Journal of Circuits, Systems, and Computers
Vol. 16, No. 4 (2007) 527–540
© World Scientific Publishing Company

A CONFIGWARE APPROACH FOR HIGH-SPEED PARALLEL ANALYSIS OF GENOMIC DATA

HEITOR S. LOPES*, CARLOS R. ERIG LIMA† and NORTON J. MURATA

Bioinformatics Laboratory/CPGEI,
Federal University of Technology – Paraná,
Av. 7 de setembro 3165, 80230-901 Curitiba (PR), Brazil
**hslopes@pesquisador.cnpq.br*
†erig@utfpr.edu.br

Revised 9 April 2007

Many problems in bioinformatics represent great computational challenges due to the huge amount of biological data to be analyzed. Reconfigurable systems can offer custom-computing machines, with orders of magnitude faster than regular software, running in general-purpose processors. We present a methodology for using a configware system in an interesting problem of molecular biology: the splice junction detection in eukaryote genes. Decision trees were developed using a benchmark of DNA sequences. They were converted into logical equations, simplified, and submitted to a Boolean minimization. The resulting circuit was implemented in reconfigurable parallel hardware and evaluated with a five-fold cross-validation procedure, run in a second level of parallelism. The average accuracy achieved was 90.41% and it takes 18 ns to process each data record with 60 nucleotides.

Keywords: FPGA; DNA; pattern recognition; decision tree.

1. Introduction

Thanks to the several genome-sequencing projects in the world, a huge amount of biological data has been accumulated. Several problems in bioinformatics are related to the analysis of such data. However, to deal with such "information explosion", efficient computational resources are necessary, not only to store data, but also, to process and analyze them.[1,2]

Recent advances in computational techniques have allowed the analysis of genomic data more efficiently, but at the expense of hard computational demand. Field programmable gate arrays (FPGAs) chips have become popular in recent years. Due to its availability and performance, they have been used in powerful reconfigurable systems. Systems based on reconfigurable hardware (or simply, configware) can offer custom-computing machines for specific applications, with orders of magnitude faster than regular software running in general-purpose processors.[3,4]

Usually, biological sequence analysis involves a large amount of data (for instance, the whole human genome has more than three billion nucleotides), and so, efficient computational resources are necessary to accelerate its analysis, preferably, exploring parallelism.[5,6] Hence, the main motivation of this work is to explore the flexibility and performance of reconfigurable computing to a hard computational problem posed by the analysis of DNA sequences.

More specifically, this work aims at developing a methodology for the application of a configware system (see next section) in a specific problem of molecular biology: the splice junction detection in eukaryote genes.[7,8] Using a data-mining approach, we created a data classifier based on a decision tree, which was further transformed into a logic circuit and implemented in a reconfigurable device. This system is capable of analyzing large amounts of DNA data and identifying splice junctions with a very high throughoutput, thanks to multiple levels of parallelism.

1.1. *Reconfigurable computation*

Hardware reconfigurable systems are associated with the emergence of programmable logic devices (PLDs) in the 1990s, breaking the balance point between flexibility and performance. They can achieve high performance with low implementation cost. Also, they override the bottleneck of Von Neumann's machines implemented with ordinary microprocessors, allowing massive low-level parallelism. Reconfigurable hardware is programmable by reconfiguration of its structure — a midterm between hardware and software approaches. An algorithm structurally programmed in reconfigurable hardware is also known as configware.[3] Reconfigurable computation allies the performance of hardware-based solutions and the flexibility of software-based solutions, allowing the exploration of the inherent parallelism of some computational tasks.

The synthesis of logical circuits in PLDs is done using computer-aided design systems, allowing the simultaneous use of different project interfaces. Examples of interfaces languages used are graphical (using schematics), VHSIC hardware description language (VHDL), and Altera hardware description language (AHDL). VHDL has become the standard language for hardware description.[9] Currently, there are several commercial computer-aided design platforms for developing reconfigurable hardware systems. In general, they provide an integrated development environment, allowing project, simulation, test, and documentation of digital circuits.

The objective of the reconfigurable hardware concept is to enable an easy and quick adaptation of a project to the continued technological evolution, aiming improved portability and interchange ability of the final system. By means of dividing the structure into small functional blocks, with very specific dedicated interfaces, the modularization of the project becomes efficient. As a consequence, the management and integration of multidisciplinary design team is facilitated, as well as the adaptation of a particular block to keep pace with the evolution of the technology. Some advantages of using reconfigurable hardware are[10]:

- Real parallelism, without following Von Newmann's model;
- Modular and hierarchical development of a project;
- Project cycle timing reductions, allowing top–down and bottom–up project methodologies;
- Availability of several development interfaces and environments;
- Availability of ready-to-use tested functions (IP-core), reducing the project cycle for high-complexity functions.

