Exploiting Reconfigurability for Text Search

Roger D. Chamberlain, Mark A. Franklin, Ronald S. Indeck
Exegy Inc., St. Louis, MO
{rchamberlain,mfranklin,rindeck}@exegy.com

Introduction

Since their inception, it has been clear that, in principal, FPGAs can be reconfigured at will; that is, whenever desired. In practice, however, this capability is rarely used. Typically, during application development new configurations are regularly loaded and tested. As part of support operations, new versions of applications are loaded into on-board non-volatile storage and used to alter FPGA configuration at the next reboot. Beyond this, however, there is little production use of the ability to reconfigure modern FPGAs.

At Exegy Inc., we have constructed a network-attached appliance that exploits frequent FPGA reconfiguration as its regular mode of operation. When an application is invoked, the appropriate configuration is loaded into the FPGA. In addition, which configuration is appropriate is not simply a function of the application itself, but is also a function of the parameters provided to the application. Here, we describe the practical utility of regular FPGA reconfiguration in response to user needs.

The Exegy A2000 Appliance

Exegy Inc., in collaboration with Washington University in St. Louis, has developed the A2000 appliance [1,2], a network appliance that provides network-attached storage augmented with a high-performance, application-level computing capability.

Figure 1 illustrates the internal architecture of an individual appliance. Data flows off the disks into an FPGA. The FPGA provides reconfigurable logic that has its function specified via HDL. Results of the processing performed on the FPGA are delivered to the processor. By delivering the high-volume data directly to the FPGA, the processor can be relieved of the requirement of handling the bulk of the original data set.

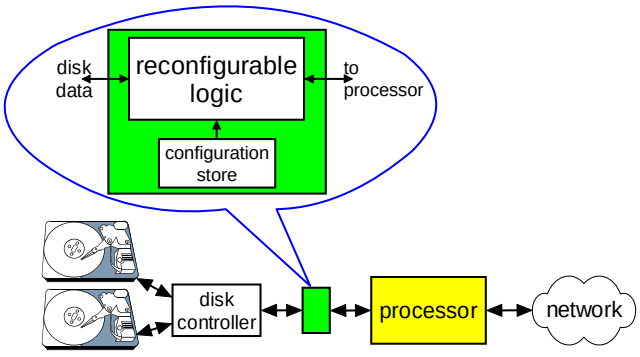

Figure 1: Exegy appliance architecture

Associated with the reconfigurable logic is a configuration store that maintains a fixed number of FPGA configurations. This configuration store is managed by software, with the ability to *insert* configurations into the store, *read* configurations from the store, and *load* configurations from the store into the FPGA. While the insert operation is slower (due to the limitations of the flash memory used for non-volatile storage), the read and load operations require only 20 ms to complete. This is comparable to disk operation times associated with seeking and/or rotational latency.

Text Search Application

A commercial product that runs on the A2000 appliance is Exegy TextMiner Version 1.2. TextMiner supports text search on unindexed bulk data sets at rates of 600 MB/s or greater from the on-board data store, 400 MB/s from an attached SAN, or 800 MB/s from a 10 GigE network. Search functions include exact matching, approximate matching, regular expression matching, and combining operations. On the appliance, the three alternative matching operations are deployed on the FPGA with the combining operations taking place on the general-purpose processor.

Exact match. The exact matching operations are based upon Rabin-Karp theory [3]. The algorithm is as follows. The keywords of interest are hashed into a bit-vector position. Text to be searched is then hashed and the resulting bit-vector position is checked for the presence of a keyword. On a hit, there is either a keyword match or a hashing collision. In either event, the hit is delivered from the FPGA to the processor where software determines whether a true positive keyword match or a false positive hashing collision has occurred.

The exact match search engine can search for tens of thousands of keywords in a single pass over the data set. With an ingest capability of 8 characters per clock and running at 100 MHz, a single engine can support a throughput rate of 800 MB/s.

Approximate match. With approximate matching [4], keywords in a query can be specified with a number k of allowed character substitutions or miss-matches. Keywords can be specified to be either case sensitive or case insensitive. Also, individual characters in a keyword can be designated as “don’t care” and will match any character.

The approximate match search engine is illustrated in Figure 2. Input data flows through a shift register at the top of the figure. Keywords are stored in a set of compare registers. Fine-grain comparison logic determines whether there is a match at the character level. This includes bit-masking capability to support wildcarding (e.g., don’t cares) and case insensitivity. The count of character matches out of the fine-grain comparison logic is checked

The authors are on leave from Washington University in St. Louis at Exegy Inc., www.exegy.com

against a threshold in the word-level comparison logic and a match signal is asserted if the requisite number of character matches is detected. In the example of the figure, the keyword "house" will match the data "horse" if $k = 1$ (the number of allowed character substitutions is one).

