An Efficient Handwritten Digit Recognition Method on a Flexible Parallel Architecture

Aymeric Poulain Maubant, Yvon Autret (2), Guy Léonhard,
Gérald Ouvradou, André Thépaut

Arte Laboratory
Télécom Bretagne
BP832, 29285 Brest CEDEX, France

(2) Université de Bretagne Occidentale, Faculté des Sciences
6, av. Le Gorgeu
29287 Brest CEDEX, France
E-mail: Aymeric.PoulainMaubant@enst-bretagne.fr

Abstract

This paper presents neural and hybrid (symbolic and subsymbolic) applications downloaded on the distributed computer architecture ArMenX. This machine is articulated around a ring of FPGAs acting as routing resources as well as fine grain computing resources and thus giving great flexibility. More coarse grain computing resources -Transputer and DSP- tightly coupled via FPGAs give a large application spectrum to the machine, making it possible to implement heterogeneous algorithms efficiently involving both low level (computing intensive) and high level (control intensive) tasks. We first introduce the ArMenX project and the main architecture features. Then, after giving details on the computing of propagation and back-propagation of the multi-layer perceptron on ArMenX, we will focus on a handwritten digit (issued from a zip code data base) recognition application. An original and efficient method, involving three neural networks, is developed. The first two neural networks deal with the 'reading process', and the last neural network, which learned to write, helps to make decisions on the first two networks outputs, when they are not confident. Before concluding, the paper presents the work of integration of ArMenX into a high level programming environment, designed to make it easier to take advantage of the architecture flexibility.

1: Introduction

The ArMenX project started in 1992. On account of the works on Artificial Neural Networks (ANNs) jointly developed at the *Université de Bretagne Occidentale*, we decided to focus our efforts onto this applicative area. The ArMenX project is involved in two research areas. The first is hardware and software engineering with the aim of a full operating machine supporting a neural development environment. The second deals with research in the area of automatic problem solving involving co-operation between symbolic and connectionist paradigms [1]. Our goal is to exhibit the potentiality of ArMenX in implementing this kind of heterogeneous algorithms (hybrid systems).

The next section depicts the main features of ArMenX. Then subsequent sections deal with the implementation of a multi-layer perceptron (MLP) solving the handwritten digit recognition problem. We first show how this is done for a single neural network, and next for a hybrid system involving three different cognitive approaches. Last section presents the neural development environment.

2: The ArMenX Architecture

Our major aim in designing ArMenX was to propose a wide application spectrum and a modular parallel machine. Thus, the architecture is built around a high bandwidth communication ring implemented by

1086-1947/96 $5.00 © 1996 IEEE
Proceedings of MicroNeuro '96

FPGAs. Two kinds of processor modules are tightly coupled to each FPGA : a general purpose processor (GPP) module and a digital signal processor module (DSP) for vector computations. Finally, a low bandwidth communicating network interconnects the GPP modules to support asynchronous point to point communications, whereas global synchronous communication is supported by the FPGA ring. We now describe the machine in full details.

The basic structure of ArMenX

ArMenX is a purely distributed architecture, involving no centralised subsystem such as a global bus, a shared memory or a central clock generator. The architecture is organised in three processing layers involved in a replicated scheme of several nodes (there is no theoretical limits to the extension of one instancied machine). Each node, called *ArMenX node* in the paper, is made up of the following layers (see figure 1) :

- the GPP layer including asynchronous communication channels,
- the DSP layer,
- the FPGA layer lying in a central position interconnecting the previous two layers and supporting two high bandwidth communication channels.

To design our prototype that we call ArMenX-10, we instanciated the architecture with Transputer T805 as the GPP, Xilinx X4010 as the FPGA and Motorola DSP56002 as the DSP.

Although T805 was not an up-to-date processor, we chose it for two main reasons. It offers the best compacity with regards to its functionalities, which is not far from negligible for a parallel architecture, and provides a lot of software tools, including distributed operating systems. More generally, the overall current machine does not use the last generation of each processor. It is used for implementing numerous applications (multilayer perceptrons, hopfield network [7], hybrid systems, [1]) while some studies are carried for designing a new generation on the same concept (with for instance the TMS320C80 DSP from Texas instruments, and double-port video rams). The reader might be interested by [5] and [8] which present similar architectures.