An additional motivation for the use of reconfigurable hardware in the implementation of algorithms is the wide availability of high-performance devices in the market. For instance, some of the most recent FPGAs present characteristics such as above 700 I/O pins, controlled impedance and dedicated lines for the operation in differential mode, special internal modules (multipliers, pulse width modulation (PWM), dedicated registers for high-performance operations), and high-speed RAM storage capacity (up to 2 Mbits). Currently, commercial devices with up to five million of logical cells can be found, allowing the implementation of one or more full processors with memory and peripherals in a single chip.

1.2. *Decision trees for data classification*

A decision tree is regarded as a predictive model capable of mapping the observation of the attributes of an input pattern to conclude about it. A decision tree is represented by a graph, or more exactly, a tree in which each internal node corresponds to the test of the value of a given attribute (categorical or numeric); vertices represent the values of a given attribute; and leaf nodes are the predicted class of the mapping. Hence, the path from the root to a given leaf is a conjunctive rule that classifies an input pattern into a predefined class.

Decision trees are widely used in data mining because they have some advantages over other classification methods, such as:

- They are simple and easily understood.
- They represent low computational cost even for the analysis of large data sets.
- Usually, input data need little or no preprocessing.
- Both numeric and categorical type of attributes can be easily handled.
- An explanation rule is easily drawn for a given input pattern once classified.

Each node can test one (univariate) or more attributes (multivariate). The test of a node can have two or more outcomes. For the first case, the tree is known as binary tree. The overall number of classes can also be two or more. A particular type of decision tree is the univariate Boolean decision tree which can be described by a set of disjunctive normal form (DNF) rules.

A decision tree can be trained by splitting the training set into subsets based on an attribute value test. A recursive procedure is applied to each derived subset, and it is completed when splitting is either nonfeasible, or a singular classification can be applied to each element of the derived subset.[11]

The induction of a decision tree is based on supervised learning. Therefore, like other data classification methods, if the training set is not sufficiently large, there will be overfitting and the generalization performance will be poor. One of the most used technique for validating the performance of a classifier (not only decision trees) is by using cross-validation. In this technique, first, the training set is divided into k mutually exclusive subsets of equal size. Then, for each subset i, training is performed by using the union of all remaining subsets. The performance is then evaluated, usually by computing the error rate. The same procedure is repeated for each subset and the overall performance is the average of the individual performances on the subsets.[12]

1.3. *Detecting splice junctions in eukaryotes*

In the DNA of eukaryotes (organisms having a membrane surrounding the nucleus of their cells), there are many pieces of sequences that are not expressed in an amino acid chain of a protein. The DNA sequences whose complements are not present in the final messenger-RNA product are called introns ("int" for intervening sequences), and those retained and expressed are called exons ("ex" for expressed sequences). Introns are removed as a result of an excision and rejoining process referred to as splicing, taken place inside of the cell.[7] The exact boundary between an intron and an exon (and vice-versa) is called a splice junction or splicing site.

Biological knowledge allows establishing some rules to determine intron–exon (IE) and exon–intron (EI) boundaries.[8] Basically, these rules specify some nucleotide sequences that can be expected in both sides of a boundary. That is, somewhere at the end of an intron and at the beginning of an exon (for IE) and somewhere at the end of an exon and at the beginning of an intron (for EI).[8] However, as a consequence of the large biological variability of living beings, these rules are not exact and relevant errors can occur. Therefore, many algorithms and methods have been proposed for finding splice junctions, mainly in the context of machine-learning.[11,13] Some of them have succeed to achieve reasonable accuracies for this problem, but at the expense of high computational cost. We will show in this work that a high accuracy is possible, attainable with an equally high processing speed, by implementing a trained parallel classifier into a configware system.

For simplicity, the problem can be divided in two independent subproblems: given a DNA sequence, recognize exon/intron boundaries (EI sites or "donors"), and recognize intron/exon boundaries (IE sites or "acceptors").

2. Some Related Work

In the recent literature, reconfigurable computation is a methodology that has been sparsely explored in molecular biology applications.[2] In most cases, it is used for achieving high-performance computing, not attainable by regular computers.

