1997 First International Conference on Knowledge-Based Intelligent Electronic Systems, 21-23 May 1997, Adelaide, Australia. Editor, L.C. Jain

High Speed Intelligent Machine through Programmable Hardware : Application to Genomic Systems

Eric Lemoine *
el@adb.fr

Laurent Maillet-Contoz †
maillet_contoz@lirmm.fr

David Merceron *†
merceron@lirmm.fr

Jean Sallantin†
sallantin@lirmm.fr

Abstract

This paper proposes a new technology for intelligent machines, based on the concept of programmable hardware. To build an intelligent system, the designer has to adapt it to the problem. First we show that programmable hardware is an intermediate step for building configurations, in order to choose the best architecture. In this case, the tasks are performed in a time period that respects human cognitive capacities. Next is detailed a multilevel model composed of the cognitive, software and hardware levels. An experimental platform has been built based on programmable hardware, and used in a "Grand Challenge" problem: knowledge discovery in genetic sequence databases, to compare the relative efficiencies of programmable hardware and classical Von Neumann based architecture. Programmable hardware has shown to have a significantly faster response time, which is essential for modern day intelligent machine user interaction.

1 Introduction

The international communication networks impose performant machines to manage the communication flows that it generates. Needs of the users are highly heterogeneous and each kind of usage forces constraints on the response delays. An important point therefore is to satisfy the user by assuring a response time of queries. Generalist machines may be applied to solve problems from large different areas, since they are programmable as wanted. Nevertheless, this is obtained with a loss of computing power, since the instruction set is obviously not adapted to the problem to deal with. At the opposite, dedicated machines are optimized for a special usage, in a specific context. Thanks to this optimization, they have great performances, but they can only be applied to a small range of problems.

Nowadays, the increasing amount of multimedia data imposes a great speed up of the applications, in order to allow knowledge acquisition and keep interaction with the user. Newell [16] found that low level knowledge acquisition cognitive tasks were performed in a time bracket between 2 and 99 seconds, the average time being about 10 seconds. According to Newell, all the tasks may be organized in several levels (Fig. 1).

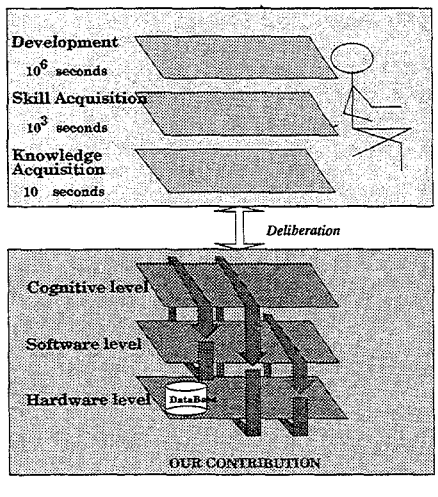

Figure 1: The user analyses and controls activities according to the results produced by the system. This mechanism is generally called **deliberation mechanism**. The mentioned levels follow Newell's architecture.

*Advanced DataBase (ADB), 2 rue Niepce, 60200 Compiègne - FRANCE
†LIRMM, UMR 5506 Université Montpellier II - CNRS 161, Rue Ada 34392 Montpellier Cedex 5 - FRANCE

0-7803-3755-7/97/$5.00

At first, to respect this interaction time, a speed up could be proceeded by software, but this is not enough, since the unceasing increase of the flow of data can not be contained by a purely software solution. Therefore, another kind of speed up should be proposed, in order to deal with a real size problem as large as the genetic domain. To get a good speed up, while having an adequation of the machine to the application and the environment, a solution consists in adapting the architecture of the machine contextually to the problem [11]. Conceptually, the speed up is provided by a dynamically reconfigurable dedicated machine, taking into account the environment.