The DSP layer is intended to achieve vector computations. The choice of the DSP56002, which operates only 24 bit fixed arithmetic, was led by economic considerations according to neurocomputation requirements. Before choosing the processor, we made some studies [3] which demonstrated that floating point

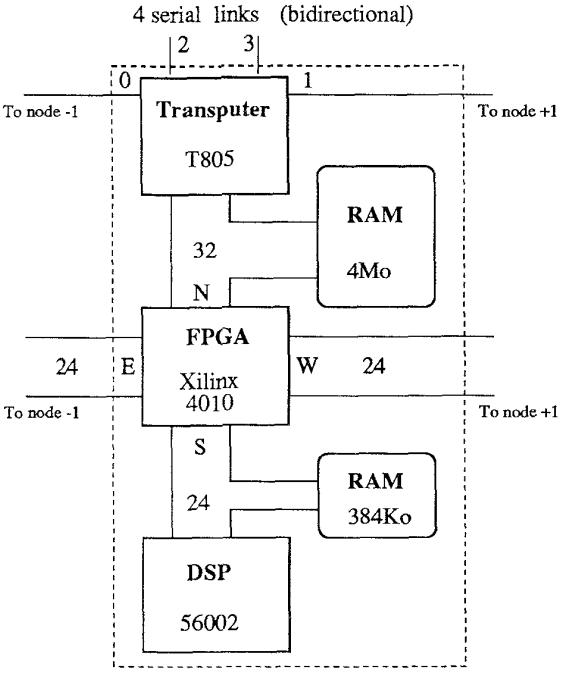

Figure 1. The ArMenX-10 node

arithmetic was not essential in running ANN algorithms like the back propagation. The 24 bit fixed arithmetic of the DSP56002 is convenient for ANN computations as for a wide area of signal processing and provides significant cost-saving with regards to floating point DSPs. The DSP56002 achieves a sustained rate of the "multiply-accumulate" statement of 50 nanoseconds. This feature is of especial interest for processing ANNs because this statement is the kernel of neurocomputations [4].

The FPGA layer lies between these two previous modules. The X4010 is directly connected to the communication bus of the two processors. This disposition allows very strong interaction between the three modules. Depending on the hardware loaded into the X4010, the memory of the processor modules may, for instance, be shared among them; it can also be filled-up or flushed from the two high bandwidth I/O channels without any action from the processors. These I/O channels are also 24 bit data sized like the DSP data path because they are intended to carry data flow processed by the DSP layer.

Some control signals are exchanged between the three layers to monitor their interactions. The three active components being programmable, a download path is implemented from the T805 because it can itself be booted from one of its links. Thus the T805 has to be connected to a device providing the code (e.g. a host computer or another Transputer). The X4010 has several dedicated control pins which are directly mapped in the addressing space of T805 for this purpose. In this way, the T805 can reset, programme or check the X4010. Once the latter is programmed with a design able to access the DSP bus, the T805 downloads the DSP code into the DSP RAM via the X4010. Control signals, involving event inputs, are also managed by the X4010 in order to synchronise module activity. The high bandwidth channels are intended to be monitored by a full handshake protocol in order to avoid the need for a global clock.

3: Mapping a multi-layer perceptron onto ArMenX

The application which is described below is our target application. Thanks to this application, we intend to explore the potentialities of the machine and to extract a know-how about its programming, in order to define convenient software tools. This application is about the hand-written digit recognition problem. We have chosen this framework because it is one of the most popular and well known problems among the neuro-computing community ; the neural network downloaded on the machine is the most basic ANN model, a one hidden layer perceptron.

In our application, the bitmap of a handwritten digit is pre-processed in order to obtain an image of 16 by 16 pixels coding a 16 level grey scale (for more details refer to [2] and [9]). This image is the input of the MLP. The size of the MLP layers are respectively, 256 units for the input layer, 60 for the hidden layer and 10, one for each digit class, for the output layer. Each layer is fully connected to the next one. For the purpose of this article, we will use a 5-nodes ArMenX machine. The general case, along with various other methods, is discussed in [10].