The similarity between two protein/DNA sequences by means of a dynamic programming algorithm is explored in Ref. 14. This work describes the implementation of the Needleman–Wunsh algorithm on a giant bio-inspired computational tissue made of 3200 FPGAs, denominated BioWall. This approach presents two main drawbacks: the practical reproduction of the described prototype is impossible and no performance or comparison results are present in the paper. The work of Luethy and Hoover[2] describes several complementary computing strategies available to perform biological sequence analysis, including hardware acceleration based on FPGA. Although no comparative analysis is presented in this work, it can be useful for describing other techniques for sequence analysis computing.

Oliver *et al.*[15] presented a new approach to compute multiple sequence alignments using a FPGA-based accelerator board for a desktop computer. Its speedup was around 12 times faster when compared to a software-only version. Yamaguchi *et al.*[16] present a hardware acceleration approach to compute the Smith–Waterman algorithm comparison between query sequences and database sequences. In this paper, the dynamic programming algorithm is also computed in parallel using FPGAs. In addition, Marongiu *et al.*[6] present a parallel hardware generator for the design and prototyping of dedicated systems to the analysis of biological sequences. More recently, Armstrong *et al.*[17] relate the preliminary steps toward using a reconfigurable system for protein folding using a simple lattice model.

3. Methodology

3.1. *Database*

The database used in this work is available at the UCI Machine Learning Repository[18] and was build by Towell,[19] and later used in other works.[11,13] The database was first used in Towell's PhD. thesis to evaluate hybrid machine-learning algorithms (based on neural networks) that uses examples to inductively refine preexisting knowledge.

The database is composed by 3190 instances, each of them with a 60 nucleotide-long sequence. Nucleotides are either Adenine, Cytosine, Guanine, or Thymine, and represented by letters A, C, G, or T, respectively. All genomic data were extracted from GenBank,[20] using genes of primates. Each sequence was previously classified into one of the three classes: IE, EI, or Neither, corresponding to an IE boundary, an EI boundary, or neither of them, respectively.

In his work, Towell used a 10-fold cross-validation methodology[12] on 1000 examples randomly selected from the complete data set, and obtained classification error rates in the ranges 5.74–16.32% and 7.55–17.41%, for EI and IE classes, respectively.[19] Using the rules derived from biological knowledge,[8] only a poor classification rate can be achieved (95.8% of class Neither, 40% of class IE, and only 3% of class EI). Therefore, many machine-learning algorithms were proposed to solve this problem. A comprehensive comparison of the performance of 23 different classification methods using this database can be found in Ref. 11.

```
U30          . . .           U1|D1          . . .            D30  Class
CCAGCTGCATCACAGGAGGCCAGCGAGCAG|GTCTGTTCCAAGGGCCTTCGAGCCAGTCTG  EI
TTCAGCGGCCTCAGCCTGCCTGTCTCCCAG|GTCTCTGTCCTTCCACCATGGCCCTGTGGA  IE
CAAAAGAACAAAGCTGGAGGCATCACGCTA|CCTGACTTCAAACTATACTACAAGGCTACA   N
```

Fig. 1. Three examples of records from the splice-junction database.

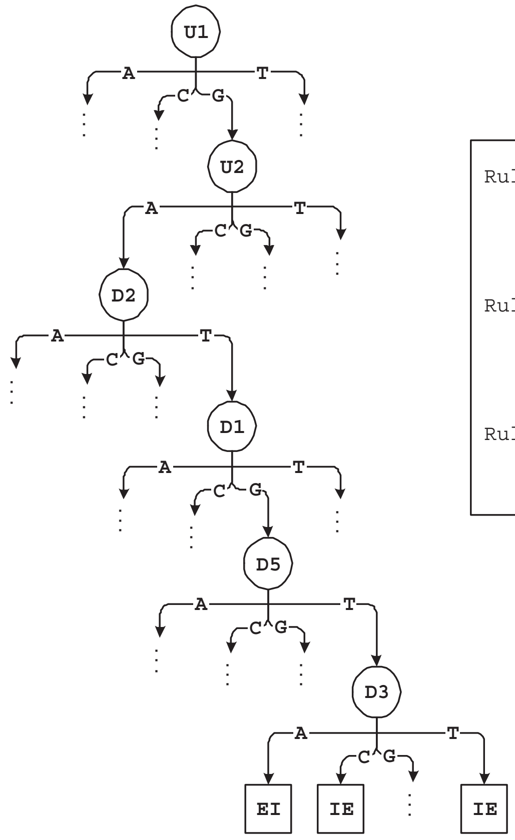
Originally, the database had 768 instances for IE, 767 for EI, and 1655 for Neither. When the exact nucleotide is not known, other codes are used to represent this ambiguity. The amount of these codes is not significant in the data set used, representing only 0.027% of the total nucleotides. Therefore, a total of 15 instances were excluded from the original database due to the presence of nonstandard codes.

