Run Time Reconfiguration of FPGA for Scanning Genomic DataBases

Eric Lemoine and David Merceron
LIRMM UMR 9928 CNRS/Montpellier II
161 rue ADA - 34392 Montpellier cedex 5 - FRANCE

Abstract

This paper evaluates the feasibility of reconfiguring an FPGA at run time, and tests its performance using a "Grand Challenge Problem", the high speed scanning of genomic sequence databases. Algorithm implementation into a XC3090 FPGA is described, and methods proposed for generating a placed Xilinx Netlist File that can be efficiently routed at run time by the Automated Placing and Routing Xilinx tools, in order to increase the speed and the density of the design. The same algorithm carefully optimised on a RISC processor has been compared with the run time reconfigurated FPGA, and shows the latter to have an improvement in speed of two to three orders of magnitude.

1 Introduction

Knowledge discovery in genetic sequence databases is considered to be a "Grand Challenge Problem"[5]. The "Challenge" is to be able to scan the entire Human Genome Database (800 MBytes) and provide a result in real time, so that there is profitable interaction between the user biologist and the system.

An accelerator dedicated to this task, and more precisely to that of homology detection, can contribute significantly to performance, as previously described in the literature [10, 4, 19, 9, 11, 13]. This is important given the exponential increase in size of the genomic sequence database due to world-wide progress in the field of genome mapping and sequencing. Specialized hardware for genetic sequence database is vitally important from a practical point of view: it is able to present the database as an everyday tool, and can be considered by the biologist user as an *in silico* lab. Any system aiming to satisfy these criteria must be able to demonstrate real time interaction with the end user, without being too expensive.

In [13] we propose a system, A^2R^2, that satisfies the interaction time constraint. A^2R^2 was implemented into the DECPeRLe-1 board. The latter is a recon figurable board with 23 FPGA XC3090, made at th Paris Research Laboratory of Digital by the team o J. Vuillemin [2, 3, 18]. The key to A^2R^2 computin performance lies in its modification of the bitstrean file at run time.

However the DECPeRLe-1 system has many fea tures which are not used by A^2R^2, but which incu system overheads. It was therefore necessary to desig a new card with the objective of preserving, or ever improving, existing performance at the same time a significantly reducing the cost. It soon became eviden that the number of FPGAs was the most importan cost factor. Therefore it was decided to investigate th possibility of a complete reconfiguration of FPGAs a run time to increase their densities and thus reduc the number used. This has been reported by previou studies [6, 8].

In this paper we present a scanning algorithm which implements queries into a FPGA at run time Two kinds of dynamic reconfiguration have been explored, the complete generation of an Xilinx Nelist File (XNF) together with its compilation, and the direct modification of a bitstream derived from a configuration bitstream database. The target FPGA was the XC3xxx series, and experimentation was performed on a DECPeRLe-1 board in order to validate the dynamic configuration concept. Run time reconfiguration enabled the entire human genome to be scanned in less than 100 seconds.

The first part of this study describes the scanning algorithm and its use in a biological context. The second describes implementation of the algorithm into a FPGA, after which run time reconfiguration is detailed and explained. In the final section the scanning speed of the algorithm on DECPeRLe-1 board is compared against a workstation.