Propagation Let $\mathbf{W_1}$ and $\mathbf{W_2}$ be the synaptic matrices between the input layer and the hidden layer, and between the hidden layer and the output layer, respectively. We note $\mathbf{I_0}$ the input vector of the network, $\mathbf{I_1}$ the input vector of the hidden layer, $\mathbf{O_1}$ its output vector, and $\mathbf{I_2}$ and $\mathbf{O_2}$ the same vectors for the ouput layer. $\sigma()$ is a sigmoid function. We have :

$$\underbrace{\begin{pmatrix} \omega^1_{1,1} & \cdots & \omega^1_{12,1} & \cdots & \omega^1_{49,1} & \cdots & \omega^1_{60,1} \\ \vdots & \ddots & \vdots & & \vdots & \ddots & \vdots \\ \omega^1_{1,256} & \cdots & \omega^1_{12,256} & \cdots & \omega^1_{49,256} & \cdots & \omega^1_{60,256} \\ \theta^1_{1,1} & \cdots & \theta^1_{12,1} & \cdots & \theta^1_{49,1} & \cdots & \theta^1_{60,1} \end{pmatrix}}_{\mathbf{W_1}}$$

$$\mathbf{W_2} = \begin{pmatrix} \omega^2_{1,1} & \cdots & \omega^2_{2,1} & \cdots & \omega^2_{9,1} & \cdots & \omega^2_{10,1} \\ \vdots & \ddots & \vdots & & \vdots & \ddots & \vdots \\ \omega^2_{1,40} & \cdots & \omega^2_{2,40} & \cdots & \omega^2_{9,40} & \cdots & \omega^2_{10,40} \\ \theta^2_{1,1} & \cdots & \theta^2_{2,1} & \cdots & \theta^2_{9,1} & \cdots & \theta^2_{10,1} \end{pmatrix}$$

$$\mathbf{I_1} = (\mathbf{I_0}, 1) \cdot \mathbf{W_1} \textit{ and } \mathbf{O_1} = \sigma(\mathbf{I_1})$$

$$\mathbf{I_2} = (\mathbf{O_1}, 1) \cdot \mathbf{W_2} \textit{ and } \mathbf{O_2} = \sigma(\mathbf{I_2})$$

$(\mathbf{I_0}, 1)$ is a 257 elements vector, the last value being 1, for the multiplication with the thresholds $\theta^1_{1..40,1}$. $\mathbf{W_1}$ and $\mathbf{W_2}$ are distributed on ArMenX : each matrix is vertically cut in 5 equal parts and the weights are loaded in the respective DSP ram. For instance, the first node deals with columns 1 to 12 of $\mathbf{W_1}$ and columns 1 and 2 of $\mathbf{W_2}$ (and so on, in a regular manner).

With this scheme, the propagation algorithm on ArMenX is as follows (in this algorithm, "layer" designates the MLP layers) :

Step 1 *Layer 1* For $\mathbf{I_1} = (\mathbf{I_0}, 1) \cdot \mathbf{W_1}$, each DSP has 12x257 *multiply and accumulate* operations to do. Thus, each node has a partial part of $\mathbf{I_1}$.
Step 2 *Layer 1* Each DSP calculates the sigmoid (implemented as a LUT, for instance). Thus, each node has a partial part of $\mathbf{O_1}$.
Step 3 Distribute the partial parts of $\mathbf{O_1}$ in a *direct memory access* session, via the FPGA ring. This gives a complete $\mathbf{O_1}$ vector to each node.
Step 4 *Layer 2* For $\mathbf{I_2} = (\mathbf{O_1}, 1) \cdot \mathbf{W_2}$, each DSP has 2x41 *multiply and accumulate* operations to do. Thus, each node has a partial part of $\mathbf{I_2}$.
Step 5 *Layer 2* As in step 2, each DSP calculates the sigmoid. Thus, each node has a partial part of $\mathbf{O_2}$.

