Gene Matching Using JBits

Steven A. Guccione and Eric Keller

Xilinx, Inc., 2100 Logic Drive, San Jose, CA 95124 (USA)
{Steven.Guccione,Eric.Keller}@xilinx.com

Abstract. As the emerging field of bioinformatics continues to expand, the ability to rapidly search large databases of genetic information is becoming increasingly important. Databases containing billions of data elements are routinely compared and searched for matching and near-matching patterns. In this paper we explore the use of run-time reconfiguration using field programmable gate arrays (FPGAs) to provide a compact, high-performance matching solution to accelerate the searching of these genetic databases. This implementation provides approximately an order of magnitude increase in performance while reducing hardware complexity by as much as three orders of magnitude when compared to existing commercial systems.

1 Introduction

One of the fundamental operations in computing is string matching. Here two linear arrays of characters are compared to determine their similarity. This operation can be found across a wide range of algorithms and applications. One area where string matching has recently received a renewed interest is in the area of bioinformatics, in particular in the area of searching genetic databases.

With the initiation of the Human Genome Project [2] in the early 1990s, the amount of data to be searched, as well as the number of searches being performed on this data has continued to increase. Because of the size and ongoing growth of this problem, specialized systems have been commercially introduced to search these databases of genetic information.

In this paper we present a system used to implement one of the most popular genetic search algorithms, the Smith-Watermann algorithm, using run-time reconfiguration. This approach provides not only smaller, faster circuits, but also reduces the input-output requirements of the system while simplifying hardware / software interfaces.

2 The Smith-Watermann Algorithm

While many applications performing string matching look for an exact match to the searched data, many other applications are interested in finding approximate matches. This requires a somewhat more complex algorithm than the search for exact matches. In the early 1970s, at least nine independent discoveries of a dynamic programming algorithm for searching for inexact matches when comparing strings were published. This algorithm has found use in fields as diverse as speech and image processing, cryptography, text processing, and artificial intelligence [14].

One area where inexact string matching has become important is in the searching genetic databases [13]. The optimal algorithm for inexact search in the field of bioinformatics is typically known as *Smith-Watermann* and uses a dynamic programming technique. The algorithm compares two strings S and T by performing a pairwise comparison of each element in the two strings, then computing a score to determine the similarity of the two strings. Figure 1 gives a two-dimensional representation of the algorithm. The two strings S and T are compared and intermediate values a, b and c are used to produce the intermediate result, d. This calculation is repeated once for each pairwise element comparison.

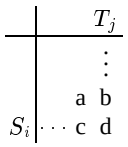

Fig. 1. Pairwise comparisons in the Smith-Watermann matching algorithm.

The matching algorithm itself is given in Figure 2. If the elements being compared in the two strings are the same, the value a used to calculate the result value d. If the elements in the two strings are not the same, then the value of a plus some *substitution* penalty is used. The result value d is determined by taking the minimim of this value, the value of b plus some *insertion* penalty and the value of c plus some *deletion* penalty.

$$d = min \begin{cases} \begin{cases} a & \text{if } S_i = T_j \\ a + sub & \text{if } S_i \neq T_j \end{cases} \\ b + ins \\ c + del \end{cases} \tag{1}$$

Fig. 2. The Smith-Watermann matching algorithm.

In the case where string S is of length m and string T is of length n, the algorithm begins by comparing S_0 and T_0 and proceeds onward until a final value of d is calculated at the comparison of S_m and T_n. This value of d is the *edit distance* between the two strings.

Because of the pairwise comparison between each element in string S with each element in string T, the algorithm has a computational of complexity of $O(mn)$. Many other matching algorithms popularly used in searching genetic databases including BLAST, FASTA, Needleman-Wunsch and others have been published and implemented to provide fast searching of genetic databases. While these algorithms provide high performance, they are all sub-optimal and in some way compromise the quality of the result. For this reason, Smith-Watermann is the preferred algorithm, but performance

issues often result in the use of one of these sub-optimal algorithms in its place. A good overview of the various searh algorithms is avalaible on-line at the the Paracel WWW site [4].