A new kind of hardware, such as the Configurable Array Logic [9], or Programmable Active Memories [3], uses programs to upgrade hardware efficiency to cope with certain tasks. By nicely programming this hardware, the machine is adapted to the problem. This flexibility results from the possibility to adapt the hardware to any kind of application with a development time similar to that of software design. Execution time should be compared to that of special-purpose hardware. However, the hardware currently used does not yet allow a dynamic reconfiguration of the machine during the execution time. This should be possible with a future generation of hardware, particularly with the Xilinx 6200 family. The objective of this study is to demonstrate that programmable hardware can give an intelligent machine the capacity to interact with humans, to succeed in a cognitive task. We thus implement into an experimental platform a "Grand Challenge Problem" [5], knowledge discovery in genetic sequence databases [15, 14, 17]. The "Challenge" is to both scan the entire Human genome database (800 MBytes) and control the result in real time, to enable a profitable interaction between the user biologist and our system. We call scanning a genomic database, the search of a pattern in a sequential list of characters in the four letter alphabet {A,C,G,T} which are the four nucleotides found in the genome. The entire responsibility of the results relevance is given to the human agent (the biologist), who may initiate new queries or refine previous ones, the so-called control of the results.

This paper is organized as follows: an introductive section gives an overview of programmable hardware. Then we present an architecture for an efficient scanning of a genomic database according to user rules (patterns). Before conclusion, we experiment and compare our programmable hardware platform, made of a workstation and a programmable prototype realized by a Research Center of Digital Equipment Corporation, with the workstation itself, and show that the latter is unable to scan the entire database in a human-scale time period.

2 Programmable hardware : an overview

The first requirement of artificial cognitive systems is that the increase in performance necessary for a complex set of actions be controlled from within the system. As in many other domains we have to deal with the trade-off between generality and performance, in other words programmable computers or special purpose computers. We propose programmable hardware to bridge the gap between flexibility and computing power.

2.1 Programmable hardware

The programmable hardware concept, as well as its implementation with Field Programmable Gate Arrays (FPGAs), was introduced by different teams at the end of the 80's. Vuillemin, Lopresti and Kean described various systems based on this concept [3, 8, 9], and indeed, this concept has been first proposed in the literature since the 70's. One of the first to have expressed it was Schaffner [18]. However, it was only with the arrival of the SRAM based FPGA in 1985 by Xilinx that it became possible to implement this concept.

This flexibility results from the possibility to adapt the hardware to any kind of application with a development time similar to that of software design. Execution time should be compared to that of special-purpose hardware thanks to the realization, through a downloadable bitstream, of any synchronous logic circuits directly at runtime [6, 1, 2]. Furthermore, by integrating some data only known at runtime into the design process, we can increase the behavioral specification of the application. This optimization then increases the overall performance of the system. The aim of dynamic reconfiguration is to bridge the performance gap between flexible FPGA and full custom ASIC (Application Specific Integrated Circuit). In addition, this technique enables the use of intrinsic bit level parallelism without the overhead of a communication network, as with most general purpose parallel machines. Ganglion, for example, a programmable hardware connectionist classifier realized by C. Cox and W. Blanz [6] that opened the way to real utilization of the concept of reconfiguration, achieved much of the performance and density benefits from full-custom circuits.

2.2 FPGA

A FPGA is a grid of small memory 16x1 bits interconnected by a network across the whole grid, as shown in Figure 2.

