Integrating FPGA Technology and Database Management Systems on Solaris Systems

Daryl Popig[1], Debra Ryle[1] and Eric Stahlberg[2]

[1]Franklin University, 201 South Grant Street, Columbus, OH 43215
Email: {ryle01, popig01}@email.franklin.edu
[2] Ohio Supercomputer Center, 1224 Kinnear Road Columbus, OH 43212
Email: eas@osc.edu

Abstract

Database management systems have evolved as efficient mechanisms to manage large and diverse amounts of data. Rapid searching and analysis of database information is a normal requirement of most database management systems. This can be a particularly challenging aspect to maintain with semi-structured or unstructured data. The advent of more readily programmable Field Programmable Gate Array (FPGA) systems offers a significant opportunity to address the challenge of rapid searching for semi-structured or unstructured data within the context of a database management system. Combining the robust Sun operating environment for databases with the custom programmability of the FPGA provides a particularly suitable solution environment. By researching the problem of searching relational databases in general, and life-science databases in particular, this presentation will cover the steps taken and recommendations made for generally integrating FPGA technology for use with database management systems.

1. Introduction

Efficient database searching remains a key priority for businesses today. The increased effort to bring increasing amounts of unstructured data under management adds a unique challenge to maintaining high-speed searching. The traditional approach remains to create, manage, and update large numbers of indexes to achieve fast searching. Unfortunately, this requires that the indexes become very large while pre-computing and storing predicted search elements for unstructured data.

Field Programmable Gate Arrays (FPGAs) have demonstrated promise of potential increased speed for searching unstructured bioinformatics data for a number of years with such products as the DeCypher system from TimeLogic Inc. Recent availability of sufficiently capable *and programmable* FPGA hardware in enterprise-class systems has opened the door for closer integration of FPGA technology in enterprise operations.

This project's objective was to explore the feasibility of using specialized external FPGA hardware to rapidly search through largely unstructured data stored in an enterprise database management system. This paper explores integration between database and hardware technologies to deliver acceptable performance for complex data searching. By utilizing the FPGA technology for search algorithms, we

take advantage of the low level hardware's inherent parallelism to increase the speedup of the search process. Searches were initiated from a Solaris system running a high performance Oracle database utilizing a FPGA accelerator with high bandwidth and low latency interconnects. The hardware was custom programmed for the desired search criteria.

2. Motivation

Increasing amounts of bioinformatics and chemistry data are being collected in databases as a result of more pervasive laboratory automation. Frequently, the information collected is unstructured and unindexed due to the variability of the experiments, equipment, and diversity of research exploration. Providing an additional challenge is the limited support for unstructured text searching with standard SQL. Rather than attempt to impose a common data structure and large amounts of indexing on the collected information, we decided to pursue a project to exploit FPGA technology and search the information in a raw and largely unstructured form.

OSC has utilized FPGA hardware in its operational environment since 2001, utilizing the TimeLogic DeCypher system for high-speed bioinformatics sequence searching. Specific canned algorithms were provided by the vendor for common algorithms of BLAST, Smith-Waterman, and Hidden-Markov Model searching. The FPGA accelerated algorithms regularly demonstrated search speedups on the order of 40 to 100 times relative to non-FPGA implementations. Unfortunately, the closed environment of the TimeLogic system precluded extending the technology for more generalized searching.

Rather than make the simplifying assumption that the database system and FPGA would co-exist on the same system, we decided to define the project scope to depict an existing production enterprise database environment. In this case, we chose the common scenario of an Oracle 10G database running on a Solaris server. We further decided to focus efforts early on the common need for rule-based and "fuzzy" string searching in a largely unstructured domain of character data -- in this case chemical and bioinformatics data. By focusing on string searching as the prototype implementation, the technology and capabilities demonstrated in the project have direct application in many other general database searching situations.

Having completed the proof-of-concept project, a noninvasive development environment is created for defining and testing more complex search algorithms, specifically for bioinformatics data, two-dimensional chemical structure data, and three-dimensional spatial searching where the parallelism available in the FPGA can be more completely exploited.