We underline the distribution phases. They are needed to reconstitute complete vectors (like $\mathbf{O_1}$) on each node, when partial computations (source of parallelism) is impossible. The scheme depicted in this paper is the simplest one : only few data (less than 400 bytes) circulate on the FPGA ring. Details of the overall process can be found in [12].

Backpropagation and weights updating Backpropagation is the process of updating the synaptic weights with respect to the error (difference between the target vector and $\mathbf{O_2}$: $\mathbf{E}$) calculated at the output layer. The target vector (a 10 elements vector with nine 0 and one 1) was distributed on the nodes along with $\mathbf{I_0}$. We have the following equations :

$$\delta_2 = \mathbf{E} \otimes \sigma'(\mathbf{I_2})$$

$$\delta_1 = (\mathbf{W_2^-} \times \delta_2) \otimes \sigma'(\mathbf{I_1})$$

$\mathbf{W_2^-}$ being $\mathbf{W_2}$ without the last line of thresholds, and $\otimes$ the Hadamar product. With this scheme, the backpropagation algorithm on ArMenX is as follows :

> **Step 1** *Layer 2* Each DSP computes 2 elements of $\mathbf{E}$.
> **Step 2** *Layer 2* Each DSP computes 2 elements of δ_2. As the complete vector is needed on every node, it is redistributed in a single *dma* session.
> **Step 3** *Layer 1* Each DSP computes 2x10 *mac* to do, and 2 multiplications. This gives partial δ_1 on each node, which are redistributed in a *dma* session.

It is now possible to compute the synaptic weights updating on each node. Partial updating is carried on each node, and the overall updating equations are as follows :

$$\mathbf{W_2}(t+1) = \mathbf{W_2}(t) + \alpha \mathbf{O_1} \cdot \delta_2$$

$$\mathbf{W_1}(t+1) = \mathbf{W_1}(t) + \alpha \mathbf{I_0} \cdot \delta_1$$

Performance of the target application

Thanks to the systolic execution scheme adopted, the performance running the MLP increases almost linearly with N, the number of nodes providing that layer sizes are all greater than N. This is obtained thanks to overlapping the communication activity and processor activity which requires a careful management of the DSP internal memories. In order to improve this point, we look at the possibility of using a dual port memory instead of the conventional memory shared by the DSP and the FPGA.

For the forward phase, the sustained performance exhibited by the machine is about 5 MCPS per node (MCPS : Million Connections processed Per Second). For a training iteration involving backpropagation, the performance is about 1.5 MCUPS per node (MCUPS : Million Connections Updated Per Second). We are confident to obtain these results up to a 16 node machine size. In the context of our target application this leads the MLP to answer a digit identification in a time less than 150 microseconds, that is 400,000 digits per minute. For training the MLP with a data base of 10,000 digit instances, for example, less than 5 seconds are needed for a sweep of the base.

The price of one ArMenX node is about 2000 US dollars.

4: Pixmaps, contours and writing learning for digit recognition

Although we supposed during the previous section a pixmap representation of the digits, other digit representations have been used on ArMenX. The first one is a contour representation, and the collaboration between both pixmaps and contours approaches is discussed in [9]. The second one is a more symbolic representation, close to a formal description on how to write a digit ("lower the pen, then draw different types of curves and lines, and raise the pen"). This symbolic representation is discussed in [10].

In this section, we discuss a hybrid system (using symbolic and subsymbolic materials) which we call a 'hybrio'. We want here to stress that a particular relationship must be established between the 'neurotician' (the cognitician of hybrid systems) and the hybrid system he develops : an educational process is used to improve the performances of the system, which implements very deep coupling between the symbolic and subsymbolic paradigms, [6].

Each neural network of our *hybrio* is similar to the one discussed in the pervious section, and it is possible to use several ArMenX nodes per neural network. However, for the simplicity of our paper we will now consider that ArMenX has three nodes. The first one is used for what we call the pixmap module, the second one for the contour module and the last one for the writing module.