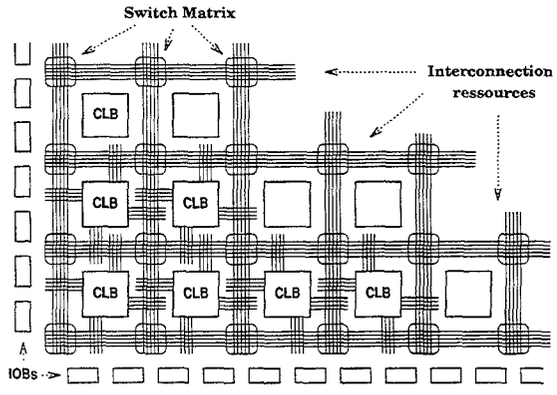

Figure 2: A PAM implementation: the Field Programmable Gate Array

These small memories are lookup tables that can implement any logic functions of four variables. In the FPGA that we used, the lookup tables are grouped in pairs with two additional 1 bit registers and form a CLB (Configurable Logic Block), the basic element of the Xilinx FPGA. The network is fully configurable, each interconnection being made with pass transistors controlled by the state of a one bit static memory. The bitmap (bitstream file in FPGA terminology) formed by all these one bit memories plus the CLBs look-up table is downloaded into the FPGA before any execution. Changing the design of an FPGA is thus done by downloading a new bitstream file that configures the interconnections between binary functions (CLB look-up table). The reader may refer to [4] for a complete description of FPGA.

The main advantages of Programmable Hardware for an AI machine are as follows:

1. Fine grained parallelism respects the time scales between levels in the system.
2. Reconfiguration adds flexibility and suppresses potential overheads induced by the use of parallelism.
3. Reconfiguration of programmable hardware is generally simpler than to reprogramme processors network.

2.3 Measures of computing power

A common unit for computing power is needed to compare the computation realized with and without programmable hardware. For that purpose we use the Bop and Bops previously defined by Vuillemin in [20].

One Bop circuit is any gate with at most three inputs and one bit of internal state, such as a full adder. From this definition we can define a Bops as one binary operation per second. A Bops is delivered by any Bop circuit operating at 1 Hz. Obviously GBops, namely 10^9 Bops, is a more useful measure.

Let us now define in Bop the complexity of some arithmetic and logic operations :

+ One $n + n \mapsto n + 1$ bit addition each nanosecond is worth n GBops. Subtraction, integer comparison and logical operations are bit-wise equivalent to addition.

× One $n \times m \mapsto n+m$ bits multiplication each nanosecond is worth nm GBops. Division, integer shifts and transitive bit permutations are bit-wise equivalent to multiplication; consequently, so is a $n \mapsto m$ Look-Up Table, LUT, or Random Acess Memory, RAM, access.

For example, a 100 MHz, 32 bit microprocessor that can execute all arithmetic operations in one cycle can deliver a virtual computing power of $10^8 \times 32 \times 32 \approx$ 100 GBops. But if the real data are coded in 4 bits the computing power delivered drops to 1.6 GBops.

In this section, we have presented the programmable Hardware to bridge the gap between flexibility and computing power. This is a nice way to get several configurations of an architecture. Now, let us take advantage of this architecture variability in a multi-level architecture.

3 A three level architecture

We present in this section an architecture to efficiently manage the information streams and the different tasks at several levels of abstraction, thanks to programmable hardware. This architecture is applied to genomic systems.

3.1 Principles of a multi-level architecture

In a multi-level architecture, a level :

- should be **stable, autonomous** and **regular**.
- has its own language.
- has specific complexity and timing constraints.
- has limited computing resources.

The architecture we propose is composed of the Cognitive, Software, and Hardware Levels. They are roughly corresponding to the "Performance", "Temporary storage", and "Primitive actions" levels in the architecture of the mind [16, page 81]. They will be detailed in the following. Each task is associated with a level in the three-level architecture, according to its computing time, and communicates with the other levels through information streams. When tasks no longer respect the constraints imposed by a level, they are **delegated** and decomposed to a lower level until they reach a point where they are compatible with time and complexity orders of magnitude.

A level and an information stream are compatible when the returned information volume is included in the level working values, and respects the time and complexity constraints of the level. If the information volume is compatible with the constraints of the level, then the results are transmitted to the cognitive level, and are analyzed by the user. If the results are not relevant, and in order to provide more useful information to the user, a new design is generated for the programmable hardware. This design is built with the main core of the "faulty" design with the user query modification. As soon as the bitstream is generated the process of acquisition is started again, but now the new information extracted concerns the method behavior.