0-8186-7086-X/95 $04.00

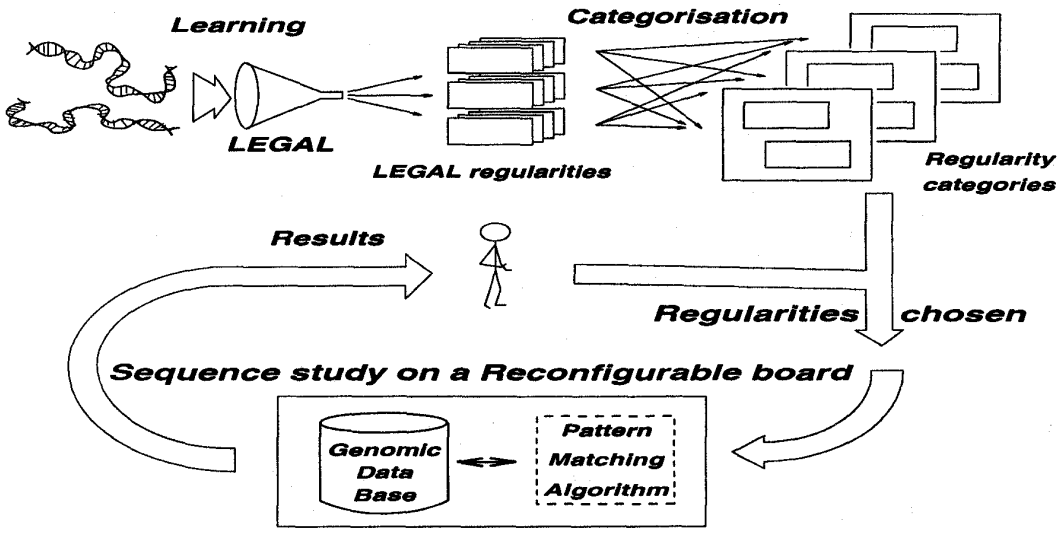


Figure 1: Overview of the genome scanning system

2 Scanning the Genome

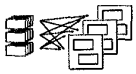
Homology detection in large databases is probably the most time consuming operation in molecular genetic computing systems. Moreover, the progress made all around the world concerning the mapping and sequencing of the genome of Homo Sapiens and other species has increased the size of databases exponentially to an extent where even the best workstation would not be able to reach the scanning speed required. In order to answer this need we propose an algorithm, A^2R^2, and its implementation on a FPGA based system; the DECPeRLe-1 board.

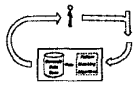
Two kinds of algorithms are used to search in molecular genetic databases. The first kind is based on dynamic programming and the second on word processing. A^2R^2 belongs to the second kind.

The main restriction used to be that the quality of the result was directly connected to the query (i.e a set of motifs) given to A^2R^2, and therefore brought back to the user the responsability for the quality of search. This is no longer a problem, since queries are now produced directly by a learning algorithm, Legal [15, 14]. Moreover the query has been improved by the adjunction of a decision function that reduces the output bandwidth of A^2R^2.

Genetic sequence analysis can be described as the union of three main components: a biologist, a pattern matching algorithm and a genetic sequences database. The objective of the biologist is to extract new knowledge from the database from a minimum of initial information. Generally this initial information is reduced to a small set of sequences or fragment of sequences already inter-associated. The task of the knowledge system is to produce a larger set of sequences with common characteristics controlled by the biologist. The knowledge search is mainly formulas in Conjunctive Normal Form (motifs [13]) associated with a Majority And operator. Figure 1 summarizes the process;


This step produces regularities from a set of examples and counter examples. Legal, the learning algorithm, outputs several sets of regularities. These regularities are premium knowledges extracted by the system.


The regularities are clusterized by the biologist and transforms into a query. This query interrogates the database by means of a pattern matching algorithm.


The results from the pattern matching algorithm on the query previoulsy defined are sent to the biologist. He can then modify his request according to the search result, and reinitiate a new query. Thus an interaction loop is established between the biologist and the system. The quality and productivity of this kind of interaction depend on the reaction time of the computer system to a request of the biologist. Indeed, the type of questions asked by the user biologist differs according to whether he had to wait a few hours or only a few minutes.

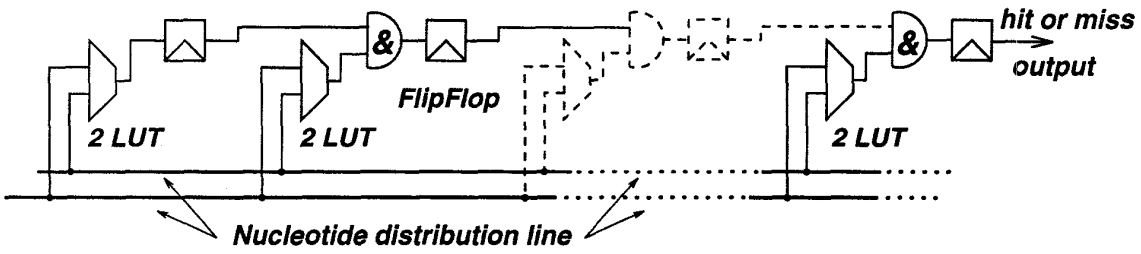


Figure 2: A motif detector

2.1 Motifs

Motifs are formed by a sub-alphabet vector, (ie an alphabet reduced to authorized letters at the position in the motif) [12, 13]. For example [A,G]••[A,T]G•[A,C][C,T,G] is a motif (• is a short cut for the whole nucleotides alphabet; [A,T,G,C]) that can match with sequences such as ATTTGAAC, GATAGCCT and so on. These motifs will be expressed in conjunctive normal form as follows: $\bigwedge_{0 \leq i \leq 7} x_i \in \mathcal{A}_i$ Where for instance $x_6 \in \mathcal{A}_6$ is the sub-alphabet in positions 6, $x_6 \in \mathcal{A}_6 = (x_6 = \mathrm{A} \vee x_6 = \mathrm{C})$ and x_6 the sixth nucleotide variables of the window on the sequence scanned.