3 FPGA Implementations

One of the limitations in performance inherent in the Smith-Waternamm algorithm is that it does not completely parallelize. Because results from S_{i-1} and T_{j-1} are used to compute the value at (S_i, T_j), the computation proceeds serially across both the i and j axes. These data dependencies, however, permit some level of parallelization. If the nxm comparisons performed in the algorithm is viewed as a two dimensional array, then the algorithm can be seen as proceeding from the upper left corner of the array, where S_0 is compared to T_0, downward and to the right until the final value of d is computed using the comparison of S_n and T_m. The data dependencies indicate that calculations may proceed in parallel across diagonals of this array. In addition, all communication of results are local, with data being passed to neighbors in the array.

This structure makes the problem amenable to the use of a systolic array, as described by Kung [9]. In this approach, data is pumped through an array of simple, identical processors, with results being produced on each clock cycle. An early implementation of the Smith-Waternamm algorithm was done using custom VLSI by Lipton and Lopresti [10] [11] [12]. It was this approach that was used by Houang in his work with the *Splash 2* reconfigurable logic based processor [8] [6].

It was the outstanding results of Houang with the *Splash 2* system which led us to re-visit this approach. While the interest in searching genomic databases has increased over the last decade, so has the speed and density of reconfigurable logic technology. While a simple porting of the original *Splash 2* design to more modern FPGA devices would provide an excellent demonstration of the advancements in FPGA hardware technology, it was also desirable to explore some of the advances in FPGA software technology, in particular the use of run-time reconfiguration.

4 The JBits Implementations

Rather than using the standard VHDL design flow to implement the Smith-Watermann algorithm, the Xilinx *JBits* toolkit was used [7]. The *JBits* toolkit is a set of Java tools and APIs that permit direct impementation and reconfiguration of circuits for the Xilinx Virtex family of FPGAs. *JBits* was particularly useful in the implementation of this algorithm because there were several opportunities to take advantage of run-time circuit customization. In addition, the systolic approach to the computation permitted a single parameterizable core representing the processing element to be designed, then replicated as many times as necessary to implement the fully parallel array.

The logic implementation of the algorithm is shown in Figure 3. Each gray box represents a LUT / flip-flop pair. This circuit demonstrates four different opportunities for run-time circuit customization. Three of these are the folding of the constants for the insertion, deletion and substitution penalties into the LUTs. Rather than explicitly feeding a constant into an adder circuit, the constant can be embedded in the circuit,

resulting in (in effect) a customized constant adder circuit. Note that these constants can be set at run time and may be parameters to the circuit.

The fourth run-time optimization is the folding of the match elements into the circuit. In genomic databases, a four character alphabet is used to represent the four bases in the DNA molecule. These characters are typically denoted *A* for adenine, *T* for thymine, *G* for guanine and *C* for cytosine. In this circuit, each character can be encoded with two bits. The circuit used to match S_i and T_j does not require that both strings be stored as data elements. In this implementation, the S string is folded into the circuit as a run-time customization. Note that unlike the previous optimizations, the string values are not fixed constants and will vary from one run to another. This means that the entire string S is used as a run-time parameter to produce the customized circuit.

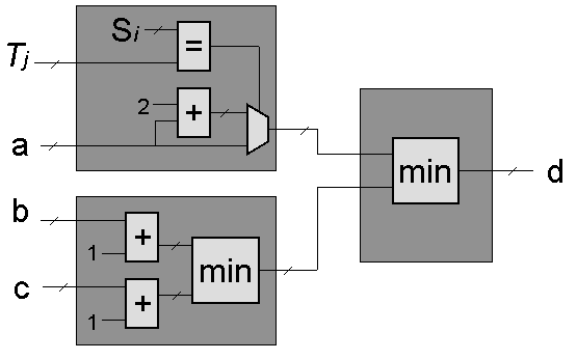

Fig. 3. The combinational logic of the Smith-Watermann circuit.

The basic combinational logic of the algorithm must now be combined with memory elements to produce the systolic processing element. Figure 4 shows the combinational logic combined with flip-flop memory elements used to produce the systolic processing element.