Switching from a task acting on user data to a meta task acting on the behavior of the previous task is done at the cost of a reconfiguration. We should like to point out that the programmable hardware can perform efficiently these two tasks.

A previously computed design is loaded onto the board to produce new results compatible with the software level. Since library design loading only requires one second, and since pattern detectors are characterized by their entry number and the associated threshold, it is very rapidly decided whether a library design is relevant or not.

3.2 Illustration in genetic domain

Let us now illustrate on a pattern matching system the principles of production of an architecture. The global schema of the biological application is represented in Fig. 3

A set of examples and counter examples, (ie biological sequences chosen by the biologist), is learned by the LEGAL system [15, 14], as shown in Figure 3. For each input set, the learning algorithm produces a set of regularities that characterizes it and its associated concept. A regularity is composed of several patterns associated with a decision function, which is a non-linear function formed with a Majority And (MajAnd). This function returns ONE if the number of its inputs at ONE is over a given threshold. The user chooses some regularities to build queries, that are computed on the DECPeRLe-1 Board. Let us now have a look at the levels of the architecture.

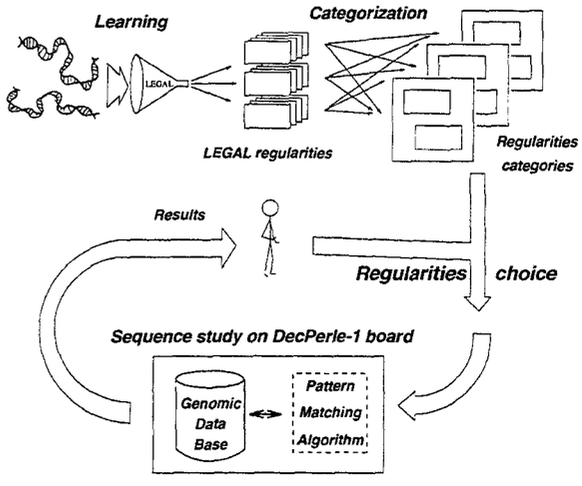

Figure 3: Overview of the genome scanning system: Machine learning methods are classifiers. Their results are regularities categories expressed by sets of Majority ANDs.

3.3 Cognitive Level

This level controls the cognitive tasks of the system. It manages the control of learning, by control of all sets of examples, counter examples, and rules. It also manages the control of decision thresholds used by the learning process. The acquisition of knowledge level is carried out according to a "propose and revise" process, where it is possible to dynamically modify a rule by modifying the decision threshold associated with it, thus limiting the number of sequences corresponding to the rules. Following this, it is also possible to modify the validation rate of the rules. At this point, the operations are restricted to a propositional calculation extension, so that statements can be evaluated in threshold conjunctive normal form. The objective is to obtain a volume of results compatible with cognitive activity and to engage a deliberation process with the user.

3.4 Software Level

This level manages the production of learning rules, which enable the demand for information to be limited with regard to the system controller. It also produces the decision thresholds used at the hardware level, as well as request wiring. During the application, the user chooses,

at cognitive level, queries from a set of candidate queries, furnished by the learning system. All the requests should be formulated in a standard way, namely conjunctive normal form, so that choosing a previously computed design is quite easy. The queries are applied on patterns. The patterns are formed by a sub-alphabet vector, (ie an alphabet reduced to authorized letters at the position in the pattern) [12, 13]. For example [A,G]..[A,T]G.[A,C][C,T,G] is a pattern (. 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. Thanks to a smart implementation, the complexity of a patterns of length 8 equal 8 Bop. The different kinds of queries are :