2.2 Majority Logic

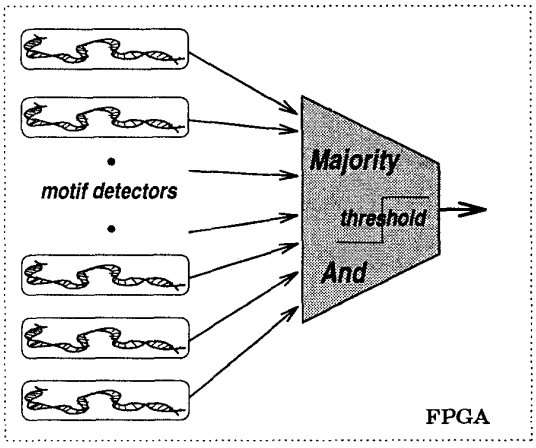


Figure 3: FPGA utilisation with motif detectors

In order to implement the majority vote needed for empirical proof we used a Majority And (MajAnd), that is a parallel counter with a threshold output. A (n,m) Parallel Counter (PCs) is an n-inputs circuit that produces a $m = \lceil \log_2(n) \rceil$ bit binary count of the number of its inputs that are ONEs. PCs are extensively used in VLSI design to implement fast multipliers, multioperand adders, and digital neural networks. A well known property of any PC is the recursive decomposition that enables an assembly of smaller PCs to form a greater one. Dadda [7] has described an optimal combination of (3,2) PCs as building blocks (Fig 6). (3,2) PC is simply a Full Adder that adds 3 bits of the same weight and produces two outputs. A (n,m) PC would require less than n Full Adder if we used the Dadda decomposition.

The decision function that transforms a (n,m) PC in MajAnd with n inputs is a simple threshold detector.

The utilisation of a MAJ& with motif detectors is shown in Figure 3.

3 Implementation into a XC3XXX FPGA

Figure 2 introduces the logic description of a motif detector. We will stress three points:

- As the database is scanned one nucleotide by one the detector is fully pipelined.
- As the nucleotide alphabet has only four letters (A, T, G, C), the motifs are stored in several 2 input Look-up tables, one LUT by position.
- Because of the output retiming, each LUT sees the same nucleotide at the same time. Therefore the fan out of the nucleotide distribution lines is high.

Two implementations are possible. The first one is to pack in the same CLB the two stages of the detector, (2 times (one 2 LUT, one AND gate and one Flip Flop)). This gives 4 CLBs for a detector of length 8. The second is to pack in the same CLB, two stages each from two different detectors. The distribution of the nucleotide is always done by the horizontal long

lines and because of the high fan out driven by this long line we charge it with a tristate buffer as shown in the Figure 5.

These motif detector implementations are very compact, moreover they reduce the routing constraints since direct connect or very close connections are made. In addition, with only one CLB in the data path between two flip flops the timing constraints are low. The main advantage of the nucleotide distribution network is the possibility offered to the placement tool to choose every position for the detector as it is depicted in Figure 4

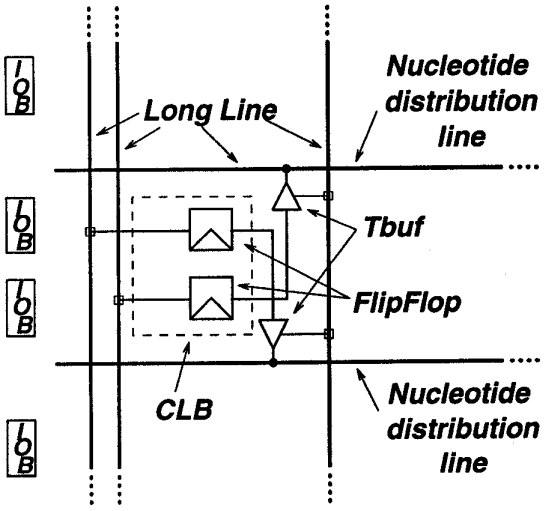


Figure 5: One Head of the nucleotide distibutions line

3.1 Majority And Implementation

The Dadda decomposition (Fig 6) of a parallel counter is made with Full Adder (FA) and Half Adder (HA). The fan in of both the FA and the HA was always less than three and the fan out is two, so the FA and HA were implemented straight into one CLB. Because of retiming constraints, some CLBs were added to the bypassing wire so that flip flop could be added.