Advantages of FPGA Searching

The advantage of reconfigurable computing opens the possibility of using hardware to implement or enhance a specific algorithm through hardware acceleration. As the gate densities increase in the newer FPGA devices, and as better development tools emerge for using them, more opportunities are now available for implementing this type of

reconfigurable hardware into an FPGA-based computing platform.

A FPGA is a microchip that contains an array of configurable logic blocks that act as programmable elements and a programmable switch matrix. These blocks are routed internally through the column-based channels (switch matrixes) that can interact with other programmable blocks, or it can be routed to an external I/O pin. Current FPGAs can contain well over one million logic elements along with SRAM including microprocessor cores. These FPGAs can easily be programmed by sending a bit pattern stream to the standard JTAG port. After programming is complete, the device is ready to be used.

The benefits of FPGAs in supercomputing are realized when intensive computations are being done in parallel inside the hardware. This will free other processors to do other work tasks. FPGAs perform tasks much more efficiently than a software program doing the same calculation by going through a series of steps. To contrast this, a serial program or thread has to fetch each instruction and then execute it. These fetch and execute instructions eat CPU cycles. As a large volume of data manipulation is involved, a high number of CPU cycles will be eaten by a software solution. However, with FPGA hardware, there is no fetch and execute cycle. The data is streamed in and usually some sort of combination logic is used to look for patterns or high-speed logic blocks such as adders, matrix multiply, or shifting operations are used. The comparison is performed on the data at the FPGA's clock speed of 150 to 200 Mhz, providing the potential for a very large speedup over that of the pure software solution. With the trend of larger gate counts and faster clock rates, the potential of FPGA accelerated database searching becomes increasingly significant.

3. Equipment and Environment

This project was completed using systems at the Ohio Supercomputer Center Springfield (OSC-S) site, which focuses on data-intensive computing infrastructure and applications research.

Networking Interconnect Environment

OSC Springfield is connected to the primary OSC site through Ohio's Third Frontier Network (TFN). TFN, a statewide fiber-optic network managed by OSC, connects major metropolitan areas in Ohio with high network bandwidth. Internal connectivity between the database server and external FPGA host system consists of gigabit ethernet connectivity.

Database Server Environment

A Sun V1280 operating Solaris 9 was used as a representative mid-size enterprise database server. Oracle 10G (10.1.0.2) was chosen as the database management system for this project. Oracle is widely used for scientific data storage and offers seamless application integration with "C" and Java, as required to communicate between the FPGA and the end-user client sessions. The FPGA API required "C" for programming interfaces between the database and FPGA. We were able to leverage Oracle's PL/SQL feature to execute external "C " procedures launched from a client's JDBC session,

passing the query criteria and data to the FPGA for processing.

FPGA Environment

The FPGA hardware environment used for this project was hosted on the Cray XD1 high performance computer system. The current system configuration breaks down as:

- Twelve 64-bit AMD Opteron processors configured as six two-way symmetric processors (SMPs) with a clock frequency of 2.2 Ghz. running on Linux
- Twelve Rapid Array Processors for handling the communications within the chassis switch fabric
- Six Xilinx Virtex-2 FPGA of part type XC2VP50 with a clock frequency of 199 MHz acting as application acceleration processors

The FPGA is accessed by the Opteron processor through the HyperTransport bus interface, which is running at 800 MHz with 16-bit transfers being done on every clock. The FPGA is a 128 Mbyte window mapped into the virtual address space of the Opteron processor. A set of API functions written in "C" are used to interface to the FPGA. HyperTransport requests are issued to the Rapid Array Processor (RAP) when the Opteron does a read or write access to the FPGA address space. These Read/Write requests are forwarded by the RAP to the FPGA through a RapidArray Transport (RAT) interface. This interface is provided by Cray as a VHDL core and as part of the internal FPGA interface logic in the RT core. The RT core is essentially a state machine that has a transfer start, data valid, read/write strobes and address decode along with RapidArray fabric request acknowledgements. The speed of the RAT matches that of the HyperTransport bus, but has a simpler protocol to reduce the logic inside the FPGA.