4.1: Cooperation between pixmaps and contours

For this work, we use two handwritten digit data bases of 10000 examples : the first one is the pixmap representation and the second, issued from the first one, is the contour representation. For each digit, 500 examples are kept for test purposes, and 500 are used for learning. The contour representation is a 64x4 vector. It is obtained after normalization of a long vector of contours, where directions (W,N,E,S) and composed directions (NW, NE...) follow the Freeman encoding.

Recognition rates for the pixmap representation are 94,7%. The confusion matrix for this method is presented in table 1.

	0	1	2	3	4	5	6	7	8	9	Rate %
0	491	4	0	0	1	1	0	0	1	2	98,2
1	0	491	2	0	3	0	0	3	1	0	98,2
2	2	2	475	1	5	0	0	2	9	4	95,0
3	0	6	0	451	3	6	1	2	14	17	90,2
4	0	11	1	0	480	0	4	3	1	0	96,2
5	0	1	3	6	0	473	6	1	6	4	94,6
6	3	0	0	0	2	6	483	1	5	0	96,6
7	0	11	1	2	6	0	0	468	5	7	93,6
8	0	5	0	0	0	0	0	10	480	5	96,0
9	4	10	4	8	0	0	2	14	14	444	88,8

Table 1. Confusion matrix for the pixmap module

Recognition rates for the contour representation are 93,8%. These rates are obtained on a public data base given by the french post office research center. The confusion matrix for this method is presented in table 2.

	0	1	2	3	4	5	6	7	8	9	Rate %
0	493	2	0	1	1	2	0	0	0	1	98,6
1	1	484	3	0	0	1	0	2	5	4	96,8
2	3	10	424	2	7	1	4	12	28	9	84,8
3	1	0	1	472	0	5	0	6	2	13	94,4
4	3	3	5	1	467	1	12	7	1	0	93,4
5	1	0	0	6	0	475	6	4	5	3	95,0
6	1	0	2	1	2	0	490	0	4	0	98,0
7	5	9	1	3	5	1	0	455	13	8	91,0
8	5	10	0	0	0	0	5	0	480	5	96,0
9	14	8	0	0	2	2	0	2	22	450	90,0

Table 2. Confusion matrix for the contour module

It is worth noticing that errors do not systematically occur on the same digits for both approaches. Thus, we can make the two approaches cooperate and give better overall results.

It has been shown in [9] that the best method of cooperation is the simplest one. We just multiply outputs from the two modules, and obtain a recognition rate of 97,36% with a reject rate of 7%. The new confusion matrix is presented in table 3.

Generally, digits which are well recognized (activation of the corresponding output neuron close to 1) by one of the two modules are recognized by the cooperative approach. Improvement comes from digits which are not well recognized by one module (mean activation distributed on some output neurons), or not recognized at all but classified in second position by the modules.

	0	1	2	3	4	5	6	7	8	9	Rate %
0	498	0	0	1	1	0	0	0	0	0	99,6
1	0	495	1	0	1	1	0	1	0	1	99,0
2	0	1	491	0	2	0	1	1	3	1	98,2
3	0	0	1	479	2	5	0	4	1	11	95,8
4	0	4	1	1	485	4	5	4	0	0	97,0
5	0	0	0	1	0	494	4	2	5	3	97,0
6	1	0	1	0	3	0	494	0	1	0	98,8
7	1	9	1	1	3	0	0	478	3	4	95,6
8	0	5	0	0	0	0	0	0	495	0	99,0
9	6	8	0	4	2	0	0	6	6	468	93,6

Table 3. Confusion matrix for the cooperation between pixmaps and contours modules

Multiplication of the outputs allow to give more importance to these types of activation.

It should be possible to improve results for the following classes of digits : '2', '3', '4', '7' and '9'. This will be addressed in the next section.

4.2: Learning to write

Some studies have been carried out for improving the above results. [10] deals with re-entrance of activations between hidden layers of both neural networks. A complementary approach is dealt in this section. The basic idea is that improvements are possible from a simultaneous learning of digit recognition (reading) and digit production (writing). A symbolic representation of the digits has been chosen, and a very small set of rules can help the first two modules to give a decision, with respect to the computation of the third module, dedicated to learning to write.