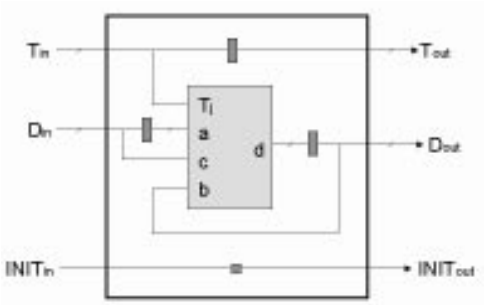

Fig. 4. The Processing Element circuit.

This design uses a feature of the algorithm first noted by Lipton and Lopresti [10]. For the commonly used constants, 1 for insert/delete and 2 for substitution, b and c can only differ from a by +1 or -1, and d can only differ from a by either 0 or 2. Because of this modulo 4 encoding can be used, thus requiring only 2 bits to represent each value. The final output edit distance is calculated by using an up-down counter at the end of the systolic array. For each step, the counter decrements if the previous output value is one less than the current one and it increments otherwise. The up-down counter is initialized to the match string length which makes zero the minimum value for a perfect match. Figure 5 give the circuit diagram of the complete systolic array.

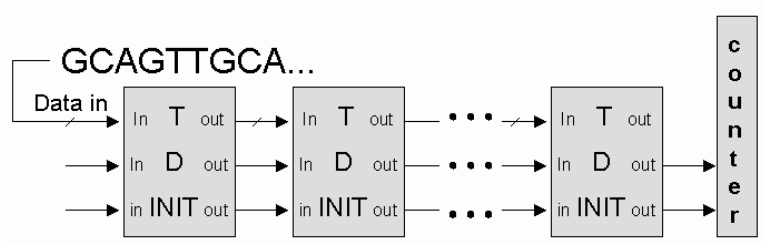

Fig. 5. The complete systolic array circuit.

Further optimizations were performed on the circuit to more efficiently map the design to the Virtex architecture. Shown in Figure 6 is the optimized circuit efficiently mapped to the Virtex architecture. These optimizations make use of the Virtex carry chain, which reduced the delay of the circuit since general routing was not needed. The optimization is evident in the equation in Figure 7 which is equivalent to the one in Figure 2 . The equation is basically a wide or gate which is efficiently implementable with the Virtex carry-chain. Another optimization evident from the transformed equation is the fact that d is equal to a or $a + 2$. Because of this the least significant bits of a and d are always equal. Therefore, only 1 bit is needed to represent d.

5 Comparison with Splash 2 Implementation

The advances in semiconductor technology over the past decade make comparisons between this implementation and that of the *Splash 2* system interesting. In the *Splash 2* system, as many as 16 boards each containing 17 Xilinx XC4010 FPGAs comprised the system. By counting only LUT / flip-flop pairs, a system similar in capabilities to the fully expaned *Splash 2* could be implemented using approximately ten Virtex XCV1000 devices. Similarly, approximately four Virtex 2 XC2V6000 devices could also implement such a system. In addition, clock speeds in FPGAs have increased dramatically over the last decade.

It is also interesting to isolate the advantages due to the use of run-time reconfiguration. The original *Splash 2* design was implemented in VHDL and produced a circuit using approximately 33 LUT / flip-flop pairs per processing element. By comparison,

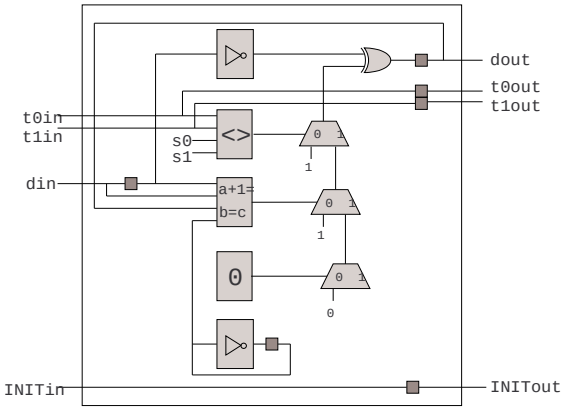

Fig. 6. The optimized logic of the Smith-Watermann circuit.