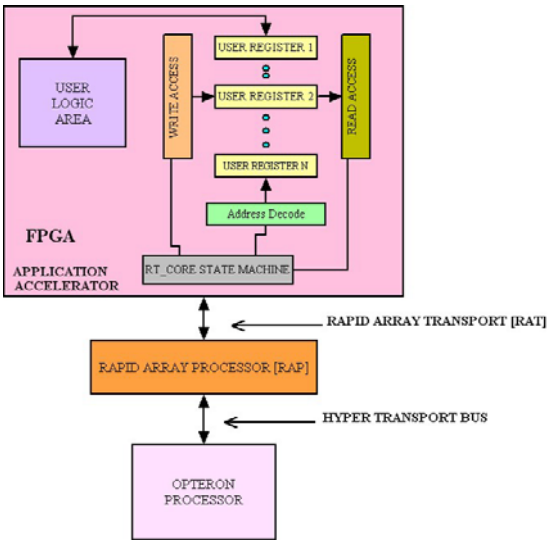

Figure 3.1 FPGA Internal Architecture

All of the FPGA code and the substring search code were written in VHDL using the Xilinx ISE software version 6.3 on a Windows XP platform. VHDL is a type of hardware descriptive language and was chosen as our programming tool due to Cray's previous application accelerator development with the Xilinx ISE tools.

4. Technical Approach

Programmable FPGAs can be easily integrated into an enterprise infrastructure to validate the impact on database searching. Relational database management systems such as Oracle, DB2, MYSQL and SQLServer, offer high performance techniques to store and retrieve data. Integration with Oracle utilized PL/SQL packages and "C" shared libraries to retrieve data and send information through network

connections to a remote host with an FPGA for searching. A simple substring search was chosen as the remote FPGA search program to provide comparability to functionality present with standard SQL.

The Substring Search

Our initial research showed that a simple brute force strategy would be just as effective as a more complicated search algorithm. This project's application made use of 11 quadword size (64 bits) user registers accessible to the Opteron processor. Three registers were used together to make a word width of 24 bytes in length. Three were used for the search key, three for the search data, and three for a key mask. The other two registers were used for control and to read the search results.
We set our word lengths at 24 bytes. This was accomplished by using three sets of three 64-bit user registers as shown in Figure 4.1. These register sets supported the following:

- Input data string to be searched
- The search key
- A mask to handle variable length search strings

When new data is written to the FPGA registers, a bit called "new data" is set. This bit causes the key, data, and mask to be loaded into the substring search registers on the final data write. When new data is cleared, the synchronous search is started and the data shifting is clocked at the FPGA user clock frequency.

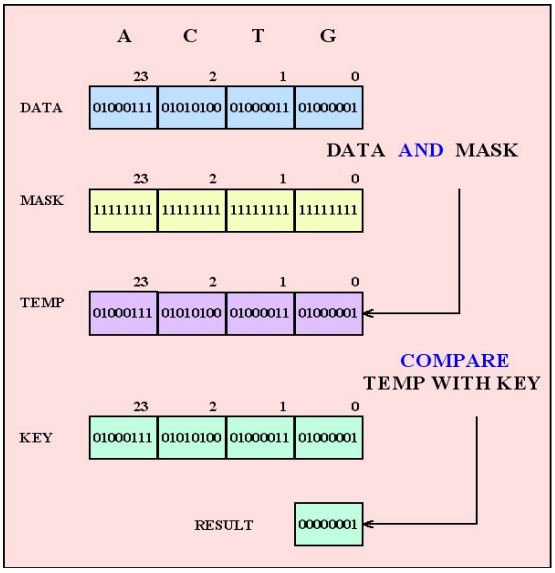

Figure 4.1 – FPGA Application - Search Matching Logic

The steps involved at the hardware level for exact string matching are as follows:

1. Shift each byte in the data register to the left, and then feed the top byte back to the beginning.
2. Compare the data register against the mask register (all bits on), with an "AND" operation to mask off bytes that are outside of the key length.
3. Store the result of the "AND" operation into a temp data register.
4. Check to see if the key is the same as the shifted/masked-off data by doing a comparison. If they are equal, set the LSB position in the result register.