- Motif detection : The query may be the detection of a motif. For example, the previously introduced pattern will be expressed in conjunctive normal form as follows : $(x_0 = A \vee x_0 = G) \wedge (x_1 \vee \neg x_1) \wedge (x_2 \vee \neg x_2) \wedge (x_3 = A \vee x_3 = T) \wedge (x_4 = G) \wedge (x_5 \vee \neg x_5) \wedge (x_6 = A \vee x_6 = C) \wedge (x_7 = C \vee x_7 = T \vee x_7 = G)$
- Looking for biological properties in patterns by using a rule, which is a set of patterns associated with a threshold.
- Detecting functional properties as primate splice junction sites by using a set of rules and a threshold [15]

3.5 Hardware Level

3.5.1 DECPeRLe-1

A DEC workstation running at 40 MHz with a Reconfigurable board constitutes the experimental platform. The board contains 23 Xilinx 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 (1 Bop), can be evaluated, at each cycle, in the array. Thus at 40 MHz the virtual computing power of the board is $\approx$ 290 GBops. The maximum bandwidth between this array and the memory is 320 MBytes/s with an access time from SRAM to FPGAs of 50 ns. The hardware is connected to a DecStation 5000/240 by a bus TurboChannel at 100 MBytes/s peak and around 20 MBytes/s sustained with 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. 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 board [19].

3.5.2 Patterns and Majority And

A Majority And (MajAnd) is a parallel counter with a threshold output. A (n,m) Parallel Counter (PCs) is a n-inputs circuit that produces a $m = \lceil \log_2(n) \rceil$ bit binary count of the number of its inputs that are ONEs. A (n,m) PC would have an n Bop complexity if we used the Dadda decomposition [7]. The threshold function that transforms a (n,m) PC in MajAnd with n inputs is implemented with a $m \mapsto 1$ look-up table. The main advantage of this implementation is the possibility to change the threshold by writing straight into the bitstream configuration of the FPGAs, and avoid a complete and slow redesign of the whole FPGA. The degree of complexity of a MajAnd $(n) = n + \lceil \log_2(n) \rceil$ Bop.

3.5.3 Runtime dynamic reprogrammation of rules and Majority threshold

Two kinds of reprogrammation are done here, the use of which depends on the degree of modification requiered by the user control. If the biologist seems to be unsatisfied by his previous results and needs to try totally new rules (the size and the number of rules may change), then the hardware part needs a full reconfiguration. In this case, the system takes in charge the rules and the Majority And threshold given by the biologist, and generates automatically the design from the Xilinx Netlist File to the bitstream. In a second case, the user may just want to refine his request (rules + threshold), then, for few modifications, he should not have to wait a long time for the hardware level to be reprogrammed. As a matter of fact, few modifications in a request should be solved in few milliseconds. Thus, we directly modify the bitstream as recommended in [11][6]. This kind of reconfiguration increases both the density and the speed of the design. Once the biologist has given his rules and threshold, the system changes the existing bitstream and downloads it in the FPGA. These two kinds of hardware reprogrammation are runnable by naive users since the system takes in charge the entire reprogrammation from a set of user-defined rules.

In this section, we have presented a three-level architecture, that deals with the cognitive, software and hardware tasks. Let us now have a look at the experimentation on the biological application.

4 Real World Experimentation

Let us identify now the information streams and evaluate them in a software and hardware implementation.

4.1 Information flows

The streams in our biological application are :

- Cognitive level/Software level stream : The stream between cognitive level and software level provides a query formulation for the software level. In our application, the user delegates the building of a query to a learning machine. The query is composed of a set of rules and a decision function using these rules. In return, the stream from software level to cognitive level presents results to the user, in a lapse of time compatible with his mental process.
- Cognitive level/Hardware level stream : This stream is an information stream composed on the one hand of cognitive and software levels, and on the other of software and hardware levels. It is used for rule selection by the user at the Cognitive level. These rules are then managed by the software level, to be executed or to be translated into a sequence of instructions or a bitstream for the hardware level.
- Hardware/Software stream : The software to hardware stream is the characterization of information intended for the hardware task level : data from query wiring and decision threshold fixing. Inversely, the hardware to software stream returns results from the decision mechanism, enabling the learning system to continue with its task.