$$d = \begin{cases} a & \text{if } b \text{ or } c \text{ equals } a-1 \text{ or } S_i = T_i \\ a+2 & \text{if } b \text{ and } c \text{ equal } a+1 \text{ and } S_i \neq T_i \end{cases} \tag{2}$$

Fig. 7. The transformed Smith-Watermann matching algorithm with a insert cost of 1, delete cost of 1, and substitute cost of 2.

the *JBits* circuit uses only six LUT / flip-flop pairs, a savings of approximately a factor of 5.5. Much of this savings comes directly from the ability to fold constants and variables into the circuit at run-time.

Clearly, some of the optimizations involving fixed constants such as the insert, delete and substitution penalties could, and probably were, folded into the circuit in the original VHDL design. These types of optimizations are commonly performed by standard static design tools and do not specifically require run-time reconfiguration. The ability to parameterize the circuit based on the string data, however, would not typically be performed by standard design tools. Customizing a circuit in this manner would require that the design tools be re-run for each string being matched. This would be impractical in most cases. Circuit customization at run-time, however, makes this an attractive option.

In addition, folding the string data into the circuit results in further savings. By making the string data external to the circuit, two flip-flops per processing element must be used to shift in and store this string data. This alone results in a 30 per cent increase in circuit size. In addition, because the match string must be loaded into the circuit as data, other control lines and circuitry must be used to manage this process. Finally, time must be spent loading this data separately from the circuit configuration. When the data is folded into the circuit through run-time customization, it is loaded as part of the circuit and incurs no additional overhead. Loading the data may substantially increase the time required to perform matching in large arrays.

6 Other Current Implementations

As the computing demmands of bioinformatics has continued to increase, commercially available solutions to the problem of searching genetic databases have become available. Today the three major systems used commercially all take different approaches. It should also be noted that these systems all support a variety of matching algorithms in addition to Smith-Watermann. Table 1 gives a comparison of the various technologies currently available to perform Smith-Watermann matching. For a historical comparison, the *Splash 2* work of Houang has also been included.

Table 1. This displays both performance and hardware size for various implementations.

	Processors per Device	Devices	Updates per sec
Celera (Alpha cluster)	1	800	250B
Paracel (ASIC)	192	144	276B
TimeLogic (FPGA)	6	160	50B
Splash 2 (XC4010)	14	272	43B
JBits (XCV1000-6)	4,000	1	757B
JBits (XC2V6000-5)	11,000	1	3,225B

The first system listed in Table 1 is from Celera Genomics, Inc. Celera Genomics is a commercial company owned by Applera, Inc. which completely sequenced the human genome in June 2000 [1]. This work was done in parallel (some say in competition) with the publically funded Human Genome Program. Celera uses an 800 node Compaq Alpha cluster for their database searches. This arrangement is able to perform approximately 250 billion comparisons per second. The major advantage of such a multiprocessor system is its flexibility. The drawback, however, is the large cost associated with purchasing and maintaining such a large server farm.

The second system listed in the table is made by Paracel, Inc. Paracel takes a custom ASIC approach to the matching problem [3]. Their system uses 144 identical custom ASIC devices, each containing approximately 192 processing elements. This produces 276 billion comparisons per second, which is comparable to Celera's server farm approach, but using significantly less hardware. Interestingly, Paracel was bought in June 2000 by Applera, Inc., the company which also owns Celera Genomics.

TimeLogic, Inc. also offers a commercial system but uses FPGAs and describes their system as using "reconfigurable computing" technology [5]. They currently have six processing elements per FPGA device and support 160 devices in a system. This system performs approximately 50 billion matches per second. This is significantly lower in performance than the Celera Genomics or Paracel systems, but the use of FPGAs results in a more flexible system which does not incur the overheads of producing a custom ASIC.

For historical comparisons the *Splash 2* system is included in this table. Although the results are nearly a decade old, the fully loaded *Splash 2* system contains 272 FP-

GAs, each supplying 14 processing elements, producing a match rate of 43 billion matches per second. These are surprisingly respectable numbers for ten year old technology in a rapidly changing field.