Using a database terminology, each of the 3175 records has 61 attributes: the first 60 are prediction attributes (nucleotides), and the last, the goal attribute (class). Prediction attributes are named as $U30, U29, \ldots, U2, U1$, $D1, D2, \ldots, D30$, where U and D mean, respectively, up and downstream nucleotides relative to the central (splicing) point. That is, in the records of this database, the exact boundary between intron–exon or exon–intron, when existing, is found between the 30th and 31st nucleotide of the sequence. Figure 1 shows three examples drawn from the database, one for each class.

3.2. *Decision tree and classification rules*

Using the database mentioned above, a classifier was constructed. Since this is a partitioning problem, it was suggested that a decision-tree would be suitable.[11] Therefore, we built a decision tree using the well-known C4.5[21] induction algorithm, available in the software Weka,[12] version 3.4. A standard five-fold cross-validation procedure was done: the database was divided into five mutually exclusive partitions, preserving class proportionality. A classifier was created using four of these partitions and tested with the remaining. Results are computed as the average of the five possible combinations of training/testing partitions. In the current literature, there is no clear consensus about the number of folds for cross-validation procedures. For software-based implementations, 10-fold cross-validations are more frequently used, although it is computationally expensive when compared with a five-fold computation. In this work, we prefer the five-fold alternative to save limited resources of the FPGA, since every evaluation is done in parallel. Saving memory and logic resources of the physical device is important to enable further levels of parallelization, as explained later.

The average size of the obtained decision trees were 8-nodes depth and 164 leaf nodes. Figure 2 shows a partial branch of a tree. The top node tests $U1$ position, the remaining nodes shown are for the following positions to be tested, until reaching the leaf nodes. Leaf nodes of the tree correspond to the predicted class when all



```
Rule 1: if U1=G and U2=A and D2=T
        and D1=G and D5=T and D3=T
             then class=IE

Rule 2: if U1=G and U2=A and D2=T
        and D1=G and D5=T and D3=C
             then class=IE

Rule 3: if U1=G and U2=A and D2=T
        and D1=G and D5=T and D3=A
             then class=EI
```

Fig. 2. Partial branches of a decision tree obtained with the C4.5 algorithm.

conditions of the inner nodes are met, that is, the conjunction of the antecedents of the rule. For instance, the three leaf nodes of Fig. 2 are represented by the conjunctive rules on the right-hand side of the figure.

Decision trees encompass some redundancy, given the number of leaf nodes in comparison with the number of classes (IE, EI, and Neither). This redundancy suggests the possibility of simplifying the tree using some technique to minimize the set of logical expressions obtained. This issue will be explored in the next section.

By default, the C4.5 algorithm does not prune the generated tree. Consequently, a full tree is presented to the user for the sake of completeness. The inspection of the induced decision trees leads to a straightforward simplification, simply by excluding all branches that do not classify any instance. A deeper analysis of the pruned trees also revealed that not all 60 attributes (nucleotide positions up and downstream) are relevant for this classification task. In fact, 16.6 attributes, in average, were present in the five trees created during the cross-validation procedure.

3.3. *Boolean minimization*

After obtaining a pruned tree, the next step is to convert attributes to a binary code and build a truth table, relating output to inputs. Then, we apply a Boolean minimization method in order to find the most economical logic network capable of describing the input/output relationships. Considering that, a network of logic gates will be implemented in the FPGA, this procedure aims at finding the network with minimal number of gates. In our specific case, we will obtain simple Boolean rules for two (independent) classes, EI and IE. Class Neither is the default class when an instance is not classified as EI or IE. Since there are four different nucleotides (A, C, G, and T), two binary digits are necessary to represent them. In the same way, there are three classes, and, again, two binary digits are used. For the inputs, the following convention was used: A = 00, C = 01, G = 10, and T = 11, and for the outputs, EI = 01, IE = 10, and Neither = 00. Output code 11 is undefined. Table 1 shows three examples of the encoding procedure: the value of the relevant attributes (nucleotides up and downstream and class) and the corresponding binary encoding. The examples correspond to the conjunctive rules of the branches down to the leaf nodes in Fig. 2. Notice that, in this table, "don't-care" positions (X) were not considered.

For the minimization, we used Boolean minimizer.[22] This software is based on the "Espresso" algorithm,[23] and optimizes Boolean expressions by determining the minimal representation of functions with multiple inputs and binary output. This approach is similar to the well-known Quine–McCluskey algorithm. An important characteristic of Espresso is that it is a fast technique for the detection and elimination of prime implicants, as well as for the generation of a reduced form of the prime implicant table. This algorithm uses a branch-and-bound technique for solving the minimum cover problem arising from the Boolean minimization.