4.2 Formal evaluation using BOPS

4.2.1 Software implementation

At the "Software level", pattern detection is performed as follows :

```
p = 0;
for (; p < nb_of_patterns; p++) {
  match = patterns[i] ^ sequence_window;
  match = match | ((match >> 1) & mask1);
  match = match | ((match >> 2) & mask2);
  MajAnd += match == mask3; }
```

A connection 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, pattern detection would take 12 cycles. The MajAnd calculation does not add any cycles since it is integrated into pattern 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 patterns of length up to 8 is:

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

Where $\tau_{\mu P}$ is the cycle time of the microprocessor used, (we assume a microprocessor with a CPI equal to one).

The evaluation in Bop of this code gives us :

- add, compare, or, and, xor = w Bop for w bits by word processor.
- load from a p entry table = $\log_2(p) \times w$ Bop.
- shift i position = $i \times w$.
- Total : 12 instructions whose 2 shift and 1 load
  $9\,w + (w + 2w) + \log_2(p)\,w = (12 + \log_2(p)) \times w$ Bop.

For instance with the DECstation 5000/240 with a R3000 microprocessor: $\tau_{\mu P} = 25\,10^{-9}$s, $w = 32$, and 32 patterns we can compute: $\mathcal{N} \approx 10^5$nucleotides per second The computing power used for this task is :

$$\frac{1}{12 \times \tau_{\mu P}} \times 32 \times (12 + \log_2(2^5)) \approx 1.8\,\text{GBops}$$

4.2.2 Hardware implementation

Pattern detectors and MajAnds 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 programmable 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 programmable 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 requires two scanning of the database. In our case requests issued by the LEGAL learning program always have less than 800 patterns and 50

MajAnds that is the implementation limit into the DECPeRLe-1 board. The calculation time in this case is : $T_{FPGA} = \mathcal{N} \times \tau_{FPGA}$.

The computing power required in Bops is given by : $5 \times p \times \mathcal{N} \times \tau_{FPGA}$

5 Experimental timing

5.1 Cognitive Level/Software Level stream timing

This stream consists in compiling the software unit, which takes less than 1 second, and in sending the results to the cognitive level. The step is almost immediate, but still depends on the method used to present results.

5.2 Cognitive Level/Hardware Level stream timing

Once the user has queried the system, the queries must be wired onto the hardware unit. This stream is totally dependent on wiring and routing tool performance. The system can easily wire 60 regularities per FPGA XC3090 with a 8 letter width pattern size. This step requires around 100 seconds per FPGA for wiring, placing and downloading the queries onto the hardware unit :

- Set of rules translation : 1 second
- Routage (custom design only) : 10 seconds
- Bitstream modification & writing of the LUT : 1 second
- Loading : 10^{-1} seconds

This genetic sequences study (Fig. 4) compares execution on a Von Neumann 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 in the DECPeRLe-1 board.

As a result, execution on the Von Neumann 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 difference in execution time is clearly shown in our model by the levels. The first 100 seconds correspond to a treatment at the Software Level, after which the order of magnitude is no longer respected and the Hardware Level assisted by the programmable hardware takes over. The speed at this level is several orders of magnitude faster, with the result that execution speed is greatly increased. Systems which have both software and hardware thus commence execution at the software level, and then progress to the hardware level. The human cognitive mechanisms are respected, because the response time is less than 100 seconds in the general case. A purely software execution takes more than 10000 seconds, which does not allow user cognitive activity. In addition to this, execution time is number pattern dependent, which is not the case when using our system (under 800 patterns and 50 MajAnd).