The symbolic representation of digits is very natural : there are two symbols for "the pen is lowered" and two others for "the pen is raised", thus taking in account digits like 7 where, in french, the pen must be raised during the writing process. The other symbols represent arcs of differents curvatures, small buckles and cross patterns.

250 random digits have been isolated from the pixmap learning base, and presented on screen to the *neurotician*. Using a graphical tablet, we have been able to produce a third digit base, made of symbolic representations. This is really a simple process : just draw above a presented pixmap, and events from the tablet are memorized. A statistical analysis of the events data allows to find 11 classes of curvatures, from sharp bends to quasi straigth lines.

We then downloaded a 256x60x13 MLP in the third node : the first two neurons of the output layer represent the presence or absence of a second 'low down' and

'raise up' respectively, while the next 11 represent the 11 classes of curvatures. Whereas the first two neurons fire in a 0-1 manner, the last 11 fire in a manner proportionnal to the number of elementary arcs we received from the tablet. A learning process has been done to associate the pixmaps to output activation close to a prototypical activation extracted from the statistical analysis. The complete process is described in [10].

Up to now, this third module does not classify anything. A statistical analysis of the third module activations allows to find which output neurons are likely to fire when a digit is presented. For instance, neurons 1 and 2 (lower and raise the pen) do fire almost every time for digit 7, and never for digit 0. It is thus a new way to distinguish these two digits. Numerous relations of this type can be found. When classifications proposed by the first two modules do not seem confident (when for instance the output activation is too low, typically less than .3), the third module is activated and, with respect to a simple rule established on account of the precedent couple relations, it can reject the digit classified first by the reading modules. It is thus possible to accept the digit classified second, which in most cases is the solution (or if the writing module finds this cannot be a good choice either, we accept the digit classified in third...).

As the learning of the writing module took place on 5% of the learning base, it was not that perfect. The strategy of reclassification gave more digits in error than it corrected misclassified digits by the first two modules alone. We ask our system to show which digits were misclassified by the writing modules decisions : among all the newly introduced errors, we observed for instance the '2' with a large buckle on the upper half part. It seems that those type of '2' were not well drawn during the first learning phase of the writing module. We did a second learning process on 150 random digits in the learning base, with 14 '2', some of which with this buckle pattern. By stressing this character when drawing on the digit pixmap during the construction of the new writing base, we act as we explained a child how to write a '2', before he draws lines of '2'. Other misclassified digits by the third module were stressed in the same manner, and this drastically reduced the amount of new introduced errors : in fact, the third module now introduces less new errors than it corrects misclassified digits by the reading modules, improving the performances of the overall system. We are currently working on the last percent of error, finding how to teach the system how to write ambiguous digits, and testing it on never seen digits, drawn on the graphical tablet.

5: A High Level Programming Neural Network Environment

One of the most restrictive factors in the implementation of Artificial Neural Network is that an ANN application or algorithm must be completely recoded each time a new architecture is targeted. One suitable solution is to use a highly standardised environment which gives a generic description of the ANN to allow the targeting on various platforms such as general purpose, parallel or sequential hardware without modifications. The **MimEdens** environment performs this task and offers an easy way to use and to program them. This environment is part of the GALATEA project[11], which was a major European neurocomputing project under ESPRIT II. This section will review the basic issues of GALATEA and the concept of virtual machines. The implementation on the ArMenX machine is then described.

5.1: The MimEdens environment

The GALATEA project which ended in 1994 aims to promote the application of neural networks by European industry and to provide standardised hardware and software system for development and execution of ANN applications. The system will encompass:

- a **General Purpose NeuroComputing hardware (GPNC)**, which is a heterogeneous distributed architecture based on a dynamically reconfigurable interconnection network with a workstation. This architecture allows to buid systems based on the association of neurocomputing operators, digital signal processors, general purpose microprocessors, parallel machines, etc.
- a sophisticated neural programming environment, the **Neural Network Programming System (NNPS)**, allowing the efficient use the GNPC,
- a **"Silicon Compiler"** for rapid and low cost production of ASICs.