After the Boolean minimization of the truth-table, two logical expressions are obtained: first one for EI and the other for IE, both in DNF. The process is repeated for each tree generated in the cross-validation procedure, previously mentioned.

3.4. *Hardware implementation*

The physical synthesis of the logic circuits was implemented in a reconfigurable device from Altera,[24] more precisely, a Stratix-II EP2S60, using VHDL in the

Table 1. Three examples of encoded Boolean expressions derived from the decision trees.

...	$U5$	$U4$	$U3$	$U2$	$U1$	$D1$	$D2$	$D3$	$D4$	$D5$	...	Class
X	X	X	A	A	G	G	T	T	X	T	X	IE
—	—	—	00	00	10	10	11	11	—	11	—	10
X	X	X	A	A	G	G	T	C	X	T	X	IE
—	—	—	00	00	10	10	11	01	—	11	—	10
X	X	X	A	A	G	G	T	A	X	T	X	EI
—	—	—	00	00	10	10	11	00	—	11	—	01

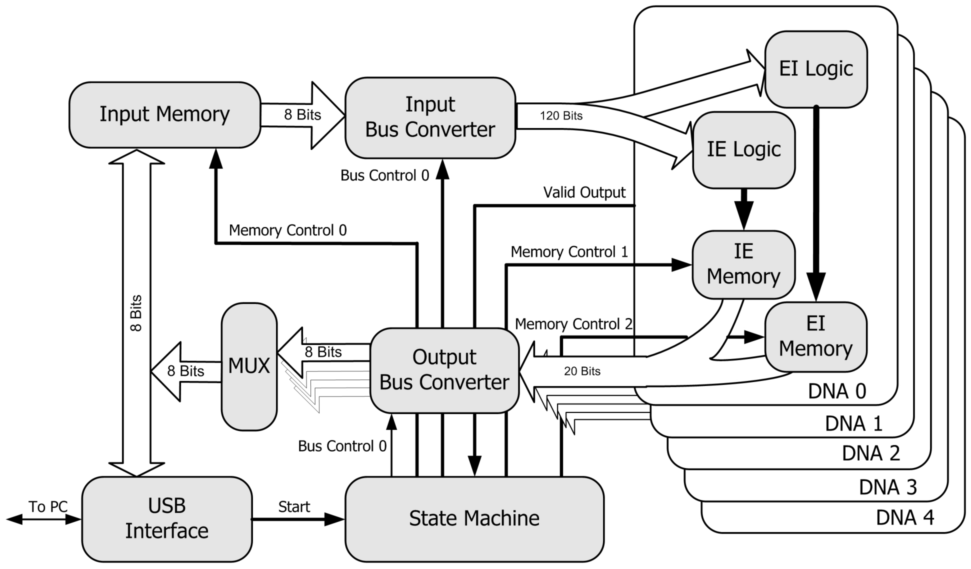


Fig. 3. Simplified block diagram of the configware system.

Altera's development platform Quartus II. The implementation occupied less than 1% of the logic cells and 21% of the memory of the physical device. The system was run with a 55.55 MHz clock.

3.4.1. *Block diagram*

A block diagram of the implemented configware is shown in Fig. 3, and described below. In this block diagram, it is possible to emphasize two basic structures in the implemented architecture: the controller, implemented with a synchronous state machine, and the data path, responsible for input, processing (DNA's blocks), and output of data.

- USB interface: this USB 1.0 interface, external to the FPGA, was implemented to receive/send data from/to a PC. A user-friendly software for data manipulation was developed in the PC using Visual Basic to access this interface;
- Input memory: this 8 bits-wide random access memory (RAM) is used for temporary storage of the set of instances received from the PC, and has 4096 positions;
- Input bus converter: this block groups 15 successive bytes from the input memory in such a way to form a 120 bits-wide bus to the DNA block;
- DNA: this is the main block of the system, where the logical expressions obtained in the previous steps were implemented as logical circuits (details below). Two dedicated 20 bits-wide memories store the results of IE and EI detections. There are five pairs of DNAs, identified as $\mathrm{DNA}_0, \mathrm{DNA}_1, \ldots, \mathrm{DNA}_4$, representing the five classifiers for EI and IE, according to the five-fold cross-validation procedure, mentioned before. Notice that, in contrast to traditional (software-based)

computation, the cross-validation is done in a parallel way. This is our high level of parallelism.