The threshold function is implemented with a $m \mapsto 1$ look-up table. When m is superior to 5 the number of input of a XC3XXX CLB, the $m \mapsto 1$ LUT is decomposed in two or more CLBs. The main advantage of this implementation is the possibility to change the threshold by writing straight into the bitstream configuration of the FPGAs, thus avoiding a complete redesign of the entire FPGA.

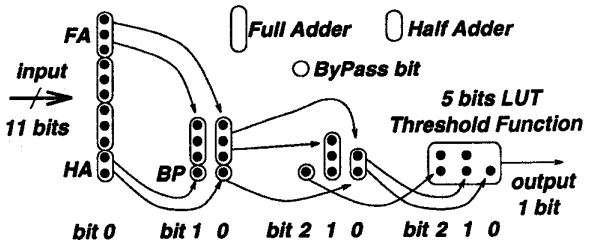


Figure 6: ≈ Dadda decomposition of a 11 bits Majority And

4 Run Time Reconfiguration

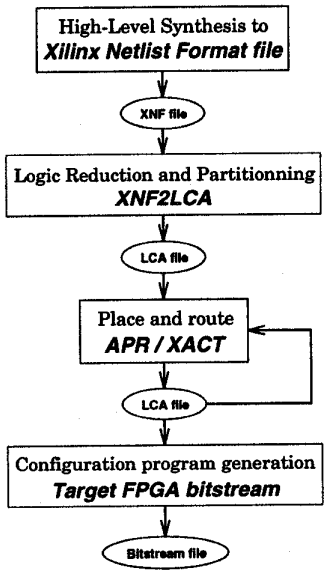


Figure 7: Main step of a Xilinx FPGA design synthesis

Figure 7 presents the design process of a FPGA and shows the three main files that mark the progression towards a FPGA layout. Dynamic reconfiguration can be characterized by the file used as a starting point for the reconfiguration.

4.1 An Overview

Dynamic reconfiguration of Xilinx FPGA can be executed in three different ways to increase the density and/or the speed of design:

- Alternate between several designs already in bitstream file. As pointed out by Eldredge & Hutchings [8] this type of run-time reconfiguration in-

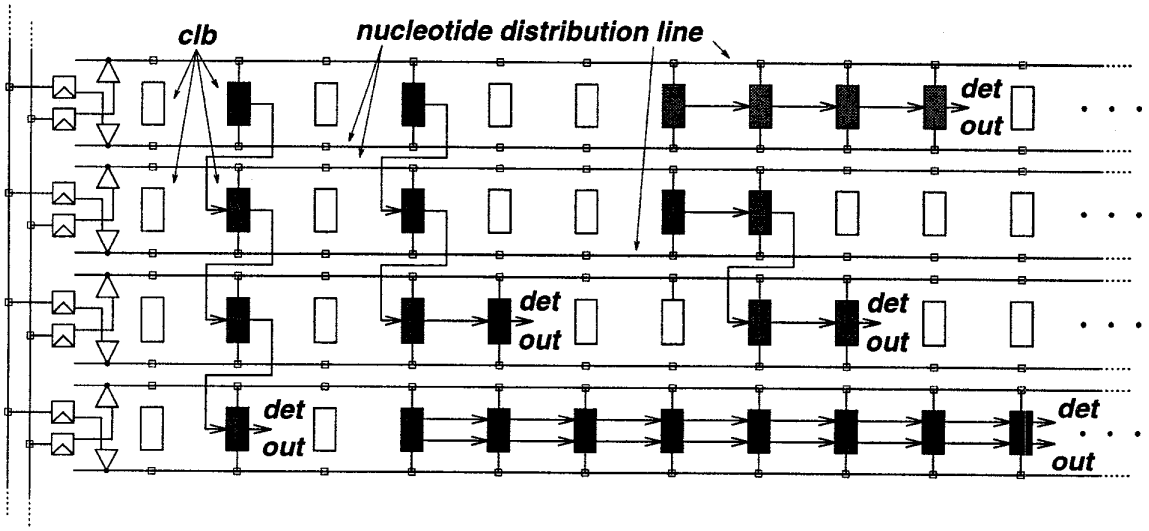


Figure 4: Possible placement of a motif detector (slice view of the FPGA)

creases the hardware density of the FPGA. Indeed, sequential and distinct parts of an algorithm can be implemented in different designs and scheduled at run time. Thus one algorithm is implemented in several virtual FPGAs and executed on one.