The MimEdens environment is a general open programming system. It consists of six components as shown in the figure 2 :

- an object oriented neural network programming language called N
- an algorithm library written in N,
- graphical tools for building and debugging N programs,

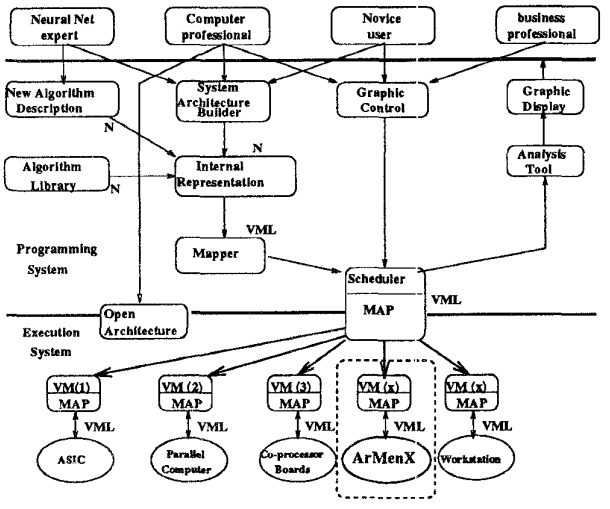

Figure 2. The MimEdens environment

- additional graphical tools for controlling neural network applications,
- an intermediate level language called VML,
- a mapper which will distribute the VML code over the hardware system.

As previously mentioned the hardware system can be constructed by the association of various elements to built a GNPC system. To perform this task, the MimEdens environment is based on the Virtual Machine (VM) concept and Virtual Machine Language (VML). The VML code is automatically generated by the N compiler, although it can be used for direct hand-coding of various applications, not only neural ones.

A Virtual Machine (VM) is a set of hardware and software components. It is basically composed of two modules, a communication and an execution module:

- the communication unit will support the communication with the host running the NNPS and the other VM of the GNPC, or peripherals. It is responsible for the exchange of commands, code and data beetween the host and the target machine(s).
- the execution unit is responsible of the execution of the VML code and primitives.

The VML language is composed of a rich set of matrix and vector arithmetic operations, which consist the core of all ANN computation. It is a structured language close to C but with a restricted structure. Most of these arithmetic primitives are inherently parallel and should be easy to implement on parallel hardware. Their efficient implementation is the key issue for a performance effective design. VML programs consist of one or more rules, which may be invoked by the scheduler (NNPS) or by calls from other rules. Our goal is to implement a VM on ArMenX, to be able to include it on the list of the targeted machine in order to have a neurocomputer with a standard NNPS.

5.2: The ArMenX implementation

As the efficient implementation depends on the good parallelisation of the VML primitives, we first concentrate our efforts on this task. In this approach, the ArMenX machine is only used as a VML coprocessor, which uses Remote Procedure Call (RPC) to execute the VML primitives. The VML code is executed on the host VM and only a few primitives are executed by RPC on ArMenX (see [12]).

In this first approach only a few primitives of the matrix operations are implemented on the machine. The implementation of these VML primitives working on matrix and vector operations is directly targeted on the DSP level. The high speed ring is used to perform the data update on all the nodes of the machine. The Transputer network is only used to control the communication and the machine. The first transputer has a specific job. It must connect the machine to the workstation. The VM is implemented on the workstation and the ArMenX machine is directly hooked on it and called via a specific primitive which are added in the VML code by a code parser. This allows us to compare the performance of the VML primitives between ArMenX and the workstation.

This implemention is currently in progress and an optimisation is studied. The present work shows that the parallel matrix computation can be well achieved on the ArMenX machine because of the DSP level and the high speed ring used to update the data. Other FPGA configurations will be studied to accelerate the parallel computation of the VML primitives. Also a compiler will be written to perform the automatic parallelisation and optimisation of the VML codes on ArMenX, in order to make the best use of all machine features.