Finally, the *JBits* implementations using a Xilinx XCV1000 Virtex device implements 4,000 processing elements in a single device running at 188 MHz in the fully optimized version. This results in over 750 billion matches per second. And if the newer Virtex 2 family is used, a single XC2V6000 device can be used to implement approximately 11,000 processing elements. At a clock speed of over 280 MHz, this give a matching rate of over 3.2 *trillion* elements per second.

7 Conclusions

A gene matching system using run-time reconfiguration and operating on a single FPGA device has been presented. This system is able to perform Smith-Watermann matching at a rate of over three billion matches per second. This compares favorably to the currently available systems used commercially in this field. In the area of performance, the run-time reconfiguration approach provides an order of magnitude increase over both custom ASIC and multiprocessor systems, while reducing the hardware complexity by two to three orders of magnitude.

Such results would often indicate that some system parameter, usually flexibility, has been lost. This, however, is not necessarily true. It is possible to use similar techniques to implement other matching algorithms other than Smith-Watermann using run-time reconfiguration. Interestingly, there may be little or no advantage to implementing sub-optimal matching algorithms using this approach. Because this implementation appears to be limited more by data input / output than by processing power, implementing a faster algorithm may not provide substantial increases in performance. This would make the sub-optimal algorithms much less desirable.

As the field of bioinformatics continues to grow, and various fields from drug design to law enforcement come to rely on this technology, it is expected that interest in high performance matching systems will continue to grow. Reconfigurable logic and run-time reconfiguration promise to permit faster, less expensive systems to be produced to meet these needs.

References

1. Celera Genomics, Inc. World Wide Web site http://www.celera.com/, 2002.
2. The Human Genome Project Information. World Wide Web site http://www.ornl.gov/hgmis/, 2002.
3. Paracel, Inc. World Wide Web site http://www.paracel.com/, 2002.
4. Paracel, Inc. Frequently Asked Questions (FAQ). World Wide Web page http://www.paracel.com/faq/, 2002.
5. TimeLogic, Inc. World Wide Web site http://www.timelogic.com/, 2002.
6. Maya Gokhale, William Holmes, Andrew Kosper, Sara Lucas, Ronald Minnich, and Douglas Sweely. Building and using a highly parallel programmable logic array. *IEEE Computer*, pages 81–89, January 1991.

7. Steven A. Guccione, Delon Levi, and Prasanna Sundararajan. JBits: A java-based interface for reconfigurable computing. In Richard Katz, editor, *Second Annual Military and Aerospace Applications of Programmable Devices and Technologies Conference (MAPLD)*, September 1999.
8. Dzung T. Hoang. Searching genetic databases on splash 2. In Duncan A. Buell and Kenneth L. Pocek, editors, *IEEE Workshop on FPGAs for Custom Computing Machines*, pages 185–191, Los Alamitos, CA, April 1993. IEEE Computer Society Press.
9. H. T. Kung. Why systolic architectures? *IEEE Computer*, 15(1):37–46, January 1982.
10. Richard Lipton and Daniel Lopresti. A systolic array for rapid string comparison. In Henry Fuchs, editor, *1985 Chapel Hill Conference on Very Large Scale Integration*, pages 363–376. Computer Science Press, 1985.
11. Richard Lipton and Daniel Lopresti. Comparing long strings on a short systolic array. In Will Moore, Andrew McCabe, and Roddy Urquhart, editors, *Systolic Arrays*, pages 181–190. Adam Hilger, 1986.
12. Daniel P. Lopresti. P-NAC: A systolic array for comparing nucleic acid sequences. *IEEE Computer*, pages 98–99, July 1987.
13. David Sankoff and Joseph B. Kruskal, editors. *Time Warps, String Edits, and Macromolecules: The Theory and Practice of Sequence Comparison*. Addison-Wesley Publishing Company, Inc., Reading, Massachusetts, 1983.
14. Robert A. Wagner and Michael J. Fischer. The string-to-string correction problem. *Journal of the ACM*, 21(1):168–173, 1974.