- Modify directly the bitstream file according to information only known at run time. This approach, presented for the first time by Cox & Blanx [6], increases both the density and the speed of the design. For example Cox and Blanx implement multipliers of 8bits variables by 8bits constants. The 8bits constants are hardwired in the look-up table of the CLBs. At run-time the constants are written into the bitstream and then the bitstream is downloaded in the FPGA. As a mutiplier by a constant needs less CLBs and runs faster than a true multiplier, the overall design is much denser and faster when many multiplications are needed.

- Generate the design at runtime from a Xilinx Netlist File to bitstream according to information only known at run time as in the previous approach. The main idea is the same as the Cox & Blanx approach with however a better gain in density and speed at the expense of more complicated realization. An example can be found in [16] but the strategy used for the generation of the XNF does not be apply to a real run time reconfiguration.

We next describe our own approach, which integrates these three in order to implement at runtime a query processor in an FPGA board.

4.2 Bitstream Modification

As can be seen below there is a choice of two techniques for reconfiguration at run time. The easiest in terms of both implementation and design speed is the bitstream modification.

Two operations are allowed; changing the motif in a detector and changing the threshold in a MAJ&. Thus once the reference bitstream have been chosen, the LUT of the CLBs are generated according to the placement file. The database of reference bitstream contains several basic designs and obviously grows with each new fully reconfigurated design. Therefore if we restrain ourself to only changing the LookUp Table (LUT) the run time reconfiguration is straightforward, as the LUT writing is done on the fly before downloading in the FPGA [12, 13].

Run time generation of a netlist, full reconfiguration, adds an important time overhead, but is more flexible and increases both the speed and density of the FPGA.

4.3 Full Reconfiguration

The main difficulty of full reconfiguration is to obtain a predictable result from the APR tool and avoid the most time consuming part; the annealing placement. As we want a very high speed (more than

33 MHz for the XC3090-100MHz part) the first decision was to use a maximum of one CLB between two flip flops, so that the design is fully pipelined. This is a not a drawback as the database is scanned one nucleotide by one and the motif detectors are sequential (Fig 2).

The second decision was taken after we counted the maximum number of motif detectors we could put into a XC3090. One motif detector requires 4 CLBs, so that a maximum of 80 detectors can be placed into the XC3090. Since a MAJ& requires at most one CLBs by input, this gives the rule of thumb of 5 CLBs by detector and the maximum drops to 60. As a result the biggest MAJ& that can be implemented into the XC3090 is a 60 input MAJ&. The best solution was therefore to predesign all the MAJ& with 2 to 60 inputs.

4.3.1 Automated Placement Techniques

We did not write a new general placement algorithm. Instead we defined a specific placement tool for our application. Basically, two kind of objects should be placed, the motif detectors and the Majority Ands. The nucleotide distribution lines enable the motif detector to be placed everywhere in the FPGA. The most difficult to place automatically with a good routing result is the MAJ&. The strategy we chose was to retain one relative placement by MAJ&, and these placement because of the reduced number of possible MAJ&s, was made at the compilation time and stored with the MAJ& wiring description.

This relative placement was obtained with the Automated Place and Route Xilinx tool but with considerable constraints. The main idea was to fill the FPGA with placed motif detector. Then allow just a rectangular zone of $a \times b$ free CLB for a n input MAJ& ($n \leq a \times b$). The MAJ& was not placed but each FA or HA output was stored in a flip flop and also the bypass wires. Furthermore the two outputs of a FA or HA were grouped in the same CLB. In these conditions the APR performes a quick and good placement that enable the clock, of a XC3090-100 MHz with 7 ns setup time and a 5 ns clock to output time, to be set at 40 MHz.

This performance is possible thanks to the low routing resource required for the motif detector associated with its flexibility. What is more the full pipelining of the MAJ& helped the APR tools to find a very close placement between the CLBs forming the MAJ&.