Database and Application Integration

The first challenge was to determine how the client would reach an application that could send data to the FPGA for processing and return the results back to that session. We want the client to reach this application through a "thin-layer" browser session using JDBC connectivity to the Oracle database. Once the request is sent to the Oracle

database, PL/SQL packages are executed to retrieve data and send rows out externally by calling a “C” shared library file to pass data to the FPGA. This client session remains persistent waiting for results of matching rows to be returned to the FPGA.

The second challenge was to determine how to communicate seamlessly between the database and FPGA although they were not running on the same server. The FPGAs were installed on a server inside a Linux cluster that was not directly reachable from the database server. Ideally, the optimal solution is to locate the database locally on the same server as the FPGAs but we had to overcome the constraint of using the FPGA accelerators remotely. Figure 4.2 illustrates our solution to use TCP stream sockets (2 sets) to provide communications directly between the database and FPGA. One set of sockets opens a path between the database server and Cray head node (known as node6 of the cluster to the outside). The second set of sockets opens a path between node6 and the Cray node (node1) configured with the FPGAs. We refer to this pathway as the Remote Server Socket (RSS) tunneling.

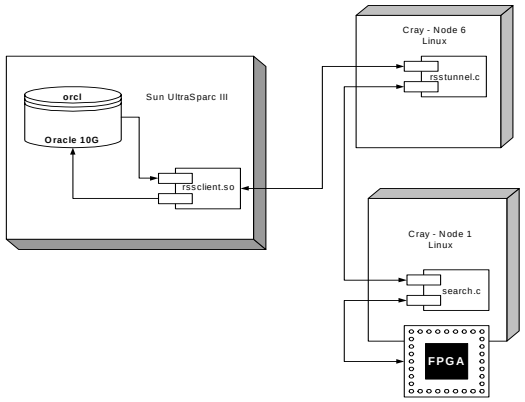

Figure 4.2 Remote Server Socket (RSS) Tunneling Architecture

With this architecture, we were successful at sending 100,000 rows of data through both socket layers to the FPGA and returning results back to the client session.

Concerned with the performance we were seeing (results are published in Results, section 5), we chose to simplify the architecture by relocating the database on the head node server (node6) in the Linux cluster. This would still allow for remote client connections from web interfaces but reduces the concern for I/O and network overhead encountered in the sockets tunnel.

Figure 4.3 shows the optimized architecture can still access the FPGAs remotely yet simplifies the database connectivity. Database requests now stream through only one socket layer.

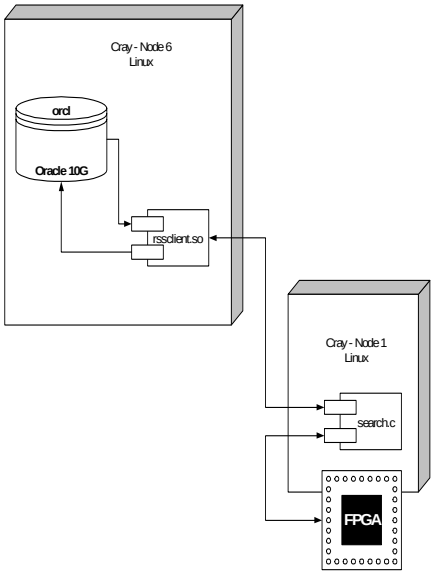

Figure 4.3 Optimized Remote Access Architecture

Pre-execution Configuration and Initialization

It is important to note that before any client can request an FPGA search, two steps *must* be completed or have been started in the background on the node1 server:

1) The server sockets program must be running to open port 10241 listening for client requests. The server sockets programming code is included in the Search API program, search.c, and can run as a Unix background process.
2) The FPGA chip must be loaded with the customized program containing the programmable logic. In our project, the substring search logic is compiled in the program called subsearch50.bin and loaded using the fcu utility program.