— IE/EI logic: The input of each IE or EI logic is 120 bits, corresponding to 60 nucleotides, and the output is 1 bit representing the presence/absense of a splicing junction in the record under analysis. This is the low level of parallelism, since a whole record is processed at once, thanks to the combinatorial circuit. In a conventional Von Neumann architecture, each term of the logical equation of the decision tree would be processed sequentially. The logical expressions obtained after the Boolean minimization were materialized into combinational logic circuits. Figure 4 shows the combinatorial logic circuit corresponding to the three leaf nodes of Fig. 2, encoded as described in the previous section. In this example, the circuit is already simplified and has two outputs corresponding to the IE and IE classes. The unused inputs are not shown for the sake of clarity. Recall that this figure shows only a small part of the combinatorial circuits that implement the whole decision tree;

— IE/EI memory: The output bit of an IE/EI logic triggers a corresponding output memory. These distributed memories, two for each DNA block, store the number of the input record in which an EI/IE boundary was found. Each output memory is 20 bits-wide and has 1024 positions. Therefore, each DNA block can process up to 1 M records (that is, 60 M nucleotides) having up to 1 K splice junctions of each type identified.

- Output bus converter: it transforms the information stored in the IE/EI memories from a 20-bits bus to a 8-bits output bus, doing three successive reads. In Fig. 3, five 8-bits buses are shown, one for each DNA block. A bus multiplexer allows each DNA block to be accessed by the USB interface and send data to the PC;
- State machine: this is the control module of the system and implements the finite state machine detailed in the next section.

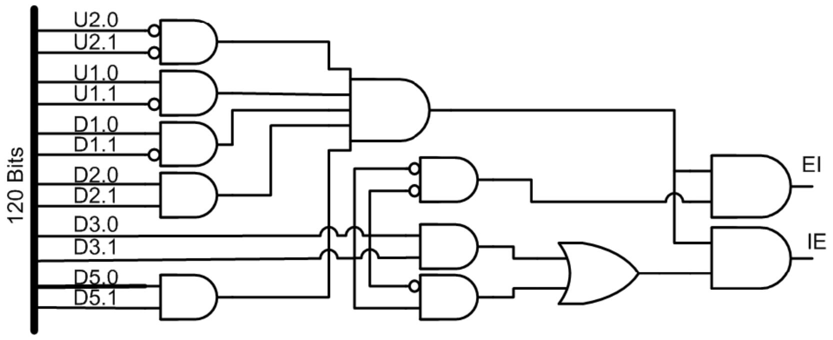


Fig. 4. Part of the combinatorial circuit implemented in the FPGA, corresponding to the leaf nodes of Fig. 2.

3.4.2. *Finite state machine*

Processing takes place in three phases, under the control of a finite state machine: data are sent from PC to the internal input memory (via USB), data are processed (60 nucleotides per clock cycle), and output memories are read to the PC. The finite state machine that controls the system is detailed in Fig. 5 and Table 2. In this machine, state S1 corresponds to the data transfer from the PC to the input memory block. In state S2, successive input memory reads are done to compose the 120 bits-wide bus to feed each DNA block. The DNA processing through the combinatorial circuit occurs in state S3 (the fastest state). Finally, in state S4 the conversion, selection, and transfer of the results are done.

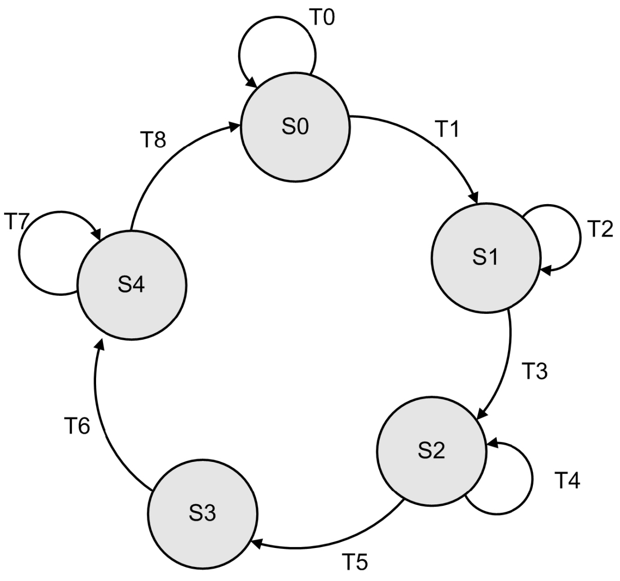


Fig. 5. Diagram of the finite state machine.

Table 2. States and transitions of the finite state machine.