Thus the placement at run time was performed in three steps:

- Place all the MAJ& in the FPGA according to the precalculated placement.
- Then use the flexibility of detector implementation to place the output of the detector in the nearest position from the input of the MAJ&. We have experimentally found that the best choice is to fill up the left side of the Xilinx with the tristate buffers and detector in double line, two detectors sharing 8 CLBs (see Fig 4), putting the MAJ&s and then placing the remaining detectors.
- Eventually store the information relative to the position of each motif look-up table and each threshold look-up table necessary for the direct bitstream modification.

This placement strategy constrains the APR tools in such a way that they cannot generate a bad routing. All the MAJ& CLB are placed, their inputs set. The 2 nucleotide inputs of each detector CLB are set too, thus only a few CLB have one free input. Under these constraints the APR tool always finds a way of less than 13 ns (XC3090-100) between two flip flops.

5 Results

We describe below the experimental platform used to support the biological application, which is based on Xilinx XC3090-100 MHz FPGAs.

5.1 A Pure Software Implementation

In order to compare the work done by one FPGA against a workstation, motif detection plus the majority and is performed as follows (C language):

```
p = 0;
for (; p < nb_of_motifs; p++)
/* 3 OPs: compare, branch, add */
   {
    match = motifs[i] ^ sequence_window;
         /* => 2 OPs: load, xor */
    match = match |((match >>1)& mask1);
       /* => 3 OPs: and, shift, or */
    match = match |((match >>2)& mask2);
       /* => 3 OPs: and, shift, or */
    etmaj += match == mask3;
        /* => 1 OP: compare, add */
   }
```

A branch counts as one operation, but has the value of zero cycles. Given the hypothesis that the hardware

executes one instruction per cycle ($CPI = 1$), with perfect cache, motif detection would take 12 cycles. The MajAnd calculation does not add any cycles since it is integrated into motif detection, and in any case the result of the match needs to be stored. Therefore the number of nucleotides per second, $\mathcal{N}$, scanned with p motifs of length up to 8 is:

$$\mathcal{N} = \frac{1}{12 \times p \times CPI \times \tau_{\mu P}}$$

where $\tau_{\mu P}$ is the cycle time of the microprocessor used and CPI the average number of integer instructions per cycle.

For instance with the DECstation 5000/240 with a 40 MHz R3000 microprocessor: $\tau_{\mu P} = 25\,10^{-9}$s, $w = 32$, we can scan (in nucleotides per second):

$p = 10$ $\quad 3.3\,10^5$ nuc.s^{-1}
$p = 25$ $\quad 1.3\,10^5$ nuc.s^{-1}
$p = 50$ $\quad 6.6\,10^4$ nuc.s^{-1}
$p = 100$ $\quad 3.3\,10^4$ nuc.s^{-1}

5.2 The Experimental Platform

A DEC workstation running at 40 MHz with a Reconfigurable board attached to it constitutes the DECPeRLe-1 experimental platform. The board contains 23 Xilinx XC3090-100 FPGAs and 4 MBytes of SRAM. 16 FPGAs are connected in a 2D array with local interconnection and memory buses. 7360 (23×320: 320 CLBs per FPGA), binary functions, equivalent to 1 bit arithmetic and logic unit, can be evaluated, at each cycle, in the array. The maximum bandwidth between this array and the memory is 320 MBytes/s with an access time from SRAM to FPGAs of 50ns. The hardware is connected to a DecStation 5000/240 by a bus TurboChannel at 100 MBytes/s peak and around 20 MBytes/s sustained by the disk.

On this UNIX workstation the design is developed in C++ and translated through Xilinx tools into a bitstream configuration ready to be loaded onto the FPGAs. However in our applications the Xilinx Netlist File (XNF) is generated directly. The bitstream configuration can be considered as an unique nanoinstruction of 1.4 Megabits width that loads itself in less than 50 ms onto the card [3, 1, 18, 17].

5.3 FPGA Timing

Motif detectors and Majority Ands are implemented spatially rather than temporally since all operations can be pipelined so that they always have a combined time equal to one cycle, at the expense of several latency cycles. An eight nucleotide detector takes 4 CLBs, a CLB being the base unit of reconfigurable circuits, and a n entry MajAnd always takes less than n CLBs. Hence the scanning time formula:

$$T_{FPGA} = \left\lceil \frac{5 \times p}{n_{CLB}} \right\rceil \times \mathcal{N} \times \tau_{FPGA}$$

Where τ_{FPGA} is the cycle time of the reconfigurable board and n_{CLB} is the number of CLBs available in a circuit or a multi-circuit card. If the number of CLBs required to implement the detectors and the MajAnd is greater than the total number of available CLBs, the design needs to be divided into two and therefore the database needs to be scanned twice.