Oracle Programs

To start the request for a FPGA search, the client attaches to an Oracle session with rights to execute a PL/SQL package, MDT.FPGA_SEARCH. The following sample code shows how to retrieve results from a FPGA search using SQL*Plus interface:

```
declare
        v_return_value varchar2(127);
begin
for  a in  (select rownum, title_no from
mdt.pdb_title)
loop
v_return_value =
MDT.FPGA_SEARCH.SEND_KEY(key_stri
ng'||'~'||a.rowid||'.'||a.title_no);
end loop;
```

The delimiter "~" separating the key and row data is required by the FPGA Search API. (For generality and ease of implementation, we sent both the key and data for each row during our simple implementation but this could be optimized later.) The results are received into the variable v_return_value and are to be used in the client's application for displaying.

Oracle supports the execution of external programs (outside the database) from within its PL/SQL packages. In our application, we send one row of data as a string parameter to an external "C" program. Figure 4.4 shows the Oracle step starting with an initiation of a new RPC process. Next, the new session spawns an extproc agent configured through SQL*Net. The remote process can execute a "C" program in a shared library.

To build our search package in Oracle, the following database objects are created:

1. Create a LIBRARY object giving the full path to the shared library:

```
CREATE LIBRARY rssclient_c as
 '../../scripts/bin/rssclient.so';
```

2. Create the FPGA_SEARCH package, containing a function with the following required syntax:

```
CREATE OR REPLACE PACKAGE
            BODY FPGA_SEARCH
  AS
  FUNCTION
            SEND_KEY(SEARCH_KEY
            IN VARCHAR2)
       RETURN VARCHAR2
  IS
  EXTERNAL
  LIBRARY rssclient_c
  NAME "fpga_search"
  LANGUAGE C
  PARAMETERS(SEARCH_KEY
            string,
         RETURN STRING);
end FPGA_SEARCH
```

Notice that the FPGA_SEARCH.SEND_KEY function has parameters for both sending and receiving strings of data as it executes a "C" function called fpga_search() in the shared library, rssclient.so. The fpga_search() program starts a client socket session, sends the data passed from the database, and waits to hear a reply from the FPGA about whether this string contains a match. As a reply is received, it is returned to the Oracle session initiating the request. That request reply is passed back to the Oracle function as a return value. In our previous search example, the variable v_return_value, holds the results from the completed FPGA search.

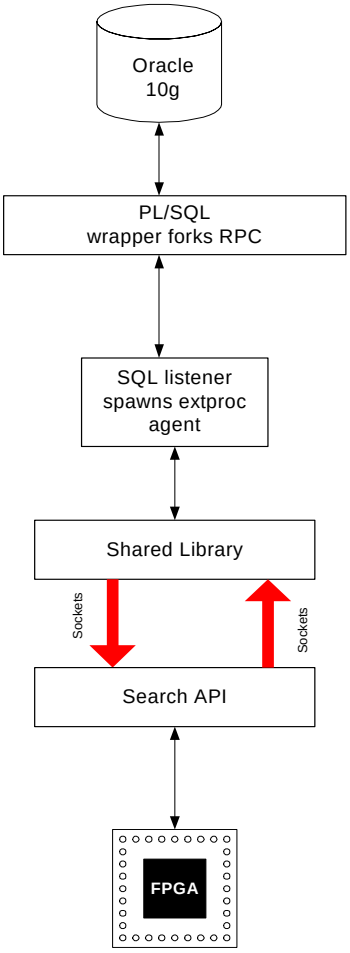

Figure 4.4 – FPGA Search Request Architecture

Socket Communication

TCP stream socket connections are used to open an application connection between the database server (node6) and FPGA server (node1). Port 10241 must be opened on node1 and listening for client requests from node6. The logic of the sockets programming is built into two programs, client and server. The rssclient.so shared library file contains client logic that receives a row of data from the database session into a temporary buffer, opens a socket on any port of node6, and "sends" the buffer over to the server socket listening on node1. The client's socket remains open

waiting for a "receive" call back from node1 with results of the FPGA search. Once the "receive" call returns, the results are sent to the database client's session buffers and the client's socket is closed.