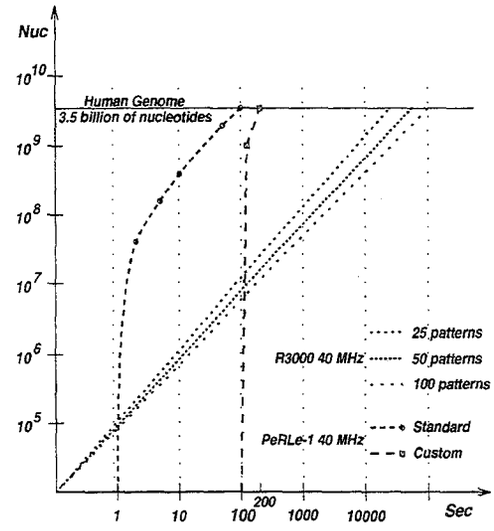

Figure 4: Hardware and software genetic sequence study comparison : The "standard" curve shows the results for a library-design, and the "custom" is obtained with a computed design.

6 Conclusion

Experiments using programmable hardware have demonstrated their ability to respect user interactive time scales when dealing with a data query situation such as knowledge discovery in genetic sequence databases. Programmable hardware has also shown itself to be flexible, since it can be applied to the control of a task during execution by the addition of a piece of hardware. This control possibility increases the overall cognitive capacity of the system [10], and in our view is the key to the stability and autonomy of the different levels in the architecture. To become intelligent systems, the machine should be adapted to the problem. This adaptation may be data-dependent. In this case,

programmable hardware proposes interesting solutions. But in a global frame, the reconfigurable architectures are an intermediate step for achieving the best architecture.

The next step for reconfiguration is for learning algorithms to be directly implemented into the hardware, using dynamically reconfigurable FPGAs. This will open the way for improved skill acquisition, and hence significantly contribute to the progress of Intelligent Machines.

References

[1] P. M. Athanas and H. F. Silverman. Processor reconfiguration through instruction-set metamorphosis. *Computer*, 26(3):11–18, March 1993.

[2] P. Bertin, D. Roncin, and J. Vuillemin. Programmable active memories: a performance assessment. In G. Borriello and C. Ebeling, editors, *Research on Integrated Systems: Proceedings of the 1993 Symposium*, pages 88–102, 1993.

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

[4] Stephen D. Brown, Robert J. Francis, and Jonathan Rose Zvonko G Vranesic. *Field-Programmable Gate Arrays*. Kluwer Academic Publishers, 1992.

[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 Journal of Solid-State Circuits*, 27(3):288–299, March 1992.

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

[8] M. Gokhale, W. Holmes, A. Kopser, 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.

[9] J. P. Gray and T. A. Kean. Configurable hardware: A new paradigm for computation. In *Decennial CalTech Conference on VLSI*, pages 277–293, Pasadena, CA, March 1989.

[10] Barbara Hayes-Roth. Intelligent control. *Artificial Intelligence*, 59(1-2):213–220, February 1993.

[11] E. Lemoine and D. Merceron. Run time reconfiguration of FPGA for scanning genomic databases. In D. A. Buell and K. L. Pocek, editors, *Proceedings of IEEE Workshop on FPGAs for Custom Computing Machines*, pages 90–98, Napa, CA, April 1995.

[12] Eric Lemoine. Reconfigurable hardware for molecular biology computing systems. In *Procc. 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 *Procc. 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 *Procc. of the First Int. Conf. on Intelligent Systems for Molecular Biology*, pages 292–300. AAAI, 1993.

[16] Allen Newell. *Unified Theories of Cognition.* Harvard University Press, 1990.

[17] H. Ripoche, E. Mephu Nguifo, and J. Sallantin. Indexing protein sequences with minos. In *Proc of the Genome Informatic Workshop of Japan*, pages 49–58, 1994.

[18] M. Schaffner. Processing by data and program blocks. *IEEE Transactions on Computers*, 27(11):1015–1028, November 1978.

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

[20] Jean E. Vuillemin. On computing power. *Lecture Notes on Computer Sciences*, 782:69–86, December 1993.