The calculation time in this case is: $T_{FPGA} = \mathcal{N} \times \tau_{FPGA}$.

5.4 Experimental timing

Once the user has queried the system, the queries must be wired onto the hardware unit. This stream (the slowest) is totally dependent on the Xilinx APR tool performance. The system can wire up to 60 motifs per FPGA XC3090 (used on the DecPerle-1) with a 8 letter width motif size. The time is decomposed as follow:

- Query translation to XNF: around 100 milliseconds.
- XNF to bitstream for one FPGA: around 90 second, LCA routage + bitstream generation.
- Download 100 milliseconds on the DECPeRLe-1 board.

Figure 8 synthezises the results. This genetic sequence study compares execution on a general purpose machine and on our experimental platform. The difference between the standard and the custom reconfiguration is in the time needed to reconfigure in part from a generic design (standard option) or the whole from scratch (custom option). In this case the generation of the design takes around 100 seconds during which time data cannot be treated on the DECPeRLe-1 board.

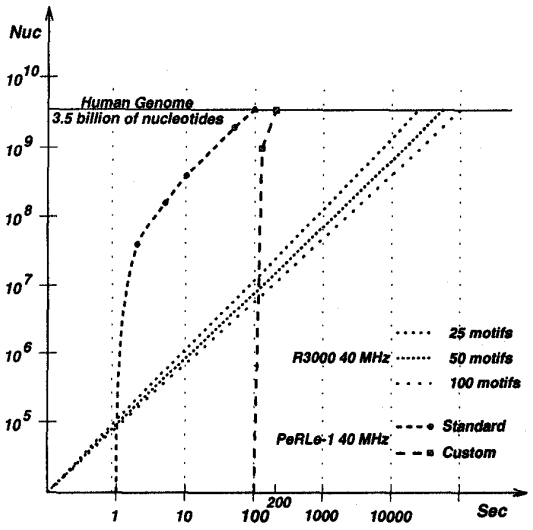


Figure 8: FPGA versus software scanning of a genetic sequence database

The dashed line curves are DECPeRLe-1 card performances, and the dotted line curves are purely software execution results. The "standard" curve shows the results for a library-design, and the "custom" is obtained with a computed design. The former treats human genome in 90 seconds, and the latter needs 100 seconds more to compute and load the design.

As a result, execution on the general purpose machine is more performant during this time period, the two curves intersecting at 10^7 nucleotides. Against this, the DecPeRLe-1 card, which can treat 40 million nucleotides per second, takes less than 90 seconds to treat all the human genome, which would not be possible with a purely software application. The first 100 seconds corresponds to a treatment at the software level, after which the reconfigurable hardware takes over. The speed of the FPGA is two to three orders of magnitude faster. From another point of view the speed of the reconfigurable system allows the user biologist to interact in real time with the machine, whereas the software solution requires batch processing dramatically changing the way the biologist interacts with the machine.

6 Conclusions

The search for knowledge in genetic sequence databases can be considered as a good "Grand Challenge Problem" example for modern day workstations, as the time taken to scan the entire human genome for a complex query varies from 5 hours to more than one day.

In this article we demonstrate that adding a FPGA based reconfigurable coprocessor to a workstation improves calculation time by two to three orders of magnitude in the given context. This increase in performance can only be obtained by a run time reconfigurated FPGA, as previously reported by Cox & Blanz, and Eldredge & Hutchings [6, 8].

Two methods of reconfiguration have been used. One consists of modifying the bitstream prior to run time in order to be able to enter problem data only known at execution time. The other method is more ambitious, since it generates a new bitstream from a placed netlist (XNF). After studying the implementation of base operators, motif detectors and Majority Ands, a placement technique has been defined which optimises the routing process. The result is a design generated at run time that has a guaranteed clock speed of 40 MHz.

The techniques proposed in this study constitute one more step in the process of improving FPGA speed and density through complete FPGA dynamic reconfiguration at run time. The arrival of new FPGAs with dynamically modifiable reconfiguration memories will enable their potential to be more fully exploited, and will open the way for their use in many new fields of application.