Search API Program

The Search API program is written in "C" and handles three tasks. The first task opens a connection to the FPGA, and the second opens and maintains the socket server connection to port 10241 on node1. The server runs a "while loop" logic listening for client socket requests. Neither the FPGA or sockets server closes unless the client sends a string value of "SHUTDOWN." This shutdown logic has been helpful to ensure that all buffers are completely flushed in our benchmark testing.

The third task is to handle incoming client requests by parsing the received string data into key and search strings, passing them into the FPGA registers, and initiating the search. The API program then issues a "READ" request to the FPGA to determine if a match has been made. If the match is successful, the row data is "sent" back to the client. If the match is unsuccessful, a NULL value to "sent" back to the client. The client's socket is closed and the sockets server continues to listen for the next request.

5. Results

Our findings had unexpected but promising results. While programming for the FPGA was not trivial, we are able to show the feasibility of seamless integration with the database technology even between distinct hosts. To evaluate FPGA integration potential, we tested the FPGA substring search algorithm against a similar database search performed on the Oracle database. A common table was used in both searches containing 100,000 rows of alpha-numeric, unindexed data.

Four different test scenarios were used:

Test 1	Invoke a local SQL*Plus session running on the database server through single socket agent. (one hop)
Test 2	Invoke a remote SQL*Plus session running on the Internet connection through single socket agent. (one hop)
Test 3	Invoke a local SQL*Plus session running on the database server through double socket agents. (two hops)
Test 4	Invoke a remote SQL*Plus session running on the Internet connection through single socket agents. (two hops)

Test results show that the Integrated FPGA search performed slower than the database search, as seen in Figure 5.1. In test 2, the FPGA search was only three milliseconds longer than the database search but all other benchmarks show considerably longer results.