State	State description	Event	Event description	Next state
S0	Initialize counters	T0	Start = 0	S0
		T1	Start transition to 1	S1
S1	Input memory write	T2	Crt0 < 15	S1
	(15 bytes are written)	T3	Crt0 = 15	S2
S2	Input bus conversion	T4	Crt0 > 0	S2
		T5	Crt0 = 0	S3
S3	DNA analysis	T6	DNA valid = 1	S4
S4	IE/EI read	T7	Crt1 < 3	S4
	(3 bytes are read)	T8	Crtl = 3	S0

4. Results and Conclusions

We proposed a methodology for solving an interesting problem in molecular biology: detecting the splice junction of eukaryote genes. We used a decision tree induced by a machine-learning algorithm with cross-validation. The decision tree was simplified and turned into logical equations. Then, they were submitted to a Boolean minimization procedure, yielding two logical expressions for detecting EI and IE boundaries in a 60-nucleotides-wide window. These logical expressions were implemented as combinational circuits in a reconfigurable device.

We have explored parallelism in two levels: at the decision tree (a whole input register is processed at once) and at the cross-validation (all five instances of the classifier are processed in parallel). In a future work, we will explore further the massive parallelization capabilities of the configware, aiming at achieving a much higher throughoutput.

As a result of the five-fold cross-validation procedure, the average classification accuracy achieved was 90.41%. A direct comparison of this result with other approaches in the literature (for the same data set) is not possible due to different methodologies and partitions for training/testing and cross-validation. However, this result is either better or similar to the accuracy rates of other methods,[11,13,19] thus, demonstrating the adequacy of a classifier using decision tree. It should be noted that, to date, no method has achieved 100% of accuracy yet. Possibly, this is not due to the limitation of the methods employed, but due to the inherent imprecision of the data set.

The most important result of the work is regarding the processing speed. The time necessary to process a single record of 60 nucleotides was 18 ns, and for a single partition (with an average of 638×60 nucleotides), 11.484 μs. Recall that this processing time is for a single PE, not the set of five in parallel. Considering that all the five PEs run in parallel, the whole data set is processed in the same 11.484 μs. Using the same rules derived from the decision tree, we implemented a program in C++ programming language to compare processing speed. This program was run in a desktop computer with Pentium IV processor running at 3.2 GHz and with 512 MBytes of RAM, under Microsoft Windows XP. The program was run several times using the whole data set, and the average processing time was 24 ms. The speed-up achieved by the configware approach, relative to the software approach, was around 2000 times. However, this comparison must be carefully interpreted and cannot be generalized. For instance, the running clock of the desktop computer is 57 times faster than the configware clock. If it was possible to run our system in the same clock rate, the final speed-up would be extremely high. Also, the architecture of both systems are radically different, not only how data are input and output but, mainly, how data are processed.

Therefore, the high processing speed enables the configware system to achieve a performance clearly unattainable by common software-based systems, running on regular Von Neumman's machines. For instance, if we suppose that the memory resources were not limited and we had to analyze the complete human genome

(3×10^9 nucleotides) using the same 60 nucleotides-wide window, the proposed system would take around 0.9 s, with a single pair of PEs (i.e., one for EI and other for IE).

The current drawbacks of the system are the limited communication speed, due to the use of an USB 1.0 interface (limited to maximum data transfer rate of 12 Mbits/s), and the limited internal memory (up to 2.5 Mbits). For the analysis of larger amounts of data, this speed and memory limitations can represent important bottlenecks. Future implementation will use an USB 2.0 interface, with a maximum transfer rate of 480 Mbits/s, improving significantly the communication rate, and FPGA devices with higher capacity.

It is a matter of fact that many problems found in bioinformatics represent great computational challenges. Even if we succeed to reduce the complexity of underlying computational analytical models by introducing restrictions and simplifications, frequently, some problems demand an unacceptable processing time or too expensive computational resources. Using configware-based approaches, the one proposed here, it is possible to circumvent several deficiencies found in conventional processing systems for hard computational tasks.

The main goal of this work is the construction of independent blocks that can be joined together in different topologies to solve bioinformatics problems. Reconfigurable systems constitute a powerful methodology that can present practical, efficient, and fast solutions for problems that are usually dealt with software approaches, such as gene detection, protein folding, sequence alignment, and others. We believe that the promising aspects of this technology are flexibility, performance, and real parallelism.

Future work will focus on the evaluation of different topologies for massive parallelism and the implementation of the training of the classifier in configware.

Acknowledgment

This work was partially supported by the Brazilian National Research Council — CNPQ, under grants 501900/2004-7, 506479/2004-8, and 305720/2004-0.