7 Acknowledgements

A great thanks to the DECPeRLe-1 team P. Bertin, P. Boucard, D. Roncin, M. Shand, H. Touati and J. Vuillemin for having introduced to us the reconfigurable concept and helping us along the way.

References

[1] Patrice Bertin. *Mémoires actives programmables : conception, réalisation et programmation.* PhD thesis, Université de PARIS VII, 1992. in French.

[2] Patrice Bertin, Didier Roncin, and Jean Vuillemin. Introduction to programmable active memories. In *Systolic Array Processors*, pages 300–309. J. McCanny et al, 1989.

[3] Patrice Bertin, Didier Roncin, and Jean Vuillemin. Programmable active memories: a performance assessment. In *Proc of the Symp. on Integrated Systems.* University of Washington, 1993.

[4] E. T. Chow, T. Hunkapiller, J. C. Peterson, and B. A. Zimmerman. Biological information signal processor. In *Proceedings of the Int. Conf. on Aplication Specific Array Processor*, pages 144–160, 1991.

[5] Comittee on Physical, Mathematical, and Engineering Sciences. Grand Challenges: High-performance computing and communications. National Science Fondation, Washington D.C., 1991.

[6] C. E. Cox and W. E. Blanz. GANGLION - a fast field-programmable gate array implementation of a connectionist classifier. *IEEE J. Solid-State Circuits*, 27(3):288–298, March 1992.

[7] Luigi Dadda. Some schemes for parallel multipliers. *Alta Frequenza*, 19:349–356, 1965.

[8] James G. Eldredge and Brad L. Hutchings. RRANN: the run time reconfiguration artificial neural network. In *Proc of Custom Integrated Circuits Conference*, pages 77–80. IEEE, 1994.

[9] Barry Fagin and J. Gill Watt. FPGA and rapid prototyping technology use in a special purpose computer for molecular genetics. In *Proc. of the Int. Conf. on Computer Design*, pages 496–501. IEEE, 1992.

[10] M. Gokhale, W. Holmes, A. Kosper, S. Lucas, R. Minnich, D. Sweely, and D. Lopresti. Building and using a highly parallel programmable logic array. *IEEE Computer*, 24(1):81–89, January 1991.

[11] Dzung T. Hoang. Searching genetic database on splash 2. In D. A. Buell and K. L. Pocek, editors, *Proc. of the Workshop on FPGAs as Custom Computing Machines*, pages 185–191, Napa, CA, April 1993. IEEE.

[12] Eric Lemoine. Reconfigurable hardware for molecular biology computing systems. In *Proc. of Int. Conf. on Application-Specific Array Processors*, pages 184–187, 1993.

[13] Eric Lemoine, Joel Quinqueton, and Jean Sallantin. High speed pattern matching in genetic data base with reconfigurable hardware. In *Proc. of the 2nd Int. Conf. on Intelligent Systems for Molecular Biology*, pages 269–276. AAAI, 1994.

[14] Engelbert Mephu Nguifo. Galois lattice: A framework for concept learning-design, evaluation and refinement. In Koutsougeras & al, editor, *Proc. of 6th Intl. conference on Tools for Artificial Intelligence*. IEEE, November 1994.

[15] Engelbert Mephu Nguifo and Jean Sallantin. Prediction of primate splice junction gene sequences with a cooperative knowledge acquisition system. In *Proc. of the First Int. Conf. on Intelligent Systems for Molecular Biology*, pages 292–300. AAAI, 1993.

[16] M. Robert, P. Gorria, J. Mitéran, and S. Turgis. Architecture for a real time classification processor. In *Proc of Custom Integrated Circuits Conference*, pages 197–200. IEEE, 1994.

[17] Hervé Touati. Perle1DC: a C++ library for the simulation and generation of DECPeRLe-1 designs. Technical Report TN4, Digital Paris Research Laboratory, 1992.

[18] J. Vuillemin, P. Bertin, D. Roncin, M. Shand, H. Touati, and P. Boucard. Programmable active memories: Reconfigurable systems come of age. *to appear in IEEE Transaction on VLSI Systems*, 1995.

[19] C. T. White, R. K. Singh, P. B. Reintjes, J. Lampe, B. W. Erickson, W. D. Dettloff, V. L. Chi, and S. F. Altschul. BioSCAN: A VLSI-based system for biosequence analysis. In *Proceedings of the Int. Conf. on Computer Design*. IEEE, 1991.