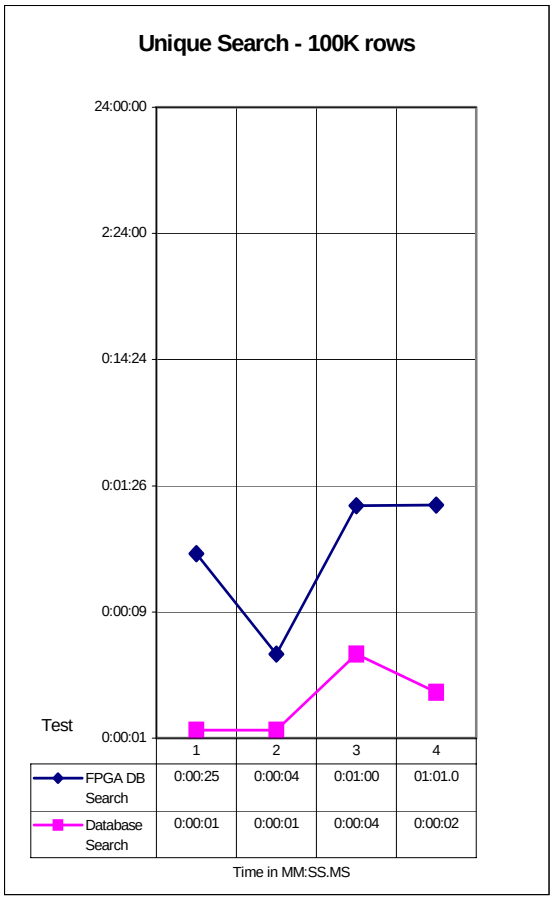

Figure 5.1 - Unique row searching, Integrated FPGA vs. Database

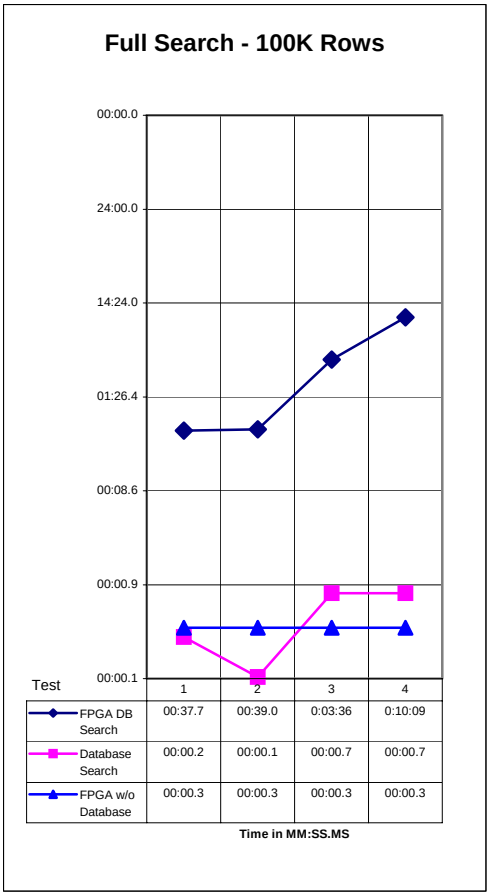

Figure 5.2 – Full table searching, Integrated FPGA, Database, and FPGA

Our second set of testing on a full 100,000 row search for duplicate values continued to show the same trend with database searches returning results faster than the Integrated FPGA as shown in Figure 5.2. Included in Figure 5.2's graph are the benchmark results from a FPGA search test run directly on the FPGA server with no network overhead.

Closer examination helps explain FPGA integrated search performance delays. In our application, we send data one row at a time to the FPGA over the network and back. Our proof-of-concept application depended heavily on continuous data movement across the sockets network through the Linux OS into the FPGA as shown by our timing results. In contrast, the database search takes advantage of the local server memory structures and the 8K block fetches for its performance. Not surprisingly, a configuration with the FPGA local to the database server is expected to be optimal.

Using a timing program provided by Cray for the FPGA, we tested a brute-

force search directly (with no network interface) against an API written in "C." After the new data flag is set, the shift register is going to start shifting left and comparing on each positive side of the user clock of 199 MHz. For each substring search, the FPGA benchmark is .12 microseconds:

((1/199) x 10^6 * 24) = 5ns * 24 = 0.12uS

Performing a search through 500,000 randomly generated values, the FPGA took 1.7 seconds to finish. Using the raw FPGA and Test 1 benchmarks, findings confirm that the overhead of moving the data in and out of the FPGA accounts for 37.4 seconds of the total 37.7 seconds.. Figure 5.2 shows the FPGA spends only .3 microseconds to search through 100,000 data values. With further reduction in I/O overhead, the FPGA now becomes a viable solution nearly matching the same performance as the database search. Database applications using a family of algorithms that used FPGA pipelining and internal data movement would showcase it as superior for an application accelerator.

6. Challenges and Limitations

Since the application of FPGAs in high performance computing platforms is still quite new, there are many unanswered questions.

The first challenge is to define which algorithms can readily exploit FPGA fine-grained parallelism for searching. A closely related challenge is algorithm partitioning and determining which proportion of a given algorithm should be implemented into the FPGA as hardware.

The second challenge is the level of expertise needed to work with hardware descriptive languages such as VHDL or Verilog. Even though tools are currently being developed to take an algorithm written and tested in "C" and then convert it to VHDL, knowing VHDL will still be required.

A third challenge to more readily exploit FPGAs in database searching is the absence of standard interfaces with the IP core. We found a lack of common APIs to transfer data into and out of the FPGA. Standards will be needed to make it easier to integrate applications at a higher level of abstraction than software can provide.

A fourth challenge focuses specifically on FPGA code development. As the logic design grows inside the FPGA and newer FPGAs become increasingly dense, the Place And Route (PAR) times are increasing exponentially. (The PAR time is the time it takes the development platform to map the logic functions from the VHDL onto the target FPGA.) The PAR times are rising with the increased number of gates and corresponding algorithm complexity. At the same time the amount of memory required on the development platform has also increased. While virtual memory is a possible option to address the increased memory demands, using a swap file on the hard drive would even further increase the PAR time beyond the level of acceptability.