6: Conclusion

We worked during three years on handwritten digit recognition. Tuning carefully the neural algorithms and studying real applications needed important computing resources. Thanks to its flexibility and its performances, the ArMenX machine was well adapted to the research of efficient approaches towards an elegant

solution. The numerous simulations carried on the machine justify the proposition of a recognition model based on the cooperation of two specialized modules (dedicated in the reading skill). This model has been successfully applied on a real-world task. According to the french post research center, our results appear to be the state of the art in june 1995 (the algorithms have been tested on their private data bases, with which they can compare the work of different research teams). We are still working on the integration of the third module (dedicated to the writing skill) with the first two, testing this approach on public handwritten digit bases ; we hope it could be tested soon with success on private digit bases from our post office research center.

7: Acknowledgement

We would like to thank B. Angeniol of the Mimetics Co. for his precious help in targeting ArMenX for the MimEdens environment. We also would like to thank M. Gilloux and B. Lemarie of the SRTP Nantes (Post Office Research Center) for providing the public databases of french zip codes. The ArMenX architecture is patented under European Patent 0552074, property of France Telecom.

References

[1] A. Poulain Maubant, G. Ouvradou, A. Thépaut, "Hybrid Systems on a Multi-grain Parallel Architecture", Proc. of PPAI-93, Second International Workshop on Parallel Processing for Artificial Intelligence, Chambery, France, August 29-30, 1993.

[2] Autret, Y., Thépaut, A., Ouvradou, G., Le Drezen, J., and Laisne, J.-D. "Parallel learning on the ArMenX machine by defining sub-networks" In *International Joint Conference on Neural Networks*, volume 1, pages 915–918, Nagoya, Japan, 1993.

[3] Y. Autret, G. Ouvradou, A. Thépaut, "Transputers and Programmable Logic for Neural Networks", in Proc. of Int. Conf. of Transputers'92, Advance Research and Industrial Applications, ed. M. Becker et al. pp. 262-272, Arc et Senans, France, May 1992.

[4] Ramacher, U. "Synapse – a neurocomputer that synthesizes neural algorithms on a parallel systolic engine" *Journal of Parallel and Distributed Computing*, 14:306–318, 1992.

[5] Nelson Morgan et al., "The Ring Array Processor : a multiprocessing peripheral for connectionist applications" in J. Parallel Distrib. Comput, vol. 14, pp.248-259, 1992.

[6] Kerckhoffs, E. J. H. "A view on problem-solving paradigms including neurocomputing" In *Spring School Proceedings on AI and Parallelism*, Delft University of Technology, The Nederlands, 1992.

[7] J.-D. Kant, J. Le Drezen, J. Bigeon "Electromagnetic Field Parallel Computation with a Hopfield Neural Network" in proc. of CEFC'94, sixth IEEE Conference on Electromagnetic Field Computation, Aix-les-Bains, july 1994.

[8] Müller, U. A., Gunzinger, A., and Guggenbühl, W. "Fast neural net simulation with a dsp processor array." IEEE Transactions on Neural Networks, 6(1):203–213. 1995.

[9] Thépaut, A. "Contribution à l'étude des machines hybrides : application à la reconnaissance des chiffres manuscrits" PhD thesis, Université de Montpellier. 1995.

[10] Poulain Maubant, A. "Collaboration entre les Réseaux Connexionnistes et le Traitement Symbolique. Mise en Œuvre sur une Machine Parallèle à Grain Multiple" PhD thesis, Université de Rennes I. 1995.

[11] J. C. Taylor, M. L. Recce, and A. S. Mangat. "Flexible operating environnement for matrix based neurocomputers" In Springer-Verlag, editor, *New Trends in Neural Computation*, International Workshop on Artificial Neural Networks, IWANN 93, pages 382–387, 1993.

[12] G. Léonhard, E. Cousin, J.D. Laisne, J. Le Drezen, G. Ouvradou, A. Poulain Maubant, A. Thépaut. "ArMenX : a flexible platform for signal and image processing" In J. Schewel, editor, *Field Programmable Gate Arrays (FPGAs) for Fast Board Development and Reconfigurable Computing*, volume 2607, P.O. Box 10, Bellingham, WA 98227-0010 USA, 1995. SPIE, Photonics East: Intelligent Systems and Advanced Manufacturing.