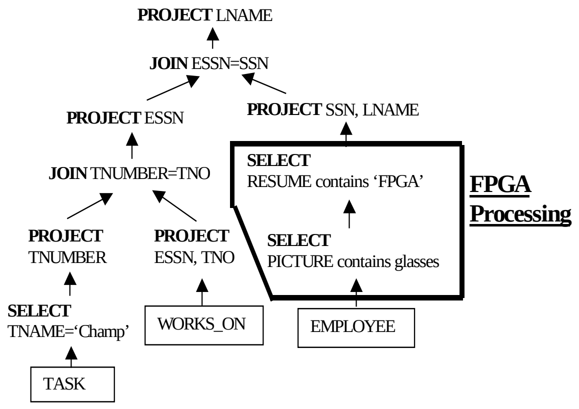
For the **PROJECT** ESSN operation, it is necessary to remove duplicate ESSN rows and keep only distinct ones. This can be done either with brute force comparisons or with hashing. In either case, parallel comparison can again be applied. (Computation of the hash function can be pipelined with the comparison, in the case of using hashing.) The query preprocessor will need to make the decision and choose the most efficient way of implementation. Note that the three tables in the database are locked during the whole transaction process so as to avoid any modification to the tables during the transaction by other applications.

Example 2

This example corresponds to another query **Q2** (i.e., Find the last names of employees wearing glasses who work on a task named "Champ" and whose resume contains the keyword "FPGA"). In this case text/image databases are involved.



As shown in the previous figure, the image processing of the pictures and the searching of the text documents should be handled with the FPGA resources As a matter of fact, if there are not enough FPGA resources, the query preprocessor may decide that it is better to implement ONLY the text/image processing part on the FPGA since those parts provide the maximal amount of parallelism. Such a partition is shown in the following figure. Apparently there are many design options other than the two previous designs. It is the responsibility of the query preprocessor to rank those different design options and choose the "best" one for implementation.



Query Preprocessor Internal

The figure as shown at the end of the paper schematically illustrates the two major tools that need to be developed: "Query Preprocessor" and the "FPGA Design Tool". Given a C/C++ application program (i.e., C/C++ file) that contains ODBC statements as an input, a user may identify some parts that have nothing to do with database operations but can be accelerated with FPGA boards. The user can then use some domain-specific FPGA design tools, whether they are for image processing, video processing, or text string processing, to produce some FPGA designs and insert the corresponding host code interface in the C/C++ file. In addition, the user may add some preprocessor directives to help specify the parts in the C/C++ file that can be put into a query tree. The resulting C/C++ file, FPGA design files, and FPGA library of basic database operators act as inputs to the "Query Preprocessor" and the "FPGA Design Tool".

4. Conclusions

Multimedia databases have many applications that need to be accelerated. In this paper some such potential applications are accelerated by using a commercial FPGA coprocessor board based on XC4085. FPGA designs for text searching and image matching are described and their performances summarized. A potential design for a database JOIN operator is studied and a query optimization preprocessor is then proposed.

Acknowledgments

This research was supported by Systran Federal Corp., a DAGSI/AFRL grant and an Ohio State research challenge grant.

REFERENCES

[1] J. Villasenor, B. Schoner, K. Chia, C. Zapata, H. Kim, C. Jones, S. Lansing, and B. Mangione-Smith, "Configurable Computing Solutions for Automatic Target Recognition," in IEEE Symposium on FPGA Custom Computing Machines, pp. 70--79, 1996.

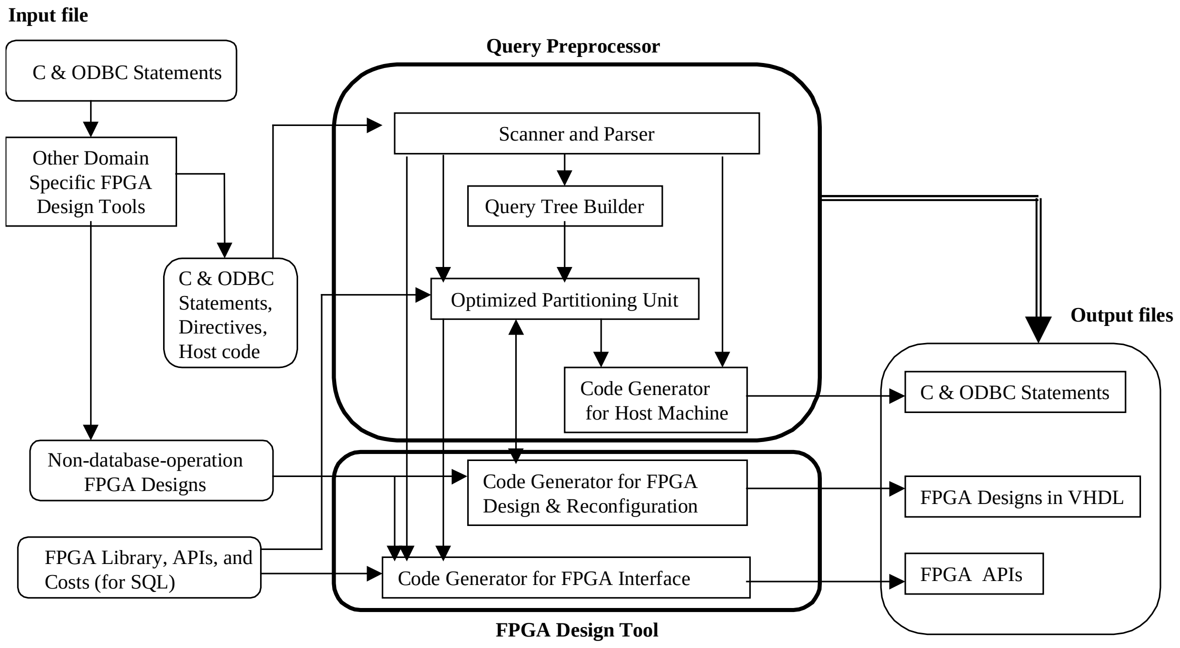
[2] J.S.N. Jean, X. Liang, B. Drozd, K. Tomko, and Y. Wang, "Automatic Target Recognition with Dynamic Reconfiguration," to appear in Journal of VLSI Signal Processing.

[3] M. Alderight, E.L. Gummati, V. Piuri, and G.R. Sechi, "A FPGA-based Implementation of a Fault-Tolerant

Neural Architecture for Photon Identification," in Proc. of ACM/SIGDA International Symposium on FPGAs, pp. 166-172, 1997.

[4] M. Shand and L. Moll, "Hardware/Software Integration in Solar Polarimetry," in IEEE Symposium on FPGA Custom Computing Machines, pp. 18-26, 1998.

[5] H. Blume, H.M. Bluthgen, C. Henning, and P. Osterloh, "Integration of High-Performance ASICs into Reconfigurable Systems Providing Additional Multimedia Functionality," IEEE International Conference on Application-Specific Systems, Architectures, and Processors (ASSP), July 2000.


Input file
C & ODBC Statements
Other Domain Specific FPGA Design Tools
C & ODBC Statements, Directives, Host code
Query Preprocessor
Scanner and Parser
Query Tree Builder
Optimized Partitioning Unit
Code Generator for Host Machine
Non-database-operation FPGA Designs
FPGA Library, APIs, and Costs (for SQL)
Code Generator for FPGA Design & Reconfiguration
Code Generator for FPGA Interface
FPGA Design Tool
Output files
C & ODBC Statements
FPGA Designs in VHDL
FPGA APIs