7. Conclusions

Our experience with this project proved that the concept of noninvasive interfacing an FPGA to an enterprise database system is entirely possible. We

have demonstrated a seamless integration strategy with a popular database environment (Oracle 10G on Solaris) and a programmable FPGA server (Cray XD1). While using resource management frameworks such as the Sun Grid Engine, we wanted to demonstrate the feasibility of hardware resource sharing without first deploying more complicated grid and resource management infrastructures. In both deployment architectures, we were successful at application integration with a remote FPGA. This further demonstrates the ease in which an FPGA server can be integrated into a production database environment, with minimal disruption, for query research, development and optimization.

Programming FPGAs is not trivial, but that challenge is now less of a burden due to increased capabilities of FPGAs. FPGAs are now more attractive to re-configurable computing applications for which they would not have been considered in the past. Due to the higher logic densities and new developer tools, the field of reconfigurable computing opens new doors in areas such as bioinformatics, digital signal processing (DSP), video image processing, packet processing and data encryption -- just to name a few.

Acknowledgements

The compressed timeframe and success of this project would not have been possible without significant external support. The authors would like to acknowledge the assistance and cooperation of Cray Inc. and in particular Steve Margerm for providing critical information and support when developing the FPGA search application. We would also like to thank Sherie Vallo who represents Xilinx Inc. with Avnet Electronics Inc for her help in providing the ISE development tool. We would also like to thank Don Swartout of Franklin University for providing sample chemistry data. We would also like to acknowledge Sherry Sun, Jim Gregory and Pete Carswell of OSC for technical support accommodations made to the OSC Springfield environment which enabled the completion of this project.

References

Compton, K. & Hauck, S. (2002). *Reconfigurable computing: A survey of systems and software*. Retrieved Nov 12, 2004, from the World Wide Web: http://www.ee.washington.edu/people/faculty/hauck/publications/ConfigCompute.pdf

Fechner, U., Franke, L., Renner, S., Schneider, P. & Schneider, G. (2003).*Comparison of Correlation Vector Methodes for Ligand-Based Similarity Searching.* Journal of Computer-Aided Molecular Design, 17, 687-698.

Harrison, A., South, D., Willett, P. & Artymiuk, P. (2003) *Representation, Searching and Discovery of Patterns of Bases in Complex RNA Structures.* Journal of Computer-Aided Molecular Design, 17, 537-549.

Hunter, J. (2004). *Calling os commands from pl/sql using external procedure*. Retrieved December 27, 2004 from World Wide Web: http://www.idevelopment.info/data/Oracle/DBA_tips/PL_SQL/PLSQL_1.shtml

Jean, J., Dong, G., Zhang, H., Guo, X. & Zhang, B. (nd*). Query Processing with An FPGA Coprocessor Board*. Retrieved Nov 05, 2004, from the World Wide Web: http://www.cs.wright.edu/people/faculty/gdong/RelQoptimizeFPGA.pdf

Kramer, A., Horn, H., & Rice, J. (2003*). Fast 3D molecular superposition and similarity search in databases of flexible molecules.*, Journal of Computer-Aided Design, 17, 13-38.

Kuramochi, M & Karypis, G. (2002). *An Efficient Algorithm for Discovering Frequent Subgraphs*. Retrieved Jan 22, 2004, from the World Wide Web: http://www.cs.umn.edu/~kuram/papers/fsg-long.pdf

Pellerin, D. & Taylor, D. (1997). *VHDL Made Easy,* Upper Sadle River: Prentice Hall.

Skahill, K. (1996). *VHDL for Programmable Logic,* New York: Addison-Wesley.

Urman, S., Hardman, R. & McLaughin, M., (2004) *PL/SQL Programming*, Oracle Press New York: McGraw-Hill