References

1. M. Kanehisa and P. Bork, Bioinformatics in the post-sequence era, *Nat. Genet.* **22**(suppl.) (2003) 305–310.
2. R. Luethy and C. Hoover, Hardware and software systems for accelerating common bioinformatics sequence analysis algorithms, *DDT: Biosilico* **2** (2004) 12–17.
3. J. Becker and R. Hartenstein, Configware and morphware going mainstream, *J. Syst. Architec.* **49** (2003) 127–142.
4. S. Hauck, The roles of FPGA's in reprogrammable systems, *P. IEEE* **86** (1998) 615–636.
5. H. S. Lopes and G. L. Moritz, A distributed approach for multiple sequence alignment using a parallel virtual machine, *Proc. 27th Ann. Int. Conf. IEEE EMBS* (2005), pp. 2843–2846.

6. A. Marongiu, P. Palazzari and V. Rosato, Designing hardware for protein sequence analysis, *Bioinformatics* **19** (2003) 1739–1740.
7. A. L. Lehninger, D. L. Nelson and M. M. Cox, *Principles of Biochemistry*, 2nd edn. (Worth Publishers, New York, 1998), pp. 134–137.
8. J. D. Watson, H. H. Hopkins, J. W. Roberts, J. A. Steitz and A. M. Weiner, *The Molecular Biology of the Gene* (Benjamin-Cummings, Menlo-Park, 1987).
9. The Institute of Electrical and Electronics Engineers, *1076 IEEE Standard VHDL Language Reference Manual* (IEEE Press, New York, 2002).
10. S. A. Ito and L. Carro, A comparison of microcontrollers targeted to FPGA-based embedded applications, *Proc. IEEE 13th Symp. Integrated Circuits and Systems Design* (2000), pp. 397–402.
11. D. Michie, D. J. Spiegehalter and C. C. Taylor, *Machine Learning, Neural and Statistical Classification* (Ellis Horwood, Chichester, 1994).
12. I. H. Witten and E. Frank, *Data Mining*: *Practical Machine Learning Tools and Techniques with Java Implementations* (Morgan Kaufmann, San Francisco, 2000).
13. M. O. Noordewier, G. G. Towell and J. W. Shavlik, Training knowledge-based neural networks to recognize genes in DNA sequences, *Advances in Neural Information Processing Systems*, Vol. 3, eds. R. Lippmann *et al.* (Morgan Kaufmann, San Francisco, 1991).
14. M. Canella, F. Miglioli, A. Bogliolo, E. Petraglio and E. Sanchez, Performing DNA comparison on a bio-inspired tissue of FPGAs, *Proc. IEEE Int. Parallel and Distributed Processing Symp.* (2003), pp. 193–199.
15. T. Oliver, B. Schmidt, D. Natghan, R. Clemens and D. Maskell, Using reconfigurable hardware to accelerate multiple sequence alignment with ClustalW, *Bioinformatics* **21** (2005) 3431–3432.
16. Y. Yamaguchi, T. Maruyama and A. Konagaya, High speed homology search with FPGAs, *Proc. Pacific Symp. Biocomputing* (2002), pp. 271–282.
17. N. B. Armstrong, Jr., H. S. Lopes and C. R. E. Lima, Preliminary steps towards protein folding prediction using reconfigurable computing, *Proc. IEEE Int. Conf. Reconfigurable Computing and FPGA's*, San Luis de Potosi, Mexico (IEEE Computer Press, Piscataway, 2006), pp. 92–98.
18. S. Hettich, C. L. Blake and C. J. Merz, UCI repository of machine learning databases, University of California at Irvine (1998), http://www.ics.uci.edu/~mlearn/MLRepository.html.
19. G. G. Towell, Symbolic knowledge and neural networks: Insertion, refinement and extraction, PhD. thesis, Department of Computer Science, University of Wisconsin-Madison (1991).
20. D. A. Benson, I. Karsch-Mizrachi, D. J. Lipman, J. Ostell and D. L. Wheeler, GenBank, *Nucleic Acids Res.* **34** (2006) D16–D20.
21. J. R. Quinlan, *C4.5: Programs for Machine Learning* (Morgan Kaufmann, San Francisco, 1993).
22. Boolean minimizer, http://www.gmdsoft.de/mitsch/software/boolmin/.
23. R. Rudell and A. Sangiovanni-Vincentelli, Exact minimization of multiple-valued functions for PLA, *Proc. Int. Conf. Computer-Aided Design* (1986), pp. 352–355.
24. Altera Corporation, http://www.altera.com/.