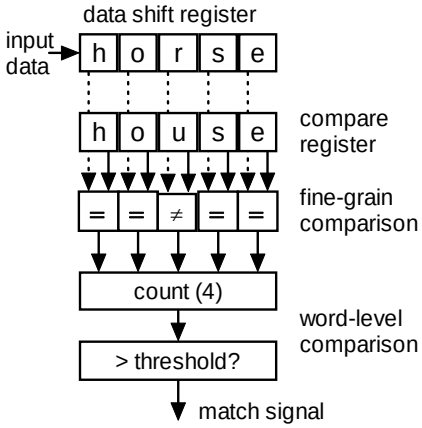

Figure 2: Approximate match search engine

The approximate match search engine supports tens of keywords being searched in a single pass over the data set. With an ingest capability of 8 characters per clock and running at 100 MHz, a single engine can support a throughput rate of 800 MB/s.

Regular expression match. The algorithm for regular expression matching operations [5] uses a novel pipelining strategy that defers state-dependent logic to the last stage, enabling single-cycle state transitions (Figure 3). A regular expression compiler is used to encode contiguous strings of *m* input characters and compress the transition table through indirection.

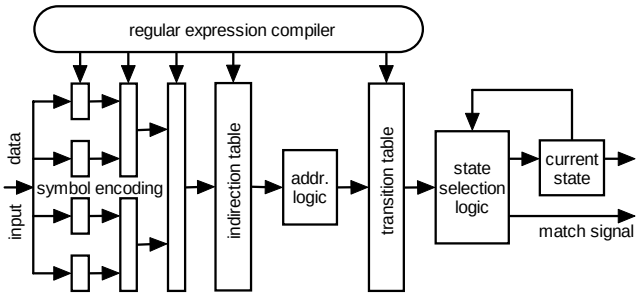

Figure 3: Regular expression search engine

The regular expression search engine supports 50 regular expressions and has an ingest capability of 4 characters per clock. Running at 100 MHz, the throughput is 400 MB/s.

Combining operations. While each of the above search engines has a distinct architecture and associated FPGA configuration, upon a keyword match each returns both the match and match position. Software on the processor is then used to resolve the combining operations including the Boolean operators AND, OR, and NOT as well as proximity operators NEAR and ANDTHEN. The operators AND, OR, and NOT perform their traditional Boolean logic functions at the file level. The operator NEAR is equivalent to AND with the additional constraint that the matching keywords must be within a given distance of one another in the file. The operator ANDTHEN is equivalent to NEAR with the additional constraint that the first keyword must occur earlier in the file than the second keyword.

By way of illustration [2] the query:

```
((Bush NEAR[200] Baseball) AND
 (Blair NEAR[200] Soccer))
```

expresses the following conditions: (1) the string "Bush" is found within 200 characters of the string "Baseball"; (2) the string "Blair" is found within 200 characters of the string "Soccer"; and (3) both conditions (1) AND (2) hold.

FPGA Reconfiguration

Each time that a search is invoked by the user, the type of search requested is determined from the query: exact, approximate, or regular expression. At this point, the appropriate configuration for the FPGA is loaded from the configuration store. This is done in parallel with the initial data set read operations (directory lookup, file open, initial data queuing, etc.) that also operate at millisecond time scales. In this way a set of applications can be prepositioned to quickly begin executing on the FPGA as they are needed and the constraints associated with limited FPGA capacity are overcome.

Conclusions

We describe the use of FPGA reconfiguration to support application requirements greater than the traditionally exploited revision update function. Based upon the parameters specified to a text search application, different configurations are loaded into the FPGA and executed, actually reconfiguring the reconfigurable logic on a regular basis and thus providing support for a set of functions that can quickly initiate execution.

References

[1] R. D. Chamberlain, R. K. Cytron, M. A. Franklin, and R. S. Indeck, "The *Mercury* System: Exploiting Truly Fast Hardware for Data Search," in *Proc. of Int'l Workshop on Storage Network Architecture and Parallel I/Os*, pp. 65-72, September 2003.

[2] M. A. Franklin, R. D. Chamberlain, M. Henrichs, B. Shands, and J. White, "An Architecture for Fast Processing of Large Unstructured Data Sets," in *Proc. of 22nd Int'l Conf. on Computer Design*, pp. 280-287, October 2004.

[3] R. M. Karp and M. O. Rabin, "Efficient randomized pattern-matching algorithms," *IBM Journal of Research and Development* **31**(2):249-260, March 1987.

[4] Q. Zhang, R. D. Chamberlain, R. S. Indeck, B. West, and J. White, "Massively Parallel Data Mining Using Reconfigurable Hardware: Approximate String Matching," in *Proc. of Workshop on Massively Parallel Processing*, April 2004.

[5] B. C. Brodie, R. K. Cytron, and D. E. Taylor, "A Scalable Architecture For High-Throughput Regular-Expression Pattern Matching," in *Proc. of 33rd Int'l Symp. on Computer Architecture*, June 2006.